UNIVERSITE PIERRE & MARIE CURIE
LA SCIENCE A PARIS

Institut Formation Doctorale

THÈSE DE DOCTORAT DE
L'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique, Télécommunications et Électronique

Présentée par

M. Renaud BARATE

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Apprentissage de fonctions visuelles pour un robot mobile par programmation génétique

Préparée au
Laboratoire Électronique et Informatique (UEI)
de l'École Nationale Supérieure de Techniques Avancées (ENSTA)
32 Boulevard Victor, 75739 Paris Cedex 15

soutenue le 26 Novembre 2008

devant le jury composé de :

M. Jacques BLANC-TALON	Examinateur
M. Pierre COLLET	Rapporteur
M. Jean LOUCHET	Directeur de thèse
M. Antoine MANZANERA	Directeur de thèse
M. Maurice MILGRAM	Examinateur
M. Fawzi NASHASHIBI	Rapporteur

Résumé

En robotique mobile, les techniques d'apprentissage qui utilisent la vision artificielle représentent le plus souvent l'image par un ensemble de descripteurs visuels. Ces descripteurs sont extraits en utilisant une méthode fixée à l'avance ce qui compromet les capacités d'adaptation du système à un environnement visuel changeant. Nous proposons une méthode permettant de décrire et d'apprendre des algorithmes de vision de manière globale, depuis l'image perçue jusqu'à la décision finale.

L'application visée est la fonction d'évitement d'obstacles, indispensable à tout robot mobile. Nous décrivons de manière formelle la structure des algorithmes d'évitement d'obstacles basés sur la vision en utilisant une grammaire. Notre système utilise ensuite cette grammaire et des techniques de programmation génétique pour apprendre automatiquement des contrôleurs adaptés à un contexte visuel donné.

Nous utilisons un environnement de simulation pour tester notre approche et mesurer les performances des algorithmes évolués. Nous proposons plusieurs techniques permettant d'accélérer l'évolution et d'améliorer les performances et les capacités de généralisation des contrôleurs évolués. Nous comparons notamment plusieurs méthodes d'évolution guidée et nous en présentons une nouvelle basée sur l'imitation d'un comportement enregistré. Par la suite nous validons ces méthodes sur un robot réel se déplaçant dans un environnement intérieur. Nous indiquons finalement comment ce système peut être adapté à d'autres applications utilisant la vision et nous proposons des pistes pour l'adaptation d'un comportement en temps réel sur le robot.

Mots-clés : Vision, robotique mobile, programmation génétique, évitement d'obstacles

Abstract

Title : Learning Visual Functions for a Mobile Robot with Genetic Programming

Existing techniques used to learn artificial vision for mobile robots generally represent an image with a set of visual features that are computed with a hard-coded method. This impairs the system's adaptability to a changing visual environment. We propose a method to describe and learn vision algorithms globally, from the perceived image to the final decision.

The target application is the obstacle avoidance function, which is necessary for any mobile robot. We formally describe the structure of vision-based obstacle avoidance algorithms with a grammar. Our system uses this grammar and genetic programming techniques to learn controllers adapted to a given visual context automatically.

We use a simulation environment to test this approach and evaluate the performance of the evolved algorithms. We propose several techniques to speed up the evolution and improve the performance and generalization abilities of evolved controllers. In particular, we compare several methods that can be used to guide the evolution and we introduce a new one based on the imitation of a recorded behavior. Next we validate these methods on a mobile robot moving in an indoor environment. Finally, we indicate how this system can be adapted for other vision-based applications and we give some hints for the online adaptation of the robot's behavior.

Keywords : Vision, mobile robotics, genetic programming, obstacle avoidance

Remerciements

Je tiens avant tout à remercier mon directeur de thèse M. Antoine Manzanera. Ses conseils et son optimisme m'ont été d'une aide précieuse tout au long de cette thèse. Je le remercie pour la liberté qu'il m'a laissée dans mes recherches et pour tout le temps qu'il m'a consacré lors de l'écriture des articles et de ce manuscrit. Merci également à M. Jean Louchet d'avoir co-encadré cette thèse. Ses conseils lors de nos entrevues ont toujours été fructueux.

Je souhaite remercier les membres du jury de m'honorer de leur présence lors de cette soutenance de thèse. Merci à M. Fawzi Nashashibi et à M. Pierre Collet qui ont accepté d'être rapporteurs pour cette thèse. Le temps très limité qui leur était laissé pour relire le manuscrit ne les a pas empêché d'écrire des rapports très détaillés et constructifs, et ce malgré leurs nombreuses obligations. Je les remercie pour leurs remarques et commentaires qui m'ont permis de grandement améliorer ce manuscrit de thèse.

Merci également à M. Maurice Milgram de faire partie du jury malgré ses nombreuses contraintes professionnelles. Je tiens à remercier M. Jacques Blanc-Talon et à travers lui la DGA pour avoir assuré la majeure partie du financement de cette thèse.

Cette thèse n'aurait pas pu se dérouler dans d'aussi bonnes conditions sans l'ensemble du personnel de l'ENSTA et plus particulièrement de l'UEI. Je remercie M. Thierry Bernard sans qui cette thèse n'aurait pas vu le jour ainsi que M. Alain Sibille, directeur de l'UEI, pour sa disponibilité et son engagement m'ayant permis de participer à plusieurs conférences internationales parfois lointaines. Merci à M. David Filliat sans qui les expériences sur le Pioneer 3DX n'auraient pas pu être réalisées. Merci à Mme Jacqueline Darrozès pour son aide avec les innombrables détails administratifs qui font partie du parcours de thèse. Merci également aux ingénieurs, techniciens et enseignants-chercheurs sans qui le laboratoire ne pourrait pas fonctionner.

Je tiens à remercier particulièrement les autres doctorants et stagiaires de l'UEI pour leur soutien quotidien et tous les moments partagés durant ces trois années. Merci à Nicolas, Paul, Taha, Marc, Raffaele, Amine, Sylvain, Marc, Christine, Asad, Adrien, Mathieu, ainsi qu'à ceux que j'ai connus plus brièvement.

Je remercie aussi tous les amis extérieurs à l'ENSTA de m'avoir régulièrement permis de m'ouvrir l'esprit et de réaliser qu'il y a une vie en dehors de la thèse. La liste serait trop longue, et le risque qu'elle soit incomplète trop grand, mais qu'ils soient tous remerciés pour leur bonne humeur et leur soutien. Je les prie de m'excuser pour mes nombreuses absences ces derniers mois.

Je souhaite remercier particulièrement mes parents et ma sœur d'avoir su me donner le goût pour les sciences et de m'avoir toujours soutenu pendant toutes ces années. Les retours dans le pays de Gex et mes montagnes natales sont toujours une grande source de réconfort.

Merci enfin à Estelle d'avoir supporté cette thèse et le temps que j'ai consacré à celle-ci durant ces trois années. C'est sa présence et son soutien indéfectible qui m'ont permis d'avancer au quotidien et de finir cette thèse. Merci infiniment.

Table des matières

Introduction

Parmi les nombreux organes sensoriels développés par la nature, la vision est celui qui apporte l'information la plus riche et la plus complète sur le milieu environnant. La vue permet en effet de percevoir simultanément l'environnement proche et lointain avec une quantité de détails inaccessible à tout autre sens. Les animaux évolués peuvent grâce à la vue s'orienter et se déplacer, repérer et chasser une proie ou au contraire fuir un prédateur. Son utilisation représente un tel avantage que l'évolution de la vision fait partie des explications plausibles à l'explosion cambrienne. Cette période qui s'étend d'environ 540 à 520 millions d'années avant notre ère a vu l'extinction soudaine d'un grand nombre d'espèces, l'apparition tout aussi rapide de traits anatomiques totalement nouveaux et la formation de la plupart des grands groupes d'animaux actuels. L'évolution de la vision sur quelques millions d'années pourrait avoir déclenché une course au développement entre proies et prédateurs, ce qui expliquerait ces changements soudains dans la morphologie des animaux.

La vision s'avère de plus remarquablement adaptative. Si la recherche de proie ou la fuite de prédateur ne représente généralement plus le quotidien de l'homme aujourd'hui, la vision est toujours aussi indispensable pour l'orientation et la navigation, pour la reconnaissance d'objets et de personnes ainsi que pour la communication gestuelle et écrite. Paradoxalement, son utilisation est tellement naturelle et simple pour l'homme qu'il est difficile d'appréhender la complexité des traitements réellement effectués pour acquérir et interpréter cette information. Pourtant la perception visuelle est le résultat d'une multitude d'opérations réparties sur plusieurs chaînes de traitement parallèles qui occupent une grande partie du cerveau humain [Ungerleider 94].

De ce point de vue, il est intéressant de constater que les chercheurs en robotique ont rapidement essayé de faire bénéficier les robots de capacités de vision en supposant qu'un robot doté de vision serait plus à même d'effectuer des tâches essentielles telles que la navigation dans un environnement inconnu par exemple. Il s'est rapidement avéré que le problème était beaucoup plus complexe qu'il n'y paraissait et paradoxalement, les recherches actuelles en vision pour la robotique sont souvent moins ambitieuses que celles qui ont été menées dans les années 80.

Cela ne signifie pas qu'aucun progrès n'a été réalisé dans le domaine. Des tâches comme le positionnement d'un bras robot par rapport à des repères visuels ou la détection d'anomalies sur une pièce peuvent maintenant être effectuées de manière fiable. Cependant il s'agit plutôt dans ce cas de problèmes liés à l'utilisation de la vision dans l'industrie, pour lesquels l'environnement et les conditions d'acquisition peuvent être parfaitement maîtrisés. Des systèmes ont également été développés pour des robots mobiles autonomes, permettant par exemple de construire une carte métrique de l'environnement ou de suivre une route et détecter les véhicules en utilisant uniquement la vision. Néanmoins ces systèmes restent très sensibles à toute

variation de l'information visuelle, qu'il s'agisse d'un changement de luminosité, de texture ou de point de vue. En termes de capacité d'apprentissage ou d'adaptation à un environnement changeant, la vision humaine reste encore très largement inégalée.

Une des difficultés majeures qui se pose aux systèmes artificiels pour l'apprentissage de la vision est que la quantité de données délivrée par les capteurs d'images est très importante et il est difficile d'extraire l'information pertinente de ce flot de données. Les techniques d'apprentissage automatique existantes nécessitent généralement de disposer d'une information réduite et significative en entrée. De ce fait, les systèmes effectuant de l'apprentissage sur des données visuelles vont le plus souvent commencer par extraire des caractéristiques ou descripteurs représentant l'information visuelle de manière condensée. L'apprentissage en lui-même est ensuite réalisé à partir de ces descripteurs. Ce parti pris, très compréhensible d'un point de vue computationnel, restreint toutefois grandement les possibilités d'utilisation de la vision et explique en grande partie le manque d'adaptativité de la plupart des systèmes artificiels. En effet, les mécanismes d'attention visuelle mis en œuvre dans le cerveau humain permettent d'adapter les traitements visuels de bas niveau en fonction du contexte et de la tâche en cours [Desimone 95]. Cette influence réciproque entre traitements *bottom-up* et sélection *top-down* semble être centrale pour l'adaptativité de la vision humaine. Une des hypothèses principales de cette thèse est ainsi que l'apprentissage du processus de vision dans sa globalité est bien plus à même de résoudre des tâches complexes et de s'adapter à des conditions changeantes qu'un apprentissage utilisant uniquement des descripteurs visuels extraits de l'image par une fonction fixée à l'avance. Afin de démontrer la validité de cette hypothèse, cette thèse s'articulera selon le plan suivant.

Le chapitre 1 débutera par une présentation du contexte et des motivations de cette recherche et justifiera le choix de restreindre cette étude à la vision monoculaire. Nous présenterons ensuite différentes approches qui ont été proposées pour aborder les problématiques d'apprentissage et d'adaptation de la vision en robotique mobile et nous situerons cette thèse par rapport à ces approches.

Nous présenterons au chapitre 2 la principale application étudiée dans nos travaux, à savoir la fonction d'évitement d'obstacles. Nous décrirons différentes méthodes permettant d'éviter les obstacles en utilisant la vision, dans le but d'extraire une base de primitives visuelles exploitables dans notre système. Nous formaliserons la représentation de ces primitives et leur structuration en algorithmes en utilisant une grammaire. Nous présenterons également le processus de génération d'algorithmes à partir de cette grammaire, issu de travaux en programmation génétique.

Le chapitre 3 décrira les méthodes d'évolution artificielle employées pour sélectionner et améliorer ces algorithmes. Nous présenterons notamment les environnements de simulation et le milieu réel dans lesquels les expériences ont été réalisées et nous justifierons les fonctions choisies pour évaluer la performance des différents algorithmes. Nous décrirons également plusieurs techniques liées à l'évolution artificielle que nous avons mises en œuvre dans notre système pour améliorer les performances des algorithmes.

Enfin, nous présenterons au chapitre 4 plusieurs expériences permettant de démontrer les capacités d'adaptation au contexte visuel de notre système. Nous montrerons comment l'utilisation de méthodes d'évolution guidée peut améliorer la performance des algorithmes évolués et accélérer leur développement. Nous testerons les capacités de généralisation de ces algorithmes

et montrerons comment une modification structurelle peut améliorer celles-ci. Nous validerons également ces résultats sur un robot réel se déplaçant dans les couloirs de notre laboratoire. Pour finir, nous ouvrirons des pistes pour de futures recherches visant à améliorer ce système et à l'appliquer à d'autres tâches.

Les principales contributions de cette thèse sont :

- la prise en compte du processus de vision dans sa globalité
- la définition formelle d'une structure d'algorithmes de vision facilement extensible et réutilisable
- le développement des méthodes associées permettant de faire évoluer des algorithmes efficaces basés sur cette structure

Chapitre 1

Systèmes de vision adaptatifs pour la robotique

Le contexte global de cette thèse est la conception d'un système de vision adaptatif pour la navigation des robots. Nous allons présenter dans ce premier chapitre différentes approches de la vision pour la robotique, en nous concentrant sur celles qui permettent au robot d'apprendre ou de s'adapter à son environnement. Cet état de l'art est loin d'être exhaustif, les travaux dans ce domaine étant nombreux depuis vingt-cinq ans. Le but est surtout de donner un aperçu des différentes méthodes qui ont été employées dans ce domaine, de leurs avantages et de leurs limitations. Nous essaierons ensuite de situer cette thèse par rapport à ces approches.

1.1 Contexte et motivations

1.1.1 Motivations pour l'utilisation de la vision monoculaire

À l'heure actuelle, les fonctions de navigation des robots utilisent le plus souvent des capteurs de distance (sonars, télémètres lasers, capteurs infrarouges) et / ou des capteurs faisant appel à des éléments extérieurs (balises, GPS). Le principal avantage de ces systèmes est que l'information de position ou de distance est très facile à calculer. Ils présentent néanmoins plusieurs inconvénients.

Les capteurs de distance sont des capteurs actifs ce qui les rend généralement plus gourmands en énergie et plus facilement détectables. Cela peut être rédhibitoire pour des applications militaires par exemple. De plus, les sonars et les capteurs infrarouges souffrent d'un manque de précision. Ils sont généralement utilisés uniquement pour détecter des obstacles proches. Les télémètres lasers sont quant à eux beaucoup plus précis mais ils sont potentiellement dangereux et restent relativement chers.

Les capteurs utilisant un élément extérieur sont par définition dépendants du système de positionnement. L'utilisation de balises limite les applications du système aux environnements préparés, et l'installation des balises nécessite en soi une source extérieure d'énergie et un accès à différentes zones de l'environnement. Cela exclut donc l'exploration de zones inconnues. De plus, le robot devient incapable de se positionner si une balise est détruite ou ne fonctionne plus.

Le GPS est plus fiable mais il est inutilisable en environnement intérieur ou dans un contexte d'exploration planétaire.

À l'inverse, les caméras sont aujourd'hui des capteurs bon marché et peu gourmands en énergie. Elles fournissent de plus une information très riche sur l'environnement du robot. Le fait que la vision soit également très utilisée par les animaux et les humains pour se localiser et naviguer dans leur environnement tend à montrer qu'il s'agit d'une capacité sous-exploitée en robotique à l'heure actuelle. Bien sûr, les données visuelles sont complexes à traiter pour en extraire une information utilisable par un système de navigation. C'est cette difficulté que nous cherchons à surmonter dans cette thèse.

Notons également qu'un grand nombre de systèmes de vision, aussi bien en robotique que dans la nature, utilisent en fait la stéréovision. Cela consiste à mettre en correspondance deux images ou plus, issues de capteurs distants généralement de quelques centimètres, afin d'extraire une information de distance. Nous avons choisi au contraire de nous limiter à la vision monoculaire pour différentes raisons :

- Les systèmes de stéréovision nécessitent plusieurs caméras, précisément positionnées et calibrées. Ils sont donc plus chers et plus complexes à mettre en place.
- La nature nous montre qu'il est tout à fait possible de se déplacer dans un environnement sans utiliser la stéréovision. Une grande partie des animaux ont les yeux placés sur les côtés de la tête. Cela leur procure un champ de vision très large pour détecter au mieux les prédateurs mais le champ de recouvrement permettant la stéréovision est alors quasi-nul. Cela ne les empêche pas pour autant de se déplacer efficacement dans leur environnement. De plus, le rapprochement des yeux chez la plupart des animaux capables de stéréovision ne permet pas d'utiliser celle-ci sur des distances supérieures à un ou deux mètres. Enfin, un animal ou un humain borgne est tout à fait capable de s'orienter et de se déplacer, même dans un environnement inconnu et sans période d'adaptation.
- La mise en correspondance d'une paire d'images stéréoscopique ne nécessite pas une puissance de calcul moindre en général. Si l'on considère à nouveau l'œil des animaux, il a été montré que des traitements monoculaires rétiniens très simples sont suffisants pour réaliser certaines tâches, comme la détection de contraste ou de mouvement (voir [Lettvin 59] par exemple). La stéréovision est réalisée par des traitements plus complexes dans le cortex visuel, et nécessite de toute manière un pré-traitement monoculaire important.
- Une des plates-formes cibles pour notre système est une rétine artificielle numérique programmable que nous présenterons rapidement à la fin de ce chapitre. Il s'agit d'un capteur de vision "intelligent", c'est-à-dire que le traitement de l'image est effectué directement sur le capteur. Ce genre de système perd de son intérêt s'il faut utiliser plusieurs images issues de différents capteurs.

1.1.2 Apprentissage et adaptation

Quels que soient les capteurs utilisés, la plupart des systèmes robotiques utilisent encore aujourd'hui un programme codé "en dur" au préalable pour accomplir une tâche donnée. Ce type de programme peut s'avérer très efficace et fiable tant que l'environnement du robot ne change pas. La plupart des robots utilisés dans l'industrie sont ainsi préprogrammés, ce qui n'empêche

pas la réalisation de tâches complexes comme par exemple le positionnement précis d'un bras robot par rapport à des amers visuels (problème de l'asservissement visuel) [Chaumette 06].

Cependant, ces systèmes sont souvent complexes à programmer et fonctionnent très mal lorsque l'environnement change ou en cas d'événement imprévu. Dans le cas d'un robot mobile, les conditions et l'environnement sont presque toujours variables. Les recherches dans ce domaine s'intéressent donc aux mécanismes d'apprentissage et d'adaptation que l'on peut observer dans la nature pour concevoir des robots plus robustes aux changements et imprévus.

L'apprentissage peut toutefois avoir lieu à différents moments et de différentes manières. Les systèmes de type SLAM (*Simultaneous Localization And Mapping*) permettent au robot d'apprendre de manière autonome la carte de son environnement et de se localiser dans celui-ci [Davison 03]. Le comportement du robot est cependant toujours pré-codé et ne pourra donc pas s'adapter en fonction de l'environnement.

Les systèmes d'apprentissage *offline* sont utilisés pour programmer le robot de manière plus souple et plus facile. Il s'agit typiquement de systèmes d'apprentissage supervisé où un opérateur va "montrer" au robot la tâche à effectuer. Le programme utilisé par le robot est alors développé ou paramétré automatiquement pour accomplir cette tâche mais il ne pourra plus être modifié par la suite pendant le fonctionnement du robot.

Enfin, les systèmes d'apprentissage ou d'adaptation *online* sont conçus pour que le robot puisse changer son comportement pendant le fonctionnement afin de s'adapter à de nouvelles conditions. Il s'agit le plus souvent de systèmes d'apprentissage non supervisé où une fonction de coût va indiquer au robot si son comportement actuel est adapté ou non.

1.2 Les différents aspects de la vision pour la robotique

La réalisation d'un état de l'art sur les travaux en vision pour la robotique est une tâche quelque peu ardue, tout d'abord car les recherches sur ce sujet ont été très nombreuses, mais surtout car il ne s'agit pas d'un domaine de recherche clairement défini. Les problématiques de vision en robotique se situent à l'intersection entre plusieurs domaines très différents comme le traitement d'images, la planification, la construction de cartes ou la compréhension de la vision d'un point de vue neurologique. Un état de l'art sur ces recherches risque donc fort de ressembler à un inventaire à la Prévert rassemblant des travaux sans véritable lien entre eux.

Un bon point de départ pour tenter de classifier ces travaux est la revue réalisée par DeSouza sur ce sujet, qui n'est pas très récente mais présente bien les différents aspects de la vision pour la robotique [DeSouza 02]. À partir de celle-ci, nous allons décrire rapidement les aspects qui nous semblent importants pour caractériser les différents travaux dans ce domaine.

Représentation de l'environnement. Cet aspect n'est pas directement lié à la vision et se retrouve dans tous les systèmes robotiques. Les questions qui se posent ici sont de savoir comment le robot se représente l'environnement et comment cette représentation est acquise. L'environnement peut être représenté sous la forme d'une carte métrique ou d'une carte topologique (c'est-à-dire sous la forme d'un ensemble de lieux reliés entre eux sans notion explicite de

distance). Le robot peut également ne disposer que d'une représentation implicite de l'environnement (par exemple un ensemble de prises de vue associées à des règles ou actions, sans lien explicite entre elles), voire n'utiliser aucune représentation auquel cas la planification d'un déplacement n'est pas possible. La représentation peut être fournie au robot au préalable, généralement sous forme de carte, ou elle peut être apprise par le robot au cours de son déplacement dans l'environnement. Les systèmes de SLAM, par exemple, se situent dans ce dernier cas.

Structure de l'environnement Un environnement structuré dispose de points de repères qui peuvent être utilisés par le robot pour se déplacer, comme par exemple le fait qu'un mur est généralement vertical ou qu'une route est une bande continue entourée de deux lignes blanches. À l'inverse, un environnement non structuré ne dispose pas de tels repères. C'est le cas par exemple pour les systèmes d'exploration planétaire comme le *Mars Pathfinder Rover*. Les systèmes de vision sont généralement conçus pour une utilisation intérieure ou extérieure, rares sont ceux qui fonctionnent dans les deux cas. Les caractéristiques principales d'un environnement intérieur sont une illumination relativement constante, un sol uniforme et plat ainsi qu'une structure géométrique constituée de couloirs, de portes et de pièces ou bureaux. Les environnements extérieurs subissent quant à eux de grandes variations d'illumination et sont généralement moins structurés. Il existe donc souvent un lien entre la structure de l'environnement et le fait qu'il soit intérieur ou extérieur mais ce n'est pas forcément le cas. Une route est un environnement extérieur structuré et à l'inverse un bâtiment effondré peut être vu comme un environnement intérieur non structuré.

Extraction ou utilisation des informations visuelles La plupart des applications de vision pour la robotique nécessitent tout d'abord d'extraire certaines informations pertinentes de l'image, puis utilisent ces informations pour une tâche donnée (contrôle, cartographie, etc.). La première phase d'extraction de l'information est plutôt un problème de traitement d'image, tandis que la seconde n'est pas forcément spécifique aux applications de vision. Les travaux en vision pour la robotique ne s'intéressent souvent qu'à une seule de ces phases, auquel cas il est utile de préciser quel est le problème traité. Notons que dans certains cas cette distinction est effectuée seulement de manière implicite, voire pas du tout.

Nous allons présenter dans les sections suivantes différentes approches de la vision pour la robotique et quelques travaux significatifs pour chacune d'elles. Nous essaierons de situer ces travaux, ainsi que cette thèse, par rapport aux aspects présentés ici.

1.3 Approches analytiques

Les approches analytiques aux problèmes de vision utilisent de manière extensive les propriétés géométriques liées à la projection de l'environnement en trois dimensions sur le plan image à deux dimensions. Cela suppose en général que les données précises de calibration de la caméra sont disponibles, même si certains travaux s'intéressent à l'utilisation d'une caméra partiellement calibrée ou à la calibration automatique.

1.3.1 Asservissement visuel

Les recherches en asservissement visuel s'intéressent à l'utilisation d'informations visuelles pour définir une loi de contrôle permettant à un robot d'atteindre une position fixée à l'avance. Cette position cible est en réalité définie par la position que prennent plusieurs amers visuels dans l'image provenant de la caméra lorsque le robot se trouve à la position cible. Le robot peut être initialement placé de manière aléatoire dans l'environnement tant que les amers visuels en question se trouvent dans son champ de vision. L'asservissement visuel permet de déterminer la loi de contrôle qui va amener le robot de sa position initiale vers la position cible.

Les premiers travaux dans ce domaine s'intéressaient principalement au positionnement d'un bras robot disposant de six degrés de liberté [Espiau 92, Corke 93]. Cependant un certain nombre de recherches depuis une dizaine d'années s'intéressent au contrôle de robots mobiles non-holonomes avec les mêmes méthodes [Chen 06, Mariottini 07]. Notons que les amers visuels définissant la position cible peuvent également se situer sur un objet mobile. Dans ce cas les techniques d'asservissement visuel sont utilisées pour effectuer du suivi de cible ou du maintien de formation entre plusieurs robots mobiles [Vidal 04].

Nous allons maintenant présenter les principes mis en œuvre par les techniques d'asservissement visuels. Il s'agit uniquement d'une introduction rapide sur le sujet. Le lecteur peut se reporter notamment à l'article de François Chaumette pour un tutoriel plus détaillé [Chaumette 06]. La figure 1.1 présente les données de base du problème d'asservissement visuel, c'est-à-dire la position initiale, la position cible et les images correspondantes.

On distingue deux approches différentes d'asservissement visuel : l'asservissement visuel basé sur la position et l'asservissement visuel basé sur l'image. Dans le premier cas, les amers sont utilisés pour déterminer la position de la caméra, et la loi de contrôle sera établie à partir de cette information de position. Dans le deuxième cas, on utilise directement la position des amers visuels dans l'image pour déduire la loi de contrôle.

Asservissement visuel basé sur la position Ce cas se ramène principalement à un problème de reconstruction 3D et à l'utilisation d'une loi de contrôle relativement simple. L'idée est ici d'utiliser les amers pour déterminer la position actuelle de la caméra dans l'espace en trois dimensions et la position cible. La loi de contrôle va chercher à réduire la distance entre ces deux positions. Il peut s'agir simplement d'effectuer une translation directe de la position initiale vers la position cible et une rotation pour placer la caméra selon l'orientation souhaitée, mais d'autres lois de contrôle sont également possibles. Le principal inconvénient de cette méthode est qu'elle ne traite pas directement le problème de la reconstruction 3D, la caméra étant en fait considérée à la base comme un capteur 3D. Des méthodes ont été proposées en traitement d'image pour résoudre ce problème mais elles nécessitent généralement une connaissance préalable de l'objet 3D sur lequel sont situés les amers visuels. Cette méthode étant particulièrement sensible aux erreurs sur l'estimation de la position de la caméra, elle est difficile à employer lorsque l'environnement n'est pas parfaitement connu.

Asservissement visuel basé sur l'image Cette technique d'asservissement visuel utilise directement les coordonnées des amers visuels dans le plan image afin de déterminer la loi de contrôle. Plus précisément, on va chercher à exprimer le déplacement de ces amers dans l'image

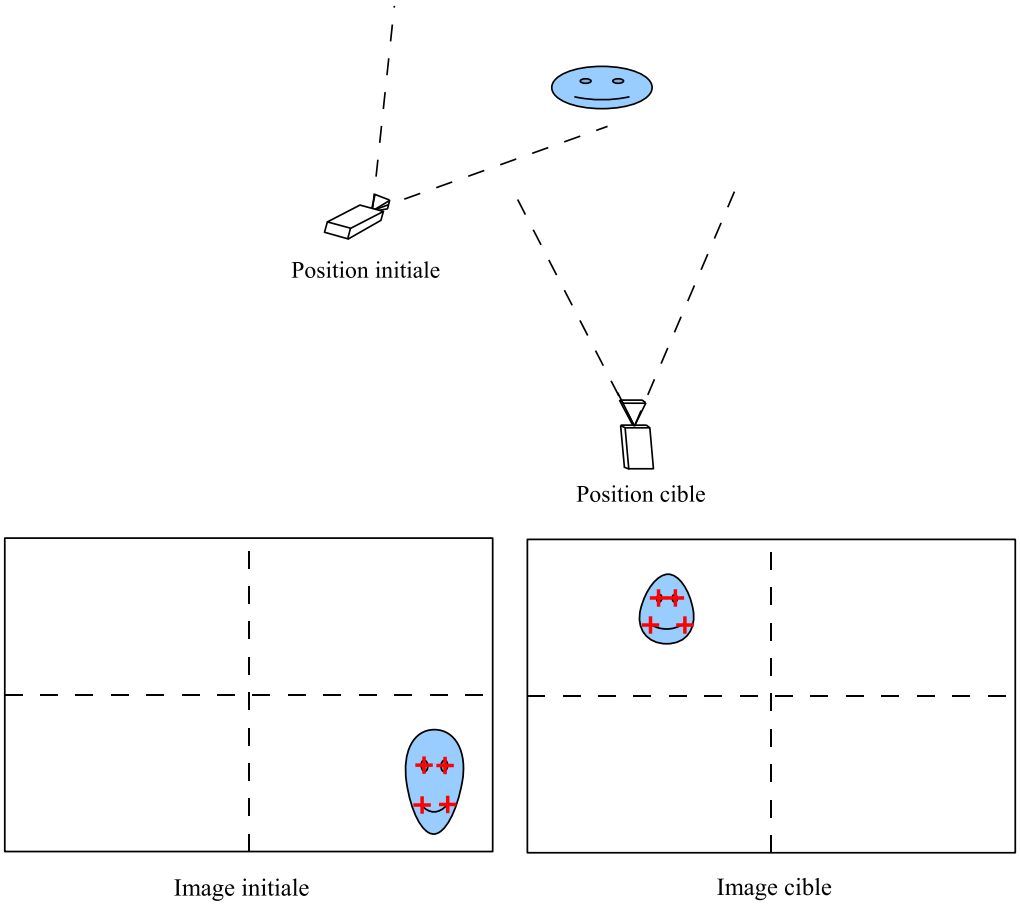


FIG. 1.1 : *Représentation du problème d'asservissement visuel. Les croix rouges représentent les amers visuels qui vont être utilisés. Le but est de déterminer la loi de commande qui va permettre d'amener ces amers de leur position initiale dans l'image vers la position cible.*

en fonction du déplacement de la caméra. Il faut pour cela disposer des paramètres de calibration de la caméra et également d'une approximation de la profondeur des points dans l'image. Cette estimation de la profondeur nécessite également l'utilisation de techniques de reconstruction 3D, mais cette méthode s'avère beaucoup plus robuste que la précédente aux imprécisions dans cette estimation. Les erreurs à ce niveau ralentiront la convergence vers la position cible mais n'auront que peu d'impact sur la précision de la position finale. À partir de cette analyse, la loi de contrôle va déterminer le déplacement à effectuer pour réduire l'erreur sur la position des amers dans l'image. Un inconvénient majeur de cette méthode est qu'il existe des minima locaux dans cette fonction d'erreur qui peuvent empêcher la loi de contrôle d'amener la caméra jusqu'à la position cible.

Notons que ces deux méthodes peuvent être adaptées pour une utilisation en stéréovision, auquel cas la reconstruction 3D est grandement facilitée. Essayons à présent de situer les approches d'asservissement visuel en fonction des aspects de la vision pour la robotique présentés au §1.2. Tout d'abord, ces approches n'utilisent pas de représentation cartographique de l'environnement. Elles nécessitent toutefois certaines informations préalables comme l'image acquise

par la caméra à la position cible et éventuellement une connaissance de l'objet 3D utilisé comme repère visuel. Cela limite l'utilisation de ce genre de technique aux cas où l'environnement est accessible et connu au préalable. Par contre, elles n'utilisent pas d'autres informations sur la structure de l'environnement, ce qui les rend utilisables en environnement intérieur comme extérieur, structuré ou non. Enfin, ces approches ne répondent pas au problème d'extraction de l'information visuelle, elles considèrent la caméra comme un capteur fournissant directement la position des amers visuels.

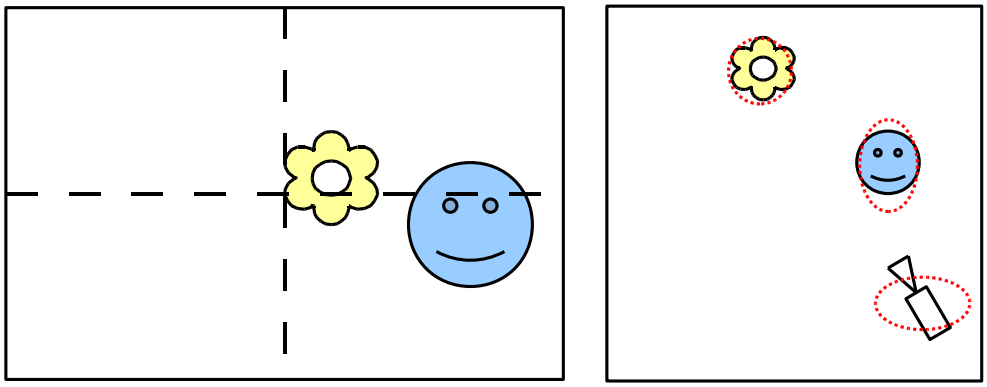
En résumé, ces méthodes servent au positionnement précis d'un robot pendant ce qu'on peut appeler une phase d'approche, durant laquelle un repère visuel donné est toujours visible. Une application industrielle typique est le positionnement d'un bras robot par rapport à une pièce à usiner. Ces méthodes peuvent aussi être utilisées en robotique spatiale pour l'arrimage d'un module à une station spatiale par exemple. En robotique mobile, des applications possibles sont le positionnement du robot sur sa station de rechargement, le suivi d'un autre robot ou le maintien en formation de plusieurs robots. Ce type de méthode peut aussi être utilisé pour contrôler le robot à travers une suite de *waypoints* représentés par des images prises par la caméra en différents lieux. Cependant il est nécessaire dans ce cas d'acquérir au préalable un certain nombre d'images assez rapprochées les unes des autres et ces méthodes sont mal adaptées pour faire face à des environnements variables. De ce fait, l'asservissement visuel est peu utilisé pour des tâches de navigation.

1.3.2 Localisation et cartographie basée sur la vision (*Vision-Based SLAM*)

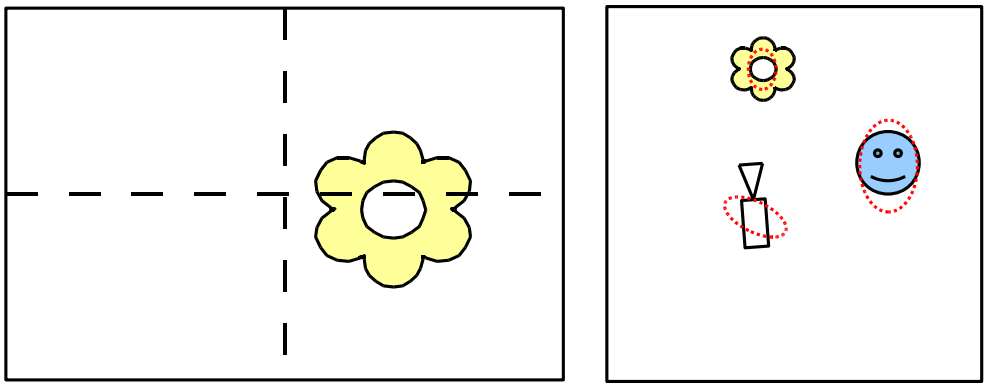
Cette approche a pour but d'effectuer simultanément la construction d'une carte de l'environnement dans lequel se déplace la caméra et la localisation de la caméra dans celle-ci. Elle s'inspire d'un côté des travaux de localisation et cartographie simultanée (*Simultaneous Localization And Mapping* ou *SLAM*) qui se basent généralement sur des capteurs de distance (voir [Dissanayake 01] par exemple). D'un autre côté, cette approche reprend un certain nombre de concepts développés dans les travaux de reconstruction 3D à partir du mouvement (*Structure From Motion* ou *SFM*, voir [Pollefeys 99] par exemple) en leur ajoutant la notion de traitement en temps réel. Cette approche de SLAM basée sur la vision est apparue il y a une dizaine d'années avec les travaux d'Andrew Davison principalement [Davison 03, Davison 07]. Elle est très utilisée aujourd'hui car elle représente un moyen pratique et efficace d'effectuer de la cartographie et de la localisation en utilisant uniquement la vision.

Il s'agit d'une approche bayésienne, les positions de la caméra et des différents repères visuels de l'environnement étant représentées avec une certaine incertitude. Ces incertitudes sont de plus liées entre elles : si l'on a observé plusieurs repères proches dans l'environnement, une diminution dans l'incertitude sur la position de l'un d'eux nous permettra également de diminuer l'incertitude sur la position des autres. Cette idée générale est représentée sur la figure 1.2.

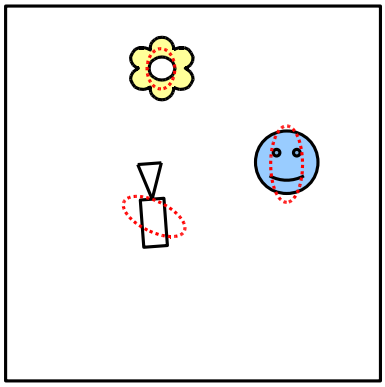
En pratique, toutes les positions, les incertitudes et les corrélations entre elles sont représentées sous la forme d'un seul vecteur d'état et d'une matrice de covariance complète. L'idée est de mettre à jour ce vecteur et cette matrice à chaque nouvelle image pour prendre en compte les nouvelles informations. Cette mise à jour est effectuée en utilisant un filtre de Kalman étendu [Welch 95]. Cette méthode présente plusieurs avantages :



(a) *Observation de deux repères visuels proches et estimation de leurs positions et de celle de la caméra*



(b) *Nouvelle observation d'un des repères visuels et réduction de l'incertitude sur sa position et sur celle de la caméra*



(c) *Réduction de l'incertitude sur la position du deuxième repère visuel*

FIG. 1.2 : *Illustration simplifiée du processus de cartographie et localisation simultanées en utilisant la vision. Les ellipses en pointillés rouges représentent les incertitudes sur les positions.*

- L'estimation des nouvelles positions et la mise à jour du vecteur d'état et de la matrice de covariance ne nécessitent que les données à l'étape précédente et les nouvelles observations. Cela permet donc un traitement en temps réel des données, contrairement aux techniques de SFM qui nécessitent de traiter l'ensemble de la séquence d'images pour extraire les informations.
- Les incertitudes sur les mesures et sur le processus en cours étant prises en compte de manière explicite, le système sera plus adapté à l'utilisation de capteurs bruités qu'un système supposant des mesures exactes.
- Cette méthode facilite grandement la prise en compte de plusieurs types de capteurs. Pour un robot mobile roulant, on peut par exemple fusionner les données de la caméra et celles de l'odométrie pour obtenir des estimations de position plus précises.

Les méthodes de SLAM présentent également certains inconvénients qui font toujours l'objet de recherches :

- La taille de la matrice et la complexité de la mise à jour à chaque étape varient en $O(N^2)$, N étant le nombre de repères visuels présents dans la carte. En pratique, cela signifie qu'il devient difficile d'effectuer du SLAM en temps réel au-delà de quelques centaines de repères dans la carte. Plusieurs méthodes ont été proposées pour tenter de réduire cette complexité (voir [Montemerlo 02] par exemple).
- Un autre problème, commun à toutes les méthodes de localisation et de cartographie, est celui de la détection de fermeture de boucle et de la réinitialisation du système. Ce problème provient du fait qu'après un long trajet dans l'environnement sans retour en arrière, l'incertitude sur la position du robot va croître. Lorsque le trajet effectué est une boucle, il devient difficile de détecter quand le robot revient à un endroit qu'il avait déjà visité auparavant. Cette détection est pourtant cruciale pour ne pas créer de doublons dans la carte et causer d'importantes erreurs de positionnement. Là aussi, des méthodes ont été proposées pour résoudre ce problème (voir [Angeli 08] par exemple) mais il s'agit encore d'un sujet de recherche ouvert car la détection de fermeture de boucle doit être accompagnée d'une mise à jour majeure de la carte qui n'est pas triviale.

Si l'on considère les différents aspects sur la vision pour la robotique présentés au §1.2, il est clair que la question de la représentation de l'environnement est centrale pour les méthodes de SLAM basées sur la vision. Cette représentation est ici construite par le système lui-même, généralement sous la forme d'une carte métrique. Notons toutefois que dans certains cas la carte est décomposée de manière hiérarchique avec une carte topologique au plus haut niveau et des sous-cartes métriques à chaque nœud topologique. La structure de l'environnement n'est pas utilisée en général, ce qui rend ces systèmes adaptés à tout type d'environnement. Ces systèmes ne traitent généralement pas le problème de l'extraction des informations visuelles et se concentrent plutôt sur l'utilisation qui en est faite, c'est à dire la cartographie et la localisation. Le choix des primitives visuelles utilisées est issu d'un compromis entre la robustesse de la détection et la complexité algorithmique. Dans ses travaux, Davison utilise le détecteur de Shi et Tomasi [Shi 94] avec des fenêtres assez larges (11×11 pixels typiquement). D'autres travaux utilisent plutôt les *features* SIFT (*Scale Invariant Feature Transform*, [Lowe 04]), plus robustes mais plus longs à calculer. Notons finalement que les travaux de SLAM n'abordent pas du tout le problème du contrôle du robot, qui est généralement effectué manuellement.

1.4 Approches bio-inspirées

Si les approches présentées jusqu'à maintenant se basaient principalement sur une analyse mathématique des propriétés géométriques liées à la projection sur le plan image, d'autres travaux sont partis des connaissances biologiques dont l'on dispose sur la vision pour les appliquer à la robotique. Les objectifs recherchés sont souvent similaires mais les méthodes utilisées pour les atteindre sont ici radicalement différentes.

Les travaux en robotique bio-inspirée s'inscrivent en général dans l'approche Animat, initiée par Jean-Arcady Meyer, Stewart Wilson et Agnès Guillot au début des années 90. L'idée est de s'inspirer des comportements animaux observés dans la nature pour concevoir des robots réellement autonomes, capables de "survivre" et de s'adapter à un environnement changeant pour accomplir une tâche [Meyer 91, Wilson 91]. L'autre idée associée à ce principe est d'utiliser la modélisation et les expérimentations avec des robots pour mieux comprendre le fonctionnement des systèmes biologiques. L'ambition de l'approche Animat est ainsi d'explorer la frontière entre systèmes biologiques et systèmes artificiels. Cependant la plupart des travaux en robotique bio-inspirée ne s'intéressent que peu au problème de la vision, nous nous concentrerons ici uniquement sur ceux qui en font une utilisation majeure.

1.4.1 Utilisation du flux optique inspirée par le vol des abeilles

Des travaux en biologie expérimentale ont montré que certains insectes, et notamment les abeilles, se basent en grande partie sur le mouvement apparent pour se guider dans leur environnement. L'idée de base est que lorsqu'une abeille ou un robot se déplace, le mouvement perçu des objets qui l'entourent sera d'autant plus important que l'objet en question est proche. Il s'agit du principe bien connu de parallaxe. Il a été montré que les abeilles peuvent utiliser cette information de mouvement apparent notamment pour se centrer dans un couloir (évitement d'obstacles), mesurer la distance parcourue (odométrie visuelle) ou réguler leur vitesse en phase d'atterrissage [Srinivasan 96].

En robotique, le mouvement apparent est généralement représenté sous la forme de flux optique, c'est à dire d'un champ de vecteurs où chaque vecteur correspond au mouvement perçu en un point donné de l'image. Nous présenterons au §2.1.1 les méthodes de calcul du flux optique et les différentes manières de l'utiliser. Nous allons uniquement présenter ici quelques travaux montrant comment l'utilisation de principes issus de l'observation du vol des abeilles peut permettre de concevoir des systèmes robotiques efficaces.

Santos-Victor a ainsi conçu un système utilisant deux caméras montées de manière à pointer latéralement par rapport au déplacement du robot (stéréovision divergente) [Santos-Victor 93, Santos-Victor 95]. Le principe du déplacement est de tourner à gauche lorsque le mouvement perçu est plus important à droite et inversement. De plus, la vitesse adoptée par le robot est inversement proportionnelle au mouvement perçu moyen. Santos-Victor a montré que ce robot pouvait de cette manière se centrer automatiquement au milieu d'un couloir et adapter sa vitesse en fonction de la largeur de celui-ci, deux comportements qui ont également été observés chez les abeilles.

Des travaux similaires ont été réalisés par Coombs avec ici deux caméras frontales disposant d'angles de vision différents [Coombs 98]. La caméra avec un angle de vision large (environ

115°) est utilisée comme précédemment pour calculer la différence entre le mouvement perçu du côté gauche et du côté droit du robot. La caméra avec un angle de vision étroit (environ 45°) est utilisée pour détecter les obstacles frontaux en calculant la divergence du flux optique. Le robot peut ainsi se déplacer dans les couloirs du laboratoire durant plusieurs minutes sans heurter d'obstacles. Plus récemment, ce type d'expérience a été réalisé avec des robots volants. Laurent Muratet a ainsi utilisé le flux optique pour piloter un hélicoptère en simulation avec une seule caméra [Muratet 05]. L'angle de vue doit dans ce cas être assez élevé, notamment pour éviter les problèmes liés aux angles morts. Stefan Hrabar a quant à lui utilisé un hélicoptère miniature réel et comparé l'utilisation d'une caméra omnidirectionnelle et de deux caméras latérales. Il a aussi couplé son système à base de flux optique avec un système de stéréovision afin de mieux détecter les obstacles frontaux [Hrabar 06]. Enfin, Jean-Christophe Zufferey a montré qu'il est possible d'utiliser le flux optique pour l'évitement d'obstacles avec deux caméras 1D montées latéralement sur un drone ultra-léger de 30 grammes [Zufferey 06].

Une autre utilisation possible du flux optique est l'odométrie visuelle. Cela consiste à intégrer le mouvement perçu sur le temps afin de déterminer la distance parcourue. Iida a ainsi fait la démonstration de ce type de système sur un robot volant de type dirigeable [Iida 03]. Comme pour tout système d'odométrie, les erreurs sont intégrées au fur et à mesure du temps et les trajectoires reconstruites finissent par s'éloigner des trajectoires réelles. Cependant, ce type de système est insensible aux glissements et blocages de roues qui dégradent fortement les performances des systèmes d'odométrie classiques. Le couplage des deux permet ainsi de créer un système d'odométrie plus précis que l'un des deux pris indépendamment. Corke a comparé les performances d'un système d'odométrie visuelle utilisant le flux optique avec un système utilisant des techniques de *Structure From Motion* assez proches des systèmes de *Visual SLAM* présentés précédemment. Ce dernier s'avère plus précis sur de longues trajectoires mais au prix d'une complexité plus élevée [Corke 04].

Ces systèmes n'utilisent pas et ne construisent pas de représentation de l'environnement, même si les trajectoires issues de l'odométrie visuelle pourraient ensuite être utilisées pour de la cartographie. Ils n'utilisent pas forcément la structure de l'environnement, mais dans le cas des robots roulants ils supposent généralement que le sol est plat et que donc la composante verticale du flux optique latéral n'est pas pertinente. De plus, le flux optique ne peut être calculé correctement que si l'environnement est suffisamment texturé, ces systèmes ne sont donc pas totalement indépendants de la structure de l'environnement. Enfin, les méthodes présentées ici traitent à la fois le problème de l'extraction de l'information visuelle (calcul du flux optique) et celui de son utilisation (contrôle du robot ou odométrie).

1.4.2 Utilisation de cellules de lieu pour la navigation d'un robot

Des recherches en neurobiologie ont mis en évidence l'existence de neurones particuliers chez le rat, situés plus particulièrement dans l'hippocampe, permettant de coder une représentation spatiale de l'environnement [O'Keefe 78]. Ces cellules de lieu (*place cells*) s'activent lorsque le rat se trouve à un emplacement donné dans l'environnement. D'autres cellules s'activent de manière similaire en fonction de la direction à laquelle la tête du rat fait face. Il a été montré que ces neurones jouent un rôle important pour la localisation et la navigation chez le rat.

Plusieurs travaux ont été menés depuis pour tenter de retranscrire ce fonctionnement sur un robot. Le but de cette approche est double : Il s'agit premièrement de valider et d'affiner les modèles issus de la neurobiologie en les confrontant à une expérimentation pratique. Deuxièmement, ce type de système peut fournir au robot une méthode robuste et adaptative pour représenter son environnement, planifier des trajectoires et naviguer dans celui-ci.

Les travaux d'Angelo Arleo combinent ainsi des informations visuelles (allothétiques) avec des données de déplacement (idiothétiques) et codent les différents lieux sur des neurones de type cellules de lieu [Arleo 04]. L'information visuelle est ici extraite à des positions fixes de l'image selon un maillage représentant la densité des récepteurs sur la rétine (c'est à dire plus denses dans la partie centrale que sur la périphérie). Le codage lui-même se fait à l'aide d'un jeu de filtres de Gabor répondant à différentes fréquences et orientations. Il montre également qu'il est possible après cet apprentissage d'entraîner le robot pour qu'il se dirige vers un lieu donné de l'environnement avec des techniques de type apprentissage par renforcement. Cet apprentissage est de plus adaptatif : Si l'on ajoute un obstacle dans l'environnement, le robot est capable de mettre à jour les trajectoires à utiliser pour atteindre le but.

Des travaux similaires ont été réalisés par Philippe Gaussier et son équipe avec une attention plus particulière portée au codage des transitions entre les différents lieux. Cette représentation permet d'effectuer de la planification de trajectoire et d'apprendre au robot à se diriger vers différents points de l'environnement en fonction de signaux de motivation (faim, soif, etc.) [Gaussier 02]. Des développements plus récents utilisent une caméra omnidirectionnelle et détectent les repères visuels significatifs avec un filtre de type différence de gaussiennes (DoG). Plusieurs expériences sont réalisées pour tester la robustesse de l'encodage des cellules de lieu, notamment en environnement extérieur où les repères visuels sont moins nombreux et également en supprimant une partie du champ visuel [Giovannangeli 06].

La représentation de l'environnement est donc ici de type topologique et elle est de plus entièrement apprise par le robot. Ce système fonctionne en environnement structuré ou non, mais le fait que les repères visuels soient souvent moins nombreux et moins caractéristiques en environnement extérieur complique quelque peu la tâche dans ce cas. Un avantage de ce type d'approche est que le problème est traité dans son ensemble, depuis l'extraction des informations visuelles jusqu'à la représentation de l'environnement et son utilisation pour une tâche donnée. Ces systèmes sont de plus adaptatifs, c'est-à-dire beaucoup plus à même de faire face à des changements dans l'environnement que ceux présentés jusqu'à maintenant. Le principal inconvénient est que la représentation sous forme neuronale peut être difficile à interpréter d'un point de vue extérieur. Cela complique nettement les interactions avec d'autres types de systèmes ou des humains. Il serait par exemple très difficile de fournir une carte déjà réalisée de l'environnement au robot ou à l'inverse d'extraire du système une carte interprétable par un humain.

1.4.3 Robotique évolutionnaire

Le principe de la robotique évolutionnaire est d'utiliser des techniques d'évolution artificielle pour développer des contrôleurs pour des robots. L'avantage de cette approche est que la représentation utilisée pour les contrôleurs est très libre. Il est possible de faire évoluer des programmes avec des techniques de type programmation génétique, mais aussi des réseaux de

neurones ou des contrôleurs à base de règles floues. De plus l'évolution artificielle est bien adaptée pour explorer de grands espaces de paramètres, surtout si ceux-ci sont liés de manière non linéaire, ce qui est souvent le cas dans les applications robotiques. Nous reviendrons plus longuement sur les principes des algorithmes évolutionnaires et de la programmation génétique dans les chapitres suivants. Nous allons uniquement présenter ici quelques travaux significatifs dans ce domaine.

La robotique évolutionnaire a été un domaine de recherche très actif dans les années 90. La revue de Maja Matarić et Dave Cliff sur le sujet donne une vision assez complète des travaux de l'époque [Matarić 96]. Notons que le processus d'évolution se déroule le plus souvent en simulation car les conditions d'évaluation sont plus faciles à contrôler et le temps peut être accéléré. Cependant certains travaux ont cherché à faire évoluer des contrôleurs directement sur un robot réel. Par exemple, Floreano et Mondada ont fait évoluer des contrôleurs d'évitement d'obstacles utilisant des capteurs infrarouges et un comportement de guidage vers une source de lumière directement sur un robot Khepera [Floreano 96]. Toutefois l'évolution durait ici 10 jours, ce qui rend la reproduction des expériences assez difficile. Cette notion d'évolution *online* (sur le robot) ou *offline* (en simulation) est centrale en robotique évolutionnaire, de même que la notion d'évolution comme apprentissage d'une tâche ou d'évolution au cours de l'utilisation (adaptation) [Walker 03].

Les travaux en robotique évolutionnaire utilisent probablement encore moins la vision comme source d'information que les autres travaux en robotique. Une explication pour cela est sans doute que les traitements liés à la vision sont généralement assez lourds d'un point de vue computationnel et qu'ils ralentissent grandement l'évaluation des contrôleurs. Le processus d'évolution nécessitant quelques milliers ou dizaines de milliers d'évaluations, il est clair que l'utilisation de la vision peut considérablement ralentir l'évolution des contrôleurs. Nous allons présenter ici quelques travaux qui font néanmoins une utilisation importante de la vision.

Les premiers à utiliser des méthodes évolutionnaires pour créer des contrôleurs basés sur la vision furent probablement Dave Cliff, Inman Harvey et Phil Husbands [Harvey 94, Cliff 97]. Leurs premières expériences furent réalisées en simulation avec seulement deux capteurs de luminosité en guise de système de vision. Néanmoins ils transposèrent par la suite leur système sur un vrai robot disposant d'une caméra. Pour effectuer des expériences sur un robot réel sans avoir à se soucier des problèmes de batteries et de repositionnement, ils utilisent un système suspendu à un cadre permettant un déplacement sur deux dimensions. La caméra est dirigée vers le bas où un miroir rotatif orienté à 45° permet de reproduire ce que percevrait un robot roulant réel. La vision sur ce système est limitée à trois valeurs moyennes calculées en des zones données de l'image de base de taille 64 $\times$ 64. Cependant la taille et la position de ces zones sont paramétrées par l'évolution. Le système de contrôle, également paramétré par l'évolution, est un réseau de neurones qui produit en sortie les commandes moteur pour le déplacement du robot. Les contrôleurs évolués sur ce système peuvent ainsi se diriger vers une zone blanche dans l'environnement qui est plus sombre, ou suivre un cylindre blanc se déplaçant dans cet environnement.

Davide Marocco et Dario Floreano ont quant à eux utilisé des méthodes évolutionnaires pour créer des contrôleurs de vision active [Marocco 02]. L'évolution se déroule sur un robot réel de type Koala dans une arène préparée de 2 mètres de côté avec des murs blancs de 30 cm de haut. Les images de la caméra sont sous-échantillonnées pour correspondre à une rétine de 5 $\times$ 5 pixels qui sert d'entrée au système. Le contrôleur est un réseau de neurones dont les sorties

correspondent aux commandes moteur pour le déplacement du robot et aux commandes de la caméra qui est de type *pan-tilt*. Les contrôleurs évolués sont ainsi capables de se déplacer dans l'arène sans heurter les murs. De plus, l'évolution a permis de sélectionner une stratégie pour que la caméra suive les repères visuels de l'environnement importants pour cette tâche. Dans ce cas, il s'agit du bord qui sépare les murs blancs du sol plus sombre. Ces travaux en vision active ont également été utilisés pour effectuer de la reconnaissance de formes, piloter une voiture dans une simulation de course automobile et sélectionner des repères visuels notamment [Floreano 04, Suzuki 07].

Plus récemment, Byron Boots a utilisé des méthodes évolutionnaires avec un simulateur très simple pour mettre en évidence le fait qu'un système visuel évolué n'est pas générique mais qu'il est en réalité adapté pour les situations rencontrées durant l'apprentissage [Boots 07]. Il utilise pour cela différents environnements dans lesquels les perceptions visuelles sont similaires alors que la géométrie est différente. Il s'inspire ici d'illusions d'optiques du type chambre d'Ames pour construire ces environnements. Il montre notamment que les contrôleurs évolués dans un environnement présentant une géométrie normale produisent des trajectoires très particulières dans les environnements à géométrie déformée. Ces comportements particuliers sont comparables à des observations faites sur des sujets humains dans ce type d'environnement.

Un point commun à tous ces travaux est que la perception visuelle est en fait très simplifiée, se résumant en général à quelques valeurs moyennes mesurées sur l'image. Comme indiqué précédemment, cela peut s'expliquer par le souci de restreindre la complexité des traitements pour que les évaluations restent relativement rapides. Dans le cas des réseaux de neurones, cela permet aussi de limiter la taille du réseau et donc d'obtenir des résultats plus facilement interprétables. Cependant cela ne permet pas d'utiliser les informations contenues dans l'image à une résolution plus fine, comme typiquement les informations de texture. Notons toutefois que d'autres travaux en robotique évolutionnaire ont utilisé des images en haute résolution, comme par exemple le sonar visuel de Martin que nous détaillerons au §3.1.3.3 [Martin 06].

Jusqu'à aujourd'hui, les travaux en robotique évolutionnaire s'intéressent plutôt aux fonctions de navigation de base et donc n'utilisent pas et ne créent pas de représentation de l'environnement. Ils ne font généralement pas usage d'informations particulières sur la structure de l'environnement, ou plutôt ils essaient d'apprendre automatiquement quelles sont ces propriétés de structure qui sont intéressantes pour une tâche donnée. Enfin, ces travaux couvrent l'ensemble des traitements, depuis l'extraction d'informations visuelles jusqu'à leur utilisation, généralement sans faire de distinction explicite entre ces deux phases.

Notons pour conclure cet état de l'art qu'un certain nombre de travaux en vision pour la robotique ne s'inscrivent pas dans les approches présentées ici. Il s'agit en général de systèmes ad-hoc conçus uniquement pour répondre à un problème donné, parfois en utilisant certaines des techniques que nous avons présentées dans ce chapitre. Ces travaux n'ouvrent pas en général de nouvelles pistes de recherche, c'est pourquoi nous ne les détaillerons pas dans cette thèse. Cependant certains ont abouti à des systèmes très efficaces, par exemple en conduite automatique sur route.

1.5 Bilan et objectifs de la thèse

L'objectif principal de cette thèse est la conception d'un système permettant d'adapter les capacités du robot en fonction du contexte visuel. Dans notre approche, ce terme de contexte visuel regroupe les informations liées à la structure de l'environnement dans lequel évolue le robot et celles de plus bas niveau liées à l'aspect des objets dans cet environnement (texture, illumination, etc.). La grande diversité de ces informations visuelles et des manières de les exploiter nous a poussé à nous intéresser aux techniques d'évolution artificielle, bien adaptées pour gérer des représentations à structure variable et explorer de grands espaces d'état. Cette thèse s'inscrit donc directement dans le cadre de la robotique évolutionnaire que nous avons présentée au §1.4.3. Comme nous l'avons indiqué, l'utilisation de ces techniques de manière *online* sur un robot réel présente un certain nombre d'inconvénients en terme de durée, d'autonomie du robot et de reproductibilité des expériences. Nous avons donc choisi de rester dans le cadre des systèmes d'évolution *offline*. De ce fait, il s'agit plus d'apprentissage que d'adaptation au sens où nous les avons défini au §1.1.2. Cependant un certain nombre de techniques développées ici sont également utilisables dans le cadre d'une adaptation *online*. Nous présenterons à la fin de cette thèse plusieurs pistes pour l'adaptation en temps réel du comportement du robot en fonction de l'environnement visuel.

Pour effectuer une tâche de manière autonome, un robot mobile doit avant tout être capable de se déplacer de manière sûre sans assistance externe. Cela implique de disposer d'une fonction d'évitement d'obstacles rapide et fiable, en particulier lors de l'exploration d'un environnement inconnu. Dans l'idéal cette fonction doit de plus être suffisamment générique ou adaptative pour faire face à un environnement changeant contenant des obstacles mobiles, qui seront le plus souvent des humains. Cela nous a motivé à choisir l'évitement d'obstacles comme première application de notre système. Il existe différentes techniques d'évitement d'obstacles basées sur la vision monoculaire que nous présenterons en détail au §2.1. Aucune des techniques existantes ne permet au robot d'éviter les obstacles dans toutes les conditions et tous les environnements. Cela rend ce problème d'autant plus intéressant à étudier et à tenter de résoudre en prenant en compte le contexte visuel. Nous prendrons toutefois soin de concevoir notre système de manière suffisamment générique pour pouvoir être utilisé pour d'autres applications utilisant la vision, comme par exemple la localisation ou la reconnaissance d'amers visuels. Nous indiquerons en fin de thèse de quelle manière ces problèmes peuvent être traités avec notre système.

La prise en compte du contexte visuel implique la nécessité d'inclure toute la chaîne de vision dans le processus d'apprentissage, et plus particulièrement la vision bas niveau. La plupart des travaux présentés précédemment font en réalité une utilisation très restreinte de la vision. Toute la partie concernant l'extraction d'informations depuis l'image est fixée, et l'apprentissage s'effectue uniquement sur l'utilisation qui est faite de cette information extraite. Ce parti pris est très réducteur par rapport à l'utilisation qui est faite de la vision par les animaux, où tout le processus est intimement lié au contexte visuel et à la tâche effectuée. Nous pensons qu'une approche plus globale de la vision est indispensable pour développer de vraies capacités d'apprentissage et d'adaptation sur des robots mobiles. Nous chercherons donc ici à apprendre conjointement quelle information doit être extraite de l'image et comment l'utiliser.

Comme nous l'avons indiqué précédemment, les travaux utilisant la vision en robotique évolutionnaire traitent généralement des images de quelques pixels seulement. Cela s'explique aisément d'un point de vue computationnel, mais l'utilisation d'images en haute résolution est

indispensable dans notre approche. Nous voulons en effet prendre en compte un maximum d'indices visuels, et notamment des informations de texture qui ne sont disponibles qu'en haute résolution. Heureusement, nous bénéficions de ce point de vue du développement continu de la puissance de calcul disponible, ce qui rend envisageable des expériences qui restaient impensables il y a dix ans. Notons toutefois que ce que nous entendons par "haute résolution" reste limité, puisqu'il s'agit ici d'images de 320×160 pixels. Cependant le coût computationnel reste élevé et les expériences présentées dans cette thèse durent généralement plusieurs jours. Un des inconvénients majeurs est que cela rend impossible toute analyse statistique digne de ce nom sur la variabilité des résultats obtenus.

Enfin, nous cherchons à développer des algorithmes de vision implémentables et efficaces sur différentes architectures matérielles. Nous visons en particulier les systèmes embarqués de type capteurs de vision "intelligents", comme la rétine artificielle numérique programmable développée par Thierry Bernard au laboratoire UEI de l'ENSTA [Paillet 99]. Celle-ci présente l'intérêt de pouvoir à la fois acquérir et traiter l'image en consommant extrêmement peu d'énergie. Cette propriété et son encombrement réduit en font un capteur de vision particulièrement intéressant pour la robotique. Un autre avantage de cette rétine est qu'elle facilite le traitement d'images en haute résolution, le calcul étant effectué de manière massivement parallèle. Par contre, les contraintes au niveau du jeu d'instructions et la quantité limitée de mémoire disponible la rendent plus adaptée pour certains traitements bien particuliers. Certaines primitives visuelles pourront ainsi être calculées très rapidement tandis que d'autres le seront plus lentement, voire ne pourront pas être calculées du tout. Cela nous a poussé à choisir des techniques de programmation génétique pour l'implémentation de notre système, celles-ci permettant de définir explicitement la base de primitives disponibles pour le développement des algorithmes. Notons toutefois que pour des raisons de simplicité d'implémentation et pour ne pas restreindre inutilement les capacités de notre système, les expériences présentées dans cette thèse ont été réalisées uniquement sur des plates-formes matérielles plus classiques (PC + caméra). Néanmoins le souci de la portabilité sur d'autres architectures matérielles est resté présent dans l'ensemble de notre approche.

Pour situer cette thèse par rapport aux aspects présentés au §1.2, nous concevons ici un système n'utilisant pas de représentation interne de l'environnement, même si un certain nombre de techniques pourraient être réutilisées pour de la cartographie ou de la localisation. Notre ambition est de réaliser un système pouvant s'adapter à tout type d'environnement, intérieur ou extérieur, structuré ou non. Nous ne distinguons pas explicitement la phase d'extraction d'informations visuelles de celle d'utilisation de ces informations, cependant ces deux phases seront bien présentes de manière implicite. En résumé, cette thèse a pour objectif la conception automatique d'algorithmes d'évitement d'obstacles basés sur la vision et adaptés à un contexte visuel donné.

Chapitre 2

Architecture des algorithmes de vision

Nous avons présenté précédemment différentes approches au problème de la vision pour des applications robotiques et nous avons défini à partir de celles-ci les objectifs de cette thèse. Notre but étant de développer une méthode d'apprentissage d'algorithmes de vision adaptés pour une tâche donnée, il est nécessaire avant tout de définir cette tâche. Nous avons choisi l'évitement d'obstacles comme première application de ce système car il s'agit d'une fonction primordiale en robotique mobile. Nous allons tout d'abord présenter les méthodes existantes permettant d'éviter les obstacles en utilisant la vision monoculaire. En menant cette étude, nous définirons les primitives visuelles utiles à l'évitement d'obstacles. Nous décrirons ensuite de manière formelle ces différentes primitives et la manière dont elles peuvent être agencées et combinées. Cette description formelle sera effectuée sous la forme d'une grammaire, qui constituera la base de la représentation de l'architecture des algorithmes. Pour finir, nous présenterons la méthode utilisée pour construire les algorithmes d'évitement d'obstacles à partir de cette description sous forme de grammaire. Nous aurons ainsi défini un cadre formel de description d'algorithmes de vision pour la synthèse automatique. Celui-ci constitue la base de toutes les expériences qui seront présentées au chapitre 4 de cette thèse.

2.1 Algorithmes de vision pour l'évitement d'obstacles

Nous avons mentionné au chapitre 1 le fait que la plupart des systèmes robotiques se basent sur des capteurs de distance pour la navigation. C'est encore plus particulièrement vrai des systèmes d'évitement d'obstacles car la notion de distance est centrale ici. Il serait pourtant erroné de considérer que la connaissance de la distance absolue des obstacles dans toutes les directions est nécessaire pour les éviter. Une estimation de leur distance relative est généralement tout à fait suffisante, et cette estimation peut être faite de manière implicite à partir d'autres indices contextuels. La vision monoculaire peut fournir de tels indices en grand nombre, le tout étant de pouvoir les interpréter.

Les méthodes existantes d'évitement d'obstacles basées sur la vision monoculaire se divisent en deux grandes classes. Les premières sont les méthodes basées sur le mouvement, qui utilisent en général le flux optique que nous avons présenté rapidement au §1.4.1. Nous allons détailler davantage dans cette section les méthodes permettant de calculer et d'utiliser ce flux

optique. Les secondes sont les méthodes basées sur l'apparence. Celles-ci cherchent à utiliser toutes sortes d'indices visuels non liés au mouvement comme la position du sol et des murs ou la texture des objets. Nous présenterons certaines de ces méthodes par la suite, toujours dans le but d'extraire les primitives visuelles utilisables pour l'évitement d'obstacles.

2.1.1 Méthodes basées sur le mouvement

Ces méthodes d'évitement d'obstacles se basent sur la notion de mouvement apparent dans l'image. Selon le principe de la parallaxe, lorsqu'une caméra se déplace avec un mouvement de translation dans son environnement, le mouvement apparent des différents objets sera d'autant plus important qu'ils sont proches. En calculant ce mouvement apparent, on peut donc déterminer quels sont les obstacles les plus proches et décider de la commande à utiliser pour les éviter. Nous avons présenté au §1.4.1 quelques travaux utilisant ce principe.

Le mouvement apparent est généralement représenté sous la forme de flux optique, c'est à dire d'un champ de vecteur de la taille de l'image dans lequel chaque vecteur représente le mouvement apparent du pixel correspondant dans l'image. Nous allons maintenant détailler les propriétés géométriques du flux optique et les manières de l'exploiter pour éviter les obstacles, puis nous présenterons les différentes techniques permettant de le calculer.

2.1.1.1 Propriétés géométriques et utilisation du flux optique

D'un point de vue géométrique, le flux optique est la projection sur l'image du mouvement des objets dans le référentiel de la caméra. Pour les applications robotiques, on considère généralement que la caméra est fixée sur un robot qui se déplace et que les autres objets de l'environnement sont immobiles. Le mouvement apparent est donc uniquement causé par le déplacement du robot. Dans le cas d'un robot roulant, on considère également que tous les déplacements se font dans un plan horizontal et que l'axe de la caméra est horizontal (figure 2.1). Notons qu'il s'agit d'une simplification parfois un peu abusive, car les vibrations subies par le robot lorsqu'il se déplace peuvent faire varier la direction de l'axe optique aussi bien verticalement qu'horizontalement. Cependant l'influence de ces vibrations est généralement limitée par rapport aux autres problèmes que nous présenterons par la suite, et il existe de plus des systèmes de stabilisation qui permettent de les éliminer presque complètement.

Nous adoptons dans cette section la notation sous forme de pseudo-vecteurs pour représenter les vitesses angulaires et la notation avec un point pour désigner les dérivées par rapport au temps : $\dot{x} = \frac{dx}{dt}$. Dans la figure 2.1, l'objet observé est au point O et la caméra au point R. On représente ici le problème dans un repère euclidien d'origine R et de vecteurs directeurs ($\vec{u}_x$, $\vec{u}_y$, $\vec{u}_z$). L'axe x correspond à l'axe optique, l'axe z à la verticale. Les coordonnées du point O dans ce repère sont (x, y, z). $\vec{V}$ est la vitesse de déplacement linéaire du robot, $\vec{\omega}$ est sa vitesse angulaire, $D = \| \overrightarrow{\mathrm{RO}} \|$ est la distance entre la caméra et l'objet et δ l'angle entre l'axe optique et la direction de déplacement du robot. Le flux optique est représenté par la vitesse angulaire $\vec{F}$, mouvement apparent de l'objet O depuis le point R.

On peut le décomposer en une composante $\vec{F_T}$ qui provient de la translation du robot et une composante $\vec{F_R}$ qui provient de sa rotation : $\vec{F} = \vec{F_T} + \vec{F_R}$. La composante de rotation est triviale

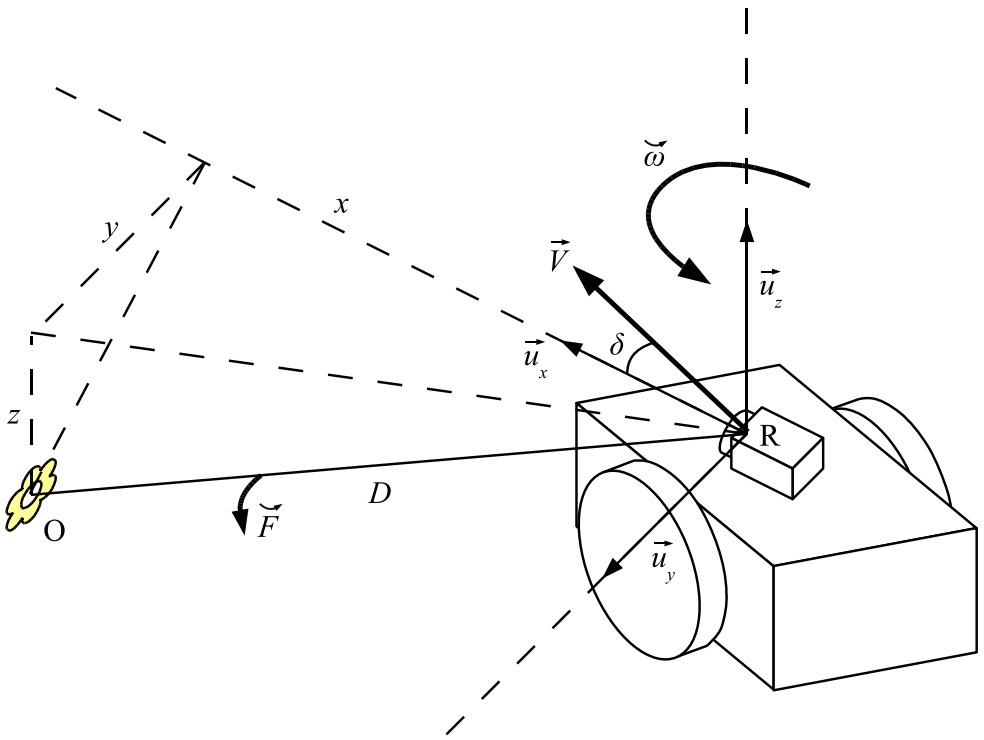


FIG. 2.1 : *Déplacement du robot générant le flux optique.*

à calculer, elle dépend uniquement des vitesses de rotation du robot et de la caméra qui ont lieu uniquement dans le plan horizontal :

$$\vec{F_R} = -(\omega + \dot{\delta})\,\vec{u_z}$$

Le mouvement apparent lié à la translation est situé dans le plan défini par les vecteurs $\vec{V}$ et $\overrightarrow{\mathrm{RO}}$. On peut l'exprimer ainsi :

$$\vec{F_T} = \frac{\vec{V} \wedge \overrightarrow{\mathrm{RO}}}{D^2}$$

Le mouvement apparent total est donc donné par la formule :

$$\begin{aligned} \vec{F} &= \vec{F_T} + \vec{F_R} \\ &= \frac{\vec{V} \wedge \overrightarrow{\mathrm{RO}}}{D^2} - (\omega + \dot{\delta})\,\vec{u_z} \end{aligned}$$

On constate que seule la partie liée au mouvement de translation dépend de la distance. Pour l'évitement d'obstacles, seule cette information de distance nous intéresse. Il sera donc nécessaire d'éliminer la composante liée à la rotation. Nous présenterons par la suite comment cela peut être effectué. Pour commencer, nous allons exprimer le mouvement de l'objet dans le repère de la caméra en fonction des variables définies précédemment. Le détail des calculs de dérivation utilisés dans les paragraphes suivants sera présenté en annexe A.

Définition du flux optique dans le repère de la caméra Le déplacement du point O dans le repère (R, $\vec{u_x}$, $\vec{u_y}$, $\vec{u_z}$) peut s'exprimer par les relations suivantes :

$$\dot{x} = -V\cos\delta + (\omega + \dot{\delta})y \quad (2.1)$$

$$\dot{y} = -V\sin\delta - (\omega + \dot{\delta})x \quad (2.2)$$

$$\dot{z} = 0 \quad (2.3)$$

$$\dot{D} = -\frac{V}{D}(x\cos\delta + y\sin\delta) \quad (2.4)$$

Il peut être utile de faire quelques remarques sur les variables utilisées ici. V et ω sont les vitesses de déplacement linéaire et angulaire du robot. Elles sont obtenues typiquement en utilisant l'odométrie du robot et seront donc limitées en précision, surtout si le robot dérape. δ est l'angle entre l'axe optique et la direction du déplacement. Cette variable dépend de plusieurs phénomènes. Tout d'abord la caméra peut être montée latéralement pour mesurer le flux optique latéral, cet angle lié au montage peut être précisément mesuré. Deuxièmement en cas de dérapage, le robot ne se dirigera pas dans la direction de l'axe des roues. L'angle ainsi créé est par contre difficile à mesurer ou à estimer avec du matériel conventionnel. Troisièmement le déplacement visuel n'est pas mesuré instantanément mais entre deux ou plusieurs images successives espacées dans l'espace et dans le temps. Cela va également créer un angle entre l'axe optique et la direction du déplacement comme indiqué sur la figure 2.2. $\dot{\delta}$ est la vitesse de variation de cet angle δ. Elle peut être causée par une rotation de la caméra si celle-ci est de type *pan-tilt*, auquel cas cette valeur est connue. Mais elle est également causée par la variation des phénomènes indiqués ci-dessus ce qui va causer une imprécision supplémentaire sur l'effet de la rotation, comme indiqué par les termes $(\omega + \dot{\delta})$ dans les équations précédentes. Toutes ces imprécisions vont bien entendu avoir un effet sur l'utilisation du flux optique et vont affecter grandement l'estimation des distances des obstacles.

La plupart des objectifs de caméra sont construits de manière à projeter l'image sur un plan. Pourtant il ne s'agit pas forcément de la projection idéale pour utiliser le flux optique. Nous allons donc comparer les propriétés géométriques de plusieurs types de projection afin de déterminer les avantages et inconvénients de chacun.

Projection sphérique Comme son nom l'indique, une projection sphérique est obtenue en projetant l'image sur une surface sphérique. C'est le cas pour l'œil humain et celui de la plupart des animaux. Cette projection est donc adaptée lorsqu'on essaie de reproduire au mieux la vision humaine. Elle peut être obtenue en effectuant une transformation d'une image plane, ce qui a bien entendu un coût computationnel, ou en utilisant un objectif spécial de type *fisheye*. La figure 2.3 représente la perception de l'image en coordonnées sphériques.

Les coordonnées utilisées sont ici α, angle entre la direction de l'axe optique et la direction de l'objet, et β qui est l'angle entre le plan formé par $(\overrightarrow{RO}, \vec{u_x})$ et le plan horizontal. Notons qu'on appelle généralement ces angles ϕ et θ en coordonnées sphériques, nous les avons renommés pour ne pas créer de confusion avec les coordonnées polaires présentées ensuite. Nous avons effectué une simplification au niveau de la représentation de l'angle α : ce que l'on observe réellement sur une projection sphérique est la longueur $\alpha_i = f\alpha$ où f est la distance focale de l'objectif. Les propriétés géométriques étant les mêmes au facteur f près, nous utilisons ici

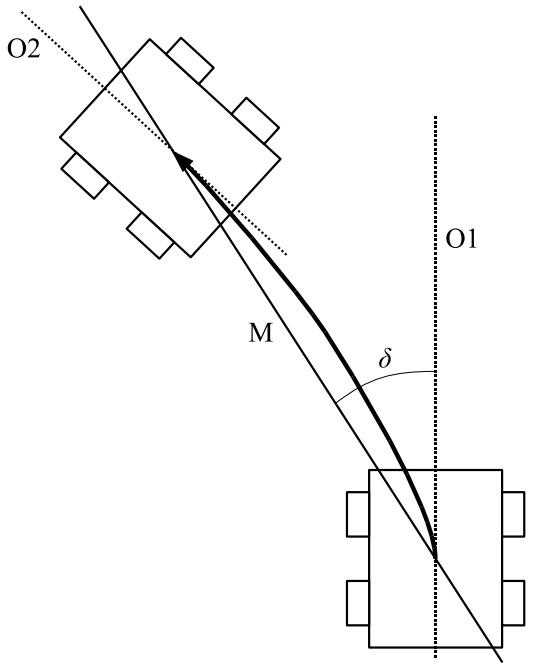


FIG. 2.2 : *Angle généré par la séparation de deux images successives dans le temps. Le flux optique est calculé dans le repère correspondant à la première position du robot (axe optique O1), mais le déplacement correspondant a eu lieu selon l'axe M.*

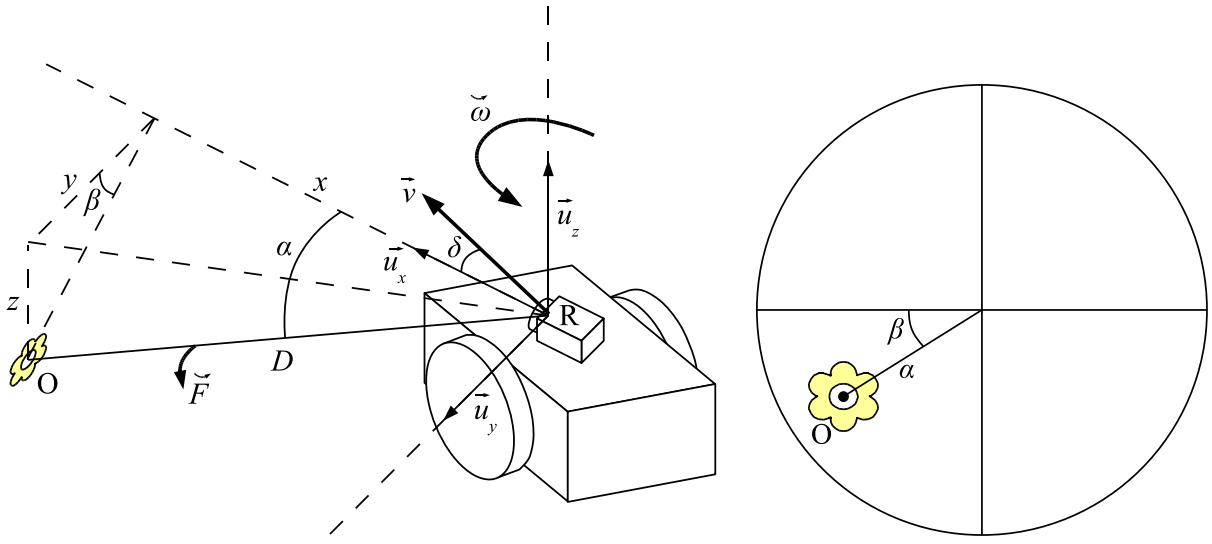


FIG. 2.3 : *Représentation de l'image perçue en coordonnées sphériques.*

directement l'angle α. Le flux optique est représenté en coordonnées sphériques par les relations suivantes :

$$\dot{\alpha} = \frac{V}{D}(\cos\delta\sin\alpha - \cos\alpha\cos\beta\sin\delta) - \cos\beta(\omega + \dot{\delta}) \quad (2.5)$$

$$\dot{\beta} = \frac{V\sin\delta\sin\beta}{D\sin\alpha} + \frac{\sin\beta}{\tan\alpha}(\omega + \dot{\delta}) \quad (2.6)$$

On peut déduire plusieurs propriétés de ces relations. Tout d'abord, sur la ligne verticale au centre de l'image, on a $\beta = \frac{\pi}{2}$ et on peut donc directement obtenir la distance des obstacles D avec la relation suivante :

$$D = \frac{V}{\dot{\alpha}}\cos\delta\sin\alpha$$

Cela signifie que la rotation n'a aucun effet sur $\dot{\alpha}$ dans cette partie de l'image. Mais la propriété la plus intéressante de la projection sphérique est que si on élimine les mouvements de rotation est que le mouvement est colinéaire à l'axe optique ($\omega = 0$, $\dot{\delta} = 0$ et $\delta = 0$), on obtient directement la distance des obstacles en tout point de l'image par la relation :

$$D = \frac{V\sin\alpha}{\dot{\alpha}}$$

De plus, dans ce cas et seulement dans ce cas, $\dot{\beta} = 0$ sur toute l'image. Cela permet de vérifier si les mouvements de rotation ont bien été éliminés.

Projection polaire Dans le cas d'une projection polaire, les coordonnées des points sont l'azimut ψ et l'élévation θ comme indiqué sur la figure 2.4. Il n'y a pas à notre connaissance d'objectif permettant d'obtenir directement une image en coordonnées polaires. Une transformation est donc nécessaire pour obtenir celle-ci. Ici aussi, nous simplifions les calculs en utilisant directement ψ et θ au lieu de $\psi_i = f\psi$ et $\theta_i = f\theta$.

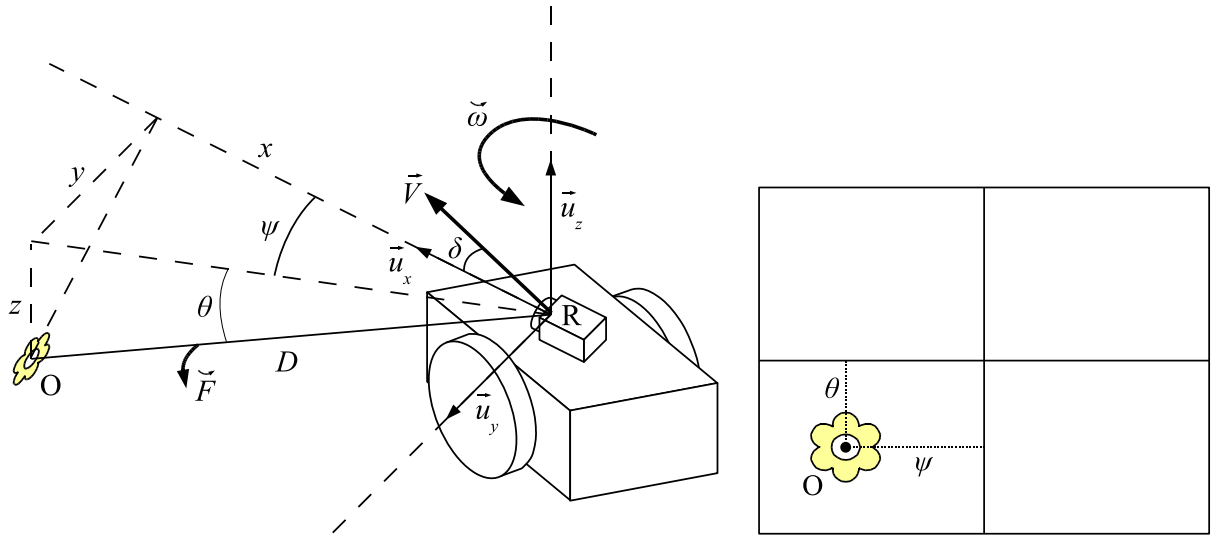


FIG. 2.4 : *Représentation de l'image perçue en coordonnées polaires.*

Le flux optique est représenté en coordonnées polaires par les relations :

$$\dot{\psi} = \frac{V\sin(\psi-\delta)}{D\cos\theta} - (\omega+\dot{\delta}) \tag{2.7}$$

$$\dot{\theta} = \frac{V\sin\theta\cos(\psi-\delta)}{D} \tag{2.8}$$

Cette représentation présente plusieurs avantages importants. Tout d'abord, $\dot{\theta}$ ne dépend pas du mouvement de rotation et permet donc de calculer les distances même lorsque le robot tourne. De plus, une rotation du robot selon l'axe vertical entraîne une translation horizontale de l'image en coordonnées polaires. De ce fait, il est très facile ici de supprimer les rotations ou de recaler l'axe de la caméra sur l'axe du mouvement lorsque les paramètres correspondants sont connus. Notons que la projection cylindrique, que nous ne détaillerons donc pas ici, présente des propriétés similaires.

Projection plane Cette projection est de loin la plus utilisée puisque c'est celle qui est produite par la plupart des caméras. Nous verrons que ce n'est pas forcément la représentation la plus pratique pour calculer la distance des obstacles mais elle présente néanmoins plusieurs propriétés intéressantes. La formation de l'image est présentée sur la figure 2.5.

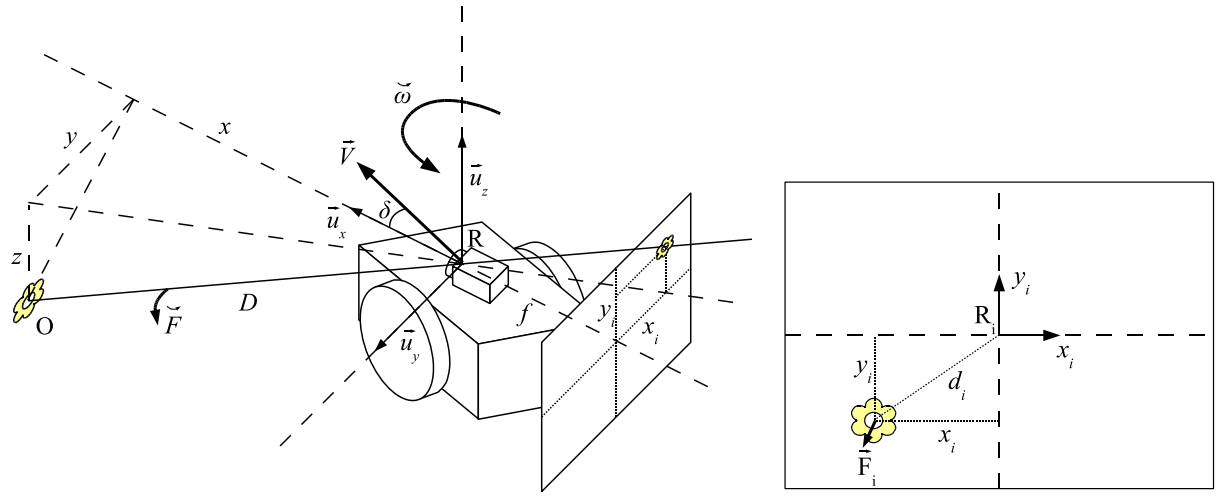


FIG. 2.5 : *Représentation du flux optique projeté sur le plan image*

f est la distance focale de l'objectif utilisé, x_i et y_i sont les coordonnées de la projection du point O dans l'image. $d_i = \sqrt{x_i^2+y_i^2}$ est la distance de ce point au centre de l'image. On considère que l'origine du repère image est au centre de celle-ci, que l'axe des x_i est orienté vers la droite et l'axe des y_i vers le haut. Le flux optique se représente alors par les relations suivantes :

$$\dot{x_i} = \frac{fV}{x}\left(\frac{x_i\cos\delta}{f}+\sin\delta\right) + f(\omega+\dot{\delta})\left(\frac{x_i^2}{f^2}+1\right) \tag{2.9}$$

$$\dot{y_i} = \frac{Vy_i\cos\delta}{x} + (\omega+\dot{\delta})\frac{x_iy_i}{f} \tag{2.10}$$

Notons tout d'abord que le flux optique est ici exprimé en fonction de x et non pas de D. Il est en effet plus facile en projection plane de calculer x qui correspond à la profondeur de l'obstacle, c'est à dire à la distance entre la caméra et le plan perpendiculaire à son axe contenant l'obstacle. Cette valeur est d'ailleurs plus utile lorsqu'on cherche à déterminer le temps avant impact d'un robot se dirigeant droit sur un mur, comme nous le détaillerons par la suite. Il est tout de même possible de retrouver D à partir de x en utilisant l'expression suivante déduite géométriquement de la figure 2.5 :

$$D = \frac{x}{f}\sqrt{f^2 + d_i^2}$$

Comme pour la projection en coordonnées sphériques, sur la ligne verticale centrale de l'image (lorsque $x_i = 0$), $\dot{y}_i$ ne dépend pas du mouvement de rotation et peut donc être utilisé pour retrouver la distance des obstacles. C'est malheureusement à peu près la seule propriété utilisable directement en projection plane. Par contre, lorsqu'on arrive à éliminer les rotations et à recaler l'axe optique sur la direction du mouvement, on obtient :

$$\dot{d}_i = \frac{V d_i}{x} \quad \text{d'où} \quad x = \frac{V d_i}{\dot{d}_i}$$

On peut donc très facilement déduire x à partir de la norme du flux optique dans ce cas. De plus, si V est inconnue, on peut tout de même déduire le temps avant impact t_I :

$$t_I = \frac{x}{V} = \frac{d_i}{\dot{d}_i}$$

Dans cette thèse, nous avons choisi d'utiliser uniquement une projection plane. En effet, si la projection sphérique est intéressante pour obtenir la distance des obstacles, elle est nettement plus difficile à produire en simulation. La projection plane est obtenue simplement par un produit matriciel, pour lequel les cartes graphiques sont de plus grandement optimisées. Comme nous l'avons vu, la projection polaire ne peut être obtenue sans une transformation de l'image qui est elle aussi coûteuse d'un point de vue computationnel. Enfin, la profondeur x calculée en projection plane est aussi facilement utilisable que la distance D pour l'évitement d'obstacles.

Élimination de la rotation et recalage de l'axe optique Comme nous l'avons indiqué auparavant, seule la composante du flux optique liée au mouvement de translation nous intéresse pour l'évitement d'obstacles. De plus, il est nécessaire de connaître la direction de cette translation pour pouvoir calculer les distances à partir du flux optique. Cela se comprend aisément en examinant les relations présentées précédemment. Dans tous les cas nous avons deux relations comportant quatre inconnues : D, V, δ et $\omega + \dot{\delta}$. Même si l'on peut se passer de V pour le calcul du temps avant impact, ce problème reste essentiellement mal posé.

La première solution pour le résoudre consiste à éliminer mécaniquement une des inconnues. La méthode la plus simple pour cela consiste à décomposer le mouvement du robot en une suite de déplacements en ligne droite et de virages sur place. Cela permet d'éliminer la composante rotationnelle durant le déplacement en ligne droite. Si de plus la caméra est montée dans l'axe des roues, on élimine à la fois δ et $\omega + \dot{\delta}$. Cette méthode très simple est de loin la plus

robuste pour utiliser efficacement le flux optique. Nous avons donc choisi de l'implémenter en incluant ce type de mouvement dans notre base de primitives de déplacement. L'inconvénient est que dans ce cas le mouvement est saccadé. En conservant un mouvement fluide, il est possible d'essayer d'éliminer δ en fixant la caméra colinéairement à l'axe des roues ou d'éliminer la rotation en tournant la caméra sur le robot de manière contrarotative (en fixant $\dot{\delta} = -\omega$). Cependant ces méthodes n'éliminent pas les effets présentés précédemment dus au dérapage et au décalage des images dans le temps. Elles seront donc peu précises.

La deuxième solution consiste à estimer δ, $\omega + \dot{\delta}$ ou les deux à l'aide d'autres capteurs pour rendre les équations précédentes solubles. Comme nous l'avons vu précédemment, il peut être intéressant dans ce cas d'utiliser une projection polaire car les corrections correspondantes se font alors très simplement. Une variante de cette méthode consiste à utiliser les valeurs issues des capteurs pour transformer les images avant le calcul du flux optique. Cela présente l'avantage de limiter le déplacement entre les images et donc de faciliter le calcul du flux optique. Nous avons implémenté une méthode de ce type permettant à la fois d'éliminer la rotation et de recaler l'axe optique sur la direction du mouvement. Elle nécessite uniquement de connaître la position et l'orientation précises du robot au moment où ont été capturées les images utilisées pour le calcul du flux optique (figure 2.6). La transformation est effectuée sur les images en projection plane et consiste uniquement en un changement de perspective. Celui-ci est effectué grâce à un produit matriciel, calcul pour lequel la plupart des processeurs récents sont optimisés. Cette méthode est donc à la fois efficace et assez légère d'un point de vue computationnel. Elle peut également être effectuée en coordonnées polaires. Dans ce cas, la correction consiste en une simple translation de l'image. Par contre, la transformation de l'image en coordonnées polaires est un traitement plus lourd d'un point de vue computationnel. Notons que dans tous les cas, la précision du résultat dépendra directement de la précision des capteurs. Nous avons ainsi obtenu de très bons résultats en simulation avec cette méthode car les données de position sont exactes. Nous l'avons testée sur un robot réel en utilisant l'odométrie avec beaucoup moins de succès. Il serait intéressant de la tester avec des capteurs plus précis comme une centrale inertielle par exemple.

La troisième solution consiste à tenter de résoudre le problème de manière analytique. En effet, δ et $\omega + \dot{\delta}$ sont identiques pour tous les points de l'image, seule la distance D varie. Il existe des méthodes mathématiques pour résoudre ce problème en posant des contraintes supplémentaires sur D, mais le choix de ces contraintes s'avère alors très délicat. Ces méthodes ont de plus un coût computationnel important ce qui les rend difficilement utilisables en temps réel. Enfin, le calcul du flux optique étant généralement effectué également en imposant certaines contraintes de régularité sur la forme du flux, il est fort probable que des effets indésirables apparaissent à cause de la combinaison de ces contraintes. Nous n'avons donc pas tenté d'implémenter ce type de solution dans le cadre de cette thèse.

Utilisation du flux optique Une fois ces corrections effectuées, il existe plusieurs manières d'exploiter le flux optique. Une possibilité est bien sûr de calculer la distance D en tout point de l'image avec les relations présentées précédemment et d'établir une loi de commande à partir de ces distances. Cette méthode nécessite toutefois une grande précision dans le calcul du flux et dans les corrections de rotation et de direction. Une méthode beaucoup plus simple et beaucoup plus robuste consiste à calculer la moyenne de la norme du flux optique sur les parties

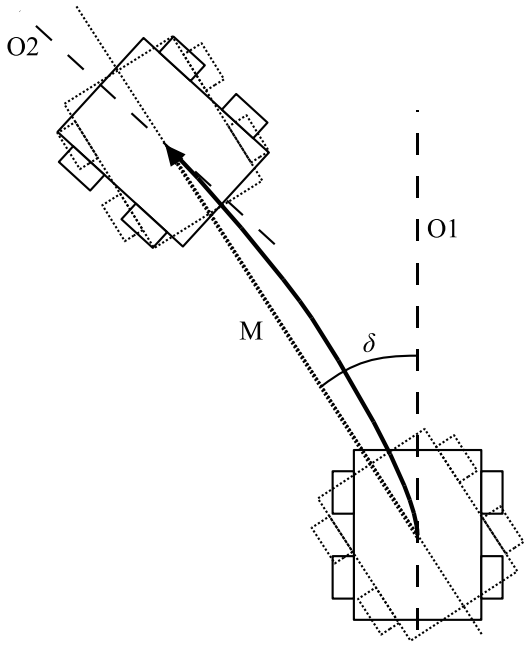


FIG. 2.6 : *Méthode utilisée pour éliminer la rotation et recaler l'axe optique sur la direction du mouvement avant de calculer le flux optique. Les images sont transformées afin de correspondre à ce qu'aurait perçu le robot si le mouvement avait été uniquement translationnel (positions et vecteur en pointillés). Cette transformation est effectuée par un changement de perspective de l'image plane.*

gauche et droite de l'image, et à déterminer la vitesse angulaire en fonction de la différence entre ces moyennes. On peut également calculer le temps avant impact dans la partie centrale de l'image pour éviter les collisions frontales. Notre système permet de créer plusieurs variantes de cette méthode. Plus précisément, les primitives implémentées dans notre système pour extraire l'information du flux optique sont :

- La projection du flux optique sur l'axe horizontal (égale à $\dot{x}_i$)
- La projection du flux optique sur l'axe vertical (égale à $\dot{y}_i$)
- La norme euclidienne du flux optique (égale à $\dot{d}_i$)
- La norme de Manhattan du flux optique (égale à $|\dot{x}_i| + |\dot{y}_i|$)
- Le temps avant impact (égal à $t_I = \frac{d_i}{\dot{d}_i}$)

Les primitives par lesquelles cette information pourra ensuite être exploitée calculent des valeurs moyennes dans des zones données définies sur l'image. Elles seront donc tout à fait adaptées pour reproduire un comportement de type équilibrage du flux entre les côtés droit et gauche de l'image. Ces primitives seront décrites en détail au §2.3.2.

Maintenant que nous avons présenté en détail la géométrie et l'utilisation du flux optique, nous allons nous intéresser aux différentes manières de calculer ce flux. Cette présentation sera toutefois beaucoup plus rapide car la littérature sur le sujet est abondante.

2.1.1.2 Calcul du flux optique

Le calcul du flux optique consiste à déterminer le mouvement apparent à partir d'un couple ou d'une séquence d'images. De nombreuses méthodes ont été proposées depuis une trentaine d'années pour effectuer ce calcul. Nous allons présenter rapidement les principales. Pour plus de détail, le lecteur peut se reporter notamment à l'article de Barron qui décrit une dizaine de méthodes de calcul de flux optique et compare leur performance sur des séquences d'images de synthèse et d'images réelles [Barron 94].

Méthode différentielle de Horn et Schunck Les méthodes différentielles utilisent les dérivées spatio-temporelles calculées sur l'image pour déterminer les vecteurs de déplacement correspondant au flux optique. Cela suppose que l'intensité des pixels reste constante au cours du temps, c'est à dire $I_t(x, y) = I_{t+1}(x + v_x, y + v_y)$ où $I_t(x, y)$ est l'intensité au temps t du pixel de coordonnées (x, y) et où (v_x, v_y) est le vecteur vitesse correspondant au flux optique. En utilisant les dérivées spatio-temporelles de l'image, cette équation peut s'écrire :

$$\frac{\partial I_t}{\partial t} + \frac{\partial I_t}{\partial x} v_x + \frac{\partial I_t}{\partial y} v_y = 0 \tag{2.11}$$

Cette équation de contrainte du gradient est la base de toutes les méthodes de calcul différentielles. Le problème vient du fait qu'elle est essentiellement mal posée, car elle compte deux inconnues (v_x et v_y) pour une seule équation linéaire. Il est donc nécessaire de poser d'autres contraintes sur la forme du flux optique afin de la résoudre. La différence entre toutes les méthodes différentielles vient du choix de ces contraintes et dans une moindre mesure de la méthode utilisée pour le calcul des dérivées spatio-temporelles.

Horn et Schunck supposent que le mouvement apparent provient du déplacement d'objets d'une certaine taille dans le champ de vision, et par conséquent que le flux optique résultant doit être lisse et régulier. Pour modéliser cela, ils utilisent une contrainte de régularité globale sur le flux optique [Horn 81]. Cette contrainte cherche à minimiser le gradient des vecteurs vitesse du flux optique. Formellement, ils cherchent à minimiser l'erreur E_G sur l'équation (2.11), ainsi que le gradient de vitesse E_V^2, soit :

$$E_G = \frac{\partial I_t}{\partial t} + \frac{\partial I_t}{\partial x} v_x + \frac{\partial I_t}{\partial y} v_y \tag{2.12}$$

$$E_V^2 = \left(\frac{\partial v_x}{\partial x}\right)^2 + \left(\frac{\partial v_x}{\partial y}\right)^2 + \left(\frac{\partial v_y}{\partial x}\right)^2 + \left(\frac{\partial v_y}{\partial y}\right)^2 \tag{2.13}$$

Cette minimisation s'effectue de manière globale sur toute l'image. L'erreur totale à minimiser s'écrit donc :

$$E_{\mathrm{HS}}^2 = \int\int (\alpha^2 E_V^2 + E_G^2)\, dx\, dy \tag{2.14}$$

Le coefficient α^2 sert à paramétrer le poids respectif des deux contraintes (contrainte de gradient et régularité du flux). La minimisation globale s'effectue grâce à une méthode itérative.

Nous avons inclus cette primitive de calcul de flux optique dans notre système. Le coefficient α^2 est alors défini automatiquement par le processus d'évolution.

Méthode différentielle de Lucas et Kanade La méthode de Lucas et Kanade se base également sur l'équation de contrainte du gradient (2.11). Ils utilisent quant à eux une contrainte locale de régularisation du flux optique [Lucas 81]. Celle-ci consiste à minimiser l'erreur sur l'équation (2.11) en considérant une fenêtre de voisinage autour du pixel pour lequel on calcule le flux. Formellement, on cherche à minimiser E_{LK} qui s'exprime :

$$E_{\mathrm{LK}} = \sum_{(x,\,y)\in W} w(x,y)^2 \left(\frac{\partial I_t}{\partial t} + \frac{\partial I_t}{\partial x} v_x + \frac{\partial I_t}{\partial y} v_y \right)^2 \tag{2.15}$$

W désigne une fenêtre de voisinage autour du pixel concerné, $w(x,y)$ est une fonction définie sur cette fenêtre et qui donne plus d'importance aux pixels du centre qu'à ceux de la périphérie. Typiquement, il s'agit d'une fonction gaussienne.

La qualité des résultats pour les méthodes différentielles dépend beaucoup de la précision avec laquelle sont déterminées les dérivées partielles. En particulier, ces méthodes sont très sensibles aux phénomènes d'aliasing spatial et surtout temporel. Ce dernier problème peut toutefois être partiellement résolu en utilisant une décomposition pyramidale de l'image (approche multi-résolutions). Nous avons inclus une primitive de calcul de flux optique de type Lucas-Kanade dans notre système. La taille de la fenêtre W est alors définie par le processus d'évolution.

Méthodes basées sur une mesure de corrélation Ce second type de méthodes permet de déterminer le flux optique en recherchant les meilleures corrélations locales entre deux images successives. Le principe est simple : il s'agit de rechercher, pour une région donnée de l'image, la région la plus similaire dans l'image suivante. Afin de limiter la complexité du calcul, cette recherche s'effectue uniquement dans le voisinage de la région initiale. De manière formelle, on cherche à minimiser la somme des carrés des différences sur l'ensemble de la région, soit :

$$E_C = \sum_{(i,\,j)\in W} w(i,j)^2 (I_t(x+i, y+j) - I_{t+1}(x+v_x+i, y+v_y+j))^2 \tag{2.16}$$

W désigne la région considérée autour du pixel (x, y), $w(i, j)$ est une fonction définie sur cette région et qui peut donner un poids plus important aux pixels du centre qu'à ceux de la périphérie. La meilleure corrélation est trouvée en calculant la valeur de E_C pour différentes valeurs de (v_x, v_y). Afin de limiter la complexité du calcul, il est nécessaire de fixer des tailles raisonnables pour la région W et le voisinage dans lequel on va rechercher (v_x, v_y). Dans notre système, nous avons fixé la taille de W à 3×3 pixels. La recherche de la meilleure corrélation s'effectue dans un carré de 7×7 pixels centré sur la position (x, y).

Les méthodes basées sur la corrélation sont plus robustes que les méthodes différentielles lorsque le bruit et l'aliasing temporel sont importants. Ici aussi, il est possible d'utiliser une approche multi-résolution lorsque le mouvement est important entre les images successives. Par contre, ces méthodes ne permettent pas de détecter précisément des mouvements subpixelliques.

Notons également que le résultat de ces méthodes est généralement un déplacement en pixels entre deux images. Pour obtenir la vitesse F_i en pixels / seconde, il convient donc de diviser ce déplacement par l'intervalle de temps entre deux images.

De nombreux travaux se sont basés sur les méthodes décrites ci-dessus pour les rendre plus robustes ou mieux adaptées à des applications particulières. D'autres méthodes de calcul existent également (méthodes basées sur la fréquence par exemple) que nous ne détaillerons pas ici. Nous avons choisi d'implémenter uniquement les trois méthodes décrites précédemment car elles permettent d'effectuer un calcul très rapide du flux optique, ce qui est un critère déterminant pour l'utilisation dans un système embarqué.

D'une manière générale, les méthodes basées sur le flux optique fonctionnent bien lorsque l'environnement visuel est fortement texturé et que les obstacles sont larges. Pour des environnements uniformes et des objets fins, l'information liée au mouvement apparent devient très difficile à extraire. On préfère dans ce cas là utiliser des méthodes basées sur l'apparence plutôt que sur le mouvement.

2.1.2 Méthodes basées sur l'apparence

2.1.2.1 Détection de sol

Ces méthodes ont pour but d'extraire des informations permettant d'éviter les obstacles à partir d'images statiques. Il s'agit généralement d'informations basées sur la texture, les couleurs ou la luminosité. Pour un robot mobile terrestre, cela peut être utilisé pour détecter le sol dans l'image par exemple. En effet, dans le cas où le sol est plat et où les obstacles sont posés sur le sol, leur distance peut directement être déduite à partir de leur hauteur dans l'image (figure 2.7). Horswill a appliqué cette méthode pour la navigation d'un robot dans un environnement intérieur [Horswill 93]. Il montre que ce type d'environnement est très structuré et que cette structure peut être utilisée pour mettre en œuvre des algorithmes de vision très simples et efficaces. Son système de vision bas niveau utilise la détection du sol par la texture et une segmentation des lignes de fuite pour déterminer l'orientation d'un couloir. Il arrive ainsi à se déplacer de manière autonome dans les couloirs du laboratoire. Cependant il échoue lorsque le sol est éclairé de manière trop inégale ou que la texture du sol change.

Les travaux de Lorigo ont amélioré cette méthode pour un fonctionnement dans des environnements différents [Lorigo 97]. Le système suppose que la partie inférieure de l'image correspond au sol. En recherchant les zones similaires dans le reste de l'image, le robot peut détecter les obstacles. Plusieurs modules de vision sont utilisés pour augmenter la robustesse du système. Le robot ainsi équipé est capable d'éviter les obstacles aussi bien en environnement intérieur qu'extérieur. Les cas où il échoue restent les situations où les ombres sont très prononcées ou lorsque la texture du sol change brutalement.

Ulrich a encore amélioré la méthode en ajoutant de la mémoire au système [Ulrich 00]. La texture du sol est ainsi identifiée comme étant celle de la zone que le robot vient de traverser. La détection est ainsi plus robuste mais les cas où le robot ne parvient pas à identifier correctement les obstacles restent les mêmes.

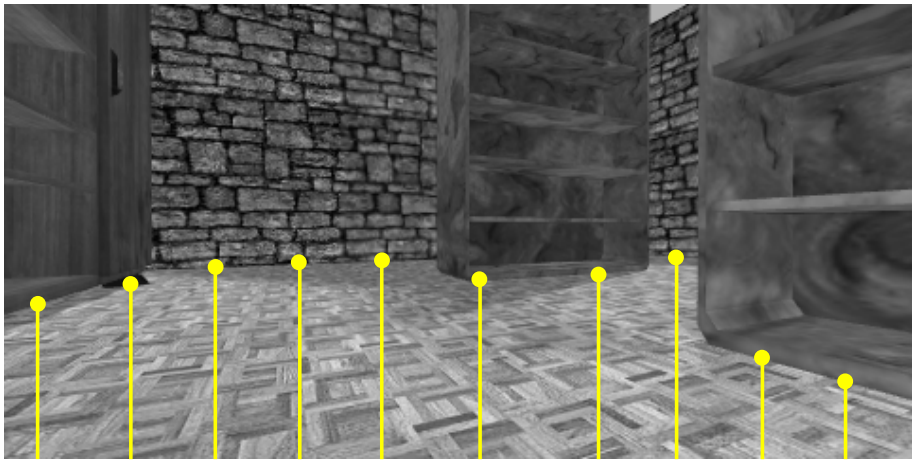


FIG. 2.7 : *Illustration de l'évitement d'obstacles par détection de sol. La distance des obstacles est liée à leur hauteur dans l'image, comme indiqué par les lignes jaunes. Cette information peut ensuite être utilisée pour éviter les obstacles les plus proches. Dans cet exemple, le robot tournerait à gauche.*

2.1.2.2 Détermination de la profondeur à partir de la texture

Une autre méthode consiste à apprendre la relation entre des indices visuels et leur distance par rapport à la caméra. L'idée sous-jacente est que les images d'environnements naturels ou artificiels ont une certaine structure : par exemple, le ciel se trouve toujours en haut de l'image et le sol en bas. En analysant cette structure on peut déterminer la profondeur des différentes parties de l'image [Michels 05, Saxena 08]. L'apprentissage est réalisé ici de manière supervisée à partir d'une base d'images pour laquelle la distance des différents objets est connue. L'acquisition de ces images se fait grâce à une caméra et un télémètre laser à balayage. Chaque image est découpée en petites zones sur lesquelles on applique des filtres de texture (filtres de Law). En utilisant une modélisation sous forme de champ de Markov, le système va apprendre la relation entre la réponse de ces filtres de texture et la distance des obstacles correspondants. Ce système a été utilisé avec succès pour le pilotage d'un robot dans un environnement naturel. Les inconvénients de ce système sont que l'acquisition de la base d'apprentissage nécessite un matériel coûteux, les résultats sont très dépendants de cette base d'apprentissage et la puissance de calcul nécessaire pour calculer tous les filtres et déterminer les distances en temps réel est élevée.

Pour pouvoir utiliser des méthodes de ce type dans notre système, nous avons inclus plusieurs filtres spatiaux permettant de mettre en évidence des informations de texture ou de bords entre différentes zones. Ces informations peuvent ensuite être utilisées par des primitives qui vont calculer des moyennes sur différentes zones de l'image. Si ces zones sont placées dans la partie inférieure de l'image, cela permet de reproduire des systèmes de type détection de sol. Ces primitives seront décrites plus en détail au §2.3.2.

2.2 Présentation de la chaîne de traitement

La section précédente avait pour but de présenter différentes techniques d'évitement d'obstacles basées sur la vision monoculaire. On peut déduire de ces techniques, et plus généralement des travaux en traitement d'images, que les algorithmes de vision se présentent toujours plus ou moins selon la même structure. Nous allons présenter cette structure dans cette section afin de justifier les choix réalisés pour notre système en termes de primitives de traitement et d'organisation de ces primitives.

2.2.1 Filtrage des images

La première étape d'un algorithme de traitement d'image consiste à appliquer un certain nombre de filtres sur celle-ci. Ces filtres peuvent faciliter les traitements suivants (utilisation de filtres gaussiens passe-bas avant un calcul de flux optique par exemple) ou directement mettre en évidence les parties de l'image qui nous intéressent pour une application donnée (filtre de texture permettant de segmenter le sol dans l'image par exemple). Il est possible également d'utiliser plusieurs filtres séquentiellement. Par exemple, l'utilisation du flux optique pour déterminer les distances des obstacles peut se voir comme l'utilisation de trois filtres successifs :

1. Filtre gaussien permettant d'atténuer le bruit dans l'image.
2. Calcul de flux optique qui va transformer l'image en un champ de vecteurs.
3. Projection de ce champ de vecteur pour obtenir une image dépendant de la profondeur. Cette projection peut s'effectuer sur un axe (horizontal ou vertical) ou par un calcul de norme.

Ce filtrage a pour but de préparer l'image à l'étape d'extraction d'information qui vient ensuite. Cependant cette étape de filtrage n'est pas toujours présente, certains algorithmes extraient l'information directement à partir de l'image originale.

2.2.2 Extraction d'informations

Cette deuxième étape a pour but d'extraire de l'image les informations nécessaires pour l'application choisie. Le but est de réduire la grande quantité de données que représente une image en un petit ensemble d'informations plus facilement exploitables. Ces informations sont généralement représentées par quelques valeurs scalaires. Les primitives utilisées pour cette étape d'extraction d'information peuvent se concentrer sur des portions réduites de l'image (extracteurs de coins ou de bords) ou sur des zones plus importantes (calcul de moyenne sur une partie de l'image).

Dans notre système, nous nous concentrons sur les primitives permettant d'extraire de l'information sur les zones plus importantes de l'image. Ce choix vient du fait que l'évitement d'obstacles se fait généralement en identifiant des zones comme les murs ou le sol plutôt que des objets particuliers. Par exemple, l'utilisation du flux optique se fait généralement en découpant l'image en plusieurs zones d'une certaine taille, voire uniquement deux zones lors d'un simple équilibrage droite-gauche. De même, la détection de sol se fait en utilisant des fenêtres d'une certaine taille dans la partie inférieure de l'image.

Notons toutefois qu'en environnement intérieur, les détecteurs de bords peuvent extraire des informations géométriques utiles pour déterminer la structure de l'environnement (séparation entre les murs et le sol, emplacement des portes). Il pourrait donc être intéressant par la suite d'adapter ce système à l'utilisation de telles primitives.

2.2.3 Utilisation des informations

Une fois les informations extraites sous forme d'un ensemble de valeurs scalaires, la dernière étape consiste à prendre une décision en fonction de ces informations. Selon l'application visée, cette décision peut être une simple valeur booléenne (l'image fait partie de la catégorie visée ou non), une commande moteur (ce sera le cas pour l'évitement d'obstacles) ou un ensemble plus complexe de traitements (mise à jour d'une carte par exemple).

Dans notre cas la commande moteur sera représentée sous la forme de deux valeurs scalaires : La vitesse linéaire et la vitesse angulaire requises. Cette dernière étape consiste donc à transformer un ensemble de valeurs issues de l'image en un couple de valeurs utilisé pour générer la commande moteur. Cette transformation s'effectue à l'aide d'opérateurs scalaires classiques (addition, soustraction, multiplication et division).

2.3 Structure et génération automatique d'algorithmes

Les techniques et les primitives visuelles présentées précédemment nous ont servi de base pour créer la structure des algorithmes utilisés dans notre système. Dans cette section, nous allons décrire les primitives et la structure de manière plus formelle, puis nous présenterons la méthode utilisée pour générer automatiquement des algorithmes de vision aléatoires à l'aide d'une grammaire.

2.3.1 Représentation des algorithmes

Les algorithmes utilisés ici sont représentés sous forme d'arbres. La racine correspond à la donnée de sortie, les feuilles sont des données d'entrée ou des constantes. Les nœuds internes correspondent aux primitives qui vont être utilisées pour transformer les données. La figure 2.8 présente un exemple d'algorithme d'évitement d'obstacles représenté sous cette forme.

2.3.2 Primitives de traitement

Les primitives de traitement sont des fonctions élémentaires permettant d'effectuer une transformation sur une ou plusieurs données. C'est l'assemblage de ces primitives qui va permettre la construction d'un algorithme. Pour chaque primitive, le type et le nombre de données d'entrée est fixé, ainsi que le type de la donnée de sortie. On peut donc trouver parmi ces primitives des filtres qui vont prendre une image en entrée pour produire une image en sortie, ou alors des opérateurs scalaires qui vont prendre deux valeurs scalaires en entrée pour produire

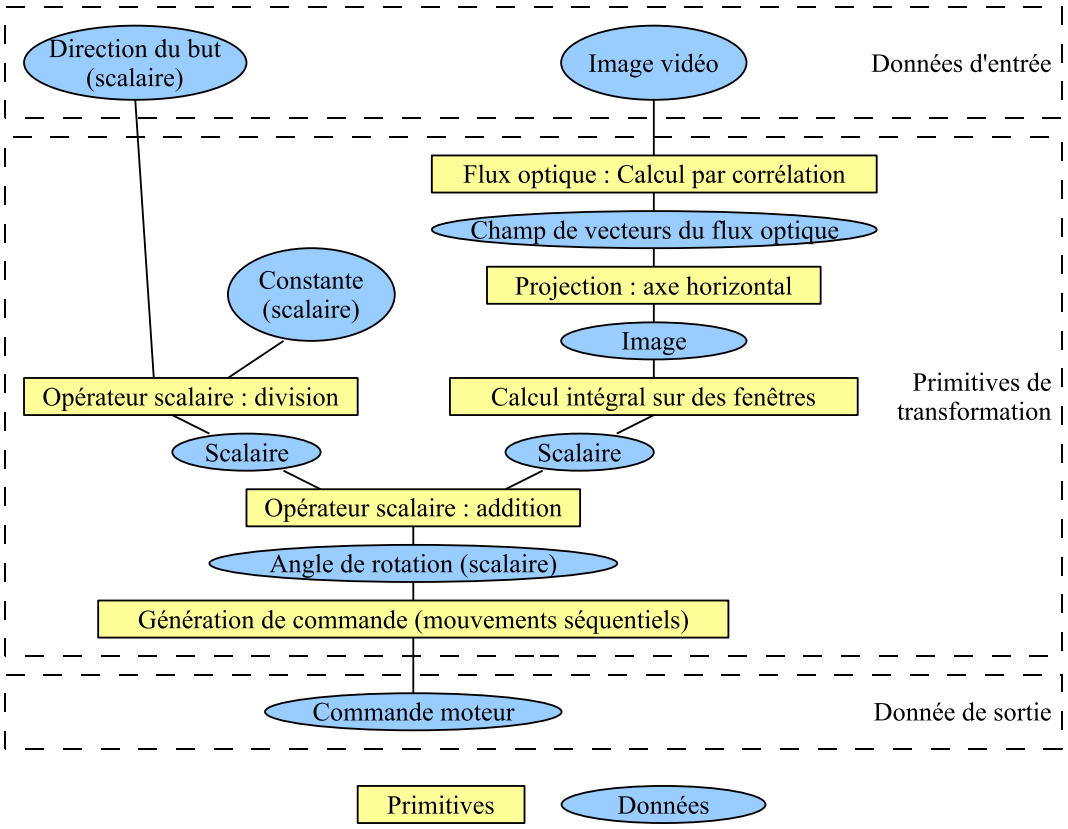


FIG. 2.8 : *Exemple d'algorithme d'évitement d'obstacles représenté sous forme d'arbre. Cet algorithme utilise le flux optique pour détecter et éviter les obstacles. La commande est le résultat d'un compromis entre évitement d'obstacles et déplacement vers une cible.*

une valeur scalaire en sortie. Nous présentons dans cette section les différentes primitives utilisées dans notre système, les types de données qu'elles manipulent ainsi que les raisons pour lesquelles nous les avons intégrées au système. La plupart de ces primitives sont paramétrables. Lorsque c'est le cas, nous préciserons quels sont ces paramètres. Ceux-ci seront fixés automatiquement par le processus de génération d'algorithmes qui sera décrit au §2.3.3. La liste de ces paramètres et leur méthode de génération seront par ailleurs détaillées en fin de chapitre dans le tableau 2.4. Pour finir, précisons que nous utilisons pour l'implémentation de ces primitives la librairie *open source* OpenCV (http://sourceforge.net/projects/opencvlibrary). Celle-ci fournit une structure de représentation d'images assez souple et implémente un certain nombre de primitives visuelles de manière très optimisée.

Filtres spatiaux (entrée : une image, sortie : une image)

Ces filtres ont pour but de mettre en évidence certains aspects de l'image, en termes de fréquence, texture ou orientation principalement. Voici la liste des filtres spatiaux utilisables dans notre système (voir aussi la figure 2.9) :

- Filtre gaussien : Il s'agit d'un filtre passe-bas, souvent utilisé pour éliminer le bruit dans l'image. Il est paramétré par l'écart-type de la fonction gaussienne utilisée comme noyau.

- Filtre laplacien : Il s'agit au contraire d'un filtre passe-haut, utilisé pour mettre en évidence les bords dans l'image. Le noyau de convolution utilisé ici est le suivant :

0	1	0
1	-4	1
0	1	0

- Filtre de seuillage : Ce filtre sert à binariser une image. Il met en évidence les parties de l'image qui dépassent un seuil de luminosité. Il peut aussi être utilisé après un filtre laplacien par exemple pour segmenter les bords de l'image. Il est paramétré par la valeur du seuil utilisé.
- Filtre de Gabor : Il s'agit d'un filtre réagissant à la fréquence et à l'orientation. Il est utile pour discriminer les textures en fonction de leur orientation. Les paramètres sont ici l'orientation du filtre, sa longueur d'onde et sa largeur de bande.
- Différence de gaussiennes : Il s'agit d'un filtre passe-bande calculé en effectuant la différence de deux filtres gaussiens. Il est utilisé pour détecter des fréquences données dans l'image. Il est paramétré par l'écart-type des deux filtres gaussiens utilisés.
- Filtre de Sobel : Il s'agit d'un filtre d'orientation, très utilisé pour détecter les lignes horizontales ou verticales dans une image. Le paramètre est ici l'orientation du filtre (verticale ou horizontale). Les noyaux de convolution utilisés sont les suivants :

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

- Sous-échantillonage : Ce filtre permet de diminuer la taille de l'image. Il élimine les hautes fréquences et permet d'accélérer les calculs suivants puisque l'image est alors réduite. Le paramètre est ici le facteur de taille utilisé.

Filtres temporels (entrée : une image, sortie : une image)

Ces filtres ont pour but de combiner plusieurs images successives afin de mettre en évidence des informations concernant le mouvement. Ces primitives sont dotées d'une mémoire interne pour stocker l'image précédente, nécessaire aux calculs. Voici la liste des filtres temporels implémentés dans notre système (voir aussi la figure 2.10) :

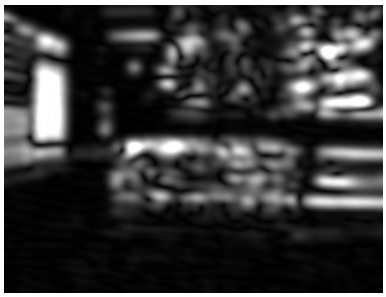
- Minimum temporel : Ce filtre prend le minimum de chaque pixel sur deux images consécutives.
- Maximum temporel : Ce filtre prend le maximum de chaque pixel sur deux images consécutives.
- Somme temporelle : Ce filtre effectue la somme de chaque pixel sur deux images consécutives. Cette somme étant ensuite normalisée, il s'agit en réalité d'une moyenne. Ce filtre permet d'atténuer le bruit dans l'image.
- Différence temporelle : Ce filtre effectue la différence de chaque pixel sur deux images consécutives. Ce filtre permet de mettre très simplement en évidence les zones en mouvement dans l'image.

(a) *Image originale*

(b) *Filtre gaussien, écart-type = 3,0*

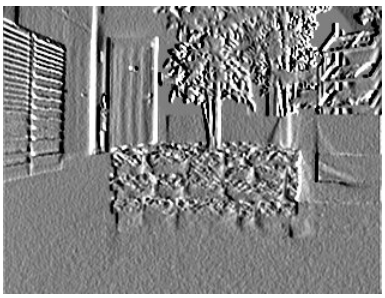
(c) *Filtre laplacien*

(d) *Filtre de seuillage, seuil = 128*



(e) *Filtre de Gabor, orientation = 100°, longueur d'onde = 8,0, largeur de bande = 1,0*

(f) *Différence de gaussiennes, écart-type 1 = 2,0, écart-type 2 = 4,0*



(g) *Filtre de Sobel, orientation verticale*

(h) *Sous-échantillonage, facteur de taille = 0,8*

FIG. 2.9 : *Résultats produits par les différents filtres spatiaux de notre système sur une image d'exemple de notre laboratoire.*

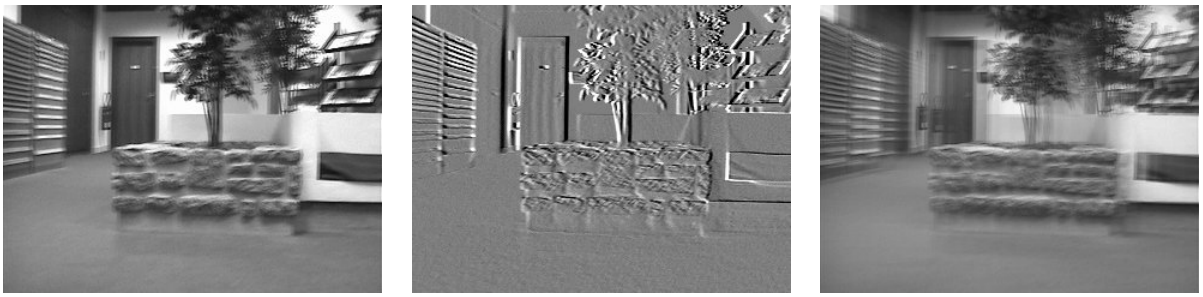
- Moyenne récursive temporelle : Ce filtre effectue une moyenne récursive sur la séquence d'images. Il permet d'atténuer le bruit et également de "lisser" temporellement les images et ainsi de diminuer les effets dus à la vibration. Cette moyenne récursive est calculée pour chaque pixel grâce à la formule :

$$M_t = \alpha I_t + (1-\alpha) M_{t-1} \tag{2.17}$$

M_t et I_t sont respectivement la moyenne récursive et l'image au temps t. Le paramètre α est un facteur multiplicatif qui va influencer le temps de convergence de cette moyenne. Plus α est faible, plus ce temps de convergence est élevé.

Calcul de flux optique (entrée : une image, sortie : un champ de vecteurs)

(a) *Image extraite de la séquence* (b) *Minimum temporel* (c) *Maximum temporel*



(d) *Somme temporelle* (e) *Différence temporelle* (f) *Moyenne récursive,* $\alpha = 0{,}3$

FIG. 2.10 : *Résultats produits par les différents filtres temporels de notre système sur une séquence d'images enregistrée dans notre laboratoire. Lors de cette séquence, le robot se déplace vers l'avant en tournant vers sa droite.*

Ces primitives permettent de calculer le mouvement apparent dans la séquence d'images et de le représenter sous la forme d'un flux optique. Elles disposent également d'une mémoire pour stocker l'image précédente. Dans ce système, tous les calculs de flux optique se font en utilisant uniquement deux images successives. Celles-ci seront transformées avant le calcul selon la méthode décrite au §2.1.1.1 pour éliminer les effets dus à la rotation du robot et au décalage entre l'axe optique et la direction du déplacement. Les différentes méthodes de calcul implémentées, déjà décrites au §2.1.1.2, sont :

- Méthode différentielle de Horn et Schunck. Le paramètre est ici le facteur α^2 qui va fixer le poids respectif des contraintes de gradient et de régularité du flux.
- Méthode différentielle de Lucas et Kanade. Le paramètre est la taille de la fenêtre utilisée pour calculer la somme des carrés des erreurs.
- Calcul par corrélation sur des régions de taille 3×3 pixels dans un voisinage de taille 7×7 pixels.

Projection du flux optique (entrée : un champ de vecteurs, sortie : une image)

Ces primitives servent à transformer le champ de vecteur du flux optique en une image donnant une indication de distance des obstacles dans les différentes directions. Les projections implémentées dans notre système sont :

- Projection sur l'axe horizontal : Cette projection a le mérite d'être très rapidement calculée et d'être précise pour les objets se trouvant sur le même plan horizontal que la caméra du robot. En effet, pour ceux-ci la composante verticale du flux optique est nulle.

- Projection sur l'axe vertical : Comme indiqué précédemment, cette projection ne donne pas d'information pour les objets situés sur le même plan horizontal que la caméra du robot. Par contre, elle présente l'avantage d'être moins sensible aux erreurs de correction du mouvement de rotation.
- Norme euclidienne.
- Norme de Manhattan.
- Temps avant impact, calculé comme indiqué au §2.1.1.1.

Ces méthodes représentent différents compromis entre la précision de l'estimation de distance, le temps de calcul et la sensibilité aux erreurs dans le flux optique. Le calcul du temps avant impact donne l'estimation de distance la plus précise mais c'est aussi le plus long à calculer et le plus sensible aux erreurs. Les projections sur les axes sont les plus rapides à calculer mais les moins précises. Les normes représentent un compromis entre les deux.

Calcul intégral sur des fenêtres (entrée : une image, sortie : une valeur scalaire)

Cette primitive sert à transformer une image en une valeur scalaire qui sera ensuite utilisée pour générer la commande moteur. Cette transformation consiste à effectuer un calcul intégral sur plusieurs fenêtres. Cela revient à calculer la moyenne de l'intensité des pixels sur les fenêtres en question. L'image est en fait séparée en deux parties selon un axe vertical central, et on considère des couples de fenêtres symétriques selon cet axe. Ce choix s'inspire de la technique d'évitement d'obstacles basée sur un équilibrage entre les côtés droite et gauche du flux optique. La technique basée sur la détection de sol peut également se voir comme un calcul de différence entre plusieurs couples de fenêtres symétriques définis dans la partie inférieure de l'image. D'une manière plus générale, on considère que cette technique permet de décrire de façon très simple la stratégie consistant à tourner à gauche s'il y a un obstacle à droite et inversement. Formellement, la méthode se décrit ainsi :

1. On définit un coefficient global α_0 pour la primitive.
2. On définit sur la partie gauche de l'image plusieurs fenêtres rectangulaires à différentes positions et de différentes tailles. On associe à chacune de ces fenêtres une autre fenêtre dans la partie droite de l'image par symétrie axiale. Un coefficient α_i et un opérateur ($+$ ou $-$) sont définis pour chaque paire.
3. La valeur scalaire produite par la primitive est une combinaison linéaire calculée grâce à la formule suivante :

$$R = \alpha_0 + \sum_{i=1}^{n} \alpha_i \mu_i \text{ avec } \mu_i = \mu_{\mathrm{L}i} + \mu_{\mathrm{R}i} \text{ ou } \mu_i = \mu_{\mathrm{L}i} - \mu_{\mathrm{R}i}$$

   Dans ces équations, n est le nombre de fenêtres, $\mu_{\mathrm{L}i}$ et $\mu_{\mathrm{R}i}$ sont les moyennes des pixels sur respectivement la fenêtre gauche et la fenêtre droite de la paire i.

Le premier paramètre de cette primitive est le coefficient α_0. Le nombre de fenêtres sera déterminé à l'aide d'une règle particulière lors de la génération des algorithmes. Pour chaque paire de fenêtres, les paramètres sont le coefficient α_i, l'opérateur à utiliser ainsi que les positions x_1, x_2, y_1 et y_2 qui vont définir la position et la taille des fenêtres.

Opérateurs scalaires (entrée : une ou plusieurs valeurs scalaires, sortie : une valeur scalaire)

Ces primitives servent à combiner plusieurs valeurs issues d'un calcul intégral entre elles ou avec des constantes. Le but est d'effectuer des compromis entre différents objectifs parfois contradictoires. Ces primitives sont :

- Addition
- Soustraction
- Multiplication
- Division
- Moyenne temporelle : Cette primitive effectue la moyenne d'une valeur scalaire sur les N dernières images. Elle permet donc simplement de lisser un résultat dans le temps. Le paramètre de cette primitive est le nombre de valeurs utilisées N.
- Test de type si A > B alors C sinon D. Cette primitive prend donc quatre valeurs en entrée pour produire une valeur scalaire en sortie.

Primitives de génération de commande (entrée : une ou deux valeurs scalaires, sortie : une commande moteur)

Ces primitives permettent de générer la commande moteur qui sera envoyée au robot à partir de valeurs scalaires calculées lors des étapes précédentes. Cette génération peut se faire de deux manières différentes :

- Commande directe : La commande est créée ici à partir de deux valeurs scalaires. La première correspond à la vitesse linéaire requise, la seconde à la vitesse angulaire requise.
- Commande séquentielle : Le déplacement du robot est ici découpé en mouvements rectilignes et en virages sur place. Cette primitive prend une seule valeur en entrée, qui correspond à l'angle de rotation du robot à la fin du déplacement rectiligne. La commande est donc utilisée ici de manière épisodique, après chaque déplacement rectiligne uniquement. Ce type de mouvement facilite l'utilisation du flux optique puisque lors du déplacement rectiligne la composante liée au mouvement de rotation est nulle. Les paramètres sont ici la vitesse de déplacement lors du déplacement rectiligne, le temps de déplacement rectiligne entre chaque virage, la tolérance sur l'angle de rotation et un coefficient qui va déterminer la vitesse de rotation en fonction de l'angle de rotation requis.

2.3.3 Génération d'algorithmes aléatoires à l'aide d'une grammaire

Après avoir présenté la structure des algorithmes de vision et les primitives que nous avons choisies pour les composer, nous allons maintenant décrire le processus de génération des algorithmes. Les méthodes que nous allons décrire ici sont issues de travaux en programmation génétique. Les aspects liés à l'évolution, la sélection et la transformation d'algorithmes seront présentés au chapitre 3. Nous nous attachons ici uniquement à détailler la génération des algorithmes.

2.3.3.1 Présentation du processus de dérivation

Le principal problème qui se pose lorsqu'on souhaite assembler un algorithme à partir de primitives manipulant différents types de données est de garantir que ce typage soit valide sur

l'ensemble de l'algorithme. Cela signifie simplement qu'il faut que le type de donnée attendu par chaque primitive soit bien celui renvoyé par la primitive précédente dans l'arbre algorithmique.

Montana a développé pour cela un système appelé programmation génétique fortement typée [Montana 95]. Il est nécessaire avant tout de définir un type pour chaque primitive et chaque donnée qui peuvent être utilisées dans les algorithmes. Prenons comme exemple un système manipulant trois types de données : des valeurs scalaires, des vecteurs à trois dimensions et des matrices 3×3. Le tableau 2.1 présente les différentes fonctions et données définies pour le système.

Fonction	Types d'entrée	Type de sortie
PRODUIT-SCALAIRE	Vecteur, Vecteur	Scalaire
PRODUIT-VECTORIEL	Vecteur, Vecteur	Vecteur
PRODUIT-SCAL-VECT	Scalaire, Vecteur	Vecteur
PRODUIT-MATRICIEL	Matrice, Matrice	Matrice
PRODUIT-MAT-VECT	Matrice, Vecteur	Vecteur
DÉTERMINANT	Matrice	Scalaire

Donnée	Type
x1	Scalaire
x2	Scalaire
v1	Vecteur
v2	Vecteur
m1	Matrice
m2	Matrice

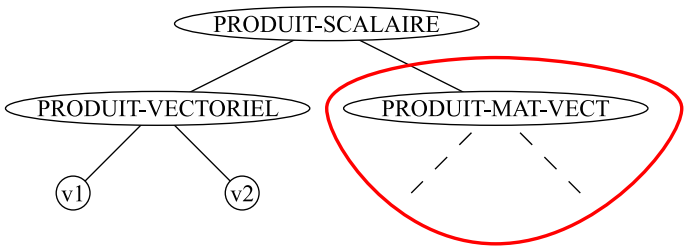
TAB. 2.1 : *Exemples de fonctions et de données définies pour un système de programmation génétique fortement typé.*

Pour générer un algorithme à partir de cette liste de fonctions et de données, on commence par définir le type attendu en sortie de l'algorithme (à la racine de l'arbre). On va ensuite sélectionner aléatoirement une fonction qui renvoie ce type ou une donnée de ce type (phase d'instanciation d'un nœud de l'arbre). S'il s'agit d'une fonction qui a été sélectionnée, ce processus est répété récursivement pour chaque entrée de la fonction. La génération continue ainsi jusqu'à ce que toutes les feuilles de l'arbre algorithmique soient des données. La figure 2.11 montre un exemple d'instanciation valide et un autre invalide.

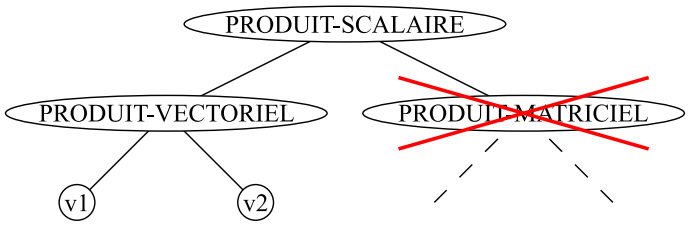
Montana a également apporté un certain nombre d'extensions à ce système, permettant notamment de manipuler des types de données différents sans redéfinir toutes les fonctions (par exemple pour utiliser des vecteurs à deux ou trois dimensions) ou de s'assurer au fur et à mesure de la génération d'un arbre que celui-ci ne dépassera pas une profondeur limite fixée à l'avance. Nous ne décrirons pas ces extensions ici car celles-ci n'ont pas d'utilité directe dans le cadre de cette thèse.

Whigham a ensuite développé un système dont le principe est très proche mais qui utilise une formalisation différente, basée sur une grammaire [Whigham 95]. Ce système est appelé programmation génétique basée sur une grammaire non contextuelle (en anglais, *context-free grammar genetic programming* ou CFG-GP). Le typage des données et des fonctions est décrit ici par un ensemble de règles, ou productions, qui vont être utilisées pour la génération des algorithmes. La grammaire correspondant à l'exemple précédent utilisant des valeurs scalaires, des vecteurs et des matrices est représentée dans le tableau 2.2.

Par convention, on représente les nœuds terminaux en minuscules et les non-terminaux en majuscules. On distingue donc une différence de représentation avec la programmation génétique fortement typée puisqu'ici les données et les fonctions sont toutes des nœuds terminaux.



(a) *Instanciation valide : La fonction PRODUIT-SCALAIRE attend un vecteur en entrée et la fonction PRODUIT-MAT-VECT renvoie un vecteur.*



(b) *Instanciation invalide : La fonction PRODUIT-SCALAIRE attend un vecteur en entrée et la fonction PRODUIT-MATRICIEL renvoie une matrice.*

FIG. 2.11 : *Exemples d'instanciation pour la génération d'algorithmes en programmation génétique fortement typée.*

(1)	DÉPART	→	SCALAIRE
(2)	SCALAIRE	→	produit-scalaire (VECTEUR, VECTEUR)
(3)	SCALAIRE	→	déterminant (MATRICE)
(4)	SCALAIRE	→	x1
(5)	SCALAIRE	→	x2
(6)	VECTEUR	→	produit-vectoriel (VECTEUR, VECTEUR)
(7)	VECTEUR	→	produit-scal-vect (SCALAIRE, VECTEUR)
(8)	VECTEUR	→	produit-mat-vect (MATRICE, VECTEUR)
(9)	VECTEUR	→	v1
(10)	VECTEUR	→	v2
(11)	MATRICE	→	produit-matriciel (MATRICE, MATRICE)
(12)	MATRICE	→	m1
(13)	MATRICE	→	m2

TAB. 2.2 : *Exemple de grammaire permettant de définir des types de données en programmation génétique basée sur une grammaire non contextuelle.*

Les nœuds non-terminaux correspondent quant à eux aux types de données. Dans une grammaire non contextuelle, la partie gauche de chaque règle est constituée d'un unique nœud non-terminal. De ce fait, l'application d'une règle ne dépend pas du contexte, c'est-à-dire de la position du nœud dans l'arbre ou des nœuds voisins.

Le processus de génération d'un algorithme est ensuite exactement le même que celui utilisé pour créer des phrases aléatoires mais syntaxiquement correctes à partir d'une grammaire de langue naturelle. Les règles de dérivation de la grammaire, appliquées récursivement, vont créer l'arbre correspondant à l'algorithme. Nous allons maintenant illustrer ce fonctionnement en utilisant la grammaire présentée précédemment.

Le symbole "DÉPART" constitue la racine de l'algorithme, c'est par lui que le processus de dérivation va commencer. Au début du processus, l'algorithme est constitué uniquement par ce nœud "DÉPART". On va lui appliquer la seule règle qui lui correspond, en l'occurrence la règle (1), comme indiqué sur la figure 2.12(a). A chaque étape du processus, on va sélectionner un nœud non terminal se trouvant en position de feuille dans l'arbre et lui appliquer une règle de dérivation qui lui correspond. S'il y a plusieurs règles, l'une d'elle est choisie aléatoirement. Dans notre exemple on applique la règle (2) au nœud "SCALAIRE" (figure 2.12(b)). On applique ensuite la règle (9) au premier nœud "VECTEUR" ainsi généré (figure 2.12(c)), la règle (8) au second nœud "VECTEUR" (figure 2.12(d)) et ainsi de suite. Le processus s'arrête lorsqu'il n'y a plus que des nœuds terminaux aux feuilles de l'arbre (figure 2.12(e)).

L'arbre de dérivation ainsi créé peut ensuite être simplifié pour être représenté sous forme d'arbre algorithmique (figure 2.13). Il peut également être représenté comme une fonction : $f(m_1, \vec{v_1}, \vec{v_2}) = \vec{v_1} \cdot (m_1 \vec{v_2})$.

Notons que si ce type de système est principalement utilisé pour garantir le typage des données, il peut aussi être vu comme une manière plus générale de restreindre la forme des algorithmes. On pourrait par exemple définir les nœuds non-terminaux DISTANCE et MASSE, ainsi que des fonctions utilisant les notions de distance et de masse. Dans ce cas, on manipule uniquement des valeurs scalaires mais la grammaire est utilisée pour éviter d'utiliser indifféremment des valeurs correspondant à des notions totalement différentes. Dans des travaux ultérieurs, Whigham décrit comment on peut guider plus précisément le processus d'évolution en introduisant des biais dans les différents opérateurs. Il est ainsi possible de fixer des probabilités pour chaque règle de la grammaire afin de guider l'évolution [Whigham 96]. D'autres travaux ont poussé plus loin l'utilisation de grammaires pour permettre notamment de définir des attributs pour chaque fonction ou donnée [Wong 97, Ross 01]. Nous ne détaillerons pas ces travaux ici car ils sortent du contexte de cette thèse. En effet, la programmation génétique utilisant une grammaire non contextuelle est largement suffisante pour notre système de génération d'algorithmes de vision.

2.3.3.2 Grammaire utilisée pour les algorithmes de vision

La grammaire que nous utilisons pour générer des algorithmes de vision est du même type que celle qui a été présentée précédemment, à la différence près que les données manipulées peuvent aussi être des images ou des champs de vecteurs. Le tableau 2.3 présente cette grammaire dans son intégralité. Pour certaines expériences, nous avons légèrement modifié cette grammaire. Ces modifications seront présentées avec les expériences concernées au chapitre 4.

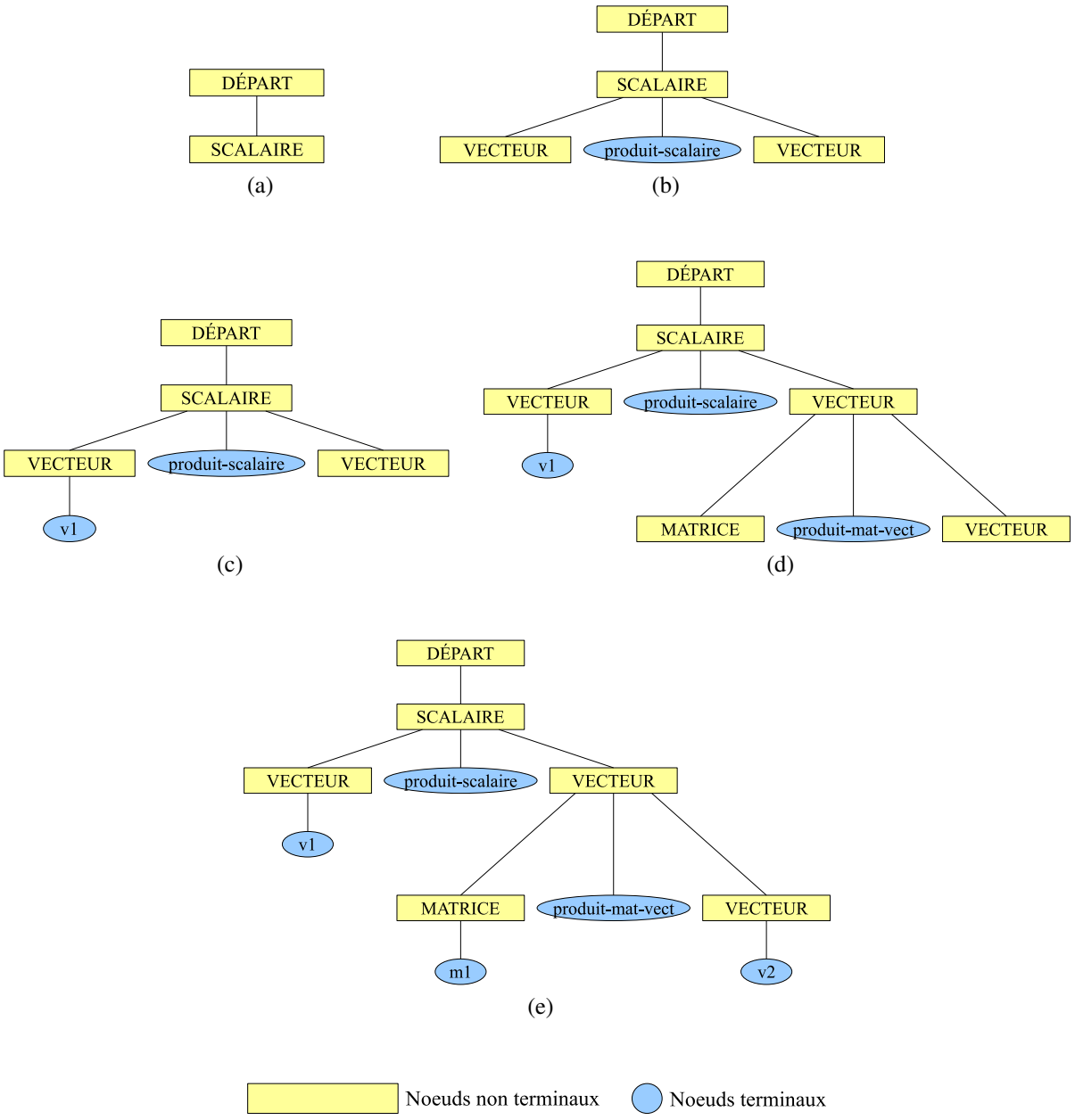


FIG. 2.12 : *Processus de dérivation permettant de générer une fonction aléatoire à partir d'une grammaire.*

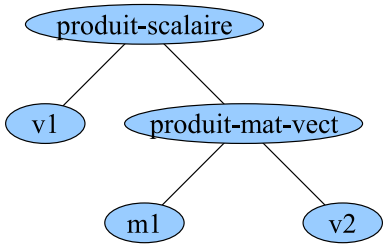


FIG. 2.13 : *Représentation sous forme simplifiée de l'algorithme produit par la grammaire.*

[1,0]	DÉPART	→	COMMANDE
[0,5]	COMMANDE	→	génération-directe (RÉEL, RÉEL)
[0,5]	COMMANDE	→	commande-séquentielle (RÉEL)
[0,1]	RÉEL	→	distance-but
[0,1]	RÉEL	→	direction-but
[0,1]	RÉEL	→	constante
[0,05]	RÉEL	→	addition (RÉEL, RÉEL)
[0,05]	RÉEL	→	soustraction (RÉEL, RÉEL)
[0,05]	RÉEL	→	multiplication (RÉEL, RÉEL)
[0,05]	RÉEL	→	division (RÉEL, RÉEL)
[0,05]	RÉEL	→	régularisation-temporelle (RÉEL)
[0,05]	RÉEL	→	si-alors-sinon (RÉEL, RÉEL, RÉEL, RÉEL)
[0,4]	RÉEL	→	calcul-intégral-fenêtres (LISTE-FENÊTRES, IMAGE)
[0,3]	LISTE-FENÊTRES	→	fenêtre
[0,7]	LISTE-FENÊTRES	→	LISTE-FENÊTRES, fenêtre
[0,3]	IMAGE	→	image-vidéo
[0,4]	IMAGE	→	FILTRE-SPATIAL (IMAGE)
[0,15]	IMAGE	→	PROJECTION (FLUX-OPTIQUE)
[0,15]	IMAGE	→	FILTRE-TEMPOREL (IMAGE)
[0,15]	FILTRE-SPATIAL	→	gaussien
[0,14]	FILTRE-SPATIAL	→	laplacien
[0,14]	FILTRE-SPATIAL	→	seuillage
[0,14]	FILTRE-SPATIAL	→	gabor
[0,14]	FILTRE-SPATIAL	→	différence-de-gaussiennes
[0,14]	FILTRE-SPATIAL	→	sobel
[0,15]	FILTRE-SPATIAL	→	sous-échantillonage
[0,2]	FILTRE-TEMPOREL	→	minimum-temporel
[0,2]	FILTRE-TEMPOREL	→	maximum-temporel
[0,2]	FILTRE-TEMPOREL	→	somme-temporelle
[0,2]	FILTRE-TEMPOREL	→	différence-temporelle
[0,2]	FILTRE-TEMPOREL	→	moyenne-récursive
[0,33]	FLUX-OPTIQUE	→	horn-schunck (IMAGE)
[0,33]	FLUX-OPTIQUE	→	lucas-kanade (IMAGE)
[0,34]	FLUX-OPTIQUE	→	corrélation (IMAGE)
[0,2]	PROJECTION	→	projection-horizontale
[0,2]	PROJECTION	→	projection-verticale
[0,2]	PROJECTION	→	norme-euclidienne
[0,2]	PROJECTION	→	norme-de-manhattan
[0,2]	PROJECTION	→	temps-avant-impact

TAB. 2.3 : *Grammaire utilisée dans notre système pour la génération des algorithmes de vision.*

Les chiffres indiqués entre crochets correspondent à la probabilité de sélection de la règle lors de la génération de l'algorithme. Cette probabilité permet de contrôler plus finement le processus de génération. On peut ainsi favoriser l'utilisation de certaines primitives en leur affectant une probabilité élevée. On peut également contrôler la croissance de l'arbre en affectant des probabilités faibles aux règles susceptibles de générer une croissance exponentielle (règle RÉEL → si-alors-sinon(RÉEL, RÉEL, RÉEL, RÉEL) par exemple).

On remarquera que ce système ne force absolument pas l'utilisation des images lors de la génération de l'algorithme. Il est tout à fait possible de générer un algorithme construisant une commande en utilisant simplement deux constantes réelles. Un tel algorithme serait tout à fait valide, même s'il présente peu d'intérêt par la suite.

2.3.3.3 Paramétrage des primitives

La plupart des primitives utilisées dans notre système peuvent être paramétrées. Par exemple, un filtre gaussien est paramétré par l'écart-type de la fonction gaussienne. Ces paramètres sont générés aléatoirement par notre système lors de la génération de l'algorithme. Pour chaque paramètre, nous définissons la distribution à utiliser pour la génération (uniforme ou normale) ainsi que les bornes du paramètre. Pour la distribution normale, nous définissons également la moyenne et l'écart-type à utiliser. Le tableau 2.4 présente ces différents paramètres et la manière utilisée pour les générer.

2.4 Bilan

Nous avons étudié dans ce chapitre différentes méthodes d'évitement d'obstacles utilisant la vision. Cette étude nous a permis de définir une base de primitives qui peuvent être utilisées pour l'évitement d'obstacles. Nous avons formalisé ces primitives, les types de données qu'elles manipulent et la structure des algorithmes de vision sous la forme d'une grammaire. Nous avons enfin présenté comment cette grammaire peut être utilisée pour synthétiser de nouveaux algorithmes.

Nous allons à présent nous intéresser au cœur du processus d'évolution, c'est à dire aux méthodes utilisées pour évaluer, sélectionner, recombiner et transformer ces algorithmes. Ces méthodes étant directement issues des recherches en évolution artificielle, nous commencerons par un rapide état de l'art sur les algorithmes évolutionnaires.

Primitive et paramètre	**Distribution**	**Min.**	**Max.**	**Moyenne**	**Ecart-type**
Commande séquentielle					
Vitesse de déplacement (cm/s)	Normale	-100	100	40	20
Temps de déplacement (s)	Normale	0,1	20	3	2
Tolérance de rotation (°)	Normale	0,0001	20	3	2
Coefficient de rotation	Normale	0,0001	20	1,5	0,5
Constante					
Valeur	Normale	-	-	0,0	50,0
Régularisation temporelle					
Nombre de valeurs	Normale	1	100	1	3
Calcul intégral sur des fenêtres					
Coefficient global α_0	Uniforme	-180,0	180,0	-	-
Fenêtre					
Position x_1	Uniforme	0,0	0,5	-	-
Position x_2	Uniforme	0,0	0,5	-	-
Position y_1	Uniforme	0,0	1,0	-	-
Position y_2	Uniforme	0,0	1,0	-	-
Coefficients α_i	Normale	-	-	0,0	3,0
Opérateur (+ ou -)	Uniforme	-	-	-	-
Filtre gaussien					
Écart-type	Normale	0,0001	20,0	3,0	2,0
Filtre de seuillage					
Seuil	Uniforme	0	255	-	-
Filtre de Gabor					
Orientation	Uniforme	0,0	180,0	-	-
Longueur d'onde	Normale	2,1	20,0	5,0	2,0
Largeur de bande	Normale	0,5	2,0	1,0	0,3
Différence de gaussiennes					
Écart-type 1	Normale	0,0001	20,0	3,0	2,0
Écart-type 2	Normale	0,0001	20,0	3,0	2,0
Filtre de Sobel					
Orientation (H ou V)	Uniforme	-	-	-	-
Sous-échantillonage					
Facteur de taille	Uniforme	0,01	1,0	-	-
Moyenne récursive					
Facteur multiplicatif α	Uniforme	0,01	1,0	-	-
Calcul de flux optique Horn-Schunck					
Facteur de pondération α^2	Normale	0,0	100,0	2,0	1,0
Calcul de flux optique Lucas-Kanade					
Taille de la fenêtre	Uniforme	1	15	-	-

TAB. 2.4 : *Paramètres des primitives utilisées dans notre système*

Chapitre 3

Évaluation et optimisation des algorithmes

Après avoir décrit la structure des algorithmes de vision, nous allons maintenant présenter les méthodes mises en œuvre pour les évaluer et les optimiser. Nous rappelons que le but est de concevoir automatiquement des algorithmes qui soient efficaces pour une application donnée, ici l'évitement d'obstacles. Il est donc nécessaire de développer des méthodes pour évaluer les performances des algorithmes dans le cadre de cette application. Dans ce chapitre, nous allons tout d'abord réaliser un rapide état de l'art sur les algorithmes évolutionnaires, et plus particulièrement sur la programmation génétique qui est une méthode évolutionnaire pour optimiser des algorithmes. Nous présenterons ensuite les environnements et les techniques utilisés pour évaluer nos algorithmes d'évitement d'obstacles. Enfin, nous présenterons plusieurs méthodes mises en œuvre pour améliorer les performances de ces algorithmes.

3.1 Optimisation par programmation génétique

3.1.1 Introduction aux algorithmes évolutionnaires

Les algorithmes évolutionnaires sont des méthodes d'optimisation inspirées par la théorie darwinienne d'évolution des espèces. Cette théorie se base sur le fait qu'un individu adapté à son environnement a plus de chances de survivre et de se reproduire qu'un individu non adapté. Lors de la reproduction, les gènes qui rendent cet individu adapté vont se transmettre à sa descendance et ainsi se propager dans la population. De nouveaux gènes peuvent également apparaître par mutation. Si ceux-ci donnent un avantage à l'individu, il y a plus de chances pour que celui-ci survive, se reproduise et les transmette à ses enfants. Dans le cas contraire, il est plus probable que l'individu meure sans transmettre ces gènes. Ce phénomène de sélection naturelle appliqué sur un grand nombre de générations a conduit les différentes espèces vivantes à évoluer pour s'adapter à leur environnement.

L'évolution artificielle se base sur ces principes de sélection naturelle et de reproduction par croisement et mutation pour produire des solutions efficaces à un problème donné. Ici, on appelle individu une solution possible pour le problème que l'on cherche à résoudre. Une population est un ensemble d'individus à un moment donné. La qualité d'une solution est mesurée par une fonction d'évaluation aussi appelée fonction de fitness. Le déroulement global d'un algorithme évolutionnaire est le suivant (figure 3.1) :

1. **Initialisation** : Le processus débute avec un ensemble de solutions possibles au problème donné. En général, cette population initiale est générée aléatoirement.
2. **Évaluation** : Chacun des individus va ensuite être évalué individuellement. Le résultat de cette évaluation est un score, aussi appelé fitness, indiquant la qualité de la solution pour le problème donnée. Cette fonction d'évaluation est donc très dépendante du problème à résoudre.
3. **Sélection** : Une partie de la population va être sélectionnée pour pouvoir se reproduire. Cette sélection dépend du score de fitness obtenu auparavant : plus ce score est élevé, plus l'individu a de chances d'être retenu.
4. **Reproduction** : Les individus sélectionnés précédemment sont utilisés pour créer de nouvelles solutions. En général cette reproduction s'effectue en deux étapes. Premièrement de nouveaux individus sont créés par combinaison de deux individus sélectionnés (croisement). Ensuite des changements aléatoires sont effectués sur ces nouveaux individus (mutation).
5. **Nouvelle génération** : Les individus ainsi créés constituent la nouvelle population. Celle-ci va servir de base à un nouveau cycle (retour à l'étape 2). Ce processus se termine après un certain nombre de générations ou lorsque les solutions ont atteint un score suffisant.

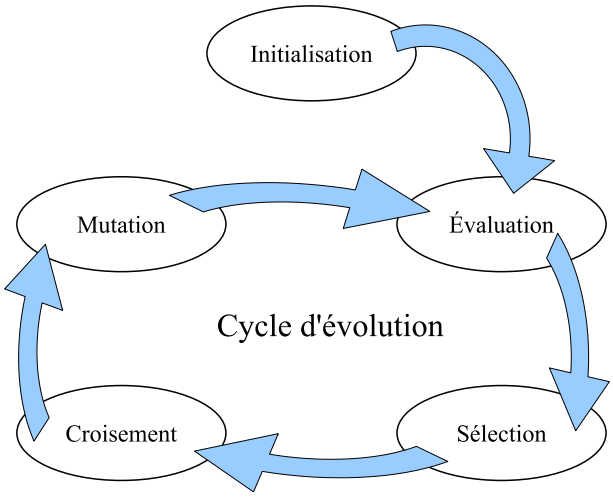
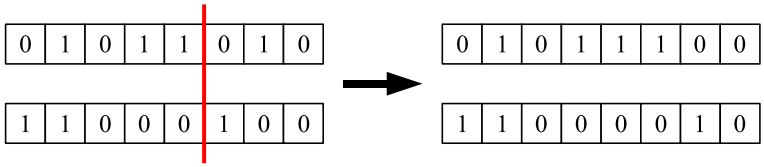


FIG. 3.1 : *Illustration du processus cyclique mis en œuvre par les algorithmes évolutionnaires.*

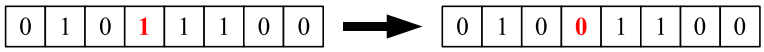
Historiquement, les premiers algorithmes évolutionnaires ont été développés dans les années 60 et 70. A cette époque, plusieurs travaux ont exploité les mêmes principes liés à la sélection et la reproduction pour des applications différentes en utilisant des représentations très différentes. Cela a mené à une segmentation des algorithmes évolutionnaires en trois grandes catégories : les algorithmes génétiques, les stratégies d'évolution et la programmation évolutionnaire. Même si de nombreux rapprochements ont eu lieu, cette séparation n'a pas totalement disparu aujourd'hui. Il reste donc utile de décrire ces grandes classes d'algorithmes. On peut noter que la programmation génétique, apparue au début des années 90, constitue une quatrième classe. Celle-ci sera décrite plus en détail au §3.1.2. Notons qu'il n'y a pas de différence fondamentale entre ces différentes classes d'algorithmes. Les principes généraux de cycle d'évolution, de sélection basée sur une fonction d'évaluation et de transformation des individus restent les

mêmes. Les différences viennent principalement de la représentation des individus et du choix des opérateurs de transformation.

Algorithmes génétiques Les plus connus des algorithmes évolutionnaires sont sans doute les algorithmes génétiques développés par John Holland au début des années 70 [Holland 75]. Comme leur nom l'indique, ces algorithmes sont directement inspirés par la génétique. Les individus sont représentés par une séquence binaire appelée génome. L'opérateur de croisement consiste à définir aléatoirement un point de croisement dans ce génome, puis à échanger entre les deux individus la partie du génome qui suit ce point de croisement. La mutation consiste simplement à modifier un bit du génome (figure 3.2).



(a) *Croisement*



(b) *Mutation*

FIG. 3.2 : *Illustration des opérateurs de transformation utilisés avec les algorithmes génétiques.*

Stratégies d'évolution Les stratégies d'évolution ont initialement été développées par Rechenberg au début des années 70 [Rechenberg 73]. Ce type d'algorithmes est principalement utilisé pour l'optimisation de paramètres réels dépendant les uns des autres. Les individus sont représentés ici par un vecteur de nombres réels. Le croisement consiste généralement à prendre pour chaque élément du vecteur la valeur d'un des deux parents aléatoirement. La mutation consiste à modifier légèrement chacun des éléments en y ajoutant un nombre aléatoire généralement issu d'une distribution normale.

Programmation évolutionnaire La programmation évolutionnaire est issue des travaux de Fogel au début des années 60 [Fogel 66]. Elle utilise des automates à états finis pour représenter les solutions. La transformation est effectuée par une mutation qui consiste à remplacer aléatoirement un élément de l'automate par un nouveau.

Hormis ces différentes classes d'algorithmes évolutionnaires, il existe une variante plus fondamentale sur la manière de représenter les solutions. Il s'agit de l'approche dite "parisienne", dans laquelle un individu ne représente qu'une partie de la solution. Il est donc nécessaire de prendre en compte un groupe d'individus, voire toute la population, pour obtenir une représentation complète de la solution recherchée [Collet 99]. Nous présenterons un exemple d'application de cette approche au §3.1.3.2.

3.1.2 Programmation génétique

La programmation génétique constitue une quatrième classe d'algorithmes évolutionnaires. Elle est issue des travaux de John Koza au début des années 90 [Koza 92] et consiste à développer des programmes ou algorithmes en utilisant des techniques évolutionnaires. Notons qu'il existe plusieurs représentations possibles pour des programmes. Nous utilisons ici une représentation arborescente qui est la plus couramment employée en programmation génétique. Certains travaux se basent sur une représentation linéaire ou sous forme de graphes [Banzhaf 98] mais nous ne détaillerons pas ces techniques ici.

Les algorithmes sont représentés ici sous forme d'arbres. Les nœuds internes de l'arbre sont des fonctions, les nœuds externes correspondent aux données manipulées par l'algorithme. Le calcul s'effectue en partant des feuilles et en évaluant les différentes expressions en remontant vers la racine. Le résultat final de l'algorithme est calculé au niveau du nœud racine (figure 3.3).

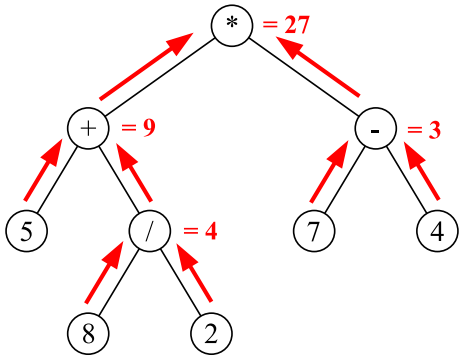


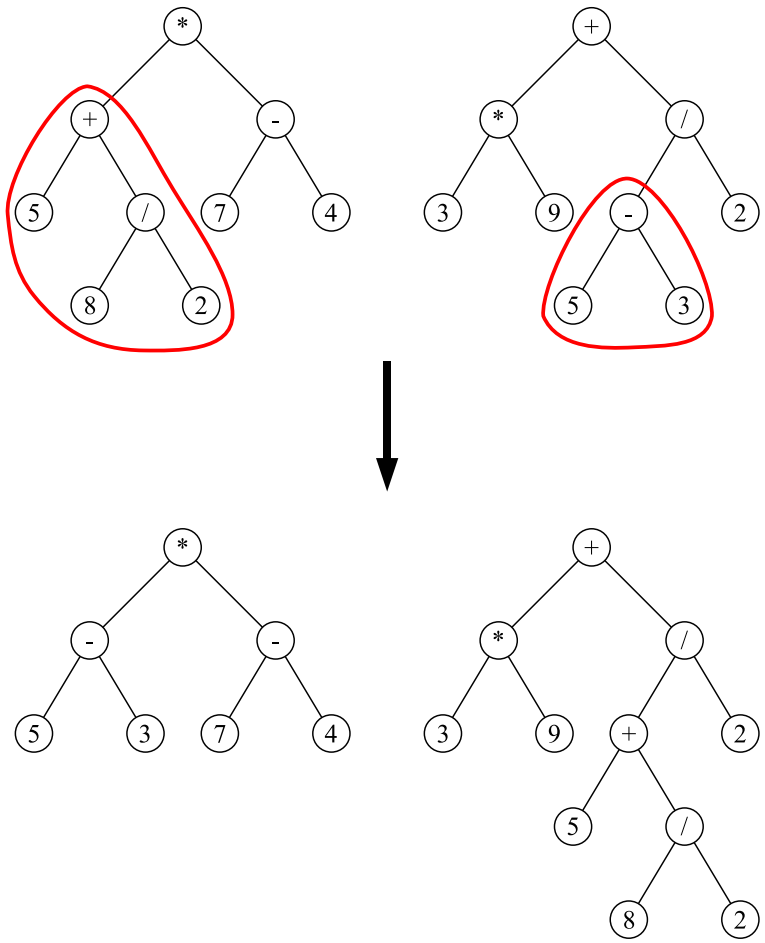
FIG. 3.3 : *Illustration du calcul d'un algorithme représenté sous forme d'arbre.*

Il faut donc définir un ensemble de fonctions (par exemple {+, -, *, /}) et un ensemble de données (par exemple {1, 2, 3, 4, 5, 6, 7, 8, 9}) utilisables dans les programmes. En général, les programmes manipulent un seul type de données (des nombres entiers dans l'exemple présenté ici). Cela simplifie le problème en garantissant la validité des algorithmes dans tous les cas.

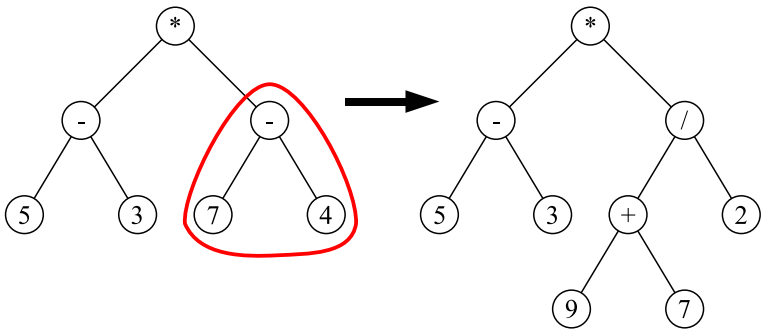
La population initiale est constituée d'algorithmes générés aléatoirement. Cette génération s'effectue en partant de la racine et en sélectionnant aléatoirement une fonction pour ce nœud. La génération se fait ensuite récursivement pour chaque nœud nouvellement créé en sélectionnant aléatoirement un nœud fonction ou donnée. La génération s'arrête lorsque toutes les feuilles de l'arbre sont des données. On définit généralement une profondeur maximum pour les arbres afin d'éviter la construction d'algorithmes trop complexes.

Les algorithmes sont ensuite évalués et sélectionnés en fonction de l'application visée. Pour l'étape de reproduction, le croisement consiste à sélectionner aléatoirement un nœud dans chaque algorithme parent, puis à échanger les sous-arbres issus de ces nœuds. La mutation consiste à sélectionner aléatoirement un nœud dans l'algorithme cible et à remplacer le sous-arbre issu de ce nœud par un sous-arbre généré aléatoirement (figure 3.4).

La suite du processus est identique à ce que nous avons présenté précédemment sur les algorithmes évolutionnaires. Les individus nouvellement créés constituent une nouvelle génération,



(a) *Croisement*



(b) *Mutation*

FIG. 3.4 : *Opérateurs de transformation utilisés en programmation génétique.*

ils vont être à leur tour évalués et sélectionnés lors d'un nouveau cycle, et ainsi de suite jusqu'à la fin du processus d'évolution.

Nous avons présenté au §2.3.3.1 des méthodes permettant de garantir la validité d'algorithmes manipulant plusieurs types de données lors de la génération initiale, notamment en utilisant une grammaire non contextuelle. Pour la transformation des algorithmes, l'opérateur de croisement consiste toujours à échanger deux sous-arbres, mais ici les deux sous-arbres doivent avoir pour racine un nœud contenant le même symbole non-terminal. L'opérateur de mutation consiste toujours à remplacer un sous-arbre par un autre généré aléatoirement. Ce nouveau sous-arbre est généré de la même manière que lors de la génération initiale de l'algorithme, et en utilisant la même grammaire. La seule différence est que l'on commence ici avec le symbole correspondant à la racine du sous-arbre et non pas avec le symbole "DÉPART". L'utilisation de ces opérateurs garantit que tous les algorithmes transformés par le système seront valides. Il est également possible de fixer des probabilités différentes pour la sélection des nœuds non-terminaux lors du croisement et de la mutation [Whigham 96]. Dans notre système, nous avons uniquement utilisé le biais sur les règles de production comme indiqué au §2.3.3.2.

3.1.3 Exemples d'applications des algorithmes évolutionnaires à la vision

Les algorithmes évolutionnaires et la programmation génétique en particulier ont été utilisés pour nombre d'applications, de la conception de circuits électriques analogiques aux systèmes de stabilisation pour des drones. Nous allons présenter ici quelques exemples d'applications plus directement liées au domaine qui nous intéresse dans cette thèse, à savoir la vision par ordinateur.

3.1.3.1 Détection de points d'intérêt

Marc Ebner a développé un système utilisant la programmation génétique pour détecter automatiquement des points d'intérêt dans une image et les mettre en correspondance dans une suite d'images [Ebner 99]. Cette approche présente plusieurs points communs avec le système développé dans le cadre de cette thèse. Tout d'abord, il considère qu'un algorithme de vision est lié de manière inhérente à la tâche pour laquelle il est utilisé. Dans son cas, la tâche visée est le calcul d'un flux optique épars sur une séquence d'images. Il définit donc pour sa fonction de fitness un certain nombre de critères qui vont refléter la qualité des points détectés pour le calcul d'un flux optique épars. Ces critères sont le nombre de points détectés, la qualité de la mise en correspondance, le ratio entre le nombre de points extraits et le nombre de points pour lesquels une correspondance a été trouvée, la non-ambiguïté des correspondances, la régularité et la densité du flux optique ainsi calculé. Il combine ensuite ces critères en une seule fonction de fitness et prouve que son système fournit de meilleures correspondances que les systèmes existants selon cette fonction. L'idée majeure est que ces opérateurs ne sont pas forcément meilleurs dans l'absolu, mais qu'ils sont plus adaptés pour effectuer la tâche fixée. Un autre point commun avec nos travaux est la liste des primitives visuelles utilisées qui est assez proche de celle que nous avons définie. Il utilise notamment des filtres de Gabor, des filtres gaussiens ainsi que des différences de gaussiennes.

Leonardo Trujillo a utilisé une approche similaire pour détecter des points d'intérêt, la principale différence résidant dans la fonction de fitness utilisée [Trujillo 06]. Ici, les détecteurs sont évalués par leur taux de répétabilité (nombre de points qui sont détectés systématiquement lorsqu'on change l'angle de vue de la scène) et la répartition des points dans l'image. Il arrive ainsi à recréer des détecteurs de points d'intérêts très simples mais efficaces comme la différence de gaussiennes ou le déterminant de la matrice hessienne. Dans des développements plus récents de ces travaux, il utilise des techniques d'évolution multi-critères basées sur le principe d'optimalité au sens de Pareto que nous décrirons au §3.3.1. Cela permet de ne pas comparer les différents critères directement et ainsi de sélectionner des opérateurs qui représentent différents compromis entre ces critères [Trujillo 08].

Ces systèmes se rapprochent fortement de notre propre approche, tout d'abord car il s'agit de générer des algorithmes de vision par programmation génétique, mais surtout parce qu'ils traitent directement le problème de l'apprentissage de la vision bas niveau. Toutefois ils prennent en compte uniquement cette première étape du processus de vision tandis que nous essayons de traiter la vision dans sa globalité, depuis la perception et l'extraction d'informations jusqu'à l'application finale.

3.1.3.2 Reconstruction 3D par stéréovision

Une autre application des algorithmes évolutionnaires à la vision est la reconstruction de scènes en trois dimensions à partir de couples d'images (stéréovision). Cette méthode a été développée initialement par Jean Louchet sous le nom d'algorithme des mouches [Louchet 02]. Elle s'inscrit dans l'approche dite parisienne, c'est à dire que la solution n'est pas représentée par un individu unique mais par un groupe d'individus. Les individus sont ici des points appelés mouches dans l'espace en trois dimensions. L'évaluation va consister à calculer la corrélation entre la région entourant la mouche dans chaque image. Si la mouche se situe sur un objet, cette corrélation sera élevée, et inversement si la mouche est loin de tout objet cette corrélation sera très probablement faible. Les opérateurs de transformation sont géométriques : pour le croisement, la nouvelle mouche se situe à mi-distance des deux parents et pour la mutation, elle est déplacée aléatoirement d'une petite distance. Après quelques générations, les mouches convergent vers la surface des objets et permettent ainsi d'effectuer une reconstruction 3D de l'environnement. Cette méthode a été utilisée également pour effectuer de l'évitement d'obstacles [Pauplin 05]. Une variante de cette approche a été proposée par Gustavo Olague. Celle-ci consiste à spécialiser les individus, qui sont ici appelés abeilles en référence aux capacités d'organisation et de communication des abeilles au sein d'une ruche. Une première catégorie d'abeilles va explorer l'image et repérer les différents objets. Une seconde catégorie va ensuite se répartir sur ces objets pour obtenir une représentation plus dense de ceux-ci [Olague 06].

Même si cette approche est très différente de la nôtre, il est intéressant de constater que des algorithmes évolutionnaires ont déjà permis d'obtenir de bons résultats en évitement d'obstacles basé sur la vision. Néanmoins comme indiqué précédemment, les systèmes basés sur la stéréovision nécessitent deux caméras précisément positionnées et calibrées ce qui les rend plus coûteux et complexes à mettre en œuvre.

3.1.3.3 Détection de sol pour l'évitement d'obstacles

Les travaux de Martin se rapprochent nettement plus de ce que nous avons réalisé pour cette thèse [Martin 06]. En effet, il utilise également la programmation génétique pour générer des algorithmes de vision, et l'application principale est l'évitement d'obstacles pour un robot mobile. La principale différence vient du fait qu'il se base uniquement sur la technique de détection de sol décrite au §2.1.2.1, tandis que nous cherchons à utiliser et combiner différentes techniques d'évitement d'obstacles. Plus précisément, son système effectue de l'apprentissage supervisé sur une base d'images enregistrée préalablement. Ces images sont acquises en laissant le robot se déplacer dans les couloirs du laboratoire. Il utilise dans ce cas un vrai sonar pour éviter les obstacles. Chaque image enregistrée est ensuite découpée en colonnes, puis étiquetée manuellement pour indiquer la limite entre le sol et les obstacles dans chaque colonne. Cet ensemble constitue la base d'apprentissage du système.

Un système basé sur la programmation génétique est ensuite utilisé pour concevoir un algorithme capable d'effectuer cet étiquetage automatiquement. La fonction de base des algorithmes développés est de déplacer une fenêtre verticalement dans la colonne. La taille de cette fenêtre est paramétrée par l'évolution. Différentes primitives visuelles peuvent ensuite être appliquées sur le contenu de cette fenêtre comme un filtre médian ou un filtre de Sobel par exemple. Le résultat de cette primitive est ensuite moyenné. Différentes valeurs issues de différents opérateurs peuvent être combinées à l'aide d'opérateurs arithmétiques. Elles peuvent également être enregistrées dans des registres et réutilisées plus tard. Deux algorithmes différents sont ainsi développés parallèlement, le deuxième pouvant réutiliser les résultats obtenus par le premier.

Les algorithmes évolués sont ensuite testés en temps réel sur un robot pour éviter les obstacles. Le programme de contrôle est ici développé manuellement et utilise les valeurs renvoyées par l'algorithme testé (correspondant à la hauteur des obstacles dans la colonne donnée) de manière similaire à l'utilisation d'un sonar. Les meilleurs algorithmes permettent ainsi au robot de se déplacer dans les couloirs du laboratoire assez rapidement et sans heurter d'obstacles. Néanmoins, comme les autres systèmes de détection de sol, ils échouent lorsque certains types de moquette recouvrent le sol. Un autre inconvénient de ce système est que le processus d'étiquetage manuel des images pour l'apprentissage peut être assez fastidieux.

3.2 Évaluation des algorithmes d'évitement d'obstacles

La phase de sélection dans les algorithmes évolutionnaires est basée sur une fonction d'évaluation aussi appelée fonction de fitness. Nous allons présenter dans cette section les différentes méthodes d'évaluation mises en œuvre dans notre système. Nous décrirons avant tout les environnements dans lesquels cette évaluation a lieu, puis nous détaillerons les fonctions de fitness utilisées.

3.2.1 Environnements d'évaluation

Les simulateurs robotiques sont un outil précieux pour le développement d'algorithmes pour robots mobiles car ils permettent à moindre coût et en toute sécurité de valider les principes mis

en œuvre. Le principal inconvénient à leur utilisation est que l'environnement est moins réaliste, et donc que les algorithmes évolués ainsi sont peu susceptibles de fonctionner en environnement réel. C'est pourquoi nous avons également réalisé des expériences sur robot réel afin de valider notre approche. Nous allons présenter ici premièrement le simulateur utilisé et les environnements de simulation, puis l'environnement réel dans lequel nous avons réalisé la seconde partie de nos expériences.

3.2.1.1 Environnement de simulation

La majeure partie des expériences présentées dans cette thèse a été réalisée dans un environnement de simulation. Les principaux avantages à l'utilisation d'un simulateur sont :

- Les expériences sont déterministes et reproductibles, ce qui s'avère très utile lors de la mise au point des algorithmes ou lorsqu'on souhaite effectuer des tests dans des conditions fixes.
- Il est possible d'évaluer tous les algorithmes dans des conditions identiques, sans problème de repositionnement ou de variation dans les conditions d'éclairement par exemple.
- Il est possible d'effectuer des expériences longues (plusieurs jours) sans se soucier des problèmes d'autonomie liés aux robots réels.

Nous utilisons ici le simulateur Gazebo qui fait partie du projet Player/Stage/Gazebo, plateforme *open source* dédiée à la robotique (http://playerstage.sourceforge.net). Ce simulateur est basé sur ODE pour la simulation physique (*Open Dynamic Engine*, http://www.ode.org) et sur OGRE pour le rendu visuel (*Object-Oriented Graphics Rendering Engine*, http://www.ogre3d.org).

Les images produites sont en niveaux de gris et codées sur 8 bits. Dans la plupart des expériences, l'angle de vue horizontal de la caméra du simulateur est fixé à 100^{o} et les images ont une résolution de 320×160. Cet angle de vue large réduit les angles morts et facilite ainsi la détection des obstacles. Dans les expériences où l'on a cherché à reproduire des conditions proches de l'environnement réel, l'angle de vue horizontal est fixé à 42^{o} et la résolution est de 320×240.

Nous avons créé plusieurs environnements de simulation afin de tester les capacités de notre système à faire évoluer des algorithmes adaptés à différentes conditions. D'une manière générale, chaque environnement est une pièce de 36 m^2 (6 m $\times$ 6 m) contenant plusieurs obstacles. Nous avons créé deux environnements très simples où ces obstacles sont des blocs cubiques de 1 m de côté. Nous avons également créé des environnements plus réalistes où les obstacles sont des étagères d'environ 2 m de long, 50 cm de large et 2 m de haut. Des aperçus de ces différents environnements sont présentés en figure 3.5.

3.2.1.2 Environnement réel

Nous avons également testé notre système en utilisant un robot réel se déplaçant dans les couloirs du laboratoire. Le robot est un Pioneer 3 DX de Mobile Robots équipé d'une caméra Canon VC-C50i (figure 3.6). Celui-ci peut être piloté en utilisant l'interface Player ce qui facilite la réutilisation des algorithmes entre les environnements de simulation et l'environnement réel.

(a) *Environnement simple contenant des blocs non texturés*

(b) *Environnement simple contenant des blocs texturés*

(c) *Environnement plus réaliste contenant trois étagères*

FIG. 3.5 : *Cartes et aperçus des environnements de simulation créés pour tester l'adaptation des algorithmes à différents contextes visuels.*

Notre système permet de contrôler le robot à l'aide d'un joystick, ce qui s'avère utile pour enregistrer des séquences d'images ou pour prépositionner le robot pour une expérience.

Les expériences sont réalisées dans un couloir de notre laboratoire. Cet environnement visuel reste relativement simple car les murs sont blancs et le sol est sombre. Cependant les différences d'éclairage et les ouvertures sur les côtés du couloir ajoutent un peu de complexité au traitement des images. La figure 3.7 présente une carte et un aperçu de ce couloir.

Nous avons également effectué des tests dans un autre couloir de l'ENSTA. L'environnement visuel est ici plus complexe : le sol est brillant et texturé et des casiers verticaux sont présents le long d'un des murs (figure 3.8).

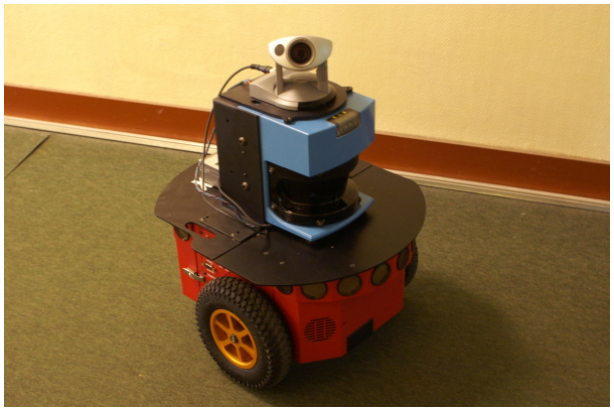


FIG. 3.6 : *Robot Pioneer 3 DX utilisé pour les expériences en environnement réel équipé d'une caméra Canon VC-C50i et d'un télémètre laser (non utilisé dans nos expériences).*

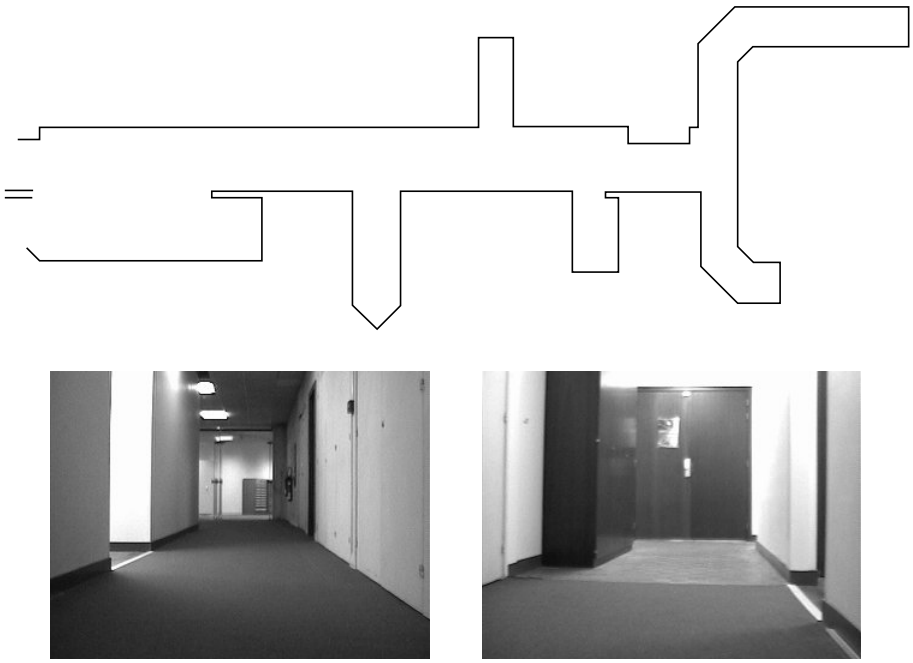


FIG. 3.7 : *Carte du laboratoire et aperçu du couloir utilisé pour nos expériences. Il s'agit du long couloir central sur la carte.*

FIG. 3.8 : *Carte et aperçu d'un autre couloir de l'ENSTA utilisé dans nos expériences.*

3.2.2 Fonctions d'évaluation

L'optimisation par algorithmes évolutionnaires nécessite l'utilisation d'une fonction d'évaluation aussi appelée fonction de fitness pour sélectionner les individus. Nous avons défini plusieurs fonctions de fitness dans notre système pour évaluer les algorithmes de vision. Cette section présente les différents critères utilisés pour évaluer les algorithmes d'évitement d'obstacles ainsi que la formalisation de ces critères en fonctions d'évaluation.

3.2.2.1 Choix des critères d'évaluation d'un comportement d'évitement d'obstacles

Il est tout d'abord nécessaire de préciser quels sont les critères permettant d'évaluer les capacités d'évitement d'obstacles d'un contrôleur. Le critère le plus simple pour cela consiste à compter le nombre de contacts ayant eu lieu avec des obstacles durant un temps donné. Le problème de cette approche est qu'un robot restant coincé contre un obstacle dès le premier contact sera mieux évalué qu'un robot capable de reprendre sa route après un choc et qui heurtera plusieurs obstacles sur son chemin. Nous avons donc choisi d'évaluer cette capacité d'évitement d'obstacles sur le temps passé à moins d'une distance donnée d'un obstacle. Cela permet ainsi de récompenser les contrôleurs capables de se sortir d'une situation de contact et également de mieux noter ceux qui resteront toujours relativement éloignés des obstacles, adoptant ainsi un comportement plus prudent.

Si on utilise ce seul critère, les contrôleurs évolués vont adopter le comportement le plus simple permettant de rester éloigné des obstacles, consistant à ne pas se déplacer. Il est alors nécessaire d'ajouter un critère d'évaluation supplémentaire, comme par exemple la distance parcourue par le robot. Les contrôleurs évolués vont alors se déplacer très rapidement en suivant une trajectoire circulaire ne comprenant pas d'obstacle, ce qui n'est guère plus intéressant. On peut également imaginer d'autres critères pour limiter les virages ou explorer la plus grande partie possible de l'environnement mais le problème de ces approches est qu'elles imposent des critères supplémentaires qui ne sont liés ni à l'évitement d'obstacles proprement dit, ni à une quelconque tâche que devrait accomplir le robot. Pour éviter d'utiliser de tels critères

d'évaluation supplémentaires, il est également envisageable de "forcer" le robot à se déplacer à une certaine vitesse ou dans un certain cône de direction par exemple. Le problème est que ces contraintes peuvent se montrer très handicapantes pour un comportement d'évitement d'obstacles efficace. Une vitesse minimum exclut ainsi la possibilité d'effectuer des marches arrières, qui peuvent pourtant présenter de grands avantages pour éviter des obstacles (et même être parfois indispensables si l'on considère un robot mobile à quatre roues de type voiture).

Cette discussion met en lumière un point important, qui est qu'un comportement d'évitement d'obstacles n'a de sens que dans le cadre d'une tâche plus large comme par exemple le déplacement vers une zone donnée ou l'exploration de l'environnement. Le second critère que nous avons choisi est donc directement lié à cette tâche, il s'agit ici de la capacité du robot à atteindre rapidement un point donné dans l'environnement. Notons qu'il ne s'agit en aucune manière de planifier un déplacement, tâche complexe bien au-delà des capacités attribuées aux contrôleurs dans cette thèse. Les déplacements nécessaires restent donc toujours simples, avec un nombre limité d'obstacles, sans cul-de-sac et avec un chemin relativement direct entre le point de départ et celui d'arrivée. Lors d'une utilisation réelle, un tel système devrait bien entendu être couplé avec un planificateur qui aurait pour tâche de découper le déplacement total en une suite de *waypoints* correspondant à des déplacements relativement simples similaires à ceux étudiés ici.

Remarquons enfin que les deux critères proposés ici ont une forte connotation temporelle (temps passé près d'un obstacle et temps nécessaire pour atteindre un point cible). D'autres critères sont bien sûr envisageables, comme par exemple le nombre de manœuvres, la longueur de la trajectoire réalisée ou encore la vitesse moyenne le long de la trajectoire. Notons toutefois que le nombre de manœuvres est un critère difficile à définir formellement et donc à évaluer objectivement. Les deux autres critères évoqués sont quant à eux assez directement liés au temps nécessaire pour atteindre la cible et nous pensons que les résultats obtenus seraient similaires avec ceux-ci.

3.2.2.2 Définition formelle des fonctions d'évaluation

Les critères présentés précédemment sont ici formalisés sous forme de fonctions. Dans cette optique, nous nous attachons d'abord à préciser les conditions expérimentales. Nous définissons ainsi un point de départ et un point cible dans l'environnement. À tout moment, le robot connaît la direction et la distance du point cible. L'objectif est d'atteindre le point cible le plus rapidement possible sans heurter d'obstacles. Pour cela, le robot est placé au point de départ puis il se déplace librement en étant guidé uniquement par l'algorithme à évaluer. L'expérience prend fin lorsque le robot est à moins de 10 cm du point cible ou après un temps maximum fixé à 30 s ou 60 s suivant les expériences. La fonction de fitness calcule deux scores indiquant la performance du robot, un score de but B et un score de contact C :

$$B = \begin{cases} t_{\mathrm{B}} \text{ si le but est atteint} \\ t_{\max} + d_{\min}/V \text{ sinon} \end{cases} \tag{3.1}$$

$$C = t_{\mathrm{C}} \tag{3.2}$$

Dans ces équations, t_B est le temps mis pour atteindre le but en secondes, $t_{\max}$ est le temps maximum de l'expérience en secondes, $d_{\min}$ est la distance au but minimum qui ait été atteinte

durant l'expérience en mètres, V est une constante qui vaut 0,1 m/s et t_C est le temps passé près d'un obstacle (c'est à dire à moins de 10 cm environ). Le but est donc de minimiser ces deux scores B et C. Nous verrons au §3.3.1 la méthode utilisée pour optimiser deux objectifs différents et parfois contradictoires.

La seconde fonction d'évaluation utilisée dans nos expériences évalue la capacité d'un algorithme à imiter un comportement enregistré, qui correspond typiquement à une séquence durant laquelle le robot était piloté manuellement. Cette séquence contient les images enregistrées par la caméra lors du déplacement ainsi que les commandes envoyées au robot à chaque instant. Les algorithmes sont évalués dans ce cas sur leur capacité à reproduire les mêmes commandes lorsqu'on repasse la séquence d'images en entrée. Formellement, la fonction de fitness calcule deux scores L et A définis comme :

$$L = \sqrt{\sum_{i=1}^{n} (l_{\mathrm{E}i} - l_{\mathrm{T}i})^2} \tag{3.3}$$

$$A = \sqrt{\sum_{i=1}^{n} (a_{\mathrm{E}i} - a_{\mathrm{T}i})^2} \tag{3.4}$$

Dans ces équations, $l_{\mathrm{E}i}$ et $a_{\mathrm{E}i}$ sont les commandes de vitesse linéaire et angulaire enregistrées pour l'image i, $l_{\mathrm{T}i}$ et $a_{\mathrm{T}i}$ sont les commandes de vitesse linéaire et angulaire issues de l'algorithme testé pour l'image i et n est le nombre d'images dans la séquence vidéo. Cette fonction d'évaluation est donc plus subjective car elle dépend entièrement de la séquence vidéo pré-enregistrée. Elle est utile pour créer des algorithmes capables de reproduire un comportement difficile à formaliser autrement.

3.3 Évolution d'algorithmes d'évitement d'obstacles efficaces

Nous allons maintenant décrire les techniques plus spécifiques mises en œuvre dans notre système pour améliorer l'efficacité des contrôleurs évolués. Certaines sont connues et largement utilisées par ailleurs, comme l'optimisation multi-objectifs. D'autres ont par contre été développées spécifiquement pour notre système, comme l'évolution en deux phases basée sur l'imitation.

3.3.1 Optimisation multi-objectifs

Comme nous l'avons indiqué au §3.2.2, nous cherchons à sélectionner les algorithmes de vision selon plusieurs critères, parfois contradictoires. De nombreuses méthodes ont été proposées pour cela, les algorithmes évolutionnaires multi-objectifs constituant un domaine de recherche à part entière. Notons que nous employons ici les termes d'objectif et de critère indifféremment. Il s'agit pour le moins d'un abus de langage, un objectif étant en réalité la minimisation ou la maximisation d'un score obtenu en évaluant la solution selon un critère donné. La technique

la plus simple pour traiter ce problème consiste à le ramener à une optimisation mono-objectif classique. On peut pour cela combiner les différents objectifs par une moyenne pondérée par exemple. L'inconvénient de cette méthode est que l'importance accordée à chaque objectif est alors arbitraire et que l'on force la comparaison entre des critères qui ne sont pas forcément comparables. On peut effectuer plusieurs optimisations avec des pondérations différentes mais le processus devient dans ce cas beaucoup plus long. Une variante de cette méthode consiste à effectuer une optimisation selon chaque critère indépendamment, et de faire le choix de la meilleure solution ensuite mais cela ne permet pas vraiment d'effectuer de compromis entre les critères.

Le premier à proposer un système pour produire en une seule exécution plusieurs solutions et sans comparer directement les différents critères fut Schaffer avec son système VEGA (Vector Evaluated Genetic Algorithm) [Schaffer 85]. Celui-ci consiste à effectuer à chaque génération plusieurs sélections indépendantes, chacune étant basée sur un seul critère à optimiser. On obtient ainsi plusieurs groupes d'individus qui sont ensuite remélangés pour les étapes de croisement et de mutation. Ce système ne permet cependant pas de sélectionner les solutions qui correspondent à un compromis entre les différents critères.

L'avancée majeure pour les algorithmes évolutionnaires multi-objectifs fut l'introduction des méthodes basées sur le principe d'optimalité au sens de Pareto. Ce principe dit qu'une solution est Pareto optimale si toute autre solution qui est meilleure selon un des objectifs est moins bonne selon un autre. Formellement, ce principe peut s'exprimer ainsi :

Soit un problème consistant à minimiser simultanément n objectifs et une fonction d'évaluation $f(x)$ qui renvoie pour une solution possible x un vecteur à n dimensions correspondant à la fitness de la solution selon chaque objectif : $f(x) = (f_1(x), f_2(x), ..., f_n(x))$. Une solution x est optimale ou non-dominée au sens de Pareto s'il n'existe aucune solution y telle que :

$$\forall i \in \{1,...,n\}, f_i(y) \leqslant f_i(x) \quad \wedge \quad \exists i \in \{1,...,n\}, f_i(y) < f_i(x) \tag{3.5}$$

Les algorithmes évolutionnaires qui se basent sur ce principe pour la phase de sélection des individus peuvent ainsi effectuer des compromis entre différents objectifs sans jamais les comparer directement. Au final, l'algorithme fournit l'ensemble des solutions non-dominées et l'utilisateur peut choisir celle(s) qui représente(nt) le meilleur compromis à ses yeux. Plusieurs algorithmes évolutionnaires ont été développés au début des années 90 en utilisant ces principes [Fonseca 95].

D'autres travaux ont ensuite amélioré ces algorithmes en favorisant une meilleure répartition des solutions le long du front de Pareto ou en utilisant une sélection élitiste pour ne pas perdre de bonnes solutions au cours de l'évolution [Zitzler 99]. Kalyanmoy Deb a ainsi amélioré l'algorithme NSGA [Srinivas 94] en incorporant une sélection élitiste et en favorisant la diversité parmi les solutions [Deb 02]. Ce nouvel algorithme NSGA-II est largement utilisé aujourd'hui en optimisation multi-critères. C'est celui que nous avons intégré à notre système et que nous allons présenter plus en détail ici.

La principale différence entre NSGA-II et un algorithme évolutionnaire classique se situe au niveau de la phase de sélection des individus. Ceux-ci sont d'abord évalués par la fonction de fitness qui va leur attribuer un score pour les différents objectifs. On va ensuite classer ces individus par rang de non-domination. Cela signifie qu'on va extraire tous les individus non

dominés de la population. Ceux-ci vont constituer le rang 1 d'individus non-dominés. On va ensuite extraire les individus non-dominés de la population restante, qui vont constituer le rang 2 et ainsi de suite jusqu'à ce que tous les individus de la population soient extraits (voir la figure 3.9). Cette méthode garantit que les individus non-dominés seront toujours mieux classés dans la population que les individus dominés.

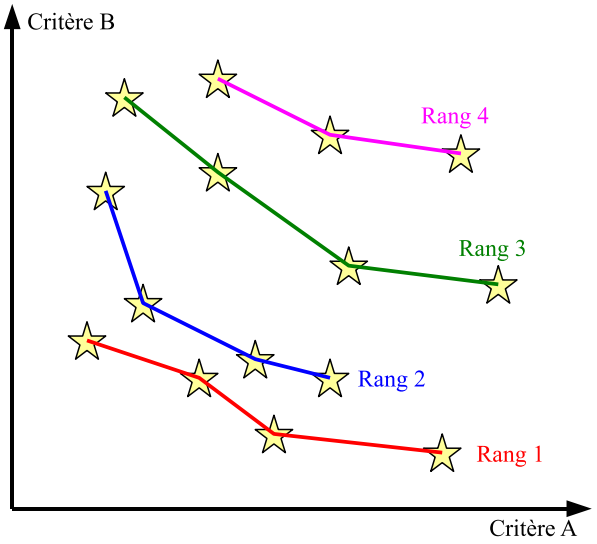


FIG. 3.9 : *Illustration du mécanisme de classement des individus par rang de non-domination dans NSGA-II. Les scores obtenus par les individus selon les critères A et B sont représentés par les étoiles. Le but est ici de minimiser les deux critères.*

La sélection pour la reproduction se fait ensuite par tournoi à deux, c'est à dire que l'on va sélectionner à chaque fois deux individus aléatoirement, et que le meilleur des deux pourra se reproduire. Pour cela, il faut que tous les individus soient ordonnés, et donc que l'on détermine un ordre à l'intérieur de chaque rang de non-domination. C'est à ce niveau que l'on va introduire une mesure permettant de favoriser une bonne répartition des individus le long du front de Pareto. On va calculer pour chaque individu une distance de voisinage (*crowding distance* en anglais) indiquant si d'autres individus sont proches de lui ou non (figure 3.10). Cette distance est la distance moyenne normalisée de ses voisins dans le même rang de non-domination selon chaque objectif. Formellement, soit n le nombre d'objectifs et l le nombre de solutions dans le rang de non-domination. On appelle $f_0^j \dots f_l^j$ les scores triés par ordre décroissant des solutions $\{0, \dots, l\}$ selon l'objectif j et i_j l'index de l'individu i dans ce classement. La distance de voisinage d_i pour l'individu i s'exprime ainsi :

$$\text{si } \exists j,\ i_j = 0 \vee i_j = l \text{ alors } d_i = \infty$$

$$\text{sinon } d_i = \sum_{j=1}^{n} \frac{f_{i_j-1}^j - f_{i_j+1}^j}{f_0^j - f_l^j}$$

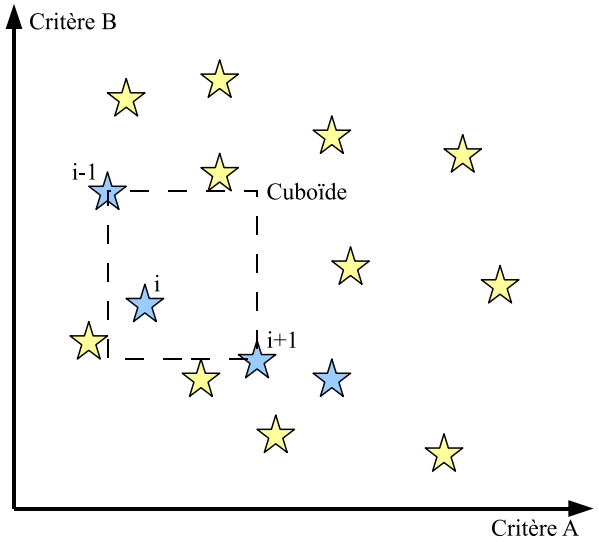


FIG. 3.10 : *Calcul de la distance de voisinage avec l'algorithme NSGA-II. Les étoiles représentées en bleu correspondent aux solutions appartenant au même rang de non-domination. La distance de voisinage pour la solution i correspond au périmètre du cuboïde représenté sur la figure.*

On classe donc les individus à l'intérieur de chaque rang de non-domination selon cette distance de voisinage. Plus cette distance est grande, plus l'individu est loin de ses voisins et plus il aura donc de chances d'être sélectionné pour se reproduire.

La dernière spécificité de NSGA-II est la manière dont l'on sélectionne les individus qui vont constituer la génération suivante. Il s'agit d'un algorithme élitiste, c'est à dire que les meilleures solutions trouvées ne seront pas éliminées de génération en génération. Pour cela, supposons que la taille de la population soit N. On constitue après l'étape de reproduction une population de taille $2N$ constituée de la génération précédente et des individus nouvellement créés. Les nouveaux individus sont évalués et toute cette population est classée comme indiqué précédemment. Les N meilleurs individus selon ce classement seront conservés pour la génération suivante, ce qui garantit que les meilleurs ne seront jamais éliminés. Cet algorithme permet ainsi de rechercher les individus les plus proches possibles du front de Pareto pour le problème donné et de sélectionner ceux qui présentent une bonne répartition le long de ce front.

3.3.2 Promotion de la diversité par découpage de la population

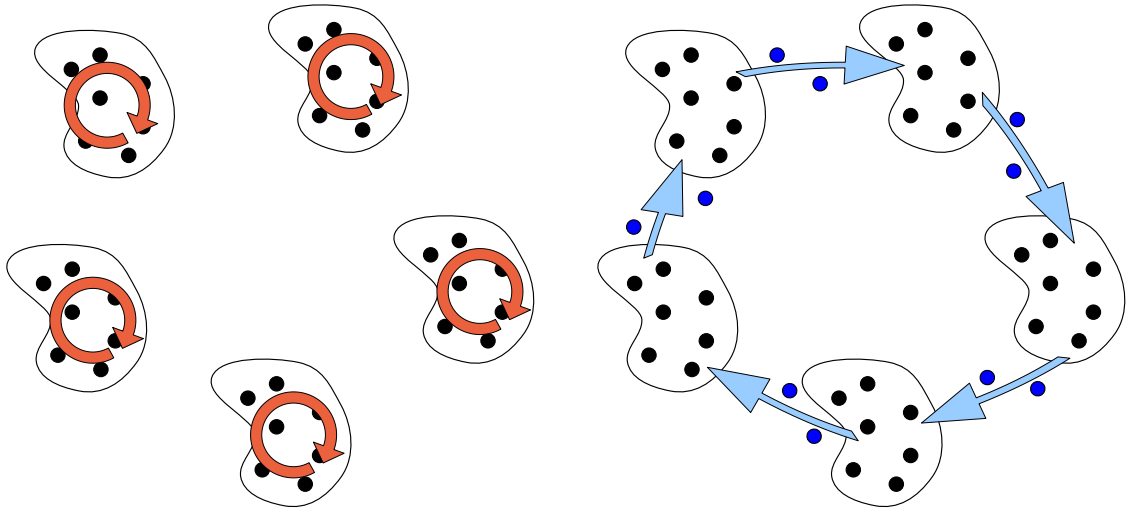
La diversité est une notion importante pour l'optimisation par algorithmes évolutionnaires. Elle va indiquer l'étendue couverte par la population dans l'espace de recherche et influe donc sur les chances de trouver les solutions optimales. De nombreux algorithmes intègrent des mesures de conservation ou de promotion de la diversité afin d'éviter les problèmes de convergence prématurée vers un optimum local. Une population où la diversité est importante s'adaptera également plus vite dans les cas où la fonction d'évaluation évolue au cours du temps. Différentes méthodes ont été proposées pour promouvoir la diversité au sein de la population :

- Les techniques les plus courantes s'appuient sur une mesure explicite de diversité pour préserver celle-ci. Ces techniques sont généralement basées sur le concept de niche écologique. Certains groupes d'individus (espèces) vont se regrouper dans une niche écologique qui correspond à un optimum local de la fonction de fitness, mais seul un nombre limité peut survivre dans chaque niche. Cela évite que tous les individus deviennent similaires. Cette technique est souvent implémentée par partage de fitness (*fitness sharing*) ou par une mesure de densité (*crowding*) comme celle présentée précédemment [Sareni 98]. Selon les cas, cette mesure est calculée directement à partir des individus (calcul dans l'espace des solutions) ou à partir des résultats qu'ils obtiennent en leur appliquant la fonction d'évaluation (calcul dans l'espace des objectifs).
- Une autre technique consiste à utiliser une structure de population particulière pour promouvoir implicitement la diversité. Différentes structures peuvent être utilisées pour cela, les plus courantes étant la structure cellulaire où les individus sont organisés spatialement sur une grille et la structure en îlots que nous allons décrire plus en détail par la suite [Tomassini 99]. D'autres variantes existent comme le modèle de patchwork [Krink 99], le modèle multinational [Ursem 99] ou le modèle basé sur la religion [Thomsen 00].
- Enfin, certaines méthodes tentent d'enrayer la perte de matériel génétique en procédant périodiquement à des suppressions massives d'individus (modèles d'extinction de masse) et en les remplaçant par de nouveaux individus créés par mutation à partir des survivants [Krink 01].

Nous avons présenté dans la section précédente l'utilisation d'une distance de voisinage dans l'algorithme NSGA-II pour promouvoir la diversité. En plus de celle-ci, nous avons également utilisé une structure de population en îlots pour prévenir les problèmes de convergence prématurée. En pratique, cela consiste à découper la population en plusieurs groupes ou îlots entre lesquels les échanges sont très limités. A l'intérieur de chaque îlot, l'évolution a lieu de manière classique. On est donc en présence de plusieurs processus d'évolution se déroulant en parallèle. De plus, toutes les N générations, un nombre M d'individus va "migrer" vers un îlot voisin et le même nombre M d'individus va être reçu depuis un autre îlot. La population des îlots reste donc constante tout au long du processus. Les connexions entre les îlots sont définies par une topologie. La plus courante est la topologie en anneau, mais d'autres sont possibles comme l'étoile, les archipels ou une topologie aléatoire. Ce processus est résumé sur la figure 3.11. Dans nos expériences, nous utilisons quatre îlots connectés selon une topologie en anneau. Toutes les 10 générations, 5 individus sélectionnés par tournoi à deux vont migrer vers l'îlot voisin.

L'objectif de ce découpage est d'empêcher les solutions représentant des optima locaux et trouvées précocement de se propager trop rapidement à l'ensemble de la population. Avec ce système une telle solution se propagera uniquement dans son îlot, et il y a des chances que de meilleures solutions soient trouvées par ailleurs avant qu'elle n'atteigne les autres. Les paramètres N et M permettent de définir la vitesse à laquelle les solutions vont se propager entre îlots. Dans le cas extrême où N vaut 1 et M est très grand, le système est équivalent à un algorithme classique avec une seule population. Dans l'autre cas extrême où M vaut 0, les évolutions sont totalement indépendantes, ce qui revient à effectuer la même expérience plusieurs fois.

Notons que le découpage de la population s'accompagne souvent d'une répartition sur différentes ressources matérielles et d'une parallélisation des traitements [Alba 02]. Ce lien n'est



(a) *Les processus d'évolution se déroulent indépendamment sur chaque îlot.*

(b) *Toutes les N générations, quelques individus vont migrer sur l'îlot voisin.*

FIG. 3.11 : *Illustration du processus d'évolution avec découpage de la population en îlots. La topologie utilisée ici pour la migration est en anneau.*

pourtant absolument pas obligatoire, un algorithme en îlots pouvant très bien être exécuté sur un seul processeur. Inversement, il est possible de paralléliser les évaluations dans un algorithme basé sur une population unique. Dans notre cas, le découpage en îlots a été implémenté en utilisant la librairie *open source* Paradiseo (http://paradiseo.gforge.inria.fr). Celle-ci fournissant également des possibilités de parallélisation basées sur MPICH2, nous avons distribué les traitements liés à chaque îlot sur un processeur différent.

3.3.3 Évolution guidée

Un problème récurrent en programmation génétique est que l'espace de recherche est très grand par rapport à la taille de la population. En effet cette dernière est limitée par le temps nécessaire à l'évaluation de tous les individus, alors que l'espace de recherche a potentiellement une dimensionnalité infinie si l'on ne limite pas la taille des arbres. De plus, le "paysage" de la fonction de fitness est généralement très irrégulier et chaotique, un changement mineur dans l'algorithme pouvant avoir des répercutions drastiques sur le résultat. De ce fait, la fonction de fitness présente beaucoup d'optima locaux ce qui rend la recherche d'un optimum global d'autant plus complexe.

Il peut donc être intéressant de mettre en œuvre des techniques pour guider le processus d'évolution afin de restreindre la recherche à des zones plus prometteuses. Les techniques existant pour cela sont de deux types :

- Celles qui vont guider explicitement l'évolution en introduisant des individus créés manuellement dans la population initiale (techniques dites d'ensemencement ou *seeding*).
- Celles qui vont guider implicitement l'évolution en découpant le problème à résoudre en sous-problèmes de difficulté croissante (techniques incrémentales).

Nous présentons ici une nouvelle technique basée sur l'imitation afin de guider l'évolution. Dans cette section, nous allons tout d'abord détailler les techniques existantes de *seeding* et d'évolution incrémentale, puis nous expliciterons notre méthode d'évolution en deux phases basée sur l'imitation.

L'utilisation des techniques de *seeding* suppose que l'on dispose ou que l'on a conçu manuellement une ou plusieurs solutions acceptables pour le problème donné. Ces solutions vont être utilisées pour guider le processus d'évolution. En pratique, cela consiste simplement à insérer dans la population initiale ces solutions particulières. Le reste de la population est constitué comme d'habitude de solutions générées aléatoirement et le processus se déroule ensuite normalement. L'avantage de cette technique est de permettre à l'évolution de produire rapidement de bonnes solutions et ce même avec de très petites populations. Par contre, cela va rendre plus difficile la découverte de solutions originales car celles-ci seront dominées par les solutions manuelles au début de l'évolution et vont généralement disparaître rapidement. Cette technique introduit donc un *a priori* très fort sur le type de solution attendu. Au final, le *seeding* est plus utile pour optimiser une solution donnée que pour réellement découvrir de nouvelles solutions originales.

De ce point de vue, l'évolution incrémentale reste plus dans la philosophie des algorithmes évolutionnaires qui est de limiter les *a priori* afin de découvrir des solutions originales. Elle consiste à décomposer le problème posé en plusieurs problèmes de difficulté croissante [Gomez 97]. L'idée sous-jacente est qu'un processus évolutionnaire classique ne sera pas toujours capable de résoudre un problème complexe. Dans de nombreux cas, les solutions aléatoires seront toutes à peu près aussi mauvaises, et leur recombinaison ne permettra pas de progresser rapidement vers de meilleures solutions. Il sera dans ce cas plus facile de résoudre une instance plus simple de ce problème, et à partir des solutions trouvées d'évoluer vers des solutions au problème initial. L'évolution incrémentale repose ainsi principalement sur une bonne décomposition du problème. L'évolution va se dérouler au début en utilisant comme fonction d'évaluation l'instance la plus simple du problème. Cette première phase peut durer un nombre fixe de générations ou jusqu'à ce que la fitness des meilleurs individus ait atteint un certain seuil. On remplace ensuite la fonction d'évaluation par une instance plus complexe du problème, et l'évolution continue à partir de la population évoluée précédemment. On procède ainsi jusqu'à la résolution du problème initial.

L'efficacité de cette méthode dépend essentiellement de la qualité de la décomposition du problème. Le postulat de base est qu'un individu efficace sur une instance simple du problème sera plus adapté à une instance plus complexe qu'un individu généré aléatoirement. Si ce postulat se révèle faux à cause d'une augmentation trop rapide de la complexité ou d'une trop grande différence entre les instances simples et les plus complexes, les solutions obtenues au final ne seront pas meilleures qu'avec un processus d'évolution classique. Cette décomposition du problème en instances de difficulté croissante est donc loin d'être triviale.

Notre méthode d'évolution en deux phases basée sur l'imitation est pour cela nettement plus simple à mettre en œuvre. Elle s'inspire à la fois des méthodes de *seeding* et de l'évolution incrémentale. Comme pour le *seeding* l'évolution va être "guidée manuellement", mais pas par l'insertion de solution préconçues dans la population. Comme pour l'évolution incrémentale le processus va être découpé en plusieurs phases mais celles-ci ne correspondent pas à différentes instances du même problème. Nous nous basons principalement sur le concept d'imitation dont l'utilité a largement été prouvée aussi bien pour l'apprentissage dans le monde

vivant que pour l'apprentissage artificiel (voir [Schaal 03] par exemple). Notons toutefois que le terme d'imitation englobe généralement le processus complet d'observation du comportement du "professeur", de transformation du point de vue et de reproduction de ce comportement par l'"élève". Ici, nous ne nous intéressons qu'à la dernière phase de reproduction d'un comportement. Il s'agit donc plus d'apprentissage supervisé que d'imitation à proprement parler.

En pratique, nous allons commencer par enregistrer un comportement de référence. Dans notre cas, cela consiste à guider manuellement le robot dans l'environnement pour atteindre le point cible tout en évitant les obstacles. Durant ce déplacement nous enregistrons toutes les données d'entrée (image de la caméra, distance et direction du but à chaque pas de temps) ainsi que les commandes données au robot. La première phase de l'évolution va consister à développer des algorithmes qui "imitent" cette trajectoire enregistrée. Pour cela, nous utilisons les fonctions d'évaluation 3.3 et 3.4 décrites au §3.2.2. Cette première phase dure la moitié de l'évolution, soit 50 générations. Pour la deuxième phase, nous gardons la population évoluée précédemment et nous changeons la fonction d'évaluation. Les individus vont ici être évalués plus objectivement sur leurs capacités d'évitement d'obstacles et leur rapidité en utilisant les fonctions 3.1 et 3.2.

L'idée est que la première phase va guider l'évolution vers des individus relativement efficaces, qui arrivent à atteindre le point cible. La deuxième phase va quant à elle optimiser ces algorithmes et les rendre plus génériques. Nous montrerons au §4.2 les résultats obtenus avec cette méthode par rapport au *seeding* et à l'évolution incrémentale. Notons déjà que cette méthode est particulièrement intéressante dans le domaine de la robotique évolutionnaire. Il est en effet beaucoup plus facile dans ce type d'application d'enregistrer des comportements de référence que de concevoir manuellement des contrôleurs efficaces ou des instances simplifiées du problème.

3.4 Bilan

Nous avons décrit dans ce chapitre la méthode employée pour faire évoluer les algorithmes de vision dont la structure a été décrite au chapitre 2. Nous utilisons pour cela des méthodes évolutionnaires et plus précisément la programmation génétique basée sur une grammaire. Nous avons également décrit les aspects plus spécifiques à notre système comme les méthodes d'évaluation et de sélection des algorithmes ainsi que la structure utilisée pour la population d'algorithmes.

La description de notre système étant à présent complète, nous allons présenter dans le chapitre 4 plusieurs expériences que nous avons réalisé afin de mettre en évidence l'adaptation des algorithmes au contexte, d'évaluer leurs capacités de généralisation et la possibilité de les utiliser dans un environnement réel.

Chapitre 4

Principaux résultats et enseignements

Ce chapitre a pour but de présenter et d'analyser les principaux résultats obtenus lors de cette thèse. Certains aspects seront décrits de manière plus complète et détaillée en annexe B. Nous allons tout d'abord vérifier l'adaptation des algorithmes évolués au contexte visuel. Nous tenterons pour cela de mettre en évidence les particularités des algorithmes évolués dans différents environnements et de faire le lien entre l'environnement utilisé et ces spécificités algorithmiques. Nous montrerons que ces contrôleurs évolués utilisent des primitives visuelles cohérentes mais présentent des performances médiocres, souvent en deçà de ce qui peut être obtenu avec des contrôleurs conçus manuellement. Nous présenterons donc ensuite les résultats obtenus en guidant l'évolution artificiellement pour avoir de meilleures performances, notamment en utilisant une méthode basée sur l'imitation d'un comportement pré-enregistré. Nous testerons également les capacités de généralisation de ces contrôleurs et décrirons les adaptations apportées à la grammaire pour améliorer le compromis entre évitement d'obstacles et déplacement vers la cible. Nous décrirons enfin les expériences que nous avons réalisées pour valider cette approche sur un robot réel. Pour conclure, nous présenterons plusieurs pistes possibles pour améliorer ce système et l'appliquer à d'autres tâches.

4.1 Différenciation des algorithmes en fonction de l'environnement

Cette section présente une première série d'expériences dont le but est de mettre en évidence l'adaptation des algorithmes évolués au contexte visuel. Nous utilisons pour cela les environnements de simulation décrits au §3.2.1.1, dont l'apparence visuelle est très différente, et nous comparons les algorithmes créés par notre système dans chaque environnement. Mais avant tout nous allons présenter les contrôleurs de référence qui ont été conçus manuellement pour chaque environnement afin de disposer d'un point de comparaison pour les algorithmes évolués.

4.1.1 Présentation des contrôleurs de référence conçus manuellement

Dans le premier environnement de simulation, constitué de blocs non texturés, nous avons conçu un contrôleur de référence utilisant un filtre de seuillage pour identifier les obstacles. Le

résultat est utilisé pour tourner dans la direction opposée à l'obstacle lorsqu'il est proche (c'est à dire dans la partie inférieure de l'image). La commande de vitesse angulaire est en réalité une moyenne pondérée entre cette commande d'évitement d'obstacles et la direction du but. Cela permet au robot de se diriger vers le but lorsqu'il n'y a pas d'obstacles proches et ainsi de ne pas se déplacer aléatoirement dans l'environnement. La vitesse est également ajustée pour que le robot ralentisse ou même recule lorsque les obstacles sont trop proches. Ce contrôleur et les trajectoires obtenues sont présentés en figure 4.1.

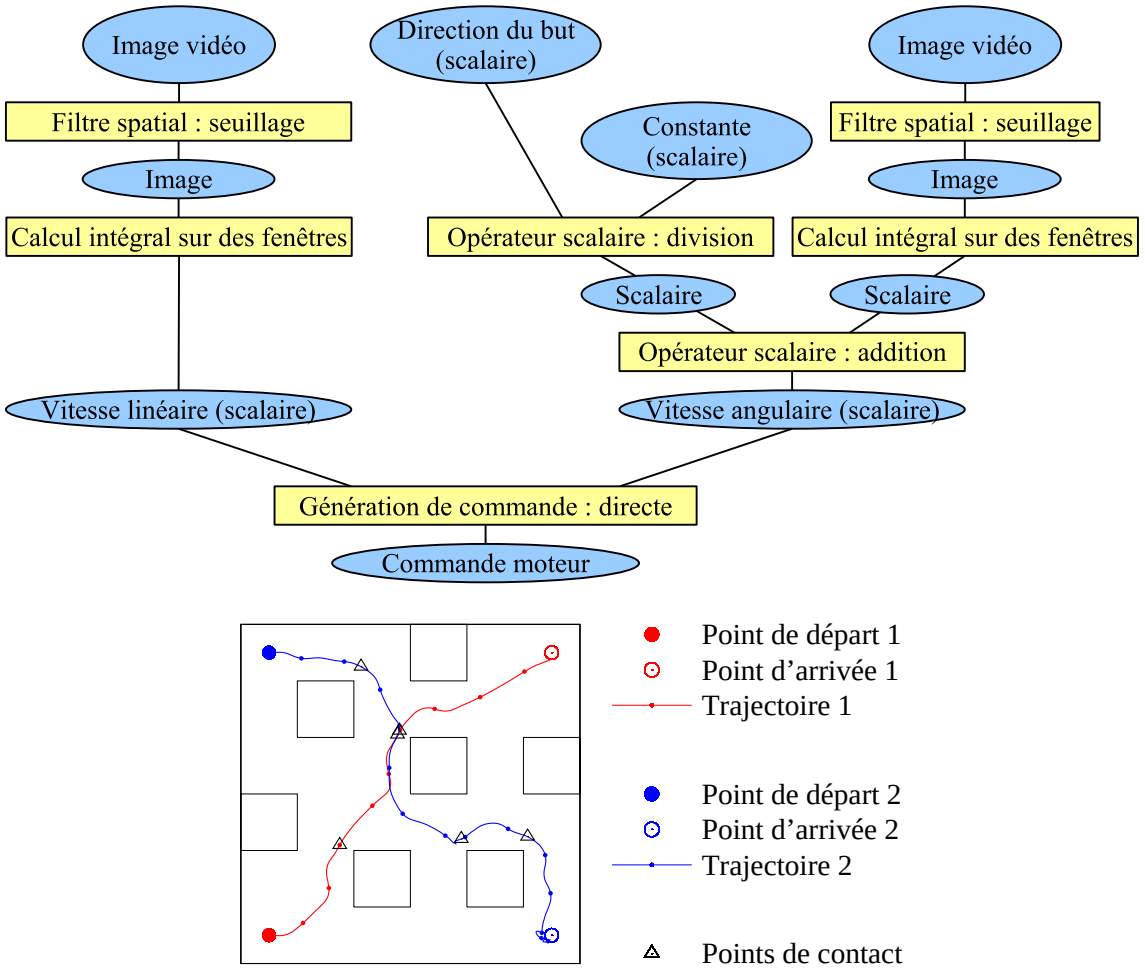


FIG. 4.1 : *Haut : Contrôleur manuel conçu pour diriger le robot dans l'environnement constitué de blocs non texturés. Bas : Trajectoires obtenues avec ce contrôleur. Dans ce graphique et tous les suivants, les points de contact désignent les endroits où le robot est passé à moins de 10 cm environ d'un obstacle. Il ne s'agit donc pas toujours d'un contact à proprement parler. Les points sur les trajectoires sont placés toutes les 2 secondes et représentent ainsi la vitesse du robot.*

Dans l'environnement constitué de blocs texturés, notre contrôleur utilise le flux optique pour détecter et éviter les obstacles. Nous rappelons que pour le calcul du flux optique, les rotations sont éliminées et l'axe optique est recalé en utilisant la méthode que nous avons présentée au §2.1.1.1 consistant à transformer au préalable l'image courante et l'image précédente.

Les données de position nécessaires pour cela sont ici fournies par le simulateur et elles sont exactes (pas de bruit simulé sur les capteurs). La commande de vitesse angulaire est comme précédemment une moyenne pondérée entre la direction du but et cette commande d'évitement d'obstacles. Par contre, la commande de vitesse linéaire est ici constante ce qui simplifie l'utilisation du flux optique. Ce contrôleur est présenté à la figure 4.2.

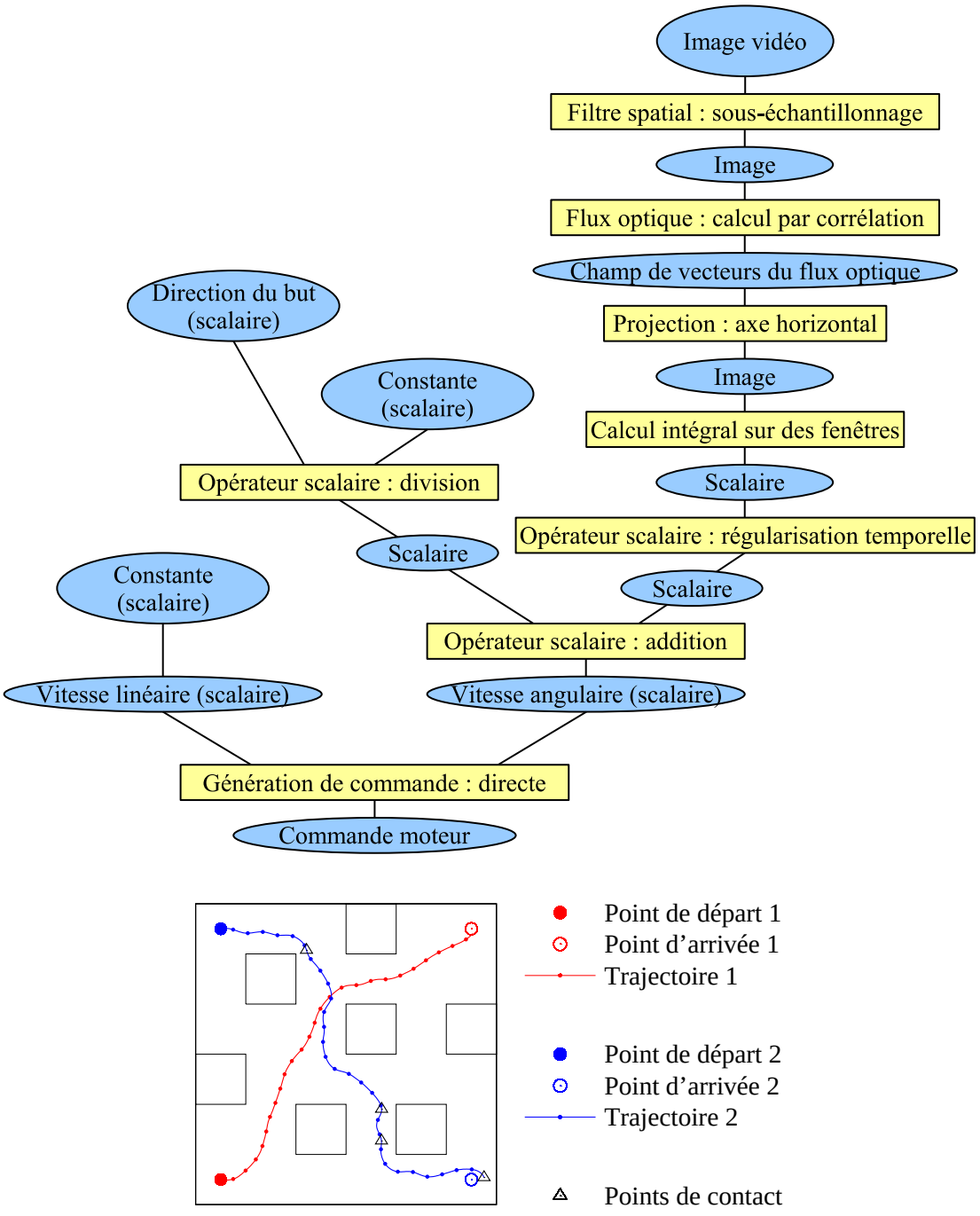


FIG. 4.2 : *Haut : Contrôleur manuel conçu pour diriger le robot dans l'environnement constitué de blocs texturés. Bas : Trajectoires obtenues avec ce contrôleur.*

Dans l'environnement plus réaliste contenant trois étagères, nous avons conçu un contrôleur basé sur le flux optique très similaire au précédent. Ses performances sont par contre moins bonnes ici car les étagères sont relativement peu texturées et donc plus difficiles à détecter par flux optique. La figure 4.3 présente les trajectoires obtenues. Le contrôleur n'est pas représenté car sa structure est identique au précédent, seuls les paramètres changent. La principale différence est que le coefficient associé aux fenêtres d'intégration de la projection du flux optique est nettement plus élevé. Cela permet d'utiliser malgré tout le flux optique, rendu moins dense par les difficultés de calcul dans les zones moins texturées. Cependant les virages sont plus brutaux et le déplacement moins lisse que précédemment.

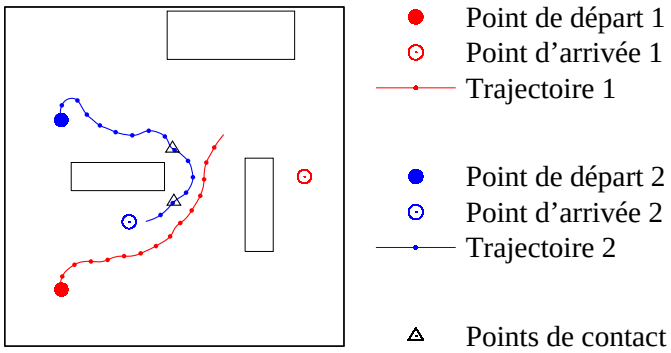


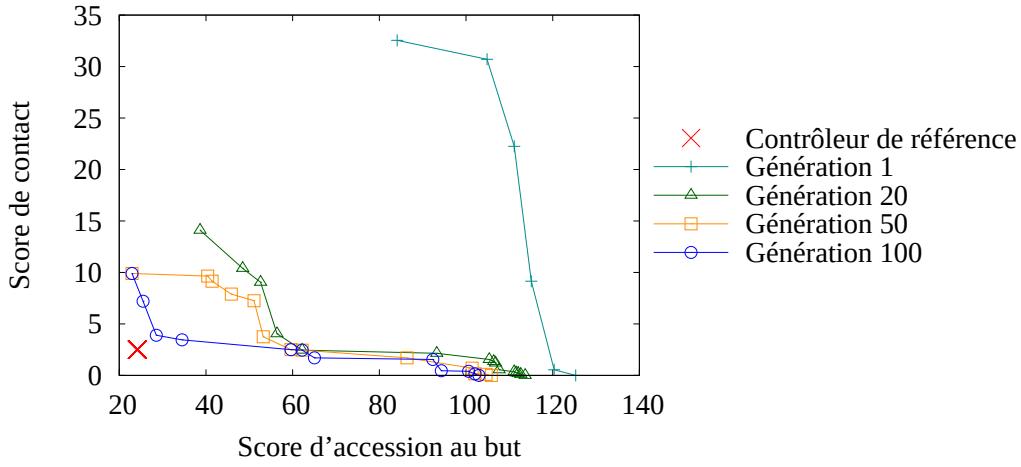
FIG. 4.3 : *Trajectoires obtenues avec le contrôleur manuel conçu pour diriger le robot dans l'environnement plus réaliste contenant trois étagères.*

Ces trois contrôleurs serviront de point de comparaison pour évaluer les performances des algorithmes évolués. Nous les réutiliserons donc dans toutes les expériences présentées dans la suite de ce chapitre.

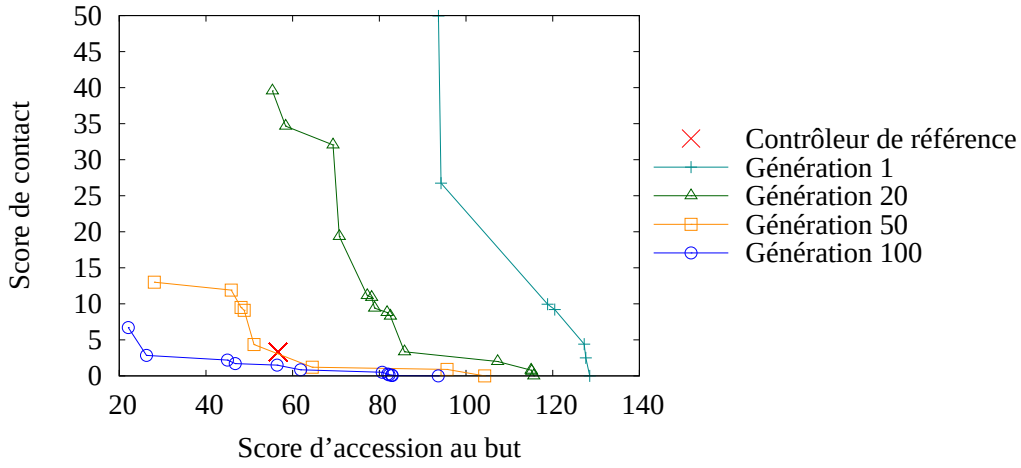
4.1.2 Déroulement du processus d'évolution

Le processus d'évolution se déroule ici en une seule phase en utilisant uniquement la fonction d'évaluation basée sur les performances objectives des algorithmes. Cette phase dure 100 générations et la population est découpée en 4 îlots de 100 individus. 40 000 évaluations sont donc effectuées pour chaque expérience. La figure 4.4 présente les performances des meilleurs individus au cours des générations lorsque l'évolution se déroule dans les différents environnement de simulation.

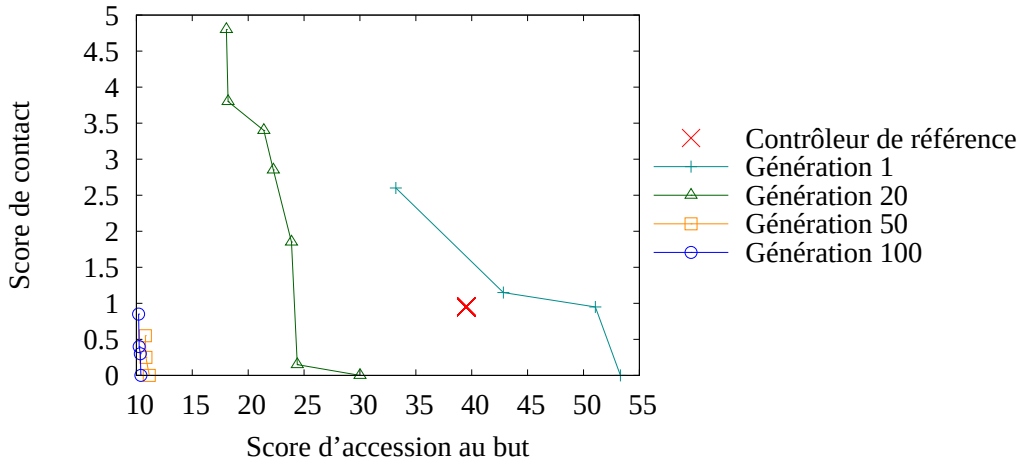
On constate qu'en termes de performances, les premières générations permettent rapidement d'améliorer les contrôleurs pour en obtenir de plus rapides et qui heurtent moins les obstacles. Pendant la seconde moitié de l'évolution, cette amélioration est plus lente. Au final, les meilleurs contrôleurs évolués au bout de 100 générations ne sont pas indiscutablement plus performants que notre contrôleur de référence, sauf dans le troisième environnement. Cela se comprend plus aisément en analysant les algorithmes utilisés par les contrôleurs évolués et les trajectoires générées.



(a) *Environnement composé de blocs non texturés*



(b) *Environnement composé de blocs texturés*



(c) *Environnement composé de trois étagères*

FIG. 4.4 : *Évolution au cours des générations des performances des meilleurs individus lors d'une évolution en une seule phase. Dans cette figure et les suivantes, les individus représentés sont ceux qui constituent le front de Pareto de l'ensemble de la population des quatre îlots.*

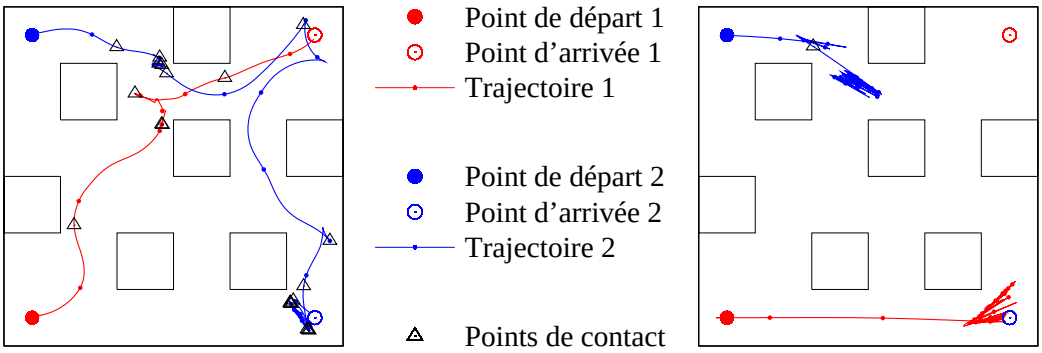
4.1.3 Présentation des contrôleurs évolués

En examinant les algorithmes des contrôleurs évolués, on constate tout d'abord qu'une grande partie d'entre eux est très difficilement interprétable car les primitives utilisées sont nombreuses et pas toujours cohérentes avec ce que l'on pourrait attendre pour effectuer de l'évitement d'obstacles. Ce phénomène est très largement lié aux problèmes de *bloating* qui sont très fréquents dans les applications de programmation génétique. Le *bloating* désigne une augmentation de la taille des programmes au cours des générations qui n'est pas liée à une augmentation de performance. Des méthodes ont été proposées pour limiter ce phénomène, nous reviendrons sur ce point au §4.5.2. Une autre explication de ce phénomène dans notre application est la génération de comportements chaotiques qui permettent au robot de ne pas rester bloqué contre des obstacles. Les contrôleurs exhibant ce type de comportement vont donc avoir tendance à obtenir de meilleurs résultats que les autres et à se propager dans la population, même si ce type de comportement ne confère pas en soi de capacité d'évitement d'obstacles. Nous éviterons dans ce chapitre de détailler ce type de contrôleur, nous présenterons uniquement ceux dont la structure reste partiellement interprétable.

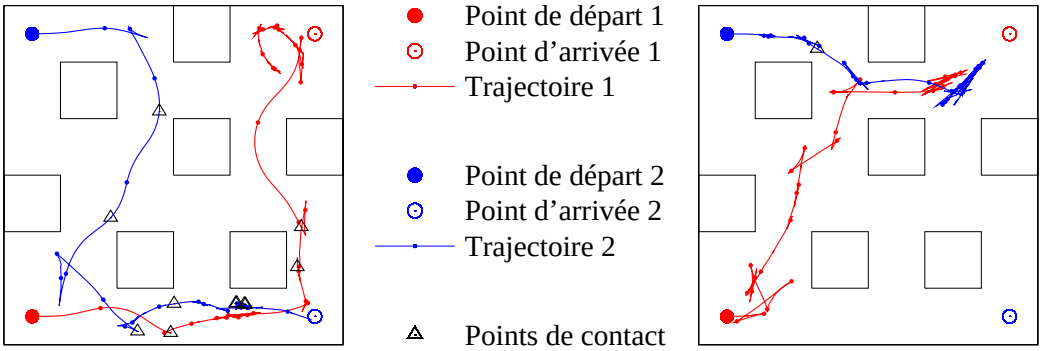
Parmi les contrôleurs évolués avec cette méthode, deux types de comportements se distinguent. Les premiers sont des contrôleurs très rapides qui vont utiliser des mouvements d'aller-retour ou des déplacements plus chaotiques pour éviter de rester bloqués contre des obstacles. Leur rapidité leur permet d'explorer une grande partie de l'environnement, et de fait, dans la majorité des cas, le point cible est atteint par hasard, même si certains contrôleurs font une utilisation pertinente de la direction du but et ainsi l'atteignent plus vite. Ces contrôleurs ne montrent aucun comportement d'évitement d'obstacles ce qui a pour conséquence un score de contact généralement plus élevé, on les retrouve donc sur la partie gauche des graphiques de la figure 4.4.

Le second type d'algorithmes présente au contraire de bonnes capacités de détection et d'évitement d'obstacles. Cependant ces algorithmes sont généralement plus lents et ont un comportement que l'on pourrait qualifier de "prudent" qui les empêche d'aller très loin dans l'environnement et d'atteindre le but. Ces algorithmes sont ceux que l'on retrouve sur la partie droite des graphiques de la figure 4.4. La figure 4.5 montre des exemples de trajectoires suivies par ces deux types de contrôleurs. Notons toutefois que dans certains cas l'évolution produit des contrôleurs rapides et évitant bien les obstacles. Ceux-ci prennent alors rapidement le dessus sur les autres et sont les seuls survivants à la fin du processus d'évolution (figures 4.4(c) et 4.5(c) à gauche).

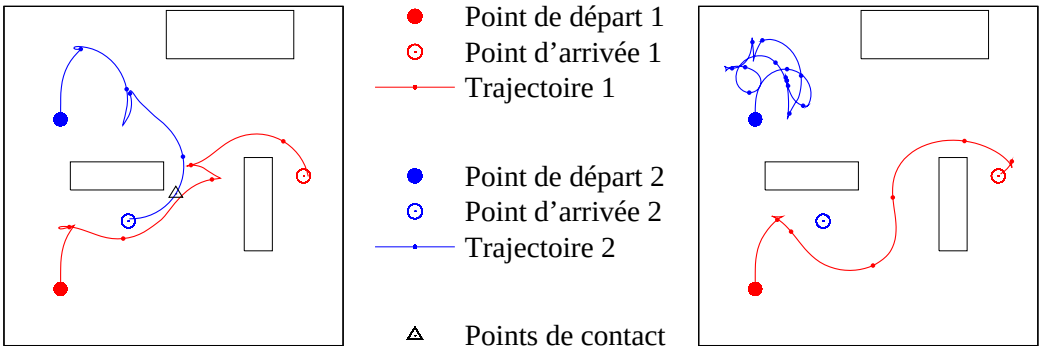
En examinant ces algorithmes évolués, on distingue des particularités quasi-systématiques dépendant de l'environnement de simulation utilisé durant l'évolution. Ainsi dans l'environnement constitué de blocs non texturés, la commande de vitesse angulaire se base généralement sur un calcul intégral effectué directement sur l'image non filtrée. En effet, les obstacles étant plus sombres que le sol, ce type de mesure peut donner une indication importante dans le cadre de l'évitement d'obstacles (figure 4.6). Dans l'environnement constitué de blocs texturés, les contrôleurs évolués utilisent généralement un filtre de Gabor ou un calcul de flux optique, qui sont tous deux bien adaptés à la détection d'obstacles texturés (figure 4.7). Dans l'environnement contenant trois étagères, les contrôleurs évolués utilisent soit une mesure directe sur l'image, soit un filtre de seuillage (figure 4.8).



(a) *Environnement composé de blocs non texturés*



(b) *Environnement composé de blocs texturés*



(c) *Environnement composé de trois étagères*

FIG. 4.5 : *Trajectoires suivies par des individus évolués lors d'une évolution en une seule phase. Certains contrôleurs sont très rapides mais évitent peu les obstacles (colonne de gauche), d'autres sont plus lents, évitent mieux les obstacles mais n'atteignent que rarement le but (colonne de droite). Le contrôleur représenté sur la figure (c) à gauche est un cas particulier où l'algorithme permet d'aller rapidement vers le but en évitant les obstacles.*

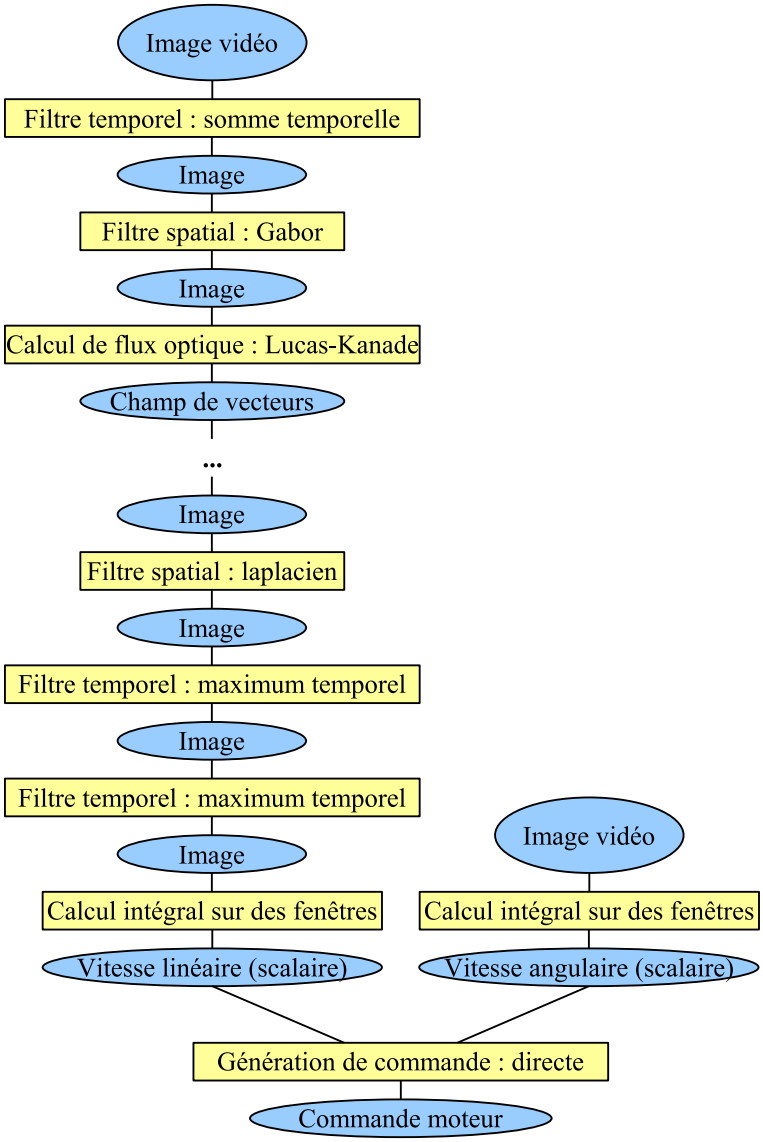


FIG. 4.6 : *Algorithme utilisé par un contrôleur évolué en une phase dans l'environnement constitué de blocs non texturés. La commande de vitesse linéaire souffre de* bloating *mais varie assez peu durant l'expérience. La commande de vitesse angulaire dépend d'une mesure effectuée directement sur l'image non filtrée. La trajectoire générée par cet algorithme est présentée sur la figure 4.5(a), colonne de gauche.*

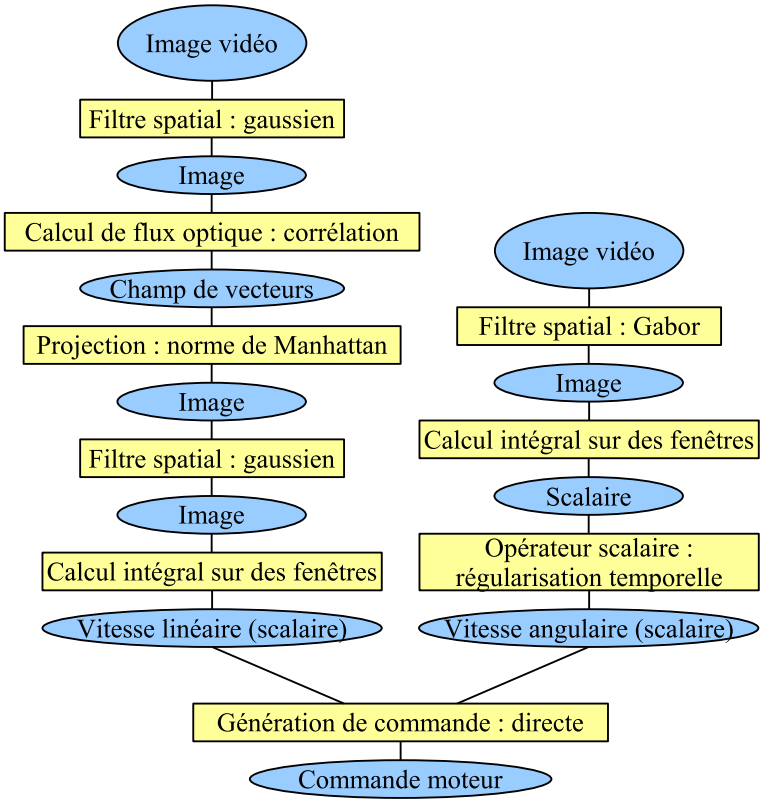


FIG. 4.7 : *Algorithme utilisé par un contrôleur évolué en une phase dans l'environnement constitué de blocs texturés. La commande de vitesse linéaire est basée sur un calcul de flux optique qui va permettre la détection des obstacles frontaux proches et générer dans ce cas une commande de vitesse négative (marche arrière). La commande de vitesse angulaire est basée sur un filtre de Gabor, bien adapté pour détecter les obstacles à une distance donnée. La trajectoire générée par ce contrôleur est présentée sur la figure 4.5(b), colonne de droite. Elle comporte peu de points de contact.*

Les enseignements importants de ces premières expériences sont donc qu'il y a une réelle adaptation des contrôleurs évolués à l'environnement visuel qui a été utilisé pour les générer. Par contre, cette adaptation n'est pas toujours suffisante pour produire des contrôleurs efficaces capables d'atteindre le but sans heurter d'obstacles. Dans la plupart des cas, ces contrôleurs restent moins performants que ceux qui ont été développés manuellement. D'une manière générale, ils développent des stratégies d'évitement d'obstacles originales mais n'arrivent pas à les combiner avec un déplacement vers le point cible.

4.2 Méthodes permettant de guider le processus d'évolution

Nous avons vu dans la section précédente que les contrôleurs évolués directement en une seule phase sont généralement peu efficaces par rapport aux contrôleurs conçus manuellement. Une des explications à ces performances médiocres est que l'espace d'état à explorer par le processus d'évolution (le nombre d'algorithmes possibles à construire avec notre grammaire) est

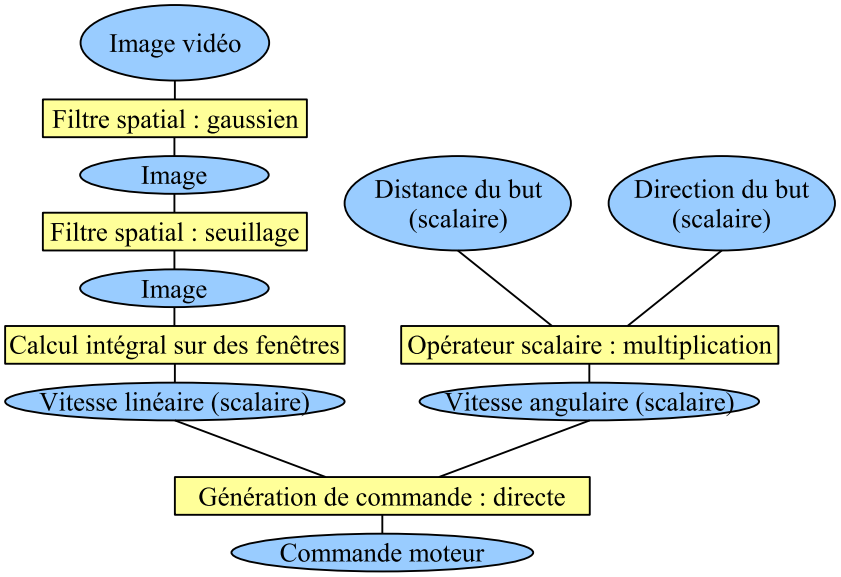


FIG. 4.8 : *Algorithme utilisé par un contrôleur évolué en une phase dans l'environnement constitué de trois étagères. La commande de vitesse linéaire est basée sur un filtre gaussien suivi d'un simple seuillage. Lorsqu'un obstacle proche est détecté par ce filtre, la commande de vitesse linéaire sera négative (marche arrière). La commande de vitesse angulaire est basée uniquement sur le produit de la distance du but avec la direction du but. Cela permet au robot de se diriger globalement vers le point cible. La trajectoire générée par ce contrôleur est présentée sur la figure 4.5(c) à gauche. On constate qu'elle est très efficace dans cet environnement, le robot arrivant rapidement au point cible sans vrai contact.*

immense en comparaison de la taille de la population utilisée (100 algorithmes dans notre cas). De plus, la fonction de fitness utilisée présente un "paysage" très accidenté, c'est-à-dire qu'une modification mineure d'un algorithme peut entraîner un changement drastique dans le résultat final. L'évolution risque donc de rester bloquée dans des minima locaux de cette fonction de fitness et de ne pas atteindre les solutions les plus intéressantes. Pour surmonter ce problème, nous proposons ici de guider artificiellement l'évolution vers les zones de l'espace d'état où il y a de bonnes chances de trouver des solutions intéressantes.

Les méthodes utilisables pour cela ont déjà été décrites au §3.3.3, nous allons ici principalement comparer les résultats obtenus avec ces différentes techniques. Les expériences présentées ici utilisent donc respectivement l'évolution en une phase déjà vue précédemment, une évolution incrémentale, une évolution guidée par *seeding* et notre méthode d'évolution en deux phases basée sur l'imitation. Notons que pour cette dernière, la grammaire a été légèrement modifiée pour supprimer l'opérateur de génération de commande basé sur des mouvements séquentiels (c'est-à-dire une suite de déplacements alternant translations et rotations sur place). En effet, celui-ci n'a pas vraiment de sens pour reproduire un déplacement enregistré qui est continu. Nous utilisons pour ces expériences l'environnement constitué de trois étagères.

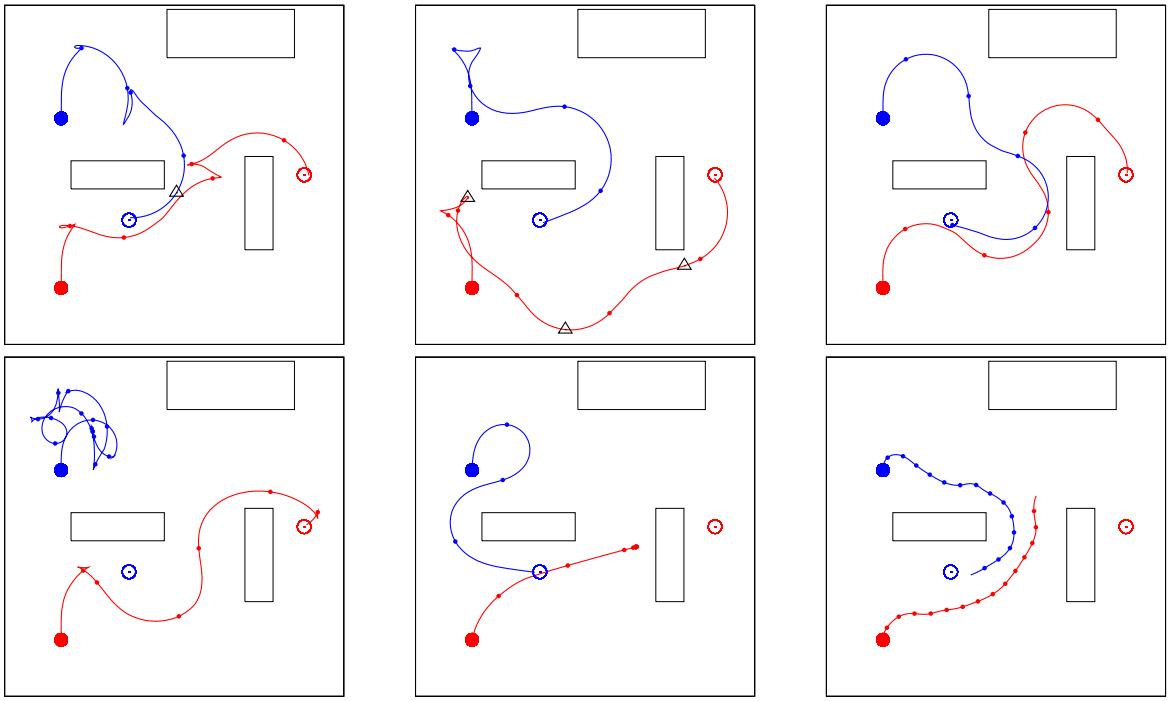
Nous avons réalisé pour chaque type d'évolution entre deux et quatre expériences. Le terme "expérience" désigne ici le processus d'évolution complet comprenant au total 40 000 évalua-

tions. En raison de la grande variabilité des résultats obtenus, nous avons choisi dans chaque cas de détailler un contrôleur issu de la meilleure expérience et un provenant de la moins bonne. Ce classement des expériences est réalisé en comparant les fronts de Pareto de la dernière génération. Ce choix reste donc objectif tant que ces fronts ne se recoupent pas, ce qui est généralement le cas. Pour une expérience donnée, le choix du contrôleur présenté est effectué parmi ceux qui font partie du front de Pareto de la dernière génération. Lorsque ce front contient plusieurs individus, nous choisissons celui dont le comportement nous semble le plus stable et le plus facilement interprétable. Ce dernier critère est donc plus subjectif. La figure 4.9 présente les trajectoires suivies par ces contrôleurs afin de mettre en évidence les particularités des algorithmes produits par les différents types d'évolution.

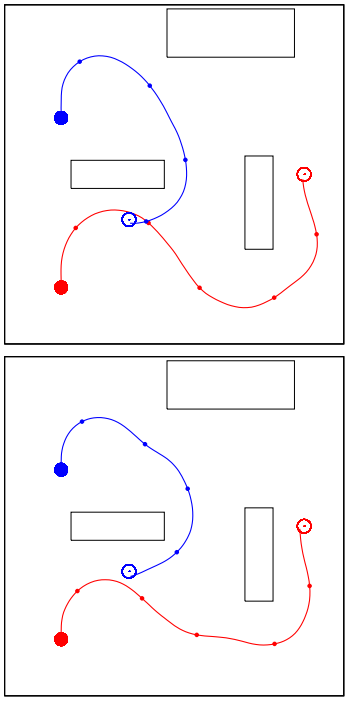
Les contrôleurs issus de l'évolution en une seule phase ont déjà été présentés au §4.1.3. Dans le cas le moins bon, les déplacements sont chaotiques et peu efficaces au final. Dans le meilleur cas, on obtient un déplacement assez rapide vers le but qui reste toutefois quelque peu saccadé à cause de plusieurs marches arrière pour éviter les obstacles. L'évolution basée sur le *seeding* produit dans le cas le moins bon un contrôleur quasi-identique à celui de référence utilisé pour amorcer l'évolution. Dans le cas le meilleur, le contrôleur évolué est capable d'atteindre le but dans les deux cas en évitant parfaitement les obstacles. Il semble qu'ici deux cas de figure peuvent se présenter : soit les individus issus du contrôleur de référence dominent rapidement le reste de la population et l'évolution ne trouvera pas de comportement différent à utiliser, soit d'autres solutions assez efficaces apparaissent rapidement et on peut obtenir des contrôleurs très efficaces au final. L'évolution incrémentale commence par produire des contrôleurs très simples dans le cas trivial où un seul obstacle est présent. Ces contrôleurs semblent par contre difficiles à adapter par la suite, et les résultats finaux sont généralement moins bons qu'avec les autres types d'évolution.

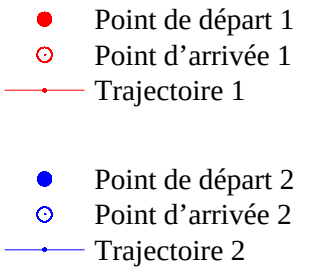
L'évolution en deux phases semble la plus stable de toutes dans le sens où les résultats obtenus sont très proches entre les différentes expériences. Il faudrait toutefois procéder à plus de tests pour vérifier cela objectivement. Les contrôleurs produits sont capables d'atteindre les deux points cibles sans heurter d'obstacle et très rapidement. De plus les trajectoires obtenues ici sont nettement plus lisses et directes que celles obtenues avec les autres méthodes. Cette propriété n'est pas du tout prise en compte par notre fonction de fitness mais peut avoir des conséquences intéressantes pour un robot réel comme une consommation d'énergie moindre et une plus grande stabilité en diminuant les risques de dérapage.

Celle-ci est également plus pratique à mettre en œuvre que les autres types d'évolution guidée. En effet, le *seeding* nécessite de concevoir au préalable un contrôleur de référence relativement efficace ce qui est loin d'être trivial. L'évolution incrémentale nécessite quant à elle de concevoir des problèmes de difficulté croissante, et le bon réglage de cette difficulté peut avoir des conséquences très importantes sur le résultat final comme nous l'avons vu. En comparaison, l'évolution en deux phases basée sur l'imitation nécessite seulement d'enregistrer une trajectoire au préalable ce qui reste relativement facile et rapide à réaliser. Nous pensons donc que ce type d'évolution pourrait facilement être réutilisé pour d'autres applications de robotique évolutionnaire.



(a) *Évolution en une phase* (b) *Évolution incrémentale* (c) *Évolution avec* seeding





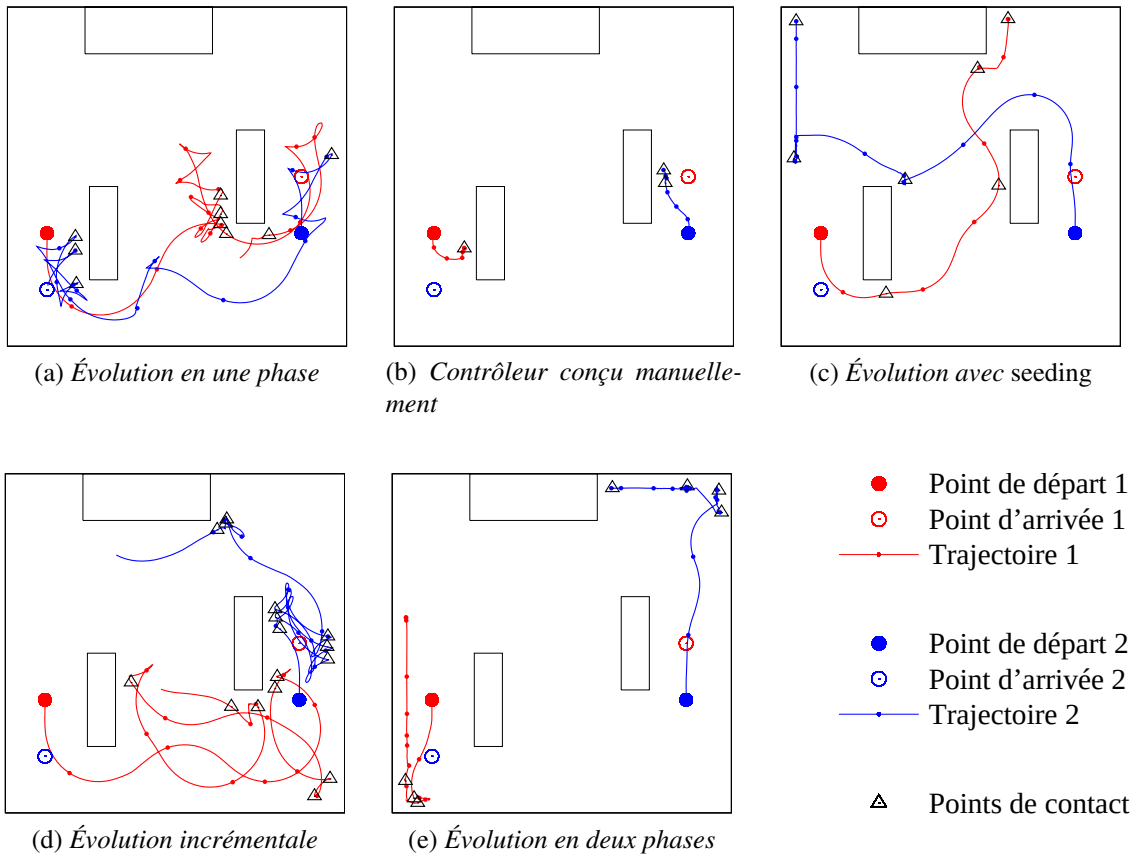
(d) *Évolution en deux phases*

FIG. 4.9 : *Trajectoires suivies par des individus évolués en utilisant différentes méthodes d'évolution. Nous représentons dans chaque cas les trajectoires de deux contrôleurs différents. Le premier est issu de la meilleure expérience et le second de la moins bonne.*

4.3 Capacités de généralisation des algorithmes

4.3.1 Test des algorithmes dans un environnement modifié

Nous allons maintenant tester les algorithmes présentés précédemment dans un environnement légèrement différent pour avoir une idée de leurs capacités de généralisation. Nous avons créé un environnement de simulation très similaire au précédent contenant trois étagères. Les obstacles sont les mêmes et l'éclairage est identique. La différence vient du fait que les obstacles ont été déplacés, ainsi que les points de départ et d'arrivée. Les contrôleurs qui sont réellement capables d'atteindre un point cible en évitant les obstacles de manière générique doivent présenter le même comportement ici que dans l'environnement utilisé lors de l'évolution. La figure 4.10 présente les trajectoires suivies par les algorithmes présentés précédemment dans cet environnement modifié.



(a) *Évolution en une phase* (b) *Contrôleur conçu manuellement* (c) *Évolution avec* seeding

(d) *Évolution incrémentale* (e) *Évolution en deux phases*

FIG. 4.10 : *Trajectoires suivies par les contrôleurs évolués dans un environnement où les obstacles et les points de départ et d'arrivée ont été déplacés.*

On constate que le comportement généré par tous ces contrôleurs est inefficace lorsqu'on déplace les obstacles et les points de départ et d'arrivée. Certains restent moins bloqués contre les obstacles mais aucun ne fait réellement preuve de capacités d'évitement d'obstacles. Cela signifie que tous les algorithmes évolués jusqu'à maintenant n'utilisent la vision que pour obtenir des repères leur permettant de suivre une trajectoire donnée. Il s'agit du phénomène bien

connu de surapprentissage, c'est à dire que les algorithmes évolués ont trop bien appris à résoudre le problème posé durant l'évolution et sont incapables de généraliser à d'autres instances du même problème. Nous allons maintenant voir comment on peut résoudre ces problèmes de surapprentissage, tout d'abord en augmentant le nombre de parcours utilisés durant l'évolution, puis en modifiant la structure des algorithmes.

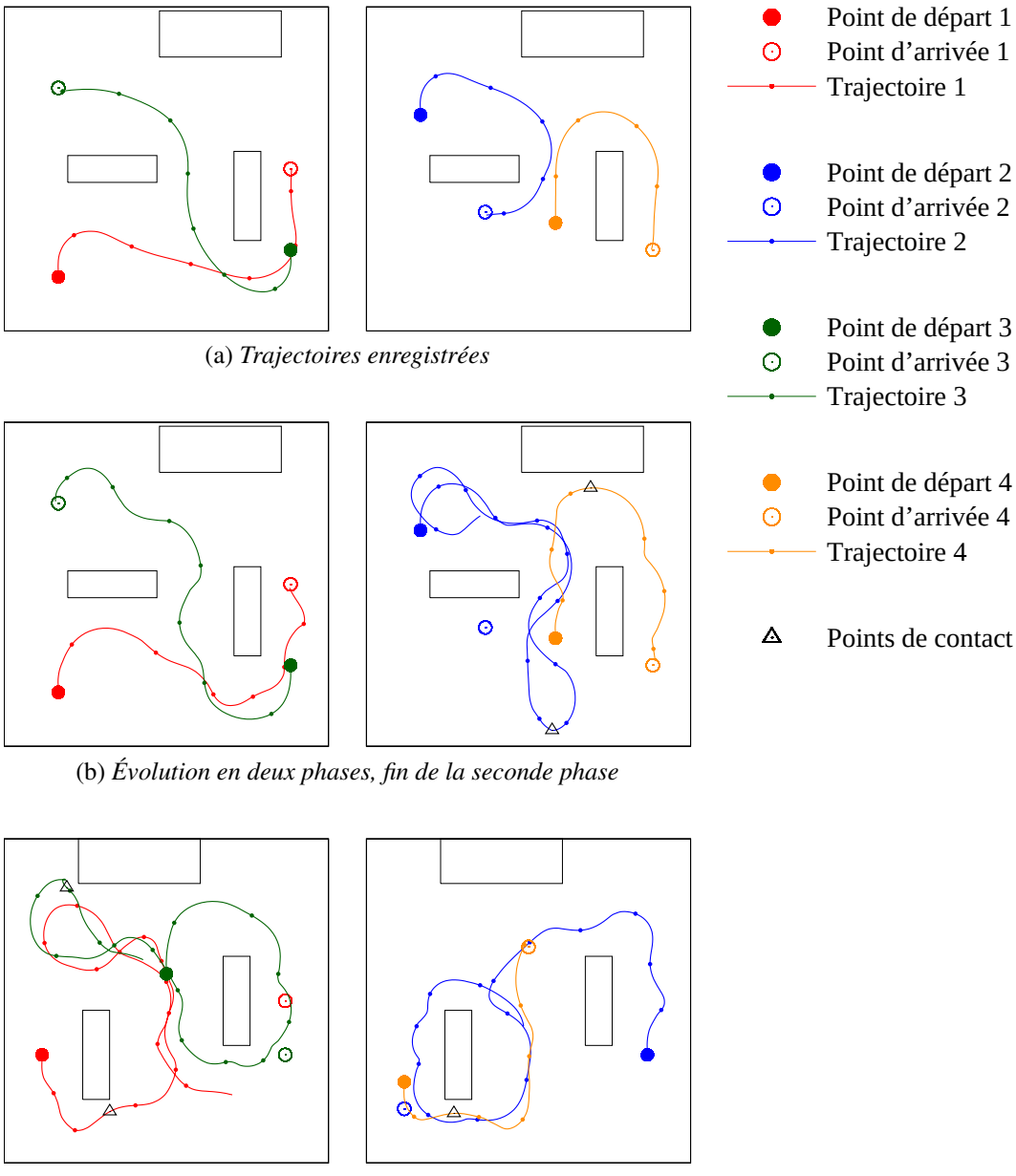
4.3.2 Augmentation du nombre de parcours

Dans cette expérience, nous allons tout simplement modifier la fonction d'évaluation afin de réaliser quatre parcours différents au lieu de deux lors du test de chaque algorithme. Le processus d'évolution étant dans ce cas deux fois plus long, nous mènerons cette expérience uniquement en utilisant l'évolution en deux phases présentée précédemment. La figure 4.11 représente les trajectoires obtenues avec un algorithme évolué de cette manière, dans l'environnement utilisé pour l'évolution et l'environnement de test dans lequel nous avons également ajouté deux parcours.

Dans l'environnement utilisé durant l'évolution, on constate que les trajectoires générées par le contrôleur évolué sont moins lisses que précédemment. Le robot passe ici très près des obstacles à deux reprises et dans une configuration il n'atteint pas le point cible. Cependant il est plus intéressant ici d'observer les trajectoires générées dans l'environnement de test. Celles-ci ne sont pas parfaites, le robot n'atteignant le point cible que dans une configuration sur les quatre étudiées. Cependant les contacts avec les obstacles sont beaucoup moins nombreux que dans les expériences présentées précédemment, ce qui montre que les capacités acquises par le robot sont ici plus génériques et ne sont pas dues à des phénomènes de surapprentissage. Le principal problème qui se pose ici est que les algorithmes évolués arrivent difficilement à effectuer un compromis entre l'évitement d'obstacles et le déplacement vers le point cible. Celui qui est présenté ici n'utilise pas du tout l'information de direction du but et ne peut donc que se déplacer aléatoirement dans l'environnement, ne trouvant le point cible que par chance. Pour résoudre ce problème, nous proposons une nouvelle structure algorithmique qui va faciliter ce compromis entre évitement d'obstacles et déplacement vers le but.

4.3.3 Utilisation d'une nouvelle architecture algorithmique

La structure algorithmique proposée ici a pour but de diminuer les problèmes de surapprentissage constatés précédemment et de faciliter le compromis entre les comportements d'évitement d'obstacles et de déplacement vers le but. Cette structure consiste à découper les algorithmes de vision générés en deux parties distinctes. Un premier algorithme va uniquement servir à déterminer s'il y a un obstacle proche du robot ou non. Il renvoie donc seulement une valeur booléenne. S'il n'y a pas d'obstacle à proximité, le robot va simplement se diriger vers le but. La commande utilisée pour cela est fixe, seule la vitesse à laquelle il se déplace dans ce cas peut être paramétrée par l'évolution. S'il y a un obstacle à proximité, on utilise un deuxième algorithme pour générer la commande permettant d'éviter cet obstacle. Ce deuxième algorithme peut pour cela réutiliser l'image filtrée générée par le premier algorithme. Cette structure peut être qualifiée de restreinte car le compromis entre évitement d'obstacles et déplacement vers le but est ici forcé. Le principe général est résumé sur la figure 4.12.



(a) *Trajectoires enregistrées*

(b) *Évolution en deux phases, fin de la seconde phase*

(c) *Contrôleur évolué testé dans l'environnement modifié*

FIG. 4.11 : *Trajectoires suivies par un contrôleur évolué en deux phases en utilisant quatre parcours différents lors de l'évaluation.*

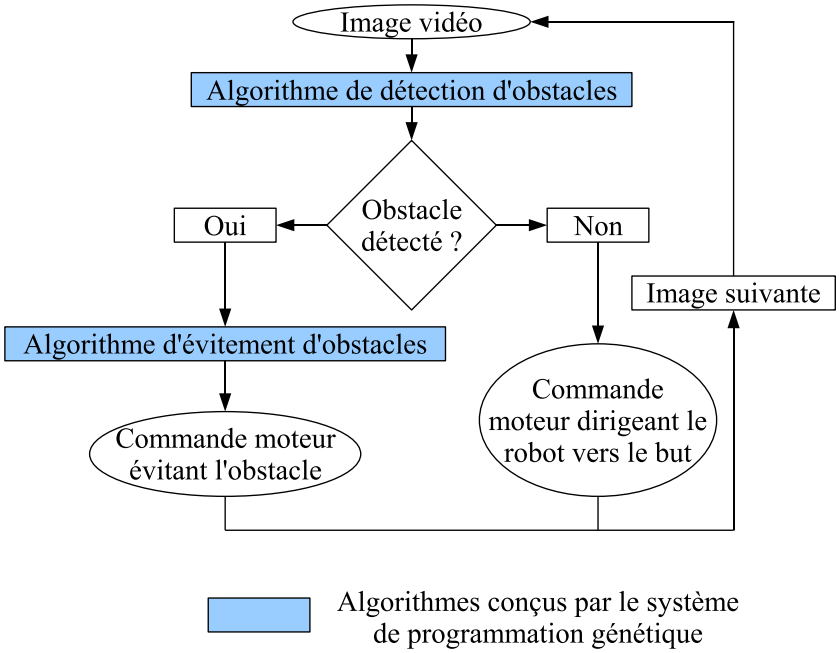


FIG. 4.12 : *Illustration de la structure algorithmique utilisée pour faciliter le compromis entre évitement d'obstacles et déplacement vers le but.*

La grammaire utilisée pour générer ces deux algorithmes est quelque peu différente de celle qui avait été présentée au §2.3.3.2. Pour le premier algorithme servant à détecter les obstacles, elle a été simplifiée pour n'utiliser qu'une seule image. Pour le second algorithme d'évitement d'obstacles, elle a été modifiée pour pouvoir réutiliser l'image filtrée issue du premier algorithme. L'opérateur *if-then-else* a de plus été supprimé pour simplifier la structure. Le tableau 4.1 présente ces deux grammaires modifiées.

Nous présentons ici les résultats obtenus en faisant évoluer des algorithmes avec cette nouvelle structure. Afin de limiter au maximum les phénomènes de surapprentissage, nous utilisons à nouveau quatre parcours pour chaque expérience au lieu de deux. La figure 4.13 présente les trajectoires obtenues avec un contrôleur évolué, à la fois dans l'environnement utilisé lors de l'évolution et dans l'environnement modifié.

Dans l'environnement utilisé lors de l'évolution, les résultats sont similaires à ce que l'on a obtenu précédemment avec des contrôleurs à structure libre. Le robot atteint ici le point cible dans les quatre configurations, avec un seul vrai contact contre un obstacle. Les différences apparaissent lorsqu'on examine le comportement en généralisation, c'est-à-dire dans l'environnement modifié. Le robot atteint le point cible dans deux configurations sur quatre et s'en approche de très près dans les deux autres. Le comportement en évitement d'obstacles n'est pas encore parfait avec un contact contre un obstacle, mais on peut dire que l'utilisation de cette structure restreinte a amélioré le compromis entre le déplacement vers le but et l'évitement d'obstacles.

Un point important que nous n'avons que rapidement abordé jusqu'ici est celui de la sélection des "meilleurs" algorithmes à la fin de l'évolution. Ce choix est à l'heure actuelle réalisé manuellement mais cela demande un certain temps et ne permet pas toujours de sélectionner les meilleurs individus. En particulier, lorsqu'on cherche des individus avec de bonnes capacités

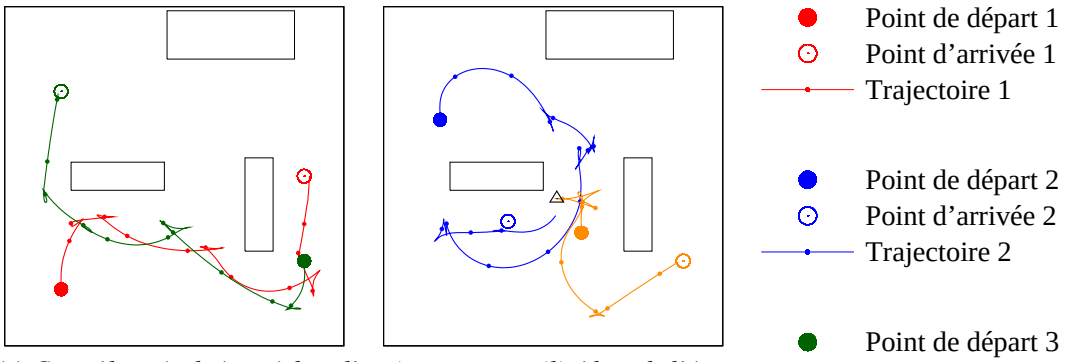
[1,0]	DÉPART	→	BOOLÉEN
[0,5]	BOOLÉEN	→	non (BOOLÉEN)
[0,5]	BOOLÉEN	→	seuil (RÉEL)
[0,3]	RÉEL	→	régularisation-temporelle (RÉEL)
[0,7]	RÉEL	→	calcul-intégral-fenêtres (LISTE-FENÊTRES, IMAGE)
[0,3]	LISTE-FENÊTRES	→	fenêtre
[0,7]	LISTE-FENÊTRES	→	LISTE-FENÊTRES, fenêtre
[0,3]	IMAGE	→	image-vidéo
[0,4]	IMAGE	→	FILTRE-SPATIAL (IMAGE)
[0,15]	IMAGE	→	PROJECTION (FLUX-OPTIQUE)
[0,15]	IMAGE	→	FILTRE-TEMPOREL (IMAGE)

(a) *Grammaire utilisée pour le premier algorithme de détection d'obstacles*

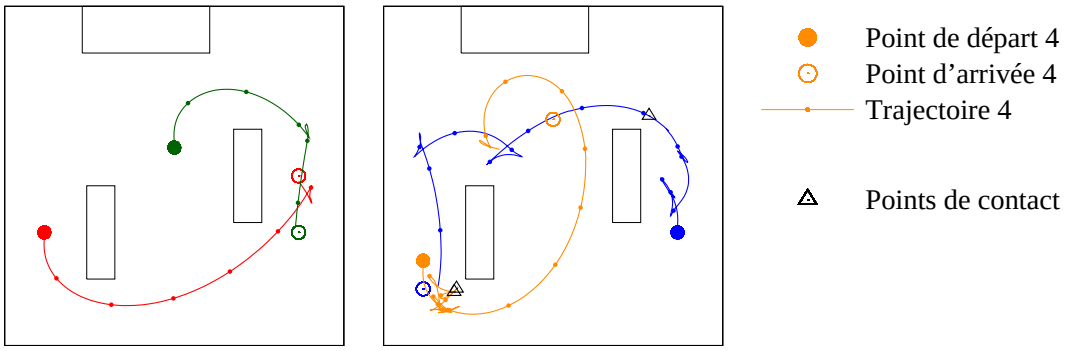
[1,0]	DÉPART	→	COMMANDE
[1,0]	COMMANDE	→	génération-directe (RÉEL, RÉEL)
[0,15]	RÉEL	→	constante
[0,075]	RÉEL	→	addition (RÉEL, RÉEL)
[0,075]	RÉEL	→	soustraction (RÉEL, RÉEL)
[0,05]	RÉEL	→	multiplication (RÉEL, RÉEL)
[0,05]	RÉEL	→	division (RÉEL, RÉEL)
[0,1]	RÉEL	→	régularisation-temporelle (RÉEL)
[0,5]	RÉEL	→	calcul-intégral-fenêtres (LISTE-FENÊTRES, IMAGE)
[0,3]	LISTE-FENÊTRES	→	fenêtre
[0,7]	LISTE-FENÊTRES	→	LISTE-FENÊTRES, fenêtre
[0,3]	IMAGE	→	image-vidéo
[0,3]	IMAGE	→	image-préfiltrée
[0,25]	IMAGE	→	FILTRE-SPATIAL (IMAGE)
[0,1]	IMAGE	→	PROJECTION (FLUX-OPTIQUE)
[0,05]	IMAGE	→	FILTRE-TEMPOREL (IMAGE)

(b) *Grammaire utilisée pour le second algorithme d'évitement d'obstacles*

TAB. 4.1 : *Grammaires utilisées pour la génération des algorithmes à structure restreinte. Les parties correspondant aux nœuds non-terminaux* FILTRE-SPATIAL, FILTRE-TEMPOREL, FLUX-OPTIQUE *et* PROJECTION *sont identiques à la grammaire présentée au §2.3.3.2 et ne sont donc pas représentées ici.*



(a) *Contrôleur évolué testé dans l'environnement utilisé lors de l'évolution*



(b) *Contrôleur évolué testé dans l'environnement modifié*

FIG. 4.13 : *Trajectoires suivies par un contrôleur évolué en deux phases avec une structure restreinte en utilisant quatre parcours différents lors de l'évaluation.*

de généralisation, on constate que ceux-ci se retrouvent rarement sur le front de Pareto à la fin de l'évolution. C'est assez logique étant donné que les individus sur le front de Pareto sont les plus susceptibles d'avoir surappris les trajectoires utilisées pendant l'évolution. De ce fait, dans ces deux dernières expériences nous avons sélectionné les individus en évaluant l'intégralité de la population finale dans l'environnement de test. L'inconvénient est que cela introduit un autre risque, qui est de sélectionner des individus qui seraient adaptés par hasard à l'environnement de test. Notons toutefois que ce dernier risque est lié uniquement à la sélection finale des algorithmes. Il n'est donc pas comparable avec les problèmes de surapprentissage d'une trajectoire constatés précédemment qui sont causés par le cumul d'un biais de sélection des algorithmes sur l'ensemble de l'évolution. Une étude statistique plus poussée serait nécessaire afin de mettre en évidence le risque réel lié à ce biais de sélection finale.

On peut également noter que cette sélection finale ne répond pas à la question "quand arrêter l'évolution pour éviter les problèmes de surapprentissage ?". Il a été proposé pour cela d'utiliser une méthodologie basée sur trois ensembles de données différents [Gagné 06], technique largement utilisée par les méthodes d'apprentissage statistique. Cela consiste à évaluer le front de Pareto à la fin de chaque génération sur un jeu de données de validation, et d'arrêter l'évolution lorsque les performances sur ce jeu de données ne progressent plus. Et puisque l'évolution a été biaisée par ces données de validation, on utilise un troisième jeu de données pour évaluer

objectivement les performances des algorithmes finaux. Cependant comme on l'a vu, les algorithmes présentant les meilleures capacités de généralisation se trouvent rarement sur le front de Pareto. Il faudrait donc évaluer l'ensemble de la population à chaque génération sur le jeu de données de validation, mais cela doublerait le temps nécessaire à l'évolution. Il s'agit donc d'un problème encore ouvert qu'il serait intéressant d'étudier plus en détail à l'avenir.

4.4 Validation en environnement réel

Dans cette section, nous allons présenter une expérience permettant de valider une partie des techniques présentées précédemment pour une utilisation en environnement réel. En raison du temps nécessaire pour réaliser l'évaluation des différents algorithmes, nous avons choisi de ne pas procéder à une évolution *online*. En effet, celle-ci nécessiterait de pouvoir repositionner le robot précisément après chaque expérience, de pouvoir le recharger automatiquement et régulièrement, et de pouvoir le laisser se déplacer plusieurs jours durant dans le laboratoire. Ces aspects étant trop contraignants, nous avons préféré continuer à procéder à une évolution *offline*. Celle-ci s'effectue en une seule phase avec la fonction d'évaluation basée sur l'imitation. Cela nous permet donc également de valider cette technique pour la production de contrôleurs adaptés à un environnement réel.

Le but est ici de concevoir un algorithme permettant au robot de se centrer dans un couloir et de se déplacer dans celui-ci sans heurter les murs. Pour cela, nous enregistrons une vingtaine de courtes séquences (2 ou 3 secondes chacune) où nous guidons le robot pour l'éloigner du mur et le centrer dans le couloir (figure 4.14). Les algorithmes sont évalués par la fonction de fitness basée sur l'imitation décrite précédemment. Le but est donc qu'ils reproduisent au mieux les commandes enregistrées manuellement. Il n'y a pas ici de notion de point de départ ou de point cible puisque seule la phase d'imitation est utilisée. Nous souhaitons que les algorithmes évolués généralisent ce comportement de centrage pour que le robot puisse évoluer dans le couloir sans but précis et sans heurter les murs. Il s'agit donc ici d'un comportement d'exploration plus que de déplacement ciblé.

L'évolution dure ici 100 générations avec uniquement la fonction d'évaluation basée sur l'imitation. Nous montrons sur la figure 4.15 les résultats obtenus par les algorithmes au cours des générations. La commande de vitesse linéaire est en fait triviale à apprendre ici car dans les séquences d'apprentissage, celle-ci est presque toujours au maximum, soit 30 cm/s. Seule la commande de vitesse angulaire est réellement intéressante à étudier ici. Nous rappelons que l'erreur moyenne indiquée correspond en réalité à la racine carrée de la somme des carrés des erreurs. Celle-ci passe d'environ 15°/s dans les premières générations à 10°/s dans les dernières. Cette différence peut sembler minime mais elle correspond à un changement important dans le comportement des contrôleurs correspondants.

Nous présentons sur la figure 4.16 un exemple d'algorithme évolué. La chaîne de filtrage n'est pas représentée en entier car elle contient au total 29 filtres. Ce nombre est principalement dû au problème de *bloating* que nous avons abordé précédemment, mais il s'agit également d'une solution trouvée par le processus d'évolution pour contourner une limitation de notre système. En effet, cet algorithme se base sur un filtre de Sobel pour détecter la limite entre le sol et les murs. Le résultat de celui-ci est un bord fin qu'il est difficile de détecter avec notre opérateur de calcul intégral sur des zones larges de l'image. Une grande partie des filtres présents

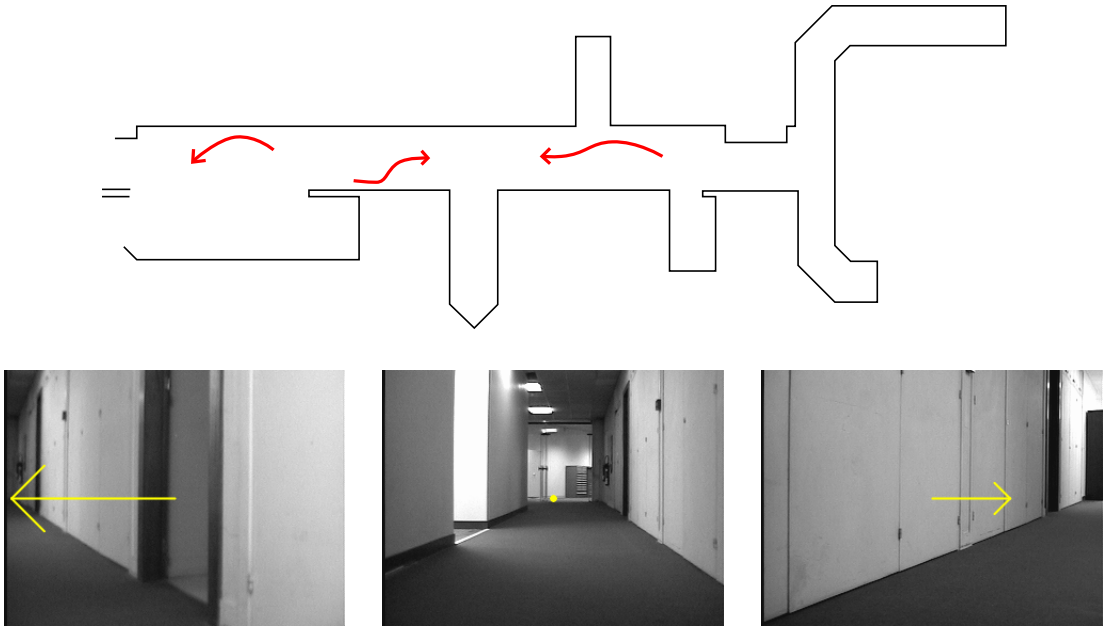


FIG. 4.14 : *Haut : Exemples de trajectoires utilisées pour l'apprentissage. Ne disposant pas de système de positionnement absolu, nous avons dessiné les trajectoires à la main dans cette figure et les suivantes. Celles-ci sont donc imprécises et ne doivent pas être interprétées comme une représentation exacte du déplacement. Bas : Exemples d'images et de commandes enregistrées. Les flèches jaunes représentent la commande de vitesse angulaire, comprise entre -30°/s (maximum à droite) et +30°/s (maximum à gauche).*

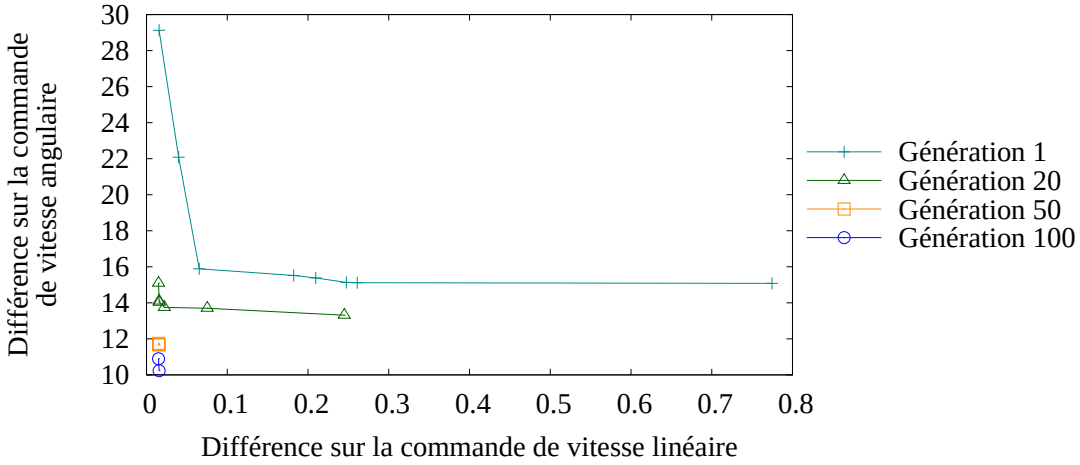


FIG. 4.15 : *Évolution au cours des générations des performances des meilleurs individus en terme d'imitation des comportements enregistrés*

ici sert en fait à élargir ce bord pour faciliter sa détection. D'autres opérateurs non implémentés ici auraient bien entendu été mieux adaptés pour cela, comme un filtre de dilatation par exemple, mais le fait que l'évolution ait réussi à contourner cette limitation est un enseignement intéressant en soi et une bonne indication de la capacité d'adaptation du système.

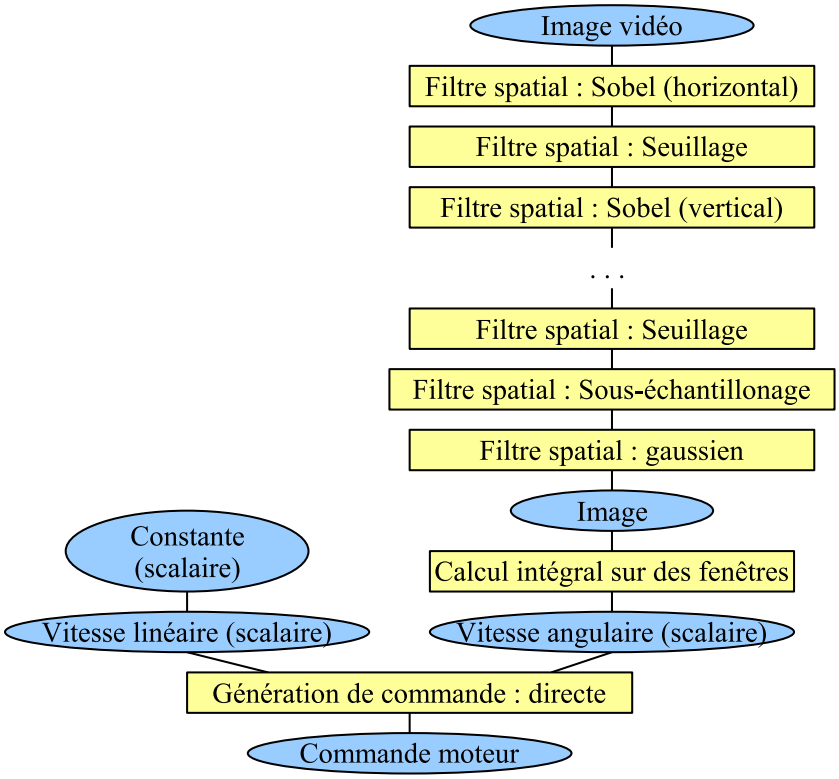


FIG. 4.16 : *Exemple d'algorithme évolué*

La figure 4.17 illustre la méthode utilisée par cet algorithme pour calculer la commande moteur. La flèche rouge indique la commande générée par l'algorithme évolué, la flèche jaune est la commande enregistrée. La chaîne de filtrage permet de mettre en évidence la limite entre le sol et le mur ainsi que la zone plus contrastée au bout du couloir. Dans l'image filtrée, le mur apparaît ainsi complètement blanc tandis que cette frontière est plus sombre. C'est cette différence qui est utilisée pour produire la commande qui va éloigner le robot du mur (la commande dépend de la différence entre la valeur moyenne des pixels dans chacune des fenêtres encadrées en rouge).

Pour tester la robustesse et les capacités de généralisation des algorithmes évolués, nous plaçons le robot à différents endroits avec un angle de 30° environ avec la direction du couloir. Nous le laissons ensuite se déplacer, guidé uniquement par l'algorithme évolué. Ce test a été réalisé dix fois avec différents points de départ. Le robot a atteint le bout du couloir sans heurter d'obstacle dans chacun des tests, sauf une fois où il a tourné dans une des ouvertures présentes sur le côté du couloir. Dans l'un des tests, il a même tourné dans un des plus petits couloirs situés à droite sur la carte. La figure 4.18 montre un exemple d'une telle trajectoire. Les deux dernières images correspondent à des situations qui n'étaient pas présentes dans la base d'apprentissage.

Nous avons également testé ce contrôleur dans un autre couloir dont l'apparence visuelle est différente. Le robot a atteint le bout du couloir sans heurter les murs dans une direction. Dans

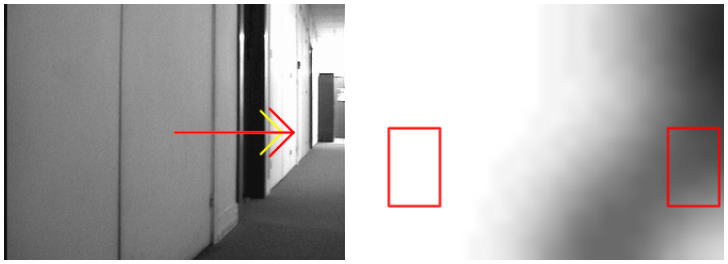


FIG. 4.17 : *Commande générée par l'algorithme évolué sur une image de la base d'apprentissage (à gauche). La même image transformée par la chaîne de filtrage (à droite).*

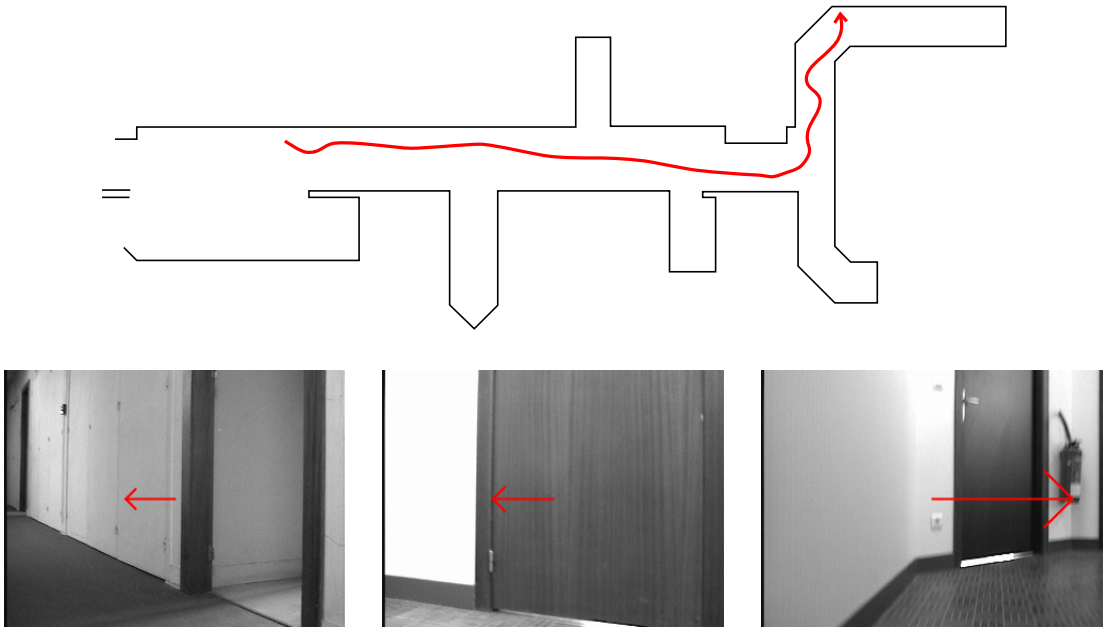


FIG. 4.18 : *Trajectoire suivie par le robot guidé par un algorithme évolué (en haut). Images enregistrées pendant ce déplacement et commandes associées (en bas).*

le sens inverse, il a heurté le mur en deux points comme indiqué sur la figure 4.19. L'image du milieu et celle de droite correspondent aux deux obstacles que le robot n'a pas évités. Cependant les résultats sont encourageants, ce couloir étant très différent visuellement de celui où la base d'apprentissage a été enregistrée.

Cette expérience montre que notre système peut produire des algorithmes adaptés au contexte visuel pour piloter un robot dans un environnement intérieur. Cependant, le robot n'a ici pas d'autre but que de se déplacer dans l'environnement sans heurter d'obstacles. Pour une application réelle, il faudrait qu'il soit également capable de se déplacer vers un lieu donné dans cet environnement. Comme nous l'avons montré au §4.3, ce problème de compromis entre plusieurs tâches n'est pas forcément trivial à résoudre. Nous allons présenter dans la section suivante plusieurs pistes qu'il serait intéressant d'explorer pour cela.

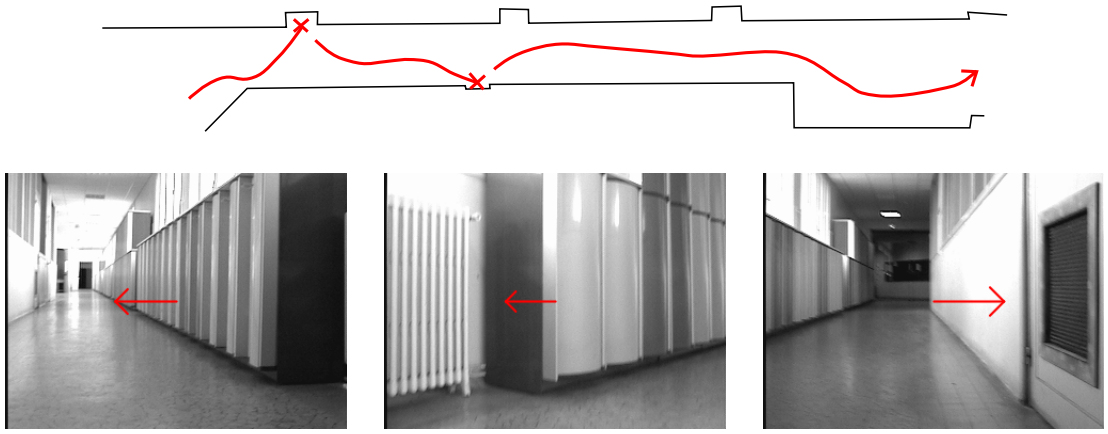


FIG. 4.19 : *Trajectoire suivie par le robot guidé par un algorithme évolué dans un couloir visuellement différent de celui présenté précédemment (en haut). Images enregistrées pendant ce déplacement et commandes associées (en bas).*

4.5 Discussion et autres pistes d'expérimentation

Nous allons résumer dans cette section les principaux avantages et inconvénients de notre système au vu des résultats présentés. Nous présenterons ensuite quelques autres pistes de recherche pour améliorer ce système. Certaines de ces pistes ont été partiellement explorées et abandonnées car les résultats n'étaient pas satisfaisants. Nous essaierons dans ce cas d'expliquer pourquoi et de tirer les enseignements de ces expériences. Dans d'autres cas, il s'agit de pistes que nous pensons intéressantes à explorer dans le futur.

4.5.1 Bilan des expériences présentées

Les expériences présentées dans ce chapitre nous ont tout d'abord permis de valider plusieurs hypothèses formulées dans cette thèse. Nous avons ainsi montré qu'il est possible d'apprendre automatiquement des algorithmes efficaces en considérant le problème de la vision de manière globale. Notre système permet en effet d'apprendre l'algorithme dans son ensemble, depuis l'image fournie par la caméra jusqu'à la commande moteur envoyée au robot. Nous avons également montré qu'il y a une réelle adaptation des algorithmes évolués à l'environnement visuel du robot durant cette phase d'apprentissage.

Un autre enseignement important est que l'utilisation de méthodes pour guider l'évolution permet d'accélérer celle-ci et d'améliorer les performances des algorithmes évolués. Pour les expériences où la taille de la population est restreinte par la durée des évaluations, ces méthodes d'évolution guidée s'avèrent être un atout non négligeable. Nous avons également montré que l'évolution en deux phases basée sur l'imitation d'un comportement pré-enregistré s'avère au moins aussi performante que les autres techniques d'évolution guidée. Elle est également plus facile à mettre en œuvre pour la plupart des applications de robotique.

L'expérience suivante a montré qu'il est possible d'améliorer les capacités de généralisation des algorithmes et le compromis entre évitement d'obstacles et déplacement vers le but en

utilisant une grammaire particulière et une structure restreinte. Le bilan de cette expérience reste toutefois plus mitigé, les performances obtenues au final en généralisation restant insuffisantes pour une utilisation sur un robot réel. Nous proposerons dans les sections suivantes plusieurs idées pour remédier à ce problème.

Pour finir, nous avons validé une partie de ces techniques sur un robot réel. Dans cette dernière expérience, il est surtout intéressant de constater que l'évolution basée sur l'imitation peut à elle seule créer des contrôleurs efficaces et présentant de bonnes capacités de généralisation. De ce point de vue, il semble que paradoxalement l'environnement réel présente moins de difficultés que les environnements de simulation utilisés. Il est toutefois clair que le problème posé est à la base plus simple. Il n'y a ici pas de compromis à effectuer entre évitement d'obstacles et déplacement vers le but, et les seuls obstacles présents sont les murs de chaque côté du couloir.

Un point que nous n'avons pas abordé est celui de la transplantation de contrôleurs évolués en simulation sur un robot réel. Intuitivement, l'idée consistant à profiter de la souplesse de la simulation pour faire évoluer des algorithmes efficaces et les utiliser ensuite sur un robot réel semble en effet prometteuse. Il s'agit néanmoins d'un problème assez complexe et récurrent en robotique évolutionnaire [Walker 03]. Pour qu'un contrôleur puisse être réutilisable dans un environnement réel, il faut en effet que les différences entre la simulation et la réalité soient minimales. Jakobi a ainsi montré qu'il est possible, en ajoutant une quantité réaliste de bruit aux capteurs et actionneurs du simulateur, d'utiliser des contrôleurs évolués en simulation sur un robot réel [Jakobi 95]. Cependant son système n'utilise que des capteurs infrarouges, qui ne sont déjà pas triviaux à modéliser. La modélisation d'une caméra suffisamment réaliste ne consiste pas simplement à ajouter du bruit sur l'image mais également à prendre en compte des phénomènes tels que la surexposition ou les variations de mise au point. Il faudrait de plus construire un environnement de simulation réaliste et similaire au milieu réel ciblé. La résolution de tous ces problèmes, qui n'a à notre connaissance pas encore été tentée, pourrait tout à fait constituer un sujet de thèse à part entière. Ces raisons nous ont donc poussé à utiliser des images réelles et une évolution basée sur l'imitation pour faire évoluer des contrôleurs efficaces en environnement réel.

Un autre problème que nous avions abordé au chapitre 1 est celui de l'impossibilité d'effectuer une analyse statistique sur les résultats en raison de la durée des évaluations. Les expériences que nous avons présentées permettent en effet de dégager certaines tendances et d'analyser individuellement des contrôleurs évolués. Il existe toutefois une grande variabilité dans les résultats obtenus en lançant plusieurs fois une expérience identique, ce qui est assez logique si l'on considère la taille restreinte de la population utilisée par rapport au nombre d'algorithmes possibles à générer. Cette variabilité et ce manque d'analyse statistique constituent clairement un handicap majeur qui devra être résolu en priorité à l'avenir pour effectuer des interprétations objectives des résultats. Cela implique de diminuer considérablement les temps d'évaluation des algorithmes pour pouvoir exécuter quelques dizaines d'expériences dans des temps raisonnables. Plusieurs possibilités sont envisageables à l'heure actuelle pour cela :

- La première, qui sera évoquée plus en détail au §4.5.2, consiste à réduire les phénomènes de *bloating* ce qui aurait comme effet secondaire de diminuer le temps d'évaluation des algorithmes.
- La seconde possibilité est de comparer les différentes méthodes sur des instances de problèmes plus simples où les temps d'évaluation seraient réduits. Néanmoins tout laisse à penser que les avantages respectifs de ces différentes méthodes dépendent grandement du

problème étudié, et rien ne prouve que des résultats obtenus sur des instances simples de problèmes soient généralisables à un problème plus complexe comme celui de la vision.

- La troisième possibilité consiste à utiliser une plate-forme matérielle plus adaptée pour ce type de calculs. Les processeurs graphiques en particulier sont employés ici uniquement pour le rendu des images. Les générations récentes de ces processeurs sont maintenant entièrement programmables et particulièrement bien adaptées aux calculs de traitement d'images massivement utilisés dans notre application. Cela permettrait également de réduire le flux de données allant du processeur graphique au processeur central qui semble être une des limitations de notre système actuel. Il s'agit là d'un travail de développement considérable mais qui permettrait de gagner au moins un ordre de grandeur dans les temps d'exécution des expériences.

4.5.2 Résolution du problème de *bloating*

Le phénomène de *bloating* que nous avons rapidement présenté dans les sections précédentes est un problème courant en programmation génétique. Il se manifeste par une taille croissante des programmes au fil de l'évolution, sans que cette taille puisse être liée à un quelconque avantage fonctionnel. Les algorithmes évolués contiennent donc des primitives qui n'ont pas d'effet significatif sur le résultat final. Dans le système présenté, nous avons pris une mesure pour limiter le *bloating* et surtout la durée des expériences, qui est de donner des scores très grands (nous rappelons que l'objectif est de minimiser les scores) aux algorithmes pour lesquels le temps de calcul dépasse un certain seuil. Cependant cela n'empêche pas l'accumulation de primitives très rapidement calculées comme nous avons pu le constater dans nos expériences.

Plusieurs solutions ont été proposées pour limiter le phénomène de *bloating* en programmation génétique. La plus utilisée est probablement la pression de parcimonie (*parsimony pressure* en anglais) qui consiste à augmenter les scores de fitness des individus en fonction de leur taille (voir [Soule 98] par exemple). L'inconvénient de cette méthode est qu'elle oblige à fixer manuellement une fonction de parcimonie (généralement un simple coefficient) pour indiquer comment la taille du programme va affecter la fonction de fitness. Des techniques ont été proposées pour fixer ce coefficient automatiquement (voir [Poli 08] par exemple) mais dans tous les cas cette méthode compare directement deux variables non comparables par essence, à savoir un score indiquant la performance d'un algorithme et une taille d'algorithme. De ce fait, il est plus intéressant de prendre en compte cette taille d'algorithme (qui peut également être une autre mesure de complexité) comme un critère indépendant et de l'intégrer dans un processus d'évolution multi-critères [Bleuler 01].

Notre système utilisant déjà une évolution multi-critères, nous avons essayé d'implémenter cette méthode directement. Ce test a été fait pour l'évolution utilisant uniquement l'imitation dont le but est de produire des contrôleurs pour un robot réel. Comme de plus la commande de vitesse linéaire n'est pas réellement nécessaire dans ce cas, nous l'avons supprimée et remplacée par un critère de taille d'algorithme. L'évolution se déroule ainsi toujours avec deux critères. Il s'est avéré que les algorithmes produits de cette manière sont en effet beaucoup plus simples et compacts que ceux créés précédemment, mais aussi beaucoup moins efficaces. Ce problème est probablement dû au fait que les algorithmes simples, utilisant uniquement 2 ou 3 primitives, prennent le dessus dans les premières générations de l'évolution, et empêchent des algorithmes plus complexes et plus efficaces d'évoluer. Ce type de problème a déjà été constaté

dans [de Jong 03]. Les auteurs suggèrent que lors de l'utilisation d'une évolution multi-critères pour limiter le phénomène de *bloating*, une attention particulière doit être apportée à la promotion de la diversité dans la population. Il est probable que dans notre cas, le système de promotion de la diversité implémenté dans NSGA-2 ne soit pas suffisant pour empêcher la convergence prématurée vers des algorithmes de petite taille et peu efficaces. Il serait donc intéressant d'essayer de résoudre ce problème avec d'autres algorithmes où la promotion de la diversité est mieux assurée.

4.5.3 Autres méthodes pour combiner différents types d'évaluation

Nous avons montré au §4.2 comment l'utilisation d'une autre fonction d'évaluation pendant une partie de l'évolution peut améliorer les résultats finaux de celle-ci. Un des points faibles de l'approche présentée est que la manière d'articuler ces deux types d'évaluation est rigide. En l'occurrence, il s'agit de deux phases successives dont la durée est fixée à 50 générations. Rien ne prouve que cette durée soit idéale, peut-être que 10 générations d'évaluation basée sur l'imitation suffisent, ou au contraire que 90 générations donneraient de meilleurs résultats. Il serait bien entendu possible de tester ces hypothèses et de déterminer empiriquement quelle est la durée idéale pour chacune des phases. Néanmoins cette approche serait très longue étant donnée la durée de nos expériences, et de plus les résultats obtenus ne seraient pas forcément généralisables à d'autres applications. Nous avons donc choisi de nous intéresser à des approches où les deux types d'évaluation ont lieu simultanément.

Notre première implémentation de ce type d'approche a été d'utiliser chaque type d'évaluation sur des îlots différents. Plus précisément, nous utilisons l'évaluation basée sur l'imitation sur trois des quatre îlots. La migration a lieu uniquement depuis ces trois îlots en direction du quatrième, où l'évaluation se fait de manière objective sur les scores d'accession au but et d'évitement d'obstacles. L'hypothèse est qu'à partir du moment où des individus assez performants vont apparaître dans un des trois premiers îlots, ils vont migrer rapidement vers le quatrième et être alors améliorés pour devenir plus robustes. En pratique, cette hypothèse s'est avérée erronée. En effet, les individus qui sont performants en imitation ne sont pas immédiatement adaptés pour l'évaluation plus objective. De ce fait, ils vont être moins performants que ceux qui auront évolués depuis le début avec cette fonction objective et ils seront tout de suite éliminés avant même d'avoir eu le temps d'être améliorés.

L'autre implémentation de cette approche que nous avons testée a été inspirée par une méthode récente pour améliorer l'évolution incrémentale. L'idée est d'implémenter celle-ci sous la forme d'une évolution multi-objectifs [Mouret 08]. Les différents types d'évaluation s'effectuent ici en parallèle et constituent autant d'objectifs à optimiser. Dans notre cas où chaque fonction d'évaluation fournit deux scores, on a donc un total de quatre objectifs à minimiser. Le problème que nous avons rencontré dans ce cas est que le front de Pareto dans cet espace à quatre dimensions devient rapidement très large, et la plupart des individus évalués se retrouvent en pratique sur ce front. Il devient alors impossible de distinguer ceux qui présentent réellement un comportement intéressant des autres. Il s'agit ici d'un problème assez courant en optimisation multi-objectifs lorsque le nombre de critères augmente. Il a été montré que l'utilisation de méthodes basées sur la notion d'ε-dominance peut nettement améliorer les résultats dans ce cas [Laumanns 02]. L'idée est ici de discrétiser l'espace des objectifs en utilisant une grille dont le pas est fixé à l'avance. Seule une solution au maximum peut être conservée pour

la génération suivante dans chaque case de cette grille. De plus, une solution ne sera pas conservée si elle se trouve dans une case dominée par une autre qui contient déjà une solution. Cette méthode permet de diminuer le nombre de solutions dans le front de Pareto et de garantir une bonne répartition des individus le long de celui-ci. Il serait donc intéressant de tester ce type de techniques sur notre système à l'avenir.

4.5.4 Extensions de l'opérateur d'extraction d'information visuelle

Une autre limitation de notre système est l'opérateur utilisé pour extraire l'information depuis les images. Celui-ci consiste uniquement à effectuer la moyenne de la valeur des pixels sur des fenêtres de position et de taille données. Certes, cette position et cette taille sont paramétrées par le processus d'évolution, mais elles restent fixes pour un algorithme donné. Il serait intéressant de ce point de vue d'essayer d'appliquer des techniques de vision active à ce système, afin de trouver et suivre automatiquement les zones intéressantes dans l'image au cours du déplacement. Nous avons déjà présenté au §1.4.3 les travaux sur la vision active de l'équipe de Dario Floreano (voir [Floreano 04] notamment). Leurs recherches s'effectuent dans le cadre de la robotique évolutionnaire avec une représentation basée sur des réseaux de neurones, mais il serait possible d'adapter ce type de système dans le cadre de la programmation génétique. La solution la plus simple pour cela serait d'ajouter aux données de sortie de l'algorithme des commandes de déplacement des fenêtres utilisées. Il serait par contre nécessaire dans ce cas de limiter le nombre de fenêtres afin de ne pas accroître la complexité du système.

Il serait également intéressant d'ajouter des méthodes d'extraction de type extraction de bords ou de points d'intérêts. En effet, les informations les plus intéressantes à exploiter dans l'image ne correspondent pas toujours à des objets larges et donc l'opérateur d'extraction basé sur des fenêtres n'est pas forcément le mieux adapté à tous les cas. Par contre, cet ajout implique la définition de nouveaux types de données et de nouveaux opérateurs pour les exploiter correctement. Cela entraînerait une augmentation de la taille de l'espace des algorithmes possibles ce qui peut avoir un effet négatif sur les performances du système.

4.5.5 Intégration de mémoire à court terme

Notre système ne disposant ni de mémoire ni de représentation interne de l'environnement, il est incapable d'effectuer des actions basées sur ses perceptions passées. Concrètement, cela signifie qu'il est incapable de se souvenir de la position d'un obstacle une fois que ce dernier est sorti de son champ de vision. Dans les environnements étudiés dans cette thèse, les obstacles restent simples et l'angle de vision est large ce qui permet de développer des algorithmes d'évitement d'obstacles efficaces sans mémoire. Par contre si l'on considère des objets de forme plus complexe ou plus larges, une certaine forme de mémoire serait un atout non négligeable pour ce type d'application.

L'intégration de mémoire dans un système de programmation génétique peut se faire assez simplement en créant des registres capables d'enregistrer des données et des opérateurs permettant d'écrire et de lire dans ces registres. L'inconvénient de cet ajout d'opérateurs est là aussi

une augmentation de la taille de l'espace des algorithmes possibles. Cela risque donc de diminuer les performances du système au final. Il serait néanmoins intéressant de tester ce type d'approche pour des expériences futures dans des environnements plus complexes.

4.5.6 Autres méthodes de combinaison de différents algorithmes

Nous avons déjà présenté au §4.3 les difficultés que présente la conception d'algorithmes devant réaliser des compromis entre plusieurs objectifs parfois contradictoires. Nous avons également proposé des solutions à ce problème, à savoir l'utilisation d'un opérateur de type *if-then-else* qui ne s'est pas avéré concluante, et l'utilisation d'une structure restreinte fixée qui nous a permis d'améliorer les capacités de généralisation de nos algorithmes.

Un inconvénient de ces deux solutions est l'utilisation d'un test qui va générer des comportements non continus à la frontière des zones où le test est vérifié et de celles où il ne l'est pas. Si de plus les paramètres évoluent le long de ces zones frontières durant les expériences, cela peut générer des comportements chaotiques fortement indésirables pour un robot mobile.

Une solution possible à ce problème est d'utiliser des règles de logique floue à la place de ces tests binaires. Cela permettrait de s'assurer que le comportement du robot reste continu dans toutes les conditions. Le seul inconvénient à cette solution est que cela nécessite de définir plusieurs variables floues et leurs domaines d'appartenance. Ces variables correspondant typiquement à des mesures sur l'image, il serait plus judicieux de laisser le système d'évolution les fixer. Cela entraînerait donc un nombre conséquent de paramètres supplémentaires pour chaque algorithme. Néanmoins, il serait très intéressant de tester ce type de système et de voir s'ils peuvent améliorer les capacités de généralisation des algorithmes notamment.

4.5.7 Adaptation *online*

Actuellement ce système est uniquement utilisé pour créer automatiquement des algorithmes de manière *offline*. Pour une application réelle, il n'est pas concevable de devoir faire évoluer des algorithmes de cette manière pour chaque type d'environnement que le robot peut rencontrer. Il est donc nécessaire de définir une méthode pour que le robot puisse adapter son comportement *online* en fonction de l'environnement courant. Cela n'est pas forcément incompatible avec notre système. On peut en effet faire évoluer des algorithmes adaptés pour différents types d'environnement, et concevoir un contrôleur de plus haut niveau qui sélectionnerait en temps réel quel algorithme utiliser en fonction de l'environnement courant.

Cependant, même dans ce cas il serait souhaitable que les algorithmes eux-mêmes puissent être modifiés en temps réel, ne serait-ce que pour prendre en compte les changements qui peuvent avoir lieu dans un environnement réel. Cette approche s'apparente au concept de *seeding* que nous avons décrit au §3.3.3. Le système serait ici initialisé avec des algorithmes résultant d'un processus d'évolution *offline*, et ceux-ci seraient adaptés par la suite en fonction de l'environnement.

Plusieurs méthodes sont envisageables pour cela, mais dans tous les cas il est nécessaire de définir une mesure indiquant la performance de l'algorithme utilisé. Cette mesure doit pouvoir être acquise en temps réel directement par le robot. Il peut s'agir du nombre de contacts qui ont

eu lieu avec des obstacles durant un temps donné, auquel cas le robot doit être équipé de capteurs de contact et d'un contrôleur bas niveau permettant au robot de se repositionner correctement après une collision. Il pourrait aussi s'agir de capteurs de distance mais cela supprimerait un des intérêts majeurs de notre système qui est justement de se passer de tels capteurs.

Une fois cette mesure de performance choisie, une première méthode d'adaptation *online* consiste à utiliser le processus d'évolution tel quel sur le robot. Les algorithmes sont dans ce cas testés séquentiellement sans réinitialisation de la position. Les conditions d'évaluation ne seront donc pas identiques pour tous les algorithmes. La fonction de fitness utilisée sera typiquement la mesure de performance choisie. Une autre méthode consiste à utiliser des techniques d'apprentissage par renforcement pour sélectionner les algorithmes efficaces. Par contre, ces méthodes ne traitent pas le problème de la génération de nouveaux algorithmes. On pourrait pour cela utiliser également des opérateurs de croisement et mutation, ou alors des opérateurs de transformations plus simples comme une descente de gradient sur les différents paramètres utilisés dans l'algorithme.

Dans tous les cas, il est également nécessaire de définir quand est-ce que le robot doit utiliser un algorithme connu pour se déplacer et quand est-ce qu'il doit en tester de nouveaux. Il s'agit de trouver un compromis entre l'exploitation d'un comportement efficace et l'exploration de nouveaux algorithmes pour pouvoir s'adapter à l'environnement. Ce compromis peut être fixé à l'avance comme un paramètre du système. Il peut également être adapté en fonction de la performance du robot. Tant que le robot ne heurte pas d'obstacles, il est plus rentable d'exploiter les algorithmes connus. S'il commence à heurter trop souvent des obstacles, c'est que les algorithmes utilisés ne sont pas adaptés à l'environnement actuel. Il est alors nécessaire d'en tester de nouveaux.

Jusqu'ici, nous avons évoqué l'adaptation *online* comme moyen de faire face à un environnement inconnu. Mais celle-ci peut également être vue comme l'intégration de techniques permettant de modifier le fonctionnement des algorithmes au cours du temps. C'est ce qui est réalisé d'une certaine manière dans le cerveau humain par les mécanismes d'attention visuelle. C'est également ce qui a été étudié en vision artificielle par les travaux en vision active, que nous avons évoqués au §4.5.4. Cependant ces derniers se focalisent en général sur le déplacement d'une zone d'intérêt dans l'image pour imiter les saccades visuelles permettant à l'oeil humain de parcourir une scène. De ce point de vue, notre système peut d'ailleurs faire preuve d'une certaine forme implicite de vision active, les déplacements du robot permettant d'amener certaines parties de l'environnement dans le champ visuel. Il serait possible d'implémenter des techniques de vision active de manière plus explicite, en permettant aux algorithmes de déplacer les fenêtres utilisées pour extraire les informations visuelles. Mais la position et la taille de ces fenêtres ne sont que des paramètres parmi d'autres dans nos algorithmes. Il serait tout à fait possible de paramétrer de la même manière d'autres primitives comme la fréquence et l'orientation auxquelles répond un filtre de Gabor par exemple, ou même d'implémenter un système de sélection de primitives similaire aux mécanismes d'inhibition observés dans les réseaux de neurones. Certes, un tel "auto-paramétrage" des algorithmes ajouterait une certaine complexité au système. Mais en imitant les mécanismes d'interaction entre traitements *bottom-up* et sélection *top-down* utilisés par le cerveau, les contrôleurs évolués présenteraient probablement une adaptativité aux changements de l'environnement plus proche de la vision humaine.

4.6 Adaptation du système pour d'autres applications

Une des contributions majeures de cette thèse a été de concevoir une structure extensible pour représenter des algorithmes de vision, et de développer les outils permettant de les faire évoluer automatiquement. Ces concepts sont réutilisables pour beaucoup d'autres applications, potentiellement toutes celles qui utilisent la vision. Les modifications nécessaires pour cela consistent à déterminer les types de données, la base de primitives et les données d'entrée et de sortie adaptés à l'application visée. Il faut également définir une ou plusieurs fonctions de fitness pour évaluer les performances des algorithmes. Nous allons décrire ici quelques applications possibles pour notre système. Nous présenterons sous forme de tableau des exemples simplifiés de grammaire et de fonctions d'évaluation pour chaque application.

4.6.1 Extraction de points d'intérêt

L'extraction de points d'intérêt dans une image consiste à trouver les points qui peuvent facilement être reconnus et suivis d'une image à l'autre. De nombreuses méthodes ont été proposées pour cela, du simple détecteur de coins aux méthodes multi-échelles invariantes aux rotations et changements de point de vue. Un certain nombre de méthodes ont également été proposées pour évaluer la robustesse de ces extracteurs. Nous avons déjà présenté au §3.1.3.1 des travaux utilisant la programmation génétique pour créer de tels extracteurs de points.

Pour ce problème, la donnée d'entrée du système est toujours l'image acquise par la caméra. Par contre, la donnée de sortie est ici une liste de points. Il est donc nécessaire de définir un ou plusieurs opérateurs permettant de construire une liste de points. Le plus couramment utilisé pour cela consiste à extraire les maxima locaux d'une image. Il est également nécessaire de transformer l'image d'entrée afin de mettre en évidence les points d'intérêt. Les filtres déjà implémentés dans notre système pourraient suffire pour cela mais il est également possible d'implémenter d'autres opérateurs. Les travaux existant sur le sujet proposent ainsi d'utiliser des fonctions arithmétiques s'appliquant à chaque pixel [Ebner 99, Trujillo 06]. Tous ces opérateurs peuvent être intégrés à notre système sans aucune modification fondamentale (tableau 4.2).

Le deuxième point à étudier est celui de la fonction à utiliser pour évaluer les algorithmes évolués. Là aussi, plusieurs critères ont été proposés pour déterminer la qualité d'un extracteur de points d'intérêt, par exemple :

- Le nombre de points extraits de l'image
- Le taux de répétabilité des détections de points
- La dispersion des points dans l'image

Des travaux ont déjà utilisé un système d'évolution multi-objectifs pour évaluer des critères de ce type indépendamment [Trujillo 08]. Il serait également très facile d'implémenter ces différentes fonctions de fitness dans notre système. Les résultats obtenus seraient très probablement similaires à ceux qui ont été présentés dans ces différents travaux. Le but n'est pas ici de trouver une nouvelle méthode, mais plutôt de démontrer l'adaptativité de notre système et du formalisme choisi pour représenter des algorithmes de vision.

[1,0]	DÉPART	→	LISTE-POINTS
[1,0]	LISTE-POINTS	→	maxima-locaux (IMAGE)
[0,3]	IMAGE	→	image-vidéo
[0,3]	IMAGE	→	FILTRE-SPATIAL (IMAGE)
[0,3]	IMAGE	→	OPÉRATEUR-UNAIRE (IMAGE)
[0,1]	IMAGE	→	OPÉRATEUR-BINAIRE (IMAGE, IMAGE)
[0,2]	FILTRE-SPATIAL	→	gaussien
[0,2]	FILTRE-SPATIAL	→	laplacien
[0,2]	FILTRE-SPATIAL	→	gabor
[0,2]	FILTRE-SPATIAL	→	différence-de-gaussiennes
[0,2]	FILTRE-SPATIAL	→	sobel
[0,34]	OPÉRATEUR-UNAIRE	→	racine-carrée
[0,33]	OPÉRATEUR-UNAIRE	→	carré
[0,33]	OPÉRATEUR-UNAIRE	→	log
[0,3]	OPÉRATEUR-BINAIRE	→	addition
[0,3]	OPÉRATEUR-BINAIRE	→	soustraction
[0,2]	OPÉRATEUR-BINAIRE	→	multiplication
[0,2]	OPÉRATEUR-BINAIRE	→	division

TAB. 4.2 : *Exemple de grammaire utilisable pour l'extraction de points d'intérêt*

4.6.2 Reconnaissance d'amers visuels

Un problème souvent associé au précédent est celui de l'encodage de descripteurs robustes à partir des points d'intérêts détectés dans l'image, et la reconnaissance de ces points dans d'autres images à partir de ces descripteurs. Les descripteurs SIFT par exemple sont très employés actuellement pour cela [Lowe 04]. Ils utilisent des histogrammes sur l'orientation des gradients dans plusieurs *patchs* d'images autour du point d'intérêt pour décrire celui-ci. Ils présentent l'avantage d'être très robustes mais sont relativement longs à calculer et leur représentation reste assez lourde (chaque point d'intérêt est décrit par un vecteur de 128 éléments). Une représentation beaucoup plus simple consiste simplement à encoder les valeurs des pixels autour du point d'intérêt, éventuellement en filtrant auparavant l'image. De multiples combinaisons de ces principes sont par ailleurs possibles. Il serait tout à fait envisageable de reprendre ces idées dans notre système pour faire évoluer des descripteurs adaptés à un contexte donné. Le tableau 4.3 présente un exemple de grammaire qui pourrait être utilisée pour cela. L'évolution multi-objectifs permettrait en outre de produire des descripteurs représentant différents compromis intéressants. Les fonctions d'évaluation utilisées pourraient par exemple être :

- La robustesse des mises en correspondance (taux de reconnaissance des points)
- La complexité de l'algorithme d'encodage
- La longueur de la représentation utilisée

[1,0]	DÉPART	→	DESCRIPTEUR
[0,4]	DESCRIPTEUR	→	valeurs-pixels (IMAGE)
[0,3]	DESCRIPTEUR	→	histogramme (IMAGE)
[0,3]	DESCRIPTEUR	→	histogrammes (PATCHS-IMAGE)
[1,0]	PATCHS-IMAGE	→	découpage (IMAGE)
[0,3]	IMAGE	→	image-vidéo
[0,3]	IMAGE	→	FILTRE-SPATIAL (IMAGE)
[0,2]	IMAGE	→	direction-gradients (IMAGE)
[0,2]	IMAGE	→	rotation (IMAGE)
[0,17]	FILTRE-SPATIAL	→	gaussien
[0,17]	FILTRE-SPATIAL	→	laplacien
[0,17]	FILTRE-SPATIAL	→	gabor
[0,17]	FILTRE-SPATIAL	→	différence-de-gaussiennes
[0,16]	FILTRE-SPATIAL	→	sobel
[0,16]	FILTRE-SPATIAL	→	sous-échantillonage

TAB. 4.3 : *Exemple de grammaire utilisable pour construire des descripteurs visuels utilisables pour la reconnaissance d'amers visuels*

4.6.3 Localisation

L'idée n'est pas ici de chercher à réimplémenter des méthodes de SLAM où la localisation est effectuée à l'aide de méthodes analytiques en utilisant le positionnement géométrique de points dans l'image. Il s'agit plutôt de résoudre des problèmes de type localisation topologique consistant à reconnaître un lieu à partir d'une image. Les techniques existant pour cela utilisent généralement un ensemble de descripteurs d'images et tentent par des méthodes statistiques d'apprendre la relation entre ces descripteurs et un lieu (voir [Filliat 07] par exemple).

Nous proposons ici d'utiliser conjointement les deux systèmes décrits précédemment avec une fonction d'évaluation spécifique liée à la localisation pour sélectionner automatiquement les types de descripteurs adaptés pour décrire et reconnaître un lieu dans un contexte donné. L'idée est de faire évoluer simultanément l'extraction de points d'intérêt et l'encodage des descripteurs, et de les évaluer par exemple avec les fonctions suivantes :

- Taux de reconnaissance des lieux enregistrés
- Complexité globale de l'algorithme

Cette approche permettrait notamment de supprimer le biais lié à l'utilisation d'une fonction d'évaluation spécifique à chaque étape. Les algorithmes seraient ici sélectionnés uniquement sur leur capacité à atteindre le but final, à savoir la reconnaissance d'un lieu.

Conclusion

Nous avons présenté dans cette thèse un système utilisant la programmation génétique pour synthétiser automatiquement des algorithmes de vision pour un robot mobile en fonction du contexte. Nous nous sommes attachés plus précisément à résoudre le problème de l'évitement d'obstacles, fonction indispensable pour l'acquisition d'un comportement autonome. Nous avons formalisé la structure d'un algorithme de vision en utilisant une grammaire, et montré qu'il est possible de faire évoluer des contrôleurs efficaces en utilisant la programmation génétique.

Nous avons également utilisé des méthodes plus spécifiques d'évolution artificielle comme l'évolution multi-critères et le découpage de la population en îlots pour améliorer notre système. Nous avons montré qu'en utilisant des techniques d'évolution guidée il est possible d'accélérer l'évolution et d'augmenter les performances des algorithmes évolués. Nous avons créé une nouvelle forme d'évolution guidée basée sur l'imitation qui s'est avérée particulièrement efficace pour notre application. Pour finir, nous avons validé nos résultats sur un robot mobile et ainsi prouvé qu'il est possible d'utiliser ce système pour concevoir des contrôleurs efficaces pour une utilisation réelle.

Les principales contributions de cette thèse sont d'aborder le problème de la vision de manière globale, de définir un formalisme extensible pour représenter des algorithmes de vision et de développer les outils permettant de créer des algorithmes efficaces en utilisant ce formalisme. Cette thèse ouvre de plus un certain nombre de perspectives pour de futurs travaux. Il serait ainsi intéressant d'adapter ce système pour d'autres applications, comme par exemple la reconnaissance d'amers visuels ou la localisation topologique. Plusieurs extensions sont également envisageables, notamment pour résoudre les problèmes de *bloating* ou utiliser simultanément plusieurs méthodes différentes d'évaluation. La piste que nous estimons la plus prometteuse consiste à mettre en œuvre des méthodes inspirées des mécanismes d'attention visuelle pour adapter les algorithmes *online* en fonction de l'environnement courant. Cela permettrait de réduire encore un peu plus le fossé entre les capacités d'adaptation de la vision artificielle et celles de la vision humaine.

Annexe A

Calculs liés à la géométrie du flux optique

Nous allons détailler dans cette annexe les calculs permettant d'obtenir les relations décrivant le flux optique dans différents types de projection, comme cela a été présenté au §2.1.1.1.

A.1 Définition du flux optique dans le repère de la caméra

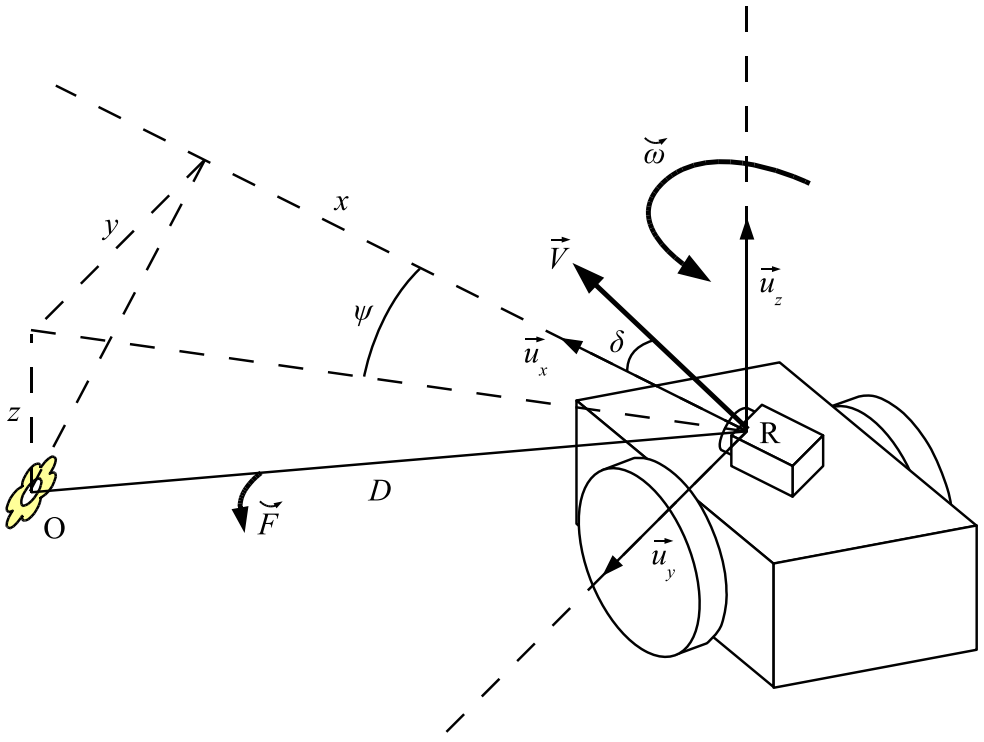


FIG. A.1 : *Déplacement du robot générant le flux optique.*

Nous reprenons en figure A.1 la représentation du mouvement générant le flux optique. Pour exprimer le déplacement du point O dans le repère (R, $\vec{u}_x$, $\vec{u}_y$, $\vec{u}_z$), nous utilisons les relations

suivantes déduites de cette figure lorsqu'on considère uniquement le mouvement de translation $\vec{V}$:

$$\begin{aligned} \dot{x} &= -V\cos\delta \\ \dot{y} &= -V\sin\delta \\ \dot{z} &= 0 \end{aligned}$$

Lorsqu'on considère au contraire uniquement le mouvement de rotation, on a :

$$\begin{aligned} \dot{\psi} &= -(\omega+\dot{\delta}) \\ \dot{z} &= 0 \\ \dot{D} &= 0 \end{aligned}$$

Exprimons à présent x et y en fonction de ψ :

$$\begin{aligned} x &= \sqrt{D^2-z^2}\,\cos\psi \\ y &= \sqrt{D^2-z^2}\,\sin\psi \end{aligned}$$

En dérivant ces expressions, on obtient :

$$\begin{aligned} \dot{x} &= -\sqrt{D^2-z^2}\,\sin\psi\,\dot{\psi} \\ &= (\omega+\dot{\delta})\,y \\ \dot{y} &= \sqrt{D^2-z^2}\,\cos\psi\,\dot{\psi} \\ &= -(\omega+\dot{\delta})\,x \end{aligned}$$

D'où les relations finales prenant en compte la translation et la rotation :

$$\begin{aligned} \dot{x} &= -V\cos\delta+(\omega+\dot{\delta})\,y \\ \dot{y} &= -V\sin\delta-(\omega+\dot{\delta})\,x \\ \dot{z} &= 0 \end{aligned}$$

Il sera également utile pour la suite de calculer $\dot{D}$. On réalise cela en utilisant la relation :

$$D^2 = x^2+y^2+z^2$$

En la dérivant, on obtient :

$$\begin{aligned} 2D\,\dot{D} &= 2x\,\dot{x}+2y\,\dot{y}+2z\,\dot{z} \\ \dot{D} &= \frac{x}{D}(-V\cos\delta+(\omega+\dot{\delta})\,y)+\frac{y}{D}(-V\sin\delta-(\omega+\dot{\delta})\,x) \end{aligned}$$

D'où finalement :

$$\dot{D} = -\frac{V}{D}(x\cos\delta+y\sin\delta)$$

A.2 Projection sphérique

Nous reprenons dans la figure A.2 la représentation du flux optique lorsqu'il est projeté en coordonnées sphériques. Notons que les angles α et β ne sont pas disponibles directement sur une image sphérique, il est nécessaire de les calculer à partir des coordonnées x_s et y_s avec les formules suivantes :

$$\begin{aligned} \alpha &= \sqrt{x_S^2 + y_S^2} \\ \beta &= \arctan \frac{y_S}{x_S} \end{aligned}$$

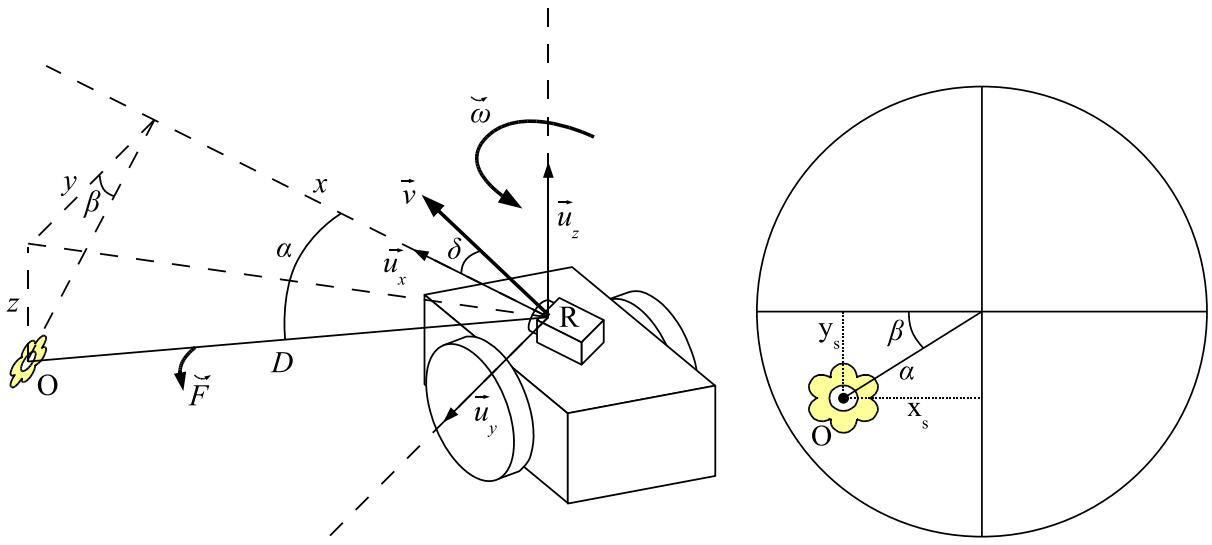


FIG. A.2 : *Représentation de l'image perçue en coordonnées sphériques.*

On obtient par géométrie à partir de la figure A.2 les relations suivantes :

$$\begin{aligned} x &= D\cos\alpha \\ y &= D\sin\alpha\cos\beta \end{aligned}$$

ainsi que les relations :

$$\begin{aligned} \cos\alpha &= \frac{x}{D} \\ \tan\beta &= \frac{z}{y} \end{aligned}$$

Pour obtenir la représentation du flux optique en coordonnées sphériques, il suffit de dériver ces deux dernières expressions :

$$\begin{aligned}
-\sin\alpha\ \dot{\alpha} &= \frac{\dot{x}}{D} - \frac{x\,\dot{D}}{D^2} \\
&= \frac{-V\cos\delta + (\omega+\dot{\delta})\,y}{D} + \frac{xV(x\cos\delta + y\sin\delta)}{D^3} \\
&= \frac{V}{D}\left(\left(\frac{x^2}{D^2} - 1\right)\cos\delta + \frac{x\,y}{D^2}\sin\delta\right) + \frac{y}{D}(\omega+\dot{\delta}) \\
&= \frac{V}{D}((\cos^2\alpha - 1)\cos\delta + \cos\alpha\sin\alpha\cos\beta\sin\delta) + \sin\alpha\cos\beta(\omega+\dot{\delta})
\end{aligned}$$

D'où finalement :

$$\dot{\alpha} = \frac{V}{D}(\cos\delta\sin\alpha - \cos\alpha\cos\beta\sin\delta) - \cos\beta(\omega+\dot{\delta})$$

On procède de même pour déterminer $\dot{\beta}$:

$$\begin{aligned}
\frac{\dot{\beta}}{\cos^2\beta} &= -\frac{z\,\dot{y}}{y^2} + \frac{\dot{z}}{y} \\
&= -\frac{\tan\beta}{y}(-V\sin\delta - (\omega+\dot{\delta})\,x) \\
&= \frac{\tan\beta}{D\sin\alpha\cos\beta}(V\sin\delta + D\cos\alpha(\omega+\dot{\delta}))
\end{aligned}$$

D'où finalement :

$$\begin{aligned}
\dot{\beta} &= \frac{\sin\beta}{D\sin\alpha}(V\sin\delta + D\cos\alpha(\omega+\dot{\delta})) \\
&= \frac{V\sin\delta\sin\beta}{D\sin\alpha} + \frac{\sin\beta}{\tan\alpha}(\omega+\dot{\delta})
\end{aligned}$$

A.3 Projection polaire

Nous reprenons dans la figure A.3 la représentation du flux optique lorsqu'il est projeté en coordonnées polaires. On obtient de cette figure les relations :

$$\begin{aligned}
x &= D\cos\theta\cos\psi \\
y &= D\cos\theta\sin\psi \\
\tan\psi &= \frac{y}{x} \\
\sin\theta &= \frac{z}{D}
\end{aligned}$$

Ici aussi, nous allons dériver ces deux dernières expressions pour obtenir la représentation du flux optique en coordonnées polaires :

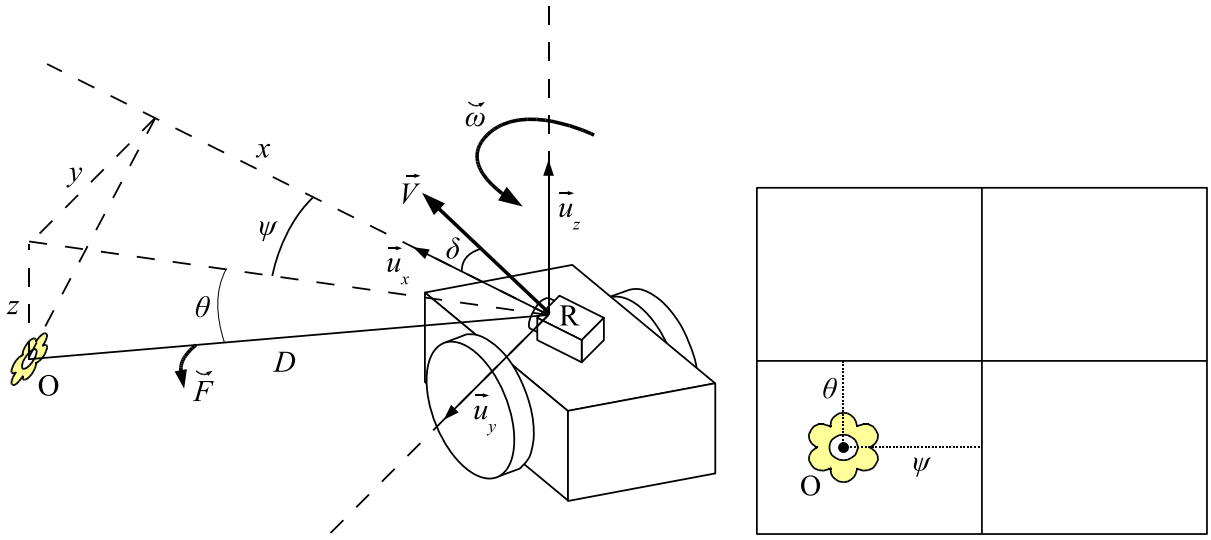


FIG. A.3 : *Représentation de l'image perçue en coordonnées polaires.*

$$\begin{aligned}
\frac{\dot{\psi}}{\cos^2\psi} &= -\frac{y\,\dot{x}}{x^2}+\frac{\dot{y}}{x} \\
&= -\frac{\tan\psi}{x}(-V\cos\delta+(\omega+\dot{\delta})\,y)+\frac{1}{x}(-V\sin\delta-(\omega+\dot{\delta})\,x) \\
&= \frac{V}{x}(\tan\psi\cos\delta-\sin\delta)-(\omega+\dot{\delta})(\tan^2\psi+1)
\end{aligned}$$

On en déduit :

$$\begin{aligned}
\dot{\psi} &= \frac{V\cos^2\psi}{x}(\tan\psi\cos\delta-\sin\delta)-(\omega+\dot{\delta}) \\
&= \frac{V}{D\cos\theta}(\sin\psi\cos\delta-\cos\psi\sin\delta)-(\omega+\dot{\delta}) \\
&= \frac{V\sin(\psi-\delta)}{D\cos\theta}-(\omega+\dot{\delta})
\end{aligned}$$

On procède de même pour déterminer $\dot{\theta}$:

$$\begin{aligned}
\cos\theta\;\dot{\theta} &= \frac{\dot{z}}{D}-\frac{z\,\dot{D}}{D^2} \\
&= \frac{zV}{D^3}(x\cos\delta+y\sin\delta) \\
&= \frac{V\sin\theta}{D^2}(D\cos\theta\cos\psi\cos\delta+D\cos\theta\sin\psi\sin\delta)
\end{aligned}$$

D'où finalement :

$$\begin{aligned}
\dot{\theta} &= \frac{V\sin\theta}{D}(\cos\psi\cos\delta+\sin\psi\sin\delta) \\
&= \frac{V\sin\theta\cos(\psi-\delta)}{D}
\end{aligned}$$

A.4 Projection plane

Nous reprenons dans la figure A.4 la représentation du flux optique lorsqu'il est projeté sur un plan image. On obtient de cette figure les relations :

$$\begin{aligned} x_i &= -f\frac{y}{x} \\ y_i &= f\frac{z}{x} \end{aligned}$$

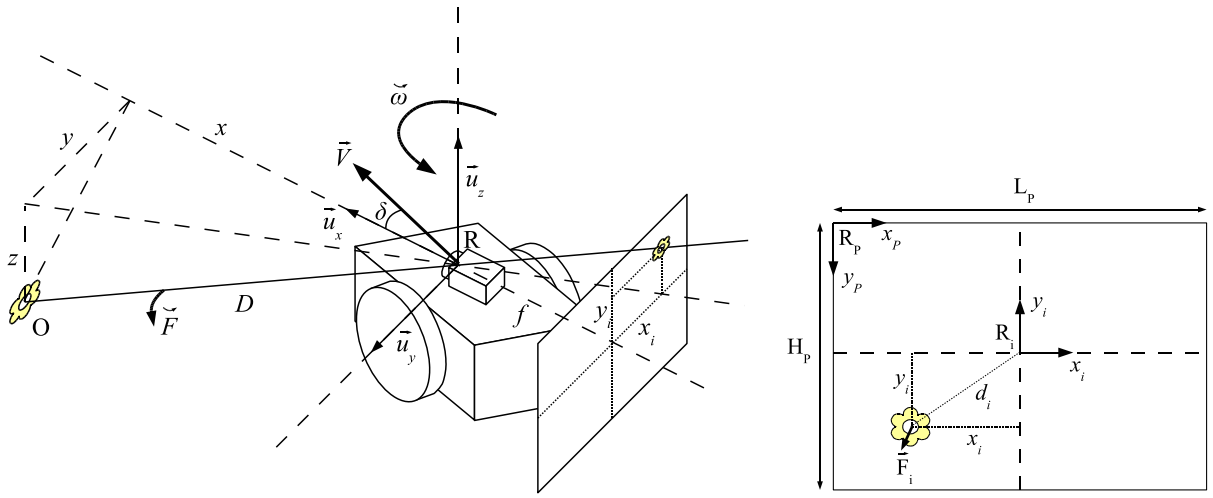


FIG. A.4 : *Représentation du flux optique projeté sur le plan image*

Nous allons dériver ces deux expressions pour obtenir la représentation du flux optique dans l'image :

$$\begin{aligned} \dot{x}_i &= \frac{fy\,\dot{x}}{x^2} - \frac{f\,\dot{y}}{x} && \text{(A.1)} \\ &= \frac{-x_i}{x}(-V\cos\delta + (\omega+\dot{\delta})\,y) - \frac{f}{x}(-V\sin\delta - (\omega+\dot{\delta})\,x) && \text{(A.2)} \\ &= \frac{V}{x}(x_i\cos\delta + f\sin\delta) + (\omega+\dot{\delta})\left(\frac{x_i^2}{f}+f\right) && \text{(A.3)} \\ &= \frac{fV}{x}\left(\frac{x_i\cos\delta}{f}+\sin\delta\right) + f(\omega+\dot{\delta})\left(\frac{x_i^2}{f^2}+1\right) && \text{(A.4)} \\ & && \text{(A.5)} \end{aligned}$$

On procède de même pour déterminer $\dot{y}_i$:

$$\begin{aligned} \dot{y}_i &= -\frac{fz\,\dot{x}}{x^2} + \frac{f\,\dot{z}}{x} \\ &= -\frac{y_i}{x}(-V\cos\delta + (\omega+\dot{\delta})\,y) \\ &= \frac{Vy_i\cos\delta}{x} + (\omega+\dot{\delta})\frac{x_iy_i}{f} \end{aligned}$$

Ces relations sont exprimées dans le repère image centré sur R_i. Il est donc nécessaire avant tout d'effectuer la transformation des coordonnées en pixel vers ces coordonnées image. Cela est effectué à l'aide des relations suivantes obtenues de la figure A.4 et où η est l'angle de vue horizontal :

$$\begin{aligned} x_i &= \left(\frac{2x_P}{L_P} - 1\right) f \tan(\eta/2) \\ y_i &= -\left(\frac{2y_P}{H_P} - 1\right) f \frac{H_P \tan(\eta/2)}{L_P} \\ \dot{x_i} &= \frac{2f\tan(\eta/2)}{L_P} \dot{x_P} \\ \dot{y_i} &= -\frac{2f\tan(\eta/2)}{L_P} \dot{y_P} \end{aligned}$$

Annexe B

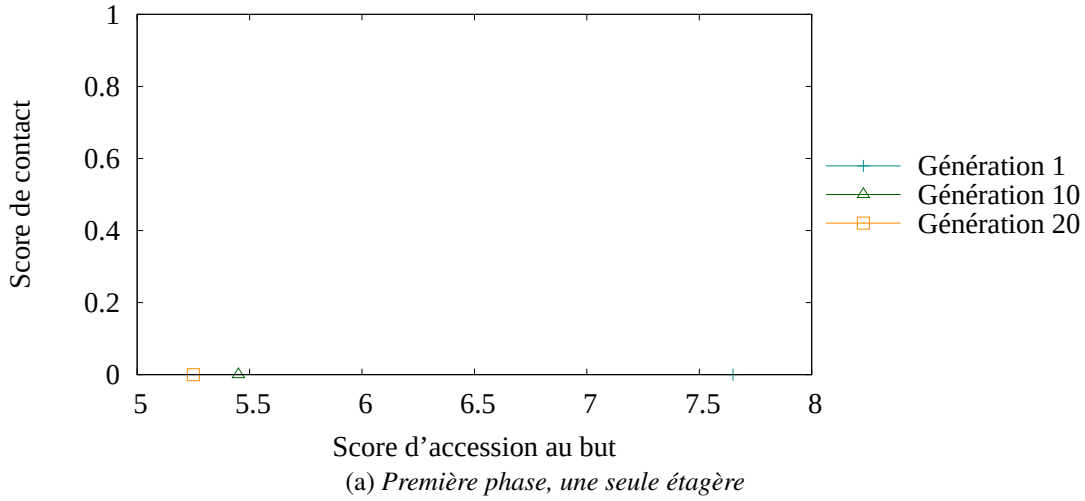
Résultats détaillés

B.1 Évolution incrémentale

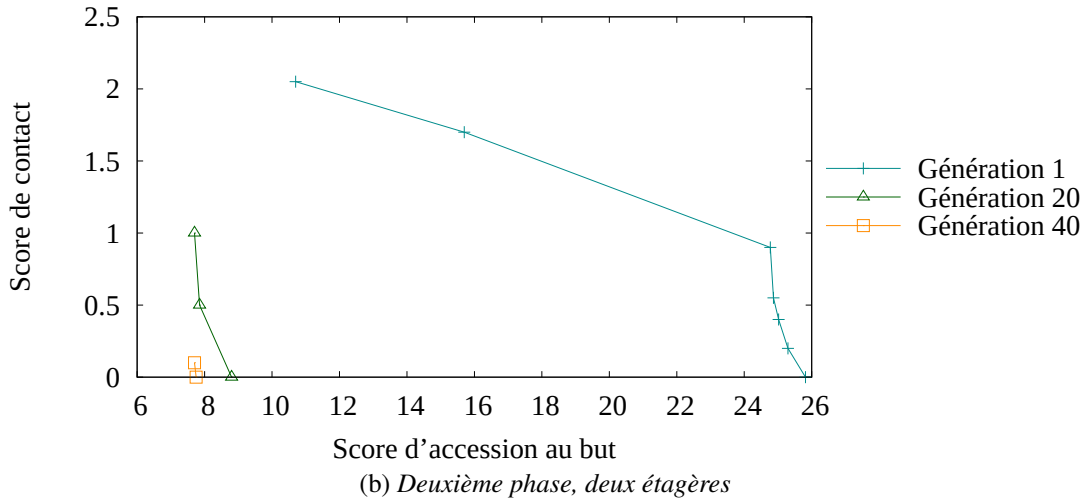
Nous allons décrire ici de manière un peu plus détaillée les résultats obtenus en utilisant l'évolution incrémentale. La figure B.1 présente l'évolution des performances des contrôleurs évolués au cours des trois phases de l'évolution. Nous rappelons que dans la première phase, seule une étagère est présente et celle-ci ne se situe pas sur la trajectoire du robot. Dans la deuxième phase, une deuxième étagère est présente. Dans la troisième phase, les trois étagères sont présentes.

Les problèmes liés à l'évolution incrémentale se comprennent très facilement en observant les trajectoires suivies par des individus évolués à plusieurs stades de l'évolution (figure B.2). Ce type d'évolution permet de produire très rapidement des contrôleurs efficaces dans le cas trivial où un seul obstacle est présent. Lorsqu'on ajoute un deuxième obstacle, le processus d'évolution arrive encore à adapter les contrôleurs qui sont capables d'atteindre le but sans contact dans les deux cas. Par contre, lorsqu'on ajoute un troisième obstacle, les algorithmes évolués ont beaucoup plus de mal à s'adapter. Le contrôleur présenté ici arrive malgré tout à atteindre le point cible dans les deux cas mais avec plusieurs contacts et une trajectoire assez chaotique. Dans d'autres cas les contrôleurs évolués restent bloqués sans pouvoir contourner les obstacles.

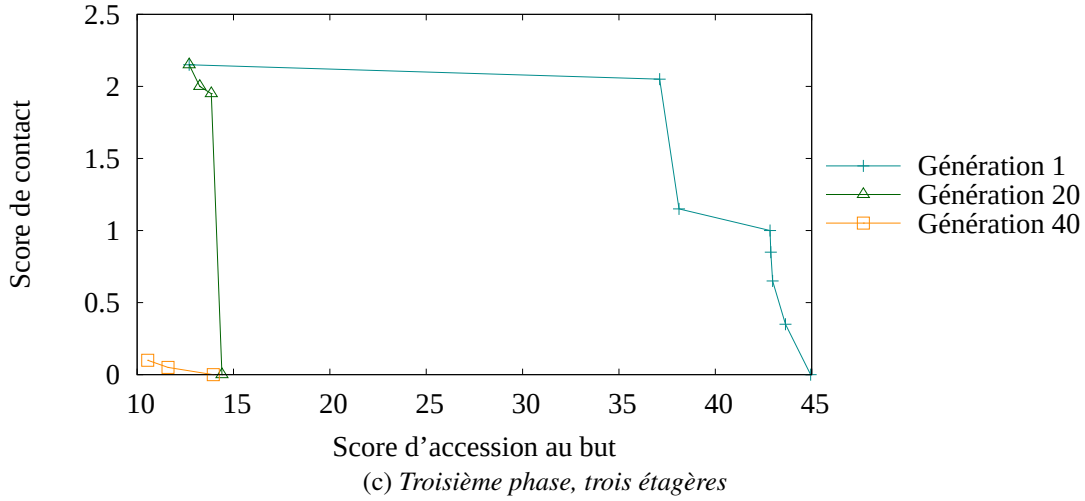
Lorsqu'on observe un contrôleur évolué (figure B.3), on constate que celui-ci reste extrêmement simple et n'utilise pas l'information de direction de but. Il ne peut donc atteindre le point cible que par hasard et peut difficilement généraliser son comportement à d'autres instances du problème, comme l'indique la trajectoire obtenue dans l'environnement de test. Il semble donc que l'évolution incrémentale soit un handicap plus qu'un avantage pour l'évolution de contrôleurs dans ce cas. Elle a en effet plutôt tendance à guider l'évolution sur de mauvaises pistes ce qui sera difficile à rattraper par la suite. Cela ne signifie pas que l'évolution incrémentale en général ne soit pas efficace, mais plutôt que le choix des problèmes de difficulté croissante est crucial et non trivial à réaliser.



(a) *Première phase, une seule étagère*

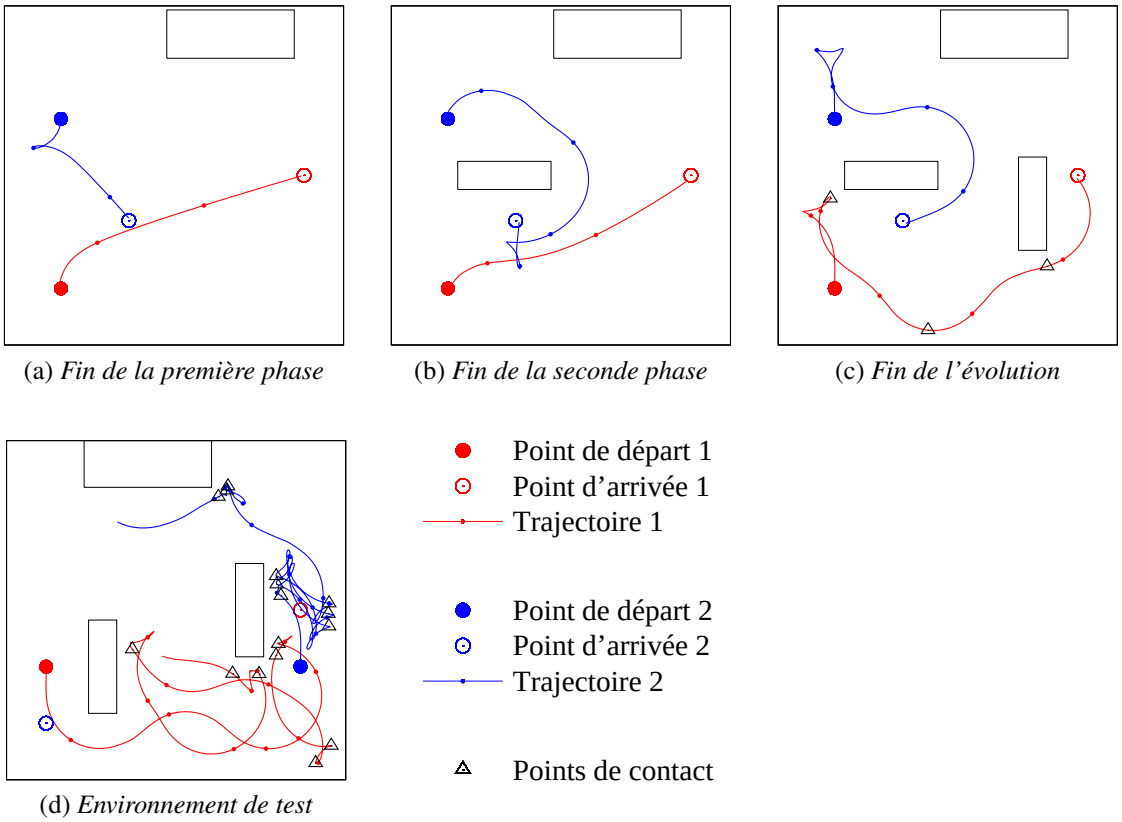


(b) *Deuxième phase, deux étagères*



(c) *Troisième phase, trois étagères*

FIG. B.1 : *Évolution au cours des générations des performances des meilleurs individus lors d'une évolution incrémentale dans l'environnement contenant trois étagères*



(a) *Fin de la première phase* (b) *Fin de la seconde phase* (c) *Fin de l'évolution*

(d) *Environnement de test*

FIG. B.2 : *Trajectoires suivies par des individus évolués grâce à l'évolution incrémentale. Les individus présentés sont sélectionnés à la fin de chaque phase.*

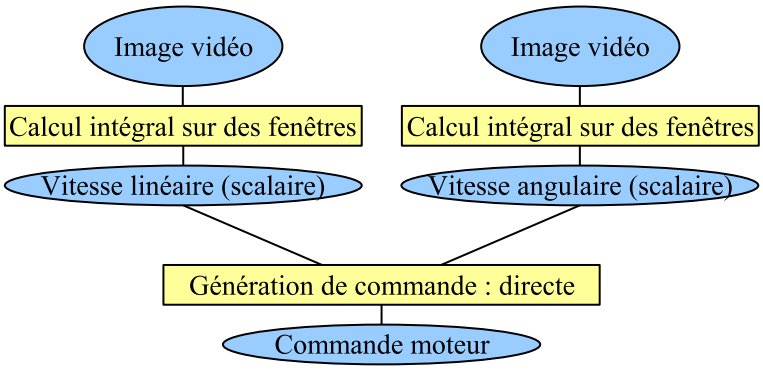


FIG. B.3 : *Algorithme utilisé par un contrôleur évolué grâce à l'évolution incrémentale.*

B.2 Évolution avec *seeding*

Le *seeding* consiste à insérer dans la population initiale d'un des îlots un contrôleur créé manuellement, en l'occurrence il s'agit de notre contrôleur de référence. La figure B.4 montre l'évolution des performances des algorithmes au cours des générations. On constate logiquement que les individus produits au début de l'évolution sont plus performants que lorsqu'on utilise une évolution simple. Ceux-ci sont en fait très similaires au contrôleur de référence comme on peut le voir sur les différentes trajectoires (figure B.5). Dans certains cas ces contrôleurs vont prendre le dessus sur le reste de la population et l'évolution restera bloquée sur ce type de comportement.

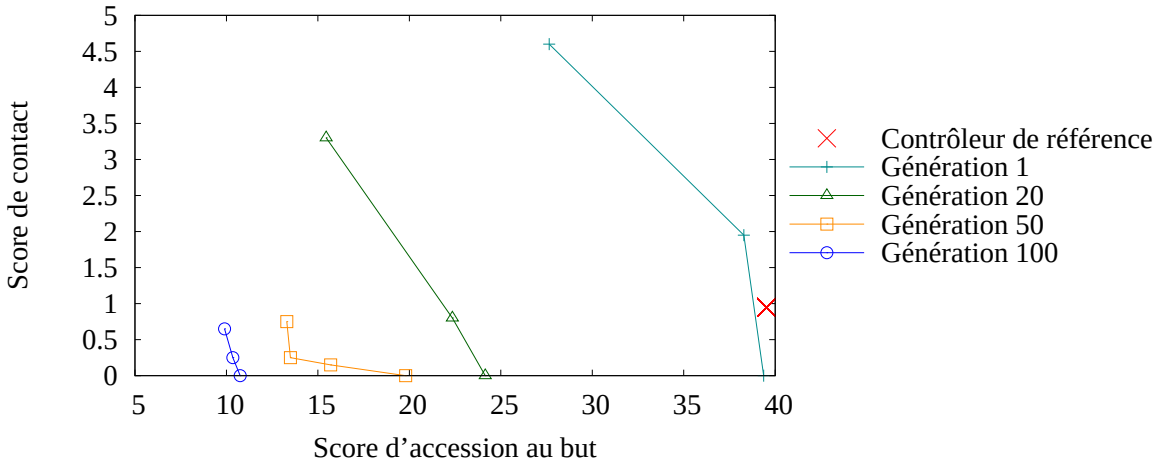
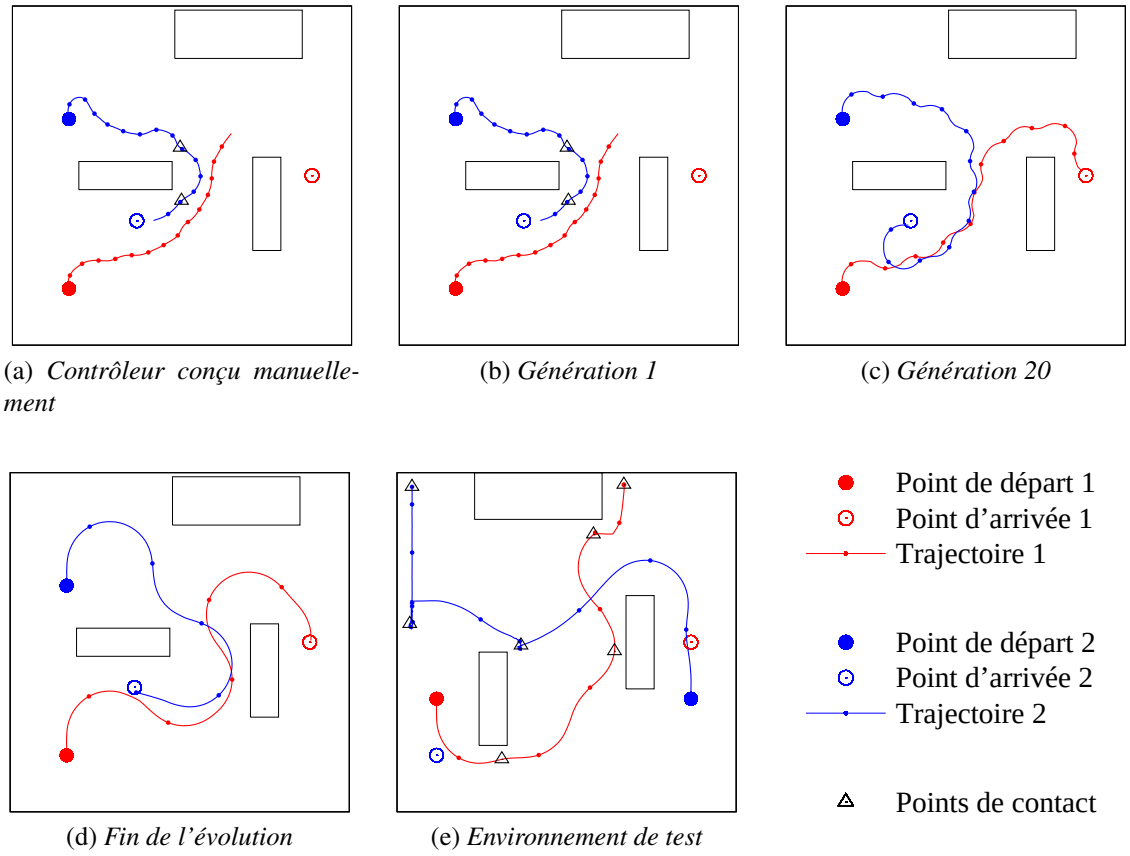


FIG. B.4 : *Évolution au cours des générations des performances des meilleurs individus lors d'une évolution avec* seeding *dans l'environnement constitué d'étagères.*

Le cas présenté ici est nettement plus intéressant. On constate en effet que l'évolution a réussi à améliorer les trajectoires et dès la génération 20 on obtient des contrôleurs capables d'atteindre le point cible dans les deux configurations sans heurter d'obstacles. À la fin de l'évolution, les contrôleurs sont capables d'effectuer ce déplacement très rapidement. Les trajectoires restent de plus assez similaires à celle du contrôleur de référence avec un déplacement en zigzag vers le but. Cela se comprend mieux en examinant un contrôleur évolué (figure B.6). En effet la structure de celui-ci est restée très similaire au contrôleur de référence. La vitesse de déplacement linéaire est toujours constante même si elle est nettement plus rapide ici. La vitesse angulaire est toujours le résultat d'une addition entre une information liée à la direction du but et une mesure sur l'image, même si cette dernière n'est plus liée à un calcul de flux optique. Ce cas montre que l'évolution a réellement réussi à améliorer un comportement donné pour produire un contrôleur plus efficace, ce qui est l'objectif du *seeding*. Néanmoins cela ne suffit pas à obtenir un bon comportement en généralisation, comme on peut le voir en observant la trajectoire obtenue dans l'environnement de test.



(a) *Contrôleur conçu manuellement* (b) *Génération 1* (c) *Génération 20*

(d) *Fin de l'évolution* (e) *Environnement de test*

FIG. B.5 : *Trajectoires suivies par des individus évolués grâce à une évolution avec* seeding. *Les individus présentés sont sélectionnés à plusieurs étapes de l'évolution.*

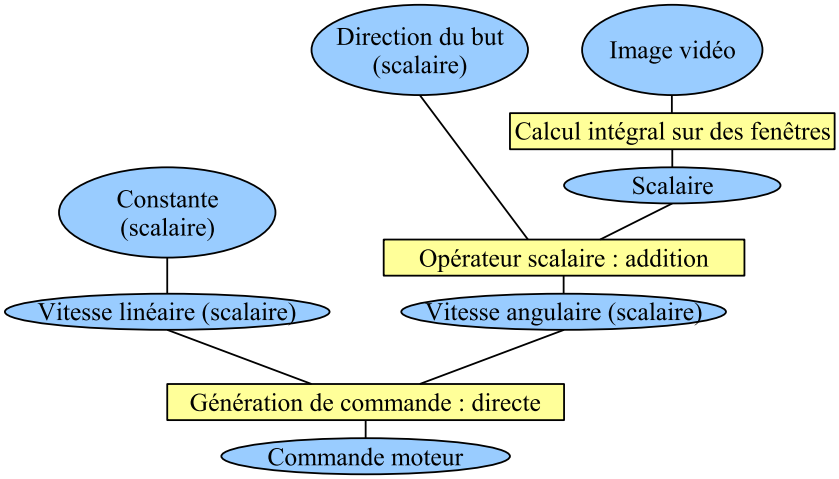
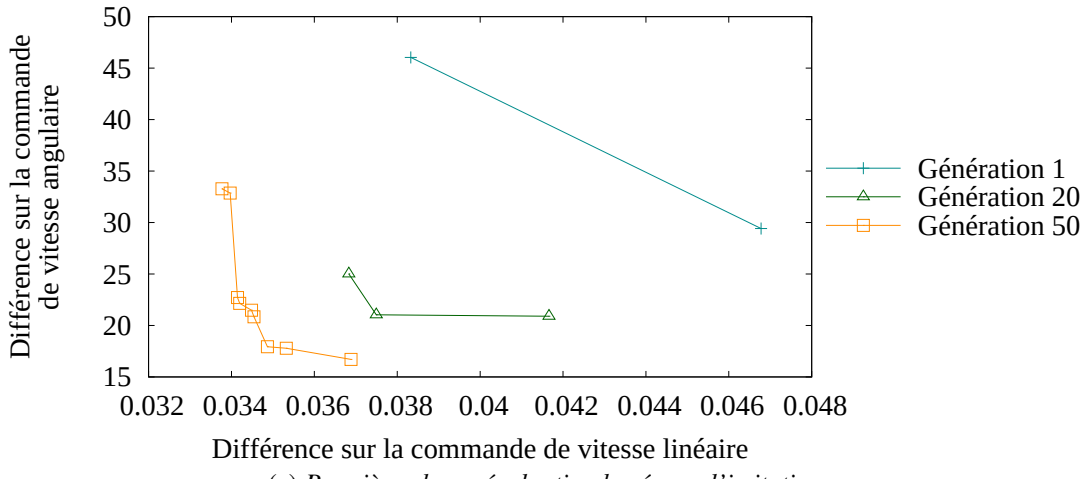


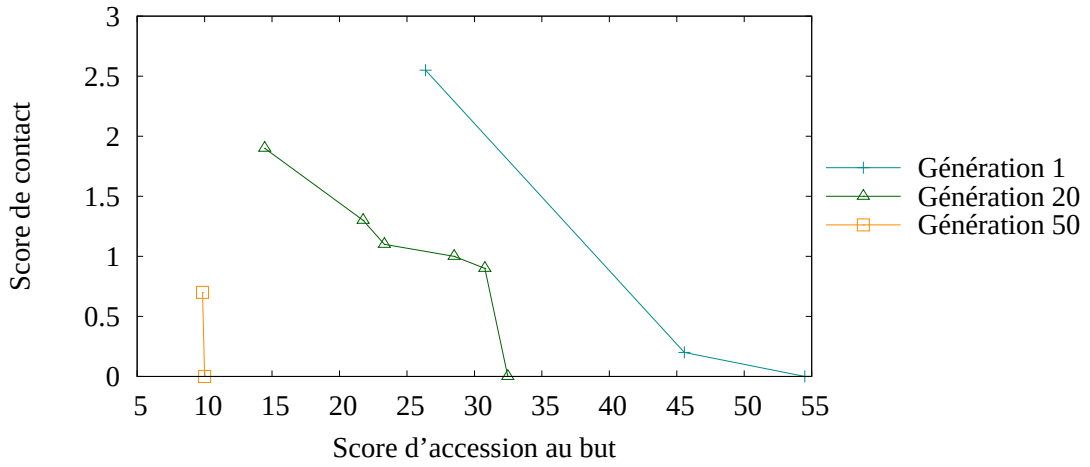
FIG. B.6 : *Algorithme utilisé par un contrôleur évolué grâce à l'évolution avec* seeding.

B.3 Évolution en deux phases

Ce type d'évolution a été développé pour cette thèse et consiste à utiliser une première phase basée sur l'imitation d'un comportement enregistré pour amorcer l'évolution. La figure B.7 présente l'évolution des performances des contrôleurs au cours des générations. En examinant les trajectoires produites (figure B.8), on constate que le début de l'évolution produit des comportements assez simples. Dans le cas présenté il s'agit uniquement d'un virage vers la droite. Toutefois on obtient à la fin de cette première phase des contrôleurs arrivant presque à reproduire la trajectoire enregistrée, mais pas suffisamment précis pour éviter les deux obstacles. La deuxième phase va ensuite optimiser ces contrôleurs qui au final sont capables d'atteindre les deux points cibles sans heurter d'obstacle, et même plus rapidement que lorsque le robot était guidé manuellement.

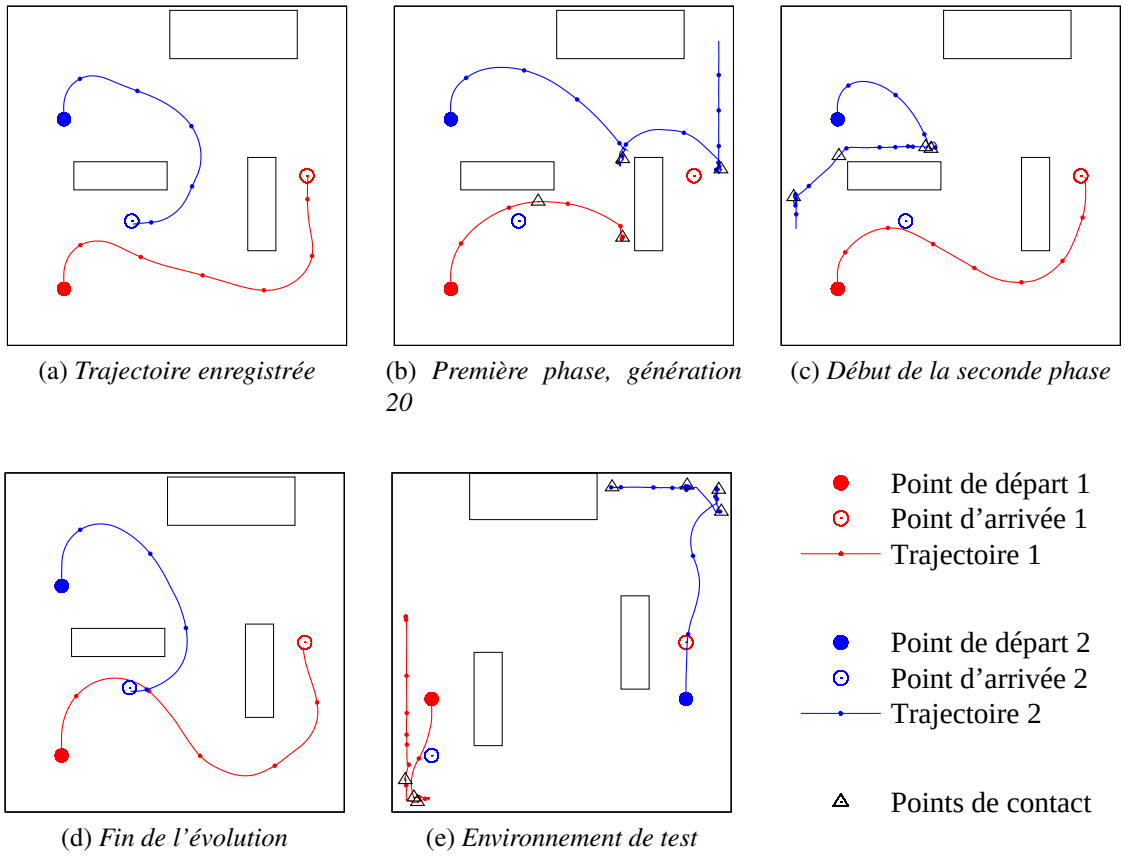


(a) *Première phase, évaluation basée sur l'imitation*



(b) *Deuxième phase, évaluation objective sur l'évitement d'obstacles*

FIG. B.7 : *Évolution au cours des générations des performances des meilleurs individus lors d'une évolution en deux phases dans l'environnement constitué d'étagères. Nous rappelons que pour la première phase, la fonction d'évaluation mesure la différence entre les commandes générées par l'algorithme testé et les commandes enregistrées.*



(a) *Trajectoire enregistrée* (b) *Première phase, génération 20* (c) *Début de la seconde phase*

(d) *Fin de l'évolution* (e) *Environnement de test*

FIG. B.8 : *Trajectoires suivies par des individus évolués grâce à l'évolution en deux phases. Les individus présentés sont sélectionnés à plusieurs étapes de l'évolution.*

Par contre, on constate en examinant la structure d'un individu évolué (figure B.9) que ceux-ci sont relativement complexes et surtout n'utilisent pas l'information de direction du but. Il s'agit ici clairement d'un cas où l'évolution a surappris la trajectoire à effectuer dans cet environnement. Ce comportement n'est pas générique, comme le montre la trajectoire obtenue dans l'environnement de test.

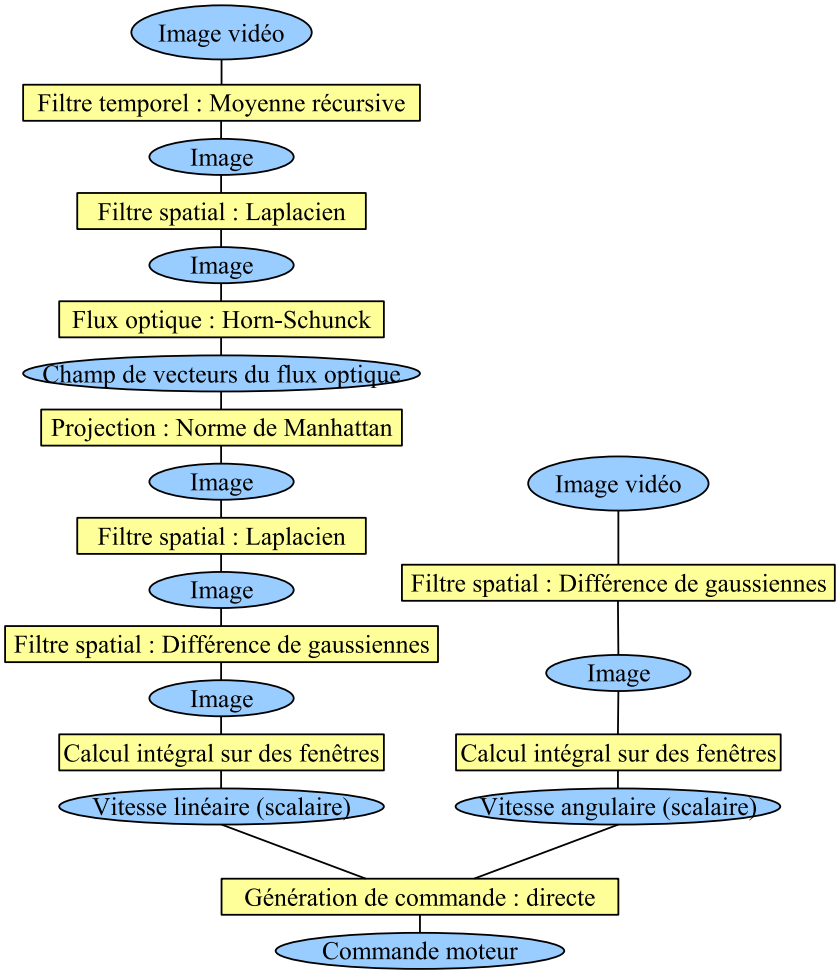
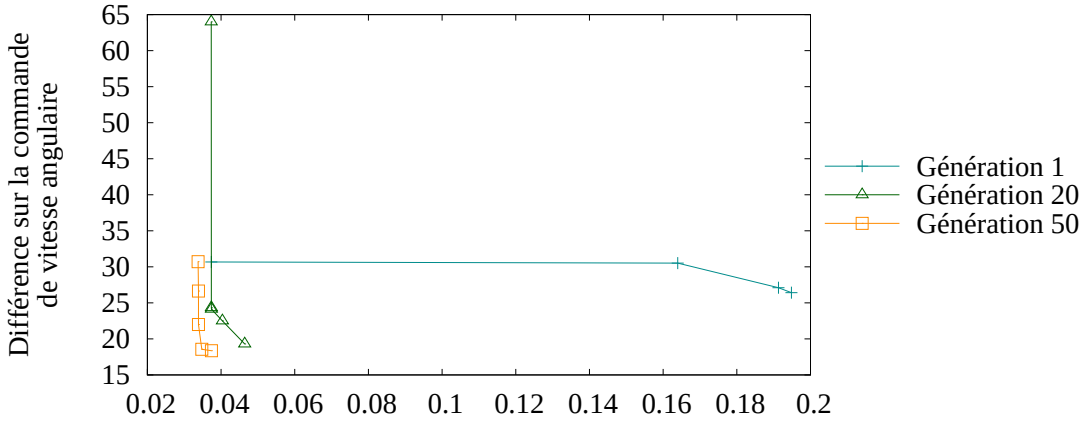


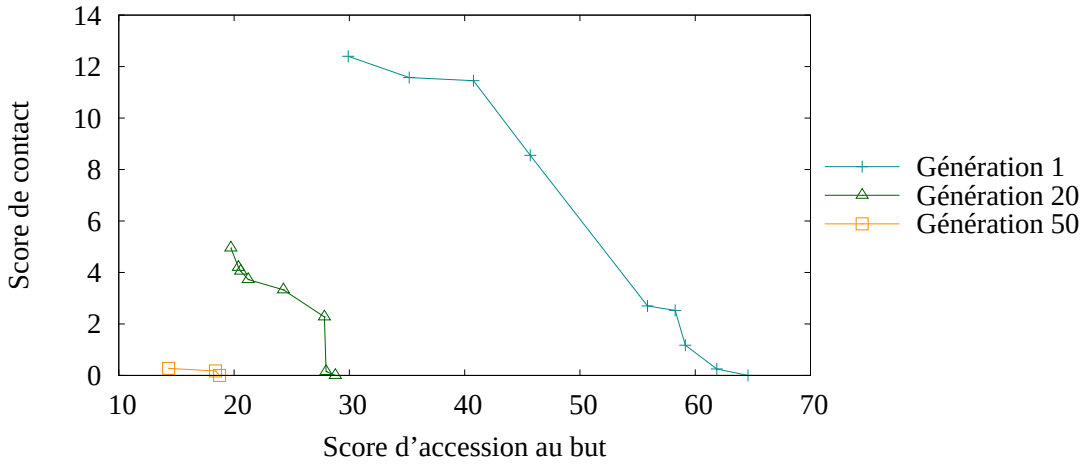
FIG. B.9 : *Algorithme utilisé par un contrôleur évolué grâce à l'évolution en deux phases.*

B.4 Évolution en deux phases avec quatre trajectoires

Cette expérience est effectuée dans les mêmes conditions que précédemment mais en utilisant quatre trajectoires au lieu de deux pour limiter les problèmes de surapprentissage. La figure B.10 montre l'évolution des performances au cours des générations et la figure B.11 montre les trajectoires correspondantes. On constate ici que l'évolution a plus de mal à trouver une trajectoire idéale et même les individus produits en fin d'expérience n'arrivent pas à atteindre le point cible dans les quatre configurations. Par contre, ce type de contrôleur est nettement plus générique que ceux obtenus précédemment. Dans l'environnement de test, il n'y a aucun vrai contact avec un obstacle.



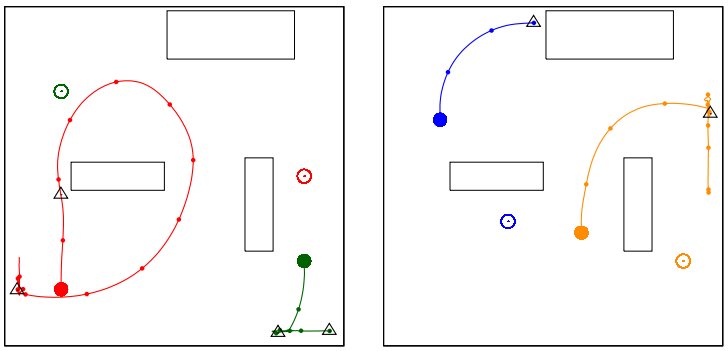
(a) *Première phase, évaluation basée sur l'imitation*



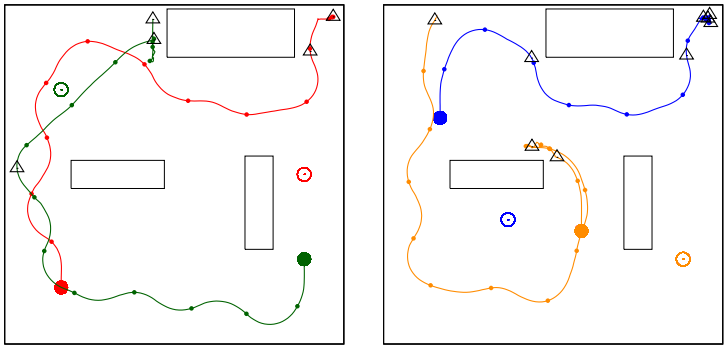
(b) *Deuxième phase, évaluation objective sur l'évitement d'obstacles*

FIG. B.10 : *Évolution au cours des générations des performances des meilleurs individus lors d'une évolution en deux phases de contrôleurs à structure libre dans l'environnement constitué d'étagères. Nous utilisons ici quatre trajectoires différentes pour limiter le phénomène de surapprentissage.*

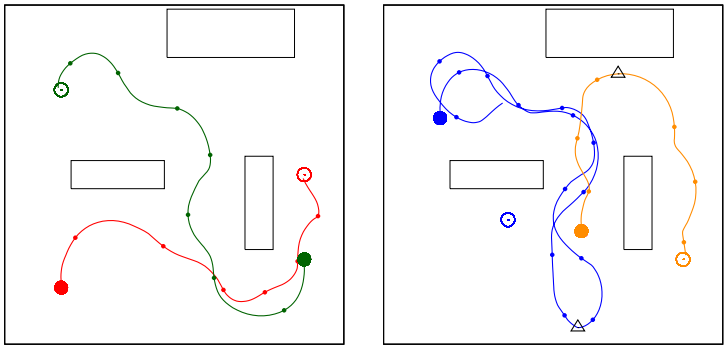
En examinant un algorithme évolué (figure B.12), on remarque que comme précédemment, il n'utilise pas l'information de direction du but. Il ne pourra donc trouver le point cible que par



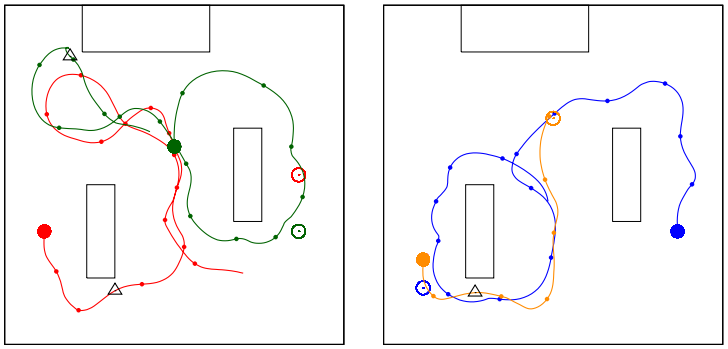
(a) *Première phase, génération 20*



(b) *Début de la seconde phase*



(c) *Fin de l'évolution*



(d) *Environnement de test*

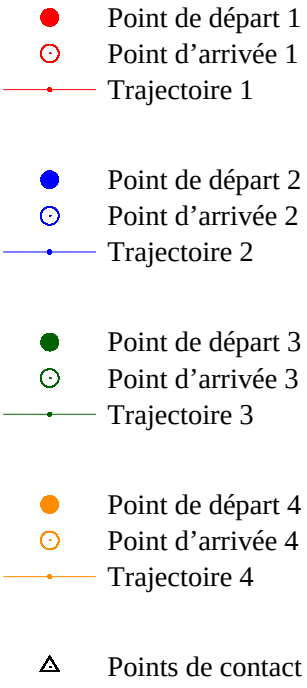


FIG. B.11 : *Trajectoires suivies par des individus évolués grâce à l'évolution en deux phases avec quatre trajectoires. Les individus présentés sont sélectionnés à plusieurs étapes de l'évolution.*

hasard. Il y a un réel problème avec ce type de structure pour produire un bon compromis entre l'évitement d'obstacles et le déplacement vers le but.

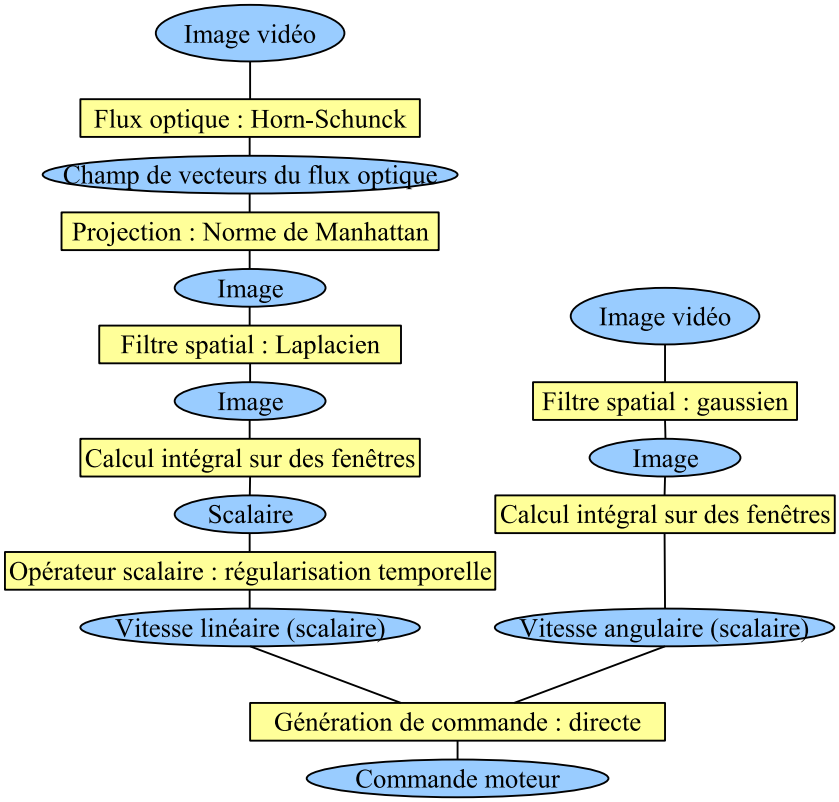
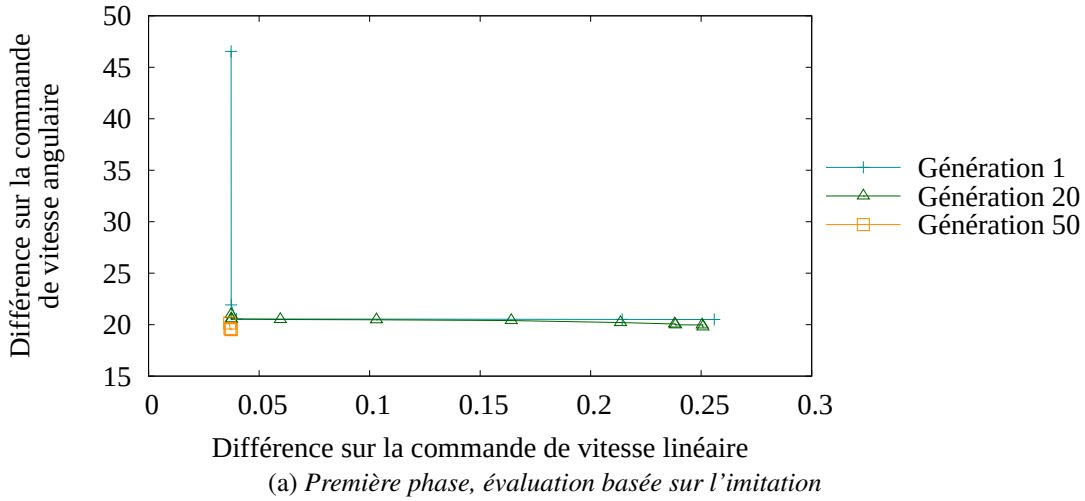


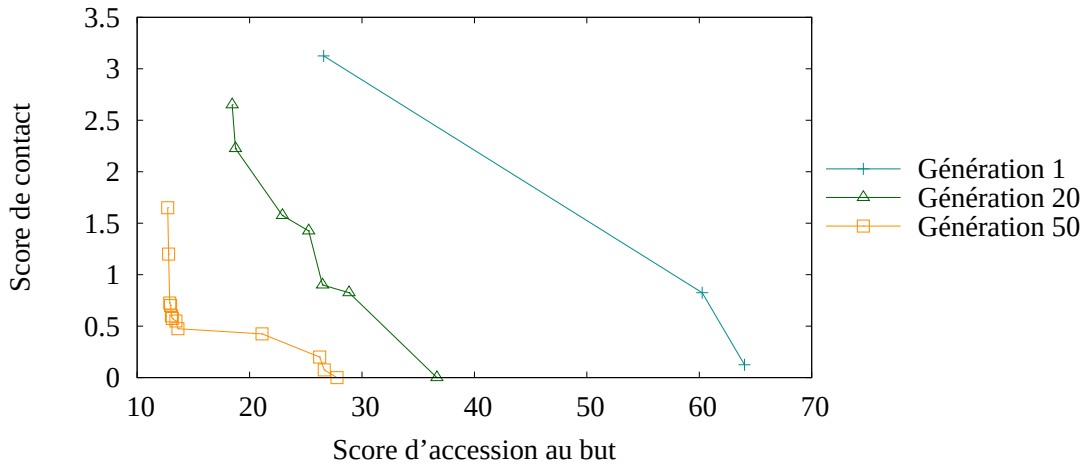
FIG. B.12 : *Algorithme utilisé par un contrôleur évolué grâce à l'évolution en deux phases avec quatre trajectoires.*

B.5 Évolution avec des algorithmes à structure restreinte

L'objectif de cette expérience est de montrer que l'utilisation d'une structure algorithmique spécifique peut faciliter le compromis entre évitement d'obstacles et déplacement vers le but. La figure B.13 montre l'évolution des performances des contrôleurs au cours des générations et la figure B.14 présente les trajectoires correspondantes. On constate ici que lors de la première phase la progression reste relativement faible. En effet, ce type de contrôleur utilise deux modes de fonctionnement différents ce qui rend plus difficile l'imitation d'une trajectoire enregistrée qui est toujours lisse. Ici la première phase n'apporte pas un avantage indéniable à l'évolution.



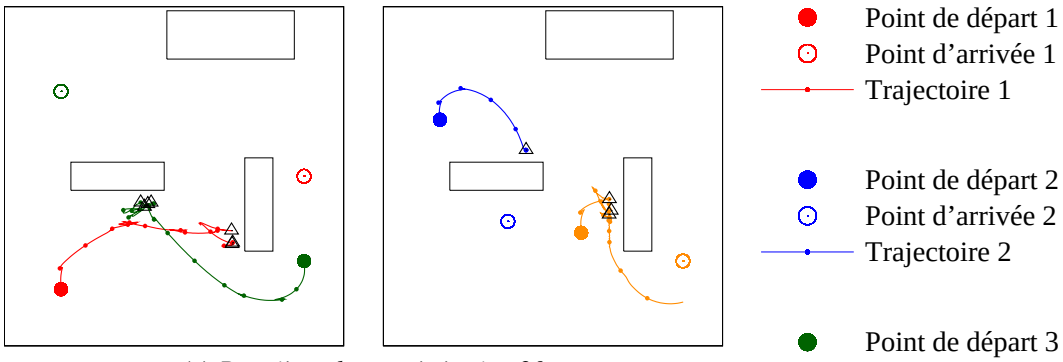
(a) *Première phase, évaluation basée sur l'imitation*



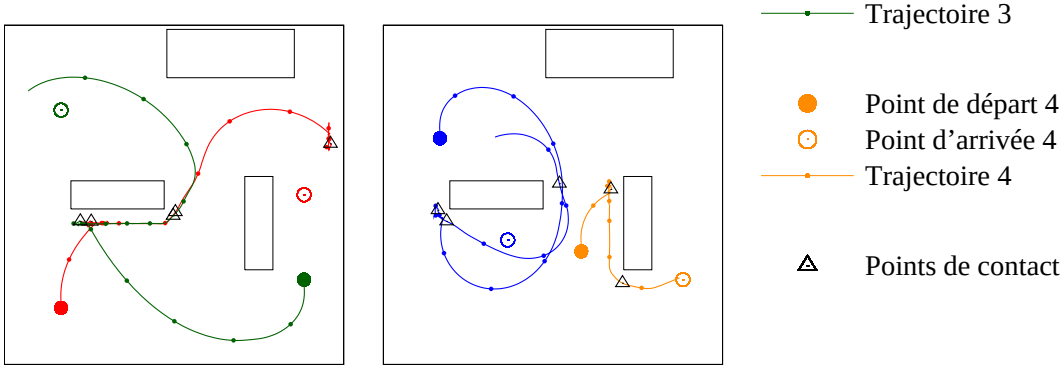
(b) *Deuxième phase, évaluation objective sur l'évitement d'obstacles*

FIG. B.13 : *Évolution au cours des générations des performances des meilleurs individus lors d'une évolution en deux phases de contrôleurs à structure restreinte dans l'environnement constitué d'étagères. Nous utilisons ici quatre trajectoires différentes pour limiter le phénomène de surapprentissage.*

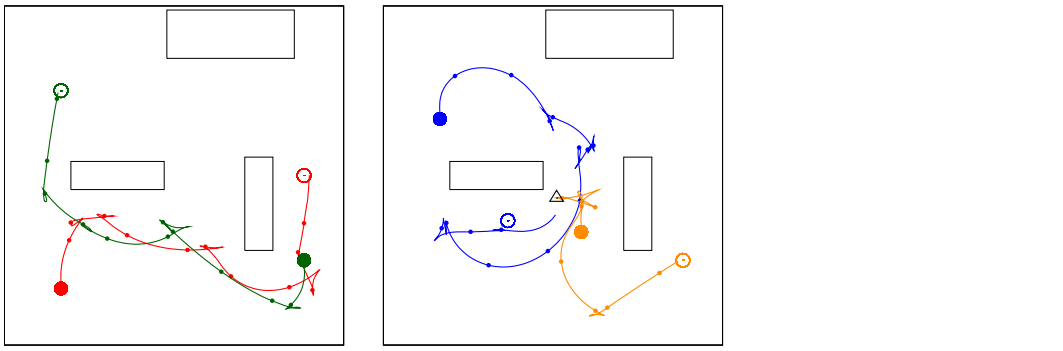
À la fin du processus d'évolution par contre, on obtient des contrôleurs très efficaces qui sont capables d'atteindre le point cible dans trois configurations sur quatre et sans heurter d'obstacle.



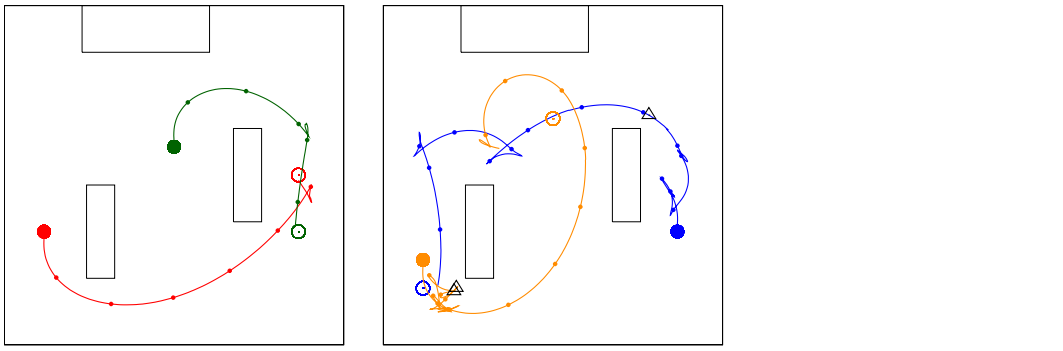
(a) *Première phase, génération 20*



(b) *Début de la seconde phase*



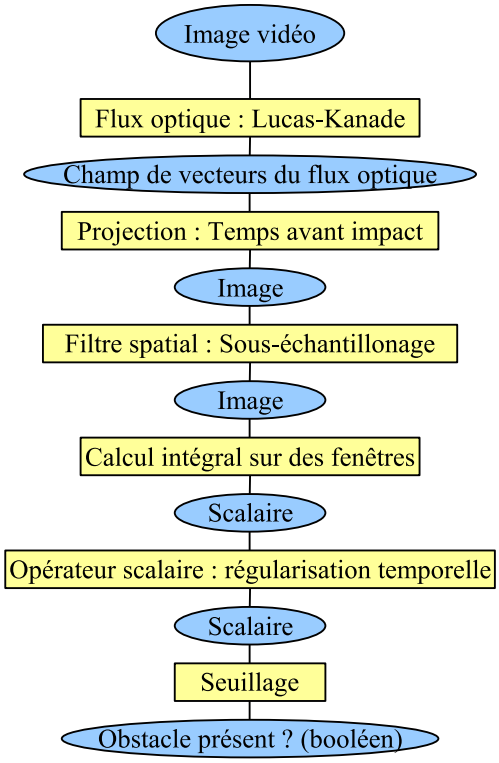
(c) *Fin de l'évolution*



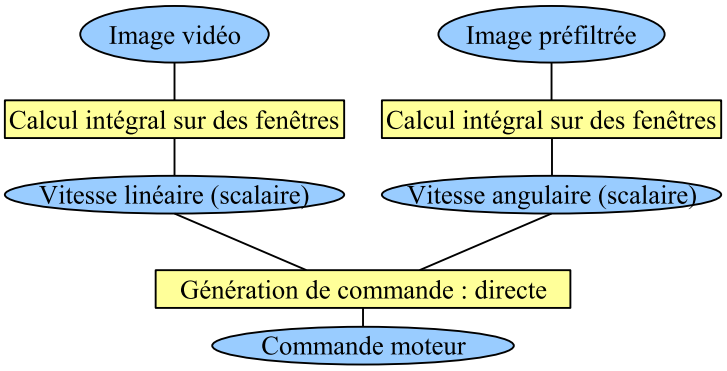
(d) *Environnement de test*

FIG. B.14 : *Trajectoires suivies par des individus évolués grâce à l'évolution en deux phases avec quatre trajectoires et une structure d'algorithme restreinte. Les individus présentés sont sélectionnés à plusieurs étapes de l'évolution.*

Plus intéressant encore, ce comportement reste très efficace en généralisation avec des performances quasiment identiques. L'algorithme utilisé ici est présenté sur la figure B.15. Il utilise un calcul de flux optique et une mesure de temps avant impact pour détecter les obstacles proches. L'algorithme d'évitement d'obstacles consiste en fait simplement à reculer dès qu'il y a un obstacle proche. Cela produit une trajectoire assez saccadée, ce qui est systématique en utilisant ce type de déplacement en deux modes, mais très efficace et généralisable.



(a) *Algorithme de détection d'obstacles*



(b) *Algorithme d'évitement d'obstacles*

FIG. B.15 : *Algorithme utilisé par un contrôleur évolué grâce à l'évolution en deux phases avec quatre trajectoires et une structure restreinte.*

Bibliographie

[Alba 02] E. Alba & M. Tomassini. *Parallelism and evolutionary algorithms*. IEEE Transactions on Evolutionary Computation, vol. 6, no. 5, pages 443–462, 2002.

[Angeli 08] A. Angeli, S. Doncieux, J.A. Meyer & D. Filliat. *Real-Time Visual Loop-Closure Detection*. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pages 1842–1847, 2008.

[Arleo 04] A. Arleo, F. Smeraldi & W. Gerstner. *Cognitive Navigation Based on Nonuniform Gabor Space Sampling, Unsupervised Growing Networks, and Reinforcement Learning*. IEEE Transactions on Neural Networks, vol. 15, no. 3, pages 639–652, 2004.

[Banzhaf 98] W. Banzhaf, P. Nordin, R. E. Keller & F. D. Francone. Genetic programming : An introduction. on the automatic evolution of computer programs and its applications. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[Barron 94] J.L. Barron, D.J. Fleet & S.S. Beauchemin. *Performance of optical flow techniques*. International Journal of Computer Vision, vol. 12, no. 1, pages 43–77, 1994.

[Bleuler 01] S. Bleuler, M. Brack, L. Thiele & E. Zitzler. *Multiobjective Genetic Programming : Reducing Bloat Using SPEA2*. In Proceedings of the 2001 Congress on Evolutionary Computation, volume 1, pages 536–543, 2001.

[Boots 07] B. Boots, S. Nundy & D. Purves. *Evolution of Visually Guided Behavior in Artificial Agents*. Network : Computation in Neural Systems, vol. 18, no. 1, pages 11–34, 2007.

[Chaumette 06] F. Chaumette & S. Hutchinson. *Visual Servo Control, Part I : Basic Approaches*. IEEE Robotics and Automation Magazine, vol. 13, no. 4, pages 82–90, 2006.

[Chen 06] J. Chen, W.E. Dixon, D.M. Dawson & M. McIntyre. *Homography-Based Visual Servo Tracking Control of a Wheeled Mobile Robot*. IEEE Transactions on Robotics, vol. 22, no. 2, pages 406–415, 2006.

[Cliff 97] D. Cliff, I. Harvey & P. Husbands. *Artificial Evolution of Visual Control Systems for Robots*. In From Living Eyes to Seeing Machines, pages 126–157. Oxford University Press, USA, 1997.

[Collet 99] P. Collet, E. Lutton, F. Raynal & M. Schoenauer. *Individual GP : an Alternative Viewpoint for the Resolution of Complex Problems*. In Proceedings

of the annual conference on Genetic and evolutionary computation, GECCO'99, pages 974–981, 1999.

[Coombs 98] D. Coombs, M. Herman, T.H. Hong & M. Nashman. *Real-Time Obstacle Avoidance Using Central Flow Divergence, and Peripheral Flow*. IEEE Transactions on Robotics and Automation, vol. 14, no. 1, pages 49–59, 1998.

[Corke 93] P.I. Corke. *Visual Control of Robot Manipulators - A Review*. In K. Hashimoto, editeur, Visual Servoing, pages 1–31. World Scientific, 1993.

[Corke 04] P. Corke, D. Strelow & S. Singh. *Omnidirectional Visual Odometry for a Planetary Rover*. In Proceedings of the International Conference on Intelligent Robots and Systems (IROS), volume 4, pages 4007–4012, 2004.

[Davison 03] A.J. Davison. *Real-time Simultaneous Localisation and Mapping with a Single Camera*. In Proceedings of the Ninth IEEE International Conference on Computer Vision, volume 2, pages 1403–1410, 2003.

[Davison 07] A.J. Davison, I.D. Reid, N.D. Molton & O. Stasse. *MonoSLAM : Real-Time Single Camera SLAM*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 29, no. 6, pages 1052–1067, 2007.

[de Jong 03] E.D. de Jong & J.B. Pollack. *Multi-Objective Methods for Tree Size Control*. Genetic Programming and Evolvable Machines, vol. 4, no. 3, pages 211–233, 2003.

[Deb 02] K. Deb, A. Pratap, S. Agarwal & T. Meyarivan. *A fast and elitist multiobjective genetic algorithm : NSGA-II*. IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pages 182–197, 2002.

[Desimone 95] R. Desimone & J. Duncan. *Neural Mechanisms of Selective Visual Attention*. Annual Reviews in Neuroscience, vol. 18, pages 193–222, 1995.

[DeSouza 02] G.N. DeSouza & A.C. Kak. *Vision for Mobile Robot Navigation : A Survey*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 2, pages 237–267, 2002.

[Dissanayake 01] M. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte & M. Csorba. *A Solution to the Simultaneous Localization and Map Building (SLAM) Problem*. IEEE Transactions on Robotics and Automation, vol. 17, no. 3, pages 229–241, 2001.

[Ebner 99] M. Ebner & A. Zell. *Evolving a task specific image operator*. In Proceedings of the First european workshop on evolutionary image analysis, signal processing and telecommunications (evoiasp), pages 74–89, 1999.

[Espiau 92] B. Espiau, F. Chaumette & P. Rives. *A New Approach to Visual Servoing in Robotics*. IEEE Transactions on Robotics and Automation, vol. 8, no. 3, pages 313–326, 1992.

[Filliat 07] D. Filliat. *A visual bag of words method for interactive qualitative localization and mapping*. In Proceedings of the International Conference on Robotics and Automation (ICRA), pages 3921–3926, 2007.

[Floreano 96] D. Floreano & F. Mondada. *Evolution of Homing Navigation in a Real Mobile Robot*. IEEE Transactions on Systems, Man and Cybernetics, Part B, vol. 26, no. 3, pages 396–407, 1996.

[Floreano 04] D. Floreano, T. Kato, D. Marocco & E. Sauser. *Coevolution of Active Vision and Feature Selection*. Biological Cybernetics, vol. 90, no. 3, pages 218–228, 2004.

[Fogel 66] L.J. Fogel, A.J. Owens & M.J. Walsh. Artificial Intelligence Through Simulated Evolution. John Wiley & Sons Inc, 1966.

[Fonseca 95] C.M. Fonseca & P.J. Fleming. *An Overview of Evolutionary Algorithms in Multiobjective Optimization*. Evolutionary Computation, vol. 3, no. 1, pages 1–16, 1995.

[Gagné 06] C. Gagné, M. Schoenauer, M. Parizeau & M. Tomassini. *Genetic Programming, Validation Sets, and Parsimony Pressure*. In Proceedings of EuroGP 2006, volume 3905 of *Lecture Notes in Computer Science*, pages 109–120. Springer, 2006.

[Gaussier 02] P. Gaussier, A. Revel, J.P. Banquet & V. Babeau. *From View Cells and Place Cells to Cognitive Map Learning : Processing Stages of the Hippocampal System*. Biological Cybernetics, vol. 86, no. 1, pages 15–28, 2002.

[Giovannangeli 06] C. Giovannangeli, P. Gaussier & J.P. Banquet. *Robustness of Visual Place Cells in Dynamic Indoor and Outdoor Environment*. International Journal of Advanced Robotic Systems, vol. 3, no. 2, pages 115–124, 2006.

[Gomez 97] F. Gomez & R. Miikkulainen. *Incremental Evolution of Complex General Behavior*. Adaptive Behavior, vol. 5, no. 3-4, pages 317–342, 1997.

[Harvey 94] I. Harvey, P. Husbands & D. Cliff. *Seeing the Light : Artificial Evolution, Real Vision*. In From Animals to Animats 3 : Proceedings of the Third International Conference on Simulation of Adaptive Behavior, pages 392–401. MIT Press, 1994.

[Holland 75] J. Holland. Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor, 1975.

[Horn 81] B.K.P. Horn & B.G. Schunck. *Determining Optical Flow*. Artificial Intelligence, vol. 17, pages 185–203, 1981.

[Horswill 93] I. Horswill. *Polly : A vision-based artificial agent*. In Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93), pages 824–829, 1993.

[Hrabar 06] S. Hrabar. *Vision-Based 3D Navigation for an Autonomous Helicopter*. PhD thesis, University of Southern California, 2006.

[Iida 03] F. Iida. *Biologically Inspired Visual Odometer for Navigation of a Flying Robot*. Robotics and Autonomous Systems, vol. 44, no. 3-4, pages 201–208, 2003.

[Jakobi 95] N. Jakobi, P. Husbands & I. Harvey. *Noise and the Reality Gap : The Use of Simulation in Evolutionary Robotics*. In Proceedings of the Third European Conference on Advances in Artificial Life, pages 704–720, 1995.

[Koza 92] J.R. Koza. Genetic programming : On the programming of computers by natural selection. MIT Press, Cambridge, MA, USA, 1992.

[Krink 99] T. Krink, B.H. Mayoh & Z. Michalewicz. *A PATCHWORK model for evolutionary algorithms with structured and variable size populations*. Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2, pages 1321–1328, 1999.

[Krink 01] T. Krink & R. Thomsen. *Self-organized criticality and mass extinction in evolutionary algorithms*. Proceedings of the 2001 Congress on Evolutionary Computation, vol. 2, pages 1155–1161, 2001.

[Laumanns 02] M. Laumanns, L. Thiele, K. Deb & E. Zitzler. *Combining Convergence and Diversity in Evolutionary Multiobjective Optimization*. Evolutionary Computation, vol. 10, no. 3, pages 263–282, 2002.

[Lettvin 59] J.Y. Lettvin, H.R. Maturana, W.S. McCulloch & W.H. Pitts. *What the Frog's Eye Tells the Frog's Brain*. Proceedings of the IRE, vol. 47, no. 11, pages 1940–1951, 1959.

[Lorigo 97] L.M. Lorigo, R.A. Brooks & W.E.L. Grimson. *Visually-guided obstacle avoidance in unstructured environments*. In Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems, volume 1, pages 373–379, 1997.

[Louchet 02] J. Louchet, M. Guyon, M.J. Lesot & A. Boumaza. *Dynamic flies : a new pattern recognition tool applied to stereo sequence processing*. Pattern Recognition Letters, vol. 23, no. 1-3, pages 335–345, 2002.

[Lowe 04] D.G. Lowe. *Distinctive Image Features from Scale-Invariant Keypoints*. International Journal of Computer Vision, vol. 60, no. 2, pages 91–110, 2004.

[Lucas 81] B.D. Lucas & T. Kanade. *An iterative image registration technique with an application to stereo vision*. In Proc. DARPA Image Understanding Workshop, pages 121–130, 1981.

[Mariottini 07] G.L. Mariottini, G. Oriolo & D. Prattichizzo. *Image-Based Visual Servoing for Nonholonomic Mobile Robots Using Epipolar Geometry*. IEEE Transactions on Robotics, vol. 23, no. 1, pages 87–100, 2007.

[Marocco 02] D. Marocco & D. Floreano. *Active Vision and Feature Selection in Evolutionary Behavioral Systems*. In From Animals to Animats 7 : Proceedings of the Seventh International Conference on Simulation of Adaptive Behavior, pages 247–255. MIT Press, 2002.

[Martin 06] M.C. Martin. *Evolving visual sonar : Depth from monocular images*. Pattern Recognition Letters, vol. 27, no. 11, pages 1174–1180, 2006.

[Matarić 96] M. Matarić & D. Cliff. *Challenges in Evolving Controllers for Physical Robots*. Robotics and Autonomous Systems, vol. 19, no. 1, pages 67–83, 1996.

[Meyer 91] J.A. Meyer & A. Guillot. *Simulation of Adaptive Behavior in Animats : Review and Prospect*. In From Animals to Animats, Proceedings of the

First International Conference on the Simulation of Adaptive Behavior, pages 2–14. The MIT Press, 1991.

[Michels 05] J. Michels, A. Saxena & A.Y. Ng. *High speed obstacle avoidance using monocular vision and reinforcement learning*. In Proceedings of the 22nd international conference on Machine learning, pages 593–600. ACM Press New York, NY, USA, 2005.

[Montana 95] D.J. Montana. *Strongly Typed Genetic Programming*. Evolutionary Computation, vol. 3, no. 2, pages 199–230, 1995.

[Montemerlo 02] M. Montemerlo, S. Thrun, D. Koller & B. Wegbreit. *FastSLAM : A Factored Solution to the Simultaneous Localization and Mapping Problem*. In Proceedings of the Eighteenth National Conference on Artificial intelligence, pages 593–598, 2002.

[Mouret 08] J.-B. Mouret & S. Doncieux. *Incremental Evolution of Animats' Behaviors as a Multi-objective Optimization*. In From Animals to Animats 10, Proceedings of the tenth International Conference on the Simulation of Adaptive Behavior, SAB'08, volume 5040 of *LNAI*, pages 210–219. Springer, 2008.

[Muratet 05] L. Muratet, S. Doncieux, Y. Brière & J.-A. Meyer. *A Contribution to Vision-Based Autonomous Helicopter Flight in Urban Environments*. Robotics and Autonomous Systems, vol. 50, no. 4, pages 195–209, 2005.

[O'Keefe 78] J. O'Keefe & L. Nadel. The Hippocampus as a Cognitive Map. Clarendon Press, 1978.

[Olague 06] G. Olague & C. Puente. *Parisian evolution with honeybees for three-dimensional reconstruction*. In Proceedings of the 8th annual conference on Genetic and evolutionary computation, pages 191–198. ACM Press New York, NY, USA, 2006.

[Paillet 99] F. Paillet, D. Mercier & T.M. Bernard. *Second Generation Programmable Artificial Retina*. In Proceedings of the Twelfth Annual IEEE International ASIC/SOC Conference, pages 304–309, 1999.

[Pauplin 05] O. Pauplin, J. Louchet, E. Lutton & A. De La Fortelle. *Evolutionary Optimisation for Obstacle Detection and Avoidance in Mobile Robotics*. Journal of Advanced Computational Intelligence and Intelligent Informatics, vol. 9, no. 6, pages 622–629, 2005.

[Poli 08] R. Poli & N.F. McPhee. *Parsimony pressure made easy*. In Proceedings of the 10th annual conference on Genetic and evolutionary computation, GECCO'08, pages 1267–1274. ACM, 2008.

[Pollefeys 99] M. Pollefeys, R. Koch & L. Van Gool. *Self-Calibration and Metric Reconstruction Inspite of Varying and Unknown Intrinsic Camera Parameters*. International Journal of Computer Vision, vol. 32, no. 1, pages 7–25, 1999.

[Rechenberg 73] I. Rechenberg. Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart-Bad Cannstatt, 1973.

[Ross 01] B.J. Ross. *Logic-based Genetic Programming with Definite Clause Translation Grammars*. New Generation Computing, vol. 19, no. 4, pages 313–337, 2001.

[Santos-Victor 93] J. Santos-Victor, G. Sandini, F. Curotto & S. Garibaldi. *Divergent stereo for robot navigation : learning from bees*. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 434–439, 1993.

[Santos-Victor 95] J. Santos-Victor, G. Sandini, F. Curotto & S. Garibaldi. *Divergent Stereo in Autonomous Navigation : From Bees to Robots*. International Journal of Computer Vision, vol. 14, no. 2, pages 159–177, 1995.

[Sareni 98] B. Sareni & L. Krähenbühl. *Fitness sharing and niching methods revisited*. IEEE Transactions on Evolutionary Computation, vol. 2, no. 3, pages 97–106, 1998.

[Saxena 08] A. Saxena, S.H. Chung & A.Y. Ng. *3-D Depth Reconstruction from a Single Still Image*. International Journal of Computer Vision, vol. 76, no. 1, pages 53–69, 2008.

[Schaal 03] S. Schaal, A. Ijspeert & A. Billard. *Computational approaches to motor learning by imitation*. Philosophical Transactions of the Royal Society B : Biological Sciences, vol. 358, pages 537–547, 2003.

[Schaffer 85] J.D. Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. In Proceedings of the 1st International Conference on Genetic Algorithms, pages 93–100, 1985.

[Shi 94] J. Shi & C. Tomasi. *Good Features to Track*. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR'94, pages 593–600, 1994.

[Soule 98] T. Soule & J.A. Foster. *Effects of Code Growth and Parsimony Pressure on Populations in Genetic Programming*. Evolutionary Computation, vol. 6, no. 4, pages 293–309, 1998.

[Srinivas 94] N. Srinivas & K. Deb. *Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms*. Evolutionary Computation, vol. 2, no. 3, pages 221–248, 1994.

[Srinivasan 96] M.V. Srinivasan, S.W. Zhang, M. Lehrer & T.S. Collett. *Honeybee Navigation en Route to the Goal : Visual Flight Control and Odometry*. Journal of Experimental Biology, vol. 199, no. 1, pages 237–244, 1996.

[Suzuki 07] M. Suzuki. *Enactive Robot Vision*. PhD thesis, École Polytechnique Fédérale de Lausanne (EPFL), 2007.

[Thomsen 00] R. Thomsen, P. Rickers & T. Krink. *A Religion-Based Spatial Model For Evolutionary Algorithms*. Parallel Problem Solving from Nature VI (PPSN-2000), vol. 1, pages 817–826, 2000.

[Tomassini 99] M. Tomassini. *Parallel and distributed evolutionary algorithms : A review*. In K. Miettinen, P. Neittaanmäki, M.M. Mäkelä & J. Périaux, editeurs, Evolutionary Algorithms in Engineering and Computer Science, pages 113–133. J. Wiley, 1999.

[Trujillo 06] L. Trujillo & G. Olague. *Synthesis of interest point detectors through genetic programming*. In Proceedings of the 8th annual conference on Genetic and evolutionary computation, pages 887–894. ACM Press, 2006.

[Trujillo 08] L. Trujillo, G. Olague, E. Lutton & F.F. de Vega. *Multiobjective design of operators that detect points of interest in images*. In Proceedings of the 10th annual conference on Genetic and evolutionary computation, GECCO'08, pages 1299–1306. ACM Press, 2008.

[Ulrich 00] I. Ulrich & I. Nourbakhsh. *Appearance-Based Obstacle Detection with Monocular Color Vision*. In Proceedings of AAAI Conference, pages 866–871, 2000.

[Ungerleider 94] L.G. Ungerleider & J.V. Haxby. *'What' and 'where' in the human brain*. Current Opinion in Neurobiology, vol. 4, no. 2, pages 157–165, 1994.

[Ursem 99] RK Ursem. *Multinational evolutionary algorithms*. Proceedings of the 1999 Congress on Evolutionary Computation, vol. 3, pages 1633–1640, 1999.

[Vidal 04] R. Vidal, O. Shakernia & S. Sastry. *Distributed Formation Control with Omnidirectional Vision-Based Motion Segmentation and Visual Servoing*. IEEE Robotics and Automation Magazine, vol. 11, no. 4, pages 14–20, 2004.

[Walker 03] J. Walker, S. Garrett & M. Wilson. *Evolving Controllers for Real Robots : A Survey of the Literature*. Adaptive Behavior, vol. 11, no. 3, pages 179–203, 2003.

[Welch 95] G. Welch & G. Bishop. *An Introduction to the Kalman Filter*. Rapport technique, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1995.

[Whigham 95] P.A. Whigham. *Grammatically-based genetic programming*. In Proceedings of the Workshop on Genetic Programming : From Theory to Real-World Applications, pages 33–41, 1995.

[Whigham 96] P.A. Whigham. *Search Bias, Language Bias and Genetic Programming*. In John R. Koza, David E. Goldberg, David B. Fogel & Rick L. Riolo, editeurs, Proceedings of the First Annual Conference on Genetic Programming, pages 230–237, Stanford University, CA, USA, 1996. MIT Press.

[Wilson 91] S.W. Wilson. *The Animat Path to AI*. In From Animals to Animats, Proceedings of the First International Conference on the Simulation of Adaptive Behavior, pages 15–21. The MIT Press, 1991.

[Wong 97] M.L. Wong & K.S. Leung. *Evolutionary Program Induction Directed by Logic Grammars*. Evolutionary Computation, vol. 5, no. 2, pages 143–180, 1997.

[Zitzler 99] E. Zitzler & L. Thiele. *Multiobjective Evolutionary Algorithms : A Comparative Case Study and the Strength Pareto Approach*. IEEE Transactions on Evolutionary Computation, vol. 3, no. 4, pages 257–271, 1999.

[Zufferey 06] J.C. Zufferey & D. Floreano. *Fly-Inspired Visual Steering of an Ultralight Indoor Aircraft*. IEEE Transactions on Robotics, vol. 22, no. 1, pages 137–146, 2006.

Articles publiés dans le cadre de cette thèse

Les travaux présentés dans cette thèse ont donné lieu à publication dans quatre conférences internationales. Les références des articles correspondants sont :

- R. Barate & A. Manzanera. *Automatic Design of Vision-Based Obstacle Avoidance Controllers using Genetic Programming*. In Proceedings of the 8th International Conference on Artificial Evolution, EA 2007, vol. 4926 of LNCS, pages 25–36. Springer, 2008.
- R. Barate & A. Manzanera. *Evolving Vision Controllers with a Two-Phase Genetic Programming System Using Imitation*. In From Animals to Animats 10, Proceedings of the tenth International Conference on the Simulation of Adaptive Behavior, SAB'08, vol. 5040 of LNAI, pages 73–82. Springer, 2008.
- R. Barate & A. Manzanera. *Generalization Performance of Vision Based Controllers for Mobile Robots Evolved with Genetic Programming*. In Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2008, pages 1331–1332. ACM Press, 2008.
- R. Barate & A. Manzanera. *Learning Vision Algorithms for Real Mobile Robots with Genetic Programming*. In Proceedings of the ECSIS Symposium on Learning and Adaptive Behavior in Robotic Systems, LAB-RS 2008, pages 47–52. IEEE computer society, 2008.