

Éléments de robotique

Mordechai (Moti) Ben-Ari
Francesco Mondada

© Moti Ben-Ari and Francesco Mondada, 2022

This work is licensed under the Creative Commons Attribution 4.0 International. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

Pour Itay, Sahar and Nofar.
Mordechai Ben-Ari

Pour Luca, Nora, and Leonardo.
Francesco Mondada

Preface

La robotique est un domaine dynamique qui gagne en importance d'année en année. C'est également un sujet que les élèves apprécient à tous les niveaux, de la maternelle aux études supérieures. L'objectif de l'apprentissage de la robotique varie en fonction de la tranche d'âge. Pour les jeunes enfants, les robots sont des jouets éducatifs ; pour les collégiens et les lycéens, la robotique peut accroître la motivation des étudiants à étudier les STIM (sciences, technologie, ingénierie, mathématiques) ; au niveau de l'introduction à l'université, les étudiants peuvent apprendre comment la physique, les mathématiques et l'informatique qu'ils étudient peuvent être appliquées à des projets d'ingénierie pratiques ; enfin, les étudiants de premier cycle et les étudiants diplômés se préparent à des carrières en robotique.

Ce livre s'adresse au milieu de la tranche d'âge : les étudiants des écoles secondaires et des premières années d'université. Nous nous concentrons sur les algorithmes robotiques et leurs principes mathématiques et physiques. Nous allons au-delà du jeu d'essais et d'erreurs, mais nous ne nous attendons pas à ce que l'étudiant soit capable de concevoir et de construire des robots et des algorithmes robotiques qui exécutent des tâches dans le monde réel. La présentation des algorithmes sans mathématiques et ingénierie avancées est nécessairement simplifiée, mais nous pensons que les concepts et algorithmes de la robotique peuvent être appris et appréciés à ce niveau, et peuvent servir de passerelle vers l'étude de la robotique aux niveaux avancés du premier et du deuxième cycle universitaire.

Le bagage requis est une connaissance de la programmation, des mathématiques et de la physique au niveau des écoles secondaires ou de la première année d'université. En mathématiques : algèbre, trigonométrie, calcul, matrices et probabilités. L'annexe B fournit des didacticiels pour certaines des mathématiques les plus avancées. En physique : temps, vitesse, accélération, force et friction.

Il ne se passe pas un jour sans qu'apparaisse un nouveau robot destiné à des fins éducatives. Quelles que soient la forme et la fonction d'un robot, les principes et les algorithmes scientifiques et techniques restent les mêmes. C'est pourquoi ce livre n'est pas basé sur un robot spécifique. Au Chap. 1, nous définissons un robot générique : un petit robot mobile autonome doté d'un entraînement différentiel et de capteurs capables de détecter la direction et la distance d'un objet, ainsi que de capteurs au sol pouvant détecter des marques sur une table ou un sol. Cette définition est suffisamment générale pour que les élèves soient en mesure d'implémenter la plupart des algorithmes sur n'importe quel robot éducatif. La qualité de l'implémentation peut varier en fonction des capacités de chaque plateforme, mais les étudiants seront en mesure d'apprendre les principes de la robotique et comment passer des algorithmes théoriques au comportement d'un robot réel.

Pour des raisons similaires, nous avons choisi de ne pas décrire les algorithmes dans un langage de programmation spécifique. Non seulement les différentes plateformes supportent différents langages, mais les robots éducatifs utilisent souvent différentes approches de programmation, comme la programmation textuelle et la programmation visuelle utilisant des blocs ou des états. Nous présentons les algorithmes en pseudocode et laissons aux élèves le soin de mettre en œuvre ces descriptions de haut niveau dans le langage et l'environnement du robot qu'ils utilisent.

Le livre contient un grand nombre d'*activités*, dont la plupart vous demandent d'implémenter des algorithmes et d'explorer leur comportement. Le robot que vous utilisez n'a peut-être pas les capacités de réaliser toutes les activités, alors n'hésitez pas à les adapter à votre robot.

Ce livre est né du développement de matériel pédagogique pour le robot éducatif Thymio (<https://www.thymio.org>). Le site Web du livre <http://elementsofrobotics.net> contient des mises en œuvre de la plupart des activités pour ce robot. Certains des algorithmes les plus avancés étant difficiles à mettre en œuvre sur des robots éducatifs, des programmes Python sont fournis. Si vous implémentez les activités pour d'autres robots éducatifs, faites-le nous savoir et nous publierons un lien sur le site Web du livre.

Le chapitre 1 présente une vue d'ensemble du domaine de la robotique et précise le robot générique et le pseudocode utilisé dans les algorithmes. Les chapitres 2–6 présentent les concepts fondamentaux des robots mobiles autonomes : capteurs, comportement réactif, machines à états finis, mouvement et odométrie, et contrôle. Les chapitres 7–16 décrivent des algorithmes robotiques plus avancés : évitement d'obstacles, localisation, cartographie, logique floue, traitement d'images, réseaux neuronaux, apprentissage automatique, robotique en essaim et cinématique des manipulateurs robotiques. Un aperçu détaillé du contenu est donné dans la section 1.8.

Remerciements :

Ce livre est le fruit d'un travail sur le robot Thymio et le système logiciel Aseba initié par le groupe de recherche du second auteur au Laboratoire des Systèmes Robotiques de l'Ecole Polytechnique Fédérale de Lausanne. Nous tenons à remercier tous les étudiants, ingénieurs, enseignants et artistes de la communauté Thymio sans lesquels ce livre n'aurait pu être écrit.

Le libre accès à ce livre a été soutenu par l'Ecole Polytechnique Fédérale de Lausanne et le Pôle de Recherche National (PRN) Robotique.

Nous sommes redevables à Jennifer S. Kay, Fanny Riedo, Amaury Dame et Yves Piguet pour leurs commentaires qui nous ont permis de corriger des erreurs et de clarifier la présentation.

Nous tenons à remercier le personnel de Springer, en particulier Helen Desmond

et Beverley Ford, pour leur aide et leur soutien.

Rehovot
Lausanne

Moti Ben-Ari
Francesco Mondada

Table des matières

1 Robots et leurs applications	15
1.1 Classification des robots	15
1.2 Robots industriels	17
1.3 Robots mobiles autonomes	19
1.4 Robots humanoïdes	21
1.5 Robots éducatifs	21
1.6 Le robot générique	24
1.7 Le formalisme algorithmique	29
1.8 Un aperçu du contenu du livre	30
1.9 Summary	34
1.10 Lectures complémentaires	35
2 Capteurs	37
2.1 Classification des capteurs	38
2.2 Capteurs de distance	38
2.3 Caméras	46
2.4 Autres capteurs	48
2.5 Etendue de mesure, résolution, précision, exactitude	49
2.6 Nonlinearity	51
2.7 Summary	54
2.8 Further reading	54
3 Comportement réactif	55
3.1 Véhicules Braitenberg	55
3.2 Réaction à la détection d'un objet	56
3.3 Réaction et virage	58
3.4 Ligne suivante	60
3.5 Présentation des véhicules par Braitenberg	67
3.6 Summary	69
3.7 Further reading	69
4 Machines à états finis	71
4.1 Machines à états	71
4.2 Comportement réactif avec état	72
4.3 Recherche et approche	73
4.4 Mise en œuvre des automates à états finis	74
4.5 Summary	77

4.6 Lectures complémentaires	77
5 Mouvement robotique et odométrie	79
5.1 Distance, vitesse et temps	80
5.2 Accélération en tant que changement de vitesse	81
5.3 Des segments au mouvement continu	83
5.4 Navigation par odométrie	85
5.5 Odométrie linéaire	85
5.6 Odométrie avec tours	86
5.7 Erreurs en odométrie	90
5.8 Codeurs de roue	92
5.9 Systèmes de navigation inertielles	94
5.10 Degrés de liberté et nombre d'actionneurs	97
5.11 Le nombre relatif d'actionneurs et de DOF	99
5.12 Mouvement holonomique et non holonomique	105
5.13 Résumé	109
5.14 Lectures complémentaires	110
6 Control	111
6.1 Modèles de contrôle	111
6.2 Contrôle on-off	115
6.3 Contrôleur proportionnel (P)	117
6.4 Contrôleur proportionnel-intégral (PI)	121
6.5 Contrôleur proportionnel-intégral-dérivé (PID)	122
6.6 Résumé	124
6.7 Lectures complémentaires	125
7 Navigation locale : Évitement d'obstacles	127
7.1 Évitement des obstacles	128
7.2 Suivre une ligne avec un code	133
7.3 Fourmis à la recherche de la nourriture	134
7.4 Un modèle probabiliste	138
7.5 Une machine pour la recherche de chemin	141
7.6 Résumé	143
7.7 Lecture complémentaire	144
8 Localisation	145
8.1 Landmarks	145
8.2 Objets dont la position est connue	146
8.3 Système de positionnement global	149
8.4 Localisation probabiliste	150
8.5 Incertitude du mouvement	155

<i>TABLE DES MATIÈRES</i>	11
8.6 Résumé	157
8.7 Lecture complémentaire	157
9 Cartographie	159
9.1 Cartes discrètes et continues	160
9.2 Le contenu des cellules d'une carte en grille	161
9.3 L'algorithme de frontière	163
9.4 La connaissance de l'environnement	169
9.5 Exemple numérique d'un algorithme SLAM	171
9.6 Activités de démonstration de l'algorithme SLAM	178
9.7 La formalisation de l'algorithme SLAM	180
9.8 Résumé	182
9.9 Lecture complémentaire	182
10 Navigation basée sur des cartes	185
10.1 Algorithme de Dijkstra pour une carte quadrillée	185
10.2 Algorithme de Dijkstra pour une carte continue	190
10.3 Planification de trajectoire avec l'algorithme A*	193
10.4 Path following and obstacle avoidance	198
10.5 Résumé	199
10.6 Lecture complémentaire	199
11 Commande à logique floue	201
11.1 Fuzzify	201
11.2 Appliquer les règles	202
11.3 Defuzzifier	203
11.4 Résumé	205
11.5 Lecture complémentaire	205
12 Traitement des Images	207
12.1 Obtention d'images	208
12.2 Une vue d'ensemble du traitement numérique des images	209
12.3 Amélioration d'image	210
12.4 Détection des contours	216
12.5 Détection des coins	219
12.6 Reconnaissance des blobs	221
12.7 Résumé	223
12.8 Lecture complémentaire	224

13 Réseaux neuronaux	225
13.1 Le système neuronal biologique	225
13.2 Le modèle de réseau neuronal artificiel	226
13.3 Mise en œuvre d'un véhicule de Braintenberg avec un ANN	229
13.4 Réseaux neuronaux artificiels : topologies	231
13.5 Apprentissage	236
13.6 Résumé	242
13.7 Lecture complémentaire	243
14 Apprentissage machine	245
14.1 Distinction entre deux couleurs	246
14.2 Analyse discriminante linéaire	251
14.3 Généralisation du discriminant linéaire	266
14.4 Perceptrons	267
14.5 Résumé	275
14.6 Lecture complémentaire	276
15 Robotique en essaim	277
15.1 Approches de la mise en œuvre de la collaboration entre robots	278
15.2 Coordination par échange local d'informations	279
15.3 Robot basée sur les interactions physiques	284
15.4 Résumé	290
15.5 Lecture complémentaire	291
16 Kinématique d'un Manipulateur Robotique	293
16.1 Cinématique de l'avant	294
16.2 Cinématique inverse	296
16.3 Rotations	300
16.4 Rotation et translation d'un cadre de coordonnées	306
16.5 Un goût de rotations tridimensionnelles	309
16.6 Les transformations tridimensionnelles	316
16.7 Résumé	317
16.8 Lecture complémentaire	318
A Unités de mesure	319
B Dérivations mathématiques et tutoriels	321
B.1 Probabilité conditionnelle et règle de Bayes	321
B.2 Normalisation	322
B.3 Moyenne et variance	322
B.4 Covariance	324
B.5 Multiplication des vecteurs et des matrices	325

<i>TABLE DES MATIÈRES</i>	13
B.6 L'aire d'un trapèze à la base d'un triangle	326
B.7 Formules algébriques pour $\cos 15^\circ$	327

Chapitre 1

Robots et leurs applications

Bien que tout le monde semble savoir ce qu'est un robot, il est difficile d'en donner une définition précise. L'Oxford English Dictionary donne la définition suivante : "Machine capable d'exécuter automatiquement une série complexe d'actions, en particulier une machine programmable par ordinateur". Cette définition comporte quelques éléments intéressants :

— "Exécuter des actions automatiquement." C'est un élément clé en robotique, mais aussi dans de nombreuses autres machines plus simples appelées automates. La différence entre un robot et un automate simple comme un lave-vaisselle réside dans la définition de ce qu'est une "série complexe d'actions". Le lavage du linge est-il composé d'une série complexe d'actions ou non ? Piloter un avion en pilote automatique est-il une action complexe ? La cuisson du pain est-elle complexe ? Pour toutes ces tâches, il existe des machines qui se situent à la frontière entre les automates et les robots. L'élément "programmable par un ordinateur" est un autre élément clé d'un robot, car certains automates sont programmés mécaniquement et ne sont pas très flexibles. D'autre part, on trouve des ordinateurs partout, il est donc difficile d'utiliser ce critère pour distinguer un robot d'une autre machine.

Un élément crucial des robots qui n'est pas mentionné explicitement dans la définition est l'utilisation de capteurs. La plupart des automates ne possèdent pas de capteurs et ne peuvent pas adapter leurs actions à leur environnement. Ce sont les capteurs qui permettent à un robot d'effectuer des tâches complexes.

Dans les sections 1.1–1.5 de ce chapitre d'introduction, nous donnons un bref aperçu des différents types de robots. La section 1.6 décrit le robot générique que nous utilisons et la sec. 1.7 présente le pseudocode utilisé pour formaliser les algorithmes. La section 1.8 donne un aperçu détaillé du contenu du livre.

1.1 Classification des robots

Les robots peuvent être classés en fonction de l'environnement dans lequel ils évoluent (Fig. reffig.classification1).. La distinction la plus courante est celle entre les robots *fixe* et *mobile*. Ces deux types de robots ont des environnements de travail très différents et nécessitent donc des capacités très différentes. Les robots fixes sont principalement des manipulateurs industriels qui travaillent dans des environnements bien définis et adaptés aux robots. Les robots industriels effectuent des tâches répétitives spécifiques, comme le soudage ou la peinture de pièces dans les usines de

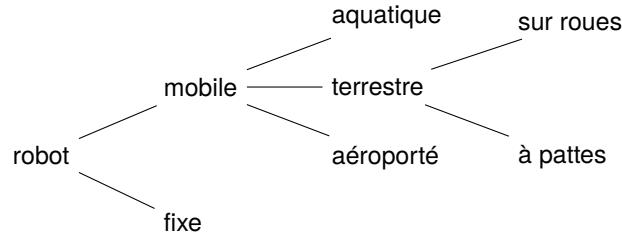


FIG. 1.1 – Classification des robots par environnement et mécanisme d’interaction

construction automobile. Avec l’amélioration des capteurs et des dispositifs d’interaction homme-robot, les manipulateurs robotiques sont de plus en plus utilisés dans des environnements moins contrôlés, comme la chirurgie de haute précision.

En revanche, les robots mobiles sont censés se déplacer et effectuer des tâches dans des environnements vastes, mal définis et incertains qui ne sont pas conçus spécifiquement pour les robots. Ils doivent faire face à des situations qui ne sont pas précisément connues à l’avance et qui évoluent dans le temps. Ces environnements peuvent inclure des entités imprévisibles comme les humains et les animaux. Les aspirateurs robotisés et les voitures à conduite autonome sont des exemples de robots mobiles.

Il n’existe pas de ligne de démarcation nette entre les tâches effectuées par les robots fixes et les robots mobiles - les humains peuvent interagir avec les robots industriels et les robots mobiles peuvent être contraints de se déplacer sur des rails - mais il est pratique de considérer les deux catégories comme fondamentalement différentes. En particulier, les robots fixes sont attachés à un support stable sur le sol, de sorte qu’ils peuvent calculer leur position en fonction de leur état interne, tandis que les robots mobiles doivent s’appuyer sur leur perception de l’environnement pour calculer leur emplacement.

Il existe trois environnements principaux pour les robots mobiles qui requièrent des principes de conception très différents car ils se distinguent par leur mécanisme de déplacement : aquatique (exploration sous-marine), terrestre (voitures) et aérien (drones). Là encore, la classification n’est pas stricte. Par exemple, il existe des robots amphibiens qui se déplacent à la fois dans l’eau et sur le sol. Les robots destinés à ces trois environnements peuvent encore être divisés en sous-classes : les robots terrestres peuvent avoir des jambes, des roues ou des chenilles, et les robots aériens peuvent être des ballons plus légers que l’air ou des avions plus lourds que l’air, qui sont à leur tour divisés en ailes fixes et en ailes tournantes (hélicoptères).

Les robots peuvent être classés selon le domaine d’application prévu et les tâches qu’ils effectuent (Fig. reffig.classification2). Nous avons mentionné les robots industriels qui travaillent dans des environnements bien définis sur des tâches de

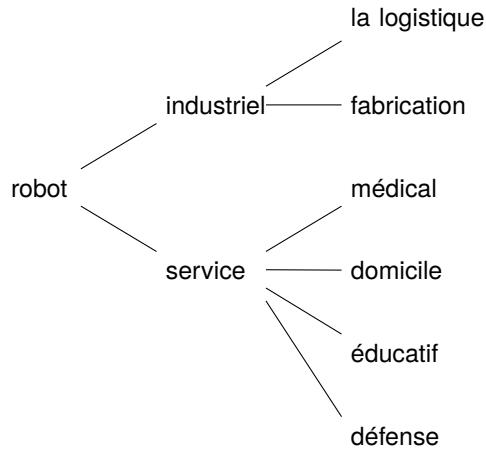


FIG. 1.2 – Classification des robots par domaine d'application

production. Les premiers robots étaient des robots industriels car l'environnement bien défini simplifiait leur conception. Les robots de service, quant à eux, assistent les humains dans leurs tâches. Il s'agit notamment des tâches ménagères comme les aspirateurs, des transports comme les voitures à conduite autonome et des applications de défense comme les drones de reconnaissance. En médecine aussi, les robots sont de plus en plus utilisés en chirurgie, en rééducation et en formation. Il s'agit d'applications récentes qui nécessitent des capteurs améliorés et une interaction plus étroite avec l'utilisateur.

1.2 Robots industriels

Les premiers robots étaient des robots industriels qui remplaçaient les travailleurs humains effectuant des tâches répétitives simples. Les chaînes de montage en usine peuvent fonctionner sans la présence de l'homme, dans un environnement bien défini où le robot doit effectuer des tâches dans un ordre précis, en agissant sur des objets placés précisément devant lui (Fig. 1.3).

On pourrait arguer qu'il s'agit en réalité d'automates et non de robots. Cependant, les automates d'aujourd'hui s'appuient souvent sur des capteurs, à tel point qu'ils peuvent être considérés comme des robots. Toutefois, leur conception est simplifiée car ils travaillent dans un environnement personnalisé auquel les humains ne sont pas autorisés à accéder pendant que le robot travaille.

Cependant, les robots d'aujourd'hui ont besoin de plus de flexibilité, par exemple, la capacité de manipuler des objets dans différentes orientations ou de reconnaître différents objets qui doivent être emballés dans le bon ordre. Le robot peut être amené



FIG. 1.3 – Des robots sur une chaîne de montage dans une usine de voitures. Source : https://commons.wikimedia.org/wiki/File:AKUKA_Industrial_Robots_IR.jpg by Mixabest (Own work). CC BY-SA 3.0 (<http://creativecommons.org/licenses/by-sa/3.0>) or GFDL (<http://www.gnu.org/copyleft/fdl.html>), via Wikimedia Commons.

à transporter des marchandises vers et depuis des entrepôts. Cela apporte une autonomie supplémentaire, mais la caractéristique de base demeure : l'environnement est plus ou moins contraint et peut être adapté au robot.

Une flexibilité supplémentaire est requise lorsque les robots industriels interagissent avec les humains, ce qui introduit des exigences de sécurité élevées, tant pour les bras robotiques que pour les robots mobiles. En particulier, la vitesse du robot doit être réduite et la conception mécanique doit garantir que les pièces mobiles ne constituent pas un danger pour l'utilisateur. L'avantage de faire travailler des humains avec des robots est que chacun peut réaliser ce qu'il fait le mieux : les robots effectuent des tâches répétitives ou dangereuses, tandis que les humains réalisent des étapes plus complexes et définissent les tâches globales du robot, car ils sont prompts à reconnaître les erreurs et les possibilités d'optimisation.

1.3 Robots mobiles autonomes

De nombreux robots mobiles sont télécommandés et effectuent des tâches telles que l'inspection de canalisations, la photographie aérienne et la neutralisation de

bombes, qui dépendent d'un opérateur contrôlant l'appareil. Ces robots ne sont pas autonomes ; ils utilisent leurs capteurs pour permettre à leur opérateur d'accéder à distance à des endroits dangereux, éloignés ou inaccessibles. Certains d'entre eux peuvent être semi-autonomes, réalisant des sous-tâches automatiquement. Le pilote automatique d'un drone stabilise le vol tandis que l'homme choisit la trajectoire de vol. Un robot dans une canalisation peut contrôler son mouvement à l'intérieur de la canalisation pendant que l'homme recherche les défauts qui doivent être réparés. Les robots mobiles entièrement autonomes ne dépendent pas d'un opérateur, mais prennent des décisions par eux-mêmes et exécutent des tâches telles que le transport de matériaux tout en naviguant sur un terrain incertain (murs et portes dans les bâtiments, intersections dans les rues) et dans un environnement en constante évolution (personnes marchant autour, voitures circulant dans les rues).

Les premiers robots mobiles ont été conçus pour des environnements simples, par exemple des robots qui nettoyaient les piscines ou des tondeuses à gazon robotisées. Aujourd'hui, les aspirateurs robotisés sont largement répandus, car il s'est avéré possible de construire des robots à prix raisonnable capables de naviguer dans un environnement intérieur encombré d'obstacles.

De nombreux robots mobiles autonomes sont conçus pour aider les professionnels travaillant dans des environnements structurés tels que les entrepôts. Un exemple intéressant est un robot pour désherber les champs (Fig. 1.4). Cet environnement est partiellement structuré, mais une détection avancée est nécessaire pour effectuer les tâches d'identification et d'enlèvement des mauvaises herbes. Même dans les usines très structurées, les robots partagent l'environnement avec les humains et leur détection doit donc être extrêmement fiable.

Le robot mobile autonome qui fait le plus parler de lui ces derniers temps est sans doute la voiture à conduite autonome. Ces dernières sont extrêmement difficiles à développer en raison de l'environnement incertain très complexe du trafic motorisé et des exigences strictes en matière de sécurité.

L'espace est un environnement encore plus difficile et dangereux. Les rovers martiens Sojourner et Curiosity sont des robots mobiles semi-autonomes. Le Sojourner a été actif pendant trois mois en 1997. Le Curiosity est actif depuis son atterrissage sur Mars en 2012 ! Bien qu'un conducteur humain sur Terre contrôle les missions (les routes à emprunter et les expériences scientifiques à mener), les rovers ont la capacité d'éviter les dangers de manière autonome.

Une grande partie de la recherche et du développement en robotique vise aujourd'hui à rendre les robots plus autonomes en améliorant les capteurs et en permettant un contrôle plus intelligent du robot. De meilleurs capteurs peuvent percevoir les détails de situations plus complexes, mais pour gérer ces situations, le contrôle du comportement du robot doit être très souple et adaptable. La vision, en particulier, est un domaine de recherche très actif car les caméras sont bon marché et les informations qu'elles peuvent acquérir sont très riches. Des efforts sont faits pour



FIG. 1.4 – *Robot mobile autonome désherbant un champ (avec l'autorisation d'Eco-robotix)*

rendre les systèmes plus flexibles, afin qu'ils puissent apprendre d'un humain ou s'adapter à de nouvelles situations. Un autre domaine de recherche actif concerne l'interaction entre les humains et les robots. Cette interaction implique à la fois la détection et l'intelligence, mais elle doit également tenir compte de la psychologie et de la sociologie des interactions.

1.4 Robots humanoïdes

La science-fiction et les médias de masse aiment représenter les robots sous une forme humanoïde. Nous connaissons tous R2-D2 et 3-CPO, les personnages robotiques des films de la Guerre des étoiles, mais le concept est très ancien. Au XVIII^e siècle, un groupe d'horlogers suisses - Pierre et Henri-Louis Jaquet-Droz et Jean-Frédéric Leschot - ont construit des automates humanoïdes pour démontrer leurs compétences mécaniques et faire la publicité de leurs montres. Aujourd'hui, de nombreuses entreprises construisent des robots humanoïdes pour des raisons similaires.

Les robots humanoïdes sont une forme de robot mobile autonome doté d'une conception mécanique extrêmement complexe pour le mouvement des bras et la locomotion par les jambes. Les robots humanoïdes sont utilisés pour la recherche sur la mécanique de la marche et sur l'interaction homme-machine. Des robots humanoïdes ont été proposés pour assurer les services et la maintenance dans une



FIG. 1.5 – Le robot Thymio.
Source : <https://www.thymio.org/en:mediakit> avec l'autorisation de École Polytechnique Fédérale de Lausanne and École Cantonale d'Art de Lausanne.



FIG. 1.6 – Robot Dash. Source : <https://www.makewonder.com/midiakit> avec l'autorisation de Wonder Workshop.

maison ou une station spatiale. Ils sont envisagés pour fournir des soins aux personnes âgées qui pourraient se sentir anxiées en présence d'une machine qui n'a pas l'air humaine. D'autre part, les robots qui ressemblent beaucoup aux humains peuvent générer de la répulsion, un phénomène appelé la *uncanny valley*.

Les robots humanoïdes peuvent être très difficiles à concevoir et à contrôler. Ils sont coûteux à construire et comportent de multiples articulations qui peuvent se déplacer de nombreuses façons différentes. Les robots qui utilisent des roues ou des chenilles sont préférés pour la plupart des applications car ils sont plus simples, moins chers et plus robustes.

1.5 Robots éducatifs

Les progrès de l'électronique et de la mécanique ont permis de construire des robots relativement peu coûteux. Les robots éducatifs sont largement utilisés dans les écoles, tant en classe que dans le cadre d'activités extrascolaires. Le grand nombre de robots éducatifs ne permet pas d'en donner un aperçu complet. Nous donnons ici quelques exemples représentatifs des robots couramment utilisés dans l'enseignement.

Robots mobiles préassemblés

De nombreux robots éducatifs sont conçus comme des robots mobiles préassemblés. La Fig. 1.5 montre le robot Thymio de Mobsya et la Fig. 1.6 montre le robot Dash de Wonder Workshop. Ces robots sont relativement peu coûteux, robustes et contiennent un grand nombre de capteurs et de composants de sortie tels que des lumières. Un avantage important de ces robots est que vous pouvez mettre en œuvre des algorithmes robotiques "prêts à l'emploi", sans investir des heures dans la conception et la construction mécanique. Cependant, les robots pré-assemblés ne peuvent pas être modifiés, bien que beaucoup d'entre eux permettent de construire des extensions en utilisant, par exemple, des composants LEGO®.

Kits de robotique



FIG. 1.7 – LEGO® Mindstorms EV3
(avec l'aimable autorisation d'Adi Shmorak, Intelitek)



FIG. 1.8 – Bras robotisés Poppy Ergo Jr
(avec l'aimable autorisation du Poppy Project)

Les kits robotiques LEGO® Mindstorms (Fig. 1.7) ont été lancés en 1998.¹ Un kit se compose de briques LEGO® standard et d'autres éléments de construction, ainsi que de moteurs et de capteurs, et d'une brique programmable qui contient l'ordinateur qui contrôle les composants du robot. L'avantage des kits robotiques est qu'ils sont flexibles : vous pouvez concevoir et construire un robot pour effectuer une tâche spécifique, en n'étant limité que par votre imagination. Un kit de robotique peut également être utilisé pour enseigner la conception mécanique aux étudiants. Les inconvénients des kits robotiques sont qu'ils sont plus coûteux que les simples robots préassemblés et que l'exploration des algorithmes robotiques dépend de la capacité de chacun à mettre en œuvre avec succès une conception mécanique robuste.

Une tendance récente consiste à remplacer les collections fixes de briques par des pièces construites par des imprimantes 3D. Le bras robotique Poppy Ergo Jr (Fig. 1.8) en est un exemple. L'utilisation de pièces imprimées en 3D permet une plus grande flexibilité dans la création de la structure mécanique et une plus grande robustesse, mais nécessite l'accès à une imprimante 3D.

Bras robotiques

Pour agir sur son environnement, le robot a besoin d'un *actuateur* qui est un composant d'un robot qui affecte l'environnement. De nombreux robots, en particulier les bras robotisés utilisés dans l'industrie, agissent sur l'environnement par le biais d'effecteurs finaux, généralement des pinces ou des outils similaires (Figs. 1.3, 14.1, 15.7). Les actionneurs des robots mobiles sont les moteurs qui font bouger le robot, ainsi que des composants tels que la pompe à vide d'un aspirateur.

Les robots éducatifs sont généralement des robots mobiles dont les seuls actionneurs sont ses moteurs et des dispositifs d'affichage tels que des lumières, des sons ou un écran. Les effecteurs peuvent être construits avec des kits de robotique ou en utilisant des composants supplémentaires avec des robots pré-assemblés, bien que des bras robotiques éducatifs existent (Fig. reffig.poppy). La manipulation d'objets introduit une certaine complexité dans la conception ; cependant, les algorithmes des

1. La figure montre la dernière version appelée EV3 introduite en 2014.

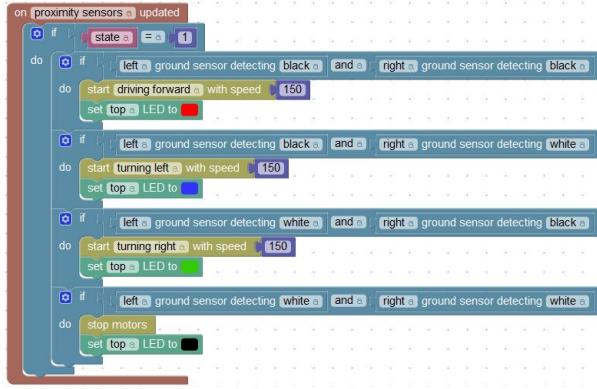


FIG. 1.9 – Logiciel Blockly pour le robot Thymio

effecteurs finaux étant similaires à ceux des robots mobiles simples, la plupart des activités de ce livre supposeront uniquement que votre robot dispose de moteurs et de dispositifs d'affichage.

Environnements de développement de logiciels

Chaque système de robotique éducative comprend un *environnement de développement logiciel*. Le langage de programmation peut être une version d'un langage de programmation standard comme Java ou Python. La programmation est simplifiée si un langage à base de blocs est utilisé, généralement un langage basé sur Scratch ou Blockly (Fig. 1.9).

Pour simplifier davantage la programmation d'un robot par de jeunes étudiants, une notation de programmation entièrement graphique peut être utilisée. Les figures 1.10 montrent le VPL (Visual Programming Language), un environnement logiciel graphique pour le robot Thymio. Il utilise des paires événement-action : lorsque l'événement représenté par le bloc de gauche se produit, les actions des blocs suivants sont exécutées.

La figure 1.11 montre l'environnement logiciel graphique pour le robot Dash. Il utilise également des événements et des actions, où les actions sont représentées par des nœuds et les événements par des flèches entre les nœuds.

1.6 Le robot générique

Cette section présente la description d'un robot générique que nous utilisons pour présenter les algorithmes de robotique. Les capacités du robot générique sont similaires à celles des robots éducatifs, mais celui que vous utiliserez n'aura peut-être pas toutes les capacités supposées dans les présentations ; vous devrez donc

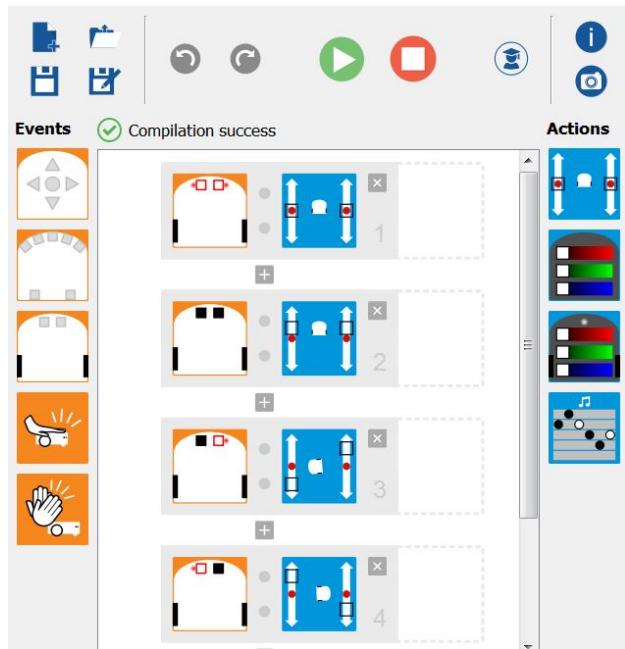


FIG. 1.10 – Logiciel VPL pour le robot Thymio



FIG. 1.11 – Les logiciels Wonder pour le robot Dash (Avec l'aimable autorisation de Wonder Workshop)

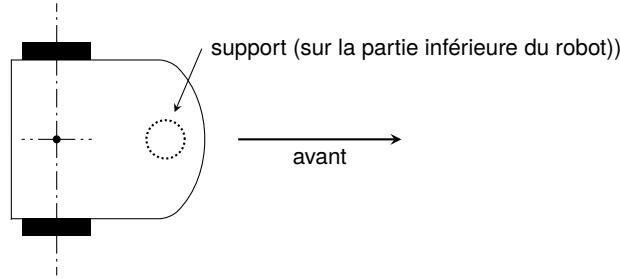


FIG. 1.12 – Robot à entraînement différentiel

improviser. Vous ne comprendrez peut-être pas encore tous les termes de la description suivante, mais il est important que la spécification soit formalisée. Des détails supplémentaires seront donnés dans les chapitres suivants.

1.6.1 Direction différentielle

Le robot est un petit véhicule autonome doté d'une *motorisation différentielle*, ce qui signifie qu'il possède deux roues entraînées par des moteurs indépendants (Fig. 1.12). Pour faire bouger le robot, réglez la puissance du moteur sur une valeur comprise entre -100 (pleine puissance en arrière), 0 (arrêt) et 100 (pleine puissance en avant). Il n'y a pas de relation préédéfinie entre la puissance du moteur et la vitesse du robot. Le moteur peut être relié aux roues par différents rapports d'engrenage, le type de pneus sur les roues affecte leur traction, et un terrain sablonneux ou boueux peut faire glisser les roues.

La figure 1.12 montre une vue du robot depuis le haut. L'avant du robot est la courbe vers la droite, qui correspond également à la direction avant du mouvement du robot. Les roues (rectangles noirs) se trouvent sur les côtés gauche et droit de l'arrière du corps du robot. Le point est le point de l'essieu situé à mi-chemin entre les roues. Lorsque le robot tourne, il tourne autour d'un axe vertical à ce point. Pour la stabilité, vers l'avant du robot se trouve un support ou une roue non motrice.

Dessin mécanique

Une ligne brisée est la notation standard en ingénierie mécanique pour l'axe de symétrie d'un composant tel qu'une roue. Lorsque la vue latérale d'une roue est affichée, l'intersection des deux axes de symétrie indique l'axe de rotation qui est perpendiculaire au plan de la page. Pour ne pas encombrer les diagrammes, nous simplifions la notation en ne montrant que des lignes brisées pour un *axe de rotation* d'un composant tel qu'une roue. En outre, l'intersection désignant un axe perpendiculaire est généralement abrégée en croix, éventuellement contenue dans

la roue ou son essieu.

L'entraînement différentiel présente plusieurs avantages : il est simple puisqu'il ne comporte que deux moteurs sans composants supplémentaires pour la direction et il permet au robot de tourner sur place. Dans une voiture, deux roues sont entraînées ensemble (ou quatre roues sont entraînées par paires) et il existe un mécanisme complexe distinct pour la direction appelé *Direction d'Ackermann*. Étant donné qu'une voiture ne peut pas tourner sur place, les conducteurs doivent effectuer des manœuvres compliquées telles que le stationnement parallèle ; les conducteurs humains apprennent facilement à le faire, mais de telles manœuvres sont difficiles pour un système autonome. Un robot autonome doit effectuer des manœuvres complexes avec des mouvements très simples, c'est pourquoi l'entraînement différentiel est la configuration préférée : il peut facilement tourner vers n'importe quel cap et se déplacer ensuite dans cette direction.

Le principal inconvénient d'un système d'entraînement différentiel est qu'il nécessite un troisième point de contact avec le sol, contrairement à une voiture qui a déjà quatre roues pour la soutenir et qui peut donc se déplacer facilement sur un terrain difficile. Un autre inconvénient est qu'il ne peut pas se déplacer latéralement sans tourner. Il existe des configurations qui permettent à un robot de se déplacer latéralement (Sect. 5.12), mais elles sont complexes et coûteuses. L'entraînement différentiel est également utilisé dans les véhicules à chenilles tels que les engins de terrassement et les chars militaires. Ces véhicules peuvent manœuvrer sur des terrains extrêmement accidentés, mais les chenilles produisent beaucoup de friction et les mouvements sont lents et peu précis.

Réglage de la puissance ou de la vitesse

La puissance fournie par un moteur est régulée par un *throttle*, comme une pédale dans une voiture ou des leviers dans un avion ou un bateau. Les moteurs électriques utilisés dans les robots mobiles sont contrôlés en modifiant la tension appliquée aux moteurs à l'aide d'une technique appelée *modulation de largeur d'impulsion*. Dans de nombreux robots éducatifs, des algorithmes de contrôle, tels que ceux décrits dans le Chap. 6, sont utilisés pour garantir que les moteurs tournent à une *vitesse cible* spécifiée. Étant donné que nous nous intéressons aux concepts et aux algorithmes de conception de robots, nous exprimerons les algorithmes en termes de fourniture de puissance et traiterons séparément le contrôle de la vitesse.

1.6.2 Capteurs de proximité

Le robot dispose de *capteurs de proximité horizontaux* qui peuvent détecter un objet à proximité du robot. Il existe de nombreuses technologies pouvant être utilisées pour construire ces capteurs, comme l'infrarouge, le laser, les ultrasons ; le

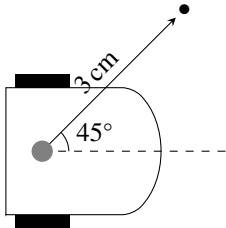


FIG. 1.13 – Robot avec capteur rotatif (point gris)

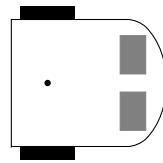


FIG. 1.14 – Robot avec deux capteurs de sol sur la partie inférieure du robot (rectangles gris)

robot générique représente les robots qui utilisent l'une de ces technologies. Nous spécifions que les capteurs doivent avoir les capacités suivantes : Un capteur de proximité horizontal peut mesurer la distance (en centimètres) entre le robot et un objet et l'angle (en degrés) entre l'avant du robot et l'objet. La figure 1.13 montre un objet situé à 3 cm du centre du robot à un angle de 45° par rapport à la direction vers laquelle pointe le robot.²

En pratique, un robot éducatif dispose d'un petit nombre de capteurs et ne peut donc pas détecter les objets dans toutes les directions. De plus, les capteurs bon marché ne seront pas en mesure de détecter des objets très éloignés et leurs mesures ne seront pas précises. Les mesures seront également affectées par des facteurs environnementaux tels que le type d'objet, la lumière ambiante, etc. Pour simplifier nos algorithmes, nous ne supposons pas de limites prédéfinies, mais lorsque vous implémenterez les algorithmes, vous devrez tenir compte de ces limites.

1.6.3 Capteurs de sol

Capteurs de sol sont montés sur la partie inférieure du robot. Étant donné que ces capteurs sont très proches du sol, la distance ou l'angle n'ont aucune signification ; au lieu de cela, le capteur mesure la luminosité de la lumière réfléchie par le sol dans des valeurs arbitraires comprises entre 0 (totalement sombre) et 100 (totalement clair). Le robot générique possède deux capteurs de sol montés à l'avant du robot (Fig. ref{fig:generic-ground}), bien que nous présentions parfois des algorithmes qui n'utilisent qu'un seul capteur. La figure montre une vue de dessus du robot, bien que les capteurs de sol se trouvent sur la partie inférieure du robot.

1.6.4 Ordinateur embarqué

Le robot est équipé d'un *ordinateur intégré* (Fig. 1.15). La spécification précise de l'ordinateur n'est pas importante mais nous supposons certaines capacités. L'ordinateur peut lire les valeurs des capteurs et régler la puissance des moteurs. Il

2. Voir Annexe A sur les conventions de mesure des angles.

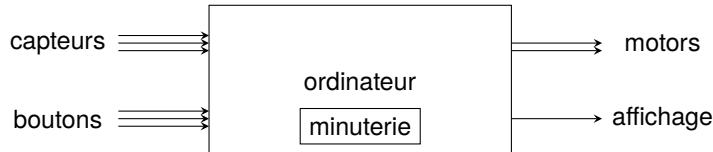


FIG. 1.15 – Ordinateur embarqué

est possible d'afficher des informations sur un petit écran ou d'utiliser des lumières colorées. Les signaux et les données peuvent être introduits dans l'ordinateur à l'aide de boutons, d'un clavier ou d'une télécommande.

Les données sont introduites dans l'ordinateur par des *événements*, par exemple en touchant un bouton. L'occurrence d'un événement entraîne l'exécution d'une procédure appelée *event handler*. L'événement peut être détecté par le matériel, auquel cas le terme *interruption* est utilisé, ou il peut être détecté par le logiciel, généralement, par *polling*, où le système d'exploitation vérifie les événements à des intervalles prédéfinis. Lorsque le gestionnaire d'événements se termine, le calcul précédent est relancé.

Les gestionnaires d'événements sont différents des programmes séquentiels qui ont une instruction initiale qui entre des données et une instruction finale qui affiche la sortie, car les gestionnaires d'événements sont exécutés en réponse à des événements imprévisibles. Le traitement des événements est utilisé pour mettre en œuvre des interfaces utilisateur graphiques sur les ordinateurs et les smartphones : lorsque vous cliquez sur une icône ou que vous la touchez, un gestionnaire d'événements est exécuté. Sur un robot, un événement peut être une entrée discrète, comme le fait de toucher une touche. Les événements peuvent également se produire lorsqu'une valeur continue, telle que la valeur lue par un capteur, est supérieure ou inférieure à une valeur prédéfinie appelée *threshold*.

L'ordinateur comprend un *timer* qui fonctionne comme un chronomètre sur un smartphone. Un chronomètre est une variable qui est *set* sur une période de temps, par exemple, 0,5 seconde, qui est représentée par un nombre entier de millisecondes ou de microsecondes (0,5 seconde est 500 millisecondes). L'horloge matérielle de l'ordinateur provoque une interruption à intervalles fixes et le système d'exploitation décrémente la valeur du minuteur. Lorsque sa valeur atteint zéro, on dit que la minuterie a *expiré*; une interruption se produit.

Les minuteurs sont utilisés pour mettre en œuvre des événements répétés, comme l'allumage et l'extinction d'une lumière. Ils sont également utilisés pour le *polling*, une alternative aux gestionnaires d'événements : au lieu d'effectuer un calcul lorsqu'un événement se produit, les capteurs sont lus et stockés périodiquement. Plus précisément, le polling se produit comme un gestionnaire d'événements lorsqu'un timer expire, mais la conception d'un logiciel utilisant le polling peut être très

Algorithm 1.1: Multiplication des nombres entiers	
integer x ← 0	
integer a, b	
1: a ← input an integer	
2: b ← input a non-negative integer	
3: while b ≠ 0	
4: x ← x + a	// Add the value a to x
5: b ← b – 1	// for each b

différente de celle d'un logiciel basé sur des événements.

1.7 Le formalisme algorithmique

Les algorithmes mis en œuvre sous forme de programmes informatiques sont utilisés par l'ordinateur embarqué pour contrôler le comportement du robot. Nous ne donnons pas de programmes dans un langage de programmation spécifique ; au lieu de cela, les algorithmes sont présentés dans un *pseudocode*, un format structuré utilisant une combinaison de langage naturel, de mathématiques et de structures de programmation. L'algorithme 1.1 est un algorithme simple de multiplication d'entiers par addition répétée. L'entrée de l'algorithme est une paire d'entiers et la sortie est le produit des deux valeurs d'entrée. L'algorithme déclare trois variables entières : x, a, b. Il y a cinq instructions dans la partie exécutable. L'indentation est utilisée (comme dans le langage de programmation Python) pour indiquer la portée de la boucle. Une flèche est utilisée pour l'affectation, afin que les symboles familiers = et ≠ puissent être utilisés pour l'égalité et l'inégalité dans les formules mathématiques.³

La puissance du moteur est définie à l'aide d'instructions d'affectation :

```
left-motor-power ← 50
right-motor-power ← -50
```

Nous avons défini nos capteurs de proximité comme renvoyant la distance à un objet détecté et son angle par rapport à la direction avant du robot, mais il sera souvent plus pratique d'utiliser des expressions en langage naturel telles que :

```
when object detected in front
when object not detected in back
```

3. Beaucoup de langages de programmation utilisent = pour l'affectation, puis == pour l'égalité et != pour l'inégalité. Cela prête à confusion car l'égalité x = y est symétrique, mais l'affectation ne l'est pas x=x+1. Nous préférons conserver la notation mathématique.

1.8 Un aperçu du contenu du livre

Les six premiers chapitres constituent le cœur des concepts et des algorithmes de la robotique.

Chapitre 1 Les robots et leurs applications Ce chapitre présente et classe les robots. Il précise également le robot générique et les formalismes utilisés pour présenter les algorithmes dans ce livre.

Chapitre 2 Capteurs Les robots sont plus que des appareils contrôlés à distance comme un téléviseur. Ils présentent un comportement autonome basé sur la détection d'objets dans leur environnement à l'aide de capteurs. Ce chapitre donne un aperçu des capteurs utilisés par les robots et explique les concepts de portée, de résolution, de précision et d'exactitude. Il traite également de la non-linéarité des capteurs et de la manière de la gérer.

Chapitre 3 Comportement réactif Lorsqu'un robot autonome détecte un objet dans son environnement, il réagit en modifiant son comportement. Ce chapitre présente les algorithmes robotiques dans lesquels le robot modifie directement son comportement en fonction des données fournies par ses capteurs. Les véhicules de Braitenberg sont des exemples simples mais élégants de comportement réactif. Le chapitre présente plusieurs variantes d'algorithmes de suivi de ligne.

Chapitre 4 Machines à états finis Un robot peut se trouver dans différents états, où sa réaction aux entrées provenant de ses capteurs dépend non seulement de ces valeurs mais aussi de l'état actuel. Les machines à états finis sont un formalisme permettant de décrire les états et les transitions entre eux qui dépendent de l'occurrence d'événements.

Chapitre 5 Mouvement robotique et odométrie Les robots autonomes explorent leur environnement en effectuant des actions. Il ne se passe pas un jour sans qu'un rapport sur l'expérience des voitures autonomes ne soit publié. Ce chapitre passe en revue les concepts liés au mouvement (distance, temps, vitesse, accélération), puis présente l'odométrie, la méthode fondamentale qu'utilise un robot pour se déplacer d'une position à une autre. L'odométrie est sujette à des erreurs importantes et il est important de comprendre leur nature.

La deuxième partie du chapitre donne un aperçu des concepts avancés du mouvement robotique : les encodeurs de roue et les systèmes de navigation inertielle qui peuvent améliorer la précision de l'odométrie, et les degrés de liberté et l'holonomie qui affectent la planification du mouvement robotique.

Chapitre 6 Contrôle Un robot autonome est un système de contrôle en boucle fermée, car les entrées de ses capteurs affectent son comportement qui, à son tour, affecte ce qui est mesuré par les capteurs. Par exemple, une voiture autonome qui s'approche d'un feu de circulation peut freiner plus fort à mesure qu'elle se rapproche du feu. Ce chapitre décrit les mathématiques des systèmes de contrôle qui assurent un comportement optimal : la voiture s'arrête effectivement au feu et le freinage est progressif et doux.

Un robot mobile autonome doit d'une manière ou d'une autre naviguer d'une position de départ à une position d'arrivée, par exemple pour apporter des médicaments de la pharmacie d'un hôpital au patient. La navigation est un problème fondamental en robotique qui est difficile à résoudre. Les quatre chapitres suivants présentent des algorithmes de navigation dans divers contextes.

Chapitre 7 Navigation locale : évitement d'obstacles L'exigence la plus fondamentale pour un robot mobile est qu'il ne s'écrase pas sur les murs, les personnes et autres obstacles. C'est ce qu'on appelle la navigation *locale*, car elle concerne le voisinage immédiat du robot et non les objectifs que le robot tente d'atteindre. Le chapitre commence par des algorithmes de suivi des murs qui permettent à un robot de se déplacer autour d'un obstacle ; ces algorithmes sont similaires aux algorithmes de navigation dans un labyrinthe. Le chapitre décrit un algorithme probabiliste qui simule la navigation d'une colonie de fourmis à la recherche d'une source de nourriture.

Chapter 8 Localisation Avant que tous les smartphones ne soient équipés d'un système de navigation GPS, nous nous déplaçons à l'aide de cartes imprimées sur papier. Un problème difficile est la localisation : pouvez-vous déterminer votre position actuelle sur la carte ? Les robots mobiles doivent résoudre le même problème de localisation, souvent sans le bénéfice de la vision. Ce chapitre décrit la localisation par des calculs trigonométriques à partir de positions connues. Il est suivi de sections sur la localisation probabiliste : Un robot peut détecter un point de repère mais il peut y avoir de nombreux points de repère similaires sur la carte. En attribuant des probabilités et en les mettant à jour au fur et à mesure que le robot se déplace dans l'environnement, il peut finalement déterminer sa position avec une certitude relative.

Chapitre 9 Cartographie Mais d'où vient la carte ? Des plans précis des rues sont facilement disponibles, mais un aspirateur robotisé ne dispose pas d'un plan de votre appartement. Un robot sous-marin est utilisé pour explorer un environnement inconnu. Pour effectuer une localisation, le robot a besoin d'une carte, mais pour créer une carte d'un environnement inconnu, le robot doit se localiser lui-même, en ce sens qu'il doit savoir à quelle distance il s'est déplacé d'un point à un autre de l'environnement. La solution consiste à

effectuer simultanément la localisation et la cartographie. La première partie du chapitre décrit un algorithme d'exploration d'un environnement pour déterminer l'emplacement des obstacles. Ensuite, un algorithme simplifié de localisation et de cartographie simultanées est présenté.

Chapitre 10 Navigation basée sur la cartographie Maintenant que le robot dispose d'une carte, supposons qu'on lui assigne une tâche qui nécessite qu'il se déplace d'une position de départ vers une position d'arrivée. Quelle route doit-il emprunter ? Ce chapitre présente deux algorithmes de planification de chemin : L'algorithme de Dijkstra, un algorithme classique permettant de trouver le plus court chemin dans un graphe, et l'algorithme A*, une version plus efficace de l'algorithme de Dijkstra qui utilise des informations heuristiques.

Les chapitres suivants présentent des sujets avancés en robotique. Ils sont indépendants les uns des autres, ce qui vous permet de choisir ceux que vous souhaitez étudier et dans quel ordre.

Chapitre 11 Contrôle par logique floue Les algorithmes de contrôle (Chap. 6) nécessitent la spécification d'une valeur cible précise : un système de chauffage a besoin de la température cible d'une pièce et un système de régulation de vitesse a besoin de la vitesse cible d'une voiture. Une autre approche, appelée logique floue, utilise des spécifications imprécises telles que froid, frais, tiède, chaud, ou très lent, lent, rapide, très rapide. Ce chapitre présente la logique floue et montre comment elle peut être utilisée pour contrôler un robot qui s'approche d'un objet.

Chapitre 12 Traitement des images La plupart des capteurs robotiques mesurent les distances et les angles à l'aide de lasers, de sons ou de lumière infrarouge. Nous, les humains, nous appuyons principalement sur notre vision. Les appareils photo numériques de haute qualité sont peu coûteux et se trouvent sur tous les smartphones. La difficulté consiste à traiter et à interpréter les images prises par l'appareil photo, ce que notre cerveau fait instantanément. Le traitement numérique des images a fait l'objet de recherches approfondies et ses algorithmes sont utilisés dans les robots avancés qui peuvent se permettre la puissance de calcul nécessaire. Dans ce chapitre, nous passons en revue les algorithmes de traitement d'images et montrons comment un robot éducatif peut faire la démonstration de ces algorithmes, même sans caméra.

Chapitre 13 Réseaux neuronaux Les robots autonomes évoluant dans des environnements très complexes ne peuvent pas disposer d'algorithmes pour toutes les situations possibles. Une voiture autonome ne peut pas connaître à l'avance tous les différents véhicules et configurations de véhicules qu'elle

rencontre sur la route. Les robots autonomes doivent apprendre de leur expérience. Il s'agit d'un sujet fondamental en intelligence artificielle, étudié depuis de nombreuses années. Ce chapitre présente une approche de l'apprentissage : les réseaux neuronaux artificiels, qui s'inspirent des neurones de notre cerveau. Un réseau neuronal utilise des algorithmes d'apprentissage pour modifier ses paramètres internes afin de s'adapter continuellement aux nouvelles situations qu'il rencontre.

Chapitre 14 Apprentissage automatique Une autre approche de l'apprentissage est une technique statistique appelée apprentissage automatique. Ce chapitre décrit deux algorithmes permettant de faire la distinction entre deux alternatives, par exemple entre un feu rouge et un feu vert. Le premier algorithme, appelé analyse discriminante linéaire, est basé sur les moyennes et les variances d'un ensemble d'échantillons. Le second algorithme utilise des perceptrons, une forme de réseau neuronal capable de distinguer des alternatives même lorsque les échantillons ne satisfont pas aux hypothèses statistiques nécessaires à l'analyse discriminante linéaire.

Chapitre 15 Swarm Robotics Si vous devez améliorer les performances d'un système, il est souvent plus facile d'utiliser plusieurs instances d'un composant plutôt que d'essayer d'améliorer les performances d'un composant individuel. Prenons l'exemple d'un problème tel que la surveillance d'une zone pour mesurer les niveaux de pollution. Vous pouvez utiliser un seul robot très rapide (et coûteux), mais il peut être plus facile d'utiliser plusieurs robots, chacun d'entre eux mesurant la pollution dans une petite zone. C'est ce qu'on appelle la robotique en essaim, par analogie avec un essaim d'insectes capables de trouver le meilleur chemin entre leur nid et une source de nourriture. Le problème fondamental de la robotique en essaim, comme dans tous les systèmes concurrents, est de développer des méthodes de coordination et de communication entre les robots. Ce chapitre présente deux de ces techniques : l'échange d'informations et les interactions physiques.

Chapitre 16 Cinématique d'un manipulateur robotique Les robots éducatifs sont de petits robots mobiles qui se déplacent sur une surface bidimensionnelle. Il existe des robots mobiles qui se déplacent en trois dimensions : les avions et les sous-marins robotisés. Les mathématiques et les algorithmes pour le mouvement tridimensionnel ont été développés dans un autre domaine central de la robotique : les manipulateurs qui sont largement utilisés dans la fabrication. Ce chapitre présente un traitement simplifié des concepts fondamentaux des manipulateurs robotiques (cinématique directe et inverse, rotations, transformées homogènes) en deux dimensions, ainsi qu'un aperçu des rotations tridimensionnelles.

Il y a deux annexes :

Appendice A Unités de mesure Cette annexe contient le Tableau A.1 avec les unités de mesure. Le tableau A.2 donne les préfixes utilisés avec ces unités.

Appendix BDérivations et didacticiels mathématiques Ce chapitre contient des didacticiels qui passent en revue certains des concepts mathématiques utilisés dans le livre. En outre, certaines dérivations mathématiques détaillées ont été rassemblées ici afin de ne pas interrompre le flux du texte.

1.9 Summary

Les robots sont présents partout : dans les usines, les maisons et les hôpitaux, et même dans l'espace. Beaucoup de recherche et de développement sont investis dans la mise au point de robots qui interagissent directement avec les humains. Les robots sont utilisés dans les écoles afin d'accroître la motivation des élèves à étudier les STIM et comme outil pédagogique pour enseigner les STIM dans un environnement concret. Ce livre se concentre sur l'utilisation de robots éducatifs pour apprendre des algorithmes robotiques et explorer leur comportement.

La plupart des robots éducatifs ont une conception similaire : un petit robot mobile utilisant un entraînement différentiel et des capteurs de proximité. Pour que ce livre soit indépendant de la plate-forme, nous avons défini un robot générique avec ces propriétés. Les algorithmes présentés dans ce livre pour le robot générique devraient être faciles à mettre en œuvre sur les robots éducatifs, même si les différents robots auront des capacités différentes en termes de performances de leurs moteurs et de leurs capteurs. Les algorithmes sont présentés dans un pseudocode indépendant du langage, qui devrait être facile à traduire dans tout langage textuel ou graphique pris en charge par votre robot.

1.10 Lectures complémentaires

Pour un aperçu non technique de la robotique, en particulier de la robotique humanoïde et d'inspiration biologique, voir Winfield [50].

L'Organisation internationale de normalisation (ISO)⁴ publie des normes pour la robotique. Sur leur site web <https://www.iso.org/> vous pouvez trouver le catalogue de la robotique (ISO/TC 299) et les définitions formelles des concepts robotiques : ISO 8373 :2012 Robots et dispositifs robotiques–Vocabulaire et ISO 19649 :2017 Robots mobiles–Vocabulaire.

Les sujets abordés dans cet ouvrage sont présentés de manière plus détaillée dans des manuels avancés de robotique tels que [14, 43]. Leurs chapitres d'introduction donnent de nombreux exemples de robots.

4. Non, ce n'est pas une erreur ! ISO est le nom officiel abrégé de l'organisation et non un acronyme dans l'une de ses trois langues officielles : anglais, français et russe.

Les robots éducatifs sont accompagnés d'une documentation sur leurs capacités et sur les environnements de développement de logiciels. Il existe également des manuels basés sur des robots spécifiques, par exemple, [47] sur la programmation du LEGO® Mindstorms et [30] sur l'utilisation de Python pour programmer les robots Scribbler. La conception du langage de programmation visuel (VPL) est présentée dans [42].

Le pseudocode est fréquemment utilisé dans les manuels sur les structures de données et les algorithmes, à commencer par le manuel classique [3]. Le style utilisé ici est tiré de [5].

Chapitre 2

Capteurs

Un robot ne peut pas se déplacer sur une distance spécifique dans une direction spécifique simplement en réglant la puissance relative des moteurs des deux roues et la durée de fonctionnement des moteurs. Supposons que nous voulions que le robot se déplace en ligne droite. Si nous réglons la puissance des deux moteurs au même niveau, même de petites différences dans les caractéristiques des moteurs et des roues feront que le robot tournera légèrement sur le côté. Les inégalités de la surface sur laquelle le robot se déplace font également tourner les roues à des vitesses différentes. L'augmentation ou la diminution de la friction entre les roues et la surface peut affecter la distance parcourue en un laps de temps donné. Par conséquent, si nous voulons que le robot se déplace vers un mur situé à 1 m et s'arrête 20 cm devant, le robot doit *sentir* l'existence du mur et s'arrêter lorsqu'il *déetecte* que le mur est à 20 cm.

Un *capteur* est un composant qui mesure certains aspects de l'environnement. L'ordinateur du robot utilise ces mesures pour contrôler les actions du robot. L'un des capteurs les plus importants en robotique est le capteur de distance qui mesure la distance entre le robot et un objet. En utilisant plusieurs capteurs de distance ou en faisant tourner le capteur, il est possible de mesurer l'angle de l'objet par rapport à l'avant du robot. Les capteurs de distance peu coûteux utilisant la lumière infrarouge ou les ultrasons sont invariablement utilisés dans les robots éducatifs ; les robots industriels utilisent fréquemment des capteurs laser coûteux car ils sont très précis.

Le son et la lumière sont également utilisés pour les *communications* entre deux robots, comme décrit dans le Chap. 15.

Une connaissance plus approfondie de l'environnement peut être obtenue en analysant les images prises par une caméra. Bien que les caméras soient très petites et peu coûteuses (tous les smartphones en possèdent une), la quantité de données contenues dans une image est très importante et les algorithmes de traitement d'image nécessitent d'importantes ressources informatiques. C'est pourquoi les caméras sont principalement utilisées dans des applications complexes comme les voitures à conduite autonome.

La section 2.1 présente la terminologie des capteurs. La section 2.2 présente les capteurs de distance, les capteurs les plus souvent utilisés par les robots éducatifs. Elle est suivie de la Sec. 2.3 sur les caméras, puis d'une courte section sur les autres capteurs utilisés par les robots (Sec. 2.4). La section 2.5 définit les caractéristiques des capteurs : portée, résolution, précision, exactitude. Le chapitre se termine par une discussion sur la non-linéarité des capteurs (Sec. 2.6).

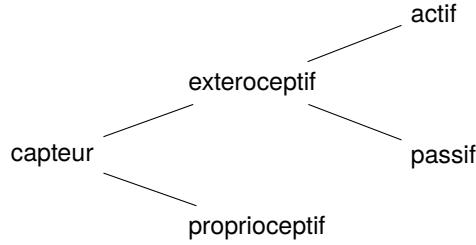


FIG. 2.1 – Classification des capteurs

2.1 Classification des capteurs

Les capteurs sont classés comme *proprioceptif* ou *exteroceptif*, et les capteurs extéroceptifs sont en outre classés comme *actif* ou *passif* (Fig. 2.1). Un capteur proprioceptif mesure un élément interne au robot lui-même. L'exemple le plus connu est le compteur de vitesse d'une voiture qui mesure la vitesse de la voiture en comptant les rotations des roues (Sec. 5.8). Un capteur extéroceptif mesure quelque chose d'extérieur au robot, comme la distance à un objet. Un capteur actif affecte l'environnement généralement en émettant de l'énergie : un télémètre sonar sur un sous-marin émet des ondes sonores et utilise le son réfléchi pour déterminer la distance. Un capteur passif n'affecte pas l'environnement : un appareil photo enregistre simplement la lumière réfléchie par un objet. Les robots utilisent invariablement certains capteurs extéroceptifs pour corriger les erreurs qui pourraient survenir avec les capteurs proprioceptifs ou pour prendre en compte les changements de l'environnement.

2.2 Capteurs de distance

Dans la plupart des applications, le robot doit mesurer la distance qui le sépare d'un objet à l'aide d'un *capteur de distance*. Les capteurs de distance sont généralement *actifs* : ils émettent un signal et reçoivent ensuite sa réflexion (le cas échéant) par un objet (Fig. 2.2). Une façon de déterminer la distance consiste à mesurer le temps qui s'écoule entre l'émission d'un signal et sa réception :

$$s = \frac{1}{2}vt, \quad (2.1)$$

où s est la distance, v est la vitesse du signal et t est le temps écoulé entre l'envoi et la réception du signal. Le facteur de moitié tient compte du fait que le signal parcourt la distance deux fois : jusqu'à l'objet et ensuite réfléchi. Une autre façon de reconstruire la distance est d'utiliser la triangulation, comme expliqué dans la section `refs.triangulating-sensors`.

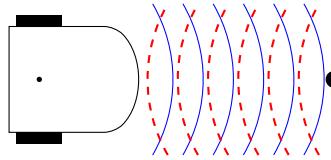


FIG. 2.2 – Mesurer la distance en émettant une onde et en recevant sa réflexion

Les capteurs de distance bon marché reposent sur un autre principe : comme l'intensité d'un signal diminue avec la distance, la mesure de l'intensité d'un signal réfléchi donne une indication de la distance entre le capteur et un objet. L'inconvénient de cette méthode est que l'intensité du signal reçu est influencée par des facteurs tels que la réflectivité de l'objet.

2.2.1 Capteurs de distance à ultrasons

Les ultrasons sont des sons dont la fréquence est supérieure à 20000 hertz, c'est-à-dire supérieure à la fréquence la plus élevée pouvant être entendue par l'oreille humaine. Il existe deux environnements où le son est meilleur que la vision : la nuit et dans l'eau. Les chauves-souris utilisent les ultrasons pour s'orienter lorsqu'elles volent la nuit, car après le coucher du soleil, il y a peu de lumière naturelle pour localiser la nourriture. Les navires et les sous-marins utilisent les ultrasons pour détecter des objets, car le son se propage beaucoup mieux dans l'eau que dans l'air. Vérifiez-le vous-même en vous baignant dans un lac boueux ou dans l'océan : à quelle distance pouvez-vous voir un ami ? Maintenant, demandez-lui de produire un son en frappant deux objets l'un contre l'autre ou en tapant dans ses mains. A quelle distance pouvez-vous entendre ce son ?

La vitesse du son dans l'air est d'environ 340 m/s ou 34,000 cm/s. Si un objet se trouve à une distance de 34 cm d'un robot, il ressort de l'équation 2.1 qu'un signal ultrasonore se déplacera vers l'objet et sera réfléchi :

Un circuit électronique peut facilement mesurer des périodes de temps en millisecondes.

L'avantage des capteurs à ultrasons est qu'ils ne sont pas sensibles aux changements de couleur ou de réflectivité des objets, ni à l'intensité lumineuse de l'environnement. Ils sont toutefois sensibles à la texture : le tissu absorbe une partie du son, tandis que le bois ou le métal le réfléchit presque entièrement. C'est pourquoi les rideaux, les tapis et les dalles de plafond souples sont utilisés pour rendre les pièces plus confortables.

Les capteurs à ultrasons sont relativement bon marché et peuvent fonctionner dans des environnements extérieurs. Ils sont utilisés dans les voitures pour détecter de courtes distances, par exemple pour faciliter le stationnement. Leur principal

inconvénient est que la mesure de la distance est relativement lente, car la vitesse du son est bien inférieure à celle de la lumière. Un autre inconvénient est qu'ils ne peuvent pas être focalisés afin de mesurer la distance à un objet spécifique.

2.2.2 Capteurs de proximité à infrarouge

La lumière infrarouge est une lumière dont la longueur d'onde est supérieure à celle de la lumière rouge, qui est la lumière de plus grande longueur d'onde que nos yeux peuvent voir. L'œil peut voir la lumière dont la longueur d'onde est comprise entre 390 et 700 nanomètres (un nanomètre est un millionième de millimètre). La lumière infrarouge a des longueurs d'onde comprises entre 700 et 1000 nanomètres. Elle est invisible pour l'œil humain et est donc utilisée dans la télécommande des téléviseurs et autres appareils électroniques.

Les capteurs de proximité sont des dispositifs simples qui utilisent la lumière pour détecter la présence d'un objet en mesurant l'intensité de la lumière réfléchie. L'intensité de la lumière diminue avec le carré de la distance de la source et cette relation peut être utilisée pour mesurer la distance approximative de l'objet. La mesure de la distance n'est pas très précise car l'intensité réfléchie dépend également de la réflectivité de l'objet. Un objet noir réfléchit moins de lumière qu'un objet blanc placé à la même distance. Un capteur de proximité ne peut donc pas faire la distinction entre un objet noir proche et un objet blanc placé un peu plus loin. C'est la raison pour laquelle ces capteurs sont appelés des capteurs de proximité et non des capteurs de distance. La plupart des robots éducatifs utilisent des capteurs de proximité car ils sont beaucoup moins chers que les capteurs de distance.

2.2.3 DéTECTEURS DE DISTANCE OPTIQUE

La distance peut être calculée à partir de la mesure du temps écoulé entre l'envoi d'un signal lumineux et sa réception. La lumière peut être de la lumière ordinaire ou de la lumière provenant d'un laser. La lumière produite par un laser est *cohérente* (voir ci-dessous). Le plus souvent, les lasers utilisés pour mesurer la distance utilisent la lumière infrarouge, bien que la lumière visible puisse également être utilisée. Les lasers présentent plusieurs avantages par rapport aux autres sources de lumière. Premièrement, les lasers sont plus puissants et peuvent détecter et mesurer la distance d'objets à longue distance. Deuxièmement, un faisceau laser est hautement focalisé, ce qui permet de mesurer précisément l'angle par rapport à l'objet (Fig. 2.3).

Lumière cohérente

Trois types de lumière sont représentés sur la Fig. 2.4. La lumière du soleil ou d'une ampoule est appelée *lumière blanche* car elle est composée de lumière de plusieurs couleurs (fréquences) différentes, émise à différents moments (phases) et émise dans différentes directions. Les

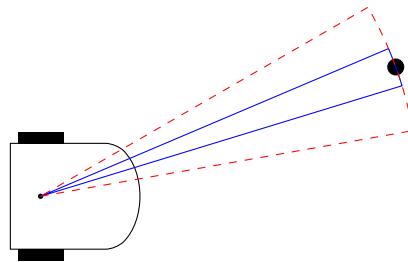


FIG. 2.3 – Largeur du faisceau de lumière laser (solide) et de lumière non-cohérente (pointillé)

diodes électroluminescentes (DEL) émettent une lumière monochromatique (lumière d'une seule couleur), mais elle n'est pas cohérente car ses phases sont différentes et elle est émise dans des directions différentes. Les lasers émettent une lumière cohérente : toutes les ondes ont la même fréquence et la même phase (le début de chaque onde est au même moment). Toute l'énergie d'une lumière est concentrée dans un faisceau étroit et la distance peut être calculée en mesurant le temps de vol et la différence de phase de la lumière réfléchie.

Supposons qu'une impulsion lumineuse soit transmise par le robot, réfléchie par un objet et reçue par un capteur sur le robot. La vitesse de la lumière dans l'air est

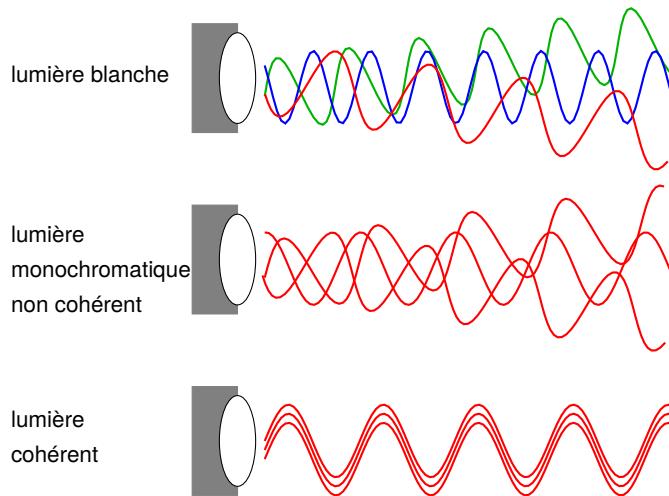


FIG. 2.4 – Lumière blanche, monochromatique et cohérente

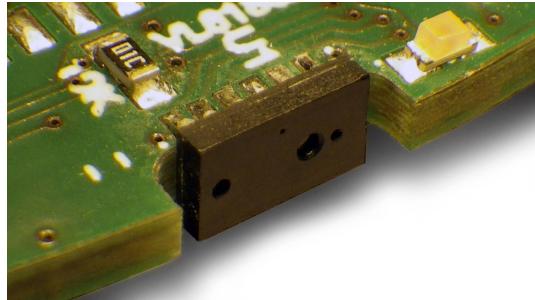


FIG. 2.5 – Un capteur de distance à temps de vol (noir) monté sur un circuit imprimé de 1,6 mm d'épaisseur (vert).

d'environ 300000000 m/s, soit 3×10^8 m/s ou 3×10^{10} cm/s en notation scientifique. Si un signal lumineux est dirigé vers un objet situé à 30 cm du robot, le temps d'émission et de réception du signal est (Fig. 2.5) :

$$\frac{2 \cdot 30}{3 \times 10^{10}} = \frac{2}{10^9} = 2 \times 10^{-9} \text{ secondes} = 0,002 \text{ microsecondes.}$$

C'est un laps de temps très court mais qui peut être mesuré par des circuits électroniques.

Le deuxième principe de mesure de distance par un faisceau lumineux est la triangulation. Dans ce cas, l'émetteur et le récepteur sont placés à des endroits différents. Le récepteur détecte le faisceau réfléchi à une position qui est fonction de la distance de l'objet par rapport au capteur.

2.2.4 Capteurs de triangulation

Avant d'expliquer le fonctionnement d'un *capteur de triangulation*, nous devons comprendre comment la réflexion de la lumière dépend de l'objet qu'elle frappe. Lorsqu'un faisceau étroit de lumière, comme la lumière cohérente d'un laser, frappe une surface brillante comme un miroir, les rayons lumineux rebondissent en un faisceau étroit. L'angle de réflexion par rapport à la surface de l'objet est le même que l'angle d'incidence. C'est ce qu'on appelle la réflexion spéculaire (Fig. 2.6). Lorsque la surface est rugueuse, la réflexion est *diffuse* (Fig. 2.7) dans toutes les directions car même les zones très proches de la surface ont des angles légèrement différents. La plupart des objets présents dans un environnement, comme les personnes et les murs, ont une réflexion diffuse. Pour détecter la lumière laser réfléchie, il n'est donc pas nécessaire de placer le détecteur à un angle précis par rapport à l'émetteur.

Les figures 2.8–2.9 montrent un capteur à triangulation simplifié détectant un objet à deux distances. Le capteur est constitué d'un émetteur laser et, à une distance d , d'une lentille qui focalise la lumière reçue sur un réseau de capteurs placés à une

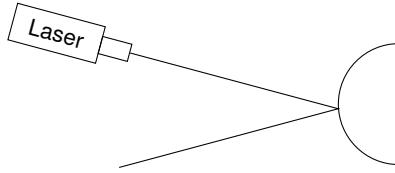


FIG. 2.6 – Réflexion spéculaire

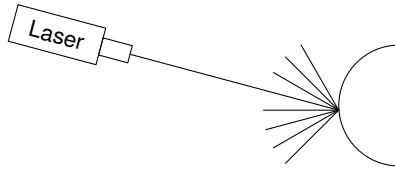


FIG. 2.7 – Réflexion diffuse

distance l derrière la lentille. En supposant que l'objet reflète la lumière de manière diffuse, une partie de la lumière sera collectée par la lentille et focalisée sur les capteurs. La distance d le long du réseau de capteurs est inversement proportionnelle à la distance s de l'objet par rapport à l'émetteur laser.

Les triangles $\triangle ll'd$ et $\triangle ss'd$ sont similaires, nous avons donc la formule :

$$\frac{s}{d} = \frac{l}{d'}.$$

Comme l et d sont fixés par construction, en mesurant d' à partir de l'indice du capteur qui détecte la lumière focalisée, on peut calculer s , la distance de l'objet au capteur. Le capteur doit être calibré en mesurant la distance s correspondant à chaque capteur dans la matrice, mais une fois qu'une table est stockée dans l'ordinateur, la distance s peut être effectuée par une consultation de la table.

De nombreux paramètres de conception affectent les performances d'un capteur de distance à triangulation : la puissance du laser, les caractéristiques optiques de l'objectif, le nombre de capteurs dans la matrice et leur sensibilité. Outre le compromis habituel entre performance et coût, le principal compromis se situe entre la portée et la distance minimale à laquelle un objet peut être mesuré. Pour une distance très courte s , la taille de la matrice de détecteurs d' devient très grande, ce qui impose une limite pratique à la distance minimale. La distance minimale peut être réduite en augmentant la distance entre l'émetteur laser et le réseau de détecteurs, mais cela réduit la portée. Un capteur à triangulation peut être caractérisé par la distance s_{opt} pour une performance optimale, la distance minimale et la portée autour de s_{opt} à laquelle les mesures peuvent être effectuées.

2.2.5 Scanner laser

Lorsque des capteurs à ultrasons ou des capteurs de proximité sont utilisés, il est possible de placer un petit nombre de capteurs autour du robot afin de détecter des objets situés à proximité du robot (Fig. 2.10). Bien entendu, l'angle par rapport à l'objet ne peut être mesuré avec précision, mais au moins l'objet sera détecté et le robot pourra s'en approcher ou l'éviter.

Avec un capteur laser, la largeur du faisceau est si faible qu'un grand nombre de lasers (coûteux) serait nécessaire pour détecter des objets à n'importe quel angle.

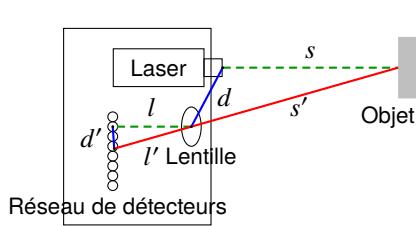


FIG. 2.8 – Triangulation d'un objet éloigné

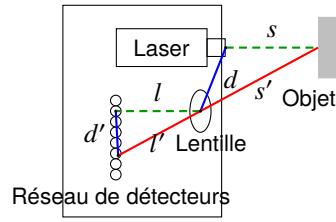


FIG. 2.9 – Triangulation d'un objet proche

Une meilleure conception consiste à monter un seul capteur laser sur un arbre rotatif pour former un *scanner laser* (Fig. 2.11). Un capteur angulaire peut être utilisé pour déterminer l'angle auquel un objet est détecté. Alternativement, l'ordinateur peut mesurer la période de temps après le passage du capteur rotatif dans une direction fixe. Une rotation complète de 360° permet à un scanner laser de générer un profil des objets dans l'environnement (Fig. 2.12).

Activity 2.1: Distance d'un capteur de distance

- Déterminez la portée maximale à laquelle les capteurs de proximité de votre robot peuvent détecter un objet. Existe-t-il également une portée minimale ou les objets peuvent-ils être détectés même s'ils sont placés en contact direct avec le capteur ? S'il existe une portée minimale, expliquez pourquoi des objets plus proches ne peuvent pas être détectés.
- Votre logiciel peut vous permettre de mesurer les valeurs numériques renvoyées par le capteur. Si tel est le cas, ces valeurs sont-elles des distances ou des valeurs arbitraires qui doivent être converties en distance ? S'il s'agit de valeurs arbitraires, trouvez une formule de conversion ou construisez un tableau qui donne les distances pour différentes valeurs renvoyées par le capteur.
- Un capteur qui n'utilise pas de lumière laser cohérente peut détecter un objet à sa gauche ou à sa droite, et pas seulement les objets qui se

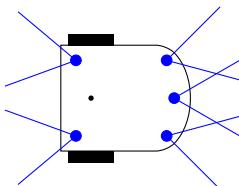


FIG. 2.10 – Cinq capteurs séparés

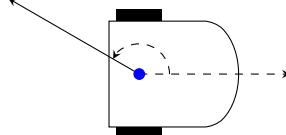


FIG. 2.11 – Un capteur rotatif

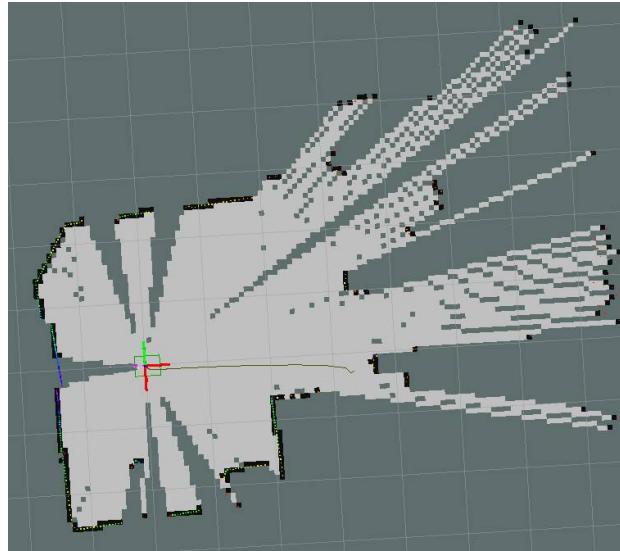


FIG. 2.12 – Carte de l'environnement obtenue par un scanner laser

trouvent directement devant lui. Mesurez l'angle auquel il est possible de détecter des objets. Les objets peuvent-ils être détectés au même angle à gauche et à droite du centre du capteur ?

- Combien de capteurs faudrait-il pour pouvoir détecter un objet placé n'importe où autour de votre robot ?

Activity 2.2: Thresholds

- Un robot mobile comme une voiture autonome ne s'arrête pas exactement devant un obstacle ; il laisse un espace supplémentaire pour la sécurité, peut-être 1 m ou 50 cm. Définissez un *threshold*, la distance minimale de sécurité par rapport à un objet, et programmez votre robot pour qu'il s'arrête à cette distance d'un objet.
- Si votre logiciel ne vous permet pas de mesurer les valeurs numériques renvoyées par le capteur, il peut vous permettre d'effectuer une action lorsque la valeur renvoyée passe un ou plusieurs seuils (par exemple, lorsque l'objet est " proche ", " moyen ", " éloigné "). Mesurez les distances correspondant à ces seuils : placez un objet à proximité du capteur et éloignez-le lentement. Notez les distances auxquelles les seuils sont

franchis.

Activity 2.3: Réflectivité

- Puisqu'un capteur de proximité infrarouge fonctionne en mesurant la lumière réfléchie par un objet, il est raisonnable de supposer que les valeurs mesurées dépendent de la caractéristique de l'objet. Répétez les expériences de l'activité 2.1 pour des objets de formes, de couleurs et de matériaux différents. Résumez vos conclusions.
- Essayez d'étendre la portée à laquelle votre capteur peut détecter un objet : utilisez un objet avec une surface métallique polie, fixez un miroir sur l'objet ou collez sur l'objet du ruban réfléchissant utilisé par les joggeurs et les cyclistes.
- Si votre robot dispose d'un capteur à ultrasons, réalisez ces expériences pour ce capteur et comparez différentes textures de la surface d'un objet.

Activity 2.4: Triangulation

- Utilisez un pointeur laser pour créer un faisceau vers un objet placé à environ 50 cm. Tamisez les lumières ou fermez les rideaux des fenêtres afin de pouvoir voir la réflexion du faisceau sur l'objet. Placez ensuite un appareil photo sur une table ou un trépied à environ 10 cm du laser et pointez-le sur l'objet ; prenez ensuite une photo. Éloignez l'objet et prenez une autre photo. Qu'observez-vous lorsque vous comparez les deux photos ?
- Déplacez l'objet de plus en plus loin du laser et de l'appareil photo, et notez les distances et la position du point sur la photo par rapport au bord de l'image. Tracez les données. Expliquez vos observations. Qu'est-ce qui définit la distance minimale et maximale que ce capteur peut mesurer ?

2.3 Caméras

Les appareils photo numériques sont largement utilisés en robotique car ils peuvent fournir des informations beaucoup plus détaillées que la simple distance et l'angle d'un objet. Les appareils photo numériques utilisent un composant électronique appelé *dispositif à couplage de charge* qui détecte les ondes lumineuses et renvoie une matrice de *éléments d'image*, ou *pixels* pour faire court (Fig. 2.13).

Les appareils photo numériques se caractérisent par le nombre de pixels capturés dans chaque image et par le contenu de ces pixels. Une petite caméra utilisée dans

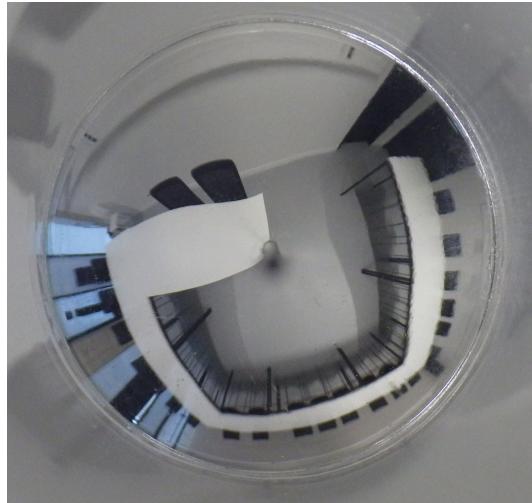


FIG. 2.13 – Une image capturée par une caméra omnidirectionnelle avec un champ de vision de 360 degrés.

un robot éducatif contient 192 rangées de 256 pixels chacune, soit un total de 49152 pixels. Il s'agit d'une très petite image : les capteurs des appareils photo numériques des smartphones enregistrent des images de millions de pixels.

Un appareil photo peut renvoyer les valeurs de chaque pixel en noir et blanc (1 bit par pixel), en nuances de gris appelées niveaux de gris (8 bits par pixel) ou en couleur rouge-vert-bleu (RVB) (3 fois 8 = 24 bits par pixel). La petite caméra de 256×192 a donc besoin d'environ 50 kilo-octets pour une seule image en niveaux de gris ou 150 kilo-octets pour une image en couleurs. Étant donné qu'un robot mobile tel qu'une voiture à conduite autonome devra stocker plusieurs images par seconde (les films et la télévision affichent 24 images par seconde), la mémoire nécessaire pour stocker et analyser les images peut être très importante.

Une caractéristique importante dans la conception d'une caméra pour un robot est le *champ de vision* de son objectif. Étant donné la position de la caméra, quelle partie de la sphère entourant la caméra est capturée dans l'image ? Pour un capteur donné dans une caméra, un objectif avec un champ de vision étroit capture une petite zone avec une haute résolution et peu de distorsion, tandis qu'un objectif avec un champ de vision large capture une grande zone avec une résolution plus faible et plus de distorsion. La distorsion la plus extrême provient d'une *caméra omnidirectionnelle* qui capture en une seule image (presque) toute la sphère qui l'entoure. La figure 2.13 montre l'image d'une salle de conférence prise par une caméra omnidirectionnelle ; la position de la caméra est indiquée par le point noir au centre. Les caméras à large champ de vision sont utilisées dans les robots mobiles car l'image peut être

analysée pour comprendre l'environnement. L'analyse de l'image est utilisée pour la navigation, pour détecter des objets dans l'environnement et pour interagir avec des personnes ou d'autres robots en utilisant des propriétés visuelles comme la couleur.

Le problème fondamental des caméras en robotique est que nous ne sommes pas intéressés par un ensemble de pixels "bruts", mais par l'identification des objets présents dans l'image. L'œil et le cerveau humains effectuent instantanément des tâches de reconnaissance : lorsque nous conduisons une voiture, nous identifions automatiquement les autres voitures, les piétons, les feux de circulation et les obstacles sur la route, et nous prenons les mesures appropriées. Le traitement d'images par un ordinateur nécessite des algorithmes sophistiqués et une puissance de traitement importante (Chap. 12). Pour cette raison, les robots équipés de caméras sont beaucoup plus complexes et coûteux que les robots éducatifs qui utilisent des capteurs de proximité.

2.4 Autres capteurs

Un *capteur tactile* peut être considéré comme un capteur de distance simplifié qui ne mesure que deux valeurs : la distance à un objet est nulle ou supérieure à zéro. Les capteurs tactiles sont fréquemment utilisés comme mécanismes de sécurité. Par exemple, un capteur tactile est monté sur la partie inférieure des petits chauffages d'appoint afin que le chauffage ne fonctionne que si le capteur tactile détecte le sol. Si le radiateur tombe, le capteur tactile détecte qu'il n'est plus en contact avec le sol et le chauffage s'arrête pour éviter un incendie. Un capteur tactile peut être utilisé sur un robot mobile pour appliquer un frein d'urgence si le robot s'approche trop près d'un mur.

Des boutons et des interrupteurs permettent à l'utilisateur d'interagir directement avec le robot.

Un *microphone* sur le robot lui permet de détecter le son. Le robot peut simplement détecter le son ou utiliser des algorithmes pour interpréter les commandes vocales.

Un *accelerometer* mesure l'accélération. L'utilisation principale des accéléromètres est de mesurer la direction de la force gravitationnelle qui provoque une accélération d'environ $9,8 \text{ m/sec}^2$ vers le centre de la terre. Avec trois accéléromètres montés perpendiculairement les uns aux autres (Fig. 2.14), on peut mesurer la *attitude* du robot : les trois angles du robot, appelés *pitch*, *yaw* et *roll*. Les accéléromètres sont abordés plus en détail dans la Sec. 5.9.1 et une tâche utilisant des accéléromètres est présentée dans la Sec. 14.4.1.

Activity 2.5: Mesure de l'attitude à l'aide d'accéléromètres

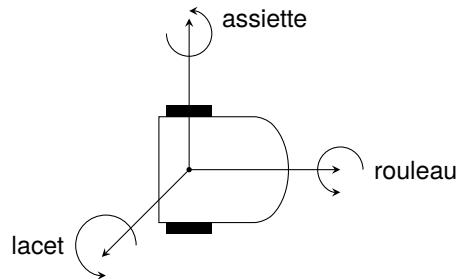


FIG. 2.14 – Accéléromètre trois axes

- Écrivez un programme qui affiche l'attitude de votre robot lorsque vous le prenez et le faites tourner autour des trois axes.
- Implémentez un jeu de votre choix en utilisant le robot comme contrôleur.
- Écrivez un programme qui fait avancer le robot et qui s'arrête si une pente est atteinte. Utilisez l'accéléromètre qui mesure l'inclinaison.

2.5 Etendue de mesure, résolution, précision, exactitude

Lorsqu'une quantité physique est mesurée, la mesure peut être caractérisée par son étendue, sa résolution, sa précision et son exactitude, des concepts souvent confondus.

L'gamme est l'étendue de l'ensemble des valeurs qui peuvent être mesurées par un capteur. Un capteur de proximité infrarouge pourrait être capable de mesurer des distances comprises entre 1 cm et 30 cm. Comme les faisceaux laser concentrent beaucoup de puissance dans un faisceau étroit, ils ont une portée beaucoup plus grande. La portée requise par un capteur de distance pour un robot se déplaçant dans un bâtiment sera d'environ 10 m, tandis qu'un capteur de distance pour une voiture autonome doit mesurer des distances d'environ 100 m.

Resolution fait référence au plus petit changement qui peut être mesuré. Un capteur de distance peut indiquer des distances en centimètres (1 cm, 2 cm, 3 cm, 4 cm, ...), tandis qu'un meilleur capteur indique des distances en centièmes de centimètres (4,00 cm, 4,01 cm, 4,02 cm, ...). Pour une voiture autonome, une résolution de l'ordre du centimètre devrait suffire : on ne garerait pas une voiture à 1 cm d'une autre, sans parler de la garer à 0,1 cm. Un robot chirurgical a besoin d'une résolution beaucoup plus élevée, car le moindre millimètre est critique lors d'une opération.

Précision fait référence à la cohérence de la mesure. Si la même quantité est mesurée à plusieurs reprises, la même valeur est-elle retournée ? La précision est très importante car des mesures incohérentes conduiront à des décisions incohérentes. Supposons qu'un capteur d'une voiture autonome mesure les distances à 10 cm près, mais que les mesures successives renvoient une large gamme de valeurs (disons, 250 cm, 280 cm, 210 cm). Lorsqu'elle essaie de maintenir une distance fixe avec un véhicule qu'elle suit, la voiture accélère et ralentit sans raison valable, ce qui rend la conduite inconfortable et gaspille de l'énergie.

Très souvent, un capteur aura une haute résolution mais une faible précision ; dans ce cas, on ne peut pas se fier à la résolution. Par exemple, un capteur de distance peut renvoyer des valeurs en millimètres, mais si la précision n'est pas suffisamment élevée, par exemple 45 mm, 43 mm, 49 mm, le capteur ne peut renvoyer des valeurs que dans le centimètre ou le demi-centimètre le plus proche.

Activity 2.6: Précision et résolution

- Quelle est la résolution des capteurs de distance de votre robot ?
- Placez un objet à une distance fixe de votre robot et enregistrez à plusieurs reprises la distance mesurée. Quelle est la précision de la mesure ?
- Mesurez la distance d'un objet dans différentes circonstances telles que des changements de température et de lumière. Allumez et éteignez le chauffage ou le climatiseur; allumez et éteignez les lumières. Les mesures changent-elles ?

Accuracy fait référence à la proximité d'une mesure par rapport à la quantité réelle mesurée. Si un capteur de distance affirme systématiquement que la distance est supérieure de 5 cm à ce qu'elle est en réalité, le capteur n'est pas précis. En robotique, l'exactitude n'est pas aussi importante que la précision, car la mesure d'un capteur ne renvoie pas directement une quantité physique. Au lieu de cela, un calcul est effectué pour obtenir une quantité physique telle que la distance ou la vitesse à partir d'une valeur électronique mesurée. Si l'imprécision est constante, la valeur du capteur peut être calibrée pour obtenir la véritable quantité physique (Sect. 2.6). Un capteur de distance utilisant la lumière ou le son calcule la distance à partir du temps de vol d'un signal $s = vt/2$. Si l'on sait que le capteur renvoie systématiquement une valeur de 5 cm trop grande, l'ordinateur peut simplement utiliser la formule $s = (vt/2) - 5$.

Activity 2.7: Accuracy

- Placez un objet à différentes distances du robot et mesurez les distances renvoyées par le capteur. Les résultats sont-ils précis ? Si non,

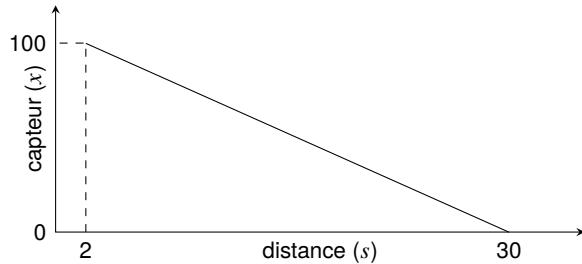


FIG. 2.15 – Valeur renvoyée comme une fonction linéaire de la distance

pouvez-vous écrire une fonction qui transforme les mesures du capteur en distances ?

2.6 Nonlinearity

Les capteurs renvoient des quantités électroniques telles que le potentiel ou le courant qui sont proportionnelles à ce qui est mesuré. Les valeurs analogiques sont converties en valeurs numériques. Par exemple, un capteur de proximité peut renvoyer 8 bits de données (valeurs comprises entre 0 et 255) qui représentent une gamme de distances, peut-être de 0 à 50 cm. Un capteur de 8 bits ne peut même pas renvoyer des angles de l'ordre de 0°–360° avec une résolution d'un degré. L'ordinateur doit traduire les valeurs numériques en mesures d'une quantité physique. La découverte de la correspondance pour cette traduction est appelée *calibration*. Dans le meilleur des cas, la correspondance sera linéaire et facile à calculer. Dans le cas contraire, si la correspondance est non linéaire, il faut utiliser une table ou une fonction non linéaire. Les tables sont plus efficaces car la recherche d'une entrée est plus rapide que le calcul d'une fonction, mais les tables nécessitent beaucoup de mémoire.

2.6.1 Capteurs linéaires

Si un capteur de distance horizontal est *linéaire*, il existe une correspondance $x = as + b$, où x est la valeur renvoyée par le capteur, s est la distance d'un objet par rapport au capteur et a, b sont des constantes (a est la pente et b l'interception avec l'axe du capteur). Supposons que le capteur renvoie la valeur 100 pour un objet à 2 cm et la valeur 0 pour un objet à 30 cm (Fig. reffig.linear).

Calculons la pente et l'ordonnée à l'origine :

$$\text{slope} = \frac{\Delta x}{\Delta s} = \frac{0 - 100}{30 - 2} = -3.57 ..$$

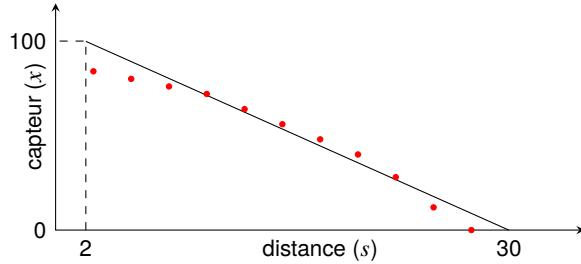


FIG. 2.16 – Valeurs expérimentales renvoyées en fonction de la distance

Lorsque $s = 30$, $x = 0 = -3,57 \cdot 30 + b$, donc $b = 107$ et $x = -3,57s + 107$. La résolution de s donne une fonction que l'ordinateur du robot peut utiliser pour associer une valeur de capteur à la distance correspondante :

$$s = \frac{107 - x}{3,57}.$$

Activity 2.8: Linéarité

- Posez une règle sur votre table et placez soigneusement le robot de sorte que son capteur avant soit placé à côté du repère 0 de la règle. Placez un objet à côté de la marque de 1 cm sur la règle. Notez la valeur renvoyée par le capteur. Répétez l'opération pour 2 cm, 3 cm, . . . , jusqu'à ce que la valeur renvoyée devienne nulle.
- Tracez un graphique de la valeur renvoyée en fonction de la distance. La réponse du capteur est-elle linéaire ? Si oui, calcule la pente et l'ordonnée à l'origine.
- Répétez l'expérience avec des objets de formes et de matériaux différents. La linéarité du graphique dépend-elle des caractéristiques de l'objet ?

2.6.2 Capteurs non linéaires

La figure 2.16 montre un résultat possible des mesures de l'activité 2.8. Les mesures sont représentées par des points et la fonction linéaire de la figure `reffig.linear`. La fonction est raisonnablement linéaire au milieu de sa plage mais non linéaire en dehors de cette plage. Cela signifie qu'il est impossible d'utiliser une fonction linéaire des valeurs brutes du capteur pour obtenir la distance d'un objet par rapport au robot.

Nous pouvons construire un tableau pour faire correspondre les valeurs des capteurs aux distances. Le tableau 2.1 est un tableau basé sur des mesures réelles

$s(\text{cm})$	x	x_l
18	14	12
16	18	16
14	22	20
12	26	24
10	29	28
8	32	32
6	36	36
4	41	40
2	44	44

TAB. 2.1 – Tableau de correspondance entre les valeurs des capteurs et les distances

avec un robot éducatif. Les mesures ont été effectuées tous les deux centimètres, de 2 cm à 18 cm ; à 20 cm, le capteur ne détecte plus l'objet. La deuxième colonne indique la valeur du capteur pour chaque distance. La troisième colonne indique les valeurs x_l qui seraient renvoyées par le capteur s'il était linéaire avec la fonction $x = -2s + 48$. Nous voyons que les valeurs réelles renvoyées par le capteur ne s'écartent pas trop de la linéarité, il ne serait donc pas déraisonnable d'utiliser une fonction linéaire.

Évidemment, il serait préférable d'avoir une entrée de tableau pour chacune des valeurs possibles renvoyées par le capteur. Cependant, cela prendrait beaucoup de mémoire et pourrait s'avérer peu pratique si la plage des valeurs renvoyées par le capteur est beaucoup plus large, par exemple de 0 à 4095 (12 bits). Une solution consiste à prendre la valeur la plus proche, de sorte que si la valeur 27 est renvoyée par le capteur dont le mappage est donné dans le Tableau 2.1, la distance sera de 12.

Une meilleure solution consiste à utiliser l'interpolation. Si vous regardez à nouveau le graphique de la Fig. 2.16, vous pouvez constater que les segments de la courbe sont à peu près linéaires, bien que leurs pentes changent en fonction de la courbe. Par conséquent, nous pouvons obtenir une bonne approximation de la distance correspondant à une valeur de capteur en prenant la distance relative sur une ligne droite entre deux points (Fig. reffig.interpolation). Étant donné les distances s_1 et s_2 correspondant aux valeurs de capteur x_1 et x_2 , respectivement, pour une valeur $x_1 < x < x_2$, sa distance s :

$$s = s_1 + \frac{s_2 - s_1}{x_2 - x_1} (x - x_1).$$

2.7 Summary

When designing a robot, the choice of sensors is critical. The designer needs to decide *what* needs to be measured : distance, attitude, velocity, etc. Then the

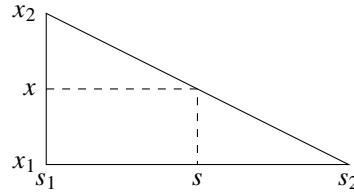


FIG. 2.17 – *Interpolation of sensor values*

designer has to make trade-offs : larger range, finer resolution, higher precision and accuracy are always better, but come at a price. For educational robots, price is the overriding consideration, so don't expect good performance from your robot. Nevertheless, the algorithmic principles are the same whether the sensors are of high quality or not, so the trade-off does not affect the ability to learn with the robot.

Any sensor connected to the robot's computer is going to return discrete values within a fixed range. The computer must be able to map these sensor values to physical quantities in a process called calibration. If the sensor is linear, the calibration results in two values (slope and intercept) that determine a linear function. If the sensor is nonlinear, a table or a non-linear function must be used.

2.8 Further reading

For an overview of sensors used in mobile robots see [43, Section 4.1]. The book by Everett [16] is devoted entirely to this topic.

Chapitre 3

Comportement réactif

Nous sommes maintenant prêts à écrire nos premiers algorithmes pour les robots. Ces algorithmes démontrent un *comportement réactif* : un événement (tel que la détection d'un objet proche par le robot) amène le robot à réagir en effectuant une action qui modifie son comportement (comme l'arrêt des moteurs). On parle de comportement purement réactif lorsque l'action est liée uniquement à l'occurrence d'un événement et ne dépend pas des données stockées en mémoire (état).

Les comportements réactifs sont ceux des *véhicules de Braitenberg* qui sont appropriés pour introduire la robotique car un comportement complexe découle d'algorithmes simples. Les sections 3.1–3.3 décrivent les véhicules de Braitenberg qui ont un comportement réactif ; dans le chapitre 4 nous présentons les véhicules de Braitenberg qui ont un comportement non réactif avec des états. La section `refs.line` présente plusieurs algorithmes pour le comportement réactif classique du suivi de ligne. Le suivi de ligne est une tâche intéressante car les algorithmes sont sensibles aux caractéristiques des capteurs et des lignes. Une calibration est nécessaire pour déterminer les seuils optimaux pour un mouvement rapide et robuste du robot. La section 3.5 donne un bref aperçu de la formulation originale des véhicules de Braitenberg dans une approche biologique avec des capteurs connectés directement aux moteurs, et non à travers un ordinateur. Plus loin dans le livre (Sect. 13.3) nous discutons de l'implémentation des véhicules de Braitenberg en utilisant des réseaux de neurones.

3.1 Véhicules Braitenberg

Valentino Braitenberg est un neuroscientifique qui a décrit la conception de véhicules virtuels présentant un comportement étonnamment complexe. Des chercheurs du MIT Media Lab ont développé des implémentations matérielles de ces véhicules à partir de *briques programmables* qui étaient les précurseurs des LEGO®. Mindstorms.¹ Ce chapitre décrit une implémentation de la plupart des véhicules Braitenberg du rapport du MIT. Le matériel du MIT utilisait des capteurs de lumière et de toucher, alors que notre robot générique est basé sur des capteurs de proximité horizontaux.

Pour faciliter la comparaison avec le rapport du MIT (et indirectement avec le livre de Braitenberg), les noms de leurs véhicules ont été conservés, même s'il peut être difficile de comprendre leur signification dans nos implémentations.

1. Le rapport du MIT utilise le terme *créatures de Braitenberg* mais nous conservons le terme original.

Algorithm 3.1: Timide
<pre> 1: when object not detected in front 2: left-motor-power ← 100 3: right-motor-power ← 100 4: 5: when object detected in front 6: left-motor-power ← 0 7: right-motor-power ← 0 </pre>

Deux véhicules sont présentés en détail en donnant tout ce qui suit :

- La spécification du comportement du robot;
- Un algorithme formalisé pour le comportement spécifié;
- Une activité qui vous demande d’implémenter l’algorithme sur votre robot.

Les autres véhicules sont présentés dans des activités qui spécifient le comportement et vous demandent de développer un algorithme et de l’implémenter sur votre robot.

3.2 Réaction à la détection d’un objet

Spécification (Timid) : Lorsque le robot ne détecte pas d’objet, il se déplace vers l’avant. Lorsqu’il détecte un objet, il s’arrête.

L’algorithme 3.1 implémente ce comportement.

L’algorithme utilise deux gestionnaires d’événements, un pour l’événement de détection d’un objet et un pour l’événement de non-détection d’un objet. Les gestionnaires d’événements sont écrits en utilisant l’instruction `when` dont la signification est :

lorsque l’événement *premier* se produit, effectuez les actions suivantes.

Pourquoi utilisons-nous cette construction et non l’instruction `while` plus familière (Algorithme 3.2) ?

Si l’on utilise l’instruction `while`, *as tant* que l’objet n’est pas détecté, les moteurs sont activés, et *tant* que l’objet est détecté, les moteurs sont désactivés. Comme le capteur détectera l’objet sur une certaine distance, les moteurs seront activés ou désactivés à plusieurs reprises. Il est probable qu’aucun dommage ne sera causé si un moteur déjà éteint est éteint et un moteur déjà allumé est allumé, mais ces commandes répétées ne sont pas nécessaires et peuvent gaspiller des ressources. Par conséquent, nous préférerons éteindre les moteurs uniquement lorsque l’objet est détecté pour la première fois et les allumer uniquement lorsque l’objet n’est pas détecté pour la première fois. L’instruction `when` donne la sémantique que nous souhaitons.

Algorithm 3.2: Timide avec while

```

1: while object not detected in front
2:     left-motor-power ← 100
3:     right-motor-power ← 100
4:
5: while object detected in front
6:     left-motor-power ← 0
7:     right-motor-power ← 0

```

Activity 3.1: Timide

- Implémente le comportement Timide.

Activity 3.2: Indécisif

- Implémente le comportement Indécisif.

Spécification (Indécisif) : Lorsque le robot ne détecte pas d'objet, il se déplace vers l'avant. Lorsqu'il détecte un objet, il recule.

- A la bonne distance, le robot va *osciller*, c'est-à-dire qu'il va avancer et reculer en succession rapide. Mesurez cette distance pour votre robot et pour des objets de réflectivité différente.

Activity 3.3: Dogged

- Implémente le comportement Dogged.

Spécification (Dogged) : Lorsque le robot détecte un objet à l'avant, il recule. Lorsque le robot détecte un objet à l'arrière, il se déplace vers l'avant.

Activity 3.4: Dogged (arrêt)

- Implémente le comportement Dogged (arrêt).

Spécification (Dogged (arrêt)) : Comme dans Activity 3.3, mais lorsqu'un objet n'est pas détecté, le robot s'arrête.

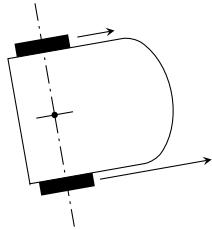


FIG. 3.1 – Tourner doucement à gauche

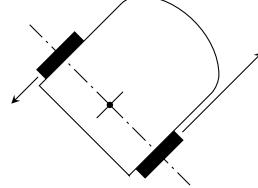


FIG. 3.2 – Virage brusque à gauche

Activity 3.5: Attractive et répulsive

— Met en œuvre le comportement Attractif et repoussant.

Spécification (Attraction et répulsion) : Lorsqu'un objet s'approche du robot par derrière, le robot s'enfuit jusqu'à ce qu'il soit hors de portée.

3.3 Réaction et virage

Une voiture tourne en modifiant l'angle de ses roues avant par rapport au châssis du véhicule. La puissance du moteur n'est pas modifiée. Un robot à entraînement différentiel ne possède pas de mécanisme de direction (comme le volant d'une voiture ou le guidon d'un vélo). Au lieu de cela, il tourne en réglant différents niveaux de puissance sur les roues gauche et droite. Si une roue tourne plus vite que l'autre, le robot tourne dans la direction opposée à celle de la roue la plus rapide (3.1). Si une roue tourne en arrière tandis que l'autre tourne en avant, le virage est beaucoup plus serré (Fig. 3.2). Sur les figures, les flèches indiquent la direction et la vitesse de chaque roue. Le *rayon de virage* est le rayon du cercle qui constitue la trajectoire du robot. On dit qu'un virage est plus serré si le rayon est plus petit. À l'extrême, si une roue tourne en avant et la seconde en arrière à la même vitesse, le robot tourne sur place et le rayon de braquage est nul.

Implémentons maintenant un véhicule de Braitenberg dont la spécification impose au robot de tourner.

Spécification (Paranoïde) : Lorsque le robot détecte un objet, il se déplace vers l'avant (en entrant en collision avec l'objet). Lorsqu'il ne détecte pas d'objet, il se tourne vers la gauche.

L'algorithme 3.3 met en œuvre ce comportement.

Algorithm 3.3: Paranoïde

```

1: when both sensors detect black
2:     left-motor-power ← 100
3:     right-motor-power ← 100
4:
5: when neither sensor detects black
6:     left-motor-power ← 0
7:     right-motor-power ← 0
8:
9: when only the left sensor detects black
10:    left-motor-power ← 0
11:    right-motor-power ← 50
12:
13: when only the right sensor detects black
14:    left-motor-power ← 50
15:    right-motor-power ← 0

```

Activity 3.6: Paranoïde

- Implémente le comportement Paranoïde.
- Dans l'algorithme, les moteurs gauche et droit sont réglés sur des puissances égales mais opposées. Expérimentez ces niveaux de puissance pour voir leur influence sur le rayon de braquage du robot.

Activity 3.7: Paranoïde (droite-gauche)

- Implémente le comportement Paranoïaque (droite-gauche).

Spécification (Paranoïaque (droite-gauche)) : Lorsqu'un objet est détecté devant le robot, ce dernier se déplace vers l'avant. Lorsqu'un objet est détecté à droite du robot, ce dernier tourne à droite. Lorsqu'un objet est détecté à la gauche du robot, le robot tourne à gauche. Lorsqu'aucun objet n'est détecté, le robot ne bouge pas.

Activity 3.8: Insécurité

- Implémente le comportement de Insécurité.

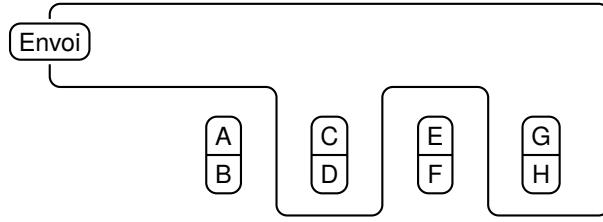


FIG. 3.3 – Un entrepôt robotisé

Spécification (Insécurité) : Si aucun objet n'est détecté à gauche du robot, réglez le moteur droit pour qu'il tourne vers l'avant et désactivez le moteur gauche. Si un objet est détecté à gauche du robot, désactivez le moteur droit et désactivez le moteur gauche pour qu'il tourne vers l'avant.

- Expérimenez avec les paramètres du moteur jusqu'à ce que le robot suive un mur sur sa gauche.

Activity 3.9: Conduite (action entraînée)

- Implémentez le comportement Conduite.

Spécification (Conduite) : Si un objet est détecté à gauche du robot, réglez le moteur droit pour qu'il tourne vers l'avant et désactivez le moteur gauche. Si un objet est détecté à droite du robot, désactivez le moteur droit et désactivez le moteur gauche pour qu'il tourne vers l'avant.

- Expérimenez les paramètres du moteur jusqu'à ce que le robot s'approche de l'objet en zigzag.

3.4 Ligne suivante

Considérons un entrepôt équipé de chariots robotisés qui apportent des objets à une zone de distribution centrale (3.3). Des lignes sont peintes sur le sol de l'entrepôt et le robot a pour instruction de suivre ces lignes jusqu'à ce qu'il atteigne l'emplacement de l'objet désiré.

Line following est une tâche qui fait ressortir toute l'incertitude de la construction de robots dans le monde réel. La ligne peut ne pas être parfaitement droite, la poussière peut masquer une partie de la ligne ou une roue peut se déplacer plus

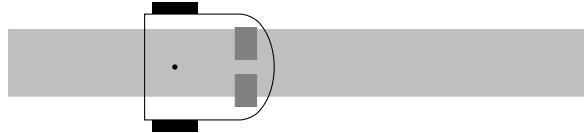


FIG. 3.4 – Un robot avec deux capteurs de sol sur une ligne

lentement que l'autre à cause de la poussière. Pour suivre une ligne, le robot doit décider s'il est sur la ligne ou non, et s'il commence à quitter la ligne d'un côté, il doit tourner dans la bonne direction pour regagner la ligne.

3.4.1 Suivi de ligne avec une paire de capteurs au sol

Pour suivre une ligne, il est possible d'utiliser une paire de capteurs au sol (Fig. 3.4).² La ligne doit être noire pour augmenter le contraste avec le sol blanc, mais la figure affiche la ligne en gris clair afin de ne pas masquer le robot et ses capteurs. Des seuils sont utilisés pour déterminer le moment où un capteur passe de la détection de la ligne à la détection du sol ou inversement.

La ligne doit être suffisamment large pour que les deux capteurs de sol détectent l'obscurité lorsque le robot se trouve directement au-dessus de la ligne. Il n'est pas nécessaire que les capteurs soient entièrement au-dessus de la ligne ; il suffit que la quantité de lumière réfléchie par la ligne sur le capteur soit inférieure au seuil défini pour le noir.

Pour mettre en œuvre le suivi de ligne, le robot doit avancer chaque fois que les deux capteurs détectent une surface sombre, indiquant qu'il se trouve sur la ligne. Si le robot commence à quitter la ligne, le capteur de sol gauche ou droit quittera la ligne en premier (Fig.3.5) :



FIG. 3.5 – Quitter la ligne

2. La figure montre une vue de dessus bien que les capteurs au sol se trouvent en bas du robot. Un capteur de sol sur un sol clair détectera beaucoup de lumière réfléchie. Si une ligne sombre est peinte sur le sol, le capteur détectera très peu de lumière réfléchie lorsqu'il se trouvera au-dessus de la ligne. Si votre sol est de couleur foncée, vous devez utiliser une ligne blanche.

Algorithm 3.4: Suivi de ligne avec deux capteurs

```

integer black-threshold ← 20
integer white-threshold ← 80

1: when black-threshold ≤ sensor value ≤ white-threshold
2:     left-motor-power ← 100
3:     right-motor-power ← 100
4:
5: when sensor value > white-threshold
6:     left-motor-power ← -50
7:     right-motor-power ← 50
8:
9: when black-threshold < sensor value
10:    left-motor-power ← 50
11:    right-motor-power ← -50

```

- Si le robot sort de la ligne vers le *gauche*, le capteur du *gauche* ne détectera pas la ligne alors que le capteur du *droite* la détecte toujours ; le robot doit tourner vers le *droite*.
- Si le robot sort de la ligne vers l'*droite*, le capteur de l'*droite* ne détectera pas la ligne tant que le capteur de l'*gauche* la détecte encore ; le robot doit tourner vers l'*gauche*.

Pour l'instant, nous spécifions que le robot s'arrête dès qu'aucun des deux capteurs ne détecte la ligne.

L'algorithme 3.4 formalise la description informelle ci-dessus.

Activity 3.10: Suivi de ligne avec deux capteurs

- Implémentation de l'algorithme 3.4.
- Utilisez du ruban adhésif noir d'électricien ou du ruban adhésif de gaffer pour créer une ligne sur le sol. (Le ruban adhésif de gaffer est utilisé sur les plateaux de théâtre et de cinéma pour attacher les câbles ou les fixer au sol. Le ruban de gaffer est généralement moins réfléchissant que le ruban d'électricien et constitue donc un meilleur choix pour la mise en œuvre d'algorithmes de suivi de ligne).
- La ligne doit présenter des angles ou des courbes qui amèneront le robot à sortir de la ligne. Exécutez le programme et vérifiez que le robot peut suivre la ligne avec ses angles et ses courbes.

- Expérimitez la puissance des moteurs pour revenir sur la ligne. Si le virage est trop doux, l'autre capteur risque de sortir de la ligne avant que le robot ne fasse demi-tour. Si le virage est trop serré, le robot risque de sortir de l'autre côté de la ligne. Dans tous les cas, les virages serrés peuvent être dangereux pour le robot et provoquer la chute de ce qu'il transporte.
- La vitesse d'avancement du robot sur la ligne est également importante. Si elle est trop rapide, le robot peut sortir de la ligne avant que les virages n'affectent sa direction. Si la vitesse est trop lente, personne n'achètera votre robot pour l'utiliser dans un entrepôt. À quelle vitesse votre robot peut-il suivre la ligne sans la quitter ?

Activity 3.11: Différentes configurations de ligne

- Quel est le virage le plus serré que votre robot peut suivre ? Peut-il suivre une ligne qui a un virage de 90° ?
- Expérimitez différentes configurations de la ligne : virages doux, virages serrés et lignes en zigzag.
- Faites des expériences avec la largeur de la ligne. Que se passe-t-il si la ligne est beaucoup plus large ou plus étroite que la distance entre les capteurs ?

Activity 3.12: Reprendre la ligne après l'avoir perdue

- Modifiez l'algorithme pour que le robot retourne sur la ligne après l'avoir perdue, c'est-à-dire lorsque les deux capteurs ne détectent plus de noir.
- Algorithme 1 : avant que les deux capteurs ne détectent plus la ligne, l'un d'eux sera le premier à ne plus détecter la ligne. Utilisez une variable pour vous souvenir du capteur qui a perdu la ligne en premier ; lorsque la ligne est perdue, tournez dans la direction opposée à la valeur de cette variable.
- Algorithme 2 : l'algorithme précédent peut ne pas fonctionner si le robot se déplace trop rapidement et sort de la ligne avant de détecter qu'un seul capteur a perdu la ligne. Au lieu de cela, si les deux capteurs ne détectent plus de noir, faites en sorte que le robot recherche la ligne sur une courte distance, d'abord dans une direction, puis dans l'autre.

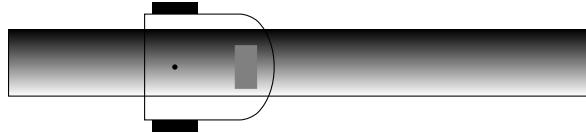


FIG. 3.6 – Une ligne avec un dégradé de gris

Activity 3.13: Configuration du capteur

- Discutez de l'effet que les modifications suivantes apportées au robot auraient sur sa capacité à suivre une ligne :
- Les événements de détection du sol se produisent plus souvent ou moins souvent.
- Les capteurs sont plus éloignés ou plus proches les uns des autres.
- Il y a plus de deux capteurs de sol sur la partie inférieure du robot.
- Les capteurs sont situés à l'arrière du robot ou le robot se déplace en arrière.
- Expérimenez ces changements s'ils peuvent être effectués sur votre robot.

3.4.2 Suivi de ligne avec un seul capteur au sol

Un robot peut suivre une ligne avec un seul capteur au sol si la réflectivité de la ligne varie sur sa largeur. La figure 3.6 montre une ligne en niveaux de gris dont la teinte varie continuellement du noir au blanc sur toute sa largeur. Le capteur de sol renvoie des valeurs comprises entre 0 et 100 en fonction de la partie de la ligne sur laquelle il se trouve.

Lorsque le robot se trouve directement au-dessus de la ligne, le capteur renvoie la valeur 50 qui se situe à mi-chemin entre le noir et le blanc. Bien sûr, nous ne nous attendons pas à ce que la valeur soit exactement 50, il n'y a donc aucune raison pour que le robot tourne à gauche ou à droite à moins que la valeur ne s'approche de 0 ou 100. Nous définissons deux seuils :

- **seuil noir** : en dessous de cette valeur, le robot quitte le côté gauche de la ligne.
- **seuil blanc** : au-dessus de cette valeur, le robot quitte le côté droit de la ligne.

L'algorithme 3.5 modifie les paramètres de puissance du moteur lorsque la valeur renvoyée par le capteur franchit les seuils.

Algorithm 3.5: Suivi de ligne avec un capteur	
	integer black-threshold $\leftarrow 20$
	integer white-threshold $\leftarrow 80$
1:	when black-threshold \leq sensor value \leq white-threshold
2:	left-motor-power $\leftarrow 100$
3:	right-motor-power $\leftarrow 100$
4:	
5:	when sensor value $>$ white-threshold
6:	left-motor-power $\leftarrow -50$
7:	right-motor-power $\leftarrow 50$
8:	
9:	when black-threshold $<$ sensor value
10:	left-motor-power $\leftarrow 50$
11:	right-motor-power $\leftarrow -50$

Activity 3.14: Ligne suiveuse avec un capteur

- Implémentation de l'algorithme 3.5.
- Expérimitez avec les seuils jusqu'à ce que le robot puisse suivre la ligne.
- Modifiez l'algorithme pour que le robot détecte le moment où il a complètement quitté la ligne. Conseil : le seul problème concerne le cas où le robot sort du côté noir de la ligne, car l'algorithme ne fait pas la distinction entre ce cas et celui où il sort du côté blanc de la ligne. Une solution consiste à utiliser une variable pour se souvenir de la valeur précédente du capteur, afin de pouvoir distinguer les deux cas.

Activity 3.15: Line following with proportional correction

- Puisque les valeurs du capteur sont proportionnelles à l'échelle de gris de la ligne, la distance approximative du capteur par rapport au centre de la ligne peut être calculée. Modifiez l'algorithme pour calculer cette distance.
- Modifiez l'algorithme pour que le réglage du moteur soit proportionnel à cette distance.
- Faites des expériences pour différentes constantes de proportion et ex-

pliquez les résultats.

3.4.3 Suivi de ligne sans gradient

Le récepteur d'un capteur de proximité possède une *aperture*, une ouverture à travers laquelle la lumière est collectée. Les ouvertures sont souvent larges pour permettre à une plus grande quantité de lumière de tomber sur le capteur afin qu'il soit réactif aux faibles niveaux de lumière. Les appareils photo ont des diaphragmes : plus la valeur du diaphragme est faible, plus l'ouverture est large, ce qui permet de prendre des photos dans des environnements relativement sombres. Si l'ouverture du capteur de proximité du sol est relativement large, un gradient sur la ligne n'est pas nécessaire. Un seul capteur peut suivre un seul *edge* d'une ligne (Fig. 3.7). La figure suppose que le robot est censé suivre le bord droit de la ligne.

- Image de gauche : Si le capteur du robot se trouve au-dessus de la ligne, peu de lumière sera détectée, le robot est donc trop à gauche du bord droit qu'il devrait suivre. Le robot doit tourner à droite.
- Image du centre : Si le capteur n'est pas sur la ligne, une grande quantité de lumière sera détectée. Le robot est donc trop à droite du bord droit qu'il doit suivre. Le robot doit tourner à gauche.
- Image de droite : Si le capteur est au-dessus du bord droit de la ligne (comme il devrait l'être), la quantité de lumière détectée sera à mi-chemin entre les deux valeurs extrêmes. Le robot peut continuer à avancer.

En bas de la figure se trouve un graphique des valeurs renvoyées par les capteurs. Les lignes en pointillés représentent les seuils entre les trois états : sur la ligne, hors de la ligne, sur le bord.

Activity 3.16: Suivi de ligne sans gradient

- Écrivez l'algorithme en détail.
- Faites des expériences pour déterminer les seuils.
- Implémentez l'algorithme.
- Comparez les performances de l'algorithme avec les algorithmes de la ligne suivante en utilisant deux capteurs et un capteur avec un gradient. Quel algorithme est le plus robuste, c'est-à-dire, quel algorithme est le plus performant à des vitesses plus élevées et est capable de suivre des virages plus serrés de la ligne ?

Activity 3.17: Suivi de ligne en pratique

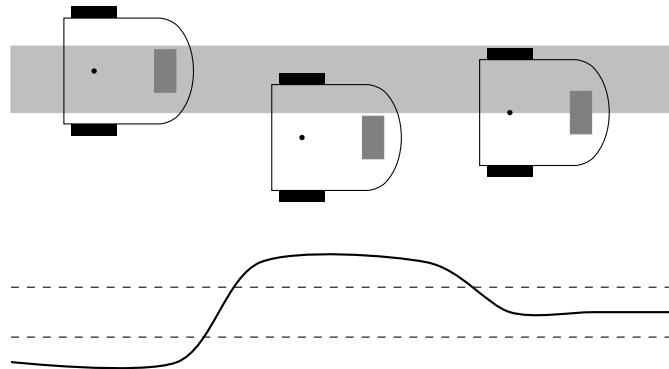


FIG. 3.7 – *Suivi de ligne avec un seul capteur et sans gradient. Above : robot se déplaçant sur la ligne, below : tracé de la valeur du capteur en fonction de la distance*

- Notre présentation du suivi de ligne est basée sur l'utilisation de capteurs qui détectent une ligne peinte ou collée au sol. Quelles autres technologies pourraient être utilisées pour représenter la ligne et la détecter ?
- Les algorithmes amènent le robot à suivre la ligne, mais pour un robot d'entrepôt, il faut un moyen d'identifier le moment où le robot a atteint le bac requis et un moyen de localiser un article spécifique dans le bac. Comment ces tâches peuvent-elles être mises en œuvre ?
- Supposons que l'entrepôt ajoute de nouveaux bacs. Quels changements doivent être apportés au robot ? Comment les algorithmes pourraient-ils être conçus pour faciliter le changement ?

3.5 Présentation des véhicules par Braitenberg

Les véhicules de Valentino Braitenberg ont été construits comme des expériences de pensée non destinées à être mises en œuvre avec des composants électroniques ou dans des logiciels. Les véhicules sont équipés de capteurs directement connectés aux moteurs, comme dans le système nerveux des êtres vivants. Certains véhicules étaient dotés d'une mémoire semblable à celle d'un cerveau.

Les figures 3.8–3.9 montrent des robots qui illustrent la présentation de Braitenberg. Ils possèdent des capteurs de lumière (les demi-cercles à l'avant des robots) qui sont directement reliés aux moteurs des roues. Plus la lumière est détectée, plus chaque roue tourne vite, comme l'indiquent les signes + sur les connexions. Si une forte source de lumière se trouve directement devant le robot, les deux capteurs

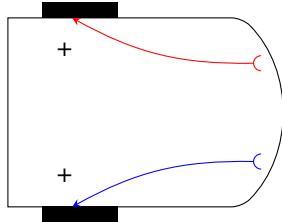


FIG. 3.8 – Véhicule lâche

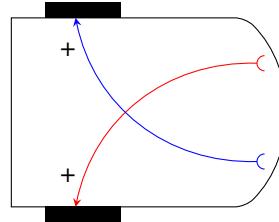


FIG. 3.9 – Véhicule agressif

renverront la même valeur et le robot avancera rapidement. Supposons maintenant que la source lumineuse se trouve sur la gauche. Pour le robot de la Fig. 3.8, la roue gauche tourne rapidement et la roue droite lentement. Il en résulte que le robot tourne brusquement à droite en s'éloignant de la source lumineuse. Braatenberg a appelé ce véhicule *coward*. Dans le cas du robot de la Fig.3.9, la roue droite tourne rapidement et la roue gauche lentement, de sorte que le robot tourne vers la source lumineuse et finit par la percuter. Ce comportement est *agressif*.

Activity 3.18: La présentation des véhicules par Braatenberg

- Mettre en place les véhicules *coward* et *aggressive*.
- Utilisez des capteurs de proximité à la place des capteurs de lumière de Braatenberg et la détection ou la non-détection d'un objet à la place de sources lumineuses plus ou moins fortes.
- Les robots des Figs. 3.10–3.11 sont les mêmes que les robots des Figs. 3.8–3.9, respectivement, sauf que les valeurs des capteurs sont négatives (les signes – sur les connexions) : plus la lumière est détectée, plus la roue tourne lentement. Supposons qu'un biais fixe soit appliqué aux moteurs afin que les roues tournent vers l'avant lorsqu'aucune source de lumière n'est détectée.
- Implémentez le robot de la Fig. 3.10. Pourquoi s'appelle-t-il *loves* ?
- Implémentez le robot de la Fig. 3.11. Pourquoi s'appelle-t-il *explorer* ?

3.6 Summary

A robot exhibits reactive behavior when its actions depend only upon the current values returned by its sensors. This chapter presented two families of reactive behavior. Braatenberg vehicles implement reactive behavior by changing the setting of the motors in response to the events of proximity sensors detecting or not detecting an object at some position relative to the robot. The vehicles demonstrate that complex behavior can result from relatively simple reactive algorithms.

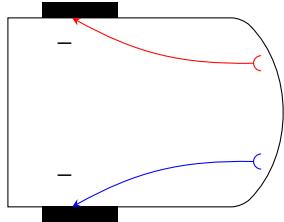


FIG. 3.10 – Véhicule d'amour

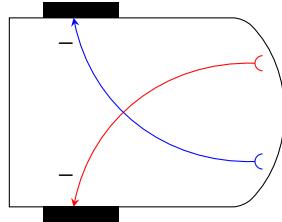


FIG. 3.11 – Véhicule d'exploration

Line following is a fundamental task in robotics. Because of uncertainties of the robot's motion and its environment, robots use landmarks such as lines to ensure that they move to their intended destination. Line following is a reactive behavior because the robot modifies its behavior in response to values returned by the ground sensors. We have given three configurations for the robot and the line, and developed algorithms for each case. The performance of the algorithms depends on the sensor thresholds and the motor speeds, which must be determined by experimentation in order to ensure that the robot moves rapidly while remaining robust to changes in the environment.

3.7 Further reading

Braitenberg's book [7] is interesting because he writes from the perspective of a neuroscientist. The book describes vehicles using imagined technology, but they are nevertheless thought-provoking. The Braitenberg vehicles described here are adapted from the hardware implementations described in [22]. An implementation of the Braitenberg vehicles in Scratch by the first author can be found at : <https://scratch.mit.edu/studios/1452106>.

Chapitre 4

Machines à états finis

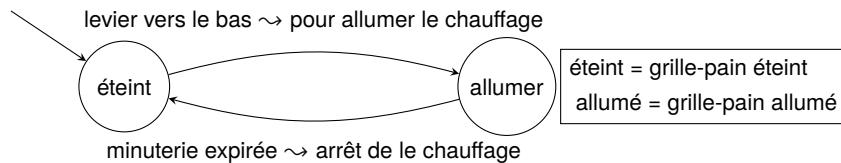
Les véhicules de Braitenberg et les algorithmes de suivi de ligne (Chap. 3) démontrent un comportement réactif, où l'action du robot dépend des valeurs *current* renvoyées par les capteurs du robot, et non d'événements survenus précédemment. Dans la section 4.1, nous présentons le concept d'état et les machines à états finis (FSM). Les sections 4.2–4.3 montrent comment certains véhicules de Braitenberg peuvent être implémentés avec des FSMs. La section 4.4 traite de l'implémentation des FSMs en utilisant des variables d'état.

4.1 Machines à états

Le concept d'état est très familier. Prenons l'exemple d'un grille-pain : initialement, le grille-pain est dans l'état *off*; lorsque vous poussez la manette vers le bas, une transition s'effectue vers l'état *on* et les éléments chauffants sont allumés; enfin, lorsque la minuterie expirée, une transition s'effectue à nouveau vers l'état *off* et les éléments chauffants sont éteints.

Un *machine à états finis (FSM)*¹ consiste en un ensemble de *états*. Il est constitué d'un ensemble de *états* s_i et d'un ensemble de *transitions* entre les paires d'états s_i, s_j . Une transition est étiquetée *condition/action* : une condition qui provoque la transition et une action qui est exécutée lorsque la transition est effectuée.

Les FSM peuvent être affichés dans des *diagrammes d'états* :



Un état est représenté par un cercle étiqueté avec le nom de l'état. Les états sont désignés par des noms courts pour gagner de la place et leurs noms complets sont indiqués dans un encadré à côté du diagramme d'état. La flèche entrante indique l'état initial. Une transition est représentée par une flèche allant de l'état *source* à l'état *cible*. La flèche est étiquetée avec la condition et l'action de la transition. L'action n'est pas continue ; par exemple, l'action tourner à gauche signifie régler les moteurs pour que le robot tourne à gauche, mais la transition vers l'état suivant se fait sans attendre que le robot atteigne une position spécifique.

1. Les machines à états finis sont également appelées automates finis.

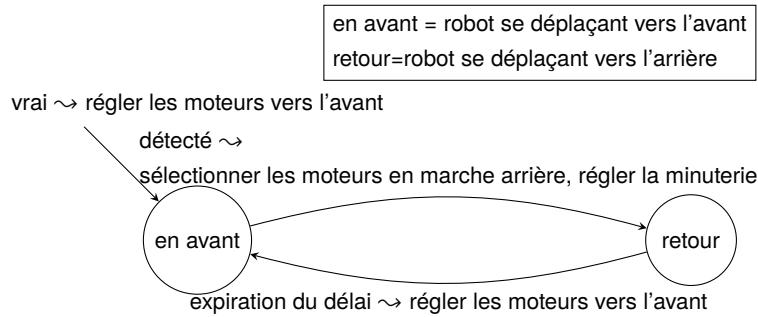


FIG. 4.1 – FSM pour le véhicule persistant de Braitenberg

4.2 Comportement réactif avec état

Voici la spécification d'un véhicule de Braitenberg dont le comportement est non réactif :

Spécification (Persistant) : Le robot se déplace vers l'avant jusqu'à ce qu'il détecte un objet. Il recule ensuite pendant une seconde et fait marche arrière pour avancer à nouveau.

La figure 4.1 montre le diagramme d'état de ce comportement.

Initialement, lorsque le système est allumé, les moteurs sont réglés pour avancer. (Dans l'état fwd, si un objet est détecté, la transition vers l'état back est effectuée, le robot recule et le minuteur est activé. Au bout d'une seconde, la minuterie expirera ; la transition vers l'état fwd s'effectuera et le robot avancera. Si un objet est détecté lorsque le robot se trouve dans l'état back, une action *no* est exécutée, car aucune transition n'est étiquetée avec cette condition. Cela montre que le comportement n'est pas réactif, c'est-à-dire qu'il dépend de l'état actuel du robot ainsi que de l'événement qui se produit.

Activity 4.1: Cohérent

— Dessinez le diagramme d'état du véhicule de Braitenberg consistant.

Spécification (Cohérent) : Le robot passe par quatre états, changeant d'état une fois par seconde : avancer, tourner à gauche, tourner à droite, reculer.

4.3 Recherche et approche

Cette section présente un exemple plus complexe d'un comportement robotique qui utilise des états.

Spécification (Recherche et approche) : Le robot effectue une recherche à gauche et à droite ($\pm 45^\circ$). Lorsqu'il détecte un objet, le robot s'en approche et s'arrête lorsqu'il est proche de l'objet.

Il existe deux configurations des capteurs d'un robot qui lui permettent de chercher à gauche et à droite, comme le montrent les figures 2.10, 2.11. Si les capteurs sont fixés au corps du robot, le robot lui-même doit tourner à gauche et à droite ; dans le cas contraire, si le capteur est monté de manière à pouvoir tourner, le robot peut rester immobile et le capteur est mis en rotation. Nous supposons que le robot a des capteurs fixes.

La figure 4.2 présente le diagramme d'état de la recherche et de l'approche. Le robot effectue initialement une recherche vers la gauche. Deux nouveaux concepts sont illustrés dans le diagramme. Il existe un *état final* étiqueté fondé et représenté par un double cercle dans le diagramme. Un automate fini est fini dans le sens où il contient un nombre fini d'états et de transitions. Cependant, son comportement peut être fini ou infini. Le FSM de la Fig. 4.2 présente un comportement fini car le robot s'arrête lorsqu'il a trouvé un objet et s'en est approché. Ce FSM a également un comportement infini : si un objet n'est jamais trouvé, le robot continue indéfiniment à chercher à gauche et à droite.

Le véhicule Persistant de Braitenberg (Fig. 4.1) a un comportement infini car il continue à se déplacer sans s'arrêter. Un grille-pain a également un comportement infini car vous pouvez continuer à griller des tranches de pain pour toujours (jusqu'à ce que vous débranchez le grille-pain ou que vous manquiez de pain).

Le deuxième nouveau concept est *nondéterminisme*. Les états gauche et droite ont chacun *deux* transitions sortantes, une pour atteindre le bord du secteur recherché et une pour détecter un objet. Le sens du non-déterminisme est que n'importe laquelle des transitions sortantes peut être prise. Il y a trois possibilités :

- L'objet est détecté mais la recherche n'est pas sur un bord du secteur ; dans ce cas, la transition vers appr est prise.
- La recherche se situe sur un bord du secteur mais aucun objet n'a été détecté ; dans ce cas, la transition de gauche à droite ou de droite à gauche est effectuée.
- La recherche se trouve à un bord du secteur au moment précis où un objet est détecté ; dans ce cas, une transition arbitraire est effectuée. C'est-à-dire que le robot peut s'approcher de l'objet ou changer la direction de la recherche.

Le comportement non déterministe du troisième cas peut faire en sorte que le robot n'approche pas l'objet lorsqu'il est détecté pour la première fois, si cet événement se produit en même temps que l'événement d'atteinte de $\pm 45^\circ$. Toutefois, après une

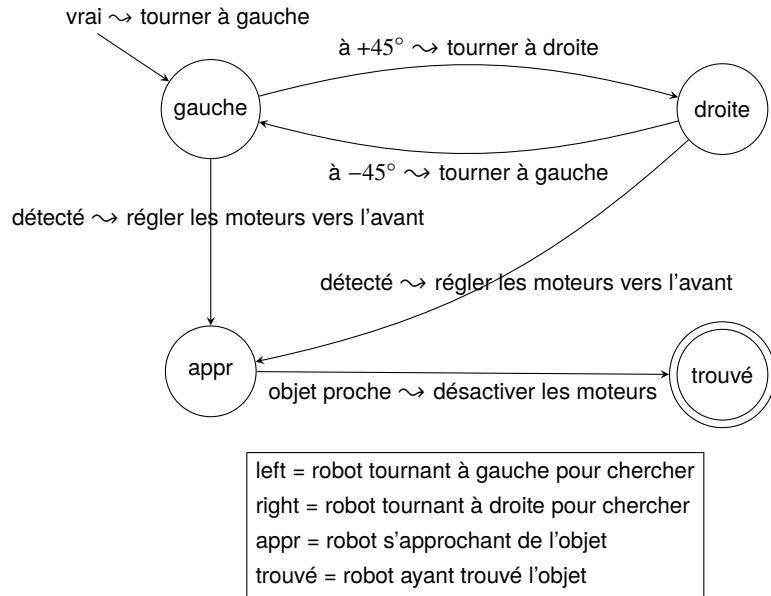


FIG. 4.2 – Diagramme d'état pour la recherche et l'approche

courte période, les conditions seront vérifiées à nouveau et il est probable qu'un seul des événements se produira.

4.4 Mise en œuvre des automates à états finis

Pour implémenter des comportements avec des états, il faut utiliser des variables. Le véhicule Persistent (Sect. 4.2) a besoin d'une minuterie pour provoquer un événement après l'expiration d'une période de temps. Comme expliqué dans la section 1.6.4, une minuterie est une variable qui est définie sur la période de temps souhaitée. La variable est décrémentée par le système d'exploitation et lorsqu'elle atteint zéro, un événement se produit.

L'algorithme 4.1 décrit comment le FSM de la Fig. 4.1 est implémenté. La variable current contient l'état actuel du robot ; à la fin du traitement d'un gestionnaire d'événement, la valeur de la variable est fixée à l'état cible de la transition. Les valeurs de current sont nommées fwd et back pour plus de clarté, bien que dans un ordinateur elles seraient représentées par des valeurs numériques.

Activity 4.2: Persistent

- Implémente le comportement Persistent.

Algorithm 4.1: Persistant	
integer timer	// In milliseconds
states current ← fwd	
1: left-motor-power ← 100	
2: right-motor-power ← 100	
3: loop	
4: when current = fwd and object detected in front	
5: left-motor-power ← -100	
6: right-motor-power ← -100	
7: timer ← 1000	
8: current ← back	
9:	
10: when current = back and timer = 0	
11: left-motor-power ← 100	
12: right-motor-power ← 100	
13: current ← fwd	

Activity 4.3: Paranoïde (sens alterné)

- Dessinez le diagramme d'état du véhicule Paranoïde (direction alternée) de Braitenberg :

Spécification (Paranoïaque (sens alterné)) :

- Lorsqu'un objet est détecté devant le robot, celui-ci se déplace vers l'avant.
- Lorsqu'un objet est détecté à droite du robot, ce dernier tourne à droite.
- Lorsqu'un objet est détecté à la gauche du robot, ce dernier tourne à gauche.
- Si le robot tourne (même s'il ne détecte plus d'objet), il alterne la direction de son virage toutes les secondes.
- Lorsqu'aucun objet n'est détecté et que le robot ne tourne pas, il s'arrête.
- Mettez en œuvre cette spécification. En plus d'une variable qui stocke l'état actuel du robot, utilisez une variable avec les valeurs gauche et droite pour stocker la direction dans laquelle le robot tourne. Définissez une minuterie avec une période d'une seconde. Dans le gestionnaire

Algorithm 4.2: Recherche et approche	
État actuel de l'← gauche	
states current ← left	
1:	left-motor-power ← 50 // Turn left
2:	right-motor-power ← 150
3:	loop
4:	when object detected
5:	if current = left
6:	left-motor-power ← 100 // Go forwards
7:	right-motor-power ← 100
8:	current ← appr
9:	else if current = right
10:	...
11:	when at +45°
12:	if current = left
13:	left-motor-power ← 150 // Turn right
14:	right-motor-power ← 50
15:	current ← right
16:	when at -45°
17:	...
18:	when object is very near
19:	if current = appr
20:	...

d'événements de la minuterie, changez la valeur de la variable de direction par la valeur opposée et réinitialisez la minuterie.

L'algorithme 4.2 est un aperçu de l'implémentation du diagramme d'état de la Fig. 4.2.

Activity 4.4: Recherche et approche

- Remplissez les lignes manquantes dans l'Algorithme 4.2.
- Implémentez l'Algorithme 4.2.

4.5 Summary

La plupart des algorithmes de robotique exigent que le robot conserve une représentation interne de son état actuel. Les conditions que le robot utilise pour décider quand changer d'état et les actions prises lors du passage d'un état à un autre sont décrites par des automates finis. Les variables d'état sont utilisées pour implémenter les machines d'état dans les programmes.

4.6 Lectures complémentaires

Le manuel classique sur les automates a été publié en 1979 par John Hopcroft et Jeffrey D. Ullman ; sa dernière édition est [23]. Pour une explication détaillée du *arbitrary* choix parmi les alternatives dans le nondéterminisme (Sect. 4.3), voir [5, Sect. 2.4].

Chapitre 5

Mouvement robotique et odométrie

Les algorithmes robotiques des chapitres précédents réagissent aux données provenant de leurs capteurs en modifiant la vitesse et la direction de leur mouvement, mais ces modifications ne sont pas quantitatives. Nous n'avons pas demandé aux robots de se déplacer deux fois plus vite ou de tourner 90° vers la droite. Dans le monde réel, les robots doivent se déplacer vers des endroits spécifiques et peuvent avoir des contraintes techniques sur la vitesse ou la lenteur de leur déplacement ou de leur rotation. Ce chapitre présente les mathématiques du mouvement robotique.

Les sections 5.1–5.2 passent en revue les concepts de distance, de temps, de vitesse et d'accélération qui devraient être familiers lors de l'introduction à la physique. La physique du mouvement est généralement enseignée à l'aide du calcul, mais un ordinateur ne peut pas traiter des fonctions continues ; il faut donc utiliser des approximations discrètes, comme décrit dans la section 5.3.

Les sections 5.4–5.6 présentent la *odometry*, l'algorithme fondamental pour le calcul du mouvement robotique. Une approximation de la position d'un robot peut être obtenue en calculant de manière répétée la distance parcourue et le changement de direction à partir de la vitesse des roues dans un court laps de temps. Malheureusement, l'odométrie est sujette à de graves erreurs, comme le montre la Sec. 5.7. Il est important de comprendre que les erreurs de direction sont beaucoup plus importantes que les erreurs de distance.

Dans l'implémentation la plus simple, la vitesse des roues d'un robot est supposée être proportionnelle à la puissance appliquée par les moteurs. La section 5.8 montre comment la précision de l'odométrie peut être améliorée en utilisant des *codeurs de roue*, qui mesurent le nombre réel de tours des roues.

La section `refs imu` présente une vue d'ensemble de la *navigation inertie*, qui est une forme sophistiquée d'odométrie basée sur la mesure de l'accélération linéaire et angulaire, puis sur l'intégration pour obtenir la vitesse et la position. Les capteurs de la navigation inertie (accéléromètres et gyroscopes) étaient autrefois très coûteux, ce qui limitait son application aux avions et aux fusées, mais une nouvelle technologie appelée *systèmes microélectromécaniques* a permis de construire des robots dotés de la navigation inertie.

Les voitures ne peuvent pas se déplacer de haut en bas, contrairement aux hélicoptères et aux sous-marins qui ont une plus grande liberté de mouvement. Cela s'exprime par le concept de *degrés de liberté (DOF)* qui fait l'objet de la section 5.10. La section 5.11 traite de la relation entre les DOF et le nombre de *actuateurs* (moteurs) dans un système robotique.

Le nombre de DOF d'un système ne signifie pas qu'un système tel qu'un véhicule peut se déplacer librement dans toutes ces directions. Une voiture peut se déplacer en tout point du plan et s'orienter dans n'importe quelle direction, mais elle ne peut pas se déplacer latéralement, ce qui nécessite une manœuvre difficile lors d'un stationnement parallèle. Cela est dû à la différence entre les DOF et les *degrés de mobilité (DOM)*, un sujet exploré dans la section 5.12, ainsi que le concept de *mouvement holonomique* qui relie les DOF et les DOM.

5.1 Distance, vitesse et temps

Supposons qu'un robot se déplace avec une *vitesse* constante de 10 cm/s pendant une période de *temps* de 5 s.¹ La *distance* qu'il parcourt est de 50 cm. En général, si un robot se déplace à une vitesse constante de v pendant une période de temps de t , la distance qu'il parcourt est de $s = vt$. Lorsque les moteurs sont alimentés, les roues tournent, ce qui entraîne le déplacement du robot à une certaine vitesse. Cependant, nous ne pouvons pas spécifier qu'une certaine puissance entraîne une certaine vitesse :

- Deux composants électriques ou mécaniques ne sont jamais exactement identiques. Un moteur est composé d'aimants et de câbles électriques dont l'interaction entraîne la rotation d'un arbre mécanique. De petites différences dans les propriétés de l'aimant et du fil, ainsi que de petites différences dans la taille et le poids de l'arbre, peuvent faire que les arbres de deux moteurs tournent à des vitesses légèrement différentes pour la même quantité de puissance.
- L'environnement affecte la vitesse d'un robot. Une friction trop faible (glace) ou trop forte (boue) peut ralentir le déplacement d'un robot par rapport à une surface pavée sèche.
- Les forces externes peuvent affecter la vitesse d'un robot. Il a besoin de plus de puissance pour maintenir une vitesse spécifique lorsqu'il se déplace en montée et de moins de puissance lorsqu'il se déplace en descente, car la force de gravité diminue et augmente la vitesse. Faire du vélo à une vitesse constante face au vent demande plus d'effort que de rouler avec le vent, et un vent latéral rend la relation entre la puissance et la vitesse encore plus compliquée.

Puisque $s = vt$, il suffit de mesurer deux de ces quantités pour calculer la troisième. Si nous mesurons la distance et le temps, nous pouvons calculer la vitesse comme $v = s/t$. Les distances relativement courtes (jusqu'à plusieurs mètres) peuvent être mesurées avec précision (à 1 cm près) à l'aide d'une règle ou d'un mètre ruban. L'application chronomètre d'un smartphone peut mesurer le temps

1. La vitesse est une *vitesse* dans une *direction*. Un robot peut se déplacer de 10 cm/s en avant ou en arrière ; dans les deux cas, la vitesse est la même mais la vitesse est différente.

avec précision (centièmes de seconde).

Activity 5.1: Vélocité sur une distance fixe

- Écrivez un programme qui règle votre robot sur une puissance avant constante.
- Marquez deux lignes distantes de 1 m sur le sol. Utilisez un chronomètre pour mesurer le temps que met le robot à se déplacer entre les lignes. Calculez la vitesse du robot. Exécutez le programme dix fois et enregistrez les vitesses. Les vitesses varient-elles ?
- Placez le robot sur le sol et faites-le fonctionner pendant 5 s. Mesurez la distance qu'il parcourt. Calculez sa vitesse. Exécutez le programme dix fois et enregistrez les vitesses. Les vitesses varient-elles ?
- Quelle méthode donne des résultats plus précis ?
- Répétez cette expérience sur différentes surfaces et discutez des résultats.

L'activité 5.1 montre que pour un réglage de puissance constant, la vitesse d'un robot peut varier de manière significative. Pour naviguer avec précision dans un environnement, un robot doit détecter les objets de son environnement, tels que les murs, les marques au sol et les objets.

5.2 Accélération en tant que changement de vitesse

Activity 5.1 a spécifié des paramètres de puissance constante et la vitesse du robot sera donc (plus ou moins) constante. Que se passe-t-il lorsque la vitesse varie ?

Activity 5.2: Changement de vitesse

- Exécutez le premier programme de Activity 5.1 en faisant varier la distance entre les marques : 0, 0,25 m, 0, 0,5 m, 1 m, 1, 1,5 m, 2 m. Pour chaque distance, exécutez le programme plusieurs fois et prenez la moyenne des vitesses calculées. Les vitesses sont-elles les mêmes pour chaque distance ?
- Pour améliorer la précision de la mesure, placez des marques sur le sol à ces distances et utilisez le minuteur du robot pour enregistrer les moments où les marques sont détectées.

Dans l'activité 5.2, vous constaterez que pour les distances les plus longues, les vitesses seront proches les unes des autres, mais que pour les distances plus courtes, les vitesses seront considérablement différentes. La raison en est que la formule $v = s/t$ suppose que la vitesse est constante sur toute la distance. En réalité, un

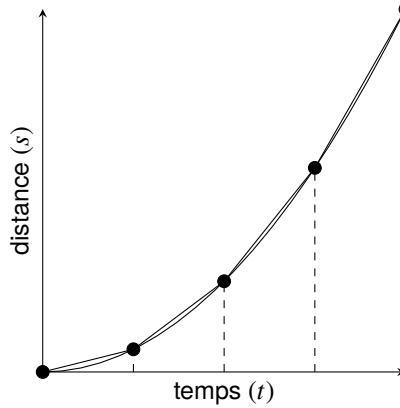


FIG. 5.1 – Un robot en accélération : la distance augmente au carré du temps

véhicule doit changer sa vitesse pour passer de l'arrêt à une vitesse constante. De même, un véhicule doit décélérer pour s'arrêter.

Pour obtenir une image fidèle du mouvement d'un robot, nous devons diviser son mouvement en petits segments s_1, s_2, \dots :

$$\begin{array}{ccccccc} & s_1 & & s_2 & & s_3 & & s_4 & & s_5 \\ \hline x_0 & & x_1 & & x_2 & & x_3 & & x_4 & & x_5 \end{array}$$

et mesurer la distance et le temps pour chaque segment individuellement. Ensuite, nous pouvons calculer les vitesses pour chaque segment. En symboles, si nous désignons la longueur du segment s_i par $\Delta s_i = x_{i+1} - x_i$ et le temps que met le robot à traverser le segment s_i par $\Delta t_i = t_{i+1} - t_i$, alors v_i , la vitesse dans le segment s_i est donnée par :

$$v_i = \frac{\Delta s_i}{\Delta t_i}.$$

La figure 5.1 est un graphique de la distance en fonction du temps pour un robot en accélération. L'axe du temps a été divisé en segments et les pentes $\frac{\Delta s_i}{\Delta t_i}$ montrent la vitesse moyenne dans chaque segment qui augmente avec le temps.

Accélération est définie comme la variation de la vitesse sur une période de temps :

$$a_i = \frac{\Delta v_i}{\Delta t_i}.$$

Lorsque la puissance du robot est réglée sur une valeur fixe, la force appliquée au robot est constante et nous nous attendons à ce que l'accélération reste constante, ce qui augmente la vitesse. Cependant, à un certain point, l'accélération est réduite à zéro, ce qui signifie que la vitesse n'augmente plus, car la puissance appliquée aux

roues est juste suffisante pour surmonter le frottement de la route et la résistance du vent.

Voyons ce qui se passe si l'on augmente le réglage de la puissance avec le temps.

Activity 5.3: Acceleration

- Écrivez un programme qui fait accélérer le robot en augmentant périodiquement la puissance. Par exemple, démarrez le robot à une puissance de 20 et augmentez-la à 40 après 1 s, puis à 60 après 2 s, à 80 après 3 s, et enfin à 100 après 4 s.
- Placez le robot sur la piste et exécutez le programme.
- Enregistrez les distances entre chaque changement de réglage de la puissance. Calculez et tracez les vitesses dans chacun de ces segments.

5.3 Des segments au mouvement continu

Au fur et à mesure que la taille des segments diminue, nous obtenons la vitesse instantanée du robot en un seul point du temps, exprimée sous forme de dérivée :

$$v(t) = \frac{ds(t)}{dt} .$$

De la même manière, l'accélération instantanée du robot est définie comme suit :

$$a(t) = \frac{dv(t)}{dt} .$$

Pour une accélération constante, la vitesse peut être obtenue en intégrant la dérivée :

$$v(t) = \int a dt = a \int dt = at ,$$

et ensuite la distance peut être obtenue en intégrant à nouveau :

$$s(t) = \int v(t) dt = \int at dt = \frac{at^2}{2} .$$

Exemple Une voiture moyenne accélère de 0 à 100 km/h en environ 10 s. Tout d'abord, nous convertissons les unités de km/h en m/s :

$$v_{max} = 100 \text{ km/h} = \frac{100 \cdot 1000}{60 \cdot 60} \text{ m/s} = 27,8 \text{ m/s} .$$

En supposant une accélération constante, $v_{max} = 27,8 = at = 10a$, l'accélération est donc de $2,78 \text{ m/s}^2$ (lire, 2,78 mètres par seconde par seconde, c'est-à-dire que chaque seconde la vitesse augmente de 2,78 mètres par seconde). La distance parcourue par la voiture en 10 s est :

$$s(10) = \frac{at^2}{2} = \frac{2,78 \cdot 10^2}{2} = 139 \text{ m} .$$

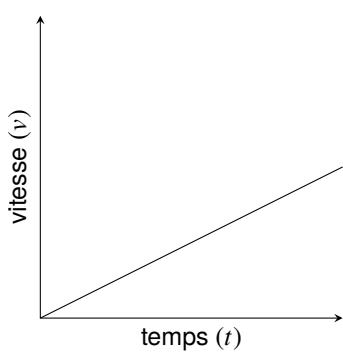


FIG. 5.2 – Vélocité pour une accélération constante

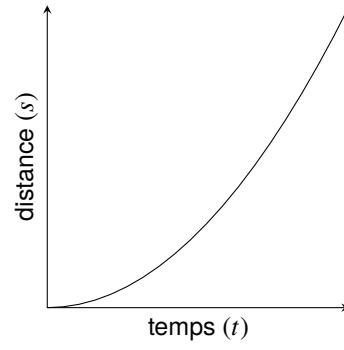


FIG. 5.3 – Distance pour une accélération constante

Activity 5.4: Computer la distance en accélérant

- Pour différents véhicules (voitures de course, motos), recherchez le temps nécessaire pour accélérer de 0 à 100 km/h. Calculez la distance parcourue.
- Supposez que l'accélération d'un véhicule augmente linéairement, c'est-à-dire que $a = kt$ pour une constante k . Que valent $v(t)$ et $s(t)$.
- Pour plusieurs valeurs de k et t , calculez les vitesses et distances finales.

Activity 5.5: Mesure du mouvement à accélération constante

- Écrire un programme qui applique le réglage de la puissance maximale à un robot.
- Placez le robot sur une surface et exécutez le programme.
- Lorsque le robot semble avoir atteint sa vitesse maximale, enregistrez le temps écoulé depuis le début de l'exécution.
- Comparez la distance mesurée à $s = at^2/2$ (Fig. 5.3).
- Courez à nouveau et mesurez les distances à intervalles de temps fixes. Calculez les vitesses à partir des distances divisées par le temps et comparez-les à $v = at$ (Fig. 5.2).
- Dans certains robots, vous pouvez définir une vitesse cible et lire la vitesse réelle. Si votre robot peut le faire, comparez les vitesses mesurées avec les vitesses calculées.

5.4 Navigation par odométrie

Supposons que vous soyez dans une voiture et que votre système de navigation émette l'instruction suivante : "Dans 700 mètres, tournez à droite". Votre tâche est maintenant très simple : Observez l'odomètre de votre voiture qui mesure la distance que vous avez parcourue. Lorsque sa valeur approche les 700 mètres au-delà de sa lecture initiale, cherchez une rue à droite. L'odomètre d'une voiture mesure la vitesse et le temps, et multiplie ces deux valeurs pour calculer la distance parcourue.

L'odométrie - la mesure de la distance - est une méthode fondamentale utilisée par les robots pour la navigation. La mesure du temps est facile grâce à l'horloge interne de l'ordinateur embarqué. La mesure de la vitesse est plus difficile : dans certains robots éducatifs, des encodeurs de roue sont utilisés pour compter les rotations des roues (Sec. 5.8), tandis que dans d'autres, la vitesse est estimée à partir des propriétés des moteurs. À partir de la distance parcourue $s = vt$, la nouvelle position du robot peut être calculée. En une dimension, le calcul est trivial, mais il devient un peu plus complexe lorsque le mouvement implique des virages. Cette section présente le calcul de la distance par odométrie, d'abord pour un robot se déplaçant linéairement, puis pour un robot effectuant un virage.

La section 5.7 montre que les erreurs de cap sont plus graves que les erreurs de distance.

L'un des inconvénients de l'odométrie (avec ou sans encodeurs de roue) est que les mesures sont indirectes : elles relient la puissance des moteurs ou le mouvement des roues aux changements de position du robot. Cela peut être source d'erreurs car la relation entre la vitesse du moteur et la rotation des roues peut être très non linéaire et varier dans le temps. En outre, les roues peuvent glisser et déraper, ce qui peut entraîner des erreurs dans la relation entre le mouvement des roues et le mouvement du robot. De meilleures estimations de la position peuvent être obtenues en utilisant un système de navigation inertiel, qui mesure directement l'accélération et la vitesse angulaire pouvant être utilisées pour déterminer la position du robot (Sec. 5.9).

L'odométrie est une forme de *localisation* : le robot doit déterminer sa position dans l'environnement. En odométrie, nous déterminons la position en mesurant le changement par rapport à la position initiale connue du robot, tandis que la localisation (Chap. 8) fait référence à la détermination de la position d'un robot par rapport aux positions connues d'autres objets tels que des points de repère ou des balises.

5.5 Odométrie linéaire

Avant d'étudier les mathématiques de l'odométrie, vous devriez essayer l'activité suivante :

Activity 5.6: Distance à partir de la vitesse et du temps

- Faites fonctionner le robot à une puissance constante pendant une période donnée et mesurez la distance parcourue.
- Répétez la mesure plusieurs fois. La distance est-elle constante ? Si non, de combien varie-t-elle en pourcentage de la distance ?
- Répétez la mesure plusieurs fois pour différents réglages de puissance. La distance mesurée est-elle linéaire en fonction du réglage de la puissance ? La variation de la distance mesurée sur plusieurs parcours dépend-elle du réglage de la puissance ?
- Répétez la mesure pour un réglage de puissance fixe mais pendant des périodes de temps différentes et analysez les résultats.

Lorsqu'une relation entre la puissance du moteur et la vitesse v a été déterminée, le robot peut calculer la distance parcourue par $s = vt$. S'il commence à la position $(0, 0)$ et se déplace en ligne droite le long de l'axe x , alors après t secondes, sa nouvelle position est $(vt, 0)$.

Cette activité doit démontrer qu'il est possible de mesurer une distance par odométrie avec une précision et une exactitude raisonnables. Une voiture autonome peut utiliser l'odométrie pour déterminer sa position afin de ne pas avoir à analyser continuellement les données de ses capteurs pour vérifier si la rue requise a été atteinte. Compte tenu des incertitudes du mouvement et de la route, la voiture ne doit pas dépendre uniquement de l'odométrie pour décider quand tourner, mais l'erreur ne sera pas grande et les données des capteurs peuvent être analysées pour détecter le virage lorsque l'odométrie indique que la voiture est à proximité de l'intersection.

L'activité 5.6 vous demande de mesurer la distance parcourue dans une dimension. Trois informations doivent être calculées si le mouvement est en deux dimensions : la *position* (x, y) du robot par rapport à une origine fixe et son *heading* θ , la direction dans laquelle le robot pointe (Fig. 5.4). Le triplet (x, y, θ) est appelé la *pose* du robot. Si le robot commence à l'origine $(0, 0)$ et se déplace en ligne droite selon un angle θ avec une vitesse v pendant un temps t , la distance parcourue est $s = vt$. Sa nouvelle position (x, y) est :

$$\begin{aligned} x &= vt \cos \theta \\ y &= vt \sin \theta . \end{aligned}$$

5.6 Odométrie avec tours

Supposons que le robot tourne légèrement à gauche car la roue droite se déplace plus rapidement que la roue gauche (Fig. 5.5). Sur la figure, le robot est orienté vers

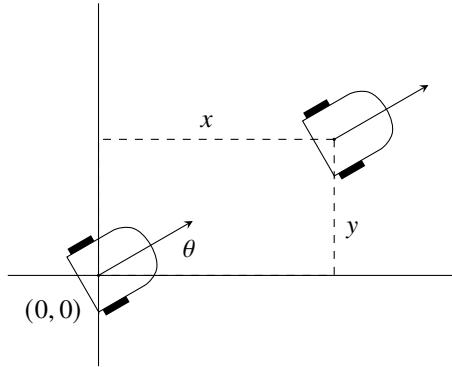


FIG. 5.4 – Position et en-tête

le haut de la page ; le point bleu représente la roue gauche, le point rouge la roue droite et le point noir le centre du robot qui se trouve à mi-chemin entre les roues. Le *baseline* b est la distance entre les roues, et d_l, d_r, d_c représentent les distances parcourues par les deux roues et le centre lorsque le robot tourne. Nous voulons calculer la nouvelle position et le nouveau cap du robot.

Nous pouvons mesurer d_l et d_r , les distances parcourues par les deux roues en utilisant la méthode décrite dans Activity 5.6 : relier la puissance du moteur à la vitesse de rotation, puis multiplier par le temps. On peut aussi utiliser le nombre de rotations comptées par les encodeurs des roues. Si le rayon d'une roue est de R et que les vitesses de rotation des roues gauche et droite sont respectivement de ω_l, ω_r tours par seconde, alors après t secondes la roue s'est déplacée :

$$d_i = 2\pi R \omega_i t, \quad i = l, r. \quad (5.1)$$

La tâche consiste à déterminer la nouvelle pose du robot après le déplacement des roues sur ces distances.

La figure 5.5 montre le robot initialement à la pose (x, y, ϕ) , où le robot est orienté vers le nord ($\phi = \pi/2$). Après une rotation de θ radians, quelle est la nouvelle position (x', y', ϕ') ? Il est clair que le cap du robot est maintenant $\phi' = \phi + \theta$, mais nous devons également calculer x', y' .

La longueur d'un arc d'angle θ radians est donnée par sa fraction de la circonférence du cercle : $2\pi r (\theta/2\pi) = \theta r$. Pour les petits angles, les distances d_l, d_c, d_r sont approximativement égales à la longueur des arcs correspondants, on a donc :

$$\theta = d_l/r_l = d_c/r_c = d_r/r_r, \quad (5.2)$$

où r_l, r_r, r_c sont les distances par rapport à P , l'origine du tour.

Les distances d_l et d_r sont obtenues à partir des rotations des roues (Eq. 5.1) et la ligne de base b est une mesure physique fixe du robot. À partir de l'équation 5.2,

$$(x, y, \phi)$$

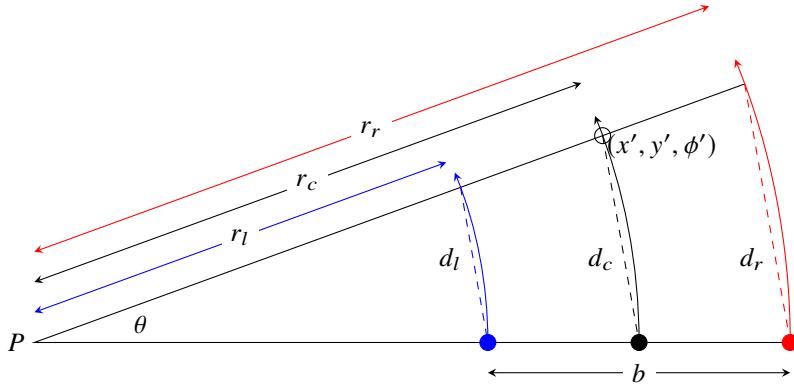


FIG. 5.5 – Géométrie d'un virage à gauche par un robot à deux roues

l'angle θ peut être calculé :

$$\begin{aligned} \theta r_r &= d_r \\ \theta r_l &= d_l \\ \theta r_r - \theta r_l &= d_r - d_l \\ \theta &= (d_r - d_l)/(r_r - r_l) \\ \theta &= (d_r - d_l)/b . \end{aligned}$$

Le centre est à mi-chemin entre les roues $r_c = (r_l + r_r)/2$, donc encore une fois par l'équation suivante :

$$\begin{aligned} d_c &= \theta r_c \\ &= \theta \left(\frac{r_l + r_r}{2} \right) \\ &= \frac{\theta}{2} \left(\frac{d_l}{\theta} + \frac{d_r}{\theta} \right) \\ &= \frac{d_l + d_r}{2} . \end{aligned}$$

Si la distance parcourue est faible, la droite notée d_c est approximativement perpendiculaire au rayon passant par la position finale du robot. Par des triangles

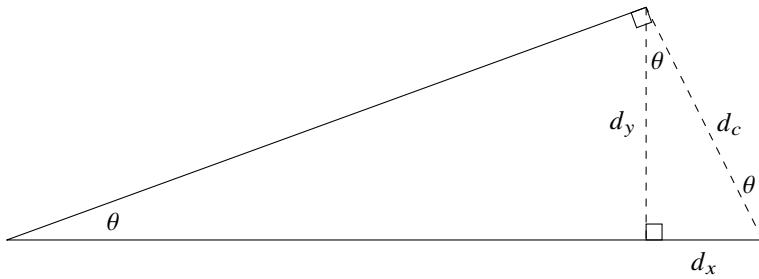


FIG. 5.6 – Changement d'intitulé

similaires, on voit que θ est la variation du cap du robot (Fig. 5.6). Par trigonométrie :²

$$dx = -d_c \sin \theta$$

$$dy = d_c \cos \theta,$$

donc la pose du robot après le virage est :

$$(x', y', \phi') = (-d_c \sin \theta, d_c \cos \theta, \phi + \theta).$$

Les formules montrent comment calculer les changements dx , dy et θ lorsque le robot se déplace sur une courte distance. Pour calculer l'odométrie sur de plus longues distances, ce calcul doit être effectué fréquemment. Deux raisons expliquent pourquoi les intervalles entre les calculs doivent être courts : (a) l'hypothèse d'une vitesse constante ne vaut que pour les courtes distances et (b) le calcul trigonométrique est simplifié en supposant que la distance parcourue est courte.

Activity 5.7: Odométrie en deux dimensions

- Write a program that causes the robot to make a gentle left turn for a specific period of time.
- Calculez la pose $(-d_c \sin \theta, d_c \cos \theta, \theta)$ et comparez le résultat avec les valeurs mesurées à l'aide d'une règle et d'un rapporteur. Exécutez le programme plusieurs fois et voyez si les mesures sont cohérentes.
- Exécutez le programme pendant différentes périodes de temps. Comment cela affecte-t-il l'exactitude et la précision du calcul de l'odométrie ?

2. Vous vous attendiez probablement à \cos pour dx et \sin pour dy . Ce serait le cas si le robot était orienté selon l'axe x . Cependant, la position initiale est $\phi = \pi/2$ et nous avons $\sin(\theta + \pi/2) = \cos \theta$ et $\cos(\theta + \pi/2) = -\sin \theta$.

5.7 Erreurs en odométrie

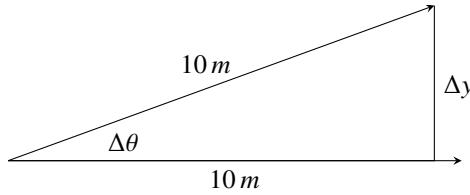
Nous avons déjà noté que l'odométrie n'est pas précise car les mesures incohérentes et les irrégularités de la surface peuvent provoquer des erreurs. Dans cette section, nous montrons que même de petits changements dans la direction du mouvement du robot peuvent causer des erreurs beaucoup plus importantes que celles causées par des changements dans son mouvement linéaire.

Pour simplifier la présentation, supposons qu'un robot doive se déplacer de 10 mètres depuis l'origine d'un système de coordonnées le long de l'axe x et qu'il doive ensuite rechercher un objet spécifique dans son environnement. Quel est l'effet d'une erreur de *up to p %*? Si l'erreur porte sur la mesure de x , la distance parcourue, alors Δx , l'erreur sur x est :

$$\Delta x \leq \pm 10 \cdot \frac{p}{100} = \pm \frac{p}{10} ; \text{meter},$$

où la valeur est négative ou positive car le robot peut se déplacer jusqu'à $p\%$ avant ou après la distance prévue.

Supposons maintenant qu'il y ait une erreur de $p\%$ dans l'*direction* du robot et, pour simplifier, supposons qu'il n'y ait pas d'erreur dans la distance parcourue. La géométrie est la suivante :



Le robot avait l'intention de se déplacer de 10 m le long de l'axe x , mais au lieu de cela, il s'est déplacé légèrement vers la gauche selon un angle de $\Delta\theta$. Calculons la déviation gauche-droite Δy . Par trigonométrie, $\Delta y = 10 \sin \Delta\theta$. Une erreur de $p\%$ dans le cap est :

$$\Delta\theta = 360 \cdot \frac{p}{100} = (3.6p)^\circ,$$

donc la déviation gauche-droite est :

$$\Delta y \leq \pm 10 \sin(3.6p).$$

Les tableaux suivants comparent la différence entre une erreur linéaire de $p\%$ (à gauche) et une erreur de cap de $p\%$ (à droite) :

$p\%$	$\Delta x (m)$	$p\%$	$\Delta\theta (\circ)$	$\sin \Delta\theta$	$\Delta y (m)$
1	.1	1	3.6	.063	.63
2	.2	2	7.2	.125	1.25
5	.5	5	18.0	.309	3.09
10	1.00	10	36.0	.588	5.88

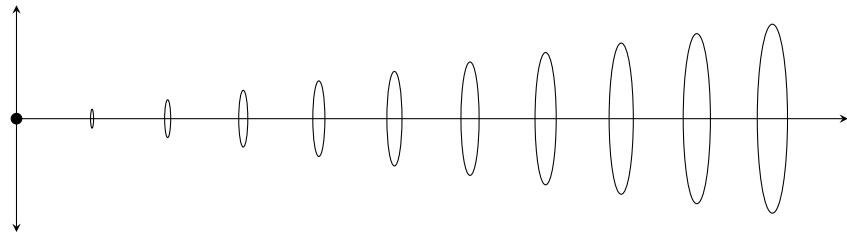


FIG. 5.7 – Erreurs d’odométrie

Pour une très petite erreur comme 2 %, l’erreur de distance après un déplacement de 10 m est de seulement .2 m, ce qui devrait placer le robot à proximité de l’objet qu’il recherche, mais une erreur de cap du même pourcentage place le robot à 1.25 m de l’objet. Pour une erreur plus importante, telle que 5 ou 10, l’erreur de distance (50 ou 100 cm) peut encore être gérée, mais l’erreur de cap place le robot à 3,09 ou 5,88 m, ce qui n’est même pas à proximité de l’objet.

L’accumulation des erreurs d’odométrie au fur et à mesure que la distance parcourue s’allonge est représentée sur la Fig. reffig.odo-errors. La position initiale du robot est désignée par le point à l’origine. En supposant une erreur d’au plus $\pm 4\%$ dans la direction linéaire et le cap, les positions possibles du robot après un déplacement de $d = 1, 2, \dots, 10$ mètres sont représentées par des ellipses. Les rayons mineurs des ellipses d’erreur résultent des erreurs linéaires :

$$.04s = 0.04, .08, \dots, .4 \text{ meters},$$

tandis que les rayons majeurs des ellipses d’erreur résultent des erreurs angulaires :

$$d \sin (.04 \cdot 360^\circ) = d \sin 14.4^\circ \approx .25, .50, \dots .25 \text{ meters}.$$

Il est clair que les erreurs angulaires sont beaucoup plus importantes que les erreurs linéaires.

Les erreurs étant inévitables, la position du robot calculée par odométrie doit être comparée périodiquement à une position absolue, qui devient la nouvelle position initiale pour les calculs ultérieurs. Les méthodes de détermination de la position absolue du robot sont présentées au Chap. 8.

Activity 5.8: Ordres d’odométrie

- Écrivez un programme pour que le robot se déplace en ligne droite sur 2 mètres. Assurez-vous que la surface est lisse afin qu’il ne dévie pas de sa trajectoire et calibrez les paramètres du moteur afin que le robot se déplace aussi droit que possible.

- Faites varier la puissance des moteurs des deux roues de façon à ce que le robot se déplace un peu plus lentement ou un peu plus rapidement qu'auparavant. Repérez sa position à intervalles fixes et voyez si l'erreur reste linéaire au cours du parcours.
- Faites varier la puissance du moteur d'une roue de sorte que le robot tourne légèrement sur le côté. Repérez sa position à intervalles fixes et voyez si les erreurs sont proportionnelles au sinus de la différence entre le cap initial et le nouveau cap.

Activity 5.9: Effet combiné des erreurs d'odométrie

- Écrivez un programme qui fait en sorte que le robot se déplace en ligne droite sur 2 mètres, puis tourne de 360°. Quelle est l'erreur de position du robot ?
- Écrivez un programme qui fait tourner le robot de 360, puis le fait se déplacer en ligne droite sur 2 de distance. Quelle est l'erreur de position du robot ? Y a-t-il une différence entre cette erreur et celle de l'expérience précédente ?
- Écrivez un programme qui permet au robot de se déplacer en ligne droite sur 2 mètres, de faire un tour de 180° et de se déplacer en ligne droite sur 2 mètres. À quelle distance se trouve-t-il de sa position de départ ?

Activity 5.10: Correction des erreurs d'odométrie

- Modifiez le programme que vous avez écrit pour l'activité 5.8 pour introduire *jitter*, une variation aléatoire de la puissance fournie au moteur. Vérifiez que la distance parcourue par le robot en un temps donné n'est pas constante, mais varie légèrement d'une course à l'autre.
- Marquez une ligne de but sur le sol et calculez le temps que le robot doit mettre pour atteindre le but.
- Lorsque le robot s'est déplacé pendant ce laps de temps, voyez s'il peut trouver le but en avançant et en reculant par petits pas jusqu'à ce qu'il détecte le but.

5.8 Codeurs de roue

L'odométrie dans un véhicule à roues comme une voiture peut être améliorée en mesurant la rotation des roues au lieu de convertir la puissance du moteur en vitesse. La circonférence d'une roue est de $2\pi r$, où r est le rayon de la roue en cm, donc si

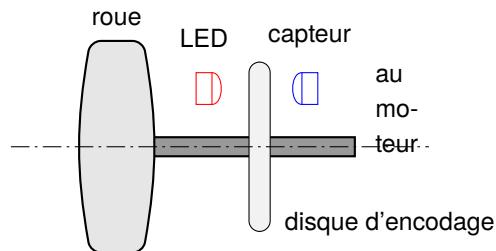


FIG. 5.8 – Codeur à roue optique

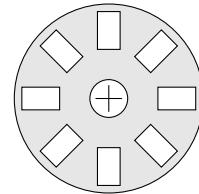


FIG. 5.9 – Disque d'encodage

n rotations sont comptées, nous savons que le robot s'est déplacé de $2\pi nr$ cm. Il est possible de construire des encodeurs de roue qui mesurent des fractions de tour. Si un signal est généré 8 fois par tour, la distance parcourue est de $2\pi nr/8$ cm, n étant maintenant le nombre de signaux comptés par l'ordinateur.

Il existe de nombreuses façons différentes de mettre en œuvre les codeurs de roue. Une conception populaire consiste à utiliser une source de lumière telle qu'une diode électroluminescente (LED), un capteur de lumière et un disque de codage fixé à l'axe de la roue (Fig. 5.8). Le disque est perforé de trous (Fig. 5.9) de sorte que chaque fois qu'un trou est opposé à la source lumineuse, le capteur génère un signal.

La prise en charge des encodeurs de roue dans les robots éducatifs varie :

- Si un robot n'a pas d'encodeurs de roue, il doit être calibré ;
- Le robot peut avoir des encodeurs de roue qui sont utilisés en interne ;
- Certains robots comme le LEGO® Mindstorms permettent à l'utilisateur de lire les encodeurs.

L'activité suivante propose une expérience permettant de mesurer la distance parcourue en comptant les tours d'une roue. Elle peut être réalisée même si votre robot ne possède pas d'encodeurs de roue ou s'ils ne sont pas accessibles.

Activity 5.11: Codage de roue

- Faites une marque au sommet d'une roue du robot à l'aide d'une craie ou en fixant un étroit morceau de ruban adhésif de couleur. Écrivez un programme qui oblige le robot à se déplacer en ligne droite pendant une durée déterminée. Exécutez le programme et prenez une vidéo du côté du robot à l'aide de la caméra de votre smartphone.
- Visionnez la vidéo et déterminez le nombre de tours en comptant le nombre de fois où la marque se trouve en haut de la roue.
- Mesurez le rayon de la roue et calculez la distance parcourue. Dans quelle mesure le résultat est-il proche de la distance réelle mesurée sur le sol ?
- Répétez la mesure en utilisant $n = 2$ puis $n = 4$ marques équidistantes

sur la roue. Déterminez le nombre de tours en comptant le nombre de fois où une marque se trouve en haut de la roue et divisez par n . Calculez la distance.

5.9 Systèmes de navigation inertie

Un *système de navigation inertie* (INS) mesure directement l'accélération linéaire et la vitesse angulaire et les utilise pour calculer la position d'un véhicule. Le terme *unité de mesure inertie* (IMU) est également utilisé, mais nous préférons le terme INS qui fait référence à l'ensemble du système. L'intégration de l'accélération depuis la pose initiale jusqu'à l'instant présent τ donne la vitesse actuelle :

$$v = \int_0^\tau a(t) dt .$$

De même, l'intégration de la vitesse angulaire donne le changement de cap :

$$\theta = \int_0^\tau \omega(t) dt .$$

Dans un SIN, nous ne disposons pas de fonctions continues à intégrer ; l'accélération et la vitesse angulaire sont échantillonnées et la sommation remplace l'intégration :

$$v_n = \sum_{i=0}^n a_n \Delta t, \quad \theta_n = \sum_{i=0}^n \omega_n \Delta t .$$

Les INS sont sujets à des erreurs dues à des imprécisions dans la mesure elle-même ainsi qu'à des variations causées par des facteurs environnementaux tels que la température, et par l'usure de l'unité. La mesure inertie est souvent combinée avec le GPS (8.3) pour mettre à jour la position avec un emplacement absolu.

Les systèmes de navigation inertie pour robots sont construits avec des *systèmes microélectromécaniques* (MEMS), qui utilisent des techniques de fabrication de circuits intégrés combinant des éléments mécaniques avec des composants électroniques qui s'interfacent avec l'ordinateur du robot.

5.9.1 Accelerometers

Si vous avez déjà pris l'avion, vous avez ressenti une force qui vous a poussé vers votre siège en raison de l'accélération rapide de l'avion au décollage. À l'atterrissement, vous êtes repoussé de votre siège. Vous pouvez également en faire l'expérience dans une voiture qui accélère rapidement ou qui effectue un arrêt d'urgence. L'accélération est liée à la force par la deuxième loi de Newton $F = ma$, où m est la masse. En mesurant la force exercée sur un objet, on mesure l'accélération.



FIG. 5.10 – Accélération vers l'avant

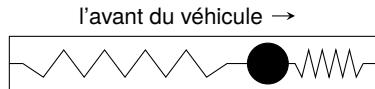


FIG. 5.11 – Décélération (freinage)

Les figures 5.10–5.11 montrent comment un accéléromètre peut être construit à partir d'un objet (appelé masse) relié à un ressort. Plus l'accélération est importante, plus la force exercée par la masse sur le ressort est grande, ce qui entraîne la compression du ressort. La direction dans laquelle la masse se déplace donne le signe de l'accélération : vers l'avant ou vers l'arrière. L'ampleur de la force est mesurée indirectement en mesurant la distance parcourue par la masse. Vous pouvez constater que les diagrammes correspondent à notre expérience : lorsqu'une voiture accélère, vous êtes repoussé sur le siège, mais lorsqu'elle décèle (freine), vous continuez à avancer.

5.9.2 Gyroscopes

Un *gyroscope* (“gyro”) utilise le principe de la force de Coriolis pour mesurer la vitesse angulaire. Ce concept est expliqué dans les manuels de physique et nous ne nous y attarderons pas ici. Il existe de nombreux types de gyroscopes :

- Les gyroscopes classiques possèdent des disques mécaniques en rotation qui sont montés sur des cardans de sorte que l'axe de rotation reste fixe dans l'espace. Ces gyroscopes sont extrêmement précis mais sont très lourds et consomment beaucoup d'énergie. On les trouve sur les véhicules de grande valeur comme les avions et les fusées.
- Les gyroscopes à laser en anneau (RLG) n'ont (presque) aucune pièce mobile et sont préférés aux gyroscopes mécaniques pour la plupart des applications. Ils sont basés sur l'envoi de deux faisceaux laser dans des directions opposées autour d'une trajectoire circulaire ou triangulaire. Si le gyroscope est en rotation, le trajet suivi par un faisceau laser sera plus long que celui suivi par l'autre faisceau. La différence est proportionnelle à la vitesse angulaire et peut être mesurée et transmise à l'ordinateur de navigation.
- Les gyroscopes vibratoires de Coriolis (CVG) fabriqués à l'aide de techniques MEMS sont présents dans les smartphones et les robots. Ils sont peu coûteux et extrêmement robustes, bien que leur précision ne soit pas aussi bonne que celle des gyroscopes présentés précédemment. Nous allons maintenant donner un aperçu de leur fonctionnement.

La figure 5.12 montre un CVG appelé *gyroscope à fourche*. Deux masses carrées sont attachées par des poutres flexibles à des ancrages montés sur la base du composant. Des moteurs forcent les masses à vibrer de gauche à droite. Si le composant tourne, les masses se déplacent vers le haut et vers le bas d'une distance propor-

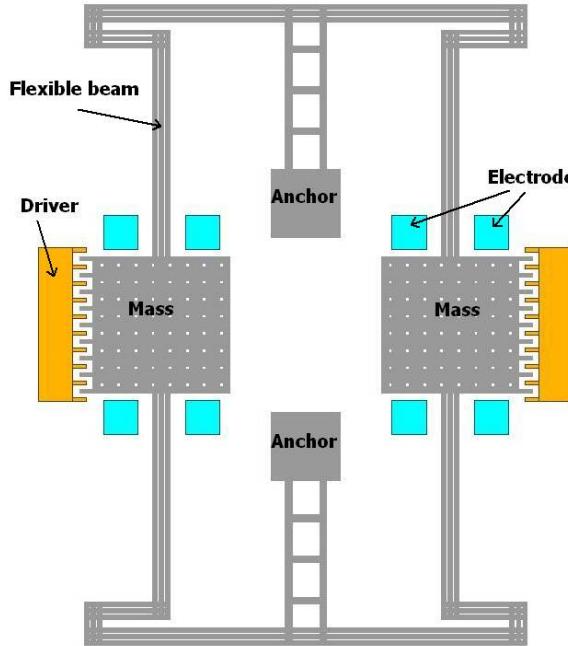


FIG. 5.12 – Gyroscope à diapason (Avec l'aimable autorisation de Zhili Hao, Old Dominion University)

tionnelle à la vitesse angulaire. Les masses et les électrodes forment les plaques des condensateurs dont la capacité augmente ou diminue lorsque les plaques se rapprochent ou s'éloignent.

La théorie du fonctionnement du gyroscope à diapason est illustrée sur la Fig. 5.13. Les masses (carrés gris) sont forcées de vibrer à la même fréquence comme les deux branches d'un diapason. Elles vibrent dans des directions différentes, c'est-à-dire qu'elles se rapprochent (flèches bleues en pointillés) ou s'éloignent l'une de l'autre (flèches rouges en pointillés). L'élément tourne autour d'un axe perpendiculaire à son centre (le cercle avec une croix indique l'axe de rotation qui est perpendiculaire au plan du papier). La force de Coriolis est une force dont la direction est donnée par le produit vectoriel croisé de l'axe de rotation et du mouvement de la masse, et dont la magnitude est proportionnelle à la vitesse linéaire de la masse et à la vitesse angulaire du gyroscope. Comme les masses se déplacent dans des directions différentes, les forces résultantes seront égales mais dans des directions opposées (flèches pleines). Les masses s'approchent ou s'éloignent des électrodes (petits rectangles) et la variation de la capacité peut être mesurée par un circuit.

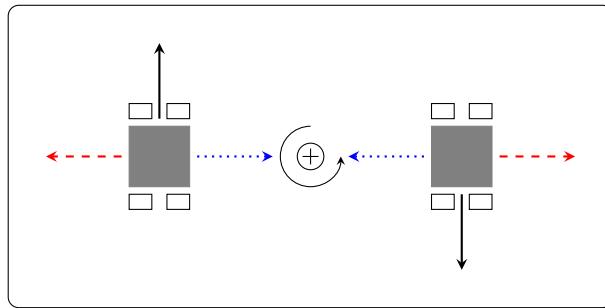


FIG. 5.13 – Physique d'un gyroscope à diapason : les flèches en pointillés rouges et les flèches en pointillés bleus indiquent la direction de la vibration ; les flèches noires pleines indiquent la direction de la force de Coriolis

5.9.3 Applications

Un système de navigation inertiel possède trois accéléromètres et trois gyroscopes afin de pouvoir calculer la position du véhicule en trois dimensions. Ceci est nécessaire pour les avions et autres véhicules robotisés. Les airbags utilisent un accéléromètre qui détecte la décélération rapide dans le sens avant-arrière qui se produit lors d'un accident de voiture. Cela provoque une expansion explosive de l'airbag. On peut imaginer d'autres applications pour ces composants dans les voitures. Un accéléromètre dans le sens haut-bas peut détecter si la voiture est tombée dans un nid-de-poule. Un gyroscope mesurant la rotation autour de l'axe vertical peut détecter un dérapage, tandis que le gyroscope mesurant la rotation autour de l'axe avant-arrière peut détecter si la voiture se retourne.

5.10 Degrés de liberté et nombre d'actionneurs

Le nombre de *degrés de liberté (DOF)* d'un système est la dimensionnalité des coordonnées nécessaires pour décrire la pose d'un robot mobile ou la pose de l'effecteur final d'un manipulateur robotique.³ Par exemple, un hélicoptère possède six DOF car il peut se déplacer dans les trois dimensions spatiales et tourner autour des trois axes. Par conséquent, une coordonnée à six dimensions ($x, y, z, \phi, \psi, \theta$) est nécessaire pour décrire sa pose.

Les termes utilisés pour décrire les rotations

Un hélicoptère peut tourner autour de ses trois axes. Les rotations sont appelées : (a) tangage : le nez se déplace de haut en bas ; (b) roulis : le

³. Cette section et les suivantes sont plus avancées et peuvent être sautées lors de votre première lecture. De plus, certains des exemples concernent des manipulateurs robotiques décrits dans le Chap. 16.

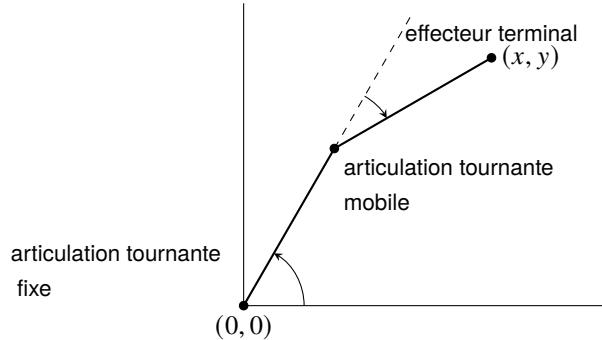


FIG. 5.14 – Un bras robotique à deux articulations avec deux DOF

corps tourne autour de son axe longitudinal; (c) lacet : le corps tourne de gauche à droite autour de l'axe de son rotor.

Le bras robotique à deux articulations de la Fig. 5.14 ne possède que deux DOF car son effecteur final se déplace dans un plan et ne tourne pas ; il peut donc être décrit par des coordonnées bidimensionnelles (x, y) . En examinant à nouveau la Fig. 5.4, vous pouvez voir qu'un robot mobile se déplaçant sur une surface plane possède trois DOF, car sa pose est définie par une coordonnée tridimensionnelle (x, y, θ) . Un train n'a qu'une seule DOF car il est contraint par les rails à se déplacer en avant (ou occasionnellement en arrière) le long de la voie. Il suffit d'une seule coordonnée (x) , la distance du train par rapport à une origine arbitraire de la voie, pour spécifier la pose du train.

Nous avons besoin de plus d'informations que les degrés de liberté pour décrire un mouvement robotique. Considérons un véhicule tel qu'une voiture, un vélo ou une chaise de bureau. Bien que trois coordonnées (x, y, θ) soient nécessaires pour décrire sa pose, nous ne pouvons pas nécessairement déplacer le véhicule directement d'une pose à une autre. Une chaise de bureau peut être déplacée *directement* en tout point du plan et orientée dans n'importe quelle direction. Une voiture ou une bicyclette située à $(2, 0, 0^\circ)$ (pointée le long de l'axe positif x) ne peut pas être déplacée directement le long de l'axe y vers la position $(2, 2, 0^\circ)$. Une manœuvre plus complexe est nécessaire.

Nous devons connaître le nombre de ses *actuateurs* (généralement des moteurs) et leur configuration. Un robot à entraînement différentiel possède deux actionneurs, un pour chaque roue, bien que le robot lui-même ait trois DOF. Les moteurs déplacent le robot le long d'un axe en avant et en arrière, mais en appliquant une puissance inégale, nous pouvons modifier le cap du robot. Le bras à deux articulations de la Fig. fig.two-link possède deux moteurs, un pour chaque articulation tournante, de sorte que le nombre d'actionneurs est égal au nombre de DOF. Enfin, un train n'a

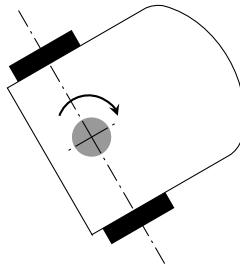


FIG. 5.15 – Un robot qui ne peut tourner qu'autour d'un axe (point gris)

qu'un seul actionneur, le moteur qui le fait avancer ou reculer dans son unique DOF.

Activity 5.12: Robot qui peut seulement tourner

- La figure 5.15 montre un robot à entraînement différentiel dont le centre de rotation est traversé par une tige fixe. La tige empêche le robot de changer de position, lui permettant uniquement de tourner autour de son axe vertical. Caractérisez cette configuration : le nombre d'actionneurs et le nombre de DOF.
- Quels types de tâches ce robot pourrait-il effectuer ? Quels sont les avantages et les inconvénients de cette configuration ?

5.11 Le nombre relatif d'actionneurs et de DOF

Analysons les systèmes où :

- Le nombre d'actionneurs est égal au nombre de DOF ;
- Le nombre d'actionneurs est inférieur au nombre de DOF ;
- Le nombre d'actionneurs est supérieur au nombre de DOF.

Le nombre d'actionneurs est égal au nombre de DOF

Un train possède un actionneur (son moteur) qui déplace le train le long de son unique DOF. Le bras robotique à deux lignes de la Fig. 5.14 possède deux actionneurs et deux DOF. Une pince robotique peut être construite avec trois moteurs qui font tourner la pince dans chacune des trois orientations (roulis, tangage, lacet). L'avantage d'avoir un nombre égal d'actionneurs et de DOF est que le système est relativement facile à contrôler : chaque actionneur est commandé individuellement pour déplacer le robot vers la position souhaitée dans le DOF qu'il contrôle.

Le nombre d'actionneurs est inférieur au nombre de DOF

Les robots mobiles ont généralement moins d'actionneurs que de DOF. Un robot à entraînement différentiel et une voiture n'ont que deux actionneurs, mais ils peuvent atteindre toutes les positions tridimensionnelles possibles dans le plan. Le fait d'avoir moins d'actionneurs rend le système moins coûteux, mais les problèmes de planification et de contrôle du mouvement sont beaucoup plus difficiles. Le stationnement parallèle d'une voiture est réputé pour sa difficulté : deux rotations et une translation sont nécessaires pour effectuer un simple déplacement latéral (Figs. 5.28–5.29).

Un exemple extrême est celui d'une montgolfière qui ne possède qu'un seul actionneur (un réchauffeur) qui injecte plus ou moins d'air chaud dans le ballon et contrôle ainsi son altitude. Cependant, les vents peuvent faire bouger la montgolfière dans l'une des trois directions de l'espace et même la faire tourner (au moins partiellement) dans trois orientations, de sorte que l'opérateur de la montgolfière ne peut jamais la contrôler précisément. Une montgolfière diffère donc d'un ascenseur : les deux ont un seul actionneur, mais l'ascenseur est contraint par son arbre à se déplacer dans une seule DOF.

Pour un autre exemple de la relation complexe entre les DOF et le nombre d'actionneurs, le lecteur est invité à étudier la commande de vol des hélicoptères. Les hélicoptères sont très maniables (encore plus que les avions qui ne peuvent pas voler en arrière), mais un pilote contrôle le vol de l'hélicoptère en utilisant seulement trois actionneurs :

- Le *cyclique* contrôle le pas du *arbre* du rotor principal qui détermine si l'hélicoptère se déplace en avant, en arrière ou sur un côté.
- Le paramètre *collective* contrôle le pas des *pales* du rotor principal qui détermine si l'hélicoptère se déplace vers le haut ou vers le bas.
- Les *pédales* contrôlent la vitesse du rotor de queue qui détermine la direction dans laquelle le nez de l'hélicoptère pointe.

Le nombre d'actionneurs est supérieur au nombre de DOF

Cela ne semble pas être une bonne idée d'avoir *plus* d'actionneurs que de DOF, mais en pratique, de telles configurations sont souvent utiles. Les systèmes des Figs. 5.16–5.17 ont plus d'actionneurs que de DOF. Le bras manipulateur robotique de la Fig. 5.16 possède quatre liaisons tournant dans le plan avec des actionneurs (moteurs) a₁, a₂, a₃, a₄ aux articulations. Nous supposons que l'effecteur final est fixé au lien de a₄ et ne peut pas tourner, donc sa pose est définie par sa position (*x*, *y*) et une orientation fixe. Par conséquent, bien que le bras ait quatre actionneurs, il n'a que deux DOF car il ne peut déplacer l'effecteur final qu'horizontalement et verticalement.

Le robot mobile avec un bras (Fig. 5.17) possède trois actionneurs : un moteur qui

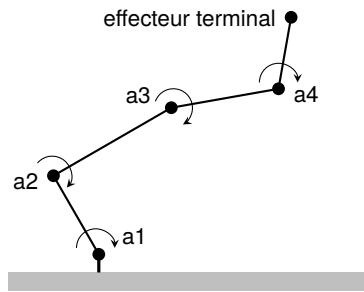


FIG. 5.16 – Bras de robot : deux DOF et quatre actionneurs

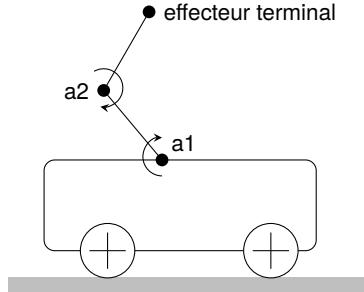


FIG. 5.17 – Robot mobile et bras : deux DOF et trois actionneurs

déplace le robot en avant et en arrière, et des moteurs pour chacune des deux articulations tournantes. Cependant, le système n'a que deux DOF puisque la pose de son effecteur final est définie par une coordonnée bidimensionnelle (x, y). Les systèmes comportant plus d'actionneurs que de DOF sont appelés *systèmes redondants*.

Dans la mesure du possible, les ingénieurs évitent d'utiliser plus d'un actionneur agissant sur le même DOF car cela augmente la complexité et le coût d'un système. La cinématique inverse (Sect. 16.2) d'un système redondant donne lieu à un nombre infini de solutions qui compliquent le fonctionnement du système. Pour le robot mobile avec le bras (Fig. 5.17), il existe un nombre infini de positions de la base et du bras qui amènent l'effecteur final à une position atteignable spécifique.

Dans certaines situations, un système redondant est nécessaire car la tâche ne pourrait pas être exécutée avec moins d'actionneurs. La figure 5.18 montre comment le bras robotique à quatre liaisons de la figure 5.16 peut déplacer l'effecteur final vers une position qui est bloquée par un obstacle et donc inaccessible par un bras à deux liaisons (figure 5.19), même si dans les deux configurations la longueur totale des liaisons est égale.

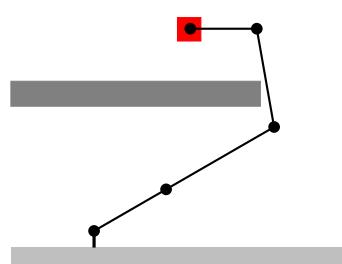


FIG. 5.18 – Arm with four actuators can reach a hidden position

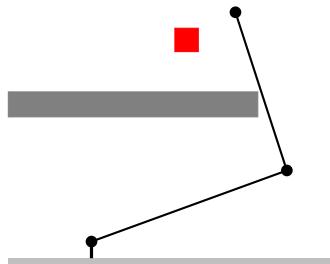


FIG. 5.19 – Arm with two actuators is blocked by an obstacle

Un avantage important des systèmes redondants provient des actionneurs ayant des caractéristiques différentes. Le robot mobile de la Fig. reffig.hi-act-low-dof-2 peut s'approcher rapidement de la cible, bien que sa position finale puisse ne pas être précise en raison d'erreurs comme un terrain accidenté. Une fois que le robot mobile s'arrête, les moteurs des articulations qui n'ont pas à faire face au terrain peuvent être positionnés avec précision. Bien que le positionnement soit précis, ces articulations n'ont pas la portée étendue de la base mobile.

Exemple d'un système comportant plus d'actionneurs que de DOF

La figure 5.20 (vues de dessus et de côté) montre une configuration avec *deux actionneurs et une DOF*. Le système modélise une grue robotisée qui déplace un poids lourd vers une position verticale spécifique. La figure 5.21 montre une grue construite à partir d'un robot Thymio et de composants LEGO®.⁴

Le système est construit à partir d'un robot mobile à entraînement différentiel, mais les roues ne sont pas directement utilisées pour contrôler le mouvement du système. Au lieu de cela, chaque roue est un actionneur indépendant. (Rappelez-vous que la puissance de chaque roue d'un robot à entraînement différentiel peut être réglée indépendamment sur n'importe quelle valeur dans une plage telle que -100 à 100).

Le robot est orienté vers la gauche. La roue motrice droite de la Fig. 5.20 (le rectangle noir en haut de la vue de dessus et caché derrière le robot dans la vue de côté) commande une paire de roues de route (grises) qui déplacent rapidement le robot en avant et en arrière. À son tour, le câble fait monter et descendre rapidement le poids.

Les roues sont montées sur une structure (en bleu) qui est fixée au corps du robot. Il existe plusieurs options pour transférer la puissance de la roue motrice droite aux roues de la route : friction, poulies et courroies, et engrenages. Chaque option a ses propres avantages et inconvénients, et les trois sont utilisées dans les voitures : l'embrayage utilise la friction, les courroies sont utilisées pour la synchronisation et pour faire fonctionner les composants auxiliaires comme les pompes à eau, et les engrenages sont utilisés dans la transmission pour contrôler le couple appliqué à chaque roue.

La roue motrice gauche (le rectangle noir en bas de la vue de dessus et à l'avant de la vue latérale) commande un treuil (rouge) qui enroule ou déroule un câble attaché au poids qui monte ou descend sur un roulement fixe. Le diamètre du treuil est beaucoup plus petit que celui des roues motrices, de sorte qu'il peut déplacer le poids par petits incrément lorsqu la roue motrice gauche tourne. L'objectif de la conception est de pouvoir effectuer un positionnement précis du poids même si le treuil déplace le câble à une vitesse beaucoup plus lente que le corps du robot.

4. Le treuil est à droite du robot et non à gauche comme dans la figure 5.20.

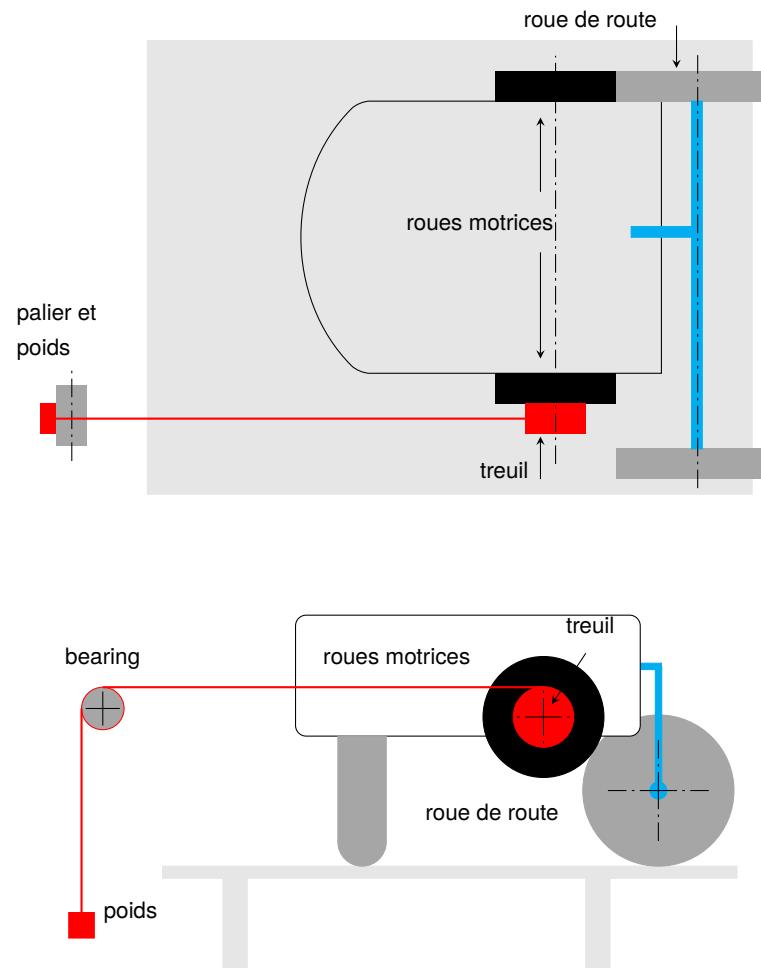


FIG. 5.20 – Grue robotique construite à partir d'un robot mobile et d'un treuil (vue de dessus en haut, vue de côté en bas); sur la vue de côté, la roue gauche n'est pas représentée

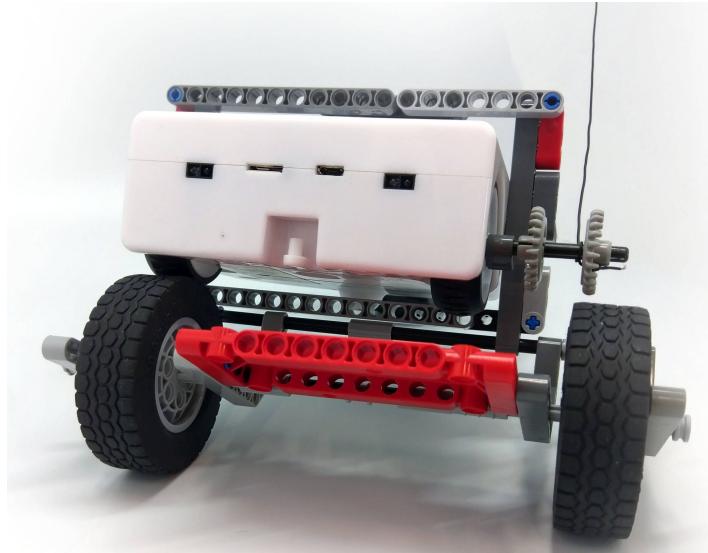


FIG. 5.21 – Grue robotique construite à partir d'un robot Thymio et de composants LEGO®

Deux activités sont prévues dans cette section. La première activité s'adresse aux lecteurs qui possèdent de bonnes compétences en construction et un kit robotique approprié. La seconde activité propose d'autres moyens de démontrer le concept de deux actionneurs dans un DOF.

Activity 5.13: Grue robotique

- Construisez la grue robotisée représentée sur la Fig. 5.20. Expliquez votre choix de mécanisme pour relier la roue motrice aux roues de la route.
- Écrivez un programme qui, étant donné la position actuelle du poids et une position cible, déplace le poids vers la position cible. Vous pouvez également envoyer des commandes aux moteurs à l'aide d'une télécommande ou d'un ordinateur connecté au robot.
- Expérimitez les vitesses de rotation relatives des roues motrices gauche et droite qui contrôlent respectivement les roues de la route et le treuil. Devez-vous déplacer les deux actionneurs séparément ou simultanément ?

Activity 5.14: Grue robotique (alternatives)

- Écrivez un programme qui permet à un robot mobile de se déplacer en avant et en arrière. Placez un morceau de ruban adhésif noir relativement loin de la position initiale du robot. L'objectif est de faire en sorte que le robot s'arrête le plus près possible du début du ruban, sans vérifier continuellement le capteur.
- Le programme comporte trois modes de fonctionnement. (1) Le robot se déplace rapidement, vérifie occasionnellement son capteur et s'arrête lorsqu'il détecte la bande. (2) Comme en (1), mais le robot se déplace lentement, en vérifiant son capteur relativement souvent. (3) Comme en (1) mais lorsque la bande est détectée, le robot recule en utilisant la vitesse et la période d'échantillonnage comme en (2).
- Exécutez le programme et comparez les résultats des trois modes : le temps jusqu'à l'arrêt du robot et l'erreur entre la position finale du robot et le début de la bande.
- Vous pouvez également exécuter le programme avec les trois modes sur un ordinateur et expérimenter les paramètres de mouvement et les périodes d'échantillonnage. Vous devrez choisir un modèle pour le mouvement : vitesse constante, accélération constante ou (de façon plus réaliste) accélération puis vitesse constante et enfin décélération lorsque la bande est détectée.

5.12 Mouvement holonomique et non holonomique

La section 5.10 a présenté le concept de degré de liberté (DOF) et le rôle du nombre d'actionneurs. Il existe un autre concept qui relie les DOF et les actionneurs dans le cas d'un robot mobile : le *degré de mobilité (DOM)*. Le degré de mobilité δ_m correspond au nombre de degrés de liberté auxquels les actionneurs peuvent *directement accéder*. Un robot mobile dans le plan possède au maximum trois DOF (position et cap (x, y)), le degré maximal de mobilité d'un robot mobile est donc $\delta_m = 3$.

Considérons les DOM de différents véhicules. Un train n'a qu'un seul degré de liberté parce qu'il ne peut que se déplacer vers l'avant le long des rails, et il a un seul actionneur, son moteur, qui affecte directement cet unique degré de liberté. Par conséquent, un train a un degré de mobilité de $\delta_m = 1$, ce qui signifie que l'actionneur peut accéder directement à l'unique DOF.

Un robot à entraînement différentiel possède trois DOF. Les deux actionneurs sont les deux moteurs qui agissent sur les roues. Ils peuvent accéder directement à deux DOF : (a) si les deux roues tournent à la même vitesse, le robot avance

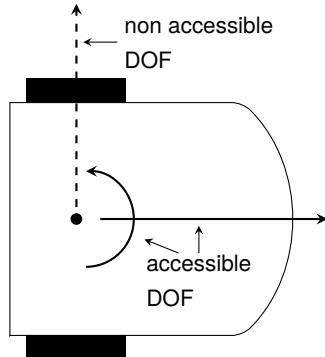


FIG. 5.22 – *DoF accessibles et non accessibles pour un robot avec entraînement différentiel*

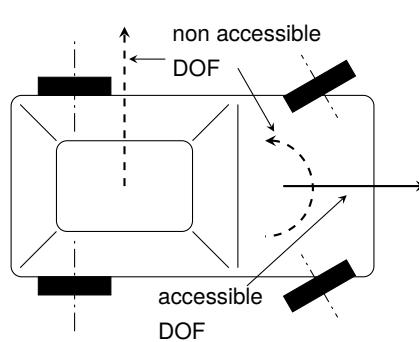


FIG. 5.23 – *DoF accessibles et non accessibles pour un robot avec direction Ackermann*

ou recule; (b) si les roues ont des vitesses opposées, le robot tourne sur place. Par conséquent, nous pouvons accéder directement aux DOF le long de l'axe de translation avant et aux DOF du cap, mais nous ne pouvons pas accéder directement aux DOF de l'axe de translation latéral (Fig. 5.22). Un robot mobile à entraînement différentiel a un degré de mobilité $\delta_m = 2 < 3 = \#DOF$.

Une voiture, comme un robot à entraînement différentiel, ne dispose que de deux actionneurs pour trois DOF : un actionneur, le moteur, donne un accès direct au degré de liberté selon l'axe longitudinal de la voiture, ce qui lui permet d'avancer et de reculer. L'autre actionneur, le volant, ne donne pas d'accès direct à d'autres DOF, il ne peut qu'orienter le premier DOF. La voiture ne peut pas tourner autour de l'axe vertical et elle ne peut pas se déplacer latéralement (Fig. 5.23). Par conséquent, une voiture n'a qu'un seul degré de mobilité, $\delta_m = 1$. Intuitivement, vous pouvez constater le degré de mobilité inférieur d'une voiture par rapport à un robot à entraînement différentiel en notant que le robot peut tourner sur place alors que la voiture ne le peut pas.

En soi, une roue standard a $\delta_m = 2$: elle peut rouler en avant et en arrière et tourner autour de l'axe vertical qui passe par le point de contact de la roue avec le sol. Une roue ne peut pas se déplacer latéralement, ce qui est en fait une bonne chose car cela empêche le véhicule de déraper sur la route dans un virage. Dans la voiture, le degré de mobilité est encore réduit à $\delta_m = 1$, car il y a deux paires de roues, une à l'avant et une à l'arrière de la voiture. Cette configuration rend impossible la rotation de la voiture autour de son axe vertical, même si les roues individuelles peuvent le faire (généralement uniquement les roues avant). La limitation à $\delta_m = 1$ donne de la stabilité à la voiture - elle ne peut pas déraper latéralement et elle ne peut pas tourner - ce qui rend la conduite à grande vitesse facile et sûre. En fait, un accident peut se

FIG. 5.24 – *Swedish wheel*FIG. 5.25 – *Omnidirectional robot*
(Courtesy LAMI-EPFL)

produire lorsque la pluie ou la neige réduit la friction de sorte que la voiture peut déraper ou tourner.

Un robot mobile autonome peut en profiter s'il a un DOM supérieur $\delta_m = 3$. Pour accéder directement à la troisième DOF, le robot doit être capable de se déplacer latéralement. Une méthode consiste à faire rouler le robot sur une boule ou une roue pivotante comme une chaise de bureau. Une autre méthode consiste à utiliser des *roues suédoises* (Fig. 5.24). Une roue suédoise est une roue standard qui possède de petites roues libres le long de sa jante afin qu'elle puisse se déplacer latéralement, ce qui permet un accès direct à la troisième DOF.

Les robots mobiles qui peuvent accéder directement aux trois DOF ($\delta_m = 3$) sont appelés *robots omnidirectionnels*. La figure 5.25 montre un robot omnidirectionnel construit avec quatre roues suédoises. Les deux paires de roues situées sur les côtés opposés du robot peuvent directement déplacer le robot à gauche, à droite, en avant et en arrière. Cette configuration est redondante mais très facile à contrôler. Pour éviter la redondance, la plupart des robots omnidirectionnels ont trois roues suédoises montées à un angle de 120° les unes des autres (Fig. 5.26). Cette configuration a $\delta_m = 3$ mais n'est pas facile à contrôler en utilisant les coordonnées familiaires x, y .

Les valeurs relatives des DOF et des DOM d'un robot définissent le concept de mouvement holonomique. Un robot a un *mouvement holonomique* si $\delta_m = \#DOF$ et il a un *mouvement non holonomique* $\delta_m < \#DOF$. Un robot holonome comme celui de la figure 5.25 peut contrôler directement toutes ses DOF sans manœuvres difficiles. La figure 5.27 montre à quel point il est facile pour le robot omnidirectionnel à roues suédoises (Fig. 5.26) d'effectuer un stationnement parallèle.

Une voiture et un robot à entraînement différentiel sont non ergonomiques car leur DOM ($\delta_m = 1$ et $\delta_m = 2$, respectivement) est inférieur à leur DOF qui est de trois. En raison de ce degré de mobilité limité, ces véhicules ont besoin de manœuvres de direction complexes, par exemple pour effectuer un stationnement parallèle. Il existe une différence significative entre les deux véhicules. Le robot à entraînement différentiel nécessite trois mouvements distincts, mais ils sont très simples (Fig. 5.28) : rotation à gauche, recul, rotation à droite. La voiture a également besoin de trois mouvements distincts, mais ils sont extrêmement difficiles à exécuter correctement (Fig. 5.29). Vous devez estimer le point de départ de la manœuvre, la précision de chaque virage et la distance à parcourir entre les virages. Le DOM

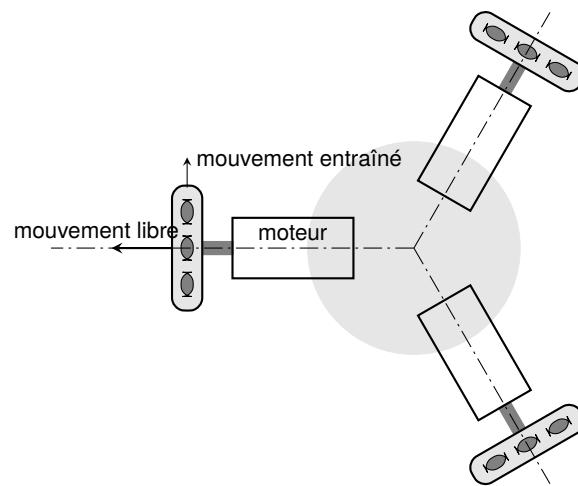


FIG. 5.26 – Robot omnidirectionnel avec trois roues suédoises

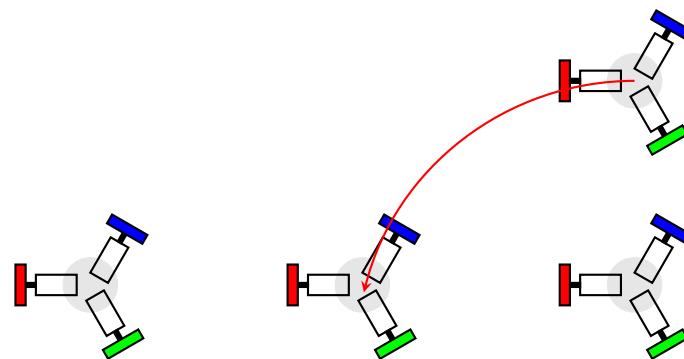


FIG. 5.27 – Parking parallèle par un robot omnidirectionnel

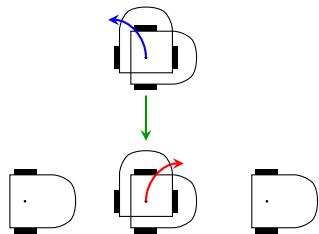


FIG. 5.28 – Parallel parking for a non-holonomic differential drive robot

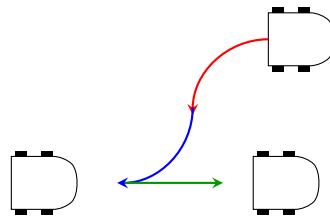


FIG. 5.29 – Parallel parking for a non-holonomic car

supérieur du robot à entraînement différentiel est avantageux dans cette situation.

Activity 5.15: Mouvement holonomique et non-holonomique

- Regardez à nouveau le robot mobile qui est contraint à un mouvement de rotation uniquement (Fig. 5.15). Quel est son degré de mobilité δ_m ? Est-il holonomique ou non ?
- La figure 5.30 montre un robot qui nettoie les murs d'un bâtiment. Il y a deux ancrages d'où descendent des câbles qui passent par des yeux fixés au corps du robot, puis par des treuils actionnés par les roues du robot. En enroulant et déroulant les câbles, le robot monte et descend le long du mur. Cependant, si les deux moteurs ne font pas bouger les roues précisément à la même vitesse de rotation, le robot se balancera d'un côté à l'autre. Combien de DOF et combien de DOM ce robot possède-t-il ? Est-il holonomique ou non ?

5.13 Résumé

Un robot mobile comme une voiture autonome ou un explorateur de Mars ne disposera pas toujours de points de repère pour sa navigation. L'odométrie est utilisée pour amener le robot à proximité de son objectif sans référence à l'environnement. Le robot estime sa vitesse et sa vitesse de rotation à partir de la puissance appliquée à ses moteurs. L'odométrie peut être améliorée en utilisant des encodeurs de roue pour mesurer le nombre de tours des roues, plutôt que de déduire la vitesse de rotation à partir de la puissance du moteur. Le changement de position d'un robot bon marché se déplaçant en ligne droite peut être calculé en multipliant la vitesse par le temps. Si le robot tourne, des calculs trigonométriques sont nécessaires pour calculer la nouvelle position et l'orientation. Même avec des encodeurs de roue, l'odométrie est sujette à des erreurs qui peuvent être très importantes si l'erreur porte sur le cap.

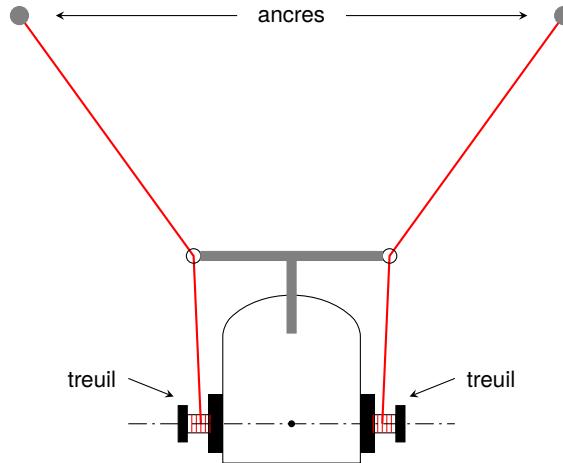


FIG. 5.30 – Robot pour nettoyer un mur

La navigation inertielle utilise des accéléromètres et des gyroscopes pour améliorer la précision de l'odométrie. L'intégration de l'accélération donne la vitesse et l'intégration de la vitesse angulaire donne le cap. Les systèmes micro-électromécaniques ont rendu la navigation inertielle suffisamment bon marché pour être utilisée en robotique.

Les DOF d'un système sont le nombre de dimensions dans lesquelles il peut se déplacer - jusqu'à trois dimensions sur une surface et jusqu'à six dimensions dans l'air ou sous l'eau - mais un robot peut être contraint d'avoir moins que le nombre maximum de DOF. Une autre considération est le nombre et la configuration des actionneurs d'un robot qui définissent son degré de mobilité. Si le DOM est égal au nombre de DOF, le robot est holonome et peut se déplacer directement d'une pose à une autre, bien qu'il puisse être difficile à contrôler. Si le DOM est inférieur au nombre de DOF, le robot est non ergonomique ; il ne peut pas se déplacer directement d'une pose à l'autre et devra effectuer des manœuvres complexes pour réaliser certaines tâches.

5.14 Lectures complémentaires

Un traitement mathématique détaillé des erreurs d'odométrie en deux dimensions est donné dans [43, Sect. 5.24]. Pour un aperçu de la navigation inertielle, voir [27, 34]. Les manuels avancés sur la robotique présentent l'holonomie : [11, 12, 43, 44].

Chapitre 6

Control

Les algorithmes de robotique prennent des décisions. Le robot se voit confier une tâche, mais pour l'accomplir, il doit prendre des mesures et ces mesures dépendent de l'environnement tel qu'il est détecté par les capteurs. Par exemple, si un robot doit transporter un objet d'une étagère d'un entrepôt à une piste de livraison, il doit utiliser des capteurs pour se diriger vers l'étagère appropriée, détecter et saisir l'objet, puis retourner au camion et charger l'objet. Seuls les robots qui agissent dans des environnements extrêmement bien définis peuvent effectuer de telles tâches sans informations provenant de capteurs. Prenons l'exemple d'un bras robotique qui assemble un appareil dans une usine ; si les pièces sont positionnées avec précision sur le plan de travail, le robot peut manipuler les pièces sans les détecter. Mais dans la plupart des environnements, des capteurs doivent être utilisés. Dans un entrepôt, il peut y avoir des obstacles sur le chemin de l'étagère, l'objet ne sera pas positionné précisément sur l'étagère et le camion n'est jamais garé exactement au même endroit. Le robot doit s'adapter à ces petites variations en utilisant des *algorithmes de contrôle* pour prendre des décisions : sur la base des données provenant des capteurs, quelles actions le robot doit-il effectuer pour accomplir la tâche ? Il existe une théorie mathématique sophistiquée du contrôle qui est fondamentale en robotique. Dans ce chapitre, nous présentons les concepts de base des algorithmes de contrôle.

La section 6.1 explique la différence entre deux modèles de contrôle : le contrôle en boucle ouverte où les paramètres de l'algorithme sont définis à l'avance et le contrôle en boucle fermée où les données des capteurs influencent le comportement de l'algorithme. Les sections 6.2–6.5 présentent quatre algorithmes de contrôle en boucle fermée de plus en plus sophistiqués. Le concepteur d'un robot doit choisir parmi ces algorithmes et d'autres similaires celui qui offre des performances adéquates pour un coût de calcul minimal.

6.1 Modèles de contrôle

Un algorithme de contrôle peut décider d'une action de deux façons. Dans un système en boucle ouverte, les paramètres de l'algorithme de contrôle sont prédéfinis et ne changent pas pendant le fonctionnement du système. Dans un système en boucle fermée, les capteurs mesurent l'erreur entre l'état souhaité du système et son état réel, et cette erreur est utilisée pour décider de l'action à entreprendre.

6.1.1 Contrôle en boucle ouverte

Un grille-pain est une machine qui effectue des actions de manière semi-autonome. Vous placez des tranches de pain dans le grille-pain, vous réglez le minuteur et vous appuyez sur le levier pour lancer l'action de grillage. Comme nous le savons tous, les résultats ne sont pas garantis : si la durée de la minuterie est trop courte, nous devons faire griller le pain à nouveau ; si la durée de la minuterie est trop longue, l'odeur de pain brûlé flotte dans la cuisine. Le résultat est incertain car le grille-pain est un système de contrôle en boucle ouverte. Il ne vérifie pas le résultat de l'action de grillage pour voir si le résultat souhaité a été atteint. Les systèmes en boucle ouverte sont très familiers : sur une machine à laver, vous pouvez régler la température de l'eau, la durée du cycle et la quantité de détergent utilisée, mais la machine ne mesure pas la "propreté" des vêtements (quelle qu'en soit la signification) et ne modifie pas ses actions en conséquence.

Un robot mobile qui se déplace vers une position cible en se basant uniquement sur l'odométrie (Sec. 5.4) utilise également un contrôle en boucle ouverte. En gardant la trace de la puissance du moteur et de la durée de fonctionnement des moteurs, le robot peut calculer la distance parcourue. Cependant, les variations de la vitesse des roues et de la surface sur laquelle le robot se déplace entraînent une incertitude sur la position finale du robot. Dans la plupart des applications, l'odométrie peut être utilisée pour déplacer le robot jusqu'au voisinage de la position cible, après quoi des capteurs sont utilisés pour déplacer le robot jusqu'à la position cible précise, par exemple en utilisant des capteurs pour mesurer la distance à un objet.

6.1.2 Contrôle en boucle fermée

Pour obtenir un comportement autonome, les robots utilisent des systèmes de contrôle en boucle fermée. Nous avons déjà rencontré des systèmes en boucle fermée dans les véhicules Braintenberg (Activité 3.5) :

Spécification (Attraction et répulsion) : Lorsqu'un objet s'approche du robot par derrière, celui-ci s'enfuit jusqu'à ce qu'il soit hors de portée.

Le robot doit *mesurer* la distance qui le sépare de l'objet et s'arrêter lorsque cette distance est suffisamment grande. Le réglage de la puissance des moteurs dépend de la mesure de la distance, mais le robot se déplace à une vitesse qui dépend de la puissance des moteurs, qui modifie la distance à l'objet, qui modifie à nouveau le réglage de la puissance, qui . . . Ce comportement circulaire est à l'origine du terme "boucle fermée".

Nous formalisons maintenant la spécification d'un système de commande en boucle fermée pour un robot (Fig. 6.1). La variable r représente la *valeur de référence*, la spécification de la tâche du robot. Dans un robot d'entrepôt, les valeurs de référence comprennent la position du robot par rapport à une pile d'étagères et la distance

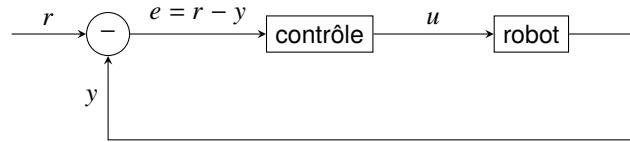


FIG. 6.1 – Un système de contrôle en boucle fermée

entre le bras de préhension et l'objet à ramasser. Une valeur de référence ne peut pas être utilisée directement par le robot ; elle doit être transformée en une *valeur de commande* u . Par exemple, si la valeur de référence est la position du robot par rapport à une étagère, la valeur de commande sera les paramètres de puissance des moteurs et la durée de fonctionnement des moteurs. La variable y représente la sortie, c'est-à-dire l'état réel du robot, par exemple, la distance à un objet.

Le modèle de la Fig. 6.1 est également appelé un *système de contrôle à rétroaction* car la valeur de sortie y est renvoyée à l'algorithme de contrôle et utilisée pour calculer la valeur de contrôle. La sortie est comparée à la valeur de référence pour calculer $e = r - y$, l'*erreur*. L'algorithme de commande utilise l'erreur pour générer le signal de commande u qui constitue l'entrée du robot.

6.1.3 La période d'un algorithme de contrôle

Les algorithmes de contrôle sont exécutés périodiquement (Algorithme 6.1). Le logiciel du robot initialise une *variable de temporisation* à la durée requise de la période d'exécution de l'algorithme, par exemple, toutes les 20 ms. L'ordinateur embarqué possède une *horloge matérielle* qui “sauta” à intervalles fixes, provoquant une interruption. L'interruption est gérée par le système d'exploitation qui décrémente la valeur de la variable timer. Lorsque la valeur de cette variable atteint zéro, la minuterie a expiré et un événement est déclenché dans le logiciel, ce qui entraîne l'exécution de l'algorithme de contrôle.

La période de l'algorithme est un paramètre important dans la conception d'un système de contrôle. Si la période est trop courte, de précieuses ressources informatiques seront gaspillées et l'ordinateur peut être surchargé au point que les commandes au robot arrivent trop tard. Si la période est trop longue, le robot ne pourra pas réagir à temps pour corriger les erreurs de son mouvement.

Exemple Considérons un robot qui s'approche d'un objet situé à 10 cm à une vitesse de 2 cm/s. Une période de commande de 1 ms entraînerait un gaspillage de ressources informatiques car le robot ne se déplacera que de 0,002 cm (0,02 mm) pendant chaque cycle de 1 ms de l'algorithme de commande. Les variations de la puissance du moteur sur de si petites distances n'affecteront pas la capacité du robot à accomplir sa tâche. À l'extrême opposé, une période de contrôle de 2 s est encore

Algorithm 6.1: Schéma de l'algorithme de contrôle	
integer period	// Duration of timer period
integer timer	// Timer variable
1: period ← ...	// Period in milliseconds
2: timer ← period	// Initialize the timer
3: loop	
4: when timer-expired-event occurs	
5: control algorithm	// Run the algorithm
6: timer ← period	// Reset the timer
// Operating system	
7: when hardware-clock-interrupt occurs	
8: timer ← timer – 1	// Decrement the timer
9: if timer = 0	// If the timer expires
10: raise timer-expired-event	// raise an event

pire : le robot se déplacera de 4 cm pendant cette période et risque de s'écraser sur l'objet. Une période de contrôle d'environ 0,25 s (250 ms) pendant laquelle le robot se déplace de 0,5 cm semble une valeur raisonnable pour commencer, car 0,5 cm est une distance significative en termes d'approche d'un objet. Vous pouvez expérimenter des périodes autour de cette valeur pour déterminer la période optimale : celle qui permet d'obtenir un comportement satisfaisant avec une période aussi longue que possible pour réduire le coût du calcul.

Activity 6.1: Définir la période de contrôle

- Dans l'exemple, nous sommes arrivés à la conclusion que la période optimale de l'algorithme de contrôle était de l'ordre de grandeur des dixièmes de seconde. Dans cette activité, nous vous demandons de réfléchir à ce que devrait être la période optimale pour d'autres algorithmes de contrôle.
- Un système de chauffage domestique contient un thermostat pour contrôler la température. Quelle serait la période optimale pour l'algorithme de contrôle ? La période dépend des paramètres techniques du système de chauffage et des propriétés physiques de la transmission de la chaleur dans les pièces. Expliquez comment vous mesureriez ces facteurs et comment ils affectent la période de contrôle.
- Considérez une voiture autonome qui essaie de se garer. Quelles hy-

- pothèses devez-vous formuler pour concevoir une période de contrôle ? Quelle serait une période raisonnable ?
- Comment les propriétés des capteurs affectent-elles la période de contrôle ? Dans l'exemple d'un robot qui s'approche d'un objet, comment la période change-t-elle si le capteur peut détecter l'objet à 2 cm, 5 cm, 10 cm, 20 cm, 40 cm ?

Nous définissons maintenant une séquence de quatre algorithmes de contrôle, chacun s'appuyant sur le précédent et fournissant un contrôle plus précis, au prix d'une plus grande complexité de calcul. En pratique, le concepteur du système doit choisir l'algorithme le plus simple qui permet au robot de remplir sa tâche.

Les algorithmes sont présentés dans le contexte d'un robot qui doit s'approcher d'un objet et s'arrêter à une distance s devant celui-ci. La distance est mesurée par un capteur de proximité et la vitesse du robot est contrôlée en réglant la puissance des moteurs.

6.2 Contrôle on-off

Le premier algorithme de contrôle est appelé l'algorithme *on-off* ou *bang-bang* (Algorithme 6.2). On définit une constante référence qui est la distance devant l'objet à laquelle le robot doit s'arrêter. La variable `mesured` est la distance réelle mesurée par le capteur de proximité. La variable `error` est la différence entre les deux :

$$\text{erreur} \leftarrow \text{référence} - \text{mesuré},$$

qui est négatif si le robot est trop éloigné de l'objet et positif s'il est trop proche de l'objet. Les puissances des moteurs sont tournées à fond en avant ou à fond en arrière selon le signe de l'erreur. Par exemple, si la distance de référence est de 10 cm et la distance mesurée de 20 cm, le robot est trop éloigné et l'erreur est de 10 cm. Par conséquent, les moteurs doivent être réglés pour se déplacer vers l'avant.

Le robot s'approche de l'objet à pleine vitesse. Lorsque le robot atteint la distance de référence par rapport à l'objet, il faut du temps pour que le capteur soit lu et que l'erreur soit calculée. Même si le robot mesure une distance exactement égale à la distance de référence (ce qui est peu probable), le robot ne pourra pas s'arrêter immédiatement et dépassera la distance de référence. L'algorithme fait alors reculer le robot à pleine vitesse, en dépassant à nouveau la distance de référence. Lorsque la minuterie déclenche une nouvelle exécution de l'algorithme de contrôle, le robot inverse sa direction et avance à pleine vitesse. Le comportement du robot est illustré à la Fig.6.2 : le robot oscille autour de la distance de référence par rapport à l'objet. Il est très peu probable que le robot s'arrête réellement à la distance de référence ou à proximité.

Algorithm 6.2: Contrôleur désactivé	
integer reference	$\leftarrow \dots$ // Reference distance
integer measured	// Measured distance
integer error	// Distance error
1:	error \leftarrow reference – measured
2:	if error < 0
3:	left-motor-power \leftarrow 100 // Move forwards
4:	right-motor-power \leftarrow 100
5:	if error = 0
6:	left-motor-power \leftarrow 0 // Turn off motors
7:	right-motor-power \leftarrow 0
8:	if error > 0
9:	left-motor-power \leftarrow -100 // Move backwards
10:	right-motor-power \leftarrow -100

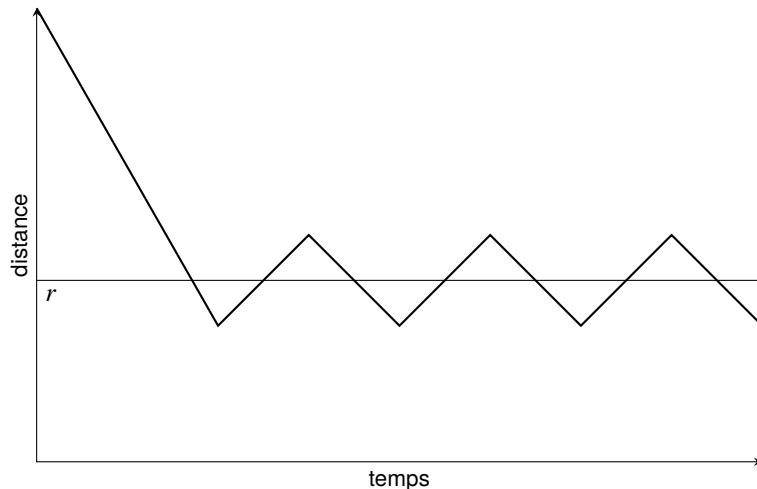


FIG. 6.2 – Comportement de l'algorithme on-off

Un autre inconvénient de l'algorithme on-off est que l'inversion fréquente et brutale de la direction entraîne des accélérations élevées. Si l'on essaie de contrôler un bras de préhension, les objets qu'il porte peuvent être endommagés. L'algorithme génère des niveaux élevés d'usure des moteurs et des autres pièces mécaniques mobiles.

Activity 6.2: Contrôleur désactivé

- Implémentez l'algorithme on-off sur votre robot pour la tâche consistant à s'arrêter à une distance de référence d'un objet.
- Exécutez-le plusieurs fois en commençant à des distances différentes de l'objet.
- L'algorithme 6.2 arrête le robot lorsque l'erreur est exactement nulle. Modifiez l'implémentation pour que le robot s'arrête si l'erreur est comprise dans une petite plage autour de zéro. Expérimitez différentes plages et voyez comment elles affectent le comportement du robot.

6.3 Contrôleur proportionnel (P)

Pour développer un meilleur algorithme, nous nous inspirons de la conduite d'un vélo. Supposons que vous êtes en train de rouler à vélo et que vous voyez que le feu de circulation devant vous est passé au rouge. Vous n'attendez pas le dernier moment, lorsque vous êtes sur la ligne d'arrêt, pour appuyer à fond sur le levier de frein ; si vous le faites, vous risquez d'être éjecté de votre vélo ! Ce que tu fais, c'est de ralentir progressivement ta vitesse : d'abord, tu arrêtes de pédaler; ensuite, tu serres doucement le frein pour ralentir un peu plus ; enfin, quand tu es sur la ligne d'arrêt et que tu vas lentement, tu serres plus fort pour arrêter complètement le vélo. L'algorithme utilisé par un cycliste peut être exprimé comme suit :

Réduisez davantage votre vitesse à mesure que vous vous rapprochez de la distance de référence.

La diminution de la vitesse est (inversement) *proportionnelle* à la proximité du feu : plus vous êtes proche, plus vous ralentisez. Le facteur de proportionnalité est appelé le *gain* de l'algorithme de contrôle. Une autre façon d'exprimer cet algorithme est la suivante :

Réduisez votre vitesse au fur et à mesure que l'erreur entre la distance de référence et la distance mesurée diminue.

Algorithme 6.3 est l'*algorithme de contrôle proportionnel* ou un *contrôleur P*.

Exemple Supposons que la distance de référence soit de 100 cm et que le gain soit de -0.8. Lorsque le robot se trouve à 150 cm de l'objet, l'erreur est de $100 - 150 = -50$ et l'algorithme de contrôle fixera la puissance à $-0.8 \cdot -50 = 40$. Le tableau 6.1

Algorithm 6.3: Contrôleur proportionnel	
integer reference	$\leftarrow \dots$ // Reference distance
integer measured	// Measured distance
integer error	// Error
float gain	$\leftarrow \dots$ // Proportional gain
integer power	// Motor power
1: error	\leftarrow reference – measured // Distances
2: power	\leftarrow gain * error // Control value
3: left-motor-power	\leftarrow power
4: right-motor-power	\leftarrow power

présente les erreurs et les réglages de puissance pour trois distances. Si le robot dépasse la distance de référence de 100 cm et qu'une distance de 60 cm est mesurée, la puissance sera réglée sur -32, ce qui entraînera un recul du robot.

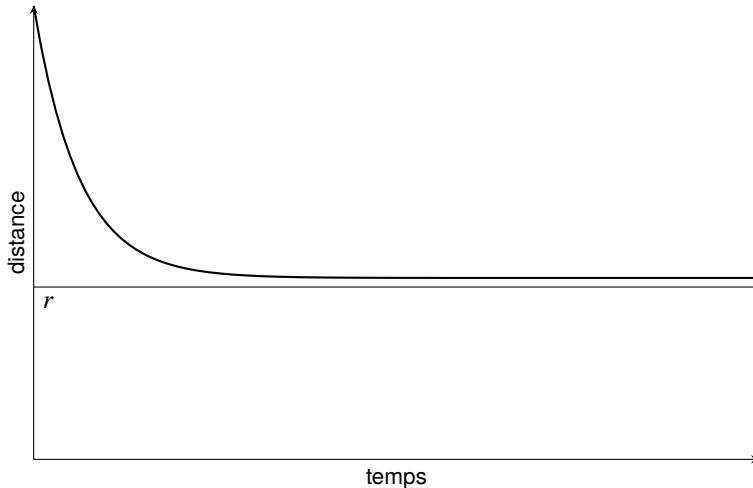
La figure 6.3 représente la distance du robot à l'objet en fonction du temps lorsque le robot est contrôlé par un contrôleur P. La ligne étiquetée r représente la distance de référence. La variation de la puissance du moteur est régulière et le robot ne subit pas d'accélérations et de décélérations rapides. La réponse est un peu lente, mais le robot se rapproche de la distance cible.

Malheureusement, le robot n'atteint pas réellement la distance de référence. Pour comprendre pourquoi cela se produit, considérez ce qui se passe lorsque le robot est très proche de la distance de référence. L'erreur sera très faible et, par conséquent, le réglage de la puissance sera très bas. En théorie, le faible réglage de la puissance doit permettre au robot de se déplacer lentement, pour finalement atteindre la distance de référence. En pratique, la puissance du moteur peut devenir si faible qu'elle ne parvient pas à surmonter la friction interne des moteurs et de leur connexion aux roues, et le robot s'arrête alors de bouger.

Il pourrait sembler que l'augmentation du gain du contrôleur P pourrait surmonter ce problème, mais un gain élevé souffre d'un sérieux inconvénient. La figure 6.4 montre l'effet du gain du contrôleur P. Un gain plus élevé (ligne rouge en pointillés)

TAB. 6.1 – Contrôleur proportionnel pour un gain de -0.8

distance	erreur	puissance
150	-50	40
125	-25	20
60	40	-32

FIG. 6.3 – Comportement du contrôleur P

permet au robot de se rapprocher plus rapidement de la distance de référence, tandis qu'un gain plus faible (ligne bleue en pointillés) permet au robot de se rapprocher plus lentement de la distance de référence. Cependant, si le gain est trop élevé, le contrôleur P fonctionne comme un contrôleur tout ou rien avec une réponse oscillante (ligne verte). Nous disons que le contrôleur est *instable*.

Il existe des situations où le contrôleur P ne peut pas atteindre la distance de référence, même dans un système idéal. Supposons que l'objet lui-même s'éloigne du robot à vitesse constante. Le contrôleur P règle la puissance maximale du moteur pour que le robot se déplace rapidement vers l'objet. Cependant, à mesure que le robot s'approche de l'objet, la distance mesurée devient faible et le contrôleur P règle la puissance à un niveau si bas que la vitesse du robot est inférieure à celle de l'objet. En conséquence, le robot n'atteindra jamais la distance de référence. Si le robot pouvait effectivement atteindre la distance de référence, l'erreur serait nulle et la vitesse du robot serait donc également nulle. Cependant, l'objet continue de s'éloigner du robot, de sorte que, un peu plus tard, le robot recommence à se déplacer et le cycle se répète. Ce mouvement de va-et-vient n'est pas l'objectif visé par le maintien de la distance de référence.

Exemple Nous utilisons les mêmes données que dans l'exemple précédent, sauf que l'objet se déplace à 20 cm/s. Le tableau 6.2 montre les erreurs et les réglages de puissance pour trois distances. Au départ, le robot va plus vite que l'objet, il va donc le rattraper. Cependant, à 125 cm de l'objet, le robot se déplace à la même vitesse que l'objet. Il maintient cette distance fixe et ne se rapprochera pas de la distance de référence de 100 cm. Si, d'une manière ou d'une autre, le robot se rapproche de

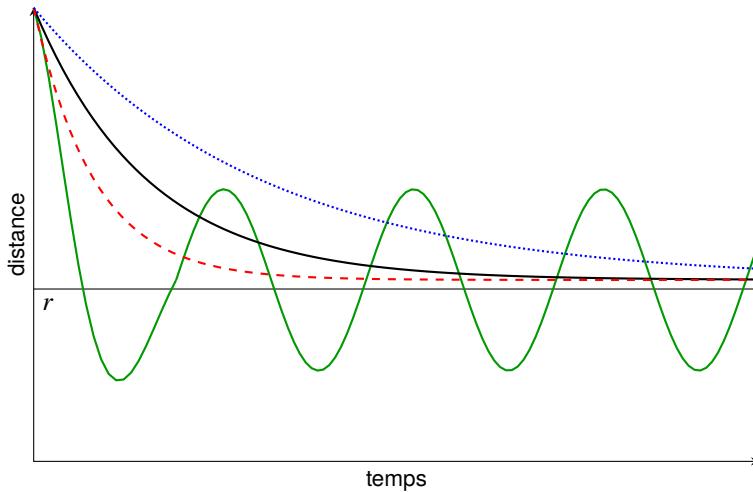


FIG. 6.4 – L’effet du gain sur le régulateur P : gain faible (ligne bleue pointillée), gain élevé (ligne rouge pointillée), gain excessif (ligne verte oscillante)

l’objet, disons à 110 cm, la puissance est réduite à 8, ce qui fait que le robot s’éloigne de l’objet.

En général, le robot se stabilise à une distance fixe de la distance de référence. Vous pouvez réduire cette erreur en augmentant le gain, mais la distance de référence ne sera jamais atteinte et le seul résultat est que le contrôleur devient instable.

Activity 6.3: Régulateur proportionnel

- Mettez en œuvre l’algorithme de contrôle proportionnel pour que le robot s’arrête à une distance donnée d’un objet. Avec quelle précision pouvez-vous atteindre l’objectif lorsque l’objet ne bouge pas ?
- Que se passe-t-il si l’objet bouge ? Pour l’objet, vous pouvez utiliser un second robot programmé pour se déplacer à une vitesse fixe.
- Expérimentez avec le gain et la période pour voir comment ils affectent les performances de l’algorithme.

TAB. 6.2 – Contrôleur proportionnel pour un objet en mouvement et un gain de -0.8

distance	erreur	puissance
150	-50	40
125	-25	20
110	-10	8

Algorithm 6.4: Contrôleur proportionnel-intégral	
integer reference ← ...	// Reference distance
integer measured	// Measured distance
integer error	// Error
integer error-sum ← 0	// Cumulative error
float gain-p ← ...	// Proportional gain
float gain-i ← ...	// Integral gain
integer power	// Motor power
1: error ← reference – measured	// Distances
2: error-sum ← error-sum + error	// Integral term
3: power ← gain-p * error + gain-i * error-sum	// Control value
4: left-motor-power ← power	
5: right-motor-power ← power	

6.4 Contrôleur proportionnel-intégral (PI)

Un *contrôleur proportionnel-intégral* peut atteindre la distance de référence même en présence de friction ou d'un objet en mouvement en prenant en compte l'erreur accumulée au fil du temps. Alors que le contrôleur P ne prend en compte que l'erreur courante :

$$u(t) = k_p e(t),$$

le contrôleur PI ajoute l'intégrale de l'erreur depuis le moment où l'algorithme commence à fonctionner jusqu'au moment présent :

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau) d\tau.$$

Des facteurs de gain distincts sont utilisés pour les termes proportionnels et intégraux afin de permettre une certaine souplesse dans la conception du contrôleur.

Lors de la mise en œuvre d'un régulateur PI, une approximation discrète de l'intégrale continue est effectuée (Algorithme 6.4).

En présence d'un frottement ou d'un objet en mouvement, l'erreur sera intégrée et entraînera le réglage d'une puissance moteur plus élevée, ce qui permettra au robot de converger vers la distance de référence. Un problème avec un contrôleur PI est que l'intégration de l'erreur commence à l'état initial lorsque le robot est éloigné de

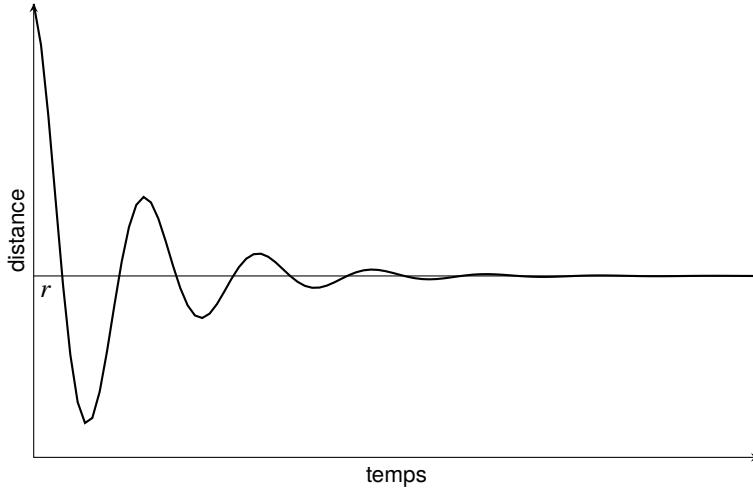


FIG. 6.5 – Comportement du contrôleur PI

l'objet. Lorsque le robot s'approche de la distance de référence, le terme intégral du régulateur aura déjà une valeur importante ; pour diminuer cette valeur, le robot doit se déplacer au-delà de la distance de référence de manière à obtenir des erreurs de signe opposé. Cela peut générer des oscillations (Fig. 6.5).

Activity 6.4: Contrôleur PI

- Implémentez un contrôleur PI qui oblige le robot à s'arrêter à une distance donnée d'un objet.
- Comparez le comportement du contrôleur PI à celui d'un contrôleur P pour la même tâche en surveillant les variables des algorithmes de contrôle dans le temps.
- Que se passe-t-il si vous empêchez manuellement le robot de bouger pendant un court instant, puis que vous le laissez partir ? Cela illustre un concept appelé *integrator windup*. Explorez ce concept en consultant des sources en ligne et trouvez une méthode pour résoudre le problème.

6.5 Contrôleur proportionnel-intégral-dérivé (PID)

Lorsque vous lancez ou bottez un ballon vers un autre joueur qui se déplace, vous ne le lancez pas vers sa position actuelle. Le temps que le ballon l'atteigne, il se sera déplacé vers une nouvelle position. Au lieu de cela, vous estimatez l'endroit où se trouvera la nouvelle position et vous y dirigez la balle. De même, un robot

dont la tâche consiste à pousser un colis sur un chariot en mouvement doit synchroniser sa poussée avec la position future estimée du chariot lorsque le colis l'atteint. L'algorithme de commande de ce robot ne peut pas être un contrôleur tout-ou-rien, P ou PI, car ils ne prennent en compte que la valeur actuelle de l'erreur (et pour le contrôleur PI, les valeurs précédentes).

Pour estimer l'erreur future, le taux de variation de l'erreur peut être pris en compte. Si le taux de variation de l'erreur est faible, le robot peut pousser le colis juste avant que le chariot ne s'en approche, tandis que si le taux de variation de l'erreur est élevé, le colis doit être poussé beaucoup plus tôt.

Mathématiquement, le taux de variation est exprimé sous la forme d'une dérivée. Un régulateur *proportionnel-intégral-dérivé* (*PID*) ajoute un terme supplémentaire aux termes P et I :

$$u(t) = k_p e(t) + k_i \int_{\tau=0}^t e(\tau) d\tau + k_d \frac{de(t)}{dt}. \quad (6.1)$$

Dans la mise en œuvre d'un contrôleur PID, le différentiel est approximé par la différence entre l'erreur précédente et l'erreur actuelle (Algorithme 6.5).

Le comportement du contrôleur PID est illustré sur la Fig.6.6. Le robot converge rapidement et en douceur vers la distance de référence.

Les gains d'un régulateur PID doivent être soigneusement équilibrés. Si les gains des termes P et I sont trop élevés, des oscillations peuvent se produire. Si le gain du terme D est trop élevé, le régulateur réagira à de courtes salves de bruit.

Activity 6.5: Contrôleur PID

- Implémentez un contrôleur PID pour la tâche d'un robot s'approchant d'un objet.
- Expérimenez avec différents gains jusqu'à ce que le robot se rapproche en douceur de la distance de référence.
- Répétez les expériences avec un objet en mouvement.

6.6 Résumé

Un bon algorithme de contrôle doit converger rapidement vers le résultat souhaité tout en évitant les mouvements brusques. Il doit être efficace en termes de calcul, mais ne pas nécessiter de réglage constant. L'algorithme de contrôle doit être adapté aux exigences spécifiques du système et de la tâche, et fonctionner correctement dans différentes conditions environnementales. Nous avons décrit quatre algorithmes, de l'impraticable algorithme tout ou rien à des algorithmes combinant des termes proportionnels, intégraux et dérivés. Le terme proportionnel garantit que les erreurs importantes entraînent une convergence rapide vers la référence, le terme

Algorithm 6.5: Contrôleur proportionnel-intégral-différentiel

```

integer reference ← ...           // Reference distance
integer measured                 // Measured distance
integer error                     // Error
integer error-sum ← 0            // Cumulative error
integer previous-error ← 0       // Previous error
integer error-diff               // Error difference
float gain-p ← ...              // Proportional gain
float gain-i ← ...              // Integral gain
float gain-d ← ...              // Derivative gain
integer power                     // Motor power

1: error ← reference – measured      // Distances
2: error-sum ← error-sum + error     // Integral term
3: error-diff ← error – previous-error // Differential term
4: previous-error ← error           // Save current error
5: power ← gain-p * error +         // Control value
   gain-i * error-sum + gain-d * error-diff
6: left-motor-power ← power
7: right-motor-power ← power

```

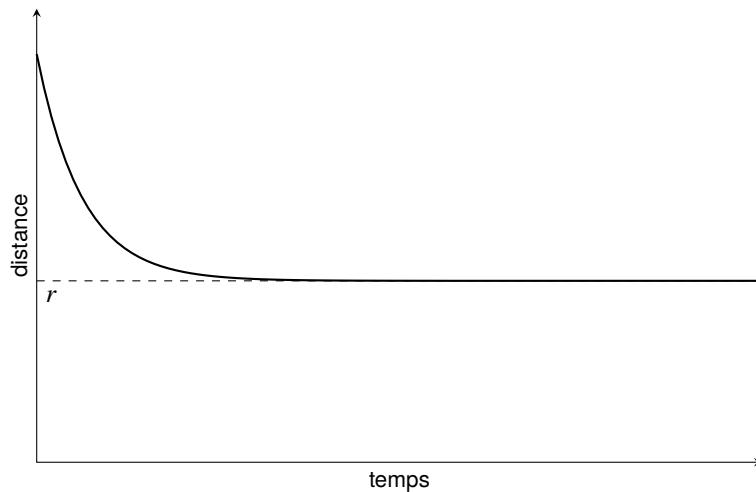


FIG. 6.6 – Comportement du contrôleur PID

integral garantit que la référence peut effectivement être atteinte, tandis que le terme dérivé rend l'algorithme plus réactif.

6.7 Lectures complémentaires

Un manuel moderne sur les algorithmes de contrôle est [1].

Chapitre 7

Navigation locale : Évitement d'obstacles

Un robot mobile doit naviguer d'un point à un autre de son environnement. Il peut s'agir d'une tâche simple, par exemple si un robot peut suivre une ligne sans obstacle sur le sol d'un entrepôt (3.4), mais la tâche devient plus difficile dans des environnements inconnus et complexes comme un rover explorant la surface de Mars ou un submersible explorant une chaîne de montagnes sous-marine. Même une voiture autonome qui circule sur une route doit faire face aux autres voitures, aux obstacles sur la route, aux passages pour piétons, à la construction des routes, etc.

La navigation d'une voiture autonome peut être divisée en deux tâches : La première, de haut niveau, consiste à trouver un chemin entre une position de départ et une position d'arrivée. Avant le développement des systèmes informatiques modernes, pour trouver un chemin, il fallait étudier une carte ou demander son chemin. Aujourd'hui, il existe des applications pour smartphones qui prennent une position de départ et une position d'arrivée et calculent des chemins entre ces deux positions. Si l'application reçoit des données en temps réel sur les conditions de circulation, elle peut suggérer le chemin qui vous permettra d'atteindre votre objectif le plus rapidement possible. Le chemin peut être calculé hors ligne ou, si vous disposez d'un système GPS capable de déterminer votre position actuelle, le chemin peut être trouvé en temps réel et mis à jour pour tenir compte de l'évolution des conditions.

Une voiture autonome doit également adapter son comportement à l'environnement : s'arrêter pour laisser passer un piéton sur un passage piéton, tourner aux intersections, éviter les obstacles sur la route, etc. Alors que la recherche de chemin de haut niveau peut être effectuée une fois avant le voyage (ou toutes les quelques minutes), la tâche de bas niveau consistant à éviter les obstacles doit être exécutée fréquemment, car la voiture ne sait jamais quand un piéton va sauter sur la route ou quand la voiture qu'elle suit va soudainement freiner.

La section 7.1 examine la tâche de bas niveau d'évitement des obstacles. La section 7.2 montre comment un robot peut reconnaître des marquages tout en suivant une ligne afin de savoir quand il a atteint son objectif. Les sections 7.3–7.5 démontrent un comportement de plus haut niveau : trouver un chemin sans carte de l'environnement. Cela se fait par analogie avec une colonie de fourmis localisant une source de nourriture et communiquant son emplacement à tous les membres de la colonie.

7.1 Evitement des obstacles

Les algorithmes présentés jusqu'à présent se sont concentrés sur la détection d'objets et le déplacement vers ceux-ci. Lorsqu'un robot se déplace vers un objectif, il est susceptible de rencontrer des objets supplémentaires appelés *obstacles* qui bloquent le chemin et empêchent le robot d'atteindre son objectif. Nous supposons que le robot est capable de détecter s'il existe un chemin libre vers l'objectif, par exemple en détectant une lumière sur l'objectif. Cette section décrit trois algorithmes d'évitement d'obstacles, où les obstacles sont des murs qui bloquent le mouvement du robot :

- Un algorithme simple de suivi de mur, qui malheureusement ne fonctionnera pas s'il y a plusieurs obstacles dans l'environnement.
- Un algorithme qui peut éviter plusieurs obstacles, mais qui doit connaître la direction générale du but (peut-être grâce à son système GPS). Malheureusement, certains obstacles peuvent bloquer le robot dans une boucle.
- L'algorithme Pledge est une petite modification du second qui permet de surmonter ce comportement erroné.

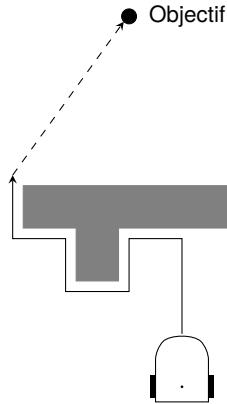
Les algorithmes utilisent les expressions conditionnelles abstraites mur devant et mur à droite, qui sont vraies si un mur se trouve à l'avant ou à la droite du robot. Le premier algorithme utilisera également l'expression conditionnelle corner-right qui est vraie si le robot se déplace autour d'un obstacle et détecte un coin à sa droite. Il existe plusieurs façons d'implémenter ces expressions, que nous étudions dans Activity 7.1.

Activity 7.1: Expressions conditionnelles pour suivre un mur

- Mettre en œuvre l'expression conditionnelle suiv du mur à l'aide d'un capteur de proximité horizontal ou d'un capteur tactile.
- Mettre en œuvre l'expression conditionnelle wall-right. Ceci est facile à réaliser avec un capteur monté à droite du robot, ou avec un capteur de distance rotatif. Si vous n'avez qu'un capteur de proximité orienté vers l'avant, vous pouvez demander au robot de se tourner légèrement vers la droite, de détecter le mur, s'il y en a un, puis de faire demi-tour.
- Mettre en œuvre l'expression conditionnelle coin-droit. Elle peut être mise en œuvre comme une extension de mur-droit. Lorsque la valeur de wall-right passe de vrai à faux, effectuer un court virage à droite et vérifier si wall-right redévient vrai.

7.1.1 suivi du mur

La figure 7.1 montre un robot qui suit un mur en maintenant sa position de manière à ce que le mur soit à sa droite (Algorithme 7.1). Si un mur est détecté

FIG. 7.1 – *Suivi du mur*

devant lui, le robot tourne à gauche de manière à ce que le mur soit à sa droite. Si un mur est détecté à droite, le robot continue à se déplacer le long du mur. Si un coin est détecté, le robot tourne à droite pour continuer à contourner l'obstacle. En même temps, le robot cherche continuellement le but (point noir). Lorsqu'il détecte l'objectif, le robot se déplace directement vers lui.

Malheureusement, l'algorithme 7.1 ne fonctionne pas toujours correctement. La figure 7.2 montre une configuration avec *deux* obstacles entre le robot et le but. Le robot ne détectera jamais l'objectif et se déplacera donc indéfiniment autour du

Algorithm 7.1: Simple suivi du mur

```

1: while not-at-goal
2:   if goal-detected
3:     move towards goal
4:   else if wall-ahead
5:     turn left
6:   else if corner-right
7:     turn right
8:   else if wall-right
9:     move forward
10:  else
11:    move forward

```

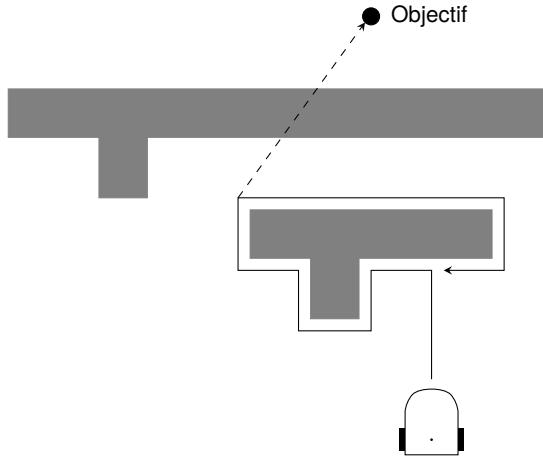


FIG. 7.2 – *Le simple suivi du mur ne permet pas toujours au robot d'atteindre l'objectif*

premier obstacle.

Activity 7.2: Simple suivi de mur

- Mettre en œuvre l'algorithme 7.1 et vérifier qu'il démontre les comportements montrés dans les Figs.7.1, 7.2.7.2.

7.1.2 Suivi de mur avec direction

Le problème de l'algorithme 7.1 est qu'il s'agit d'un algorithme local qui ne regarde que son environnement immédiat et ne tient pas compte du fait que l'algorithme de navigation de niveau supérieur connaît approximativement la direction que le robot doit prendre pour atteindre l'objectif. La figure 7.3 montre le comportement d'un robot qui "sait" que le but se trouve quelque part au nord et qui se déplace donc à un cap de 0° par rapport au nord. L'algorithme de suivi de mur n'est utilisé que si le robot ne peut pas se déplacer vers le nord.

L'algorithme 7.2 est similaire à l'algorithme précédent, à l'exception du fait qu'il préfère se déplacer vers le nord si possible. Il utilise une variable `heading` pour se souvenir de son cap actuel lorsqu'il se déplace autour de l'obstacle. Lorsque `heading` est à nouveau au nord (un multiple de 360°), le robot avance au lieu de chercher un coin.

Malheureusement, l'algorithme peut échouer lorsqu'il est confronté à un obstacle en forme de 7.4). Après avoir effectué quatre virages à gauche, il se dirige vers 360°

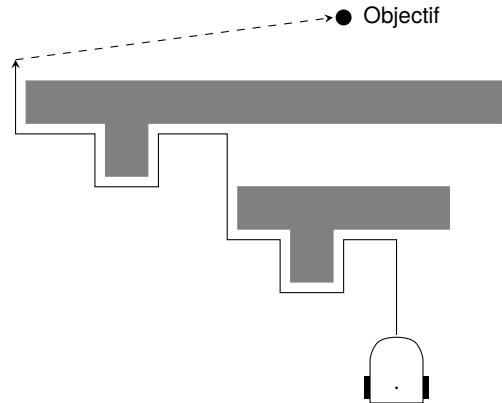


FIG. 7.3 – Mur qui suit la direction

Algorithm 7.2: Suivi des murs

```
integer heading ← 0°
```

```

1: while not-at-goal
2:   if goal-detected
3:     move towards goal
4:   else if wall-ahead
5:     turn left
6:     heading ← heading + 90°
7:   else if corner-right
8:     if heading = multiple of 360°
9:       move forward
10:    else
11:      turn right
12:      heading ← heading - 90°
13:    else if wall-right
14:      move forward
15:    else
16:      move forward

```

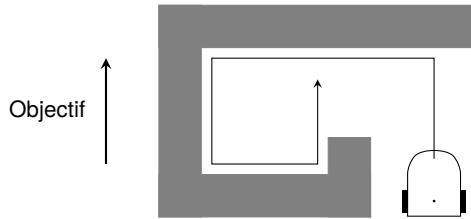


FIG. 7.4 – Pourquoi le suivi d'un mur avec direction ne fonctionne pas toujours

(également vers le nord, un multiple de 360°) et continue d'avancer, rencontrant et suivant le mur encore et encore.

Activity 7.3: Suivi du mur avec direction

- Mettre en oeuvre l'algorithme de suivi de mur avec direction et vérifier qu'il démontre le comportement montré dans la Fig. 7.4.
- Exécuter l'algorithme de suivi de mur simple (Algorithm 7.1) avec un obstacle en forme de G. Que se passe-t-il? Cela affecte-t-il notre affirmation selon laquelle cet algorithme n'est pas adapté à l'évitement d'obstacles?

7.1.3 L'algorithme de l'engagement

L'algorithme de gage modifie la ligne 8 de l'algorithme de suivi de mur en :

```
if heading = 0°
```

Le robot n'avance que lorsque son cap cumulé est égal à 0° et non lorsqu'il se déplace vers le nord–un cap qui est un multiple de 360° . Le robot évite maintenant l'obstacle en forme de "G" (Fig. 7.5) : lorsqu'il rencontre le coin (point noir), il se déplace vers le nord, mais son cap est de 360° après quatre virages à gauche. Bien que 360° soit un multiple de 360° , il n'est pas égal à 0° . Par conséquent, le robot continue à suivre le mur jusqu'à ce que quatre virages à droite soustraient 360° , de sorte que le cap total est de 0° .

Activity 7.4: Algorithme de Pledge

- Mettre en œuvre l'algorithme de promesse et vérifier qu'il présente le comportement illustré dans la Fig. 7.5.

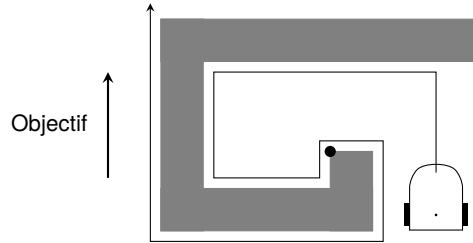


FIG. 7.5 – Algorithme de promesse de suivi du mur

7.2 Suivre une ligne avec un code

Revenons à la tâche qui consiste à trouver un chemin vers un but. Si le chemin est marqué par une ligne sur le sol, les algorithmes de suivi de ligne (Sect. 3.4) peuvent guider un robot dans l'environnement, mais le suivi de ligne n'est pas de la navigation. Pour naviguer d'une position à l'autre, nous avons également besoin d'un algorithme de localisation afin que le robot sache quand il a atteint ses objectifs. Nous n'avons pas besoin d'un algorithme de localisation continue comme ceux du chapitre 8, nous avons seulement besoin de connaître les positions sur la ligne qui facilitent l'accomplissement de la tâche. Cela ressemble à la navigation en voiture : il suffit de connaître les échangeurs, les intersections, les principaux points de repère, etc. pour savoir où l'on se trouve. Entre ces positions, il suffit de suivre la route.

La navigation sans localisation continue peut être mise en œuvre en lisant un code placé sur le sol à côté de la ligne. La figure 7.6 montre un robot équipé de deux capteurs au sol : celui de gauche détecte la ligne et celui de droite le code. Sous le robot se trouve un graphique du signal renvoyé par le capteur de droite.

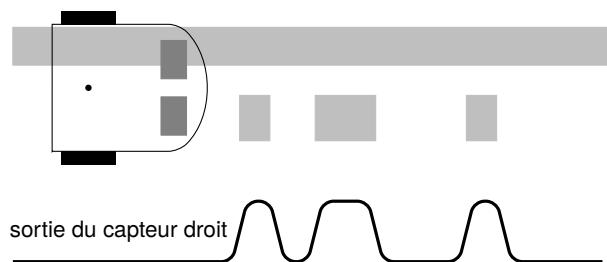


FIG. 7.6 – Un robot suit une ligne avec son capteur gauche et lit un code avec son capteur droit

Activity 7.5: Suivre une ligne tout en lisant un code

- Mettre en oeuvre le suivi de ligne avec la lecture de code comme indiqué sur la Fig. 7.6.
- Ecrire un programme qui permet à un robot de suivre un chemin.
- Placer des marques qui codent des valeurs à côté du chemin. Le robot doit afficher ces valeurs (à l'aide de la lumière, du son ou d'un écran) lorsqu'il se déplace sur les codes.

Activity 7.6: Suivi de ligne circulaire lors de la lecture d'un code

- Mettre en place une horloge en utilisant deux robots, l'un pour les minutes et l'autre pour les heures (Fig. 7.7).
- Une autre solution consisterait à faire en sorte que les deux robots se déplacent à des vitesses différentes, de sorte que l'un d'eux effectue une révolution en une heure et l'autre en un jour. Discutez de la différence entre ces deux implémentations.

7.3 Fourmis à la recherche d'une source de nourriture

Revenons maintenant à l'algorithme de haut niveau de recherche d'un chemin. S'il existe une ligne et un mécanisme de localisation comme un code, l'approche de la section précédente peut être utilisée. Cependant, même si une ligne n'existe pas, un robot peut être capable de créer sa propre ligne. L'aspect intéressant de cette méthode est que le robot n'a pas besoin de connaître sa position dans l'environnement, par exemple à l'aide d'un GPS ; au lieu de cela, il utilise des points de repère dans l'environnement lui-même pour la navigation. L'algorithme sera présenté dans le contexte réel des fourmis à la recherche de nourriture :

Il existe un nid de fourmis. Les fourmis cherchent au hasard une source de nourriture. Lorsqu'une fourmi trouve de la nourriture, elle retourne directement au nid en utilisant des points de repère et sa mémoire du chemin qu'elle a emprunté depuis le nid. Pendant le trajet de retour au nid avec la nourriture, la fourmi dépose des substances chimiques appelées *phéromones*. Au fur et à mesure que les fourmis trouvent la source de nourriture et retournent au nid, la piste accumule plus de phéromones que les autres zones visitées par les fourmis. Finalement, la quantité de phéromones sur la piste est si importante que les fourmis peuvent suivre un chemin direct du nid à la source de nourriture.

La figure 7.8 montre le nid des fourmis dans le coin inférieur gauche représenté par une lumière qui permet aux fourmis de retrouver facilement leur chemin vers

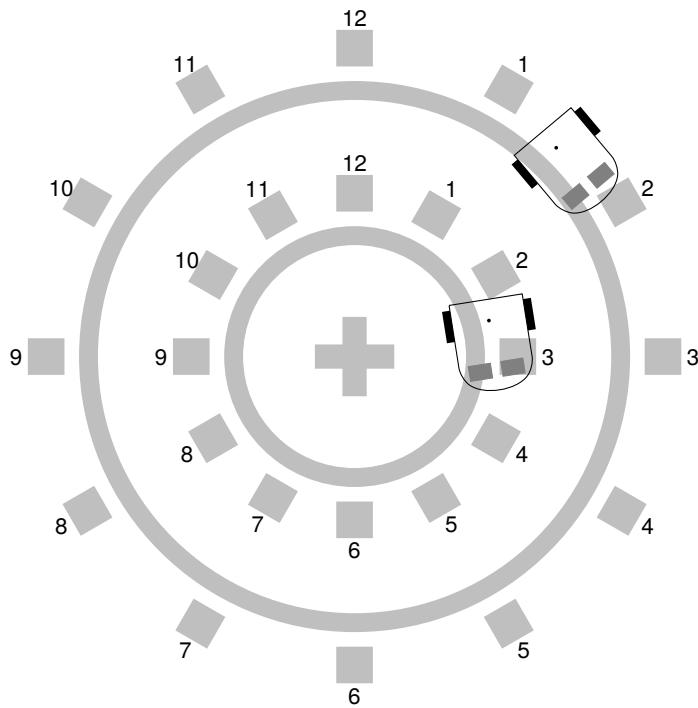


FIG. 7.7 – Une horloge robotisée : un robot indique l'heure et l'autre les minutes.

le nid. Le point sombre est la source de nourriture. La figure 7.9 montre trois pistes aléatoires qui finissent par découvrir la source de nourriture ; les fourmis retournent alors directement au nid, laissant trois lignes droites de phéromones. Cette concentration peut être utilisée ultérieurement pour trouver directement la source de nourriture.

Le comportement de la fourmi peut être mis en œuvre par un robot. Supposons qu'il existe une zone fixe à l'intérieur de laquelle le robot peut se déplacer. Comme dans la Fig. 7.8, il y a une source de nourriture et un nid. La source de nourriture est représentée par un point sombre qui peut être facilement détecté par un capteur au sol du robot. Les capteurs de proximité du robot sont utilisés pour détecter les murs de la zone. 7.7 propose deux méthodes de représentation du nid qui dépendent des capteurs supplémentaires dont dispose votre robot.

Activity 7.7: Localiser le nid

- Mettre en œuvre un programme qui oblige le robot à se déplacer vers le nid, quel que soit l'endroit où il est placé dans la zone.

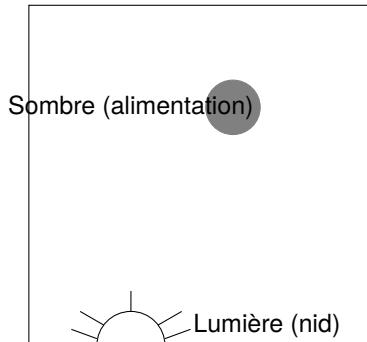


FIG. 7.8 – *Le nid de fourmis et la source de nourriture*

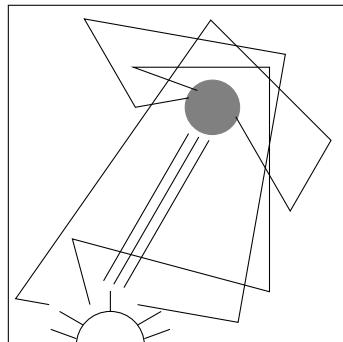


FIG. 7.9 – *Les phéromones créent une piste*

- Accéléromètres : Monter la zone sur une pente de telle sorte qu'un coin, le nid, se trouve au point le plus bas de la zone.
- Capteur de lumière : Le nid est représenté par une source de lumière qui peut être détectée par le capteur de lumière indépendamment de la position et de la direction du robot. Si le capteur de lumière est fixe et ne peut détecter la lumière que dans une certaine direction, le robot devra tourner pour localiser la source de lumière.

Simulez les phéromones en recouvrant la zone d'une feuille de papier blanc et en attachant un marqueur noir au robot de manière à ce qu'il trace une ligne partout où il se déplace. Un capteur au sol détecte les marques dans la zone. La figure 7.10 montre les lignes résultant du comportement d'un robot exécutant l'algorithme. L'activité 7.8 vous demande d'explorer la capacité du robot à détecter les zones présentant une forte densité de lignes.

Activity 7.8: Détection des zones de haute densité

- Dans la section 3.4.3, nous avons noté que les capteurs ne détectent pas un seul point géométrique mais ont plutôt une ouverture qui lit une zone relativement grande, peut-être même jusqu'à un centimètre carré (3.7). Faites des expériences avec votre capteur au sol pour voir comment les lectures renvoyées par le capteur dépendent de la largeur de la ligne. Pouvez-vous tirer une conclusion sur la largeur optimale du marqueur ? S'il est trop fin, la piste ne sera pas détectée et s'il est trop épais, les marques du mouvement aléatoire pourraient être prises pour la piste.
- Représentez la source de nourriture comme une tache relativement

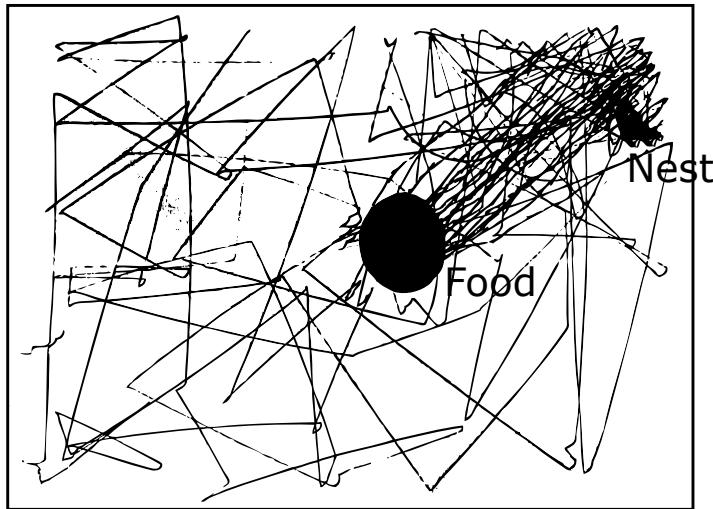


FIG. 7.10 – Un robot simulant les phéromones des fourmis

grande et totalement noire et assurez-vous qu'elle donne une lecture minimale du capteur au sol.

- La figure 7.10 montre que la piste entre la source de nourriture et le nid a une densité élevée. Expérimitez avec différents nombres de lignes et définissez un seuil efficace entre la piste et les zones de mouvement aléatoire à l'extérieur de la piste. Voyez si vous pouvez faire en sorte que le robot trace des lignes plus foncées en variant son mouvement ou en se déplaçant d'avant en arrière le long de la piste.

7.4 Un modèle probabiliste du comportement des fourmis

Un *modèle* est une abstraction d'un système qui montre comment les paramètres influencent les phénomènes. Les modèles sont utilisés, par exemple, pour étudier les schémas de circulation afin de prédire l'effet de nouvelles routes ou de nouveaux feux de circulation. Pour comprendre comment le chemin entre le nid et la nourriture est généré, cette section présente un modèle simplifié du comportement des fourmis.

La caractéristique fondamentale du comportement des fourmis est qu'elles ne disposent pas d'une carte de leur environnement et qu'elles doivent donc se déplacer au hasard pour rechercher la source de nourriture. Par conséquent, un modèle de leur comportement doit être probabiliste. Supposons que l'environnement soit une zone rectangulaire constituée d'une grille de cellules. La figure 7.11 montre une zone

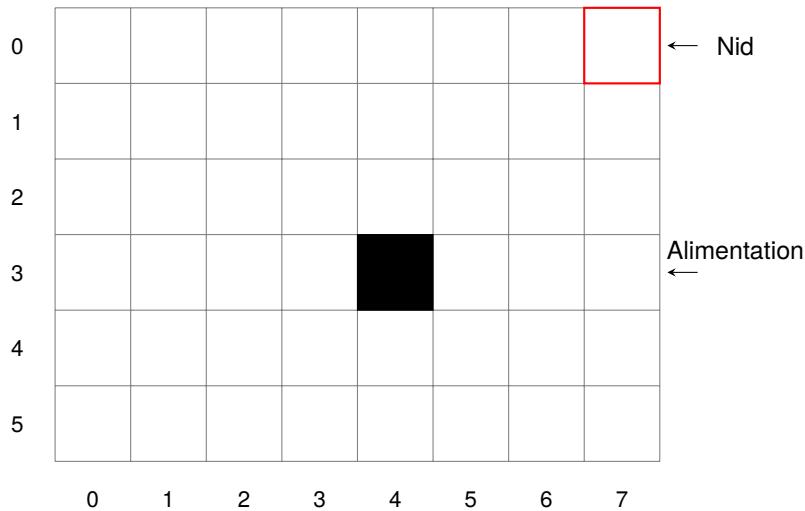


FIG. 7.11 – Représentation de l'environnement sous forme de grille de cellules

divisée en $6 \times 8 = 48$ cellules.

Coordonnées dans une grille de cellules

Tout au long du livre, les coordonnées d'une cellule dans une grille sont données sous la forme (*row, column*). Les lignes sont numérotées de haut en bas et les colonnes de gauche à droite, comme les matrices en mathématiques, mais la numérotation commence à partir de 0, comme dans le type de données array en informatique.

Sans aucune information sur la façon dont les fourmis choisissent leurs mouvements, nous supposons qu'elles peuvent se déplacer dans n'importe quelle direction avec la même probabilité. La probabilité p qu'une fourmi se trouve dans une cellule est donc égale à 1 divisé par le nombre de cellules, soit ici $p = 1/48 = 0,021$.

La probabilité que la fourmi se trouve dans la cellule où se trouve la source de nourriture est p , la même que pour toute autre cellule. Selon notre spécification du comportement de la fourmi, une fois qu'elle est entrée dans cette cellule et qu'elle a identifié la cellule comme étant la source de nourriture, elle retourne directement au nid. Dans la figure 7.11, la source de nourriture se trouve dans la cellule (3,4), de sorte qu'une fourmi visitant cette cellule doit retourner au nid à la cellule (0,7), en passant par les cellules (2,5) et (1,6). Quelle est la probabilité que la fourmi se trouve dans l'une de ces trois cellules ? Il y a deux possibilités : soit la fourmi est dans la cellule parce qu'elle s'y est déplacée au hasard avec une probabilité p , soit la fourmi s'y trouve parce qu'elle s'est déplacée vers la source de nourriture au hasard avec une probabilité p et ensuite avec une probabilité 1 s'est déplacée vers le nid.

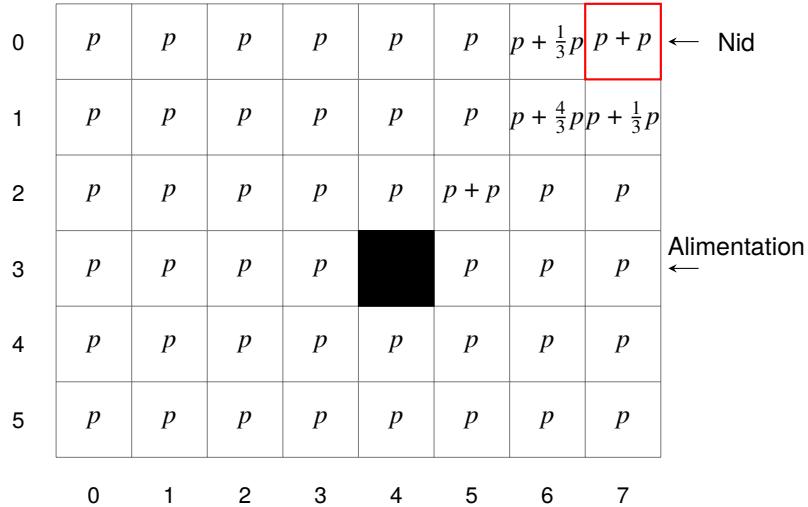


FIG. 7.12 – Probabilités de localisation de la fourmi

Par conséquent, la probabilité totale de se trouver dans l'une de ces cellules est de $p + p \times 1 = p + p = 2p$.¹ Si notre robot dessine des lignes en se déplaçant, les cellules situées sur la diagonale doivent être deux fois plus sombres que les autres cellules.

Une fois que la fourmi a atteint le nid, elle se déplace à nouveau au hasard, c'est-à-dire qu'elle choisit un voisin au hasard. En général, une cellule a huit voisins (au-dessus et au-dessous, à gauche et à droite, quatre sur les diagonales), de sorte que la probabilité est $p/8$ qu'elle se trouve dans l'un de ces voisins. Le nid, cependant, se trouve dans le coin et n'a que trois voisins, la probabilité est donc de $p/3$ qu'il se déplace vers l'un d'entre eux. La figure 7.12 montre la probabilité de l'emplacement de la fourmi après avoir trouvé la source de nourriture, être retournée au nid et avoir effectué un déplacement aléatoire supplémentaire. Lorsqu'elles sont mises en œuvre par un robot muni d'un marqueur, les cellules présentant une probabilité plus élevée deviennent plus foncées (figure 7.13).

Que pouvons-nous conclure de ce modèle ?

- Bien que les fourmis se déplacent au hasard, leur comportement de retour au nid après avoir trouvé la source de nourriture fait que la probabilité de se trouver sur la diagonale est plus élevée que partout ailleurs dans l'environnement.
- Puisque les fourmis déposent des phéromones (marques noires) dans chaque

1. Après la mise à jour des probabilités, celles-ci doivent être normalisées comme expliqué dans l'annexe B.2. Pour un autre exemple de normalisation, voir la section 8.4.

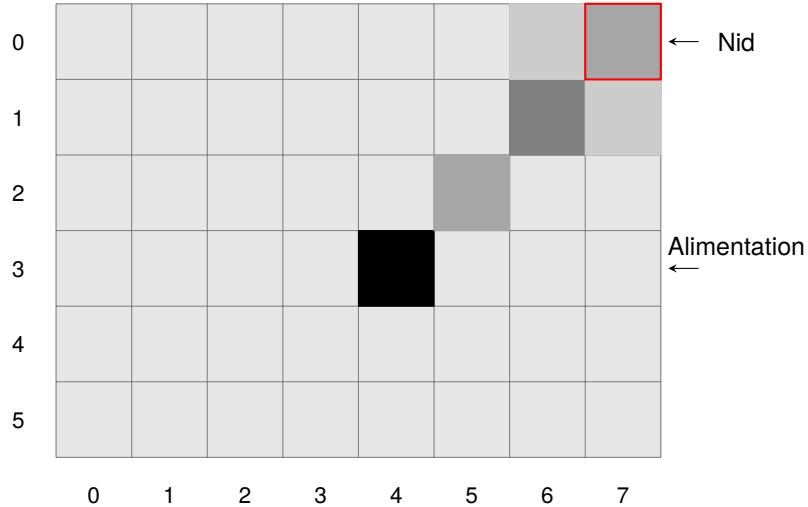


FIG. 7.13 – Probabilités de localisation d'un robot avec un marqueur

cellule qu'elles visitent, il s'ensuit que les marques sur le chemin diagonal entre la source de nourriture et le nid seront plus foncées que les marques sur les autres cellules. Finalement, les marques sur ce chemin seront suffisamment sombres pour que le robot puisse le suivre jusqu'à la source de nourriture sans effectuer d'exploration aléatoire.

- Puisque le robot visite souvent le nid, les cellules dans le voisinage immédiat du nid auront une probabilité quelque part entre la probabilité uniforme et la probabilité élevée du sentier. Il est donc important de mettre l'accent sur la piste en utilisant des méthodes telles que celles explorées dans l'activité 7.8.

7.5 Une machine à états finis pour l'algorithme de recherche de chemin

Une FSM pour la recherche de chemin par les fourmis est représentée sur la Fig. 7.14. Pour gagner de la place, les étiquettes des transitions utilisent des abréviations qui sont expliquées dans le tableau 7.1. Voici une description détaillée du comportement spécifié par ce FSM dans chaque état :

search : Dans cet état, le robot recherche au hasard les zones sombres. Il s'agit de l'état initial et la transition vrai \leadsto fwd spécifie qu'initiallement (et inconditionnellement) le robot se déplace vers l'avant et qu'une minuterie est réglée sur une période aléatoire. Lorsque le délai expire (timeout), le robot effectue un virage aléatoire, se déplace vers l'avant et réinitialise le délai. Ce mouvement aléatoire se poursuit jusqu'à ce que le robot rencontre le mur de la zone ou une marque grise à la surface de

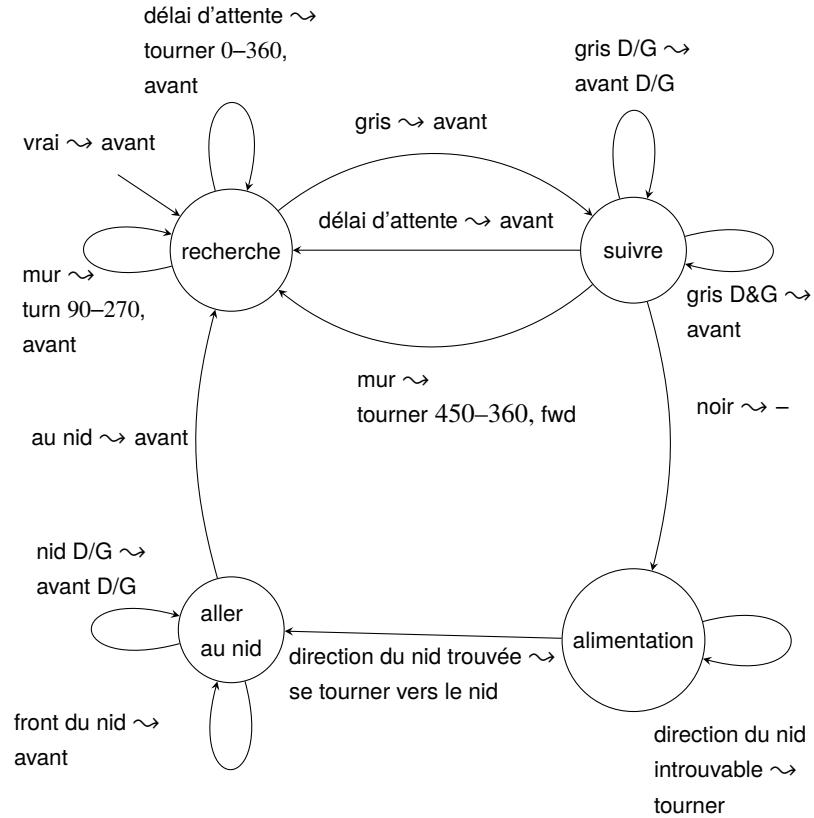


FIG. 7.14 – Machine à états pour tracer un chemin entre une source de nourriture et un nid. Voir le tableau 7.1 pour l’explication des abréviations.

la zone. S'il rencontre un mur, il effectue un virage aléatoire en s'éloignant du mur ; nous supposons que le capteur est orienté directement vers l'avant, de sorte que le virage aléatoire doit se faire dans une certaine direction, sur le côté ou à l'arrière du robot. Une fois que le robot a détecté un marquage gris, il passe à l'état suivre.

follow : Les deux auto-transitions situées au-dessus et à droite de cet état sont des transitions qui mettent en œuvre le suivi de ligne (3.4). Il existe trois autres transitions : Si un timeout se produit sans détecter de gris, le robot ne suit plus de ligne et doit retourner à l'état search. Si le robot rencontre un mur, nous voulons qu'il se détourne, mais nous lui demandons d'abord d'effectuer un tour complet de 360° pour vérifier s'il y a un marquage gris à proximité. Par conséquent, la transition inclut l'action turn 450–360. Comme le nid est situé à côté d'un mur, cette condition est également vraie lorsque le robot retourne au nid. Si le robot détecte un marquage

TAB. 7.1 – Abréviations dans la machine à états

Abréviation	Explication
avant	régler le moteur en avant
avant D/G	régler le moteur vers l'avant et vers la droite/gauche
fwd et fwd R/L régler également la minuterie sur une période aléatoire	
mur	mur détecté
délai d'attente	période de la minuterie expirée
gris D/G/D&G	gris détecté par les capteurs droit/gauche/les deux capteurs
front du nid/D/G	nid détecté à l'avant/à droite/à gauche
noir	noir détecté
direction du nid	direction de la nourriture vers le nid trouvée ou non trouvée
tourner $\theta_1-\theta_2$	tournent aléatoirement dans l'intervalle $\theta_1-\theta_2$
tourner	le robot (ou son capteur) tourne

à haute densité (noir), il conclut qu'il a atteint la source de nourriture et effectue la transition vers l'état à la nourriture.

at food : Enfin, le robot a découvert la source de nourriture. Il doit maintenant retourner au nid. Nous avons spécifié que le nid peut être détecté (Activity 7.7), mais le capteur du robot ne fait pas nécessairement face à la direction du nid. Par conséquent, le robot (ou son capteur) doit tourner jusqu'à ce qu'il trouve la direction du nid. Il se tourne alors vers le nid et passe à l'état goto nid.

goto nest : Cet état est similaire à l'état suivre en ce sens que le robot avance vers le nid, en tournant à droite ou à gauche si nécessaire pour se déplacer dans la direction du nid. Lorsqu'il atteint le nid, il retourne à l'état recherche.

Regardez à nouveau la Fig.7.10 qui montre une expérience réelle avec un robot exécutant cet algorithme. Nous voyons qu'il y a une forte densité de lignes entre le nid et la source de nourriture, mais qu'il y a aussi une densité relativement élevée de lignes à proximité du nid, pas nécessairement dans la direction de la source de nourriture. Cela peut amener le robot à revenir à une recherche aléatoire au lieu de se diriger directement vers la source de nourriture.

7.6 Résumé

Les algorithmes d'évitement d'obstacles utilisent des algorithmes connus depuis l'Antiquité dans le contexte de la navigation dans un labyrinthe. Lorsqu'ils sont utilisés pour l'évitement d'obstacles, diverses anomalies peuvent faire échouer les algorithmes, en particulier, l'obstacle en forme de G peut piéger un algorithme de

suivi de mur. L'algorithme Pledge permet de surmonter cette difficulté.

Une colonie de fourmis peut déterminer un chemin entre son nid et une source de nourriture sans connaître sa position et sans carte en renforçant un comportement aléatoire qui a un résultat positif.

7.7 Lecture complémentaire

Il existe une abondante littérature sur les labyrinthes, que l'on peut consulter en suivant les références de l'article de Wikipédia consacré à *Maze*. L'algorithme de Pledge a été découvert par John Pledge, âgé de 12 ans ; notre présentation est basée sur [2, Chap. 4]. Un projet basé sur les fourmis qui suivent les phéromones est décrit dans [33].

Chapitre 8

Localisation

La navigation par odométrie (5.4) est sujette à des erreurs et ne peut donner qu'une estimation de la pose réelle du robot. En outre, plus le robot se déplace, plus l'erreur dans l'estimation de la position, en particulier dans le cap, est importante. L'odométrie d'un robot peut être comparée à la marche les yeux fermés, en comptant les pas jusqu'à ce que nous atteignions notre destination. Comme pour l'odométrie, plus nous marchons, plus nous sommes incertains de notre position. Même en comptant les pas, nous devons ouvrir les yeux de temps en temps pour réduire l'incertitude de notre position. Pour un robot, que signifient "compter les pas" et "ouvrir les yeux de temps en temps"? Cela signifie que pour se déplacer sur de courtes distances, l'odométrie est suffisante, mais que pour se déplacer sur de plus longues distances, le robot doit déterminer sa position par rapport à une référence externe appelée point de repère. Ce processus est appelé *localisation*.

La section 8.1 commence par une activité que vous pouvez utiliser pour vous familiariser avec la relation entre l'odométrie et la localisation. La section 8.2 présente les techniques trigonométriques classiques utilisées par les géomètres pour déterminer la position d'un lieu sur terre en mesurant les angles et les distances par rapport à des positions dont l'emplacement est connu. La section 8.3 présente brièvement les systèmes de positionnement global (GPS) qui sont aujourd'hui largement utilisés pour la localisation. Il existe des environnements où le GPS n'est pas efficace : dans les bâtiments et lorsque des positions très précises sont nécessaires. Dans ces environnements, les robots utilisent des techniques de localisation probabiliste qui sont décrites dans les sections 8.4–8.5.

8.1 Landmarks

Les points de repère, tels que les lignes au sol ou les portes dans un couloir, peuvent être détectés et identifiés par le robot et utilisés pour la localisation. L'activité suivante, qui n'utilise ni ordinateur ni robot, vous aidera à comprendre l'importance de l'identification des points de repère pour corriger les erreurs d'odométrie.

Activity 8.1: Jouer avec le jeu des repères

- Choisissez un chemin dans votre maison qui vous oblige à passer plusieurs portes, par exemple, du lit de votre chambre au canapé du salon en passant par un couloir, puis retour.

- Fermez les yeux et marchez le long du chemin en appliquant les règles suivantes :
 - Vous obtenez 30 points au début de l'activité.
 - De temps en temps, vous pouvez ouvrir les yeux pendant une seconde ; cette action coûte 1 point.
 - Si vous touchez un mur, cela vous coûte 10 points.
 - Combien de points avez-vous lorsque vous avez terminé le chemin ?
 - Est-il préférable d'ouvrir les yeux fréquemment ou de parcourir le chemin sans jamais ouvrir les yeux ?

8.2 Détermination de la position à partir d'objets dont la position est connue

Dans cette section, nous décrivons deux méthodes qu'un robot peut utiliser pour déterminer sa position en mesurant les angles et les distances par rapport à un objet dont la position est connue. La première méthode suppose que le robot peut mesurer la distance à l'objet et son *azimut*, l'angle de l'objet par rapport au nord. La seconde méthode mesure les angles par rapport à l'objet à partir de deux positions différentes. Les deux méthodes utilisent la trigonométrie pour calculer les coordonnées du robot par rapport à l'objet. Si les coordonnées absolues (x_0, y_0) de l'objet sont connues, les coordonnées absolues du robot peuvent être facilement calculées.

8.2.1 Détermination de la position à partir d'un angle et d'une distance

La figure 8.1 illustre la géométrie d'un robot par rapport à un objet. Dans le diagramme, l'objet est désigné par le gros point placé à l'origine (x_0, y_0) d'un système de coordonnées. L'azimut du robot θ est l'angle entre le nord et la direction avant du robot ; il peut être mesuré à l'aide d'une boussole. Un scanner laser est utilisé pour mesurer la distance s par rapport à l'objet et l'angle ϕ entre la direction avant du robot et l'objet. Les coordonnées relatives Δx et Δy peuvent être déterminées par une simple trigonométrie :

$$\Delta x = s \sin(\theta - \phi), \quad \Delta y = s \cos(\theta - \phi).$$

A partir des coordonnées absolues connues de l'objet (x_0, y_0) , les coordonnées du robot peuvent être déterminées.

Activity 8.2: Détermination de la position à partir d'un angle et d'une distance

- Mettre en œuvre l'algorithme.

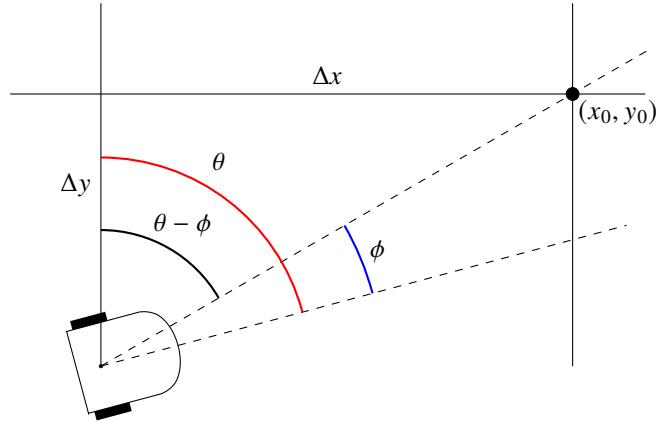


FIG. 8.1 – Détermination de la position à partir d'un angle et d'une distance

- Pour mesurer l'azimut, placez le robot dans l'alignement d'un bord de la table et appelez-le nord. Mesurer l'angle par rapport à l'objet en utilisant plusieurs capteurs horizontaux ou en utilisant un seul capteur et en faisant tourner le capteur ou le robot.
- La distance et l'angle sont mesurés à partir de la position où le capteur est monté, qui n'est pas nécessairement le centre du robot. Il se peut que vous deviez effectuer des corrections pour tenir compte de ce fait.

8.2.2 Détermination de la position par triangulation

La triangulation est utilisée pour déterminer les coordonnées lorsqu'il est difficile ou impossible de mesurer les distances. Elle était largement utilisée en topographie avant l'apparition des lasers, car il était impossible de mesurer avec précision de longues distances. Le principe de la triangulation est le suivant : à partir de deux angles d'un triangle et de la longueur du côté inclus, on peut calculer les longueurs des autres côtés. Une fois ces longueurs connues, il est possible de calculer la position relative d'un objet éloigné.

La figure 8.2 montre le robot mesurant les angles α et β par rapport à l'objet à partir de deux positions séparées par une distance c . Si les deux positions sont proches, la distance peut être mesurée à l'aide d'un mètre ruban. La distance peut également être mesurée par odométrie lorsque le robot se déplace d'une position à l'autre, bien que cette méthode soit moins précise. En topographie, si les coordonnées des deux positions sont connues, la distance qui les sépare peut être calculée et utilisée pour déterminer les coordonnées de l'objet.

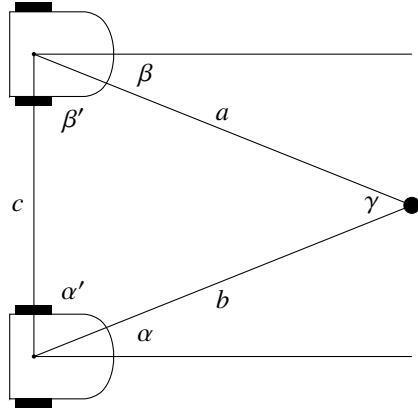


FIG. 8.2 – Triangulation

Les longueurs a et b sont calculées à l'aide de la *loi des sinus* :

$$\frac{a}{\sin \alpha'} = \frac{b}{\sin \beta'} = \frac{c}{\sin \gamma},$$

où $\alpha' = 90^\circ - \alpha$, : $\beta' = 90^\circ - \beta$ sont les angles intérieurs du triangle. Pour utiliser la loi, nous avons besoin de c , qui a été mesuré, et de γ , qui est :

$$\gamma = 180^\circ - \alpha' - \beta' = 180^\circ - (90^\circ - \alpha) - (90^\circ - \beta) = \alpha + \beta.$$

D'après la loi des sinus :

$$b = \frac{c \sin \beta'}{\sin \gamma} = \frac{c \sin(90^\circ - \beta)}{\sin(\alpha + \beta)} = \frac{c \cos \beta}{\sin(\alpha + \beta)}.$$

Un calcul similaire donne a .

Activity 8.3: Détermination de la position par triangulation

- Mettre en œuvre la triangulation.
- Initialement, effectuer une mesure de l'angle à une position, puis prendre le robot et le déplacer vers une nouvelle position pour la deuxième mesure, en mesurant soigneusement la distance c .
- Alternativement, faire en sorte que le robot se déplace de lui-même de la première position à la seconde, en calculant c par odométrie.

8.3 Système de positionnement global

Ces dernières années, la détermination de l'emplacement a été rendue plus facile et plus précise grâce à l'introduction du *Global Positioning System (GPS)*.¹ La navigation par GPS est basée sur des satellites en orbite. Chaque satellite connaît sa position précise dans l'espace et son heure locale. La position est envoyée au satellite par des stations terrestres et l'heure est mesurée par une horloge atomique très précise sur le satellite.

Un récepteur GPS doit pouvoir recevoir les données de quatre satellites. C'est pourquoi un grand nombre de satellites (24–32) est nécessaire pour qu'il y ait toujours une ligne de visée entre n'importe quel endroit et au moins quatre satellites. À partir des signaux temporels envoyés par un satellite, les distances entre les satellites et le récepteur peuvent être calculées en multipliant les temps de déplacement par la vitesse de la lumière. Ces distances et les positions connues des satellites permettent de calculer la position tridimensionnelle du récepteur : latitude, longitude et élévation.

L'avantage de la navigation GPS est qu'elle est précise et disponible partout sans équipement supplémentaire, à l'exception d'un composant électronique si petit et si peu coûteux qu'il se trouve dans chaque smartphone. La navigation par GPS pose deux problèmes :

- L'erreur de position est d'environ 10 mètres. Si elle est suffisante pour permettre à votre voiture de choisir la bonne route à une intersection, elle ne l'est pas pour effectuer des tâches qui nécessitent une plus grande précision, par exemple pour garer votre voiture.
- Les signaux GPS ne sont pas assez puissants pour la navigation en intérieur et sont sujets à des interférences dans les environnements urbains denses.

Relativité et GPS

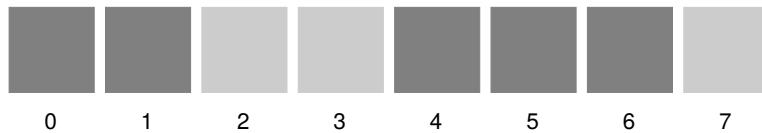
Vous avez certainement entendu parler de la théorie de la relativité d'Albert Einstein et l'avez probablement considérée comme une théorie ésotérique qui n'intéresse que les physiciens. Pourtant, les théories d'Einstein sont utilisées dans les calculs du GPS ! Selon la théorie spéciale de la relativité, les horloges des satellites tournent plus lentement que celles de la Terre (de 7,2 microsecondes par jour), car les satellites se déplacent rapidement par rapport à la Terre. Selon la théorie générale de la relativité, les horloges tournent *plus vite* (de 45,9 microsecondes par jour), parce que la force de gravité de la Terre est plus faible sur le satellite éloigné que pour nous à la surface. Les deux effets ne s'annulent

1. Le terme générique est *global navigation satellite system (GNSS)* puisque le GPS fait référence au système spécifique exploité par les États-Unis. Des systèmes similaires sont exploités par l'Union européenne (Galileo), la Russie (GLONASS) et la Chine (BeiDou), mais le terme GPS est souvent utilisé pour désigner l'ensemble de ces systèmes, et c'est ce que nous faisons ici.

pas et un facteur de correction est utilisé lors de la diffusion des signaux horaires.

8.4 Localisation probabiliste

Considérons un robot qui navigue dans un environnement connu pour lequel il dispose d'une *map*. La carte suivante montre un mur avec cinq portes (gris foncé) et trois zones où il n'y a pas de porte (gris clair) :

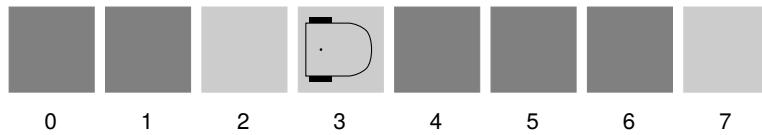


Pour plus de clarté, les portes et les murs sont dessinés comme s'ils se trouvaient sur le sol et que le robot se déplaçait au-dessus d'eux, en mesurant l'intensité à l'aide de capteurs au sol.

La tâche du robot consiste à franchir une porte spécifique, par exemple celle qui se trouve à la position 4. Mais comment le robot peut-il savoir où il se trouve ? Grâce à l'odométrie, le robot peut déterminer sa position actuelle à partir d'une position de départ connue. Par exemple, si le robot se trouve à l'extrémité gauche du mur :



il sait qu'il doit se déplacer de cinq fois la largeur de chaque porte, alors que si le robot se trouve à la position suivante :

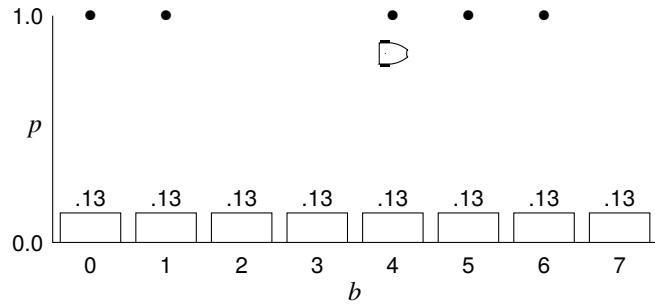


la porte souhaitée est la prochaine à droite. En raison des erreurs d'odométrie, il est fort probable que le robot se perde au fil du temps. Dans cette section, nous implementons une version unidimensionnelle d'un algorithme probabiliste de localisation *Markov* qui prend en compte l'incertitude dans les capteurs et dans le mouvement du robot, et renvoie les emplacements les plus probables du robot.

B.1 contient un bref tutoriel sur la probabilité conditionnelle et la règle de Bayes, y compris un exemple des détails des calculs de l'incertitude.

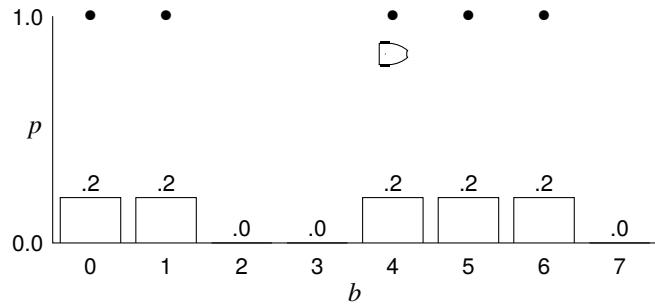
8.4.1 La détection augmente la certitude

Considérons un robot dans l'environnement ci-dessus, composé de murs et de portes, qui ne dispose d'aucune information sur son emplacement. Le robot attribue une probabilité aux huit positions où il pourrait se trouver. Au départ, il n'a aucune idée de l'endroit où il se trouve, et chaque position se verra donc attribuer la probabilité $b[i] = 1.0/8 = .125 \approx .13$, où b est appelé le *réseau de croyance* :²



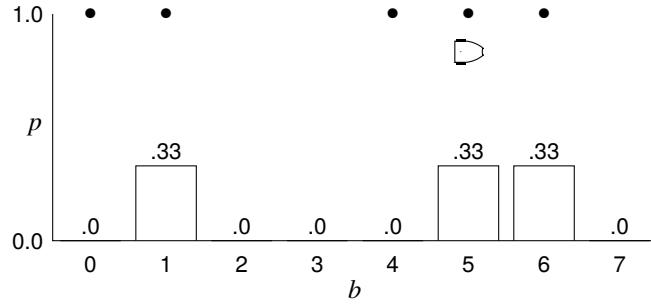
Dans les graphiques, les points indiquent les positions des portes et une petite icône indique la position actuelle du robot, qui est orienté vers la droite.

Supposons maintenant que les capteurs du robot détectent une zone gris foncé. Son incertitude est réduite, car il sait qu'il doit se trouver devant l'une des cinq portes. Le tableau de croyances indique .2 pour chacune des portes et .0 pour chacun des murs :

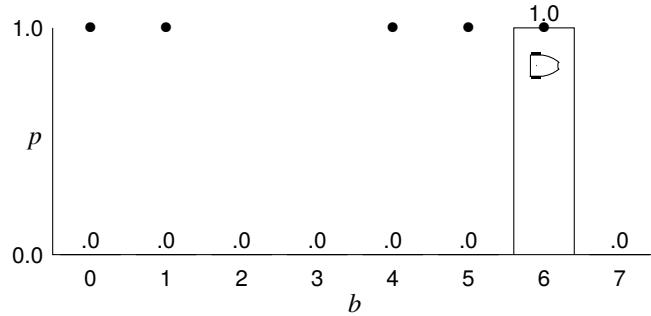


Le robot se déplace ensuite vers l'avant et détecte à nouveau une zone gris foncé. Il n'y a plus que trois possibilités : il était à la position 0 et s'est déplacé vers 1, il était à 4 et s'est déplacé vers 5, ou il était à 5 et s'est déplacé vers 6. Si la position initiale du robot était 1 ou 6, après s'être déplacé vers la droite, il ne détecterait plus de zone gris foncé et ne pourrait donc pas s'y trouver. La probabilité est maintenant de .33 pour chacune des trois positions 1, 4, 5 :

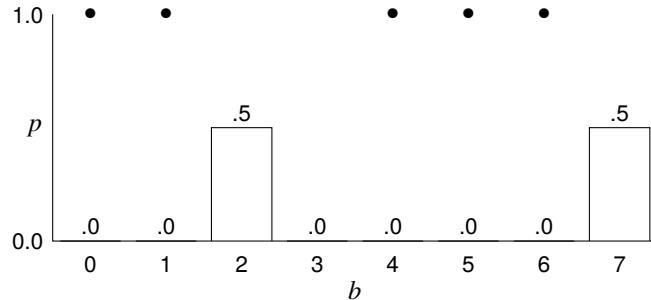
2. Toutes les probabilités seront arrondies à deux chiffres décimaux pour l'affichage.



Après l'étape suivante du robot, s'il détecte à nouveau une porte, il se trouve sans aucun doute à la position 6 :



Si le robot n'a pas détecté de porte, il se trouve soit à la position 2, soit à la position 7 :



Le robot maintient un tableau de croyances et intègre de nouvelles données lorsqu'il détecte la présence ou l'absence d'une porte. Au fil du temps, l'incertitude diminue : le robot sait avec plus de certitude où il se trouve réellement. Dans cet exemple, le robot finit par connaître avec certitude sa position devant la porte 6 ou par réduire son incertitude à l'une des deux positions 2, 7.

TAB. 8.1 – Localisation avec incertitude de détection, capteur=après multiplication par l'incertitude du capteur, norme=après normalisation, droite=après déplacement d'une position vers la droite

position porte ?	0	1	2	3	4	5	6	7
initial	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.13
capteur	0.11	0.11	0.01	0.01	0.11	0.11	0.11	0.01
norme	0.19	0.19	0.02	0.02	0.19	0.19	0.19	0.02
droite	0.02	0.19	0.19	0.02	0.02	0.19	0.19	0.19
capteur	0.02	0.17	0.02	0.00	0.02	0.17	0.17	0.02
norme	0.03	0.29	0.03	0.00	0.03	0.29	0.29	0.03
droite	0.03	0.03	0.29	0.03	0.00	0.03	0.29	0.29
capteur	0.03	0.03	0.03	0.00	0.00	0.03	0.26	0.03
norme	0.07	0.07	0.07	0.01	0.01	0.07	0.63	0.07

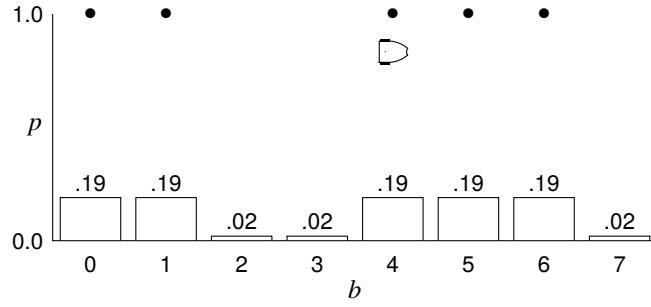
8.4.2 Incertitude dans la détection

Les valeurs renvoyées par les capteurs du robot reflètent l'intensité de la lumière réfléchie par les couleurs grises des portes et des murs. Si la différence de couleur entre une porte gris foncé et un mur gris clair n'est pas très importante, le robot peut parfois détecter une porte gris foncé comme un mur gris clair, ou inversement. Cela peut être dû à des changements dans l'éclairage ambiant ou à des erreurs dans les capteurs eux-mêmes. Il s'ensuit que le robot ne peut pas faire la distinction entre les deux avec une certitude totale.

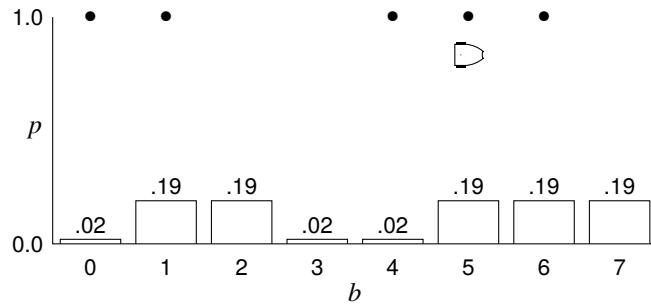
Nous modélisons cet aspect du monde en attribuant des probabilités à la détection. Si le robot détecte du gris foncé, nous spécifions que la probabilité est de .9 qu'il ait correctement détecté une porte et de .1 qu'il ait détecté par erreur un mur où se trouvait en fait une porte. Inversement, s'il détecte du gris clair, la probabilité est de .9 qu'il ait correctement détecté un mur et de .1 qu'il ait détecté par erreur une porte là où il y avait un mur.

Nous continuons à afficher les calculs sous forme de graphiques, mais vous trouverez peut-être plus facile de les suivre dans le tableau 8.1. Chaque ligne représente le tableau de croyances du robot suivant l'action écrite dans la première colonne.

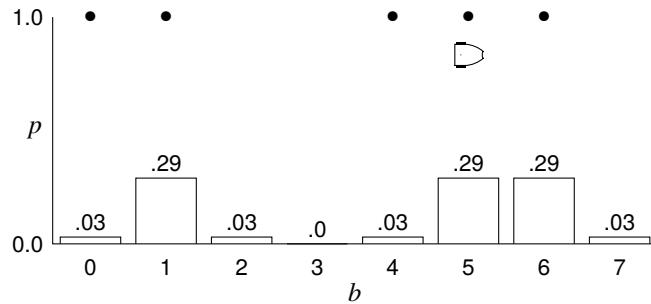
Initialement, après avoir détecté un gris foncé à un endroit où se trouve une porte, nous savons avec une probabilité de $0,125 \times 0,9 = 0,1125$ qu'une porte a été correctement détectée ; cependant, il existe toujours une probabilité de $0,125 \times 0,1 = 0,0125$ qu'un mur ait été détecté par erreur. Après normalisation (B.2), le tableau de croyances est le suivant :



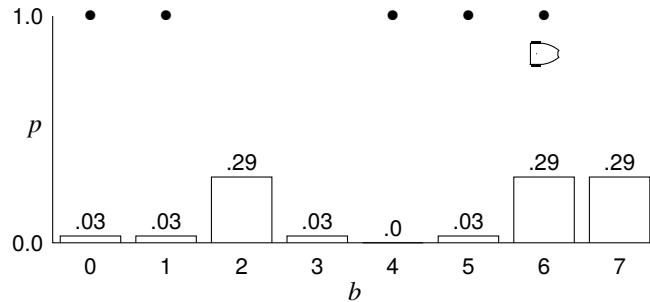
Que se passe-t-il lorsque le robot se déplace d'une position vers la droite ? Son tableau de croyances doit également se déplacer d'une position vers la droite. Par exemple, la probabilité .19 que le robot était à la position 1 devient la probabilité qu'il est à la position 2. De même, la probabilité .02 que le robot soit à la position 3 devient la probabilité qu'il soit à la position 4. La probabilité que le robot soit à la position 0 est maintenant de 0 et la probabilité b_7 devient b_8 , de sorte que les indices deviennent 1–8 au lieu de 0–7. Pour simplifier les calculs et les diagrammes de l'exemple, les indices 0–7 sont conservés et la valeur de b_8 est stockée dans b_0 comme si la carte était cyclique. Le tableau de croyances après le déplacement du robot vers la droite est le suivant :



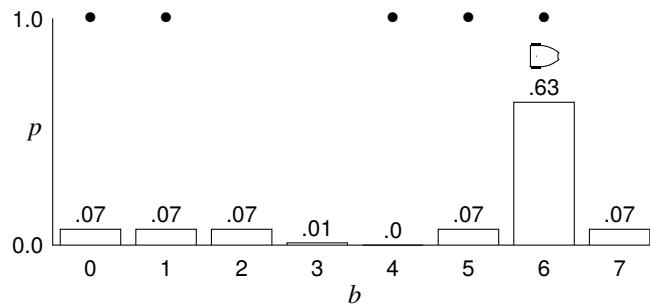
Si le robot détecte à nouveau du gris foncé, la probabilité de se trouver aux positions 1, 5 ou 6 devrait augmenter. Le calcul des probabilités et la normalisation donnent :



Le robot se déplace à nouveau vers la droite :



and senses a third dark gray area. The belief array becomes :



Il n'est pas surprenant que le robot se trouve presque certainement à la position 6.

Activity 8.4: Localisation avec incertitude dans les capteurs

- Mettre en œuvre la localisation probabiliste avec l'incertitude dans le capteur.
- Comment le comportement de l'algorithme change-t-il lorsque l'incertitude est modifiée ?
- Exécuter l'algorithme pour différentes positions de départ du robot.

8.5 Incertitude du mouvement

Outre l'incertitude liée aux capteurs, les robots sont soumis à l'incertitude liée à leur mouvement. Nous pouvons demander au robot de se déplacer d'une position vers la droite, mais il se peut qu'il se déplace de deux positions, ou qu'il se déplace très peu et reste dans sa position actuelle. Modifions l'algorithme pour prendre en compte cette incertitude.

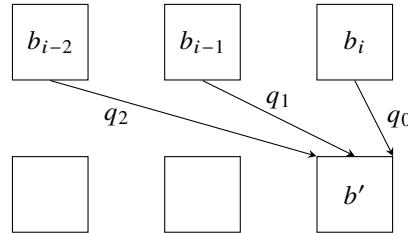
Soit b le tableau de croyances. Le tableau de croyances est mis à jour à l'aide de la formule suivante :

$$b'_i = p_i b_i,$$

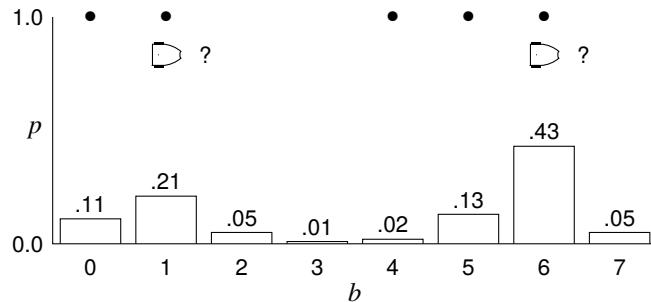
où b'_i est la nouvelle valeur de b_i et p_i est la probabilité de détecter une porte (dans l'exemple, p_i est 0,9 pour $i = 0, 1, 4, 5, 6$ et p_i est 0,1 pour $i = 2, 3, 7$). Si le mouvement est certain, le robot se déplace d'une position vers la droite, mais avec un mouvement incertain, le calcul suivant prend en compte les probabilités q_j que le robot se déplace effectivement de $j = 0, 1, 2$ positions :

$$b'_i = p_i (b_{i-2} q_2 + b_{i-1} q_1 + b_i q_0),$$

comme le montre le diagramme suivant :



It is highly likely that the robot will move correctly, so reasonable values are $q_1 = 0.8$ and $q_0 = q_2 = 0.1$. With these values for the uncertainty of the motion and the previous values for p_i , the calculation of the belief array after three moves is shown in Table 8.2 and its final value is shown in the following diagram :



Le robot est probablement en position 6, mais nous sommes moins sûrs car la probabilité n'est que de .43 au lieu de .63. Il y a une probabilité non négligeable de .21 que le robot soit en position 1.

Activity 8.5: Localisation avec incertitude dans le mouvement

- Mettre en œuvre la localisation probabiliste avec incertitude dans le calcul de la puissance du moteur.
- Comment le comportement de l'algorithme change-t-il lorsque l'incertitude est modifiée ?
- Exécuter l'algorithme pour différentes positions de départ du robot.

TAB. 8.2 – *Localisation avec incertitude de détection et de mouvement, capteur=après multiplication par l'incertitude du capteur, norme=après normalisation, droite=après déplacement d'une position vers la droite*

position porte ?	0	1	2	3	4	5	6	7
initial	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.13
capteur	0.11	0.11	0.01	0.01	0.11	0.11	0.11	0.01
norme	0.19	0.19	0.02	0.02	0.19	0.19	0.19	0.02
droite	0.05	0.17	0.17	0.04	0.04	0.17	0.19	0.17
capteur	0.05	0.17	0.02	0.00	0.03	0.15	0.17	0.02
norme	0.08	0.27	0.03	0.01	0.06	0.25	0.28	0.03
droite	0.06	0.09	0.23	0.05	0.01	0.07	0.23	0.25
capteur	0.05	0.10	0.02	0.01	0.01	0.06	0.21	0.02
norme	0.11	0.21	0.05	0.01	0.02	0.13	0.43	0.05

8.6 Résumé

L'odométrie fournit une estimation de la position d'un robot. Un robot peut utiliser des techniques d'arpentage pour calculer sa position par rapport à un objet dont la position est connue. Le GPS fournit d'excellentes données sur la localisation, mais il n'est pas toujours assez précis et les interférences avec la réception des satellites limitent son utilisation dans les environnements intérieurs. Si le robot peut détecter plusieurs objets connus et s'il dispose d'une carte de son environnement, il peut utiliser la localisation probabiliste pour estimer sa position avec une probabilité élevée, bien que cette probabilité soit réduite si les capteurs ou les mouvements du robot sont très incertains.

8.7 Lecture complémentaire

Les méthodes probabilistes en robotique sont traitées en profondeur dans [46]. Le site <http://www.gps.gov> contient une mine d'informations sur le GPS. Une mise en œuvre de la localisation probabiliste à l'aide du robot éducatif Thymio est décrite dans [49].

Chapitre 9

Cartographie

Nous avons vu qu'un robot peut utiliser sa capacité à détecter les obstacles pour se localiser, sur la base d'informations relatives à la position des obstacles ou d'autres informations sur l'environnement. Ces informations sont normalement fournies par une carte. Il est relativement facile de construire une carte des environnements industriels tels que les usines, puisque les machines sont ancrées à des endroits fixes. Les cartes sont moins pertinentes pour un aspirateur robotisé, car le fabricant ne peut pas préparer des cartes des appartements de chaque client. En outre, les robots seraient trop difficiles à utiliser si les clients devaient établir des plans de leurs appartements et les modifier chaque fois qu'un meuble est déplacé. Il va sans dire qu'il est impossible de construire à l'avance des plans de lieux inaccessibles comme le fond des océans.

La solution consiste à demander au robot de construire sa propre carte de l'environnement. La construction d'une carte nécessite une localisation afin que le robot sache où il se trouve, mais la localisation nécessite une carte, qui nécessite des Pour résoudre ce problème de la poule et de l'œuf, les robots utilisent des algorithmes de *simultaneous localization and mapping (SLAM)*. Pour effectuer le SLAM, les robots utilisent des informations connues pour être valables même dans les parties inexplorées de l'environnement, en affinant les informations au cours de l'exploration.

C'est ce que les humains ont fait pour créer des cartes géographiques. Les observations du soleil et des étoiles ont été utilisées pour la localisation et les cartes ont été créées au fur et à mesure de l'exploration. Au départ, les outils de localisation étaient médiocres : il est relativement facile de mesurer la latitude en utilisant un sextant pour observer la hauteur du soleil à midi, mais il était impossible de mesurer avec précision la longitude jusqu'à ce que des horloges précises, appelées chronomètres, soient mises au point à la fin du dix-huitième siècle. La localisation s'est améliorée, tout comme les cartes, qui comprennent non seulement les terres et les côtes maritimes, mais aussi les caractéristiques du terrain comme les lacs, les forêts et les montagnes, ainsi que les structures artificielles comme les bâtiments et les routes.

Les sections 9.1–9.2 présentent les méthodes de représentation des cartes dans un ordinateur. La section 9.3 décrit comment un robot peut créer une carte en utilisant l'algorithme de frontière. La section 9.4 explique comment la connaissance partielle de l'environnement aide à construire une carte. Un algorithme SLAM fait l'objet des trois dernières sections. L'algorithme est d'abord présenté dans la section 9.5 à l'aide

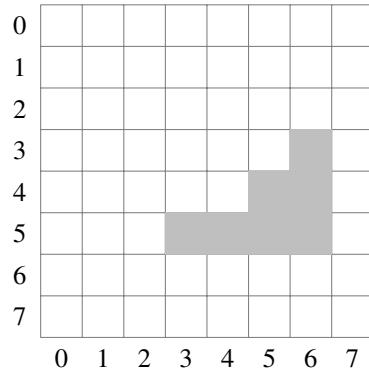


FIG. 9.1 – *Carte discrète des cellules occupées d'un objet*

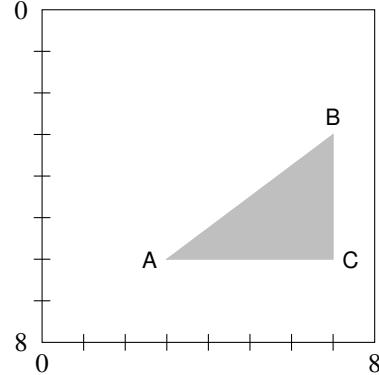


FIG. 9.2 – *Une carte continue du même objet*

d'un exemple relativement simple. Les activités pour le SLAM sont rassemblées dans la Sect. 9.6 et la Sect. 9.7 explique l'algorithme formel.

9.1 Cartes discrètes et continues

Nous sommes habitués aux cartes graphiques imprimées sur papier ou, plus couramment de nos jours, affichées sur les ordinateurs et les smartphones. Un robot, cependant, a besoin d'une représentation non visuelle d'une carte qu'il peut stocker dans sa mémoire. Il existe deux techniques de stockage des cartes : les cartes discrètes (également appelées *cartes en grille*) et les cartes continues.

La figure 9.1 montre une carte quadrillée de 8×8 avec un objet triangulaire. L'emplacement de l'objet est stocké sous la forme d'une liste des coordonnées de chaque cellule de la grille couverte par l'objet. L'objet de la figure est constitué des cellules situées à :

$$(5, 3), (5, 4), (5, 5), (4, 5), (5, 6), (4, 6), (3, 6).$$

La figure 9.2 montre une *carte continue* du même objet. Au lieu de stocker les positions de l'objet, ce sont les coordonnées des positions des limites qui sont stockées :

$$A = (6, 3), B = (3, 7), C = (6, 7).$$

Les cartes discrètes ne sont pas très précises : il est difficile de reconnaître l'objet de la Fig. 9.1 comme un triangle. Pour améliorer la précision, il faut utiliser une grille plus fine : 16 fois 16 ou même 256 fois 256. Bien entendu, plus le nombre de points de la grille augmente, plus la taille de la mémoire du robot doit être importante. En outre, un ordinateur plus puissant doit être utilisé pour traiter les cellules de la grille.

Les robots mobiles sont soumis à des contraintes de poids, de coût, de capacité de batterie, etc.

Si les objets de l'environnement sont peu nombreux et de forme simple, une carte continue est plus efficace et plus précise. Dans la figure 9.2, trois paires de nombres représentent le triangle avec beaucoup plus de précision que les sept paires de la carte discrète. En outre, il est facile de calculer si un point se trouve ou non dans l'objet à l'aide de la géométrie analytique. Cependant, s'il y a beaucoup d'objets ou s'ils ont des formes très complexes, les cartes continues ne sont plus efficaces, que ce soit en termes de mémoire ou de quantité de calculs nécessaires. L'objet de la Fig. 9.2 est délimité par des lignes droites, mais si la limite était décrite par des courbes d'ordre élevé, le calcul deviendrait difficile. Considérons une carte comportant 32 objets de taille 1, dont aucun ne se touche. La carte discrète aurait 32 coordonnées, tandis que la carte continue devrait stocker les coordonnées des quatre coins de chaque objet.

En robotique mobile, les cartes discrètes sont couramment utilisées pour représenter des cartes d'environnements, comme nous l'avons fait au chapitre 8.

9.2 Le contenu des cellules d'une carte en grille

Une carte géographique utilise des notations conventionnelles pour décrire l'environnement. Des couleurs sont utilisées : le vert pour les forêts, le bleu pour les lacs, le rouge pour les autoroutes. Des symboles sont utilisés : des points de différentes tailles pour indiquer les villages et les villes, et des lignes pour les routes, où l'épaisseur et la couleur d'une ligne sont utilisées pour indiquer la qualité de la route. Les robots utilisent des cartes quadrillées où chaque cellule stocke un nombre et nous devons décider ce qu'un nombre encode.

Le codage le plus simple consiste à attribuer un bit à chaque cellule. Une valeur de 1 indique qu'un objet existe dans cette cellule et une valeur de 0 indique que la cellule est vide. Dans la figure 9.1, le gris représente la valeur 1 et le blanc la valeur 0.

Cependant, les capteurs ne sont pas précis et il peut être difficile de savoir si une cellule est occupée par un objet ou non. Il est donc logique d'attribuer une probabilité à chaque cellule pour indiquer dans quelle mesure nous sommes certains que l'objet se trouve dans cette cellule. La figure 9.3 est une copie de la figure 9.1 avec les probabilités énumérées pour chaque cellule. Les cellules sans numéro sont supposées avoir une probabilité de 0.

On constate que les cellules ayant une probabilité d'au moins .7 sont celles considérées dans la Fig. 9.1 comme étant des cellules occupées par l'objet. Bien entendu, nous sommes libres de choisir un autre seuil, par exemple .5, auquel cas davantage de cellules sont considérées comme occupées. Dans cet exemple, nous savons que l'objet est triangulaire, nous pouvons donc voir qu'un seuil de .5 rend l'objet plus grand qu'il ne l'est en réalité, alors que le seuil plus élevé de .7 donne

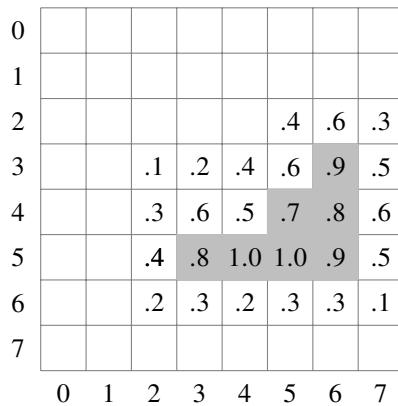


FIG. 9.3 – Une carte de grille probabiliste

une meilleure approximation.

Activity 9.1: Carte probabiliste des obstacles

- Placez votre robot sur une ligne devant un ensemble d'obstacles que le robot peut détecter avec un capteur latéral (Fig. 9.4). Si vous avez mis en œuvre la localisation comme décrit dans Activity 8.4, vous pouvez utiliser cette information pour établir où se trouve le robot. Sinon, dessinez des lignes régulières sur le sol qui peuvent être lues par le robot pour la localisation. Construisez une carte probabiliste des obstacles.
- Comment les probabilités changent-elles lorsque des obstacles sont ajoutés ou supprimés ?

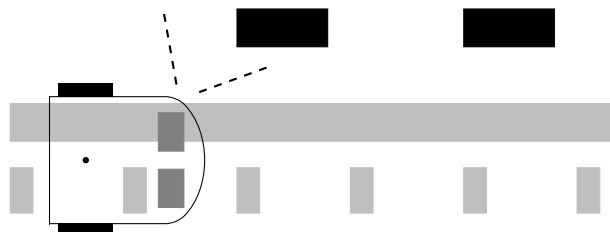


FIG. 9.4 – Les rectangles noirs représentent les obstacles à mesurer. La ligne grise guide le robot et les marques grises sont utilisées pour la localisation

?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	1	?	?	?	?	?	0.9	1	0.9	?	?	?
?	?	?	?	?	1	0.1	0.1	?	?	?	?	1	0.2	1	?	?	?
?	?	?	?	?	1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?	?
?	?	?	?	?	0.9	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?	?
?	?	?	?	?	?	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	?	?	?	?
?	?	?	?	?	?	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	?	?	?	?
?	?	?	?	?	?	?	?	0.2	0.1	0.1	0.2	0.2	0.1	?	?	?	?
?	?	?	?	?	?	?	?	1	1	0.9	1	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

FIG. 9.5 – Grid map of an environment with occupancy probabilities

9.3 Création d'une carte par exploration : L'algorithme de frontière

Prenons l'exemple d'un aspirateur robotisé qui vient d'être lâché dans votre appartement. De toute évidence, il n'est pas préprogrammé avec une carte de votre appartement. Il doit donc explorer l'environnement pour recueillir des informations qui lui permettront de construire son propre plan. Il existe plusieurs façons d'explorer l'environnement, la plus simple étant l'exploration aléatoire. L'exploration sera beaucoup plus efficace si le robot dispose d'une carte partielle qu'il peut utiliser pour guider son exploration.

9.3.1 Cartes en grille avec probabilités d'occupation

La carte de la figure 9.5 est une carte quadrillée où chaque cellule est étiquetée avec sa *probabilité d'obstacle*, qui est la probabilité qu'il y ait un obstacle dans la cellule. L'obstacle peut être un mur, une table ou toute autre chose qui ne permet pas au robot de passer par cette cellule. Les points d'interrogation représentent les cellules qui n'ont pas encore été explorées. En l'absence de toute connaissance sur le contenu d'une cellule, on peut supposer que la probabilité qu'il y ait un obstacle à cet endroit est de 0,5, puisqu'elle peut tout aussi bien être occupée ou non. Un point d'interrogation est utilisé à la place de la valeur 0,5 pour clarifier le statut inexploré des cellules.

Le centre de la carte est libre de tout obstacle et les probabilités d'occupation de ces cellules, appelées "cellules ouvertes", sont faibles (0,1 ou 0,2). Il existe trois obstacles connus, en haut à droite, en haut à gauche et en bas au centre. Les obstacles sont caractérisés par des probabilités d'occupation élevées (0,9 ou 1,0) et

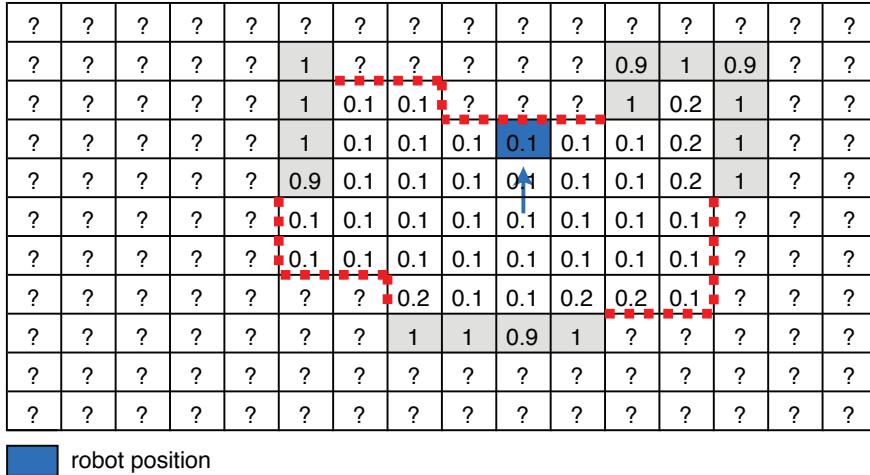


FIG. 9.6 – Le robot se déplace vers la frontière

sont indiqués par des cellules grises. Une cellule frontière est une cellule ouverte qui est adjacente (à gauche, à droite, en haut, en bas) à une ou plusieurs cellules inconnues. L'ensemble des cellules frontières est appelé *frontière*. Les lignes rouges des petits carrés de la Fig. 9.5 représentent la limite entre la frontière et les cellules inconnues de l'environnement. Les cellules inconnues adjacentes à la frontière sont les cellules intéressantes qu'il convient d'explorer afin d'élargir la carte actuelle.

9.3.2 L'algorithme de frontière

L'algorithme *frontière* est utilisé pour étendre la carte en explorant la frontière. Le robot se déplace jusqu'à la cellule frontière la plus proche, détecte s'il y a des obstacles dans les cellules adjacentes inconnues et met à jour la carte en conséquence.

La carte de la grille de la figure 9.6 est la même que celle de la figure 9.5 avec l'ajout du robot qui occupe une cellule colorée en bleu. La cellule frontière la plus proche du robot est la cellule située deux pas au-dessus de sa position initiale. La flèche indique que le robot s'est déplacé vers cette cellule. Le robot utilise ses capteurs locaux pour déterminer s'il y a des obstacles dans les cellules inconnues adjacentes. (Les capteurs peuvent détecter des obstacles dans les huit cellules adjacentes, y compris celles qui se trouvent sur la diagonale). Supposons que la cellule en haut à gauche contienne certainement un obstacle (probabilité 1.0), alors que les cellules directement au-dessus et à droite ne contiennent presque certainement pas d'obstacle (probabilité 0.1). La figure 9.7 montre la carte mise à jour avec ces nouvelles informations et la nouvelle position de la frontière.

La figure 9.8 montre le résultat de l'itération suivante de l'algorithme. Le robot

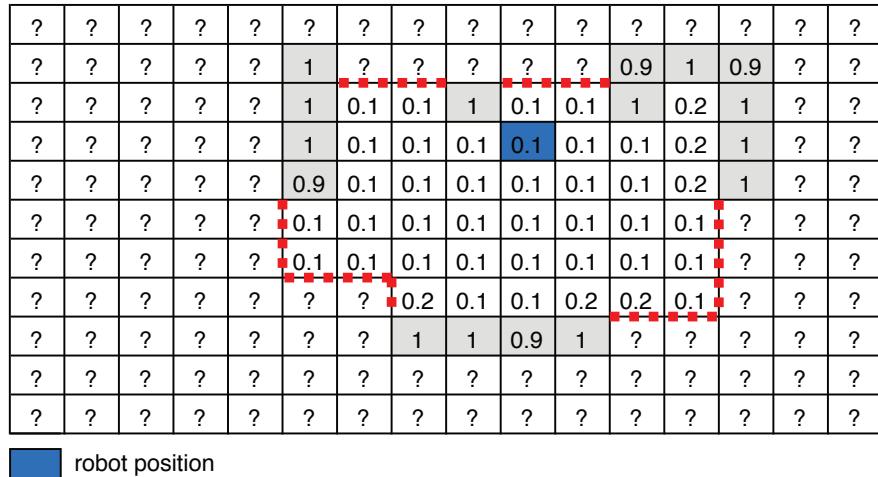


FIG. 9.7 – Le robot met à jour les cellules inconnues adjacentes à la frontière

s'est déplacé d'une cellule vers la cellule frontière la plus proche, a détecté des obstacles dans les deux cellules inconnues adjacentes et a mis à jour la carte. L'obstacle en haut à droite est parfaitement connu et il n'y a pas de cellule frontière à proximité de la position actuelle du robot.

La figure 9.9 montre l'itération suivante de l'algorithme. Le robot est bloqué par

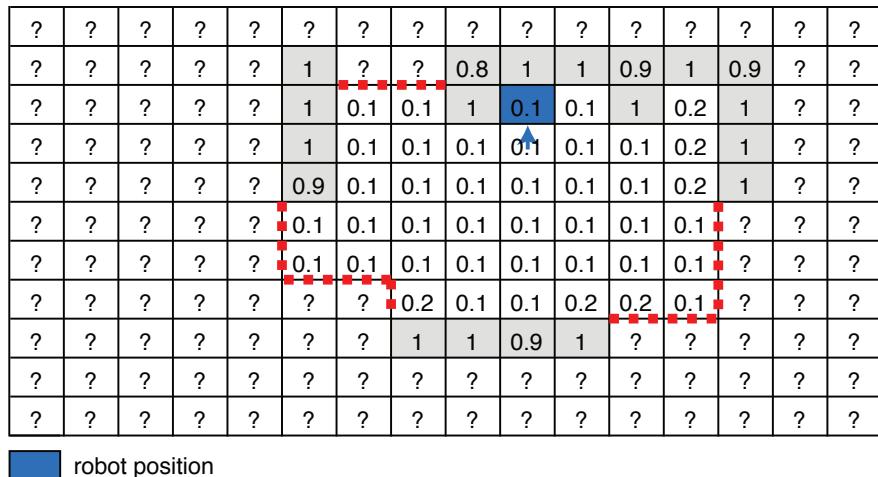


FIG. 9.8 – Deuxième itération de l'algorithme de frontière

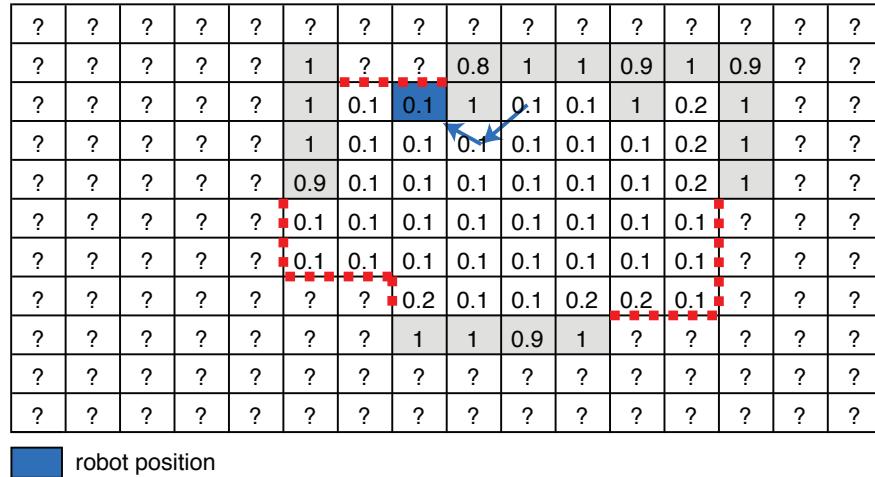


FIG. 9.9 – Le robot évite un obstacle en se déplaçant vers la prochaine frontière

l'obstacle en haut à droite et doit l'éviter en se déplaçant vers la cellule frontière la plus proche.

La figure 9.10 montre la carte complète construite par le robot après qu'il ait exploré toute la frontière, comme le montre le chemin avec les flèches bleues.

L'algorithme 9.1 formalise l'algorithme de frontière. Pour des raisons de simplicité, l'algorithme recalcule la frontière à chaque étape. Un algorithme plus sophistiqué examinerait les cellules dans le voisinage de la position du robot et ajouterait ou supprimerait les cellules dont le statut de cellule frontière a changé.

L'exemple auquel nous avons appliqué l'algorithme de frontière est un environnement relativement simple composé de deux pièces reliées par une porte (à la sixième colonne en partant de la gauche sur la Fig. 9.10), mais par ailleurs fermé à l'environnement extérieur. Cependant, l'algorithme de frontière fonctionne dans des environnements plus complexes.

L'algorithme de frontière peut être exécuté en parallèle par plusieurs robots. Chaque robot explore la partie de la frontière la plus proche de sa position. Les robots partagent leurs cartes partielles afin de maintenir la cohérence des cartes. Comme chaque robot explore une zone différente de l'environnement, la construction de la carte sera beaucoup plus efficace.

9.3.3 Priority in the frontier algorithm

L'algorithme de frontière peut utiliser des critères autres que la distance pour choisir la frontière à explorer. Considérons l'exploration de la carte quadrillée présentée sur la Fig. 9.11. Le robot se trouve dans la cellule (3, 3) marquée d'un cercle

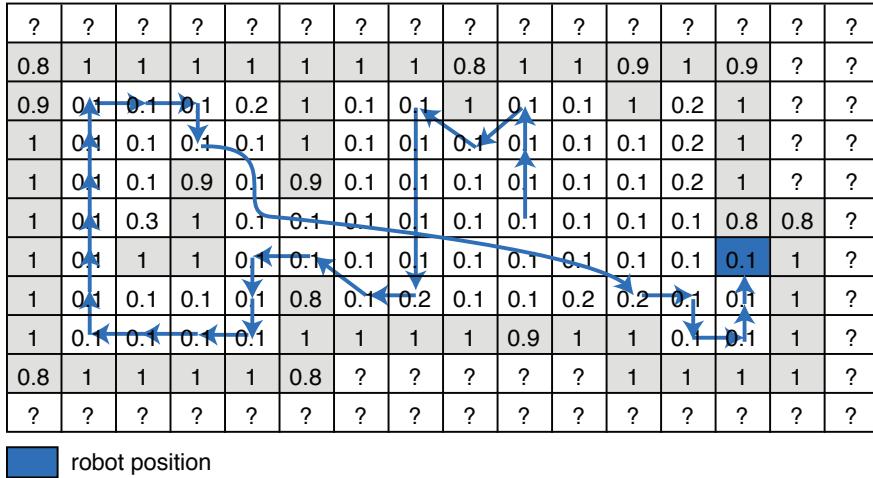


FIG. 9.10 – La carte construite par l'algorithme de frontière et le chemin exploré par le robot

bleu. Il y a six cellules d'obstacles connues et cinq cellules ouvertes connues, dont les trois cellules (1, 3), (2, 2), (3, 2), marquées par des carrés rouges, sont les cellules frontières. (Ici, les voisins diagonaux ne sont pas considérés comme adjacents).

Dans l'algorithme 9.1, le robot utilise la distance à une cellule frontière comme critère pour décider où se déplacer. Dans la figure 9.11, la cellule située à la gauche du robot à (3, 2) est la cellule la plus proche puisqu'elle n'est qu'à un pas, alors que les deux autres cellules frontières sont à deux pas. Nous pouvons considérer un critère différent en prenant en compte le nombre de cellules inconnues adjacentes à une cellule frontière. Commencer par une cellule frontière avec plus de cellules inconnues peut rendre l'algorithme plus efficace. Nous définissons la priorité d'une cellule frontière comme suit :

$$p_{cell} = \frac{a_{cell}}{d_{cell}},$$

où a_{cell} est le nombre de cellules inconnues adjacentes et d_{cell} est la distance par rapport au robot. Les priorités des trois cellules frontières sont les suivantes :

$$p_{(3,2)} = 1/1 = 1, \quad p_{(2,2)} = 2/2 = 1, \quad p_{(1,3)} = 3/2 = 1.5.$$

La priorité de la cellule (1, 3) est la plus élevée et l'exploration commence à cet endroit.

Activity 9.2: algorithme de frontière

Algorithm 9.1: Algorithme de frontière	
float array grid	// Grid map
cell list frontier	// List of frontier cells
cell robot	// Cell with robot
cell closest	// Closest cell to robot
cell c	// Index over cells
float low	// Low occupancy probability
1: loop	
2: frontier \leftarrow empty	
3: for all known cells c in the grid	
4: if grid(c) $<$ low and	
5: exists unknown neighbor of c	
6: append c to frontier	
7: exit if frontier empty	
8: closest \leftarrow cell in frontier nearest robot	
9: robot \leftarrow closest	
10: for all unknown neighbors c of closest	
11: sense if c is occupied	
12: mark grid(c)	
	with occupancy probability

- Mettre en œuvre l'algorithme de frontière. Vous devrez inclure un algorithme permettant de se déplacer avec précision d'une cellule à l'autre et un algorithme permettant de contourner les obstacles.
- Exécutez le programme sur la carte quadrillée de la Fig. 9.10. Obtenez-vous la même trajectoire ? Si ce n'est pas le cas, pourquoi ?
- Exécuter le programme sur la carte quadrillée de la Fig. 9.11.
- Modifier l'Algorithme 9.1 pour utiliser la priorité décrite dans Sect. 9.3.3. Le chemin est-il différent de celui généré par le programme original ?
- Essayez d'implémenter l'algorithme de frontière sur votre robot. Quel est l'aspect le plus difficile de la mise en œuvre ?

0	?	?	?	?	?	?	?
1	?	?	?	.1	?	?	?
2	?	?	.1	.1	1	?	?
3	?	?	.1	.1	1	?	?
4	?	1	1	1	1	?	?
5	?	?	?	?	?	?	?
6	?	?	?	?	?	?	?

0 1 2 3 4 5 6

FIG. 9.11 – Exploration d'un labyrinthe

9.4 Cartographier en utilisant la connaissance de l'environnement

Maintenant que nous savons comment explorer un environnement, examinons comment construire une carte pendant l'exploration. Au Chap. 8, nous avons vu qu'un robot peut se localiser à l'aide de repères externes et de leur représentation dans une carte. Sans ces repères externes, le robot ne peut compter que sur l'odométrie ou la mesure inertielle, qui sont sujettes à des erreurs qui augmentent avec le temps (Fig. 5.7). Comment est-il possible d'établir une carte lorsque la localisation est sujette à d'importantes erreurs ?

Même avec une mauvaise odométrie, le robot peut construire une meilleure carte s'il dispose d'informations sur la structure de l'environnement. Supposons que le robot essaie de construire le plan d'une pièce en suivant ses murs. Les différences entre les vitesses réelles des roues gauche et droite amèneront le robot à conclure que les murs ne sont pas droits (Fig. 9.12), mais si le robot sait *à l'avance* que les murs sont droits et perpendiculaires l'un à l'autre, il peut construire la carte montrée dans la Fig. 9.13. Lorsqu'il rencontre un virage serré, il comprend qu'il s'agit d'un coin à 90° où deux murs se rencontrent, et sa cartographie des angles sera donc correcte. Il y aura également une erreur lors de la mesure des longueurs des murs, ce qui peut entraîner l'écart illustré dans la figure entre le premier et le dernier mur. La figure montre un petit espace qui ne serait pas important, mais si le robot cartographie une grande zone, le problème de *fermer une boucle* dans une carte est difficile à résoudre parce que le robot n'a qu'une vue locale de l'environnement.

Considérons une tondeuse à gazon robotisée à qui l'on confie la tâche de tondre une pelouse en se déplaçant d'avant en arrière ; elle doit fermer la boucle en retournant à sa station de charge (Fig. 9.14). Il n'est pas possible de mettre en œuvre ce

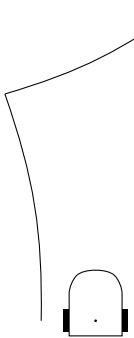


FIG. 9.12 – Mouvement perçu d'un robot basé sur l'odométrie

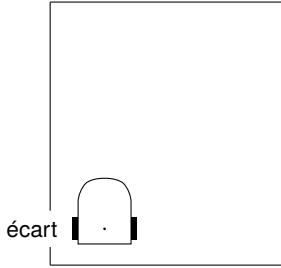


FIG. 9.13 – Odométrie et connaissance de la géométrie des murs

comportement en utilisant uniquement l'odométrie, car de petites erreurs dans la vitesse et le cap entraînent de grandes erreurs dans la position du robot. Il est très peu probable que l'odométrie seule permette au robot de tondre toute la surface de la pelouse et de retourner à sa station de charge. Des points de repère tels que des câbles de signalisation dans le sol doivent être utilisés pour boucler la boucle.

La construction de cartes peut être considérablement améliorée par l'utilisation de données de capteurs qui peuvent fournir des informations sur les caractéristiques régulières de l'environnement, en particulier à longue distance. Les caractéristiques régulières peuvent être des lignes au sol, une orientation globale ou la détection de caractéristiques qui *overlap* avec d'autres mesures. Supposons que nous disposions d'un capteur de distance capable de mesurer les distances sur une grande surface (Fig. 9.15). La mesure sur une grande surface permet au robot d'identifier des caractéristiques telles que les murs et les coins à partir d'une mesure prise à un seul endroit. Les mesures sur une grande surface facilitent l'identification des chevauchements entre les cartes locales qui sont construites à chaque endroit au fur et à mesure

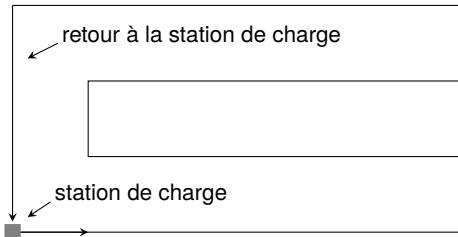


FIG. 9.14 – Une tondeuse à gazon robotisée tond une zone et retourne à sa station de charge.

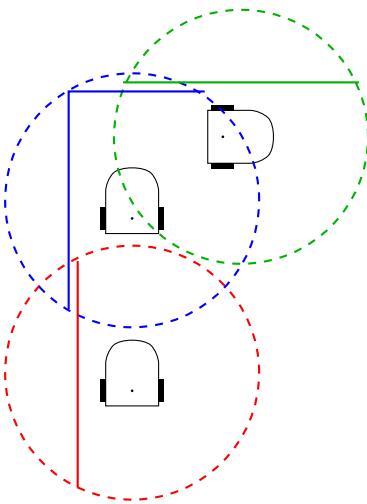


FIG. 9.15 – Les mesures des capteurs à longue portée permettent de détecter les chevauchements

que le robot se déplace dans l'environnement. En comparant les cartes locales, la localisation peut être corrigée et la carte mise à jour avec précision. C'est le sujet de l'algorithme SLAM décrit dans la section suivante.

Activity 9.3: Tondeuse à gazon robotisée

- Écrire un programme qui permet à une tondeuse à gazon robotisée de se déplacer le long d'une trajectoire en utilisant uniquement l'odométrie (Fig. 9.14). Exécutez le programme plusieurs fois. Le robot revient-il à la station de charge ? Si ce n'est pas le cas, quelle est l'ampleur des erreurs ? Les erreurs sont-elles constantes ou changent-elles d'une exécution à l'autre ?
- Placez des points de repère avec du ruban adhésif noir autour de la pelouse (Fig. 9.16). Programmez votre robot pour qu'il reconnaisse les repères et corrige sa localisation. Quelle doit être la taille des repères pour que le robot les détecte de manière fiable ?

9.5 Exemple numérique d'un algorithme SLAM

L'algorithme SLAM est assez compliqué, c'est pourquoi nous calculons d'abord un exemple numérique avant d'en donner la présentation formelle.

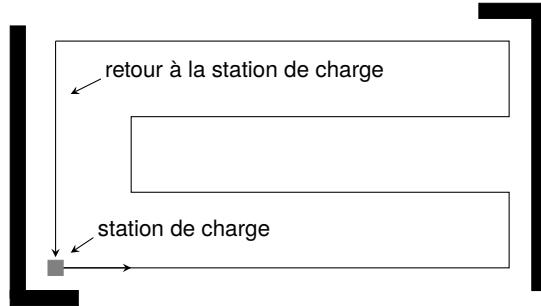


FIG. 9.16 – Une tondeuse robotisée avec des points de repère

La figure 9.17 montre un robot dans une pièce se dirigeant vers le haut du diagramme. Le robot se trouve près d'un coin de la pièce et il y a une projection du mur à la gauche du robot, peut-être un pilier de soutien du bâtiment. La Fig. 9.18 est la carte correspondante. Le gros point indique la position du robot et la flèche associée indique sa direction. La ligne pointillée épaisse représente le mur réel. Les cellules blanches représentent les endroits connus pour être libres, les cellules grises représentent les obstacles, et les cellules avec des points d'interrogation sont encore inexplorées. Une cellule est considérée comme faisant partie de l'obstacle si la majorité de sa surface se trouve derrière le mur. Par exemple, les deux segments horizontaux de la projection à partir du mur sont proches de la limite des cellules qu'ils traversent, mais ces cellules sont considérées comme faisant partie de l'obstacle parce que la quasi-totalité de leur surface se trouve derrière le mur.

Pour présenter les détails de l'algorithme SLAM, la carte est fortement simplifiée. Tout d'abord, les cellules sont beaucoup trop grandes, à peu près de la même taille que le robot lui-même. En pratique, les cellules seraient beaucoup plus petites que le robot. Deuxièmement, nous spécifions que chaque cellule (explorée) est soit libre (blanche), soit contient un obstacle (grise); les vrais algorithmes SLAM utilisent une représentation probabiliste (Sect. 9.2).

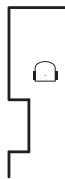


FIG. 9.17 – Un robot près du mur d'une pièce

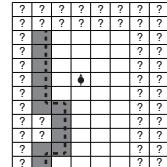


FIG. 9.18 – La carte correspondante

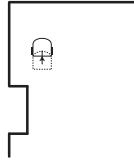


FIG. 9.19 – Le mouvement prévu du robot

?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?
?	*					?	?
?	*					?	?
?	*					?	?
?	*					?	?
?	*	*				?	?
?	*	*				?	?
?	*	*				?	?
?	*	*				?	?

FIG. 9.20 – La carte du mouvement prévu

Supposons que le robot de la Fig. 9.17 ait l'intention de se déplacer vers l'avant jusqu'à la nouvelle position indiquée sur la Fig. 9.19. La figure 9.20 montre la carte correspondant à la position après le déplacement prévu, où le robot s'est déplacé de la hauteur d'une cellule vers le haut par rapport à sa position initiale. Malheureusement, la roue droite se déplace sur une zone de faible friction et, bien que le robot se retrouve à la bonne position, son cap est trop à droite. La position réelle du robot est représentée sur la Fig. 9.21 et la Fig. 9.22 est la carte correspondante.

La figure 9.23 (qui est la même que la figure 9.20) montre la perception *intended* du robot parce que le robot s'est déplacé d'une cellule vers le haut par rapport à la carte de la figure 9.18. Dans cette position, il peut détecter l'obstacle à sa gauche et examiner les cellules inconnues devant lui.

Cependant, en raison de l'erreur d'odométrie, la perception *actuelle* du robot est différente. La figure 9.24 montre la position réelle du mur telle qu'elle est vue par le robot, superposée aux cellules, où les cellules sont colorées en gris si l'on sait que la majorité de leur zone se trouve derrière le mur. (Nous supposons que le robot peut détecter les murs à une distance allant jusqu'à cinq fois la taille d'une cellule, comme le montre le cercle en pointillé, et nous supposons également que le robot sait que tout mur a une épaisseur d'une cellule).

Il y a un décalage évident entre la carte actuelle et les données du capteur qui devraient correspondre à la partie connue de la carte. De toute évidence, le robot ne se trouve pas à l'endroit où il devrait être d'après l'odométrie. Comment ce décalage peut-il être corrigé ? Nous supposons que l'odométrie donne une estimation raison-

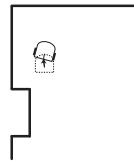


FIG. 9.21 – Le mouvement réel du robot

?	?	?	?	?	?	?	?
?	*					?	?
?	*					?	?
?	*					?	?
?	*					?	?
?	*					?	?
?	*	*				?	?
?	*	*				?	?
?	*	*				?	?
?	*	*				?	?

FIG. 9.22 – La carte du mouvement réel

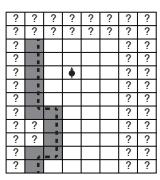


FIG. 9.23 – La perception prévue du robot

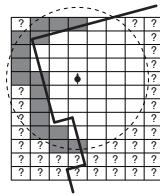


FIG. 9.24 – La perception réelle du robot

nable de la pose (position et cap) du robot. Pour chaque erreur possible relativement faible dans la pose, nous calculons ce que serait la perception de la carte actuelle et la comparons à la perception réelle calculée à partir des données du capteur. La pose qui donne la meilleure correspondance est choisie comme la pose réelle du robot et la carte actuelle est mise à jour en conséquence.

Dans l'exemple, on suppose que le robot se trouve soit dans la cellule attendue, soit dans l'une de ses quatre cellules voisines (gauche, droite, haut, bas) et que le cap du robot est soit correct, soit légèrement tourné vers la droite (15° dans le sens des aiguilles d'une montre (CW)) ou légèrement tourné vers la gauche (15° dans le sens inverse des aiguilles d'une montre (CCW)). Les $5 \times 3 = 15$ poses possibles sont illustrées dans la Fig. 9.25 et la Fig. 9.26 montre la perception de la carte calculée à partir de la carte actuelle pour chaque pose. (Pour gagner de la place, seul un fragment de 8×5 de la carte de 12×8 est affiché).

L'étape suivante consiste à choisir la carte qui correspond le mieux aux mesures du capteur. Transformons d'abord les cartes 8×5 en matrices 8×5 , en attribuant -1 aux cellules vides, $+1$ aux cellules d'obstacles et 0 aux autres cellules. La matrice de gauche de la figure 9.27 est associée à la carte actuelle et la matrice centrale de la figure est associée à la carte de perception correspondant à la pose où le robot est dans la cellule correcte mais le cap est 15° *irc* CW (l'élément central de la rangée supérieure de la figure 9.26).

Pour comparer les cartes, multipliez les éléments des cellules correspondantes. Soit $m(i, j)$ la (i, j) 'th cell de la carte actuelle et $p(i, j)$ la (i, j) 'th cell de la carte de perception obtenue à partir des valeurs des capteurs. $S(i, j)$, l'*similarité* de la (i, j) ème cellule, est :

$$S(i, j) = m(i, j) p(i, j),$$

qui peut également être exprimée comme suit

$$\begin{aligned} S(i, j) &= 1 && \text{if } m(i, j) \neq 0, p(i, j) \neq 0, m(i, j) = p(i, j) \\ S(i, j) &= -1 && \text{if } m(i, j) \neq 0, p(i, j) \neq 0, m(i, j) \neq p(i, j) \\ S(i, j) &= 0 && \text{if } m(i, j) = 0 \text{ or } p(i, j) = 0. \end{aligned}$$

La matrice de droite dans la Fig.9.27 montre le résultat de ce calcul pour les deux

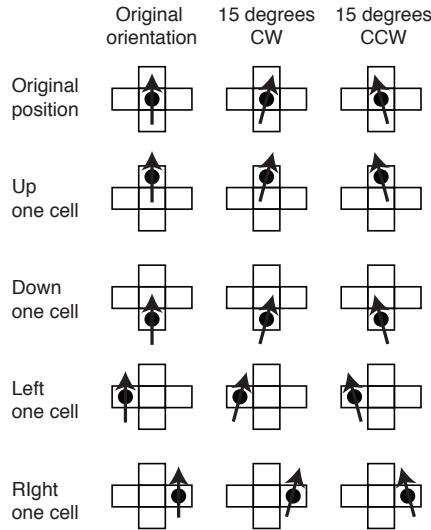


FIG. 9.25 – Positions possibles du robot

matrices à sa gauche. Il y a beaucoup de 1, ce qui signifie que les matrices sont similaires et que nous pouvons donc conclure que les cartes de perception sont similaires. Pour obtenir un résultat quantitatif, calculez la somme des similitudes afin d'obtenir une valeur unique pour toute paire m, p :

$$\mathcal{S} = \sum_{i=1}^8 \sum_{j=1}^5 S(i, j).$$

Le tableau 9.1 donne les valeurs de la similarité \mathcal{S} pour toutes les cartes de perception de la Fig. 9.26 par rapport à la carte actuelle. Comme prévu, la similarité la plus élevée est obtenue pour la carte correspondant à la pose avec la position correcte et le cap tourné de 15° CW.

Une fois ce résultat obtenu, nous corrigons la pose du robot et utilisons les données de la carte de perception pour mettre à jour la carte actuelle stockée dans la mémoire du robot (Fig. 9.28).

9.6 Activités de démonstration de l'algorithme SLAM

Les deux activités suivantes démontrent certains aspects de l'algorithme SLAM. L'activité 9.4 suit l'algorithme et est destinée à être mise en œuvre dans un logiciel. L'activité 9.5 démontre un élément clé de l'algorithme qui peut être mis en œuvre sur un robot éducatif.

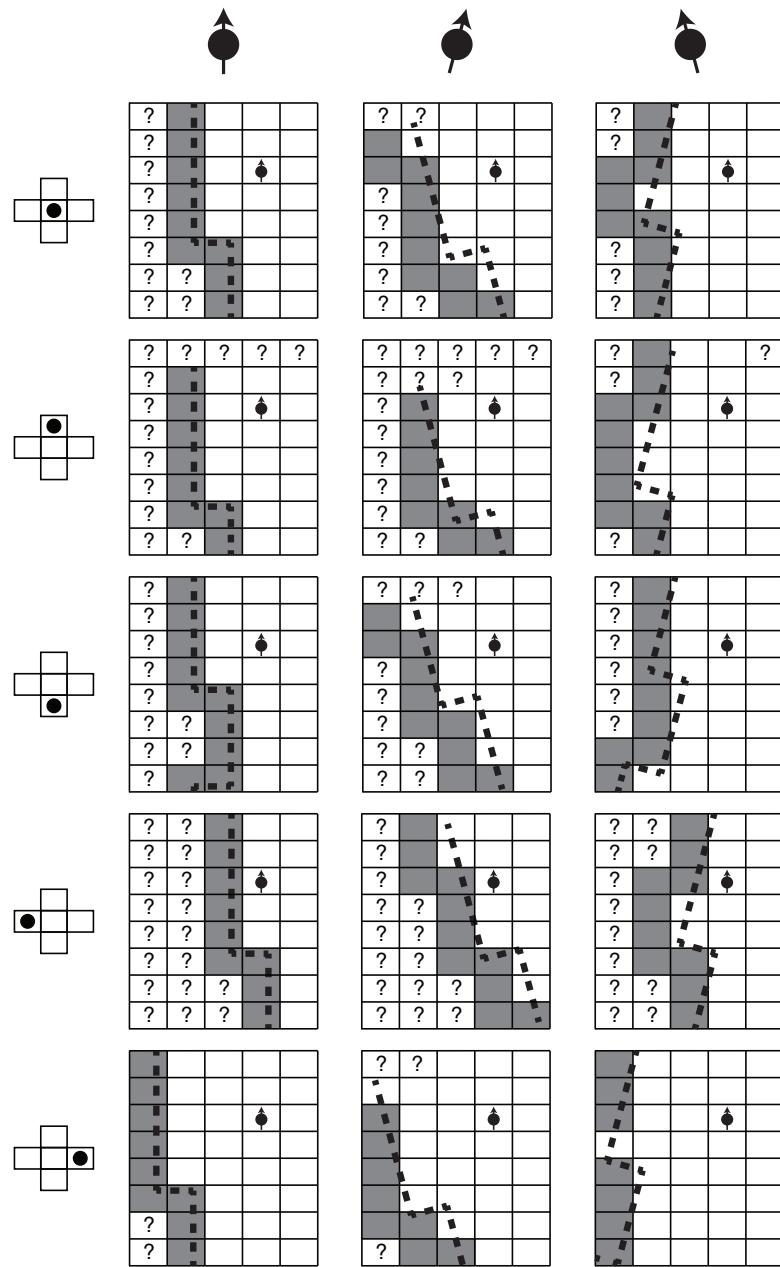


FIG. 9.26 – Estimations de la perception du robot pour différentes poses

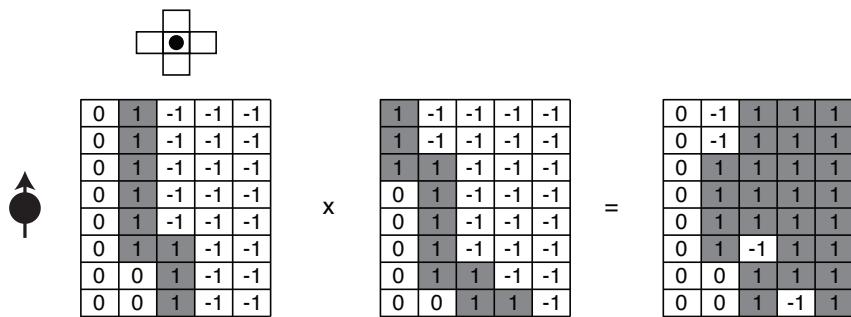


FIG. 9.27 – Calcul de la correspondance entre deux cartes

TAB. 9.1 – Similarité S de la carte basée sur le capteur avec la carte actuelle

	Orientation prévue	15° CW	15° CCW
Position prévue	22	32	20
Remonter d'une cellule	23	25	16
En bas d'une cellule	19	28	21
Gauche d'une cellule	6	7	18
Une cellule à droite	22	18	18

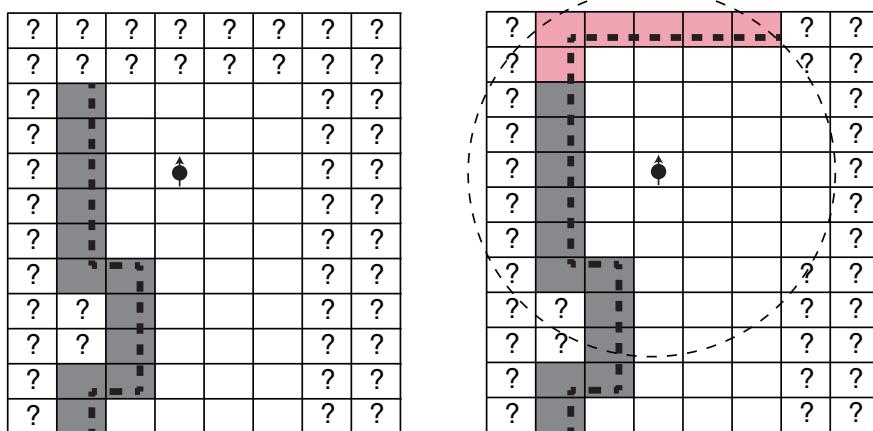


FIG. 9.28 – Carte avant et après la mise à jour en utilisant les données de la carte de perception

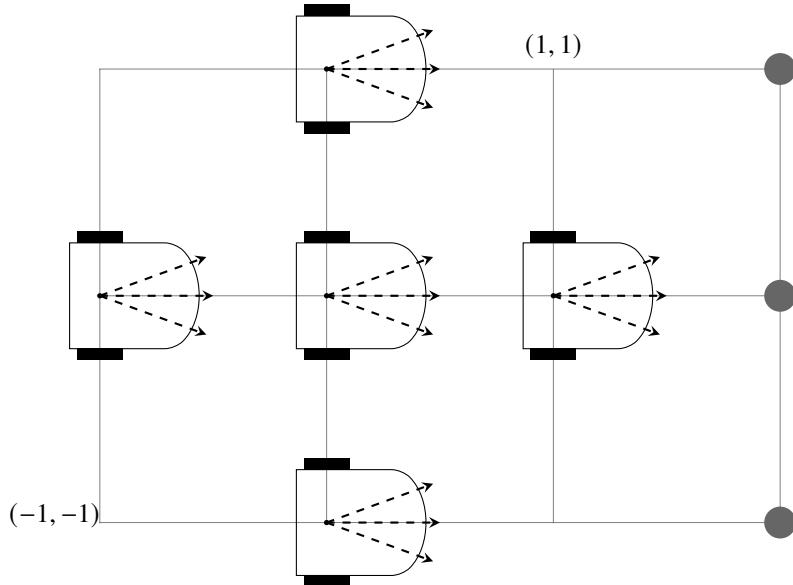


FIG. 9.29 – Configuration de l'algorithme SLAM

Les activités sont basées sur la configuration présentée dans la Fig. 9.29. Le robot est situé à l'origine du système de coordonnées avec la pose $((x, y), \theta) = ((0, 0), 0^\circ)$.¹ Compte tenu de l'incertitude de l'odométrie, le robot peut en fait être situé à n'importe laquelle des coordonnées $(-1, 0), (1, 0), (0, -1), (0, 1)$ et son orientation peut être $-15^\circ, 0, -15^\circ$ (comme le montrent les flèches en pointillés), ce qui donne 15 poses possibles. Les trois points gris aux coordonnées $(2, 2), (2, 0), (2, -2)$ représentent des obstacles connus sur la carte actuelle. (Pour gagner de la place, les points sont affichés aux coordonnées $(2, 1), (2, 0), (2, -1)$). Les obstacles peuvent être détectés par plusieurs capteurs de proximité horizontaux, mais pour les besoins des activités, nous spécifions qu'il y a trois capteurs.

Dans l'algorithme SLAM, la "perception" d'un obstacle est la valeur renvoyée par un capteur de distance. Pour éviter d'avoir à définir un modèle pour les capteurs, les activités définiront une perception comme étant la *distance* et la *orientation* entre le capteur et l'obstacle.

Activity 9.4: Localiser le robot à partir des perceptions calculées

- Pour chacune des 15 positions, calculer l'ensemble des perceptions de chaque obstacle. Par exemple, si le robot se trouve à la position

1. Il est pratique de prendre le cap du robot comme 0° .

$((0.0, 1.0), -15.0^\circ)$, l'ensemble des perceptions des trois obstacles est :

$$[(2.2, 41.6^\circ), (2.2, -11.6^\circ), (3.6, -41.3^\circ)].$$

- Étant donné un ensemble de perceptions mesurées, calculer leurs similitudes avec les perceptions des 15 poses du robot. Choisir la pose ayant la meilleure similarité comme étant la pose réelle du robot. Par exemple, pour l'ensemble des perceptions mesurées :

$$[(2.0, 32.0^\circ), (2.6, -20.0^\circ), (3.0, -30.0^\circ)],$$

et la similarité calculée comme la somme des différences absolues des éléments des perceptions, la pose avec la meilleure similarité est $((0.0, 1.0), -15.0^\circ)$.

- Expérimenter avec différentes fonctions de similarité.
- Nous avons calculé que la pose du robot est approximativement $((0.0, 1.0), -15.0^\circ)$. Supposons qu'un nouvel obstacle soit placé aux coordonnées $(3, 0)$. Calculer la perception (d, θ) de l'objet à partir de cette pose, puis calculer la coordonnée (x, y) du nouvel obstacle. Le nouvel obstacle peut être ajouté à la carte avec cette coordonnée.
- La coordonnée calculée est-elle significativement différente de la coordonnée $(3, 0)$ qui serait obtenue si le robot se trouvait à la position prévue $((0, 0), 0^\circ)$?

Activity 9.5: Localiser le robot à partir des perceptions mesurées

- Placer trois objets comme indiqué sur la Fig. 9.29.
- Écrire un programme qui mémorise l'ensemble des valeurs renvoyées par les trois capteurs de proximité horizontaux. Placez votre robot successivement dans chacune des 15 positions et enregistrez les ensembles de valeurs. Vous disposez maintenant d'une base de données de perceptions : un ensemble de trois lectures de capteurs pour chaque pose.
- Placez le robot dans l'une des positions et enregistrez l'ensemble des valeurs renvoyées par les capteurs. Calculez la similarité de cet ensemble avec chacun des ensembles de la base de données. Afficher la position associée à la meilleure similarité.
- Expérimenez en plaçant le robot dans différentes positions et plusieurs fois dans chaque position. Quelle est la précision de la détermination de la pose ?
- Expérimenter avec différentes fonctions de similarité.

Algorithm 9.2: SLAM	
matrix \vec{m} \leftarrow partial map	// Current map
matrix \vec{p}	// Perception map
matrix \vec{e}	// Expected map
coordinate $\vec{c} \leftarrow$ initial position	// Current position
coordinate \vec{n}	// New position
coordinate array \vec{T}	// Set of test positions
coordinate \vec{t}	// Test position
coordinate $\vec{b} \leftarrow$ none	// Best position
1: loop	
2: move a short distance	
3: $\vec{n} \leftarrow$ odometry(\vec{c})	// New position based on odometry
4: $\vec{p} \leftarrow$ analyze sensor data	
5: for every \vec{t} in \vec{T}	// T is the positions around n
6: $\vec{e} \leftarrow$ expected(\vec{m}, \vec{t})	// Expected map at test position
7: if compare(\vec{t}, \vec{e}) better than \vec{b}	
8: $\vec{b} \leftarrow \vec{t}$	// Best test position so far
9: $\vec{n} \leftarrow \vec{b}$	// Replace new position by best position
10: $\vec{m} \leftarrow$ update($\vec{m}, \vec{p}, \vec{n}$)	// Update map // based on new position
11: $\vec{c} \leftarrow \vec{n}$	// Current position is new position

9.7 La formalisation de l'algorithme SLAM

L'algorithme 9.2 est un algorithme SLAM qui trouve la position dont la carte de perception est la plus proche de la carte de perception obtenue à partir des données des capteurs. Le robot est localisé à cette position et la carte est mise à jour en fonction de ce qui est perçu à cette position.

L'algorithme est divisé en trois phases. Dans la première phase (lignes 2–4), le robot se déplace sur une courte distance et sa nouvelle position est calculée par odométrie. La carte de perception à cet endroit est obtenue en analysant les données

des capteurs.

En supposant que l'erreur d'odométrie est relativement faible, nous pouvons définir un ensemble de positions de test où le robot pourrait se trouver. Dans la deuxième phase (lignes 5–8), la carte attendue à chacune de ces positions est calculée et comparée à la carte actuelle. La meilleure correspondance est sauvegardée.

Dans la troisième phase (lignes 9–11), la position avec la meilleure correspondance devient la nouvelle position et la carte actuelle est mise à jour en conséquence.

En pratique, l'algorithme est un peu plus compliqué car il doit tenir compte du fait que la carte de perception obtenue à partir des capteurs est limitée par la portée de ces derniers. Le chevauchement sera partiel à la fois parce que la portée des capteurs ne couvre pas la totalité de la carte actuelle et parce que les capteurs peuvent détecter des obstacles et des zones libres en dehors de la carte actuelle. Par conséquent, la taille de la carte perçue \vec{p} sera beaucoup plus petite que la carte attendue \vec{e} et la fonction $\text{compare}(\vec{p}, \vec{e})$ ne comparera que les zones qui se chevauchent. En outre, lors de la mise à jour de la carte actuelle, les zones qui n'y figuraient pas auparavant seront ajoutées. Sur la figure 9.28, certaines cellules de la carte actuelle se trouvent en dehors du rayon de cinq cellules du capteur et ne seront pas mises à jour. Les cellules rouge clair étaient inconnues dans la carte actuelle, comme l'indiquent les points d'interrogation, mais dans la carte de perception, on sait maintenant qu'elles font partie de l'obstacle. Cette information est utilisée pour mettre à jour la carte actuelle afin d'obtenir une nouvelle carte actuelle.

9.8 Résumé

Un mouvement robotique précis dans un environnement incertain exige que le robot dispose d'une carte de l'environnement. La carte doit être conservée dans l'ordinateur du robot; il peut s'agir d'une grille de cellules ou d'une représentation graphique d'une carte continue. Dans un environnement incertain, le robot ne dispose généralement pas d'une carte avant de commencer ses tâches. L'algorithme de frontière est utilisé par un robot pour construire une carte au fur et à mesure qu'il explore son environnement. Des cartes plus précises peuvent être construites si le robot a une certaine connaissance de son environnement, par exemple s'il sait qu'il s'agit de l'intérieur d'un bâtiment composé de pièces rectangulaires et de couloirs. Les algorithmes de localisation et de cartographie simultanées (SLAM) utilisent un processus itératif pour construire une carte tout en corrigeant les erreurs de localisation.

9.9 Lecture complémentaire

Deux manuels sur la planification des trajectoires et des mouvements sont [31, 32]. Voir également [43, Chapitre 6].

L'algorithme de frontière a été proposé par Yamauchi [51] qui a montré qu'un robot pouvait utiliser l'algorithme pour explorer avec succès un bureau avec des obstacles.

Les algorithmes de SLAM utilisent les probabilités, en particulier la règle de Bayes. Les méthodes probabilistes en robotique font l'objet du manuel [46].

Un tutoriel en deux parties sur le SLAM, rédigé par Durrant-Whyte et Bailey, est disponible dans [15, 4]. Un tutoriel sur la SLAM basée sur les graphes est disponible dans [18].

Le cours en ligne de Sebastian Thrun *Artificial Intelligence for Robotics* est utile : <https://classroom.udacity.com/courses/cs373>.

Chapitre 10

Navigation basée sur des cartes

Maintenant que nous disposons d'une carte, qu'elle soit fournie par l'utilisateur ou découverte par le robot, nous pouvons discuter de *la planification des itinéraires*, un algorithme de plus haut niveau. Prenons l'exemple d'un robot utilisé dans un hôpital pour transporter des médicaments et d'autres fournitures des zones de stockage vers les médecins et les infirmières. Étant donné l'une de ces tâches, quelle est la meilleure façon d'aller du point A au point B. Il peut y avoir plusieurs façons de se déplacer dans les couloirs pour atteindre l'objectif, mais il peut aussi y avoir des chemins courts que le robot n'est pas autorisé à emprunter, par exemple des chemins qui passent par les couloirs près des salles d'opération.

Nous présentons trois algorithmes de planification du chemin le plus court entre une position de départ S et une position d'arrivée G , en supposant que nous disposons d'une carte de la zone indiquant la position des obstacles qui s'y trouvent. Edsger W. Dijkstra, l'un des pionniers de l'informatique, a proposé un algorithme pour le problème du plus court chemin. La section 10.1 décrit l'algorithme pour une carte quadrillée, tandis que la section 10.2 décrit l'algorithme pour une carte continue. L'algorithme 10.3, une amélioration de l'algorithme de Dijkstra basée sur des méthodes heuristiques, est présenté dans la section refs.astar. Enfin, la section 10.4 explique comment combiner un algorithme de planification de chemin de haut niveau avec un algorithme d'évitement d'obstacles de bas niveau.

10.1 Algorithme de Dijkstra pour une carte quadrillée

Dijkstra a décrit son algorithme pour un graphe discret de noeuds et d'arêtes. Nous le décrivons ici pour une grille de cellules (Fig. 10.1). La cellule S est la cellule de départ du robot et sa tâche consiste à se déplacer jusqu'à la cellule d'arrivée G . Les cellules contenant des obstacles sont représentées en noir. Le robot peut détecter un *voisin* de la cellule c qu'il occupe et s'y déplacer. Par souci de simplicité, nous précisons que les cellules voisines de c sont les quatre cellules qui lui sont adjacentes horizontalement et verticalement, mais pas en diagonale. La figure 10.2 montre le chemin le plus court entre S et G :

$$(4, 0) \rightarrow (4, 1) \rightarrow (3, 1) \rightarrow (2, 1) \rightarrow (2, 2) \rightarrow \\ (2, 3) \rightarrow (3, 3) \rightarrow (3, 4) \rightarrow (3, 5) \rightarrow (4, 5).$$

Deux versions de l'algorithme sont présentées : La première concerne les grilles où le coût de déplacement d'une cellule à l'une de ses voisines est constant. Dans la

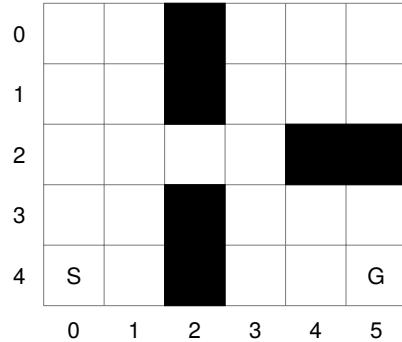


FIG. 10.1 – Carte de la grille de l'algorithme de Dijkstra

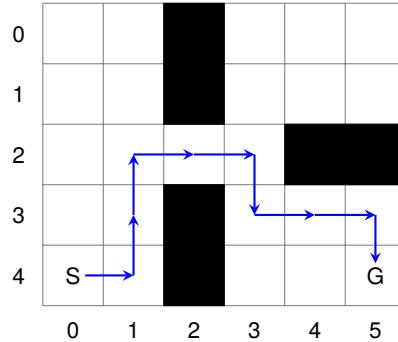


FIG. 10.2 – Le plus court chemin trouvé par l'algorithme de Dijkstra

seconde version, chaque cellule peut avoir un coût différent associé à son déplacement, de sorte que le chemin le plus court géométriquement n'est pas nécessairement le plus court lorsque les coûts sont pris en compte.

10.1.1 Algorithme de Dijkstra sur une carte quadrillée à coût constant

L'algorithme 10.1 est l'algorithme de Dijkstra pour une carte quadrillée. L'algorithme est démontré sur la grille de 5×6 cellules de la Fig. 10.3. Trois obstacles sont représentés par les cellules noires.

L'algorithme marque chaque cellule c du nombre d'étapes nécessaires pour atteindre c à partir de la cellule de départ S . Dans les figures, le nombre d'étapes est représenté par un chiffre dans le coin supérieur gauche d'une cellule. Dans un premier temps, la cellule S est marquée par 0 car aucune étape n'est nécessaire pour atteindre S à partir de S . Ensuite, marquez chaque voisin de S avec 1 puisqu'ils sont à un pas de S ; puis marquez chaque voisin d'une cellule marquée 1 avec 2. La figure 10.4 montre la carte de la grille après ces deux itérations de l'algorithme.

L'algorithme se poursuit de manière itérative : si une cellule est marquée n , ses voisins non marqués sont marqués $n + 1$. Lorsque G est finalement marquée, nous savons que la distance la plus courte entre S et G est de n . La figure 10.5 montre la carte de la grille après cinq itérations et la figure 10.6 montre la carte finale de la grille après neuf itérations, lorsque la cellule cible a été atteinte.

Il est maintenant facile de trouver le chemin le plus court en travaillant à rebours à partir de la cellule cible G . Dans la figure 10.6, le chemin le plus court est constitué des cellules colorées en gris. En partant de la cellule but à la coordonnée (4, 5), la cellule précédente doit être soit (4, 4), soit (3, 5), puisqu'elles sont à huit pas du point de départ. (Nous choisissons arbitrairement (3, 5). A partir de chaque cellule sélectionnée marquée n , nous choisissons une cellule marquée $n - 1$ jusqu'à ce que

Algorithm 10.1: Algorithme de Dijkstra sur une carte quadrillée

```

integer n ← 0           // Distance from start
cell array grid ← all unmarked // Grid map
cell list path ← empty      // Shortest path
cell current               // Current cell in path
cell c                      // Index over cells
cell S ← ...                // Source cell
cell G ← ...                // Goal cell

1: mark S with n
2: while G is unmarked
3:   n ← n + 1
4:   for each unmarked cell c in grid
5:     next to a marked cell
6:     mark c with n
7:   current ← G
8:   append current to path
9:   while S not in path
10:    append lowest marked neighbor c
11:      of current to path
12:    current ← c

```

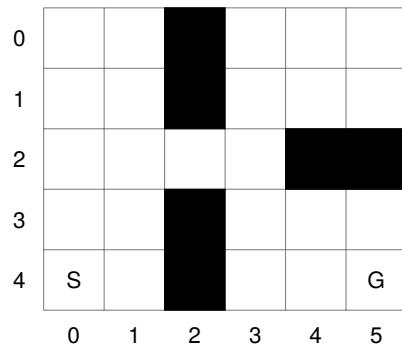


FIG. 10.3 – Carte quadrillée pour l'algorithme de Dijkstra

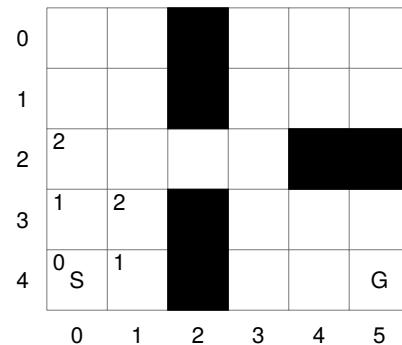


FIG. 10.4 – Les deux premières itérations de l'algorithme de Dijkstra

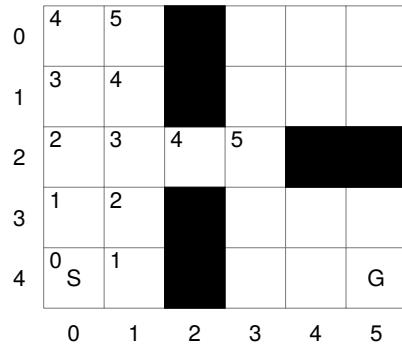


FIG. 10.5 – Après cinq itérations de l'algorithme de Dijkstra

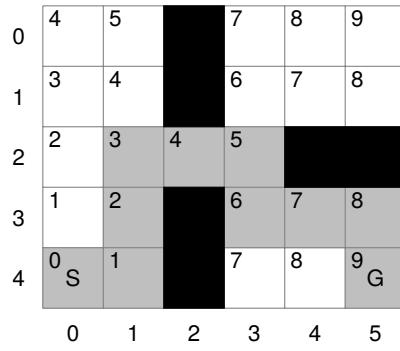


FIG. 10.6 – La carte quadrillée finale avec le chemin le plus court marqué

la cellule S marquée 0 soit sélectionnée. La liste des cellules sélectionnées est la suivante :

$$(4,5), (3,5), (3,4), (3,3), (2,3), (2,2), (2,1), (3,1), (4,1), (4,0).$$

En inversant la liste, on obtient le plus court chemin de S à G. Vérifiez qu'il s'agit du même chemin que celui que nous avons trouvé intuitivement (Fig. 10.2).

Exemple La figure 10.7 montre comment l'algorithme de Dijkstra fonctionne sur un exemple plus compliqué. La carte quadrillée comporte 16 fois 16 cellules et la cellule cible G est entourée d'un obstacle et difficile à atteindre. Le diagramme supérieur gauche montre la carte de la grille après trois itérations et le diagramme supérieur droit montre la carte après 19 itérations. L'algorithme procède maintenant à l'exploration des cellules autour des deux côtés de l'obstacle. Enfin, dans le diagramme inférieur gauche, G est trouvé après 25 itérations, et le chemin le plus court est indiqué en gris dans le diagramme inférieur droit. Nous voyons que l'algorithme n'est pas très efficace pour cette carte : bien que le chemin le plus court ne représente que 25 étapes, l'algorithme a exploré $256 - 25 = 231$ cellules !

10.1.2 Algorithme de Dijkstra avec coûts variables

L'algorithme 10.1 et l'exemple de la Fig. 10.7 supposent que le coût d'un pas d'une cellule à l'autre est constant : la ligne trois de l'algorithme ajoute 1 au coût pour chaque voisin. L'algorithme de Dijkstra peut être modifié pour prendre en compte un coût variable pour chaque étape. Supposons qu'une zone de l'environnement soit recouverte de sable et qu'il soit plus difficile pour le robot de se déplacer dans cette zone. Dans l'algorithme, au lieu d'ajouter 1 au coût de chaque cellule voisine, nous pouvons ajouter k à chaque cellule sablonneuse voisine pour refléter le coût supplémentaire.

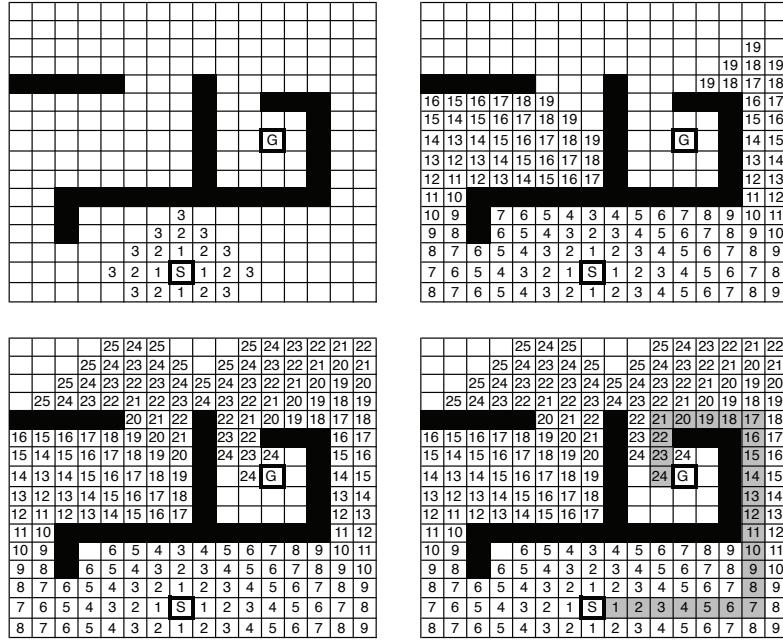


FIG. 10.7 – Algorithme de Dijkstra pour la planification d'un chemin sur une carte quadrillée. Les quatre étapes de l'exécution de l'algorithme sont représentées à partir du diagramme supérieur gauche.

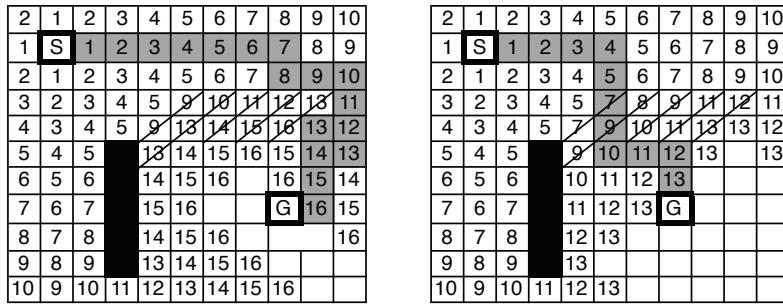


FIG. 10.8 – Algorithme de Dijkstra avec un coût variable par cellule (diagramme de gauche, coût=4, diagramme de droite, coût=2)

La grille de gauche de la Fig. 10.8 comporte quelques cellules marquées d'une ligne diagonale pour indiquer qu'elles sont recouvertes de sable et que le coût du déplacement à travers elles est de 4 et non de 1. Le chemin le plus court, marqué en gris, comporte 17 étapes et coûte également 17 puisqu'il contourne le sable.

Le chemin le plus court dépend du coût attribué à chaque cellule. Le diagramme de droite montre le chemin le plus court si le coût du passage dans une cellule contenant du sable n'est que de 2. Le chemin fait 12 pas, mais son coût est de 14 pour tenir compte du fait qu'il faut faire deux pas dans le sable.

Activity 10.1: Algorithme de Dijkstra sur une carte quadrillée

- Construire une carte quadrillée et appliquer l'algorithme de Dijkstra.
- Modifier la carte pour inclure des cellules à coût variable et appliquer l'algorithme.
- Mettre en œuvre l'algorithme de Dijkstra.
- Créer une grille sur le sol.
- Ecrire un programme qui permet au robot de se déplacer d'une cellule de départ connue à une cellule d'arrivée connue. Puisque le robot doit mémoriser sa position actuelle, utilisez-la pour créer une carte des cellules qu'il a traversées.
- Placez quelques obstacles dans la grille et inscrivez-les dans la carte du robot.

10.2 Algorithme de Dijkstra pour une carte continue

Dans une carte continue, la zone est un plan géométrique ordinaire à deux dimensions. Une approche de l'utilisation de l'algorithme de Dijkstra dans une carte continue consiste à transformer la carte en un graphe discret en traçant des lignes verticales à partir des bords supérieur et inférieur de l'environnement jusqu'à chaque coin d'un obstacle. La zone est ainsi divisée en un nombre fini de segments, chacun d'entre eux pouvant être représenté comme un nœud dans un graphe. Le diagramme de gauche de la Fig. 10.9 montre sept lignes verticales qui divisent la carte en dix segments qui sont représentés dans le graphique de la Fig. 10.10. Les arêtes du graphique sont définies par la relation d'adjacence des segments. Il existe une arête dirigée du segment A vers le segment B si A et B partagent une frontière commune. Par exemple, il existe des arêtes entre le nœud 2 et les nœuds 1 et 3 puisque les segments correspondants partagent une arête avec le segment 2.

Quel est le chemin le plus court entre le sommet 2 représentant le segment contenant le point de départ et le sommet 10 représentant le segment contenant le but ? Le résultat de l'application de l'algorithme de Dijkstra est $S \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 10 \rightarrow G$. Bien qu'il s'agisse du chemin le plus court en termes de nombre

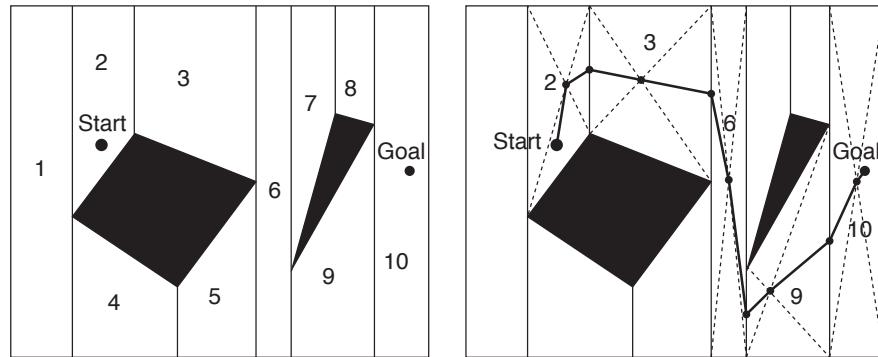


FIG. 10.9 – Segmentation d'une carte continue par des lignes verticales et chemin à travers les segments

d'arêtes du graphe, ce n'est pas le chemin le plus court dans l'environnement. La raison en est que nous avons attribué un coût constant à chaque arête, bien que les segments de la carte soient de tailles différentes.

Étant donné que chaque sommet représente un grand segment de l'environnement, nous devons savoir comment le déplacement d'un sommet à un autre se traduit par le déplacement d'un segment à un autre. Le diagramme de droite de la figure 10.9 montre une possibilité : chaque segment est associé à son centre géométrique, indiqué par l'intersection des lignes diagonales en pointillés dans la figure. Le chemin dans l'environnement associé au chemin dans le graphique va du centre d'un segment au centre du segment suivant, sauf que les positions de départ et d'arrivée sont à leurs emplacements géométriques. Bien que cette méthode soit raisonnable sans connaissance supplémentaire de l'environnement, elle ne donne pas le chemin optimal qui doit rester proche des limites des obstacles.

La figure 10.11 montre une autre approche de la planification des chemins dans une carte continue. Elle utilise un *graphique de visibilité*, où chaque sommet du graphique représente un coin d'un obstacle, et il y a des sommets pour les positions de départ et d'arrivée. Il existe une arête entre le sommet v_1 et le sommet v_2 si les

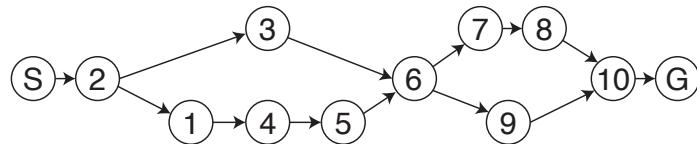


FIG. 10.10 – Le graphique construit à partir de la carte continue segmentée

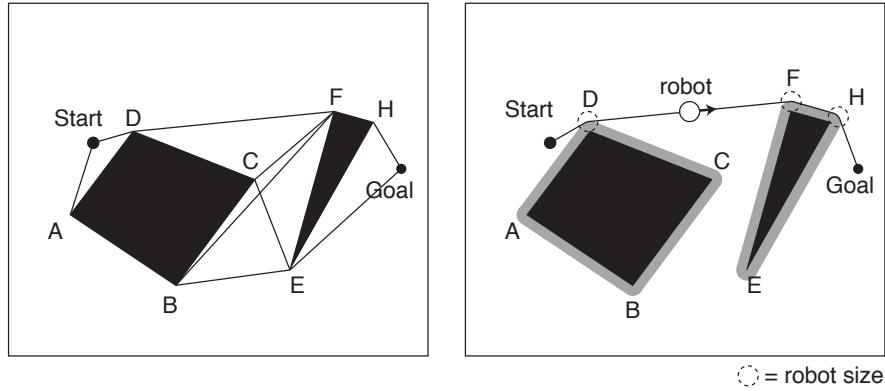


FIG. 10.11 – Une carte continue avec des lignes d'un coin à l'autre et le chemin à travers les coins.

coins correspondants sont visibles. Par exemple, il y a une arête $C \rightarrow E$ parce que le coin E de l'obstacle de droite est visible depuis le coin C de l'obstacle de gauche. La figure 10.12 montre le graphe formé par ces nœuds et ces arêtes. Il représente tous les candidats au chemin le plus court entre le point de départ et le point d'arrivée.

Il est facile de voir que les chemins dans le graphique représentent des chemins dans l'environnement, puisque le robot peut simplement se déplacer d'un coin à l'autre. Ces chemins sont les plus courts car aucun chemin, disons de A à B , ne peut être plus court que la ligne droite de A à B . L'algorithme de Dijkstra donne le chemin le plus court sous la forme $S \rightarrow D \rightarrow F \rightarrow H \rightarrow G$. Dans ce cas, le chemin le plus court en termes de nombre d'arêtes est également le chemin le plus court géométriquement.

Bien qu'il s'agisse du chemin le plus court, un robot réel ne peut pas suivre ce chemin parce qu'il a une taille finie et que son centre ne peut pas suivre le bord d'un obstacle. Le robot doit maintenir une distance minimale par rapport à chaque obstacle, ce qui peut être réalisé en augmentant la taille des obstacles de la taille du

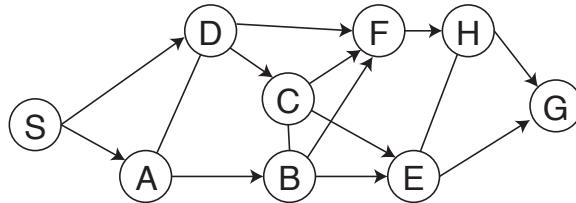


FIG. 10.12 – Le graphe construit à partir de la carte continue segmentée

robot (diagramme de droite de la Fig. 10.12). Le chemin obtenu est optimal et peut être parcouru par le robot.

Activity 10.2: Algorithme de Dijkstra pour les cartes continues

- Dessinez une version plus grande de la carte de la Fig. 10.11. Mesurez la longueur de chaque segment. Appliquez maintenant l'algorithme de Dijkstra pour déterminer le chemin le plus court.
- Créez votre propre carte continue, extrayez le graphe de visibilité et appliquez l'algorithme de Dijkstra.

10.3 Planification de trajectoire avec l'algorithme A*

L'algorithme de Dijkstra recherche la cellule cible dans toutes les directions ; cela peut être efficace dans un environnement complexe, mais ce n'est pas le cas lorsque le chemin est simple, par exemple une ligne droite jusqu'à la cellule cible. Regardez le diagramme en haut à droite de la figure 10.7 : près du coin supérieur droit de l'obstacle central, il y a une cellule à une distance de 19 de la cellule de départ. Après deux pas supplémentaires vers la gauche, il y aura une cellule marquée 21 qui a un chemin vers la cellule de but qui n'est pas bloqué par un obstacle. Il n'y a manifestement aucune raison de continuer à explorer la région située à gauche de la grille, mais l'algorithme de Dijkstra continue à le faire. Il serait plus efficace si l'algorithme savait d'une manière ou d'une autre qu'il est proche de la cellule cible.

L'algorithme A* (prononcé "étoile") est similaire à l'algorithme de Dijkstra, mais il est souvent plus efficace car il utilise des informations supplémentaires pour guider la recherche. L'algorithme A* prend en compte non seulement le nombre de pas à partir de la cellule de départ, mais aussi une *fonction heuristique* qui donne une indication de la direction préférée de la recherche. Auparavant, nous utilisions une fonction de coût $g(x, y)$ qui donne le nombre réel de pas à partir de la cellule de départ. L'algorithme de Dijkstra étend la recherche en commençant par les cellules marquées par les valeurs les plus élevées de $g(x, y)$. Dans l'algorithme A*, la fonction de coût $f(x, y)$ est calculée en ajoutant les valeurs d'une fonction heuristique $h(x, y)$:

$$f(x, y) = g(x, y) + h(x, y).$$

Nous démontrons l'algorithme A* en utilisant comme fonction heuristique le nombre d'étapes entre la cellule cible G et la cellule (x, y) *sans tenir compte des obstacles*. Cette fonction peut être précalculée et reste disponible tout au long de l'exécution de l'algorithme. Pour la carte quadrillée de la Fig. 10.3, la fonction heuristique est présentée dans la Fig. 10.13.

Dans les diagrammes, nous garderons trace des valeurs des trois fonctions f, g, h

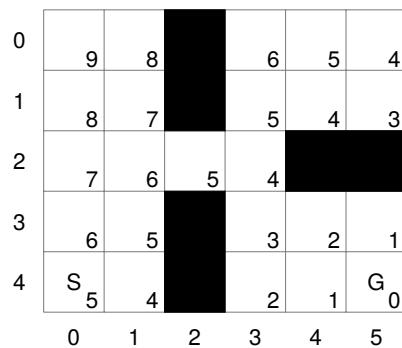


FIG. 10.13 – Fonction heuristique

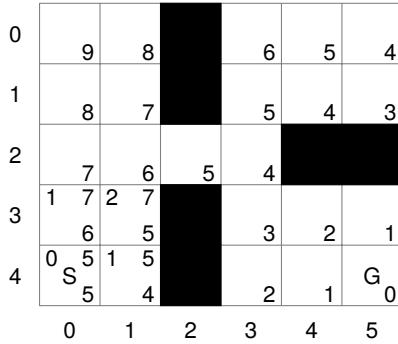


FIG. 10.14 – Les deux premières itérations de l'algorithme A*

en les affichant dans différents coins de chaque cellule :

$$\begin{array}{c} g \quad f \\ h \end{array}$$

La figure 10.14 montre la carte de la grille après deux étapes de l'algorithme A*. Les cellules (3, 1) et (3, 0) reçoivent le même coût f : l'une est plus proche de S (par le nombre d'étapes comptées) et l'autre est plus proche de G (par l'heuristique), mais toutes deux ont le même coût de 7.

L'algorithme doit maintenir une structure de données des *cellules ouvertes*, les cellules qui n'ont pas encore été développées. Nous utilisons la notation (r, c, v) , où r et c sont la ligne et la colonne de la cellule et v est la valeur f de la cellule. Chaque fois qu'une cellule ouverte est développée, elle est supprimée de la liste et les nouvelles cellules sont ajoutées. La liste est ordonnée de manière à ce que les cellules ayant les valeurs les plus faibles apparaissent en premier, ce qui facilite le choix de la cellule à développer ensuite. Les trois premières listes correspondant à la Fig. 10.14 sont :

$$\begin{aligned} &(4, 0, 5) \\ &(4, 1, 5), (3, 0, 7) \\ &(3, 0, 7), (3, 1, 7). \end{aligned}$$

La figure 10.15 montre la carte de la grille après six étapes. On peut s'en rendre compte en regardant les valeurs g dans le coin supérieur gauche de chaque cellule. La liste actuelle des cellules ouvertes est la suivante :

$$(3, 3, 9), (1, 0, 11), (1, 1, 11), (1, 3, 11).$$

L'algorithme A* choisit d'étendre la cellule (3, 3, 9) avec le f le plus bas. Les autres cellules de la liste ont une valeur f de 11 et sont ignorées, du moins pour l'instant. En continuant (Fig. 10.16), la cellule but est atteinte avec une valeur f de 9 et un plus court chemin en gris est affiché. La dernière liste avant d'atteindre le but est la

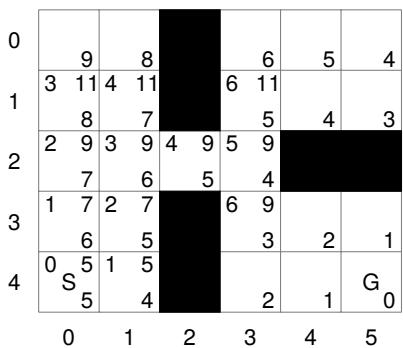


FIG. 10.15 – L'algorithme A* après 6 étapes

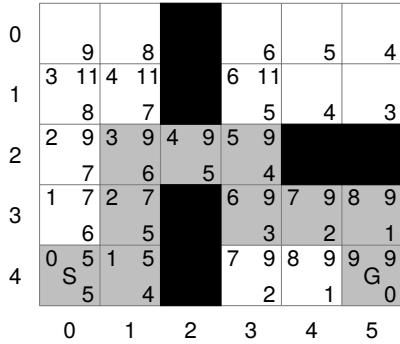


FIG. 10.16 – L'algorithme A* atteint la cellule cible et trouve le chemin le plus court

suivante :

$$(3, 5, 9), (4, 4, 9), (1, 0, 11), (1, 1, 11), (1, 3, 11).$$

Peu importe lequel des nœuds de valeur 9 est choisi : dans les deux cas, l'algorithme atteint la cellule but (4, 5, 9).

Toutes les cellules en haut à droite de la grille ne sont pas explorées car la cellule (1, 3) a une valeur f de 11 et ce ne sera jamais la plus petite valeur. Alors que l'algorithme de Dijkstra a exploré l'ensemble des 24 cellules non-obstacles, l'algorithme A* n'a exploré que 17 cellules.

Un exemple plus complexe de l'algorithme A*

Appliquons l'algorithme A* à la grille de la Fig. 10.8. Rappelons que cette grille comporte du sable sur certaines de ses cellules, de sorte que la fonction g donnera des valeurs plus élevées pour le coût de déplacement vers ces cellules. Le diagramme supérieur gauche de la figure 10.17 montre la fonction g telle qu'elle est calculée par l'algorithme de Dijkstra, tandis que le diagramme supérieur droit montre la fonction heuristique h , le nombre de pas à partir du but en l'absence d'obstacles et de sable. Le reste de la figure montre quatre étapes de l'algorithme conduisant au chemin le plus court vers le but.

Dès le diagramme du milieu à gauche, nous voyons qu'il n'est pas nécessaire de chercher en haut à gauche, car les valeurs f des cellules au-dessus et à gauche de S sont plus élevées que les valeurs à droite et en dessous de S. Dans le diagramme du milieu à droite, la première cellule de sable a une valeur de 13, de sorte que l'algorithme continue d'étendre les cellules ayant un coût inférieur de 12 vers la gauche. Dans le diagramme en bas à gauche, nous voyons que la recherche ne se poursuit pas jusqu'en bas à gauche de la carte parce que le coût de 16 est plus élevé

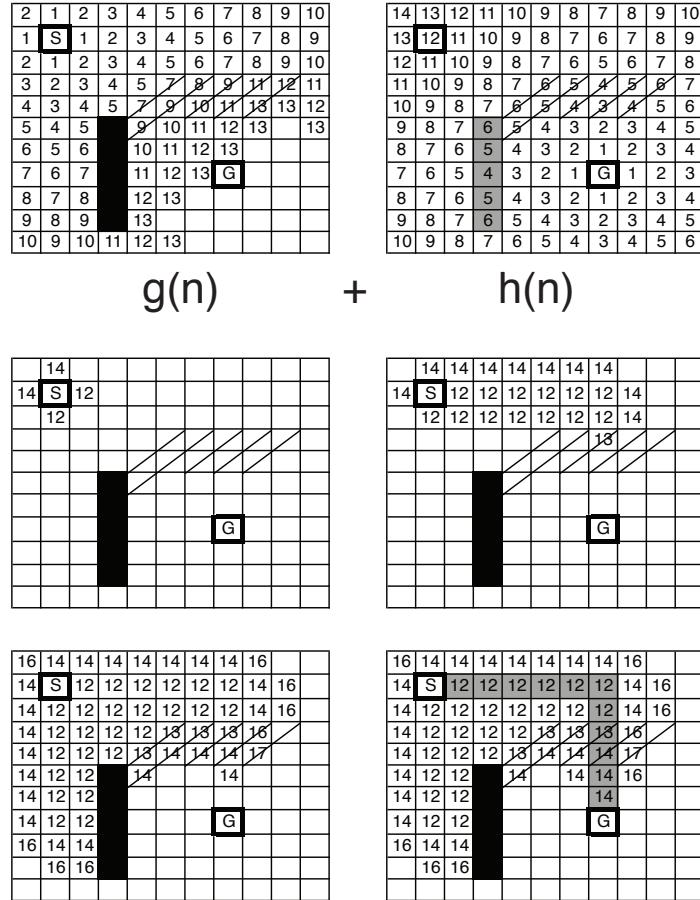


FIG. 10.17 – L'algorithme A*. En haut à gauche : le nombre d'étapes jusqu'à l'objectif. En haut à droite : la fonction heuristique. Les diagrammes du milieu et du bas montrent quatre étapes de l'algorithme.

que le coût de 14 une fois que la recherche quitte le sable. À partir de là, la cellule cible G est trouvée très rapidement. Comme dans l'algorithme de Dijkstra, le chemin le plus court est trouvé en repassant par les cellules ayant les valeurs g les plus faibles jusqu'à ce que la cellule de départ soit atteinte.

En comparant les figures 10.7 et 10.17, nous constatons que l'algorithme A* n'a eu besoin de visiter que 71% des cellules visitées par l'algorithme de Dijkstra. Bien que l'algorithme A* doive effectuer un travail supplémentaire pour calculer la fonction heuristique, le nombre réduit de cellules visitées rend l'algorithme plus efficace. En outre, cette fonction heuristique ne dépend que de la zone explorée et non des obstacles ; même si l'ensemble des obstacles est modifié, il n'est pas nécessaire de recalculer la fonction heuristique.

Activity 10.3: A* algorithm

- Appliquer l'algorithme A* et l'algorithme de Dijkstra sur une petite carte sans obstacles : placer la cellule de départ au centre de la carte et le but dans une cellule arbitraire. Comparez les résultats des deux algorithmes. Expliquez vos résultats. Le résultat dépend-il de la position de la cellule du but ?
- Définir d'autres fonctions heuristiques et comparer les résultats des algorithmes A* sur les exemples de ce chapitre.

10.4 Path following and obstacle avoidance

Ce chapitre et les précédents ont abordé deux tâches différentes mais liées : la planification de la trajectoire à haut niveau et l'évitement des obstacles à bas niveau. Comment intégrer ces deux tâches ? L'approche la plus simple consiste à donner la priorité à l'algorithme de bas niveau (Fig. 10.18). De toute évidence, il est plus important d'éviter de heurter un piéton ou de contourner un nid-de-poule que d'emprunter le chemin le plus court pour se rendre à l'aéroport. Le robot est normalement dans l'état *drive*, mais si un obstacle est détecté, il passe à l'état *avoid obstacle*. Ce n'est que lorsque l'obstacle a été franchi qu'il revient à l'état planifier le chemin afin que le chemin puisse être recalculé.

La stratégie d'intégration des deux algorithmes dépend de l'environnement. La réparation d'une route peut prendre plusieurs semaines, il est donc logique d'ajouter l'obstacle à la carte. L'algorithme de planification de trajectoire tiendra compte de l'obstacle et la trajectoire résultante sera probablement meilleure qu'une trajectoire modifiée à la dernière minute par un algorithme d'évitement d'obstacles. À l'autre extrême, s'il y a beaucoup d'obstacles mobiles tels que des piétons traversant une rue, l'algorithme d'évitement d'obstacles pourrait simplement arrêter de se déplacer et attendre que les obstacles s'éloignent. Le plan initial peut alors être repris sans

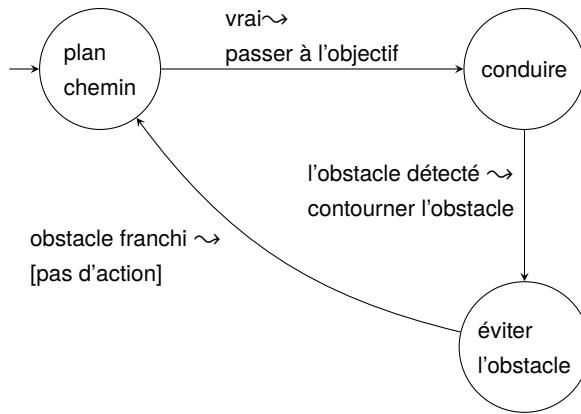


FIG. 10.18 – *Intégration de la planification de la trajectoire et de l'évitement des obstacles*

détour.

Activity 10.4: Combinaison de la planification de la trajectoire et de l'évitement des obstacles

- Modifiez votre implémentation de l'algorithme de suivi de ligne afin que le robot se comporte correctement même si un obstacle est placé sur la ligne. Essayez plusieurs des approches énumérées dans cette section.
- Modifiez votre implémentation de l'algorithme de suivi de ligne afin que le robot se comporte correctement même si d'autres robots se déplacent de manière aléatoire dans la zone de la ligne. Veillez à ce que les robots ne se heurtent pas les uns aux autres.

10.5 Résumé

La planification de trajectoire est un comportement de haut niveau d'un robot mobile : trouver le chemin le plus court d'un point de départ à un point d'arrivée dans l'environnement. La planification du chemin est basée sur une carte montrant les obstacles. L'algorithme de Dijkstra étend le chemin le plus court à toutes les cellules rencontrées jusqu'à présent. L'algorithme A* réduit le nombre de cellules visitées en utilisant une fonction heuristique qui indique la direction vers la cellule cible.

La planification des chemins est basée sur un graphe tel qu'une carte quadrillée, mais elle peut également être réalisée sur une carte continue en créant un graphe

d'obstacles à partir de la carte. Les algorithmes peuvent prendre en compte des coûts variables pour visiter chaque cellule.

L'évitement des obstacles à bas niveau doit être intégré dans la planification des chemins à haut niveau.

10.6 Lecture complémentaire

L'algorithme de Dijkstra est présenté dans tous les manuels sur les structures de données et les algorithmes, par exemple, [10, Sect. 24.3]. Les algorithmes de recherche tels que l'algorithme A* sont un sujet central de l'intelligence artificielle [39, Sect. 3.5].

Chapitre 11

Commande à logique floue

Les algorithmes de contrôle du chapitre 6 utilisaient des calculs mathématiques exacts pour déterminer les signaux utilisés pour contrôler le comportement d'un robot. Une autre approche consiste à utiliser des algorithmes de contrôle basés sur des *règles*. Un système de régulation de vitesse pourrait avoir des règles de la forme suivante :

- Si la *voiture de devant est loin* ou la *voiture de derrière est proche*, réglez la *vitesse sur rapide*.
- Si la *voiture de devant est proche*, réglez la *vitesse sur lente*.

La logique est "floue" parce que les règles sont exprimées en termes de *variables linguistiques* comme *vitesse* dont les valeurs n'ont pas de définitions mathématiques précises, mais seulement des spécifications linguistiques imprécises comme *rapide* et *lente*.

Un contrôleur à logique floue se compose de trois phases qui s'exécutent de manière séquentielle :

Fuzzify Les valeurs des capteurs sont converties en valeurs des variables linguistiques, telles que *far*, *fermeture*, *proche*, appelées *prémisses*. Chaque prémissse spécifie une *certitude* qui est la probabilité de notre croyance que la variable est vraie.

Appliquer les règles Un ensemble de *règles* exprime l'algorithme de contrôle. Étant donné un ensemble de *prémisses*, un *conséquent* est déduit. Les conséquences sont également des variables linguistiques telles que *très rapide*, *rapide*, *cruise*, *slow*, *stop*.

Defuzzify Les conséquences sont combinées afin de produire une sortie *crisp*, qui est une valeur numérique qui contrôle certains aspects du robot tels que la puissance appliquée aux moteurs.

Les sections suivantes présentent les trois phases du contrôle flou pour la tâche d'un robot s'approchant d'un objet et s'arrêtant lorsqu'il est très proche de l'objet.

11.1 Fuzzify

Lors de l'approche d'un objet, la valeur lue par le capteur de proximité horizontal passe de 0 à 100. La valeur renvoyée par le capteur est rendue floue en la convertissant en valeur d'une variable linguistique. La figure 11.1 montre trois graphiques permettant de convertir les valeurs du capteur en certitudes des variables linguistiques *far*, *round*, *closing* et *near*. L'axe x est la valeur renvoyée par le capteur et l'axe y

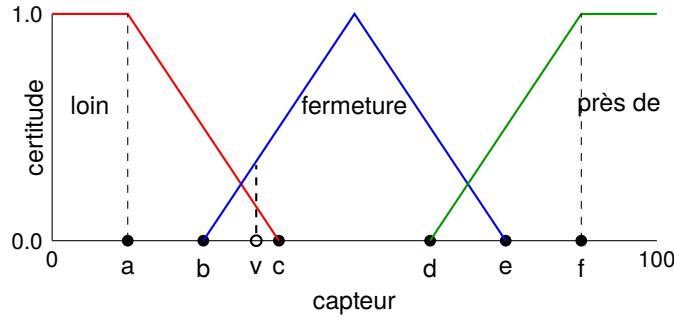


FIG. 11.1 – Fuzzifier la valeur du capteur de proximité horizontal

donne la prémisses pour chaque variable, la certitude que la variable linguistique est vraie.

Les points étiquetés sur l'axe x correspondent aux seuils : (a) far_low, (b) closing_low, (c) far_high, (d) near_low, (e) closing_high, (f) near_high. Si la valeur du capteur est inférieure à far_low, nous sommes totalement certains que l'objet est éloigné et la certitude est de 1. Si la valeur est comprise entre closing_low et far_high, nous sommes quelque peu certains que l'objet est éloigné, mais aussi quelque peu certains qu'il se rapproche. Le flou résulte du chevauchement des plages : lorsque la valeur se situe entre le point (b) et le point (c), nous ne pouvons pas dire avec une certitude absolue si l'objet est éloigné ou s'il se rapproche. Pour la valeur du capteur v d'environ 33, la certitude de far est d'environ .15 et la certitude de fermeture est d'environ .25.

11.2 Appliquer les règles

Les trois prémisses, les certitudes de far, fermeture et proche, sont utilisées pour calculer cinq conséquences en utilisant les règles suivantes :

1. Si farce alors très rapide
2. If far and closing then fast Si faronne et ferme alors rapide
3. Si fermeture alors croisière
4. If closing and near then slow (Si closing et near alors slow)
5. Si proche alors stop

Les certitudes des conséquences résultant des règles 1, 3, 5 sont les mêmes que les certitudes des prémisses correspondantes. Lorsqu'il y a deux prémisses, comme dans les règles 2 et 4, les certitudes des conséquences sont calculées à partir du minimum des certitudes des prémisses. Puisque *les deux* prémisses doivent s'appliquer, nous

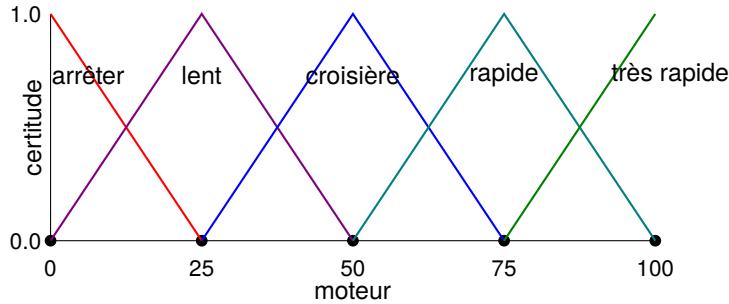


FIG. 11.2 – Défuzzifier pour obtenir le réglage croustillant du moteur

ne pouvons pas être *plus certains* du conséquent que nous le sommes de la plus petite des prémisses. Pour la valeur v dans la Fig. 11.1, la règle 2 s'applique et la certitude du conséquent est $\min(.15, .25) = .15$.

Une autre façon de combiner les prémisses est de prendre leur probabilité conjointe :

$$p(A \cap B) = P(A) \cdot P(B).$$

Pour la valeur v , la certitude du conséquent est de $0,15 \text{ fois } 0,25 = 0,0375$, soit beaucoup moins que la certitude obtenue à partir de la fonction minimale.

11.3 Defuzzifier

L'étape suivante consiste à combiner les conséquences en tenant compte de leurs certitudes. La figure 11.2 montre les puissances des moteurs de sortie pour chacun des cinq conséquents. Par exemple, si nous sommes totalement certains que la sortie est cruise, le graphique central de la figure montre que la puissance du moteur doit être fixée à 50, mais si nous sommes moins certains, la puissance du moteur doit être inférieure ou supérieure.

Supposons que la certitude du conséquent de cruise soit calculée à 0,4. Le triangle central de la Fig. 11.2 n'est alors plus pertinent car la certitude ne peut jamais être supérieure à 0,4, ce qui est représenté par un trapèze dans la Fig. 11.3.

Soit w et h la largeur et la hauteur d'un triangle. Alors l'aire d'un trapèze délimité par la droite de hauteur h' est donnée par la formule :¹

$$wh' \left(1 - \frac{h'}{2h} \right).$$

Il est possible que plus d'un conséquent ait des valeurs positives. La figure 11.3 montre les trapèzes pour le conséquent cruise avec une certitude de 0,4 et le consé-

1. La dérivation de la formule est donnée dans l'annexe B.6.

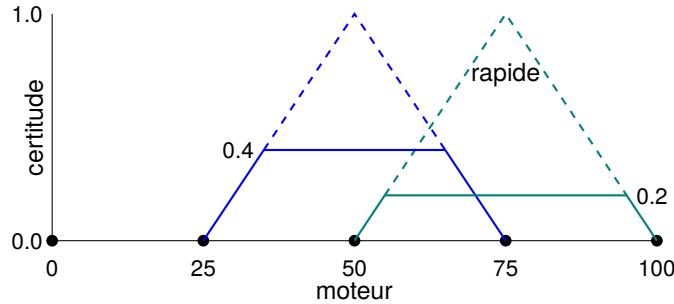


FIG. 11.3 – Domaines définis par les certitudes des conséquences

quent fast avec une certitude de 0.2. Pour $w = 50$, $h = 1$, $h'_c = 0.4$ (cruise), $h'_f = 0.2$ (fast), les aires des trapèzes a_c (cruise) et a_f (fast) sont :

$$a_c = 50 \times 0.4 \left(1 - \frac{0.4}{2}\right) = 16$$

$$a_f = 50 \times 0.2 \left(1 - \frac{0.2}{2}\right) = 9.$$

Pour obtenir une valeur précise, on calcule le *centre de gravité*. Il s'agit de la somme des surfaces des trapèzes pondérée par la valeur au centre de la base de chaque trapèze divisée par la somme des surfaces :

$$\frac{16 \times 50 + 9 \times 75}{16 + 9} = 59.$$

La valeur est plus proche de la valeur associée à cruise que de la valeur associée à fast. Cela n'est pas surprenant puisque la certitude de cruise est plus grande que la certitude de fast.

Activity 11.1: Logique floue

- Mettre en œuvre le contrôleur à logique floue pour un robot s'approchant d'un objet.
- Définir les seuils appropriés pour le capteur de proximité et les valeurs appropriées pour la défuzzification afin d'obtenir une vitesse de moteur précise.
- Comparez les résultats avec l'algorithme de contrôle proportionnel que vous avez implémenté dans Activity 6.3.

11.4 Résumé

Le contrôle par logique floue est une alternative aux algorithmes de contrôle mathématiques classiques décrits au Chap. 6. L'avantage de la commande par logique floue est qu'elle n'exige pas de spécifications mathématiques précises du comportement du robot, qui peuvent être difficiles à définir. Nous avons donné l'exemple des définitions floues de la vitesse ; d'autres exemples seraient la couleur (quand une nuance de rouge devient-elle orange ?) et la température (quand une pièce chaude devient-elle chaude ?). L'inconvénient est que le comportement de la commande à logique floue n'est pas aussi transparent que celui des algorithmes de commande classiques.

11.5 Lecture complémentaire

Les premiers travaux sur la logique floue ont été réalisés par Lotfi Zadeh [52]. Un manuel sur l'application de la logique floue au contrôle est [35]. La logique floue est également utilisée dans le traitement des images [17, Sect. 3.8].

Chapitre 12

Traitemen^tt des Images

Le capteur de distance de votre voiture autonome détecte un objet à 100 mètres devant votre voiture. Suivez-vous la voiture qui vous précède à une distance sûre ou un piéton a-t-il sauté sur la route ? Les algorithmes robotiques présentés jusqu'à présent reposent sur la mesure de propriétés physiques telles que la distance, les angles et la réflectance. Les tâches plus complexes exigent qu'un robot obtienne des informations détaillées sur son environnement, en particulier lorsque le robot est destiné à fonctionner de manière autonome dans un environnement inconnu.

Pour nous, la façon la plus évidente d'appréhender notre environnement est d'utiliser la vision. Nous tenons la vision pour acquise et ne réalisons pas à quel point notre système visuel - nos yeux et notre cerveau - est complexe. En fait, environ 30% du cerveau est utilisé pour la vision. Nous pouvons distinguer instantanément une voiture en mouvement d'un piéton qui traverse la route et réagir rapidement.

Depuis près de deux cents ans, il est possible d'enregistrer automatiquement des images à l'aide d'un appareil photo, mais l'interprétation des images reste une tâche qui incombe à l'homme. Avec l'avènement des ordinateurs, il est devenu possible de traiter et d'interpréter automatiquement les images. Les images numériques nous sont familières : les cartes météorologiques transmises par les satellites, les images médicales (radiographies, tomodensitogrammes, IRM, échographies) et les photos que nous prenons avec nos smartphones. Le traitement des images numériques est l'un des domaines les plus étudiés de l'informatique et de l'ingénierie, mais les systèmes de traitement d'images n'ont pas encore atteint les capacités du système visuel humain.

Dans ce chapitre, nous présentons un aperçu des algorithmes de traitement des images numériques et décrivons comment ils sont utilisés dans les systèmes robotiques. Les sections 12.1–12.2 donnent un aperçu des systèmes d'imagerie et du traitement numérique des images. Les sections 12.3–12.6 décrivent les algorithmes de traitement des images : amélioration par filtres numériques et manipulation de l'histogramme, segmentation (déttection des bords), et reconnaissance des caractéristiques (déttection des coins et des blobs, identification des caractéristiques multiples).

Pour des raisons de coût et de puissance de calcul, peu de robots éducatifs utilisent des caméras. Pour étudier les algorithmes de traitement d'images, vous pouvez donc les implémenter sur un ordinateur personnel en utilisant des images capturées avec un appareil photo numérique. Néanmoins, nous proposons quelques activités qui démontrent les algorithmes de traitement d'images sur un robot éducatif. Le robot se déplace sur une image unidimensionnelle et des échantillons sont lus par un capteur

au sol. Il en résulte un tableau unidimensionnel de pixels qui peut être traité à l'aide de versions simplifiées des algorithmes que nous présentons.

12.1 Obtention d'images

Dans cette section, nous donnons un aperçu des considérations relatives à la conception des systèmes d'imagerie.

Optique

Le système optique d'un appareil photo se compose d'un objectif qui concentre la lumière sur un capteur. Plus l'objectif est large, plus il peut capter de lumière, ce qui est important pour les systèmes qui doivent travailler dans des environnements sombres. Plus la longueur focale (qui est liée à la distance entre l'objectif et le capteur) est grande, plus le grossissement est important. C'est pourquoi les photographes professionnels portent des appareils photo lourds dotés de longs objectifs. Les fabricants de smartphones sont confrontés à un dilemme : nous voulons que nos téléphones soient fins et élégants, mais cela limite la longueur focale de l'appareil photo. Pour la plupart des applications robotiques, le grossissement ne vaut pas la taille et le poids nécessaires pour obtenir une longue distance focale.

Résolution

Il fut un temps où les images étaient capturées sur pellicule par une réaction chimique provoquée par la lumière frappant une feuille de plastique recouverte d'une émulsion de minuscules particules d'argent. En principe, chaque particule pouvait réagir indépendamment, de sorte que la résolution était extrêmement élevée. Dans les images numériques, la lumière est captée par des dispositifs semi-conducteurs tels que les dispositifs à couplage de charge (CCD). Un appareil photo numérique contient une puce avec un nombre fixe d'éléments dans un réseau rectangulaire. Chaque élément mesure l'intensité lumineuse de manière indépendante et ces mesures sont appelées *pixels*. Plus il y a de pixels capturés par une puce dans une zone donnée, plus la résolution est élevée. Actuellement, même les appareils photo bon marché des smartphones peuvent capturer des millions de pixels dans une seule image.

Le problème des images à haute résolution est la grande quantité de mémoire nécessaire pour les stocker. Prenons l'exemple d'un écran d'ordinateur à haute résolution avec 1920×1080 pixels et supposons que chaque pixel utilise 8 bits pour stocker l'intensité dans la plage 0–255. Une seule image nécessite environ 2 mégaoctets (Mo) de mémoire. Un ordinateur embarqué peut analyser une seule image de ce type, mais un robot mobile peut avoir besoin de stocker plusieurs images par seconde.

La puissance de calcul nécessaire à l'analyse des images est encore plus importante que la quantité de mémoire requise. Les algorithmes de traitement d'images exigent que l'ordinateur effectue un calcul sur chaque pixel. Ce n'est pas un problème pour un astronome qui analyse des images envoyées vers la terre par un télescope spatial, mais c'est un problème pour une voiture autonome qui doit prendre des décisions en une fraction de seconde.

Color

Notre système visuel est capable de distinguer une gamme de longueurs d'onde appelée *lumière visible*. Les différentes longueurs d'onde sont perçues comme des couleurs différentes. La lumière de grande longueur d'onde est appelée *rouge*, tandis que la lumière de petite longueur d'onde est appelée *violet*. L'œil humain peut distinguer des millions de couleurs différentes, même si nous n'en citons que quelques-unes : rouge, orange, jaune, vert, cyan, bleu, violet, etc. La couleur est l'un des principaux outils que nous utilisons pour identifier les objets.

Les capteurs sont capables de mesurer la lumière à des longueurs d'onde situées en dehors de la plage que nous appelons lumière visuelle : *infrarouge* pour les grandes longueurs d'onde et *ultraviolet* pour les courtes longueurs d'onde. Les images infrarouges sont importantes en robotique, car les objets chauds tels que les personnes et les voitures peuvent être détectés sous forme de lumière infrarouge brillante.

Le problème de la couleur est qu'elle triple les exigences en matière de stockage et de traitement des images. Toutes les couleurs peuvent être formées en prenant des quantités variables des trois *couleurs primaires* : rouge, vert et bleu (RVB). Par conséquent, une image couleur nécessite trois octets pour chaque pixel. Le stockage d'une seule image couleur d'une résolution de 1920×1080 nécessite plus de 6 Mo de mémoire et le traitement de l'image prend au moins trois fois plus de temps.

12.2 Une vue d'ensemble du traitement numérique des images

Le système optique d'un robot capture des images sous forme de réseaux rectangulaires de pixels, mais les tâches d'un robot sont exprimées en termes d'objets de l'environnement : entrer dans une pièce par une porte, prendre un article sur une étagère, s'arrêter si un piéton marche devant la voiture. Comment passer des pixels aux objets ?

La première étape est le *amélioration de l'image*. Les images contiennent du bruit qui provient de l'optique et de l'électronique. En outre, l'éclairage de l'environnement peut rendre une image trop sombre ou délavée ; l'image peut être tournée accidentellement ; l'image peut être floue. Tous ces problèmes sont indépendants du contenu. Il importe peu qu'une image floue représente un chat ou un enfant. Les

algorithmes d'amélioration d'image agissent généralement en modifiant les valeurs attribuées à chaque pixel, sans tenir compte de leur signification.¹

L'amélioration des images est difficile car il n'existe pas de définition formelle de ce que signifie l'amélioration d'une image. Une tache floue peut être de la saleté sur l'objectif d'un appareil photo ou une galaxie inconnue. La section 12.3 présente deux approches de l'amélioration d'image : le filtrage supprime le bruit en remplaçant un pixel par une moyenne de ses pixels voisins et la manipulation de l'histogramme modifie la luminosité et le contraste d'une image.

Les objets se distinguent par des lignes, des courbes et des zones. Une porte est constituée de trois bords droits d'un rectangle auquel il manque un petit côté. Un feu de circulation est constitué de trois disques lumineux superposés. Avant de pouvoir identifier une porte ou un feu de signalisation, les algorithmes de traitement d'image doivent déterminer quels pixels représentent des lignes, des bords, etc. Ce processus est appelé *segmentation* ou *extraction de caractéristiques* car les algorithmes doivent déterminer quels pixels font partie d'un segment d'une image.

La segmentation serait facile si les bords, les lignes et les courbes étaient uniformes, mais ce n'est pas le cas dans les images réelles. Un bord peut être incliné à un angle arbitraire et certains de ses pixels peuvent être masqués par des ombres ou même disparaître. Nous connaissons bien les *captchas* où les lettres sont intentionnellement déformées pour rendre la reconnaissance automatique très difficile alors que les humains peuvent facilement identifier des lettres déformées. Les algorithmes d'amélioration peuvent faciliter la segmentation, par exemple en remplissant les pixels manquants, mais ils peuvent également introduire des segments artificiels. La section 12.4 présente une technique de segmentation : un filtre qui détecte les bords d'une image.

La phase finale du traitement d'images consiste à reconnaître les objets. Dans la section 12.5, nous présentons deux algorithmes de détection des coins : en localisant l'intersection de deux bords et en comptant les voisins ayant des intensités similaires. La section 12.6 décrit comment reconnaître les *blobs*, qui sont des zones dont les pixels ont des intensités similaires mais qui ne sont pas délimitées par des caractéristiques régulières telles que des lignes et des courbes. Enfin, l'activité 12.6 démontre la reconnaissance d'un objet défini par plus d'une caractéristique, comme une porte définie par deux bords situés à une distance arbitraire l'un de l'autre.

12.3 Amélioration d'image

La figure 12.1 montre l'image d'un rectangle dont l'intensité est uniforme horizontalement et ombrée de sombre à clair de haut en bas. La représentation de l'image sous la forme d'un réseau rectangulaire de pixels de 6 fois 10 est illustrée dans la

1. Il existe une autre approche, appelée algorithmes de traitement *fréquence*, mais elle fait appel à des techniques mathématiques qui dépassent le cadre de cet ouvrage.

FIG. 12.1 – *Image sans bruit*FIG. 12.2 – *Image avec bruit*

figure 12.3, où chaque pixel est représenté par un niveau d'intensité lumineuse dans la plage 0–100. Regardez maintenant la Fig. 12.2 : l'image n'est plus *lisse* dans le sens où il y a trois points dont l'intensité n'est pas similaire aux intensités de ses voisins. Comparez le tableau de pixels de la Fig. 12.4 avec celui de la Fig. 12.3 : les intensités des pixels aux emplacements (2, 3), (3, 6), (4, 4) sont différentes. Il s'agit probablement d'un bruit et non d'une caractéristique réelle de l'objet photographié.

L'origine du bruit n'a pas vraiment d'importance : l'objet lui-même, la poussière sur l'objectif de l'appareil photo, la non-uniformité du capteur ou le bruit dans l'électronique. Il est impossible de se débarrasser entièrement du bruit, car nous ne pouvons jamais être sûrs qu'un pixel est un bruit ou une caractéristique réelle de l'objet, mais nous voulons améliorer l'image de manière à ce que le bruit ne soit plus perceptible.

12.3.1 Filtres spatiaux

Considérons la ligne 4 du tableau de pixels de la Fig. 12.4 :

50, 50, 50, 50, 90, 50, 50, 50, 50, 50 .

Fig. 12.5 est un graphique de l'intensité lumineuse f pour les pixels de cette rangée. Il est clair que l'un des pixels a une valeur improbable parce que sa valeur est très différente de celle de ses voisins. Un programme peut rendre chaque pixel plus proche de ses voisins en remplaçant l'intensité du pixel par la moyenne de son intensité et des intensités de ses voisins. Pour la plupart des pixels de la ligne, cela ne change pas leurs valeurs : $(50 + 50 + 50)/3 = 50$, mais le pixel de bruit et ses deux voisins reçoivent de nouvelles valeurs : $(50 + 90 + 50)/3 \approx 60$ (Fig. 12.6). Le calcul de

0	1	2	3	4	5	6	7	8	9
0	10	10	10	10	10	10	10	10	10
1	20	20	20	20	20	20	20	20	20
2	30	30	30	30	30	30	30	30	30
3	40	40	40	40	40	40	40	40	40
4	50	50	50	50	50	50	50	50	50
5	60	60	60	60	60	60	60	60	60

FIG. 12.3 – *Réseau de pixels sans bruit*

0	1	2	3	4	5	6	7	8	9
0	10	10	10	10	10	10	10	10	10
1	20	20	20	20	20	20	20	20	20
2	30	30	30	30	30	30	30	30	30
3	40	40	40	40	40	40	40	40	40
4	50	50	50	50	50	50	50	50	50
5	60	60	60	60	60	60	60	60	60

FIG. 12.4 – *Réseau de pixels avec bruit*

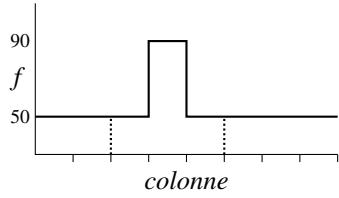


FIG. 12.5 – Plot d'intensité avant le calcul de la moyenne

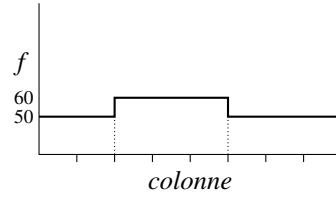


FIG. 12.6 – Plot d'intensité après le calcul de la moyenne

la moyenne a permis à deux pixels de recevoir des valeurs "erronées", mais dans l'ensemble, l'image sera visuellement améliorée car l'intensité du bruit sera réduite.

Prendre la moyenne d'une séquence de pixels est la version discrète de l'intégration d'une fonction d'intensité continue. L'intégration permet de lisser les variations locales de la fonction. Les lignes en pointillés des figures 12.5–12.6 indiquent une séquence de trois pixels et l'on peut voir que les zones qu'elles délimitent sont à peu près les mêmes.

L'opération de moyennage est réalisée en appliquant un *filtre spatial* à chaque pixel de l'image.² Pour le tableau bidimensionnel de pixels, le filtre est représenté par un tableau de 3×3 , où chaque élément du tableau spécifie le facteur par lequel le pixel et ses voisins sont multipliés. Chaque pixel a quatre ou huit voisins, selon que l'on inclut ou non les voisins diagonaux. Ici, nous incluons les pixels diagonaux dans les filtres.

Le *filtre de boîte* est :

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Les résultats des multiplications sont additionnés et la somme est divisée par 9 pour ramener le résultat à une valeur d'intensité.

L'application du filtre à chaque pixel (r, c) peut s'écrire explicitement comme suit :

$$\begin{aligned} g(r, c) = & (\\ & f(r-1, c-1) + f(r-1, c) + f(r-1, c+1) + \\ & f(r, c-1) + f(r, c) + f(r, c+1) + \\ & f(r+1, c-1) + f(r+1, c) + f(r+1, c+1) \\) / 9. \end{aligned}$$

Le résultat de l'application du filtre en boîte à l'image bruitée de la Fig. 12.4 est

2. Le terme mathématique pour appliquer une fonction g à chaque point d'une fonction f est appelé *convolution* (discrète). Pour les fonctions continues, l'intégration est utilisée à la place de l'addition de la moyenne.

0	1	2	3	4	5	6	7	8	9
0	10	10	10	10	10	10	10	10	10
1	20	20	18	18	18	20	20	20	20
2	30	30	28	28	28	26	26	30	30
3	40	40	38	43	43	41	36	36	40
4	50	50	50	54	54	51	46	46	50
5	60	60	60	60	60	60	60	60	60

0	1	2	3	4	5	6	7	8	9
0	10	10	10	10	10	10	10	10	10
1	20	20	19	19	19	20	20	20	20
2	30	30	29	25	29	28	28	30	30
3	40	40	39	41	41	40	25	38	40
4	50	50	50	52	70	50	48	48	50
5	60	60	60	60	60	60	60	60	60

FIG. 12.7 – Lissage avec le filtre en boîte FIG. 12.8 – Lissage avec un filtre pondéré

montré dans la Fig. 12.7.³ Les valeurs d'intensité ne sont plus uniformes, mais elles sont assez proches des valeurs d'origine, sauf là où il y avait des pixels de bruit. La deuxième ligne en partant du bas montre que la valeur de bruit de 90 n'apparaît plus ; au lieu de cela, toutes les valeurs de la ligne sont proches les unes des autres dans l'intervalle 46–54.

Le filtre en boîte donne la même importance au pixel et à tous ses voisins, mais un *filtre pondéré* utilise des facteurs différents pour les différents pixels. Le filtre suivant accorde beaucoup plus d'importance au pixel lui-même qu'à ses voisins :

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Il serait approprié d'utiliser ce filtre si nous pensons qu'un pixel a presque certainement sa valeur correcte, mais que nous voulons quand même que ses voisins influencent sa valeur. Après avoir appliqué ce filtre, le résultat doit être divisé par 16 pour que la somme soit convertie en valeur d'intensité. La figure 12.8 montre le résultat de l'utilisation du filtre pondéré. Si l'on regarde à nouveau la deuxième ligne en partant du bas, la valeur de 90 a seulement été réduite à 70 parce qu'un poids plus important est accordé au pixel par rapport à ses voisins.

Activity 12.1: Amélioration d'image : lissage

- Imprimez une feuille de papier avec un motif en niveaux de gris comme celui de la Fig. 12.9. Le motif comporte deux lignes noires que nous souhaitons détecter, mais aussi trois zones gris foncé (indiquées par les flèches) qui risquent d'être détectées à tort comme des lignes.
- Programmer le robot pour qu'il se déplace de gauche à droite sur le motif, en échantillonnant la sortie du capteur de sol. Examinez la sortie et fixez un seuil pour que le robot détecte à la fois les lignes noires et les

3. Le filtre n'est pas appliqué aux pixels dans les limites de l'image pour éviter de dépasser les limites du tableau. L'image peut également être complétée par des lignes et des colonnes supplémentaires.

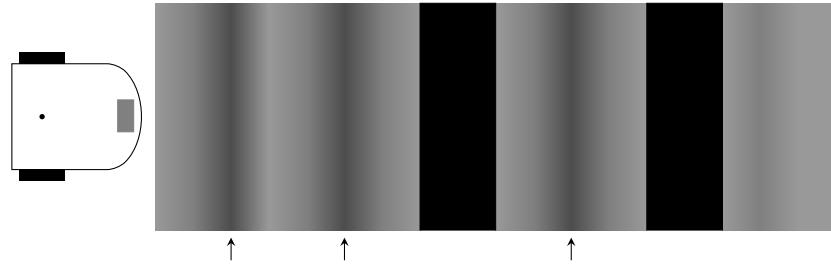


FIG. 12.9 – Amélioration unidimensionnelle de l'image

zones gris foncé. Modifiez le programme de manière à ce que, lors de son deuxième passage, il indique (par une lumière ou un son) lorsqu'il a détecté une ligne noire et une zone sombre.

- Modifier le programme pour qu'il remplace chaque échantillon par la moyenne de l'intensité de l'échantillon et de ses deux voisins. Le robot devrait maintenant détecter les deux lignes noires mais pas les zones grises.
- Expérimenter avec différents poids pour la moyenne.
- Expérmitez avec différents taux d'échantillonnage. Que se passe-t-il si vous échantillonnez le capteur de sol à intervalles très courts ?

12.3.2 Manipulation des histogrammes

La figure 12.10 montre les pixels d'une image binaire : une image où chaque pixel est soit noir, soit blanc.⁴ L'image montre un rectangle blanc 3×5 sur le fond noir. La figure 12.11 montre la même image à laquelle on a ajouté beaucoup de bruit aléatoire. En regardant l'image, il est possible d'identifier le rectangle, mais c'est très difficile à faire et le lissage de l'image n'y changera rien.

Construisons maintenant un *histogramme* des intensités (Fig. 12.12). Un histogramme est construit à partir de *bins*, où chaque bin stocke un nombre de pixels ayant une gamme d'intensités. L'histogramme de la figure contient dix cases correspondant à des intensités comprises entre 0 et 9, 10 et 19, 91 et 99. Si nous supposons que le rectangle blanc est petit par rapport à l'arrière-plan, il est facile de voir sur l'histogramme qu'il y a deux groupes de pixels, ceux qui sont relativement sombres et ceux qui sont relativement clairs. Un seuil de 50 ou 60 devrait permettre de distinguer le rectangle de l'arrière-plan, même en présence de bruit. En fait, un seuil

4. Les valeurs 10 pour le noir et 90 pour le blanc ont été utilisées à la place des plus habituelles 0 et 100 pour plus de clarté dans l'impression du tableau.

0	1	2	3	4	5	6	7	8	9
0	10	10	10	10	10	10	10	10	10
1	10	10	10	10	10	10	10	10	10
2	10	10	10	90	90	90	90	90	10
3	10	10	10	90	90	90	90	90	10
4	10	10	10	90	90	90	90	90	10
5	10	10	10	10	10	10	10	10	10

FIG. 12.10 – *Image binaire sans bruit*

0	1	2	3	4	5	6	7	8	9
0	19	17	37	19	26	11	46	27	37
1	11	24	17	30	14	43	29	22	34
2	31	37	38	63	72	86	65	64	27
3	33	38	49	73	63	66	59	76	40
4	47	13	44	90	86	56	63	65	18
5	10	34	29	14	35	31	26	42	15

FIG. 12.11 – *Image binaire avec bruit*

de 50 restaure l'image originale, tandis qu'un seuil de 60 restaure correctement 13 des 15 pixels du rectangle.

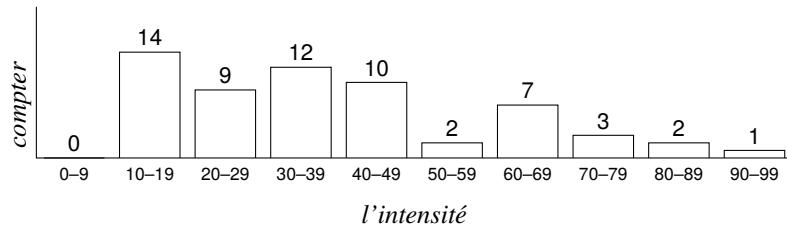
L'avantage de la manipulation de l'histogramme est qu'elle est très efficace, même sur des images de grande taille. Pour chaque pixel, divisez l'intensité par le nombre de cases et incrémentez le numéro de la case :

```
for each pixel p
    bin_number ← intensity(p) / number_of_bins
    bins[bin_number] ← bins[bin_number] + 1
```

Comparez cette opération à l'application d'un filtre spatial de 3×3 qui nécessite 9 de multiplications, 8 d'additions et une division à chaque pixel. En outre, peu de mémoire est nécessaire. Nous avons choisi 10 bins pour que la Fig. 12.12 puisse afficher l'histogramme entier, mais un histogramme complet en niveaux de gris de 8 bits ne nécessite que 256 bins.

Le choix d'un seuil en examinant un tracé de l'histogramme est facile, et si vous connaissez approximativement la fraction de l'arrière-plan couverte par les objets, la sélection du seuil peut se faire automatiquement.

Les algorithmes de manipulation de l'histogramme peuvent effectuer des améliorations plus complexes que le simple seuil binaire décrit ici. Il existe notamment des algorithmes permettant d'améliorer les images en modifiant la luminosité et le contraste d'une image.

FIG. 12.12 – *Histogramme de l'image bruitée*

0	1	2	3	4	5
0	30	30	30	30	30
1	30	30	30	30	30
2	30	30	30	30	30
3	50	50	50	50	50
4	50	50	50	50	50
5	50	50	50	50	50

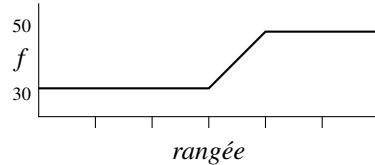


FIG. 12.13 – Image avec un bord

FIG. 12.14 – Intensité du bord

Activity 12.2: Amélioration d'image : manipulation de l'histogramme

- Modifier le programme dans Activity 12.1 pour qu'il calcule l'histogramme des échantillons.
- Comment l'histogramme change-t-il si le nombre d'échantillons est augmenté ?
- Examiner l'histogramme pour déterminer un seuil qui sera utilisé pour distinguer les lignes noires de l'arrière-plan.
- Calculer la somme des contenus des cases jusqu'à ce que la somme soit supérieure à une fraction (peut-être un tiers) des échantillons. Utiliser l'indice de la dernière case pour fixer le seuil.

12.4 Détection des contours

Les systèmes de traitement des images médicales nécessitent des algorithmes sophistiqués d'amélioration des images pour modifier la luminosité et le contraste, supprimer le bruit, etc. Cependant, une fois l'image améliorée, son interprétation est effectuée par des spécialistes qui savent quelles lignes et quelles ombres correspondent à quels organes du corps, et si ces organes sont normaux ou non. Un robot autonome n'a pas d'humain pour effectuer l'interprétation : il doit identifier des objets tels que des portes dans un bâtiment, des boîtes dans un entrepôt et des voitures sur la route. La première étape consiste à extraire des caractéristiques ou des segments tels que des lignes, des bords et des zones.

Considérons le tableau de pixels de 6×6 de la Fig. 12.13. Le niveau d'intensité de chaque ligne est uniforme, mais il existe une discontinuité nette entre les niveaux d'intensité des lignes 2 et 3. Il s'agit clairement d'un bord entre la zone sombre en haut et la zone claire en bas. Le calcul de la moyenne rendra le changement d'intensité plus lisse et nous perdrons le changement brutal au niveau du bord.

Étant donné que le calcul de la moyenne est un opérateur d'intégration qui *supprime* les changements brusques dans les intensités, il n'est pas surprenant que

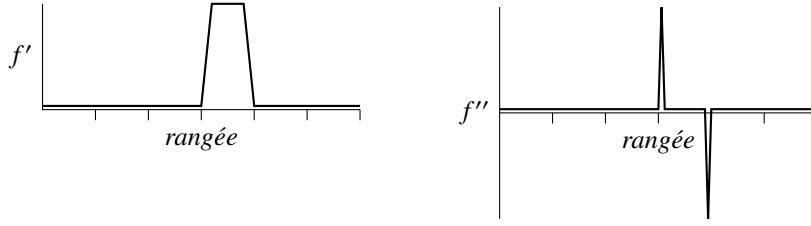


FIG. 12.15 – Dérivée première de l'intensité du bord

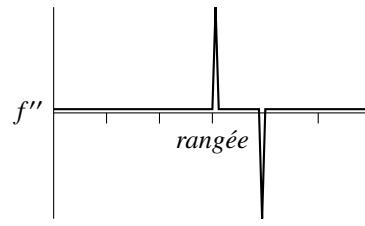


FIG. 12.16 – Dérivée seconde de l'intensité du bord

l'opérateur différentiel puisse être utilisé pour *déetecter* les changements brusques qui représentent les bords. La figure 12.14 représente l'intensité en fonction du numéro de ligne le long d'une seule colonne de la figure 12.13, bien que les intensités soient représentées sous forme de lignes et non de points discrets. L'intensité ne change pas pour les trois premiers pixels, puis elle augmente rapidement et se maintient au niveau supérieur. La dérivée première \$f'\$ d'une fonction \$f\$ est nulle lorsque \$f\$ est constante, positive lorsque \$f\$ augmente et négative lorsque \$f\$ diminue. C'est ce que montre la figure 12.15. Un bord peut être détecté en recherchant une augmentation ou une diminution rapide de la dérivée première de l'intensité de l'image.

Dans la pratique, il est préférable d'utiliser la dérivée seconde. La figure 12.16 montre un tracé de \$f''\$, la dérivée de \$f'\$ dans la figure 12.15. Le pic positif suivi du pic négatif indique une transition de l'obscurité vers la lumière ; si la transition se faisait de la lumière vers l'obscurité, le pic négatif précéderait le pic positif.

Il existe de nombreux opérateurs de dérivation numérique. L'un d'entre eux, simple mais efficace, est le *filtre de Sobel*. Il existe deux filtres, l'un pour détecter les bords horizontaux (à gauche) et l'autre pour détecter les bords verticaux (à droite) :

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}.$$

Une caractéristique d'un filtre dérivé est que la somme de ses éléments doit être égale à zéro. En effet, si l'opérateur est appliqué à un pixel dont l'intensité est la même que celle de tous ses voisins, le résultat doit être nul. Regardez à nouveau les figures 12.14 et 12.15 où la dérivée est nulle lorsque l'intensité est constante.

Lorsque les filtres de Sobel sont appliqués à la matrice de pixels de la figure 12.13, le résultat détecte clairement la présence d'un bord horizontal (figure 12.17) mais pas de bord vertical (figure 12.18).

Les filtres de Sobel sont très puissants car ils peuvent non seulement détecter une arête mais aussi calculer l'angle de l'arête dans l'image. La figure 12.19 montre une image avec un bord allant en diagonale de l'angle supérieur gauche à l'angle inférieur droit. Les résultats de l'application des deux filtres de Sobel sont présentés

0	1	2	3	4	5
0	0	0	0	0	0
1	0	0	0	0	0
2	0	80	80	80	0
3	0	80	80	80	0
4	0	0	0	0	0
5	0	0	0	0	0

FIG. 12.17 – Arête horizontale de Sobel

0	1	2	3	4	5
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

FIG. 12.18 – Arête verticale de Sobel

dans les figures 12.20–12.21. À partir des amplitudes et des signes des éléments de ces tableaux, l’angle du bord peut être calculé comme décrit dans [43, Sect. 4.3.1].

Activity 12.3: Détection d’un bord

- Imprimer un motif avec une arête vive (Fig. 12.22).
- Adapter le programme de Activity 12.1 pour que le robot échantillonne et stocke le capteur de sol lorsque le robot se déplace sur le motif de gauche à droite. Appliquer un filtre dérivé aux échantillons.

0	1	2	3	4	5
0	30	30	30	30	30
1	50	30	30	30	30
2	50	50	30	30	30
3	50	50	50	30	30
4	50	50	50	50	30
5	50	50	50	50	50

FIG. 12.19 – Bord diagonal

0	1	2	3	4	5
0	0	0	0	0	0
1	0	60	20	0	0
2	0	60	60	20	0
3	0	20	60	60	20
4	0	0	20	60	60
5	0	0	0	0	0

FIG. 12.20 – Filtre horizontal de Sobel sur une arête diagonale

0	1	2	3	4	5
0	0	0	0	0	0
1	0	-60	-20	0	0
2	0	-60	-60	-20	0
3	0	-20	-60	-60	-20
4	0	0	-20	-60	-60
5	0	0	0	0	0

FIG. 12.21 – Filtre vertical de Sobel sur une arête diagonale

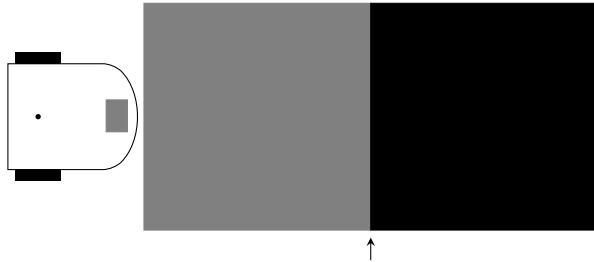


FIG. 12.22 – Activité de détection des contours

- Lors d'un second passage sur le motif, le robot indique lorsque la valeur de la dérivée n'est pas proche de zéro.
- Que se passe-t-il si le robot se déplace sur le motif de droite à gauche ?
- Lors de l'application du filtre, les résultats doivent être stockés dans un tableau séparé, et non dans le tableau utilisé pour stocker les pixels. Pourquoi ?

12.5 Détection des coins

Le rectangle noir sur fond gris de la Fig. 12.23 est plus qu'un simple ensemble d'arêtes. Les arêtes verticales forment deux coins avec l'arête horizontale. Nous décrivons ici deux algorithmes permettant d'identifier les coins dans une image. Pour simplifier, nous supposons que les coins sont alignés sur l'image rectangulaire.

Nous savons comment détecter les bords d'une image. Un coin est défini par l'intersection d'un bord vertical et d'un bord horizontal. Figure 12.24 est le tableau de 6×10 pixels pour l'image de la Fig. 12.23. Si nous appliquons les détecteurs d'arêtes de Sobel à cette matrice de pixels, nous obtenons deux arêtes verticales (Fig. 12.25) et une arête horizontale (Fig. 12.26).



FIG. 12.23 – Image d'un coin

	0	1	2	3	4	5	6	7	8	9
0	30	30	30	30	30	30	30	30	30	30
1	30	30	30	30	30	30	30	30	30	30
2	30	30	30	50	50	50	50	30	30	30
3	30	30	30	50	50	50	50	30	30	30
4	30	30	30	50	50	50	50	30	30	30
5	30	30	30	50	50	50	50	30	30	30

FIG. 12.24 – Réseau de pixels d'un coin

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0
1	0	0	20	20	0	0	-20	-20	0
2	0	0	60	60	0	0	-60	-60	0
3	0	0	80	80	0	0	-80	-80	0
4	0	0	80	80	0	0	-80	-80	0
5	0	0	0	0	0	0	0	0	0

FIG. 12.25 – Vertical edges

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0
1	0	0	20	60	80	80	80	60	20
2	0	0	20	60	80	80	80	60	20
3	0	0	0	0	0	0	0	0	0

FIG. 12.26 – Bord horizontal

L'intersection est définie pour les pixels pour lesquels la somme des valeurs absolues dans les deux tableaux d'arêtes de Sobel est supérieure à un seuil. Avec un seuil de 30, les arêtes se croisent dans les pixels (2,3) et (2,7) qui sont les coins.

Une zone uniforme, un bord et un coin peuvent être distingués en analysant les voisins d'un pixel. Dans une zone uniforme, tous les voisins du pixel ont approximativement la même intensité. Dans un bord, les intensités des voisins du pixel sont très différentes dans une direction et similaires dans l'autre. Dans un coin, les intensités des voisins du pixel présentent peu de similitudes. Pour détecter un coin, comptez le nombre de voisins similaires pour chaque pixel, trouvez la valeur minimale et identifiez comme coins les pixels présentant cette valeur minimale. La figure 12.27 montre les décomptes pour les pixels de la figure 12.24. Comme prévu, les pixels d'angle (2,3) et (2,7) ont le nombre minimum de voisins similaires.

Activity 12.4: Détecter un coin

- Mettre en œuvre la détection d'un coin par intersection d'arêtes à l'aide d'un robot équipé de deux capteurs de proximité au sol. Le robot se déplace du bas de l'image de la Fig. 12.23 vers le haut. S'il est placé au-dessus du rectangle noir, il ne détecte pas de coin, alors que s'il est placé de manière à ce qu'un capteur soit au-dessus du rectangle noir et

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0
1	0	8	7	6	5	5	6	7	0
2	0	8	6	3	5	5	3	6	0
3	0	8	5	5	8	8	5	5	0
4	0	8	5	5	8	8	5	5	0
5	0	0	0	0	0	0	0	0	0

FIG. 12.27 – Voisins similaires

0	1	2	3	4	5	6	7	8	9
0	30	30	30	30	30	30	30	30	30
1	30	30	30	30	80	80	30	30	30
2	30	30	30	80	80	80	30	30	30
3	30	30	30	80	80	80	30	30	30
4	80	30	30	80	80	30	30	30	80
5	30	30	30	30	30	30	30	30	30

FIG. 12.28 – *Blob*

0	1	2	3	4	5	6	7	8	9
0	46	42	40	50	46	44	40	33	30
1	32	46	46	46	67	73	39	47	39
2	33	40	40	73	68	63	73	44	42
3	35	41	50	67	60	71	60	37	30
4	68	46	32	44	61	77	48	42	45
5	39	37	38	34	33	40	35	37	34

FIG. 12.29 – *Blob avec bruit*

l'autre au-dessus du fond gris, il détecte le coin.

- Mettre en œuvre la détection des coins par des voisins similaires. Vérifier à plusieurs reprises les échantillons actuels des capteurs gauche et droit et les échantillons précédents des capteurs gauche et droit. Si un seul des quatre échantillons est noir, un coin est détecté.

12.6 Reconnaissance des blobs

La figure 12.28 montre un *blob* : une zone à peu près circulaire de 12 pixels de forte intensité sur un fond de faible intensité. Le blob n'a pas de limite bien définie comme un rectangle. L'avant-dernière ligne montre également deux artefacts de forte intensité qui ne font pas partie du blob, bien qu'ils puissent représenter des blobs distincts. 12.29 montre les pixels après l'ajout d'un bruit aléatoire. La tâche consiste à identifier le blob en présence de bruit, sans dépendre d'un seuil d'intensité prédéfini et en ignorant les artefacts. L'indépendance de l'identification par rapport à l'intensité globale est importante pour que le robot puisse accomplir sa tâche quelles que soient les conditions d'éclairage de l'environnement.

Pour ignorer le bruit sans prédéfinir de seuil, nous utilisons un seuil défini en fonction de l'intensité moyenne de l'image. Pour séparer les blobs les uns des autres, nous trouvons d'abord un pixel dont l'intensité est supérieure au seuil, puis nous agrandissons le blob en ajoutant les pixels voisins dont l'intensité est supérieure au seuil. Pour l'image bruitée de la Fig. 12.29, l'intensité moyenne est de 54. Étant donné que le blob occupe vraisemblablement une partie relativement petite de l'arrière-plan, il peut être judicieux de choisir un seuil légèrement supérieur à la moyenne, par exemple 60.

La figure 12.30 montre l'image après avoir attribué 0 à tous les pixels en dessous du seuil. Le blob a été détecté, mais les deux artefacts aussi. Algorithme 12.1 est un algorithme permettant d'isoler un seul blob. Tout d'abord, recherchez un pixel qui n'est pas nul; en commençant en haut à gauche, ce sera le pixel $p_1 = (1, 4)$

```

 0  1  2   3   4   5   6   7   8   9
0  0  0  0   0   0   0  0  0   0
1  0  0  0   0  67  73   0  0  0   0
2  0  0  0  73  68  63  73   0  0   0
3  0  0  0  67  60  71  60   0  0   0
4 68  0  0   0  61  77   0  0  0  62
5  0  0  0   0   0   0  0  0   0

```



FIG. 12.30 – Blob après le seuil

FIG. 12.31 – Blob à détecter

avec une intensité de 67. Développez maintenant le blob en ajoutant tous les voisins de p_1 dont les intensités sont non nulles ; il s'agit de $p_2 = (1, 5)$, $p_3 = (2, 3)$, $p_4 = (2, 4)$, $p_5 = (2, 5)$. Continuez à ajouter des voisins non nuls de chaque p_i au blob jusqu'à ce qu'il n'y ait plus de pixels ajoutés. Le résultat sera le blob de 12 pixels sans les artefacts à (4, 0), (4, 9).

L'algorithme fonctionne parce que le premier pixel non nul trouvé appartenait au blob. S'il y avait eu un pixel isolé non nul à (1, 1), cet artefact aurait été détecté comme un blob. S'il existe une estimation de la taille minimale d'un blob, l'algorithme devrait être suivi d'une vérification que le blob a au moins cette taille.

Vérifiez que l'algorithme 12.1 n'est pas sensible au niveau d'intensité en soustrayant la valeur constante 20 de tous les éléments de l'image bruitée (Fig. 12.29) et en réexécutant l'algorithme. Celui-ci devrait toujours identifier les mêmes pixels

Algorithm 12.1: Détection d'un blob
integer threshold pixel p set not-explored \leftarrow empty-set set blob \leftarrow empty-set
1: set the threshold to the average intensity 2: set pixels below threshold to zero 3: find a non-zero pixel and add to not-explored 4: while not-explored not empty 5: p \leftarrow some element of not-explored 6: add p to blob 7: remove p from not-explored 8: add non-zero neighbors of p to not-explored



FIG. 12.32 – Reconnaître la porte



FIG. 12.33 – Il ne s'agit pas d'une porte

comme appartenant au blob.

Activity 12.5: Détection d'un blob

- Ecrire un programme qui permet au robot d'échantillonner le capteur de sol lorsqu'il se déplace de gauche à droite sur le motif de la Fig. 12.31.
- Calculer l'intensité moyenne et fixer le seuil à la moyenne.
- Lors d'un deuxième passage sur le motif, après avoir détecté le premier échantillon du rectangle noir qui est en dessous du seuil, le robot fournit une indication (par la lumière ou le son) tant qu'il se déplace sur le rectangle.
- Le robot doit considérer le deuxième rectangle noir comme un artefact et l'ignorer.

Activity 12.6: Reconnaître une porte

- Dans la figure 12.32, le rectangle gris représente une porte ouverte dans un mur sombre représenté par les rectangles noirs. La figure 12.33 représente un mur sombre entre deux portes ouvertes grises. Si vous exécutez le programme à partir de l'activité 12.3, vous verrez que deux arêtes sont détectées pour les deux modèles. Modifiez le programme pour que le robot puisse faire la distinction entre les deux motifs.

12.7 Résumé

Chez les êtres humains et la plupart des animaux, la vision est le capteur le plus important et une grande partie du cerveau est consacrée à l'interprétation des signaux visuels. Les robots peuvent utiliser la vision pour effectuer des tâches avancées dans un environnement en constante évolution. La technologie des caméras numériques est très avancée et les caméras peuvent transférer des matrices de pixels

à haute résolution à l'ordinateur du robot. Les algorithmes de traitement des images numériques améliorent et interprètent ces images.

Les algorithmes d'amélioration suppriment le bruit, améliorent le contraste et effectuent d'autres opérations qui ne dépendent pas des objets apparaissant dans l'image. Ils utilisent des filtres spatiaux qui modifient l'intensité de chaque pixel en fonction de l'intensité de ses voisins. La modification de l'histogramme utilise la distribution globale des intensités dans une image pour modifier les pixels individuels.

Après l'amélioration de l'image, les algorithmes identifient les objets dans l'image. Ils commencent par détecter des propriétés géométriques simples telles que les bords et les coins, puis identifient les objets qui apparaissent dans l'image.

12.8 Lecture complémentaire

Gonzalez et Woods [17] est un manuel complet sur le traitement des images numériques qui comprend les bases mathématiques du sujet. Russ [38] est un ouvrage de référence sur le traitement des images. Szeliski [45] est un livre sur la vision par ordinateur qui va au-delà du traitement d'images et se concentre sur la construction de modèles 3D d'images. Pour les applications du traitement d'images en robotique, voir [43, Chapter 4].

Chapitre 13

Réseaux neuronaux

Le chapitre 3 décrit les comportements réactifs inspirés par les travaux de Valentino Braitenberg. Le contrôle des véhicules simples de Braitenberg est très similaire au contrôle d'un organisme vivant par son *réseau neuronal* biologique. Ce terme désigne le système nerveux d'un organisme vivant, y compris son cerveau et les nerfs qui transmettent les signaux à travers le corps. Les modèles informatiques de réseaux neuronaux sont un sujet de recherche actif en intelligence artificielle. *Les réseaux neuronaux artificiels (RNA)* permettent de mettre en œuvre un comportement complexe à l'aide d'un grand nombre de composants abstraits relativement simples qui sont modélisés sur les neurones, les composants des réseaux neuronaux biologiques. Ce chapitre présente l'utilisation des réseaux neuronaux artificiels pour contrôler le comportement des robots.

Après un bref aperçu du système nerveux biologique dans la section 13.1, la section 13.2 définit le modèle ANN et la section 13.3 montre comment il peut être utilisé pour mettre en œuvre le comportement d'un véhicule de Braitenberg. La section 13.4 présente différentes topologies de réseau. La caractéristique la plus importante des ANN est leur capacité d'apprentissage, qui leur permet d'adapter leur comportement. La section 13.5 présente une vue d'ensemble de l'apprentissage dans les ANN à l'aide de la règle de Hebbian.

13.1 Le système neuronal biologique

Le système nerveux des organismes vivants est constitué de cellules appelées *neurones* qui traitent et transmettent des informations au sein de l'organisme. Chaque neurone effectue une opération simple, mais la combinaison de ces opérations conduit à un comportement complexe. La plupart des neurones sont concentrés dans le cerveau, mais d'autres forment les nerfs qui transmettent les signaux vers et depuis le cerveau. Chez les *vertébrés* comme nous, de nombreux neurones sont concentrés dans la moelle épinière, qui transmet efficacement les signaux dans tout le corps. Le nombre de neurones dans un être vivant est immense : le cerveau humain compte environ 100 milliards de neurones, tandis que le cerveau d'une souris en compte 71 millions [21].

La figure 13.1 montre la structure d'un neurone. Il se compose d'un corps principal avec un *noyer* et d'une longue fibre appelée *axon* qui permet à un neurone de se connecter à un autre. Le corps d'un neurone a des projections appelées *dendrites*. Les axones d'autres neurones se connectent aux dendrites par l'intermédiaire de

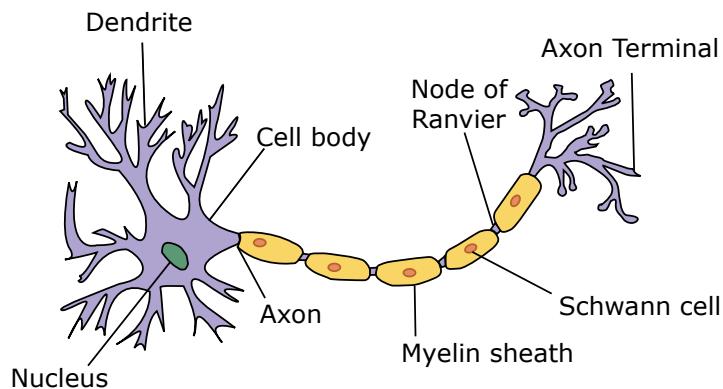


FIG. 13.1 – Structure d'un neurone. <https://commons.wikimedia.org/wiki/File:Neuron.svg> by Dhp1080 [CC BY-SA 3.0 (<http://creativecommons.org/licenses/by-sa/3.0>) or GFDL (https://en.wikipedia.org/wiki/en:GNU_Free_Documentation_License)], via Wikimedia Commons.

synapses. Les neurones fonctionnent grâce à des processus biochimiques qui sont bien compris, mais nous pouvons résumer ces processus en *impulsions* qui se déplacent d'un neurone à l'autre. Les impulsions d'entrée sont reçues par les synapses dans les dendrites et, de là, dans le corps du neurone, qui traite les impulsions et transmet à son tour une impulsion de sortie par l'axone. Le traitement dans le corps d'un neurone peut être résumé comme une fonction allant des impulsions d'entrée à une impulsion de sortie, et les synapses régulent la transmission des signaux. Les synapses sont adaptatives et constituent l'élément principal qui rend possible la mémoire et l'apprentissage.

13.2 Le modèle de réseau neuronal artificiel

Un neurone artificiel est un modèle mathématique d'un neurone biologique (Figs. 13.2–13.3 ; voir le Tableau 13.1 pour une liste des symboles apparaissant dans les diagrammes ANN). Le corps du neurone est un noeud qui remplit deux fonctions : il calcule la somme des signaux d'entrée pondérés et applique une fonction de sortie à la somme. Les signaux d'entrée sont multipliés par des poids avant que les fonctions de somme et de sortie ne soient appliquées ; cela modélise la synapse. La fonction de sortie est généralement non linéaire : (1) convertir la sortie du neurone en un ensemble de valeurs discrètes (allumer ou éteindre une lumière) ; (2) limiter la plage des valeurs de sortie (la puissance du moteur peut être comprise entre 100 et 100) ; (3) normaliser la plage des valeurs de sortie (le volume d'un son est compris entre 0 (muet) et 1 (maximum)).

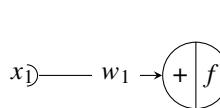


FIG. 13.2 – ANN : un neurone avec une entrée

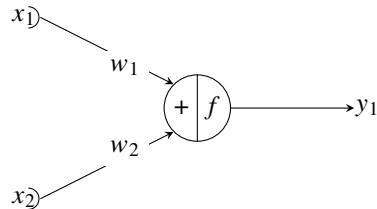


FIG. 13.3 – ANN : un neurone avec deux entrées

Les neurones artificiels sont des modèles analogiques, c'est-à-dire que les entrées, les sorties, les poids et les fonctions peuvent être des nombres à virgule flottante. Nous commençons par une activité irréaliste qui montre comment les neurones artificiels fonctionnent dans le contexte familier des portes logiques numériques.

La figure 13.4 montre un neurone artificiel avec deux entrées, x_1 et 1, et une sortie y . La signification de l'entrée 1 est qu'elle n'est pas connectée à un capteur externe, mais qu'elle renvoie une valeur constante de 1. La valeur d'entrée de x_1 est supposée être 0 ou 1. La fonction f est :

$$\begin{aligned} f(x) &= 0 && \text{si } x < 0 \\ f(x) &= 1 && \text{si } x \geq 0. \end{aligned}$$

Show that with the given weights the neuron implements the logic gate for not.

Activity 13.1: Artificial neurons for logic gates

- The artificial neuron in Fig. 13.5 has an additional input x_2 . Assign weights w_0, w_1, w_2 so that y is 1 only if the values of x_1 or x_2 (or both) are 1. This implements the logic gate for or.
- Assign weights w_0, w_1, w_2 so that y is 1 only if the values of x_1 and x_2 are both 1. This implements the logic gate for and.

TAB. 13.1 – Symboles utilisés dans les diagrammes ANN

Symbol	signification
f	Fonction de sortie des neurones
+	Somme des entrées
x_i	Entrées
y_i	Sorties
w_i	Pondération des entrées
1	Entrée constante de la valeur 1

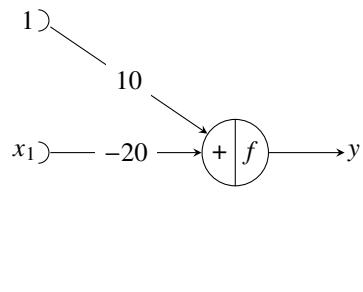


FIG. 13.4 – Neurone artificiel pour la porte *not*

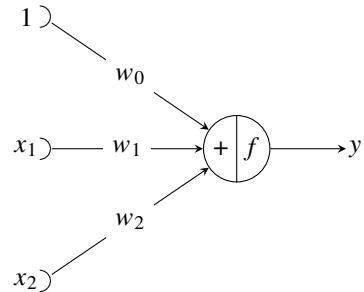


FIG. 13.5 – Neurone artificiel pour les portes *and* et *or*

- Implement the artificial neurons for logic gates on your robot. Use two sensors, one for x_1 and one for x_2 . Use the output y (mapped by f , if necessary) so that an output of 0 gives one behavior and an output of 1 another behavior, such as turning a light on or off, or starting and stopping the robot.

L’activité suivante explore le traitement analogique dans un neurone artificiel.

Activity 13.2: Neurones artificiels analogiques

- Implémenter le neurone artificiel montré dans la Fig. 13.2 de façon à ce qu’il démontre le comportement suivant. L’entrée du neurone sera la lecture d’un capteur de proximité à l’avant du robot. La sortie sera l’un des éléments suivants, ou les deux : (1) l’intensité d’une lumière sur le robot ou le volume du son émis par un haut-parleur sur le robot; (2) la puissance motrice appliquée aux moteurs gauche et droit afin que le robot recule devant un objet détecté par le capteur.
- La valeur de sortie sera proportionnelle à la valeur d’entrée : plus l’objet est proche, plus l’intensité (ou le volume) est élevée; plus l’objet est proche, plus le robot s’éloigne rapidement de l’objet.
- Modifier l’implémentation pour qu’il y ait deux entrées provenant de deux capteurs de proximité (Fig. 13.3). Donner des valeurs différentes aux deux poids w_1 , w_2 et montrer que le capteur connecté à l’entrée avec le poids le plus grand a plus d’effet sur la sortie.

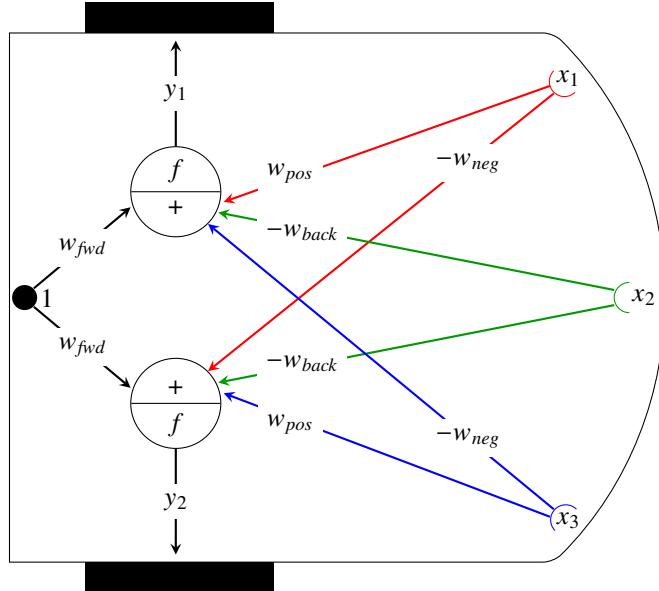


FIG. 13.6 – Réseau neuronal pour l'évitement d'obstacles

13.3 Mise en œuvre d'un véhicule de Braintenberg avec un ANN

La figure 13.6 montre un robot inspiré d'un véhicule de Braintenberg dont le comportement est implémenté à l'aide d'un simple réseau neuronal. Nous décrivons le réseau neuronal en détail et proposons ensuite plusieurs activités qui vous demandent de concevoir et d'implémenter l'algorithme.

Spécification (évitement d'obstacles) :

Le robot dispose de trois capteurs orientés vers l'avant et d'un réseau neuronal. Le robot est équipé de trois capteurs orientés vers l'avant.

- Le robot avance à moins qu'il ne détecte un obstacle.
- Si l'obstacle est détecté par le capteur central, le robot recule lentement.
- Si l'obstacle est détecté par le capteur gauche, le robot tourne à droite.
- Si l'obstacle est détecté par le capteur de droite, le robot tourne à gauche.

La figure 13.6 montre les deux neurones dont les sorties contrôlent la puissance envoyée aux moteurs des roues du robot. Le tableau 13.2 liste les symboles utilisés dans la figure.

Chaque neurone a quatre entrées. La fonction f doit être non linéaire afin de

TAB. 13.2 – Symboles de la figure 13.6 en plus de ceux du tableau 13.1

Symbol	signification
w_{fwd}	Poids pour le mouvement vers l'avant
w_{back}	Poids pour le mouvement vers l'arrière
w_{pos}	Poids pour la rotation positive des roues
w_{neg}	Poids pour la rotation négative des roues

limiter les vitesses maximales d'avance et de recul. Le gros point à l'arrière du robot représente une entrée constante de 1 qui est pondérée par w_{fwd} . Cela garantit qu'en l'absence de signaux provenant des capteurs, le robot se déplacera vers l'avant. Lors de l'implémentation de l'ANN, vous devez trouver un poids pour que les puissances de sortie du moteur soient raisonnables en l'absence de signaux provenant des capteurs. Le poids doit également garantir que l'entrée constante est similaire aux entrées provenant des capteurs.

Les valeurs x_1, x_2, x_3 proviennent des capteurs qui renvoient une valeur nulle lorsqu'il n'y a pas d'objet et une valeur positive croissante à l'approche d'un objet. Le capteur central est connecté aux deux neurones avec un poids négatif $-w_{back}$ de sorte que si un obstacle est détecté, le robot se déplacera vers l'arrière. Ce poids doit être fixé à une valeur telle que le robot recule lentement.

Les capteurs gauche et droit sont connectés aux neurones avec un poids positif pour le neurone contrôlant la roue proche et un poids négatif pour le neurone contrôlant la roue éloignée. Cela permet de s'assurer que le robot se détourne de l'obstacle.

L'activité suivante vous demande de réfléchir aux valeurs relatives des poids.

Activity 13.3: ANN pour l'évitement d'obstacles : conception

- Quelle relation doit exister entre w_{fwd} et w_{back} ?
- Quelle relation doit exister entre w_{fwd} et w_{pos} et entre w_{fwd} et w_{neg} ?
- Quelle relation doit exister entre w_{back} et w_{pos} et entre w_{back} et w_{neg} ?
- Quelle relation doit exister entre w_{pos} et w_{neg} ?
- Que se passe-t-il si l'obstacle est détecté à la fois par les capteurs de gauche et du centre ?

Dans les activités suivantes, vous devrez expérimenter avec les poids et les fonctions pour obtenir le comportement souhaité. Votre programme doit utiliser une structure de données telle qu'un tableau afin qu'il soit facile de modifier les valeurs des poids.

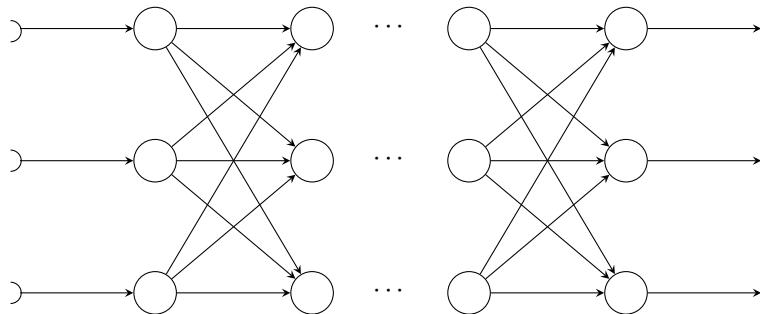


FIG. 13.7 – Réseau neuronal pour l'apprentissage profond

Activity 13.4: ANN pour l'évitement d'obstacles : mise en œuvre

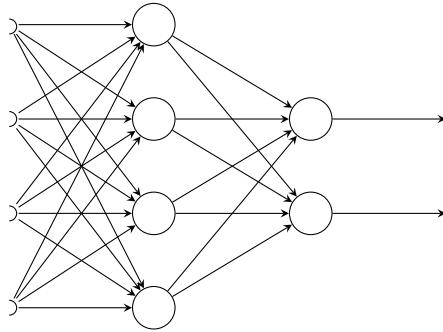
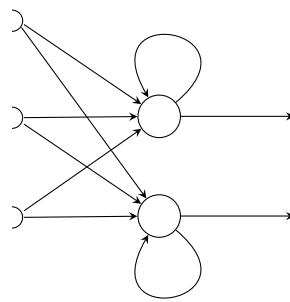
- Écrire un programme pour l'évitement d'obstacles en utilisant l'ANN de la Fig. 13.6.

Activity 13.5: ANN pour l'attraction des obstacles

- Écrire un programme pour mettre en œuvre l'attraction d'obstacles à l'aide d'un ANN :
- Le robot avance.
- Si le capteur central détecte que le robot est *très proche* de l'obstacle, il s'arrête.
- Si un obstacle est détecté par le capteur gauche, le robot tourne à gauche.
- Si un obstacle est détecté par le capteur de droite, le robot tourne à droite.

13.4 Réseaux neuronaux artificiels : topologies

L'exemple de la section précédente est basé sur un réseau neuronal artificiel composé d'une seule couche de deux neurones, chacun ayant plusieurs entrées et une seule sortie. Il s'agit d'une topologie très simple pour un réseau neuronal artificiel ; de nombreuses autres topologies peuvent mettre en œuvre des algorithmes plus complexes (Fig. 13.7). Actuellement, des réseaux neuronaux artificiels comportant des milliers, voire des millions de neurones disposés en plusieurs couches sont utilisés pour mettre en œuvre l'apprentissage profond (*deep learning*). Dans cette section, nous présentons une vue d'ensemble de certaines topologies d'ANN.

FIG. 13.8 – *Multilayer ANN*FIG. 13.9 – *ANN avec mémoire*

13.4.1 Topologie multicouche

La figure 13.8 montre un ANN avec plusieurs couches de neurones. Les couches supplémentaires permettent d'effectuer des calculs plus complexes qu'une seule couche. Par exemple, avec une seule couche, il n'est pas possible de faire avancer le robot lorsqu'un seul capteur détecte un obstacle et de le faire reculer lorsque plusieurs capteurs détectent un obstacle. La raison en est que la fonction d'une seule couche reliant les capteurs et les moteurs est monotone, c'est-à-dire qu'elle peut faire en sorte que le moteur aille plus vite lorsque l'entrée du capteur augmente ou plus lentement lorsque l'entrée du capteur augmente, mais pas les deux à la fois. La couche de neurones connectée à la sortie est appelée *couche de sortie* tandis que les couches internes sont appelées *couches cachées*.

Activity 13.6: ANNs multicouches

- L'objectif de cette activité est de comprendre comment les ANN multicouches peuvent effectuer des calculs qu'un ANN monocouche ne peut pas faire. Pour cette activité, on suppose que les entrées x_i sont comprises entre -2.0 et 2.0 , que les poids w_i sont compris entre -1.0 et 1.0 et que les fonctions f limitent les valeurs de sortie à l'intervalle -1.0 et 1.0 .
- Pour l'ANN composé d'un seul neurone (13.2) avec $w_1 = -0.5$, calculer y_1 pour des entrées par incrément de 0.2 : $x_1 = -2.0, -1.8, \dots, 0.0, \dots, 1.8, 2.0$. Tracez les résultats dans un graphique.
- Répéter le calcul pour plusieurs valeurs de w_1 . Que pouvez-vous dire sur la relation entre la sortie et l'entrée ?
- Considérons l'ANN à deux couches représenté sur la Fig. 13.10 avec des poids :

$$w_{11} = 1, w_{12} = 0.5, w_{21} = 1, w_{22} = -1 .$$

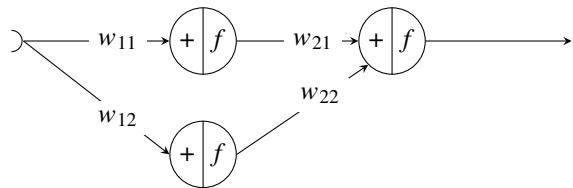


FIG. 13.10 – ANN à deux couches

Calculez les valeurs et dessinez les graphiques des sorties des neurones de la couche cachée (les neurones de gauche) et de la couche de sortie (le neurone de droite). Pouvez-vous obtenir la même sortie avec un ANN à une seule couche ?

Activity 13.7: Multilayer ANN for obstacle avoidance

- Concevoir un ANN qui met en œuvre le comportement suivant d'un robot : Il y a deux capteurs frontaux. Lorsqu'un objet est détecté devant l'un des capteurs, le robot tourne pour éviter l'objet, mais lorsqu'un objet est détecté par les deux capteurs, le robot recule.

13.4.2 Mémoire

Un réseau neuronal artificiel peut avoir des *connexions récurrentes* entre la sortie d'un neurone et l'entrée d'un neurone de la même couche (y compris lui-même). Les connexions récurrentes peuvent être utilisées pour mettre en œuvre la mémoire. Considérons le véhicule de Braitenberg pour l'évitement d'obstacles (Fig. 13.6). Il ne tourne que lorsque des obstacles sont détectés par les capteurs. Lorsqu'ils ne sont plus détectés, le robot ne continue pas à tourner. En ajoutant des connexions récurrentes, nous pouvons introduire un effet de mémoire qui permet au robot de continuer à tourner. Supposons que chacun des capteurs provoque une entrée de 0,75 dans les neurones et que la sortie soit saturée à 1,0 par la fonction de sortie non linéaire. Si les capteurs ne détectent plus l'obstacle, les entrées deviennent 0, mais la connexion récurrente ajoute une entrée de 1,0, de sorte que la sortie reste 1,0.

Activity 13.8: ANN avec mémoire

- Considérons le réseau de la Fig. 13.9 avec une fonction de sortie qui sature à 0 et 1. Les entrées et la plupart des poids sont également compris

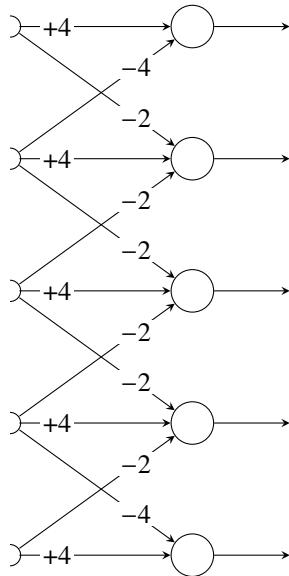


FIG. 13.11 – ANN pour le filtrage spatial

entre 0 et 1. Que se passe-t-il si le poids des connexions récurrentes de la figure est supérieur à 1 ? Que se passe-t-il s'il est compris entre 0 et 1 ?

- Modifier l'implémentation du réseau de la Fig. 13.6 pour ajouter des connexions récurrentes sur les deux neurones de sortie. Quel est leur effet sur le comportement d'évitement des obstacles du robot ?

13.4.3 Filtre spatial

Une caméra est un dispositif de détection constitué d'un grand nombre de capteurs adjacents (un pour chaque pixel). Les valeurs des capteurs peuvent constituer les entrées d'un ANN comportant un grand nombre de neurones dans la première couche (Fig. 13.11). Les pixels proches seront pris en compte par les neurones adjacents. Le réseau peut être utilisé pour extraire des caractéristiques locales telles que les différences d'intensité entre les pixels adjacents d'une image, et cette propriété locale peut être utilisée pour des tâches telles que l'identification des bords de l'image. Le nombre de couches peut être de un ou plusieurs. Cette topologie de neurones est appelée *filtre spatial* car elle peut être utilisée comme filtre avant une couche qui met en œuvre un algorithme d'évitement d'obstacles.

Exemple L'ANN de la Fig. 13.11 peut être utilisé pour distinguer les objets étroits des objets larges. Par exemple, un pied de chaise et un mur sont détectés comme des objets, mais le premier est un obstacle qui peut être évité par une séquence de virages, alors qu'un mur ne peut pas être évité et que le robot doit donc tourner autour ou suivre le mur.

Supposons que le pied de la chaise soit détecté par le capteur du milieu avec une valeur de 60, mais comme le pied est étroit, les autres capteurs renvoient la valeur 0. Les valeurs de sortie de l'ANN (de haut en bas) sont les suivantes :

$$\begin{aligned}(0 \times 4) + (0 \times -4) &= 0 \\ (0 \times -2) + (0 \times 4) + (60 \times -2) &= -120 \\ (0 \times -2) + (60 \times 4) + (0 \times -2) &= +240 \\ (60 \times -2) + (0 \times 4) + (0 \times -2) &= -120 \\ (0 \times 4) + (0 \times -4) &= 0.\end{aligned}$$

Lorsque le robot s'approche d'un mur, tous les capteurs renvoient plus ou moins les mêmes valeurs, par exemple 45, 50, 40, 55, 50. Les valeurs de sortie de l'ANN sont les suivantes

$$\begin{aligned}(45 \times 4) + (50 \times -4) &= -20 \\ (45 \times -2) + (50 \times 4) + (40 \times -2) &= +30 \\ (50 \times -2) + (40 \times 4) + (55 \times -2) &= -50 \\ (40 \times -2) + (55 \times 4) + (50 \times -2) &= +40 \\ (55 \times -4) + (50 \times 4) &= -20.\end{aligned}$$

Même si 48, la valeur moyenne renvoyée par les capteurs détectant le mur, est à peu près la même que la valeur 60 renvoyée lors de la détection du pied de la chaise, les sorties des ANN se distinguent clairement. Le premier ensemble de valeurs présente un pic élevé entouré de voisins aux valeurs négatives importantes, tandis que le second est un ensemble de valeurs relativement plates comprises entre 50 et 40. La couche de neurones peut identifier en toute confiance si l'objet est étroit ou large.

Activity 13.9: ANN pour le filtrage spatial

- Mettre en œuvre l'ANN pour le filtrage spatial dans la Fig. 13.11.
- Les entrées de l'ANN sont les relevés de cinq capteurs de proximité orientés vers l'avant. Si un seul capteur détecte un objet, le robot se tourne pour faire face à l'objet. Si le capteur central est celui qui détecte l'objet, le robot avance.
- Mettre en œuvre trois comportements du robot lorsqu'il détecte un mur défini comme les cinq capteurs détectant un objet :

- Le robot s'arrête.
- Le robot avance.
- Le robot recule.

Rappelez-vous qu'il n'y a pas d'énoncés if dans un réseau neuronal artificiel ; vous pouvez seulement ajouter des neurones supplémentaires ou modifier les poids associés aux entrées des neurones. Regardez à nouveau l'activité 13.7 qui a utilisé deux niveaux de neurones pour mettre en œuvre un comportement similaire.

- La mise en œuvre impliquera l'ajout de deux neurones supplémentaires dont les entrées sont les sorties de la première couche. La sortie du premier neurone définira la puissance du moteur gauche et la sortie du deuxième neurone définira la puissance du moteur droit.
- Que se passe-t-il si un objet est détecté par deux capteurs adjacents ?

13.5 Apprentissage

Le réglage manuel des poids est difficile, même pour de très petits réseaux tels que ceux présentés dans les sections précédentes. Dans les organismes biologiques, les synapses ont une plasticité qui permet l'apprentissage. La puissance des réseaux neuronaux artificiels vient de leur capacité à apprendre, contrairement aux algorithmes ordinaires qui doivent être spécifiés dans les moindres détails. Il existe de nombreuses techniques d'apprentissage dans les RNA ; nous décrivons dans cette section l'une des techniques les plus simples et montrons comment elle peut être utilisée dans le réseau neuronal pour l'évitement d'obstacles.

13.5.1 Catégories d'algorithmes d'apprentissage

Il existe trois grandes catégories d'algorithmes d'apprentissage :

- **Apprentissage supervisé** est applicable lorsque nous savons quelle sortie est attendue pour un ensemble d'entrées. L'erreur entre les sorties souhaitées et les sorties réelles est utilisée pour corriger les poids afin de réduire l'erreur. Pourquoi est-il nécessaire d'entraîner un réseau si nous savons déjà comment il doit se comporter ? L'une des raisons est que le réseau doit fournir des résultats dans des situations pour lesquelles il n'a pas été formé. Si les poids sont ajustés de manière à ce que le réseau se comporte correctement sur des entrées connues, il est raisonnable de supposer que son comportement sera plus ou moins correct sur d'autres entrées. Une deuxième raison d'entraîner un réseau est de simplifier le processus d'apprentissage : plutôt que de relier directement les sorties $\{y_i\}$ aux valeurs spécifiques des entrées $\{x_i\}$, il est

plus facile de placer le réseau dans plusieurs situations différentes et de lui indiquer quelles sorties sont attendues dans chaque situation.

- Dans **reinforcement learning**, nous ne spécifions pas la valeur de sortie exacte dans chaque situation ; au lieu de cela, nous indiquons simplement au réseau si la sortie qu'il calcule est bonne ou non. Le renforcement est approprié lorsque nous pouvons facilement distinguer un comportement correct d'un comportement incorrect, mais nous ne nous soucions pas vraiment de la valeur exacte de la sortie pour chaque situation. Dans la section suivante, nous présentons l'apprentissage par renforcement pour l'évitement d'un obstacle par un robot ; pour cette tâche, il suffit que le robot évite l'obstacle et nous ne nous soucions pas des réglages du moteur fournis par le réseau tant que le comportement est correct.
- **Apprentissage non supervisé** est l'apprentissage sans rétroaction externe, où le réseau s'adapte à un grand nombre d'entrées. L'apprentissage non supervisé n'est pas approprié pour atteindre des objectifs spécifiques ; il est plutôt utilisé dans les problèmes de classification où le réseau est présenté avec des données brutes et tente de trouver des tendances dans les données. Cette approche de l'apprentissage est le sujet du Chap. 14.

13.5.2 La règle de Hebbian pour l'apprentissage dans les ANNs

La règle de Hebbian est une technique d'apprentissage simple pour les ANN. Il s'agit d'une forme d'apprentissage par renforcement qui modifie les poids des connexions entre les neurones. Lorsque le réseau fait quelque chose de bien, nous renforçons cette bonne réponse : si la valeur de sortie de deux neurones connectés est similaire, nous augmentons le poids de la connexion qui les relie, tandis que si elle est différente, nous diminuons le poids. Si le robot fait quelque chose de mal, nous pouvons soit diminuer les poids des neurones connectés similaires, soit ne rien faire.

La variation du poids de la connexion entre le neurone k et le neurone j est décrite par l'équation :

$$\Delta w_{kj} = \alpha y_k x_j,$$

où w_{kj} est le poids reliant les neurones k et j , Δw_{kj} est la variation de w_{kj} , y_k est la sortie du neurone k et x_j l'entrée du neurone j , et α est une constante qui définit la vitesse d'apprentissage.

La règle de Hebbian est applicable sous deux conditions :

- Le robot explore son environnement, rencontrant diverses situations, chacune avec ses propres entrées pour lesquelles le réseau calcule un ensemble de sorties.
- Le robot reçoit des informations sur les comportements qui sont bons et ceux qui ne le sont pas.

L'évaluation de la qualité du comportement du robot peut être effectuée par un observateur humain qui donne manuellement son avis ; un système automatique peut également être utilisé pour évaluer le comportement. Par exemple, pour apprendre au robot à éviter les obstacles, une caméra externe peut être utilisée pour observer le robot et évaluer son comportement. Le comportement est qualifié de mauvais lorsque le robot s'approche d'un obstacle et de bon lorsque le robot s'éloigne de tous les obstacles. Il est important de comprendre que ce qui est évalué n'est pas l'état du robot (proche ou éloigné d'un obstacle), mais plutôt son comportement (approche ou évitement d'un obstacle). En effet, les connexions du réseau neuronal génèrent un comportement basé sur l'état mesuré par les capteurs.

Apprendre à éviter un obstacle

Supposons que nous voulions apprendre à un robot à éviter un obstacle. Une solution consisterait à laisser le robot se déplacer au hasard dans l'environnement, puis à toucher une touche lorsqu'il réussit à éviter l'obstacle et une autre lorsqu'il s'écrase contre l'obstacle. Le problème de cette approche est qu'il faudra probablement beaucoup de temps pour que le robot adopte un comportement que l'on peut qualifier de positif (éviter l'obstacle) ou de négatif (s'écraser contre l'obstacle).

Alternativement, nous pouvons présenter au robot plusieurs situations connues et le comportement requis : (1) détecter un obstacle sur la gauche et tourner à droite est bon ; (2) détecter un obstacle sur la droite et tourner à gauche est bon ; (3) détecter un obstacle devant et reculer est bon ; (4) détecter un obstacle devant et avancer est mauvais.

Cela ressemble à de l'apprentissage supervisé mais ce n'en est pas, car le retour d'information vers le robot n'est utilisé que pour renforcer les poids liés à un bon comportement. L'apprentissage supervisé consisterait à quantifier l'erreur entre les sorties souhaitées et les sorties réelles (vers les moteurs dans ce cas), et à utiliser cette erreur pour ajuster les poids afin de calculer des sorties exactes. Le retour d'information dans l'apprentissage par renforcement est binaire : le comportement est bon ou non.

L'algorithme d'évitement d'obstacles

Démontrons maintenant la règle de Hebbian pour l'apprentissage sur le problème de l'évitement d'obstacles. La Fig. 13.12 est similaire à la Fig. 13.6 sauf que des capteurs de proximité ont été ajoutés à l'arrière du robot. Nous avons également modifié la notation des poids pour les rendre plus appropriés à l'expression de la règle de Hebbian ; en particulier, les signes négatifs ont été absorbés dans les poids.

L'algorithme d'évitement des obstacles est implémenté à l'aide de plusieurs processus simultanés et sera présenté sous la forme d'un ensemble de trois algorithmes. L'algorithme 13.1 met en œuvre un ANN qui lit les entrées des capteurs et calcule

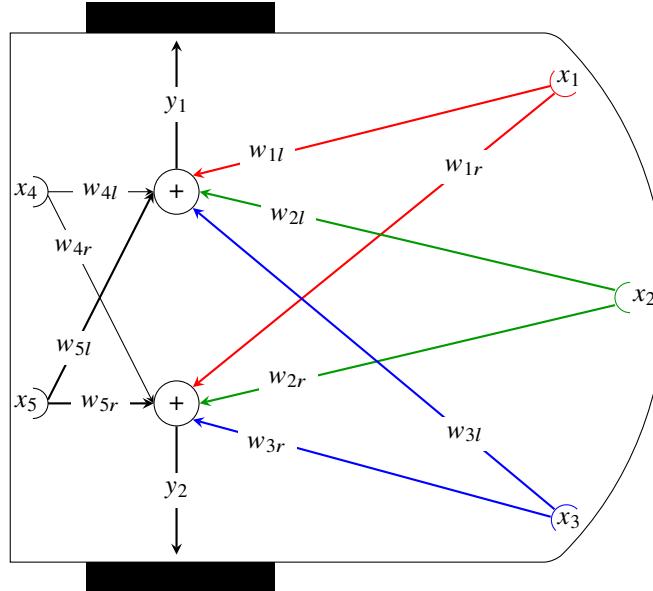


FIG. 13.12 – Réseau neuronal pour la démonstration de l'apprentissage Hebbien

les sorties vers les moteurs. Les nombres d'entrées et de sorties sont tirés de la Fig. 13.12. L'algorithme 13.2 reçoit des évaluations du comportement du robot de la part d'un humain. L'algorithme 13.3 effectue les calculs de la règle de Hebbian pour l'apprentissage.

Dans l'algorithme 13.1, une minuterie est réglée sur une période telle que 100 millisecondes. Le minuteur est décrémenté par le système d'exploitation (non illustré) et lorsqu'il expire, les sorties y_1 et y_2 sont calculées par l'13.3 ci-dessous. Ces sorties sont ensuite utilisées pour régler la puissance des moteurs gauche et droit et, enfin, la minuterie est réinitialisée.

Il y a cinq capteurs qui sont lus dans les cinq variables d'entrée :

- $x_1 \leftarrow$ capteur avant gauche
- $x_2 \leftarrow$ capteur central avant
- $x_3 \leftarrow$ capteur avant droit
- $x_4 \leftarrow$ capteur arrière gauche
- $x_5 \leftarrow$ capteur arrière droit

Nous supposons que les valeurs des capteurs sont comprises entre 0 (obstacle non détecté) et 100 (obstacle très proche), et que les valeurs des puissances motrices sont comprises entre -100 (pleine puissance en marche arrière) et 100 (pleine puissance en marche avant). Si le calcul aboutit à une saturation, les valeurs sont tronquées aux extrémités de la plage, c'est-à-dire qu'une valeur inférieure à -100 devient -100 et

Algorithm 13.1: ANN pour l'évitement d'obstacles

```

integer period ← ... // Timer period (ms)
integer timer ← period
float array[5]    $\vec{x}$ 
float array[2]    $\vec{y}$ 
float array[2,5]  $\vec{W}$ 

1: when timer expires
2:    $\vec{x} \leftarrow$  sensor values
3:    $\vec{y} \leftarrow \vec{W} \vec{x}$ 
4:   left-motor-power ←  $\vec{y}[1]$ 
5:   right-motor-power ←  $\vec{y}[2]$ 
6:   timer ← period

```

une valeur supérieure à 100 devient 100.¹ Rappelons qu'un robot à entraînement différentiel tourne à droite en fixant y_1 (la puissance du moteur gauche) à 100 et y_2 (la puissance du moteur droit) à -100, et de la même manière pour un virage à gauche.

Pour simplifier la présentation de l'algorithme 13.1, nous utilisons la notation vectorielle où les entrées sont données sous la forme d'un vecteur à une seule colonne :

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}.$$

En se référant à nouveau à la Fig.13.12, le calcul des sorties est donné par :

$$y_1 \leftarrow w_{1l}x_1 + w_{2l}x_2 + w_{3l}x_3 + w_{4l}x_4 + w_{5l}x_5 \quad (13.1)$$

$$y_2 \leftarrow w_{1r}x_1 + w_{2r}x_2 + w_{3r}x_3 + w_{4r}x_4 + w_{5r}x_5. \quad (13.2)$$

1. Algorithm 13.1 a déclaré toutes les variables comme float parce que les poids sont des nombres à virgule flottante. Si les entrées du capteur et les sorties du moteur sont des nombres entiers, une conversion de type sera nécessaire.

Exprimé en notation vectorielle, cela donne

$$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} w_{1l} & w_{2l} & w_{3l} & w_{4l} & w_{5l} \\ w_{1r} & w_{2r} & w_{3r} & w_{4r} & w_{5r} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \vec{W} \vec{x}. \quad (13.3)$$

Pour que l'algorithme apprenne, le feedback est utilisé pour modifier les poids \vec{W} (Algorithmes 13.2, 13.3). Supposons qu'il y ait quatre boutons sur le robot ou sur une télécommande, un pour chaque direction : avant, arrière, gauche et droite. Chaque fois que nous constatons que le robot se trouve dans une situation qui exige un certain comportement, nous appuyons sur le bouton correspondant. Par exemple, si le capteur gauche détecte un obstacle, le robot doit tourner à droite. Pour mettre cela en œuvre, il existe un processus pour chaque bouton. Ces processus sont présentés ensemble dans l'algorithme 13.2, où les barres obliques vers l'avant / séparent les événements et les actions correspondants.

Algorithm 13.2: Retour d'information sur le comportement du robot

```

1: when button {forward / backward / left / right} touched
2:   y1 ← {100 / -100 / -100 / 100}
3:   y2 ← {100 / -100 / 100 / -100}

```

La phase suivante de l'algorithme consiste à mettre à jour les poids de connexion conformément à la règle de Hebbian (Algorithm 13.3).

Algorithm 13.3: Application de la règle de Hebbian

```

1:    $\vec{x} \leftarrow$  sensor values
2:   for  $j$  in {1, 2, 3, 4, 5}
3:      $w_{jl} \leftarrow w_{jl} + \alpha y_1 x_j$ 
4:      $w_{jr} \leftarrow w_{jr} + \alpha y_2 x_j$ 

```

Exemple Supposons qu'au départ les poids soient tous nuls. D'après les équations 13.1–13.2, les sorties sont nulles et le robot ne se déplace pas.

Supposons maintenant qu'un obstacle soit placé devant le capteur gauche de sorte que $x_1 = 100$ tandis que $x_2 = x_3 = x_4 = x_5 = 0$. Sans rétroaction, il ne se passera rien puisque les poids sont toujours nuls. Si nous appuyons sur le bouton de droite (informant le robot que le comportement correct est de tourner à droite), les sorties sont réglées sur $y_1 = 100, y_2 = -100$ pour tourner à droite, ce qui entraîne les changements suivants dans les poids (en supposant un facteur d'apprentissage

$\alpha = 0.0001$) :

$$\begin{aligned} w_{1l} &\leftarrow 0 + (0.0001 \times 100 \times 100) = 10 \\ w_{1r} &\leftarrow 0 + (0.0001 \times -100 \times 100) = -10. \end{aligned}$$

La prochaine fois qu'un obstacle sera détecté par le capteur gauche, les sorties seront différentes de zéro :

$$\begin{aligned} y_1 &\leftarrow (10 \times 100) + 0 + 0 + 0 + 0 = 1000 \\ y_2 &\leftarrow (-10 \times 100) + 0 + 0 + 0 + 0 = -1000. \end{aligned}$$

Après avoir tronqué à 100 et -100, ces sorties amèneront le robot à tourner à droite.

Le facteur d'apprentissage α détermine l'ampleur de l'effet de $y_k x_j$ sur les valeurs de w_{kj} . Des valeurs plus élevées du facteur entraînent des effets plus importants et donc un apprentissage plus rapide. Bien que l'on puisse penser qu'un apprentissage plus rapide est toujours préférable, si l'apprentissage est trop rapide, il peut entraîner des changements indésirables tels que l'oubli des bonnes situations antérieures ou une forte accentuation des erreurs. Le facteur d'apprentissage doit être ajusté pour obtenir un apprentissage optimal.

Activity 13.10: Apprentissage hébraïque pour l'évitement d'obstacles

- Implémentez les algorithmes 13.1–13.3 et apprenez à votre robot à éviter les obstacles.
- Modifier le programme pour qu'il *apprenne* à avancer lorsqu'il ne détecte pas d'obstacle.

13.6 Résumé

Les robots autonomes doivent fonctionner dans des environnements caractérisés par un haut degré d'incertitude. C'est pourquoi il est difficile de spécifier des algorithmes précis pour le comportement des robots. Les réseaux de neurones artificiels peuvent mettre en œuvre le comportement requis dans un environnement incertain en apprenant, c'est-à-dire en modifiant et en améliorant l'algorithme au fur et à mesure que le robot rencontre des situations supplémentaires. La structure des réseaux neuronaux artificiels rend l'apprentissage techniquement simple : Un ANN est composé d'un grand nombre de petits composants simples appelés neurones et l'apprentissage est réalisé en modifiant les poids attribués aux connexions entre les neurones.

L'apprentissage peut être supervisé, par renforcement ou non supervisé. L'apprentissage par renforcement est approprié pour l'apprentissage du comportement

robotique car il exige du concepteur qu'il spécifie seulement si un comportement observé est bon ou mauvais, sans quantifier le comportement. La règle de Hebbian modifie les poids reliant les neurones en multipliant la sortie d'un neurone par l'entrée du neurone auquel il est connecté. Le résultat est multiplié par un facteur d'apprentissage qui détermine l'ampleur de la modification du poids et donc le taux d'apprentissage.

13.7 Lecture complémentaire

Haykin [20] et Rojas [37] sont des manuels complets sur les réseaux neuronaux. David Kriesel a écrit un tutoriel en ligne [28] qui peut être téléchargé gratuitement.

Chapitre 14

Apprentissage machine

Considérons un robot qui reconnaît et saisit des objets jaunes (Fig. 14.1). Il peut utiliser une caméra couleur pour identifier les objets jaunes, mais les objets apparaîtront différemment selon l'environnement, par exemple à la lumière du soleil, dans une pièce sombre ou dans une salle d'exposition. En outre, il est difficile de définir précisément ce que signifie "jaune" : quelle est la limite entre le jaune et le jaune citron ou entre le jaune et l'orange ? Plutôt que d'écrire des instructions détaillées pour le robot, nous préférerions que le robot apprenne la reconnaissance des couleurs au fur et à mesure qu'il exécute la tâche, afin qu'il puisse s'adapter à l'environnement dans lequel la tâche a lieu. Plus précisément, nous voulons concevoir un *algorithme de classification* qui peut être *entraîné* à effectuer la tâche sans fournir tous les détails à l'avance.

Les algorithmes de classification sont un sujet central de l'apprentissage automatique, un domaine de l'informatique et des statistiques qui développe des calculs pour reconnaître des modèles et prédire des résultats sans programmation explicite. Ces algorithmes extraient des *règles* des données brutes acquises par le système au cours d'une période de formation. Les règles sont ensuite utilisées pour classer un

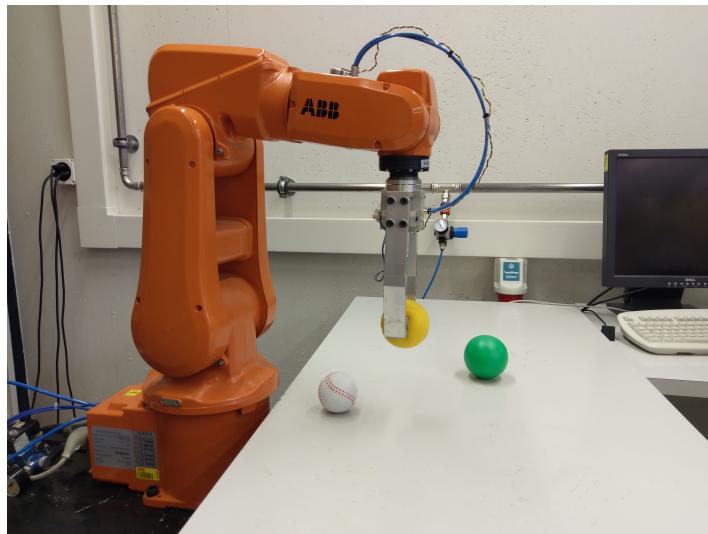


FIG. 14.1 – Bras robotique triant des balles colorées



FIG. 14.2 – *Distinguer deux nuances de gris*

nouvel objet et prendre les mesures appropriées en fonction de la classe de l'objet. Pour la tâche de reconnaissance des couleurs, nous entraînons le robot en lui présentant des objets de différentes couleurs et en lui disant quels objets sont jaunes et lesquels ne le sont pas. L'algorithme d'apprentissage automatique génère une règle de classification des couleurs. Lorsqu'on lui présente de nouveaux objets, il utilise la règle pour décider quels objets sont jaunes et lesquels ne le sont pas.

Le chapitre précédent a présenté les réseaux neuronaux artificiels qui réalisent une forme d'apprentissage automatique basée sur le renforcement. Dans ce chapitre, nous abordons les techniques statistiques basées sur l'apprentissage supervisé : pendant la période d'apprentissage, nous donnons au robot des réponses précises, par exemple si un objet est jaune ou non. La section 14.1 introduit les techniques statistiques en développant un algorithme permettant de distinguer les objets de deux couleurs. Nous présentons une technique d'apprentissage automatique appelée *analyse discriminante linéaire (LDA)*. Les sections 14.2–14.3 présentent la LDA dans le même contexte de distinction entre deux couleurs. L'ADL repose sur l'hypothèse que les données possèdent des propriétés statistiques spécifiques ; si ce n'est pas le cas, *perceptrons* peut être utilisé pour la classification, comme décrit dans la section 14.4.

Ce chapitre suppose que vous connaissez les concepts de *moyenne*, *variance* et *covariance*. Des tutoriels sur ces concepts apparaissent dans les annexes B.3–B.4.

14.1 Distinction entre deux couleurs

Nous commençons par le problème de la distinction entre des boules jaunes et des boules non jaunes. Pour simplifier la tâche, nous modifions le problème en distinguant les zones gris foncé des zones gris clair imprimées sur du papier collé au sol (Fig. 14.2). Le robot utilise deux capteurs au sol qui échantillonnent la lumière réfléchie lorsque le robot se déplace sur les deux zones.

La figure 14.3 montre un tracé des valeurs renvoyées par l'échantillonnage des deux capteurs.¹ Le robot met environ 70 secondes pour se déplacer de gauche à droite, en échantillonnant la lumière réfléchie une fois par seconde. Il est facile de

1. Les données utilisées dans ce chapitre sont des données réelles tirées d'une expérience avec le robot Thymio.

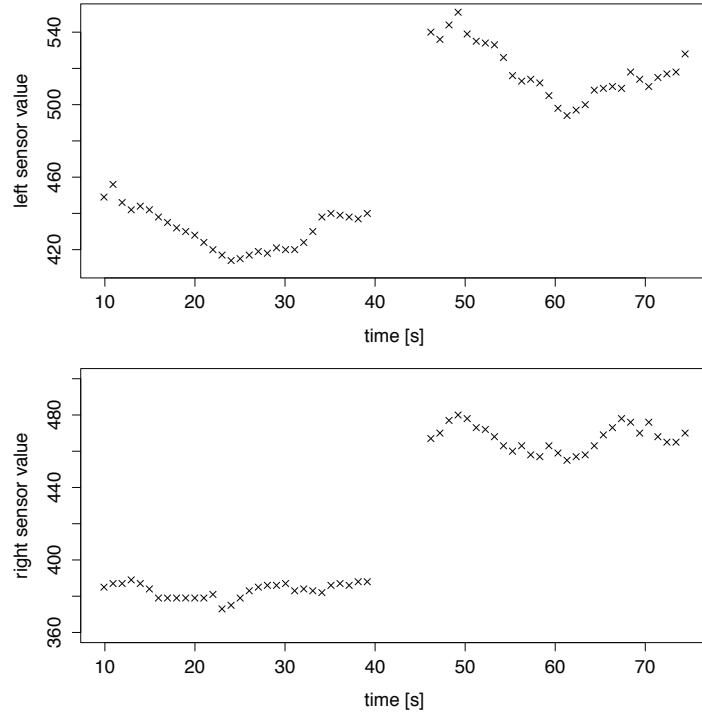


FIG. 14.3 – Plots de la lumière réfléchie en fonction du temps pour le capteur gauche (en haut) et le capteur droit (en bas)

constater que les données présentent des variations importantes, qui sont probablement dues au bruit des capteurs et à une impression irrégulière. La variabilité des résultats renvoyés par les deux capteurs est encore plus problématique. Comment le robot peut-il apprendre à distinguer les nuances de gris compte tenu de la variabilité des échantillons et des capteurs ? Nous voulons créer automatiquement une règle pour distinguer les deux nuances de gris.

14.1.1 Un discriminant basé sur les moyennes

En examinant les graphiques de la Fig. 14.3, il est facile de voir quels sont les échantillons qui se trouvent dans la zone gris foncé et ceux qui se trouvent dans la zone gris clair. Pour le capteur de gauche, les valeurs de la zone gris clair sont comprises entre 500 et 550, tandis que les valeurs de la zone gris foncé sont comprises entre 410 et 460. Pour le capteur droit, les plages sont de 460–480 et 380–400. Pour le capteur gauche, un seuil de 480 permettrait de distinguer clairement le gris clair du gris foncé, tandis que pour le capteur droit, un seuil de 440 permettrait de distinguer

clairement le gris clair du gris foncé. Mais comment choisir automatiquement ces valeurs optimales et comment concilier les seuils des deux capteurs ?

Concentrons-nous d'abord sur le capteur gauche. Nous cherchons un *discriminant*, une valeur qui distingue les échantillons des deux couleurs. Considérons les valeurs \max_{dark} , la valeur maximale renvoyée par l'échantillonnage du gris foncé, et \min_{light} , la valeur minimale renvoyée par l'échantillonnage du gris clair. Sous l'hypothèse raisonnable que $\max_{dark} < \min_{light}$, toute valeur x telle que $\max_{dark} < x < \min_{light}$ permet de distinguer les deux nuances de gris. Le point médian entre les deux valeurs semble offrir le discriminant le plus robuste.

Sur la Fig. 14.3, nous voyons que $\max_{dark} \approx 460$ se produit à environ 10 s et $\min_{light} \approx 500$ se produit à environ 60 s, nous choisissons donc leur moyenne 480 comme discriminant. Bien que cela soit correct pour cet ensemble de données particulier, il n'est généralement pas judicieux d'utiliser les valeurs maximales et minimales car elles pourraient être des *outliers* : des valeurs extrêmes résultant de circonstances inhabituelles, telles qu'un trou dans le papier qui donnerait à tort une valeur très élevée dans la zone gris foncé.

Une meilleure solution consiste à utiliser "toutes" les données et la fonction la plus simple de toutes les données est la "moyenne" des valeurs. Soit $\mu_{foncé}$ la moyenne des échantillons gris foncé et μ_{clair} la moyenne des échantillons gris clair. Un bon discriminant Δ est le point médian entre les deux moyennes :

$$\Delta = \frac{\mu_{dark} + \mu_{light}}{2}.$$

Pour les données de la Fig. 14.3, les moyennes pour le capteur de gauche et le discriminant sont :²

$$\mu_{dark}^{left} = 431, \quad \mu_{light}^{left} = 519, \quad \Delta^{left} = \frac{431 + 519}{2} = 475.$$

Un calcul similaire permet d'obtenir le discriminant pour le capteur de droite :

$$\Delta^{right} = 425.$$

Afin d'obtenir une reconnaissance optimale, nous voulons un algorithme capable de décider automatiquement lequel des deux discriminants est le meilleur. Il s'agit d'un premier pas vers la méthode décrite dans la section 14.2, où un discriminant est calculé en combinant les données des deux capteurs.

Intuitivement, plus la différence entre les moyennes des zones claires et sombres est grande :

$$\left| \mu_{dark}^{left} - \mu_{light}^{left} \right|, \quad \left| \mu_{dark}^{right} - \mu_{light}^{right} \right|,$$

plus il sera facile de placer un discriminant entre les deux classes. La différence entre les moyennes du capteur de gauche (88) est un peu plus grande que la différence

2. Dans ce chapitre, les valeurs seront arrondies à l'entier le plus proche.

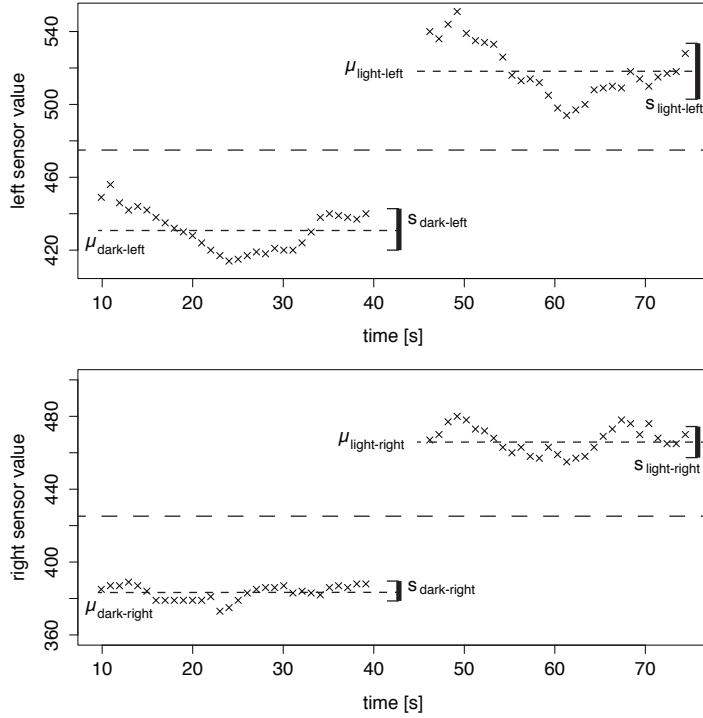


FIG. 14.4 – Figure 14.3 avec les moyennes (tirets courts), les variances (parenthèses), les discriminants (tirets longs).

entre les moyennes du capteur de droite (84). Cela nous amène à choisir le discriminant (475) calculé à partir des moyennes du capteur gauche. Cependant, d’après le graphique de la figure 14.4, il semble que ce ne soit pas le meilleur choix en raison de la grande variation dans les échantillons du capteur gauche.

14.1.2 A discriminant based on the means and variances

Un meilleur discriminant peut être obtenu si l’on considère non seulement la différence des moyennes, mais aussi la dispersion des valeurs de l’échantillon autour de la moyenne. C’est ce qu’on appelle la *variance* des échantillons. La variance s^2 d’un ensemble de valeurs $\{x_1, x_2, \dots, x_{n-1}, x_n\}$ est :³

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \mu)^2,$$

où μ est la moyenne des valeurs de l’ensemble.

3. Appendice B.3 explique pourquoi $n - 1$ est utilisé au lieu de n .

La variance calcule la moyenne des distances de chaque échantillon par rapport à la moyenne des échantillons. Les distances sont élevées au carré parce qu'un échantillon peut être supérieur ou inférieur à la moyenne, mais nous voulons une distance positive qui montre à quel point l'échantillon est éloigné de la moyenne.

Les parenthèses de la Fig. 14.4 montrent les quatre variances pour les ensembles d'échantillons des zones claires et sombres des capteurs gauche et droit.⁴ La différence entre les moyennes de gauche est légèrement supérieure à la différence entre les moyennes de droite :

$$\left| \mu_{dark}^{left} - \mu_{light}^{left} \right| > \left| \mu_{dark}^{right} - \mu_{light}^{right} \right|,$$

mais les variances du capteur de droite sont beaucoup plus petites que les variances correspondantes du capteur de gauche :

$$(s_{dark}^{right})^2 \ll (s_{dark}^{left})^2, \quad (s_{light}^{right})^2 \ll (s_{light}^{left})^2.$$

L'utilisation des variances permet une meilleure classification des deux ensembles, car un capteur avec moins de variance est plus stable, ce qui facilite la classification.

Un bon discriminant peut être obtenu en combinant les informations des moyennes et des variances. La qualité d'un discriminant J_k , pour $k = gauche, droite$, est donnée par :

$$J_k = \frac{(\mu_{dark}^k - \mu_{light}^k)^2}{(s_{dark}^k)^2 + (s_{light}^k)^2}. \quad (14.1)$$

Pour maximiser J , le numérateur - la distance entre les moyennes - doit être grand et le dénominateur - les variances des échantillons - doit être petit.

Le tableau 14.1 affiche les calculs pour l'ensemble de données de la Fig. 14.4. Le critère de qualité J pour le capteur de droite est beaucoup plus important que celui du capteur de gauche. Il s'ensuit que le point médian entre les moyennes du capteur droit :

$$\Delta^{right} = \frac{383 + 467}{2} = 425$$

est un meilleur discriminant que le point médian des moyennes du capteur gauche qui serait choisi en considérant uniquement les différences de moyennes $|\mu_{dark} - \mu_{light}|$, qui est légèrement plus grand pour le capteur gauche que pour le capteur droit.

14.1.3 Algorithme d'apprentissage de la distinction des couleurs

Ces calculs sont effectués par le robot lui-même, de sorte que le choix du meilleur discriminant et du meilleur capteur est automatique. Les détails du calcul sont donnés

4. Fig. 14.4 montre en fait la racine carrée de la variance appelée écart-type.

TAB. 14.1 – La différence entre les moyennes et les critères de qualité J

	left		right	
	dark	light	dark	light
μ	431	519	383	467
s^2	121	225	16	49
$ \mu_{dark} - \mu_{light} $		88		84
J		22		104

dans Algorithmes 14.1–14.2.⁵

Il y a deux classes C_1, C_2 et deux capteurs. Pendant la phase d'apprentissage, le robot échantillonne les zones des deux niveaux de gris indépendamment et calcule ensuite le critère de qualité J . L'échantillonnage et le calcul sont effectués pour chaque capteur, soit l'un après l'autre, soit simultanément. Après la phase d'apprentissage, le robot utilise le point médian des moyennes avec la meilleure valeur de J pour la reconnaissance des niveaux de gris.

Algorithm 14.1: Distinguer les classes (phase d'apprentissage)

```

float  $\vec{X}_1, \vec{X}_2$                                 // Sets of samples
float  $\mu_1, \mu_2$                                 // Means of  $C_1, C_2$ 
float  $s_1, s_2$                                 // Variances of  $C_1, C_2$ 
float  $\mu[2]$                                     // Means of  $\mu_1, \mu_2$ 
float  $J[2]$                                     // Criteria of quality
integer  $k$                                     // Index of  $\max(J[1], J[2])$ 

1: for sensor i=1, 2
2:   Collect a set of samples  $\vec{X}_1$  from  $C_1$ 
3:   Collect a set of samples  $\vec{X}_2$  from  $C_2$ 
4:   Compute means  $\mu_1$  of  $\vec{X}_1$  and  $\mu_2$  of  $\vec{X}_2$ 
5:   Compute variances  $s_1$  of  $\vec{X}_1$  and  $s_2$  of  $\vec{X}_2$ 
6:   Compute the mean  $\mu[i] = \frac{\mu_1 + \mu_2}{2}$ 
7:   Compute the criterion  $J[i]$  from Eq. 14.1
8:  $k \leftarrow$  index of  $\max(J[1], J[2])$ 
9: Output  $\mu[k], k$ 

```

5. Les variables en gras représentent des vecteurs ou des matrices.

Algorithm 14.2: Distinguer les classes (phase de reconnaissance)

```
float  $\mu \leftarrow$  input  $\mu[k]$  from the learning phase
float  $x$ 
```

```
1: loop
2:    $x \leftarrow$  get new sample
3:   if  $x < \mu$ 
4:     assign  $x$  to class  $C_1$ 
5:   else
6:     assign  $x$  to class  $C_2$ 
```

Activity 14.1: Caméléon robotique

- Construire un environnement comme le montre la Fig. 14.2. Imprimez deux feuilles de papier avec des niveaux de gris uniformes différents et collez-les au sol.
- Écrire un programme qui fait en sorte que le robot se déplace à une vitesse constante sur la zone d'une couleur et échantillonne la lumière réfléchie périodiquement. Répéter l'opération pour l'autre couleur.
- Tracer les données, calculer les moyennes et le discriminant.
- Implémenter un programme qui classe les mesures du capteur. Lorsque le robot classe une mesure, il affiche la couleur reconnue à la manière d'un caméléon (ou donne d'autres informations si le changement de couleur n'est pas possible).
- Appliquez la même méthode avec un deuxième capteur et comparez la séparabilité des classes en utilisant le critère J .
- Répétez l'exercice avec deux niveaux de gris très proches. Qu'observez-vous ?

14.2 Analyse discriminante linéaire

Dans la section précédente, nous avons classé des échantillons de deux niveaux de gris en nous basant sur les mesures d'un capteur sur deux ; le capteur a été choisi automatiquement sur la base d'un critère de qualité. Cette approche est simple mais pas optimale. Au lieu de choisir un discriminant basé sur un seul capteur, nous pouvons obtenir une meilleure reconnaissance en combinant les échantillons des deux capteurs. L'une de ces méthodes, appelée *analyse discriminante linéaire* (*LDA*), est basée sur les travaux pionniers réalisés en 1936 par le statisticien Ronald

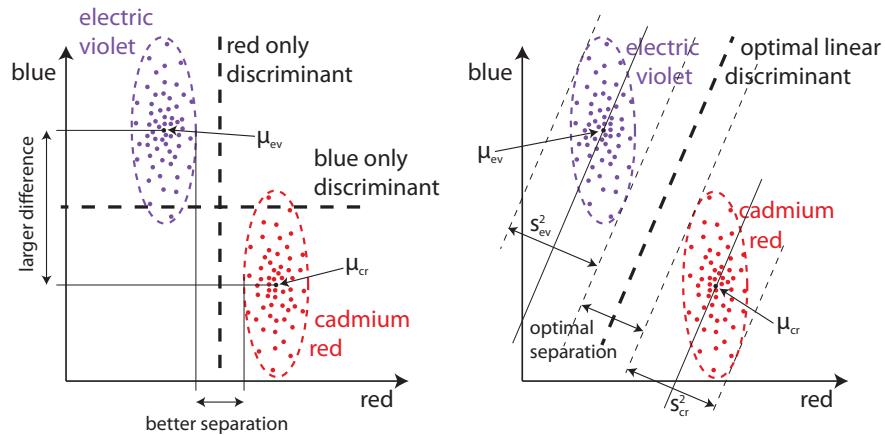


FIG. 14.5 – Un discriminant linéaire optimal pour distinguer deux couleurs

A. Fisher.

14.2.1 Motivation

Pour comprendre les avantages de la combinaison d'échantillons provenant de deux capteurs, supposons que nous devions classer des objets de deux couleurs : *violet électrique* (*ev*) et *rouge de cadmium* (*cr*). Le violet électrique est composé en grande partie de bleu et d'un peu de rouge, tandis que le rouge cadmium est composé en grande partie de rouge et d'un peu de bleu. Deux capteurs sont utilisés : l'un mesure le niveau de rouge et l'autre le niveau de bleu. Pour un ensemble d'échantillons, nous pouvons calculer leurs moyennes μ_j^k et leurs variances $(s_j^k)^2$, pour $j = ev, cr$, $k = bleu, rouge$.

Le graphique de gauche de la figure 14.5 montre des échantillons d'objets violet électrique contenus dans une ellipse en pointillés en haut à gauche et des échantillons d'objets rouge cadmium contenus dans une ellipse en pointillés en bas à droite. Les centres des ellipses sont les moyennes car les échantillons sont distribués symétriquement par rapport aux ellipses. La coordonnée y de μ_{ev} est la moyenne des échantillons des objets violets électriques par le capteur bleu et sa coordonnée x est la moyenne des échantillons de ces objets par le capteur rouge. Les moyennes des capteurs lors de l'échantillonnage des objets rouge cadmium sont affichées de la même manière. Il est facile de voir que les objets violets électriques ont plus de bleu (ils sont plus haut sur l'axe y), tandis que les objets rouges cadmium ont plus de rouge (ils sont plus loin sur l'axe x).

Le diagramme montre qu'il y a une plus grande différence entre les moyennes pour le capteur bleu qu'entre les moyennes pour le capteur rouge. À première vue,

il semble que l'utilisation du seul capteur bleu donnerait un meilleur discriminant. Cependant, ce n'est pas le cas : les lignes pointillées montrent que le discriminant rouge seul distingue complètement le violet électrique du rouge cadmium, tandis que le discriminant bleu seul classe à tort certains échantillons violets électriques en rouge cadmium (certains échantillons sont en dessous de la ligne) et classe à tort certains échantillons rouges cadmium en violet électrique (certains échantillons sont au-dessus de la ligne).

La raison de ce résultat non intuitif est que le capteur bleu renvoie des valeurs qui sont largement réparties (ont une grande variance), tandis que le capteur rouge renvoie des valeurs qui sont étroitement réparties (ont une petite variance), et nous avons vu dans la section 14.1 que la classification est meilleure si la variance est faible. Le graphique de droite de la Fig. 14.5 montre qu'en construisant un discriminant à partir des deux capteurs, il est possible de mieux séparer les objets violet électrique des objets rouge cadmium. Le discriminant est toujours linéaire (une ligne droite) mais sa pente n'est plus parallèle à l'un des axes. Cette droite est calculée en utilisant les variances ainsi que les moyennes. La méthode est appelée analyse discriminante linéaire car le discriminant est linéaire.

14.2.2 Le discriminant linéaire

14.5 est un graphique x - y de données échantillonées à partir de deux capteurs. Chaque valeur est tracée au point dont la coordonnée x est la valeur renvoyée par le capteur rouge et dont la coordonnée y est la valeur renvoyée par le capteur bleu. De même, la Fig. 14.6 est un graphique x - y des données des Figs. 14.3–14.4 qui ont été collectées lorsque le robot s'est déplacé sur deux zones grises. Dans ces graphiques, les valeurs des capteurs sont représentées en fonction du temps, mais le temps ne joue aucun rôle dans la classification, si ce n'est pour relier les échantillons qui ont été mesurés au même moment par les deux capteurs.

Dans la figure 14.6, la classification basée uniquement sur le capteur gauche correspond à la ligne verticale en pointillés, tandis que la classification basée uniquement sur le capteur droit correspond à la ligne horizontale en pointillés. Les lignes de séparation horizontale et verticale ne sont pas optimales. Supposons que l'on utilise la classification basée sur le capteur gauche (la ligne verticale) et que l'on considère un échantillon pour lequel le capteur gauche renvoie 470 et le capteur droit 460. L'échantillon sera classé en gris foncé même si la classification en gris clair est meilleure. Intuitivement, il est clair que la ligne diagonale pleine du graphique est un discriminant beaucoup plus précis que l'un ou l'autre des deux discriminants basés sur un seul capteur.

Comment un tel discriminant linéaire peut-il être défini mathématiquement afin d'être découvert automatiquement ?⁶ L'équation générale d'une droite dans le plan

6. La présentation suivante est abstraite et sera plus facile à comprendre si elle est lue en même temps

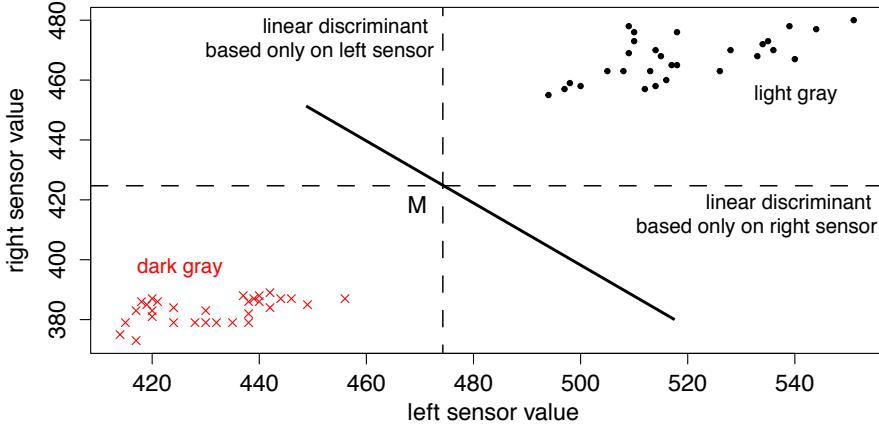


FIG. 14.6 – Tracé x-y des niveaux de gris avec des discriminants à capteur unique et un discriminant optimal

est $y = mx + a$, où m est la pente et a est l'intersection de la droite et de l'axe y lorsque $x = 0$. Une autre équation pour une droite est :

$$w_1x_1 + w_2x_2 = c, \quad (14.2)$$

où x_1 est l'axe horizontal pour les valeurs du capteur gauche, x_2 est l'axe vertical pour les valeurs du capteur droit. c est une constante et w_1, w_2 sont les coefficients des variables.

Il est pratique de représenter les coefficients sous la forme d'un vecteur colonne :

$$\vec{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}.$$

Dans cette représentation, le vecteur \vec{w} est normal à la droite discriminante et définit donc sa pente, tandis que la constante c permet au discriminant d'être n'importe quelle droite de cette pente, c'est-à-dire n'importe laquelle du nombre infini de droites parallèles ayant une pente donnée. Une fois la pente de \vec{w} déterminée, c est obtenu en entrant un point donné (x_1, x_2) dans l'14.2.

L'analyse discriminante linéaire définit automatiquement le vecteur \vec{w} et la constante c qui génèrent une ligne discriminante optimale entre les ensembles de données des deux classes. La première étape consiste à choisir un point sur la ligne discriminante. Par ce point, il existe un nombre infini de droites et nous devons choisir la droite dont la pente donne le discriminant optimal. Enfin, la valeur c peut être

que l'exemple numérique de la section 14.2.5.

calculée à partir de la pente et du point choisi. Les sous-sections suivantes décrivent chacune de ces étapes en détail.

14.2.3 Choisir un point pour le discriminant linéaire

Comment choisir un point ? L'analyse discriminante linéaire repose sur l'hypothèse que les valeurs des deux classes ont la même distribution. De manière informelle, lorsque l'on regarde un graphique x - y , les deux ensembles de points devraient avoir une taille et une forme similaires. Bien qu'il soit presque certain que les distributions ne seront pas exactement les mêmes (disons une distribution gaussienne) parce qu'elles résultent de mesures effectuées dans le monde réel, étant donné que les deux capteurs sont soumis aux mêmes types de variabilité (bruit du capteur, surface inégale du sol), il est probable qu'elles seront similaires.

Si les deux distributions sont similaires, les moyennes des échantillons de chaque capteur seront plus ou moins au même endroit par rapport aux distributions. La moyenne des moyennes de chaque capteur sera équidistante des points correspondants dans les ensembles de données. Le discriminant est choisi pour être une droite qui passe par le point M (Fig. 14.6) dont les coordonnées sont :

$$\left(\frac{\mu_{light}^{left} + \mu_{dark}^{left}}{2}, \frac{\mu_{light}^{right} + \mu_{dark}^{right}}{2} \right).$$

14.2.4 Choisir une pente pour le discriminant linéaire

Une fois que nous avons choisi le point M sur la droite discriminante, l'étape suivante consiste à choisir la pente de la droite. D'après la Fig. 14.6, nous voyons qu'il existe une infinité de droites passant par le point M qui permettraient de distinguer les deux ensembles d'échantillons. Quelle est la meilleure ligne en fonction des propriétés statistiques de nos données ?

Dans la section 14.1, nous avons cherché un moyen de décider entre deux discriminants, où chaque discriminant était une ligne parallèle à l'axe y au point médian entre les moyennes des valeurs renvoyées par un capteur (Fig. 14.4). La décision était basée sur le critère de qualité J_k (Eq. 14.1, répété ici par commodité) :

$$J_k = \frac{\left(\mu_{dark}^k - \mu_{light}^k \right)^2}{\left(s_{dark}^k \right)^2 + \left(s_{light}^k \right)^2}, \quad (14.3)$$

où $k = gauche, droite$. Le discriminant ayant la plus grande valeur de J_k a été choisi. Pour maximiser J_k , le numérateur - la distance entre les moyennes - doit être grand et le dénominateur - les variances des échantillons - doit être petit.

Maintenant, nous ne voulons plus calculer un critère de qualité basé sur les valeurs renvoyées par chaque capteur séparément, mais plutôt calculer un critère à

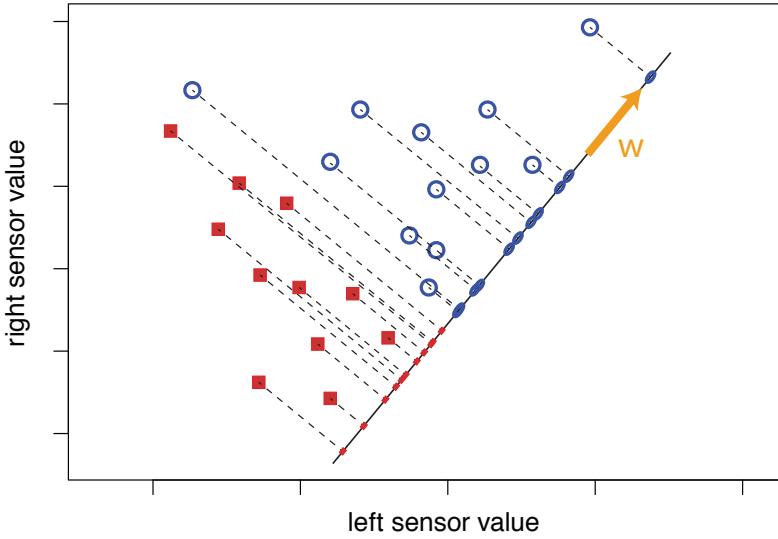


FIG. 14.7 – Projection des échantillons de deux classes sur une ligne définie par \vec{w} .

partir de toutes les valeurs renvoyées par les deux capteurs. La figure 14.7 montre un graphique x - y des valeurs de deux capteurs, où les valeurs des deux classes sont représentées par des carrés rouges et des cercles bleus. Il est clair qu'il y a un chevauchement important le long des axes x ou y séparément et qu'il est impossible de trouver des lignes parallèles aux axes qui permettent de distinguer les groupes. Cependant, si nous projetons les groupes de mesures sur la ligne définie par un vecteur :

$$\vec{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix},$$

les deux groupes peuvent être distingués par la ligne de séparation définie par l'Eq. 14.2 :

$$w_1x_1 + w_2x_2 = c,$$

où c est défini par un point de la ligne de projet situé entre les points rouge et bleu. Par analogie avec l'équation 14.3, nous devons définir un critère de qualité $J(\vec{w})$, de sorte que des valeurs plus élevées de $J(\vec{w})$ donnent de meilleures lignes discriminantes. Ensuite, nous devons trouver la valeur de \vec{w} qui maximise $J(\vec{w})$ en différenciant cette fonction, en mettant la dérivée à zéro et en résolvant pour \vec{w} .

La définition de $J(\vec{w})$ est basée sur les moyennes et variances des deux classes mais est trop complexe pour ce livre. Nous donnons sans preuve que la valeur de \vec{w} qui maximise $J(\vec{w})$ est :

$$\vec{w} = \vec{S}^{-1} (\vec{\mu}_{light} - \vec{\mu}_{dark}), \quad (14.4)$$

où :

$$\vec{\mu}_{light} = \begin{bmatrix} \mu_{light}^{left} \\ \mu_{light}^{right} \end{bmatrix}, \quad \vec{\mu}_{dark} = \begin{bmatrix} \mu_{dark}^{left} \\ \mu_{dark}^{right} \end{bmatrix}$$

sont les vecteurs moyens des deux classes et \vec{S}^{-1} est l'inverse de la moyenne des matrices de covariance des deux classes :⁷

$$\vec{S} = \frac{1}{2} \begin{bmatrix} s^2(x_{light}^{left}) & cov(x_{light}^{left}, x_{light}^{right}) \\ cov(x_{light}^{right}, x_{light}^{left}) & s^2(x_{light}^{right}) \end{bmatrix} + \frac{1}{2} \begin{bmatrix} s^2(x_{dark}^{left}) & cov(x_{dark}^{left}, x_{dark}^{right}) \\ cov(x_{dark}^{right}, x_{dark}^{left}) & s^2(x_{dark}^{right}) \end{bmatrix}.$$

Par rapport au capteur unique J_k , les moyennes sont des vecteurs à deux dimensions car nous avons une moyenne pour chacun des capteurs. La somme des variances devient la matrice de covariance, qui prend en compte à la fois les variances individuelles des deux capteurs et les covariances qui expriment la manière dont les deux capteurs sont liés.

Lorsque les valeurs de M et \vec{w} ont été calculées, il ne reste plus qu'à calculer la constante c pour définir complètement la ligne discriminante. La phase d'apprentissage de l'algorithme LDA est ainsi terminée. Dans la phase de reconnaissance, le robot utilise la ligne définie par \vec{w} et c pour classer les nouveaux échantillons.

Le calcul est formalisé dans les Algorithmes 14.3–14.4, où nous voulons distinguer entre deux classes C_1, C_2 en utilisant deux capteurs. Comparons cet algorithme avec l'Algorithm 14.1 : les deux ensembles d'échantillons \vec{X}_1, \vec{X}_2 et les moyennes $\vec{\mu}_1, \vec{\mu}_2$ et les variances \vec{s}_1, \vec{s}_2 sont des vecteurs à deux éléments, un élément pour chacun des capteurs.

14.2.5 Calcul d'un discriminant linéaire : exemple numérique

La figure 14.8 est un tracé des échantillons provenant de deux capteurs au sol, mesurés lorsqu'un robot se déplace sur deux zones grises très similaires, l'une étant noire à 83,6% et l'autre noire à 85%. Les niveaux sont si proches que l'œil humain ne peut les distinguer. Le discriminant linéaire calculé par l'algorithme 14.3 peut-il faire mieux ?

7. Voir Annexe B.4 pour la définition de la covariance et Sect. 14.2.5 pour un exemple numérique qui clarifiera les calculs.

Algorithm 14.3: Analyse discriminante linéaire (phase d'apprentissage)	
float array[n ₁ ,2] \vec{X}_1	// Samples of first area
float array[n ₂ ,2] \vec{X}_2	// Samples of second area
float array[2] $\vec{\mu}_1, \vec{\mu}_2$	// Means of C_1, C_2
float array[2] $\vec{\mu}$	// Mean of the means
float array[2] \vec{s}_1, \vec{s}_2	// Variances of C_1, C_2
float cov ₁ , cov ₂	// Covariances of C_1, C_2
float array[2] \vec{S}_{inv}	// Inverse of average
float c	// Constant of the line
1: Collect a set of n_1 samples \vec{X}_1 from C_1	
2: Collect a set of n_2 samples \vec{X}_2 from C_2	
3: Compute means $\vec{\mu}_1$ of \vec{X}_1 and $\vec{\mu}_2$ of \vec{X}_2	
4: $\vec{\mu} \leftarrow (\vec{\mu}_1 + \vec{\mu}_2)/2$	
5: Compute variances \vec{s}_1 of \vec{X}_1 and \vec{s}_2 of \vec{X}_2	
6: Compute covariances cov ₁ , cov ₂ of \vec{X}_1 and \vec{X}_2	
7: Compute \vec{S}_{inv} of covariance matrix	
8: Compute \vec{w} from Eq. 14.4	
9: Compute point M from $\vec{\mu}$	
10: Compute c from M and \vec{w}	
11: Output \vec{w} and c	

La classe C_1 est la zone gris clair et la classe C_2 est la zone gris foncé. Les éléments des ensembles de vecteurs $\vec{X}_1[n_1], \vec{X}_2[n_2]$ sont des vecteurs :

$$\vec{x} = \begin{bmatrix} x^{left} \\ x^{right} \end{bmatrix}$$

d'échantillons mesurés par les capteurs gauche et droit, où $n_1 = 192$ et $n_2 = 205$.

Nous commençons par calculer les moyennes des données de la Fig. 14.8 :

$$\vec{\mu}_1 = \frac{1}{192} \left(\begin{bmatrix} 389 \\ 324 \end{bmatrix} + \begin{bmatrix} 390 \\ 323 \end{bmatrix} + \dots + \begin{bmatrix} 389 \\ 373 \end{bmatrix} \right) \approx \begin{bmatrix} 400 \\ 339 \end{bmatrix}$$

$$\vec{\mu}_2 = \frac{1}{205} \left(\begin{bmatrix} 358 \\ 297 \end{bmatrix} + \begin{bmatrix} 358 \\ 296 \end{bmatrix} + \dots + \begin{bmatrix} 352 \\ 327 \end{bmatrix} \right) \approx \begin{bmatrix} 357 \\ 312 \end{bmatrix}.$$

Algorithm 14.4: Analyse discriminante linéaire (phase de reconnaissance)

```

float  $\vec{w}$  ← input from the learning phase
float  $c$  ← input from the learning phase
float  $\vec{x}$ 

1: loop
2:    $\vec{x} \leftarrow$  new sample
3:   if  $\vec{x} \cdot \vec{w} < c$ 
4:     assign  $\vec{x}$  to class  $C_1$ 
5:   else
6:     assign  $\vec{x}$  to class  $C_2$ 
```

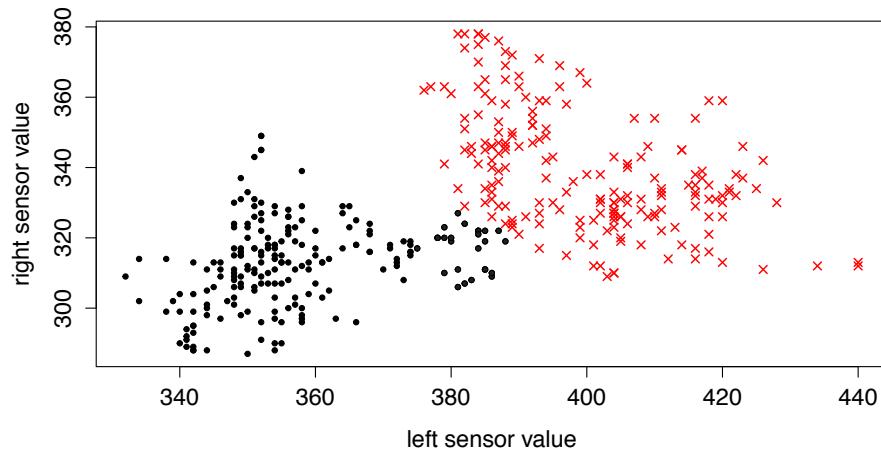


FIG. 14.8 – Zones grises similaires (points noirs de la zone la plus sombre, x rouges de la zone la plus claire)

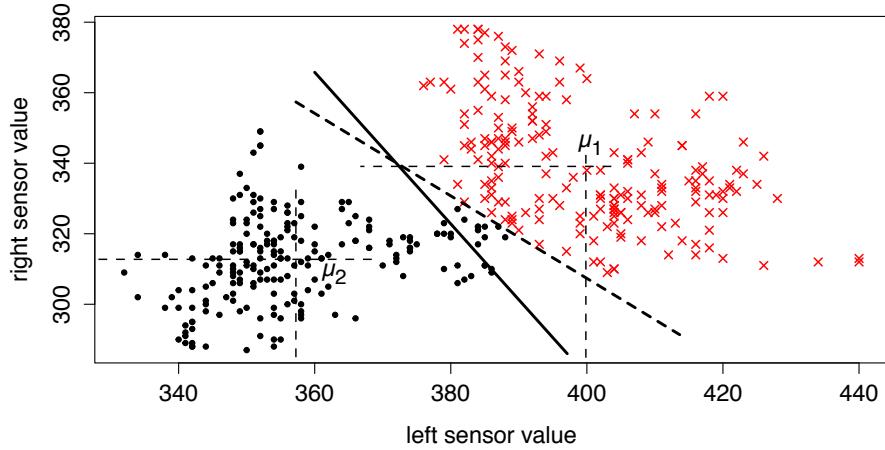


FIG. 14.9 – Moyennes pour chaque classe (lignes pointillées fines), ligne discriminante LDA (ligne continue), ligne discriminante qui distingue complètement les classes (ligne pointillée épaisse).

Plus d'échantillons ont été prélevés dans la deuxième zone (205) que dans la première (192), mais cela n'a pas d'importance pour le calcul des moyennes. Comme prévu, les moyennes $\vec{\mu}_1$ des échantillons de la zone gris clair sont légèrement plus élevées (plus de lumière réfléchie) que les moyennes $\vec{\mu}_2$ des échantillons des zones gris foncé. Cependant, le capteur gauche mesure systématiquement des valeurs plus élevées que le capteur droit. La figure 14.9 montre les données de la figure 14.8 avec des lignes pointillées fines indiquant les moyennes. Il y a deux lignes pour chaque zone : la ligne horizontale pour le capteur de droite et la ligne verticale pour le capteur de gauche.

La matrice de covariance est composée des variances des deux capteurs individuels et de leur covariance. Pour $i = 1, 2$:

$$\vec{S}_i = \begin{bmatrix} s^2(X_i^{left}) & cov(X_i^{left}, X_i^{right}) \\ cov(X_i^{right}, X_i^{left}) & s^2(X_i^{right}) \end{bmatrix}.$$

Pour les données de la Fig. 14.8, les variances des échantillons de la zone gris clair sont :

$$\begin{aligned} s^2(X_1^{left}) &= \frac{1}{191} \left((389 - 400)^2 + (390 - 400)^2 + \dots + (389 - 400)^2 \right) \approx 187 \\ s^2(X_1^{right}) &= \frac{1}{191} \left((324 - 339)^2 + (323 - 339)^2 + \dots + (373 - 339)^2 \right) \approx 286. \end{aligned}$$

L'équation B.4 de l'annexe B.4 est utilisée pour calculer la covariance. La covariance étant symétrique, la valeur ne doit être calculée qu'une seule fois.

$$\begin{aligned} \text{cov}(X_1^{\text{left}}, X_1^{\text{right}}) &= \frac{1}{191} (389 - 400)(324 - 339) + \dots + \\ &\quad (389 - 400)(373 - 339)) \\ &\approx -118. \end{aligned}$$

En rassemblant les résultats et en effectuant un calcul similaire pour X_2 , on obtient les matrices de covariance :

$$\vec{S}_1 = \begin{bmatrix} 187 & -118 \\ -118 & 286 \end{bmatrix} \quad \vec{S}_2 = \begin{bmatrix} 161 & 44 \\ 44 & 147 \end{bmatrix}.$$

L'étape suivante consiste à calculer la moyenne des matrices de covariance :

$$\mu_{\vec{S}} = \frac{1}{2} \left(\begin{bmatrix} 187 & -118 \\ -118 & 286 \end{bmatrix} + \begin{bmatrix} 161 & 44 \\ 44 & 147 \end{bmatrix} \right) = \begin{bmatrix} 174 & -38 \\ -37 & 216 \end{bmatrix},$$

et de trouver son inverse :⁸

$$\vec{S}^{-1} = \begin{bmatrix} 0.006 & 0.001 \\ 0.001 & 0.005 \end{bmatrix}.$$

Nous pouvons maintenant utiliser l'Eq. 14.4 pour calculer \vec{w} :

$$\vec{w} = \begin{bmatrix} 0.006 & 0.001 \\ 0.001 & 0.005 \end{bmatrix} \cdot \left(\begin{bmatrix} 400 \\ 339 \end{bmatrix} - \begin{bmatrix} 357 \\ 313 \end{bmatrix} \right) = \begin{bmatrix} 0.28 \\ 0.17 \end{bmatrix}.$$

Le vecteur \vec{w} donne la direction de la ligne de projection qui est perpendiculaire à la ligne discriminante. Nous utilisons maintenant l'Eq. 14.2, répétée ici :

$$w_1 x_1 + w_2 x_2 = c$$

pour calculer la constante c , en supposant que nous connaissons les coordonnées (x_1, x_2) d'un point. Mais nous avons spécifié que le point médian entre les moyennes doit être sur la droite discriminante. Ses coordonnées sont :

$$\vec{\mu} = \frac{1}{2}(\vec{\mu}_1 + \vec{\mu}_2) = \frac{1}{2} \left(\begin{bmatrix} 400 \\ 339 \end{bmatrix} + \begin{bmatrix} 357 \\ 313 \end{bmatrix} \right) \approx \begin{bmatrix} 379 \\ 326 \end{bmatrix}.$$

C'est pourquoi :

$$c = 0.28 \cdot 379 + 0.17 \cdot 326 \approx 162,$$

et l'équation de la droite discriminante est :

$$0.28x_1 + 0.17x_2 = 162.$$

La ligne discriminante est représentée par une ligne continue dans la Fig. 14.9. Etant donné un nouvel échantillon (a, b) , comparez la valeur de $0.28a + 0.17b$ à 162 : si elle est plus grande, l'échantillon est classé comme appartenant à la classe C_1 et si elle est plus petite, l'échantillon est classé comme appartenant à la classe C_2 .

8. Voir Annexe B.5.

14.2.6 Comparaison de la qualité des discriminants

Si nous comparons le discriminant linéaire trouvé ci-dessus avec les deux discriminants simples basés sur les moyennes d'un seul capteur, nous constatons une nette amélioration. En raison du chevauchement des classes dans une seule direction, le discriminant simple pour le capteur de droite ne classe correctement que 84,1

Il peut être surprenant de constater qu'il existe des lignes discriminantes capables de classer correctement la totalité des échantillons ! L'un de ces discriminants est représenté par la ligne pointillée épaisse de la figure 14.9. Pourquoi la méthode LDA n'a-t-elle pas trouvé ce discriminant ? La LDA suppose que les deux classes ont une distribution similaire (répartition des valeurs) autour de la moyenne et que le discriminant de la LDA est optimal dans cette hypothèse. Pour nos données, certains points de la deuxième classe sont éloignés de la moyenne et les distributions des deux classes sont donc légèrement différentes. Il est difficile de dire si ces échantillons sont des valeurs aberrantes, peut-être causées par un problème lors de l'impression des zones grises sur le papier. Dans ce cas, il est certainement possible qu'un échantillonnage ultérieur des deux zones aboutisse à des distributions similaires l'une à l'autre, ce qui conduirait à une classification correcte par le discriminant LDA.

14.2.7 Activités pour l'ADL

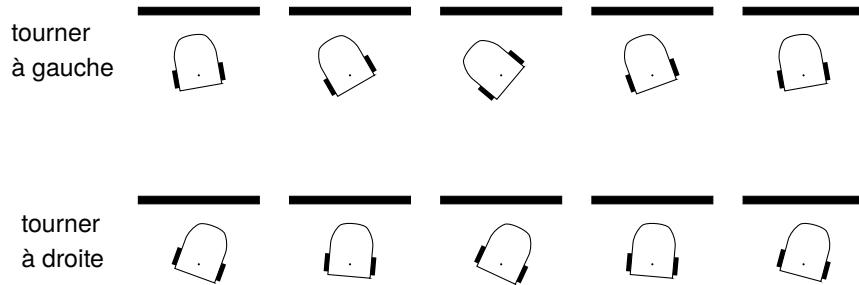
Les activités relatives à l'analyse linéaire des données sont rassemblées dans cette section.

Activity 14.2: Caméléon robotique avec LDA

- Construire un environnement comme le montre la Fig. 14.2 mais avec deux niveaux de gris très proches l'un de l'autre.
- Écrire un programme qui fait que le robot se déplace à une vitesse constante sur la zone d'une couleur et échantillonne la lumière réfléchie périodiquement. Répétez l'opération pour l'autre couleur.
- Tracer les données.
- Calculer les moyennes, les matrices de covariance et le discriminant.
- Implémenter un programme qui classe les mesures du capteur. Lorsque le robot classe une mesure, il affiche la couleur reconnue (ou donne d'autres informations si le changement de couleur n'est pas possible).

Activity 14.3: Evitement d'obstacles avec deux capteurs

- Figure 14.10 montre un robot s'approchant d'un mur. La partie supérieure du diagramme montre différentes situations dans lesquelles le

FIG. 14.10 – *Entraînement à l'évitement d'obstacles*

robot détecte le mur avec ses capteurs de droite ; il doit donc tourner à gauche pour contourner le mur. De même, dans la partie inférieure du diagramme, le robot doit tourner à droite.

- Ecrivez un programme qui enregistre les valeurs des capteurs de droite et de gauche lorsqu'un bouton est pressé. Le programme stocke également l'identité du bouton qui a été pressé ; cela représente la classe que nous recherchons pour l'évitement des obstacles.
- Entraînez le robot : Placez le robot près d'un mur et exécutez le programme. Appuyez sur le bouton gauche si le robot doit tourner à gauche ou sur le bouton droit si le robot doit tourner à droite. Répétez l'opération plusieurs fois.
- Tracez les échantillons provenant des deux capteurs sur un graphique x - y et regroupez-les par classe : tourner à droite ou à gauche pour éviter le mur. Vous devriez obtenir un graphique similaire à celui de la Fig. 14.11.
- Tracez une ligne discriminante séparant les deux classes.
- Quel est le degré de réussite de votre ligne discriminante ? Quel pourcentage des échantillons peut-elle classer avec succès ?
- Calculer le discriminant optimal en utilisant LDA. Quel est son degré de réussite ? Les hypothèses de la LDA sont-elles valables ?

Activity 14.4: Suivre un objet

- Ecrire un programme qui permet au robot de suivre un objet. Le robot avance s'il détecte l'objet devant lui ; il recule s'il est trop près de l'objet. Le robot tourne à droite si l'objet est à sa droite et tourne à gauche si l'objet est à sa gauche.
- Utiliser deux capteurs pour pouvoir visualiser les données sur un gra-

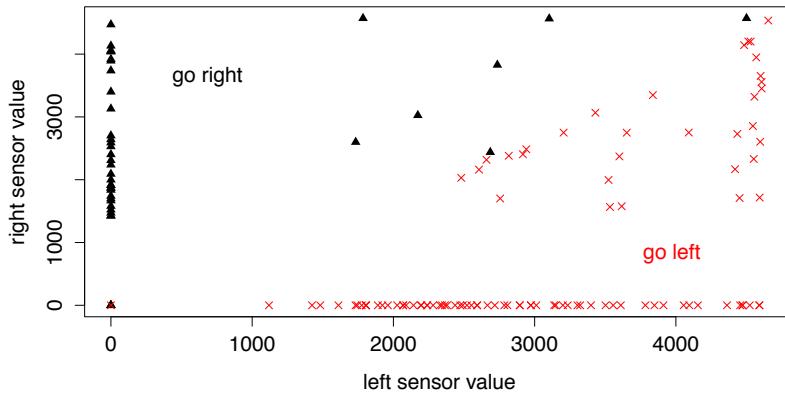


FIG. 14.11 – Données sur l'évitement d'obstacles pour la classe "aller à gauche" (x 'es rouges) et la classe "aller à droite" (triangles noirs).

- phique x - y .
- Acquérir et tracer les données comme dans Activity 14.3. Le graphique doit être similaire à celui de la Fig. 14.12.
 - Expliquez les classifications de la Fig. 14.12. Quel est le problème posé par la classification d'un échantillon comme allant vers l'avant ou vers l'arrière ? Pourquoi les échantillons qui vont vers l'avant et vers l'arrière ont-ils des valeurs différentes pour les capteurs de gauche et de droite ?
 - Proposez un algorithme pour classer les quatre situations. Pourriez-vous utiliser une combinaison de séparateurs linéaires ?

14.3 Généralisation du discriminant linéaire

Dans cette section, nous indiquons quelques moyens d'étendre et d'améliorer l'analyse discriminante linéaire.

Tout d'abord, nous pouvons avoir plus de capteurs. Les mathématiques deviennent plus complexes car avec n capteurs, les vecteurs auront n éléments et la matrice de covariance aura $n \times n$ éléments, ce qui nécessite plus de puissance de calcul et plus de mémoire. Au lieu d'une ligne discriminante, le discriminant sera un hyperplan de dimension $n - 1$. La classification avec des capteurs multiples est utilisée avec les signaux d'électroencéphalographie (EEG) du cerveau afin de contrôler un robot par la seule pensée.

14.4 a démontré une autre généralisation : la classification en plus de deux classes. Les discriminants sont utilisés pour classer chaque paire de classes. Supposons que

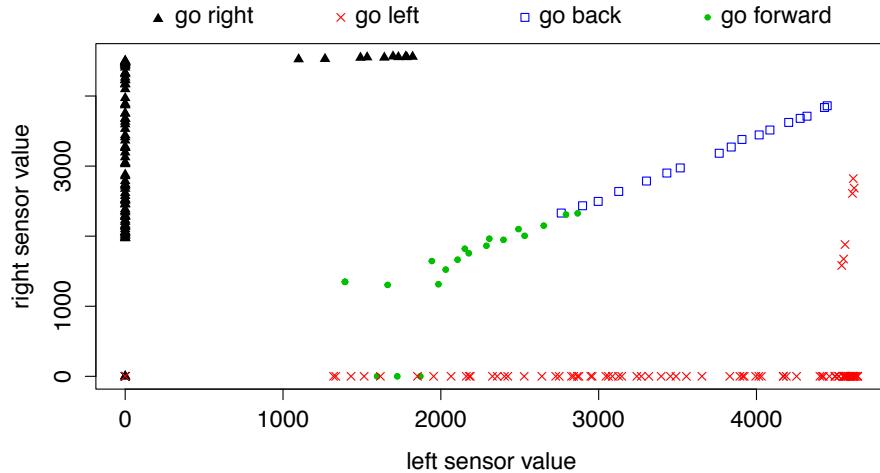


FIG. 14.12 – Données acquises lors d'une phase d'apprentissage du suivi d'un objet

vous ayez trois classes C_1 , C_2 et C_3 , et des discriminants Δ_{12} , Δ_{13} , Δ_{23} . Si un nouvel échantillon est classé dans la classe C_2 par Δ_{12} , dans la classe C_1 par Δ_{12} , et dans la classe C_2 par Δ_{23} , la classification finale sera dans la classe C_2 car davantage de discriminants affectent l'échantillon à cette classe.

Une troisième généralisation consiste à utiliser une courbe d'ordre supérieur au lieu d'une ligne droite, par exemple une fonction quadratique. Un discriminant d'ordre supérieur peut séparer des classes dont les ensembles de données ne sont pas de simples grappes d'échantillons.

14.4 Perceptrons

LDA ne peut distinguer les classes que dans l'hypothèse où les échantillons ont des distributions similaires dans les classes. Dans cette section, nous présentons une autre approche de la classification à l'aide de *perceptrons* qui sont liés aux réseaux neuronaux (Chap. 13). Nous y avons montré comment les règles d'apprentissage peuvent générer des comportements spécifiques reliant les capteurs et les moteurs ; nous montrons ici comment elles peuvent être utilisées pour classer les données en classes.

14.4.1 Détection d'une pente

Prenons l'exemple d'un robot qui explore un terrain difficile. Il est important que le robot identifie les pentes raides pour ne pas tomber, mais il est difficile de spécifier à l'avance toutes les situations dangereuses car elles dépendent de caracté-

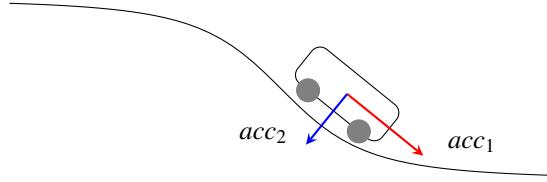


FIG. 14.13 – Un robot équipé d'accéléromètres se déplaçant sur un terrain difficile

ristiques telles que la géométrie du sol et ses propriétés (humide/sèche, sable/boue). Nous souhaitons plutôt entraîner le robot à adapter son comportement à différents environnements.

Pour simplifier le problème, supposons que le robot puisse se déplacer uniquement vers l'avant et vers l'arrière et qu'il dispose d'accéléromètres sur deux axes *relatifs au corps du robot* : l'un mesure l'accélération vers l'avant et vers l'arrière, et l'autre mesure l'accélération vers le haut et vers le bas. Un robot immobile sur une surface plane mesurera une accélération nulle vers l'avant et vers l'arrière, et une accélération vers le bas de $9,8 \text{ m/s}^2$ due à la gravité. L'accélération gravitationnelle est relativement forte par rapport à l'accélération d'un robot qui se déplace lentement, de sorte que les valeurs relatives de l'accéléromètre le long des deux axes donneront une bonne indication de son attitude.

La figure 14.13 montre un robot avançant sur une pente. Les valeurs renvoyées par les deux accéléromètres sont similaires, $acc_1 \approx acc_2$, nous pouvons donc en déduire que le robot se trouve sur une pente. Si $acc_2 \gg acc_1$, nous en déduirions que le robot se trouve sur un terrain plat car l'accéléromètre haut/bas mesure toute la force de gravité alors que l'accéléromètre avant/arrière ne mesure que la très faible accélération du robot en mouvement. La tâche consiste à faire la distinction entre une position sûre du robot et une position dans laquelle la pente commence à devenir trop raide, de sorte que le robot risque de tomber.

La figure 14.14 montre les données acquises au cours d'une session de formation pendant que le robot se déplace sur une pente. Lorsque l'opérateur de la session de formation détermine que le robot est stable (classe C_1), il lance une mesure indiquée par un \times rouge ; lorsqu'il détermine que le robot est dans une situation dangereuse (classe C_2), il lance une mesure indiquée par un triangle noir. Le défi consiste à trouver un moyen de distinguer les échantillons des deux classes.

Les lignes en pointillé de la figure indiquent les moyennes des deux ensembles de données. Il est clair qu'elles ne nous aident pas à classer les données en raison du chevauchement important entre les deux ensembles d'échantillons. En outre, la méthode LDA n'est pas appropriée car il n'y a pas de similitude dans les distributions : les échantillons lorsque le robot est stable apparaissent dans de nombreuses parties du graphique, tandis que les échantillons provenant de situations dangereuses sont

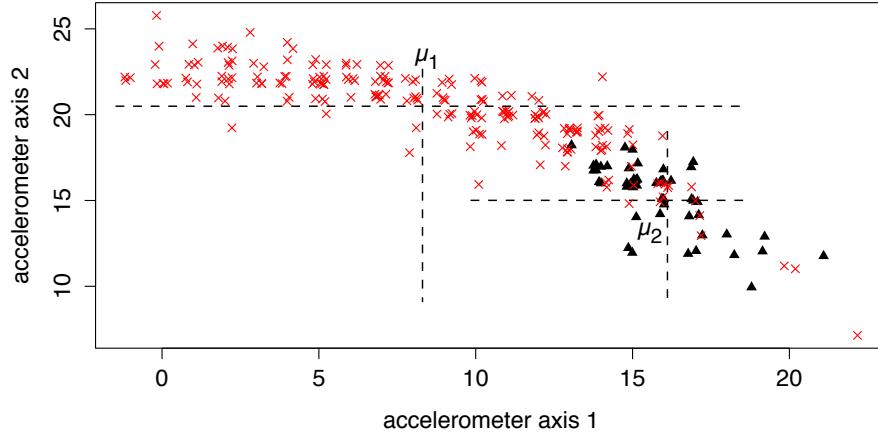


FIG. 14.14 – Détection d'une pente dangereuse à l'aide des données des accéléromètres

concentrés dans une petite zone autour de leur moyenne.

14.4.2 Classification avec perceptrons

Un *perceptron* est un neurone artificiel doté d'une structure spécifique (Fig. 14.15). Il possède une unité de sommation avec des entrées $\{x_1, \dots, x_n\}$ et chaque entrée x_i est multipliée par un facteur w_i avant la sommation. Une entrée supplémentaire x_0 a la valeur constante 1 pour établir un biais indépendant des entrées. La sortie du perceptron est obtenue en appliquant une fonction f au résultat de l'addition.

Lorsqu'il est utilisé comme classificateur, les entrées du perceptron sont les

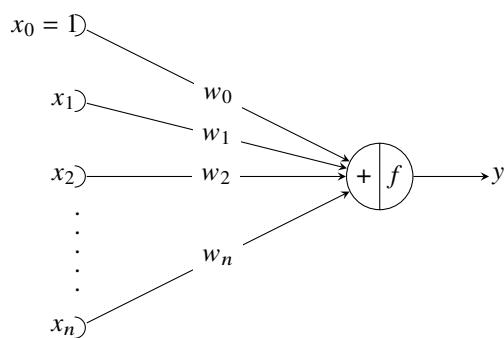


FIG. 14.15 – Un perceptron

valeurs renvoyées par les capteurs pour un échantillon à classer et la sortie sera l'une des deux valeurs indiquant la classe à laquelle l'échantillon est affecté. En général, la fonction de sortie f est simplement le signe de la somme pondérée :

$$y = \text{sign} \left(\sum_{i=0}^n w_i x_i \right) = \pm 1, \quad (14.5)$$

où une classe correspond à $+1$ et l'autre à -1 .

Les données sont normalisées de manière à ce que toutes les entrées se situent dans la même plage, généralement $-1 \leq x_i \leq +1$. Les données de la figure 14.14 peuvent être normalisées en divisant chaque valeur par 30.

Étant donné un ensemble de valeurs d'entrée $\{x_0 = 1, x_1, \dots, x_n\}$ d'un échantillon, l'objectif d'une session de formation est de trouver un ensemble de poids $\{w_0, w_1, \dots, w_n\}$ de sorte que la sortie soit la valeur ± 1 qui affecte l'échantillon à la classe correcte.

Si l'échantillon est proche de la frontière entre deux classes, la somme pondérée sera proche de zéro. Par conséquent, un perceptron est également un classificateur linéaire : pour distinguer les sorties de ± 1 , le discriminant divisant les deux classes est défini par l'ensemble des poids donnant une sortie de zéro :

$$\sum_{i=0}^n w_i x_i = 0,$$

ou

$$w_0 + w_1 x_1 + \dots + w_n x_n = 0. \quad (14.6)$$

La présentation de LDA pour un problème bidimensionnel ($n = 2$) a conduit à Eq. 14.2, qui est le même que Eq. 14.6 lorsque $c = -w_0$. La différence entre les deux approches réside dans la manière dont les poids sont obtenus : dans l'approche LDA, des statistiques sont utilisées alors que pour les perceptrons, elles résultent d'un processus d'apprentissage itératif.

14.4.3 Apprentissage par un perceptron

La recherche itérative des valeurs des poids $\{w_0, w_1, \dots, w_n\}$ commence par les fixer à une petite valeur telle que 0, 1. Pendant la phase d'apprentissage, un ensemble d'échantillons est présenté au perceptron, ainsi que la sortie attendue (la classe) pour chaque élément de l'ensemble. L'ensemble d'échantillons doit être construit de manière aléatoire et inclure des éléments de toutes les classes ; en outre, les éléments d'une seule classe doivent également être choisis de manière aléatoire. Cela permet d'éviter que l'algorithme d'apprentissage ne génère un discriminant optimal dans une situation spécifique et de garantir que le processus converge rapidement vers un discriminant optimal global, plutôt que de passer trop de temps à optimiser pour des cas spécifiques.

Algorithm 14.5: Classification par un perceptron (phase d'apprentissage)	
float array[N, n] \vec{X}	// Set of samples
float array[$n + 1$] $\vec{w} \leftarrow [0.1, 0.1, \dots]$	// Weights
float array[n] \vec{x}	// Random sample
integer c	// Class of random sample
integer y	// Output of the perceptron
1: loop until learning terminated	
2: $\vec{x} \leftarrow$ random element of \vec{X}	
3: $c \leftarrow$ class to which \vec{x} belongs	
4: $y \leftarrow$ output according to Eq. 14.5	
5: if y does not correspond to class c	
6: adjust w_i according to Eq. 14.7	
7: Output \vec{w}	

L'ajustement des poids est calculé comme suit :

$$w_i(t+1) = w_i(t) + \eta x_i y, \quad 0 \leq i \leq n. \quad (14.7)$$

Il s'agit essentiellement de la règle de Hebbian pour les ANN (Eq. 13.5.2). $w_i(t)$ et $w_i(t+1)$ sont les poids de i avant et après la correction, η définit le taux d'apprentissage, x_i est l'entrée normalisée, et y est la sortie souhaitée. Comme la fonction de signe est appliquée à la somme des entrées pondérées, y est 1 ou -1, sauf dans les rares cas où la somme est exactement égale à zéro.

Eq. 14.7 corrige les poids en ajoutant ou en soustrayant une valeur proportionnelle à l'entrée, le coefficient de proportionnalité étant le taux d'apprentissage. Une valeur faible pour le taux d'apprentissage signifie que les corrections des poids se feront par petits incrément, tandis qu'un taux d'apprentissage élevé entraînera des corrections des poids par incrément plus importants. Une fois l'apprentissage terminé, les poids sont utilisés pour classer les échantillons suivants.

Les algorithmes 14.5–14.6 sont une description formelle de la classification par perceptrons. La constante N est la taille de l'ensemble des échantillons pour la phase d'apprentissage, tandis que n est le nombre de valeurs de capteur renvoyées pour chaque échantillon.

Quand la phase d'apprentissage doit-elle se terminer ? On pourrait spécifier une valeur arbitraire, par exemple : mettre fin à la phase d'apprentissage lorsque 98% des échantillons sont classés correctement. Cependant, il n'est pas toujours possible d'atteindre ce niveau. Une meilleure méthode consiste à mettre fin à la

Algorithm 14.6: Classification par un perceptron (phase de reconnaissance)

float \vec{w} ← weights from the learning phase

float \vec{x}

integer y

- ```

1: loop
2: \vec{x} ← new sample
3: y ← output of perceptron for \vec{x}, \vec{w}
4: if $y = 1$
5: assign \vec{x} to class C_1
6: else if $y = -1$
7: assign \vec{x} to class C_2
```

phase d'apprentissage lorsque l'ampleur des corrections apportées aux poids devient faible.

#### 14.4.4 Exemple numérique

Revenons au robot qui apprend à éviter les pentes dangereuses et appliquons l'algorithme d'apprentissage aux données de la Fig. 14.14. Le perceptron a trois entrées :  $x_0$  qui vaut toujours 1,  $x_1$  pour les données de l'accéléromètre avant/arrière, et  $x_2$  pour les données de l'accéléromètre haut/bas. Les données sont normalisées en divisant chaque échantillon par 30 afin que les valeurs soient comprises entre 0 et 1. Nous précisons qu'une sortie de 1 correspond à la classe  $C_1$  (stable) et qu'une sortie de  $-1$  correspond à la classe  $C_2$  (dangereuse).

Sélectionnez un échantillon aléatoire à partir des données d'entrée, par exemple un échantillon de la classe  $C_1$  dont les valeurs de capteur sont  $x_1 = 14$  et  $x_2 = 18$ . L'entrée normalisée est  $x_1 = 14/30 = 0,47$  et  $x_2 = 18/30 = 0,6$ . La sortie du perceptron avec les poids initiaux 0,1 est :

$$\begin{aligned}
y &= \text{sign}(w_0 \times 1 + w_1 x_1 + w_2 x_2) \\
&= \text{sign}(0,1 \times 1 + 0,1 \times 0,47 + 0,1 \times 0,6) \\
&= \text{sign}(0,207) \\
&= 1.
\end{aligned}$$

Cette sortie étant correcte, il n'est pas nécessaire de corriger les poids. Choisissez maintenant un échantillon aléatoire dans la classe  $C_2$  dont les valeurs de capteur sont  $x_1 = 17$  et  $x_2 = 15$ . L'entrée normalisée est  $x_1 = 17/30 = 0,57$  et  $x_2 = 15/30 = 0,5$ .

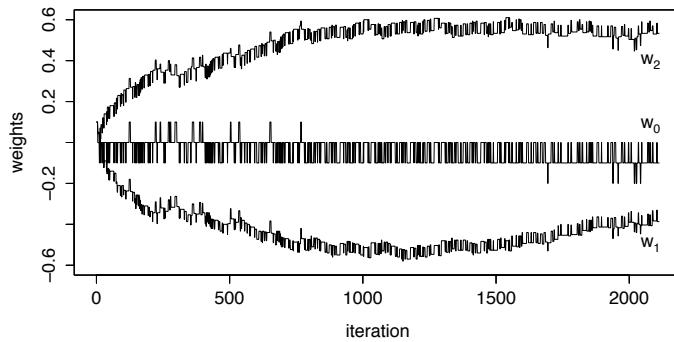


FIG. 14.16 – Évolution des poids pour l'apprentissage par un perceptron

La sortie du perceptron est la suivante :

$$\begin{aligned} y &= \text{sign}(w_0 \times 1 + w_1 x_1 + w_2 x_2) \\ &= \text{sign}(0.1 \times 1 + 0.1 \times 0.57 + 0.1 \times 0.5) \\ &= \text{sign}(0.207) \\ &= 1. \end{aligned}$$

Cette sortie n'est pas correcte : l'échantillon appartient à la classe  $C_2$  qui correspond à  $-1$ . Les poids sont maintenant ajustés en utilisant Eq. 14.7 avec un taux d'apprentissage  $\eta = 0.1$  :

$$\begin{aligned} w_0(t+1) &= w_0(t) + \eta x_0 y = 0.1 + 0.1 \times 1 \times -1 = 0 \\ w_1(t+1) &= w_1(t) + \eta x_1 y = 0.1 + 0.1 \times 0.57 \times -1 = 0.043 \\ w_2(t+1) &= w_2(t) + \eta x_2 y = 0.1 + 0.1 \times 0.5 \times -1 = 0.05. \end{aligned}$$

Ce seront les nouveaux poids pour l'itération suivante. Si nous continuons pendant 2000 itérations, les poids évoluent comme le montre la Fig. 14.16. À la fin du processus d'apprentissage, les poids sont les suivants :

$$w_0 = -0.1, \quad w_1 = -0.39, \quad w_2 = 0.53.$$

Ces poids peuvent maintenant être utilisés par la phase de reconnaissance de l'algorithme de classification 14.6. La ligne discriminante construite par le perceptron (Eq. 14.6) est :

$$-0.1 - 0.39x_1 + 0.53x_2 = 0.$$

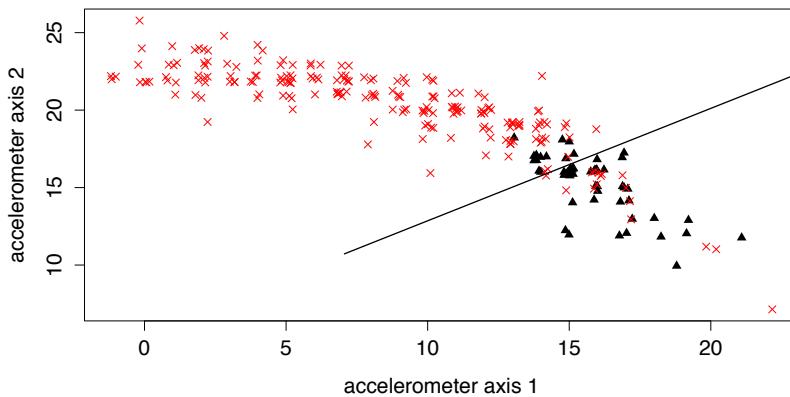


FIG. 14.17 – Ligne discriminante calculée à partir des poids du perceptron

Les coordonnées de cette ligne sont les valeurs normalisées, mais elles peuvent être retransformées en valeurs brutes obtenues à partir des accéléromètres. La ligne est représentée sur la figure 14.17 et, compte tenu du chevauchement important des classes, elle permet de les distinguer assez bien.

#### 14.4.5 Ajustage des paramètres du perceptron

Les performances d'un perceptron sont déterminées par le nombre d'itérations et le taux d'apprentissage. La figure 14.16 montre qu'il y a une forte variation des poids au début, mais que les poids se stabilisent au fur et à mesure que le nombre d'itérations augmente. Il est donc relativement simple de surveiller les poids et de mettre fin au calcul lorsque les poids se stabilisent.

Cette évolution des poids dépend fortement du taux d'apprentissage. L'augmentation du taux d'apprentissage accélère la variation au début, mais de fortes corrections ne sont pas bénéfiques lorsque les poids commencent à se stabiliser. La figure 14.16 montre clairement que même à la fin de l'exécution, les variations des poids sont importantes et oscillent autour de la valeur optimale. Cela suggère que nous réduisons le taux d'apprentissage pour réduire les oscillations, mais cela ralentira la convergence vers les poids optimaux au début de la phase d'apprentissage.

La solution consiste à utiliser un taux d'apprentissage variable qui n'est pas constant. Il doit être important au départ pour favoriser une convergence rapide vers les valeurs optimales, puis devenir plus petit pour réduire les oscillations. Par exemple, nous pouvons commencer avec un taux d'apprentissage de 0,1 et le diminuer continuellement en utilisant l'équation :

$$\eta(t+1) = \eta(t) \times 0.997.$$

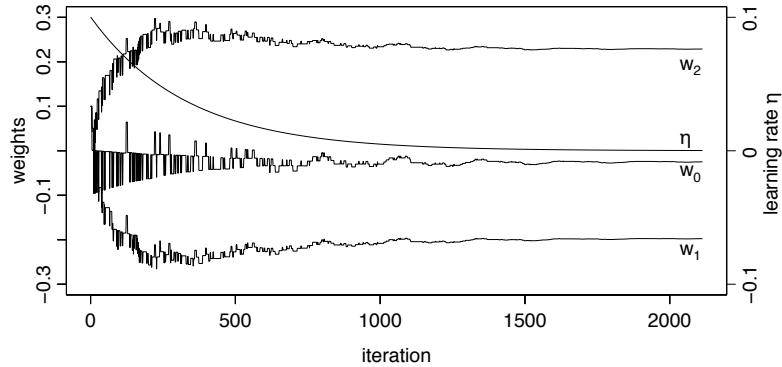


FIG. 14.18 – Evolution des poids pour l’apprentissage par un perceptron avec un taux d’apprentissage variable

La figure 14.18 montre l’évolution des poids lorsque ce taux d’apprentissage variable est utilisé. La diminution exponentielle de  $\eta$  est également représentée sur la figure. La comparaison des figures 14.16 et 14.18 montre clairement la supériorité du taux d’apprentissage variable et cette amélioration est obtenue avec très peu de calculs supplémentaires.

#### Activity 14.5: Apprentissage par un perceptron

- Prenez un ensemble de mesures des accéléromètres de votre robot sur différentes pentes et tracez les données. Pour chaque échantillon, vous devrez décider si le robot risque de tomber de la pente.
- Classifiez les données à l'aide d'un perceptron. Quelle ligne discriminante trouvez-vous ?
- Utilisez un perceptron pour classer les zones grises en utilisant les données de l'activité 14.2. Quel discriminant trouvez-vous ? Comparez le discriminant trouvé par le perceptron au discriminant trouvé par LDA.

## 14.5 Résumé

Les échantillons de deux classes peuvent être distingués en utilisant uniquement leurs moyennes ou en utilisant à la fois les moyennes et les variances. L’analyse discriminante linéaire est une méthode de classification basée sur le calcul des covariances entre les échantillons des classes. L’analyse discriminante linéaire ne

donne de bons résultats que lorsque les distributions des échantillons des classes sont similaires. Lorsque cette hypothèse ne se vérifie pas, des perceptrons peuvent être utilisés. Pour des performances optimales, le taux d'apprentissage d'un perceptron doit être ajusté, si possible dynamiquement pendant la phase d'apprentissage.

#### 14.6 Lecture complémentaire

Une présentation mathématique détaillée de l'analyse discriminante linéaire peut être trouvée dans Izenman [25, Chapitre 8]. Les manuels sur les techniques d'apprentissage automatique sont [19, 29].



## Chapitre 15

### Robotique en essaim

Les usines utilisent plusieurs robots pour atteindre des objectifs tels que la peinture et le soudage d'une voiture (1.3). L'utilisation de plusieurs robots permet de réduire le temps de fabrication en effectuant simultanément différentes tâches telles que le soudage de pièces des deux côtés d'une voiture. Ces tâches sont généralement conçues pour être indépendantes, sans collaboration étroite entre les robots. Plus tard, les robots, en particulier les robots mobiles, ont été conçus pour collaborer les uns avec les autres afin d'effectuer plusieurs actions simultanément à différents endroits.

Voici quelques exemples de tâches nécessitant la collaboration de plusieurs robots :

- Manipulation de grands éléments structurels dans les bâtiments, ainsi que dans des environnements difficiles d'accès pour les humains, comme dans l'espace ou sous l'eau.
- Exécuter une tâche grâce à la collaboration de différents types de robots : lors d'une catastrophe de grande ampleur, un drone volant peut localiser les zones où des victimes sont susceptibles d'être trouvées, tandis qu'un robot à chenilles au sol fouille ces zones, en se concentrant sur les endroits qui peuvent être cachés de l'observation aérienne par des arbres ou des décombres.
- Effectuer des mesures simultanées à différents endroits : mesurer les perturbations sonores dans différentes parties d'un bâtiment, ou surveiller la pollution après un accident industriel.

Le point commun de ces situations est que plusieurs robots participent à l'exécution d'une tâche et qu'ils doivent se coordonner parce qu'ils agissent sur le même objet physique, même s'ils ne sont pas à côté l'un de l'autre dans l'environnement.

La collaboration entre robots peut également être utilisée pour accélérer l'exécution d'une tâche en demandant à plusieurs robots de l'effectuer en parallèle. Prenons l'exemple de la mesure de la pollution sur une vaste zone : un seul robot peut parcourir toute la zone (comme un aspirateur robotisé dans un appartement), mais la tâche sera accomplie beaucoup plus rapidement si plusieurs robots se répartissent les zones à couvrir.

### 15.1 Approches de la mise en œuvre de la collaboration entre robots

Il existe deux approches principales de la conception de systèmes composés de plusieurs robots. La première est un *système centralisé*, dans lequel un composant central (l'un des robots ou un ordinateur externe) coordonne tous les robots et leurs tâches. L'avantage d'un système centralisé est qu'il est relativement simple à mettre en œuvre. Son principal inconvénient est qu'il est difficile à étendre, car l'ajout de robots ajoute une charge de traitement à la station centrale où toute l'intelligence est concentrée. Dans un système centralisé, les robots eux-mêmes peuvent être "stupides", mais la plupart des robots ont une puissance de calcul importante qui n'est pas bien utilisée dans cette architecture. Un autre inconvénient majeur d'un système centralisé est que le composant central est un point de défaillance unique. S'il cesse de fonctionner, c'est tout le système qui tombe en panne. Dans les environnements critiques, il est unacceptable d'utiliser un système qui n'est pas robuste en cas de défaillance d'un seul composant.

Si nous nous tournons vers le monde animal, nous constatons que de nombreuses activités sont *distribuées*, c'est-à-dire que des individus indépendants travaillent ensemble pour atteindre des objectifs communs à l'ensemble de la population. Les fourmis optimisent leur chemin vers les sources de nourriture non pas en dépendant d'une fourmi qui en envoie d'autres à la recherche et qui traite ensuite les informations obtenues, mais par un effort distribué de l'ensemble de la colonie de fourmis. Chaque fourmi marque le sol avec des phéromones qui sont détectées par les autres fourmis. Si quelques fourmis sont dévorées par un prédateur, le reste de la colonie survit, tout comme les connaissances incorporées dans les emplacements des phéromones. L'efficacité et la robustesse de cette approche ont été démontrées dans les algorithmes du chapitre 7.

*La robotique en essaim* est une approche distribuée de la robotique qui tente de coordonner le comportement en copiant des mécanismes inspirés du comportement des animaux sociaux. Ces mécanismes, souvent locaux et simples, permettent à un groupe d'atteindre des performances globales qui ne pourraient pas être atteintes par un individu seul. Les systèmes distribués présentent les avantages suivants :

- Ils sont robustes. La perte d'un robot sur dix ne réduit les performances du système que d'environ 10%, au lieu d'entraîner la défaillance de l'ensemble du système.
- Ils sont flexibles et évolutifs. Le nombre de robots peut être adapté à la tâche. S'il y a dix robots dans le système mais que cinq suffisent à accomplir la tâche, les cinq autres peuvent être affectés à d'autres tâches, tandis que si dix robots ne peuvent pas accomplir la tâche efficacement, dix autres peuvent facilement être ajoutés.

Ces avantages s'accompagnent d'un effort de conception et de mise en œuvre de la coordination entre les robots. En robotique en essaim, comme dans la nature, il

existe des mécanismes de coordination relativement simples qui rendent les systèmes distribués réalisables.

Ce chapitre présente deux approches de la coordination en robotique en essaim :

- Coordination basée sur l'information (Sec.15.2), où l'interaction prend la forme de communications entre les robots. Il peut s'agir d'une communication directe par la transmission explicite de messages électroniques ou d'une communication indirecte par l'insertion de messages dans l'environnement.
- Coordination physique (Sec. 15.3), où les robots individuels interagissent au niveau mécanique, soit directement en exerçant des forces les uns sur les autres, soit indirectement en manipulant un objet commun.

## 15.2 Coordination par échange local d'informations

Les communications peuvent être globales ou locales. Supposons que vous recevez un appel de vos amis et qu'ils vous informent : "Nous voyons un marchand de glaces sur la gauche. Cette information est inutile à moins qu'ils ne vous indiquent leur position actuelle. En revanche, si vous marchez côté à côté avec eux et que l'un d'eux vous dit : "Je vois un magasin de glaces sur la gauche", cette information *locale* vous permet de connaître immédiatement l'emplacement approximatif du magasin et de le localiser facilement visuellement.

Inspirée par la nature, la robotique en essaim utilise des communications locales au sein d'une architecture distribuée. Quelques zèbres situés à la périphérie d'un troupeau recherchent des prédateurs et signalent les autres par des sons ou des mouvements. Le troupeau est une stratégie de survie efficace pour les animaux, car les communications locales permettent à un grand nombre d'animaux de fuir immédiatement lorsqu'un prédateur est détecté par un petit nombre d'observateurs vigilants.

### 15.2.1 Communications directes

*Direct* L'échange local d'informations est réalisé lorsqu'un ami vous parle. Les animaux ne parlent pas, mais ils utilisent le son ainsi que le mouvement et le contact physique pour réaliser un échange local direct d'informations. Les robots mettent en œuvre des communications locales directes par voie électronique (WiFi ou Bluetooth), ou en transmettant et en recevant de la lumière ou des sons. Ils peuvent également utiliser une caméra pour détecter des changements chez un autre robot, par exemple en allumant une lumière.

Les communications locales peuvent être soit *directionnelles*, soit *non-directionnelles*. Les communications radio telles que Bluetooth sont locales (quelques mètres seulement) et le robot récepteur ne cherche généralement pas à déterminer la direction du robot émetteur. Les communications directionnelles locales peuvent être mises en

œuvre en utilisant une source lumineuse comme émetteur et un détecteur à ouverture étroite ou une caméra comme récepteur.

### 15.2.2 Communications indirectes

*Les communications locales indirectes* font référence aux communications par l’intermédiaire d’un support qui peut stocker un message transmis en vue d’un accès ultérieur. L’exemple le plus familier est le courrier, qu’il s’agisse de courrier électronique ou de courrier ordinaire, où l’émetteur compose le message et l’envoie, mais où le message reste sur le serveur ou au bureau de poste jusqu’à ce qu’il soit remis au destinataire qui, à son tour, peut ne pas y avoir accès immédiatement. Les communications indirectes chez les animaux sont appelées *stigmergy*; les animaux laissent des messages en déposant des substances chimiques que d’autres animaux peuvent percevoir. Nous avons mentionné l’utilisation des phéromones par les fourmis; un autre exemple est l’utilisation de l’urine par un chien pour marquer son territoire.

Les messages chimiques sont difficiles à mettre en œuvre dans les robots, mais les robots peuvent laisser des marques optiques sur le sol, comme nous l’avons fait dans l’algorithme simulant une colonie de fourmis à la recherche d’une source de nourriture (Sect.7.3). Dans ce cas, un seul robot a été utilisé, mais les algorithmes pourraient être facilement mis en œuvre avec plusieurs robots, car c’est le marquage qui est important, et non l’identité du robot qui a créé le marquage.

Les communications indirectes peuvent également être mises en œuvre en plaçant ou en manipulant des objets dans l’environnement. Les aspirateurs robotiques utilisent des balises qui peuvent être placées à l’entrée des pièces que le robot doit éviter. Il est possible de concevoir des balises qui enregistrent le moment où une pièce a déjà été nettoyée et cette information est communiquée (indirectement) à d’autres aspirateurs.

*Karel le robot* est un environnement utilisé pour enseigner la programmation. Les commandes déplacent un robot virtuel autour d’une grille sur un écran d’ordinateur et le robot peut déposer et détecter des "beepers" placés sur des carrés de la grille (Fig. 15.1).<sup>1</sup>

Les communications indirectes combinées à la manipulation peuvent générer des modèles intéressants. La figure 15.2 montre un environnement rempli de petits objets et de cinq robots mobiles équipés de pinces. Les robots suivent un ensemble de règles simples :

- Si le robot trouve un objet isolé, il le ramasse ;
- Si le robot trouve un objet isolé mais en tient déjà un, il pose le nouvel objet à côté de celui qui a été trouvé ;
- Le robot évite les murs et les groupes de plusieurs objets.

1. L’image est tirée de la mise en œuvre de Karel le Robot en Scratch par le premier auteur (<https://scratch.mit.edu/studios/520857>).

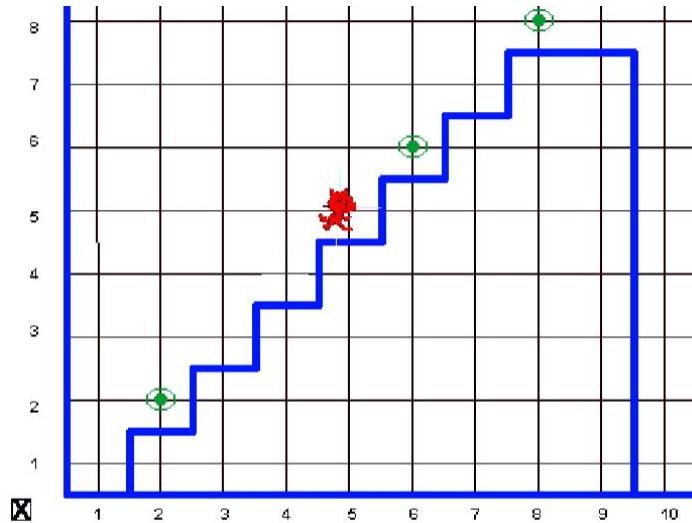


FIG. 15.1 – *Karel the Robot implemented in Scratch; the green dots are the beepers*

Il semble que ces règles amènent les robots à placer tous les objets par groupes de deux, mais ce n'est pas le cas, comme le montre la Fig. 15.3. La raison en est qu'un groupe d'objets *vus de côté* peut ressembler à un objet isolé, de sorte qu'un objet supplémentaire est placé dans le groupe. Il en résulte que les grands groupes d'objets sont assemblés uniquement par communication indirecte.

### 15.2.3 L'algorithme BeeClust

L'algorithme BeeClust est un algorithme d'essaim inspiré du comportement des abeilles. Il utilise une architecture distribuée et des communications locales pour générer un résultat global. L'algorithme BeeClust est basé sur la façon dont les très jeunes abeilles se regroupent autour des endroits où la température est optimale dans l'obscurité de leur nid. Elles mesurent les températures locales et détectent les collisions avec d'autres abeilles. L'algorithme peut être utilisé par un essaim de



FIG. 15.2 – *Robots à pince dans un environnement rempli de petits objets*

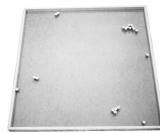


FIG. 15.3 – *Les objets ont été rassemblés en groupes*

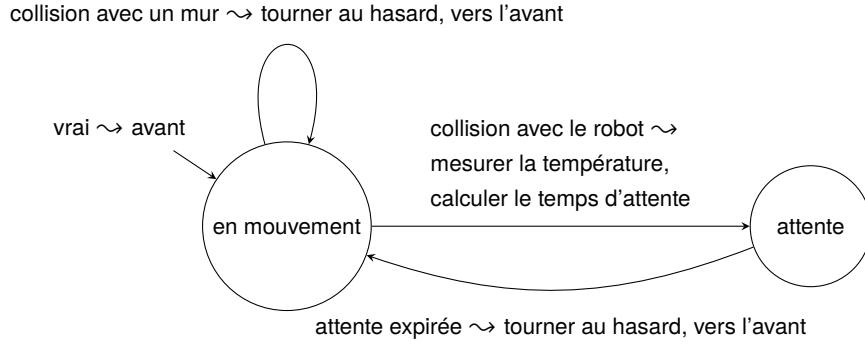


FIG. 15.4 – L’algorithme BeeClust

robots pour localiser la pollution ; au lieu de mesurer la température, chaque robot mesure une certaine quantité physique qui indique les niveaux de pollution. Avec le temps, les robots se rassembleront en groupes aux endroits où les niveaux de pollution sont élevés.

La figure 15.4 montre une machine à états qui met en œuvre l’algorithme. Le robot se déplace au hasard jusqu’à ce qu’il heurte un autre robot. À ce moment-là, il mesure la température à l’endroit de la collision. Il attend à cet endroit pendant une période de temps proportionnelle à la température qu’il a trouvée, puis se remet à se déplacer de manière aléatoire. Lorsque le robot se déplace, il évite les obstacles tels que les murs. L’algorithme utilise ce qui est peut-être la forme la plus simple de communication : la détection d’une collision avec un autre robot. La nature localisée des communications est essentielle au bon fonctionnement de l’algorithme.

Au départ, les robots se heurtent à des endroits aléatoires, mais ceux qui se trouvent dans des endroits où la température est plus élevée y restent plus longtemps, ce qui provoque des collisions supplémentaires. À long terme, la plupart des robots formeront un groupe dans la zone où les températures sont les plus élevées. Ils se heurtent fréquemment les uns aux autres, car le fait d’être dans une foule augmente le nombre de collisions. Bien entendu, ce mécanisme ne peut fonctionner qu’avec un grand nombre de robots qui génèrent de nombreuses collisions, ce qui donne lieu à de nombreuses mesures et à la formation de grappes.

#### 15.2.4 L’implémentation ASSISIbf de BeeClust

Dans le cadre du projet ASSISIbf, des chercheurs de l’université de Graz ont utilisé des robots Thymio pour mettre en œuvre l’algorithme BeeClust dans le cadre

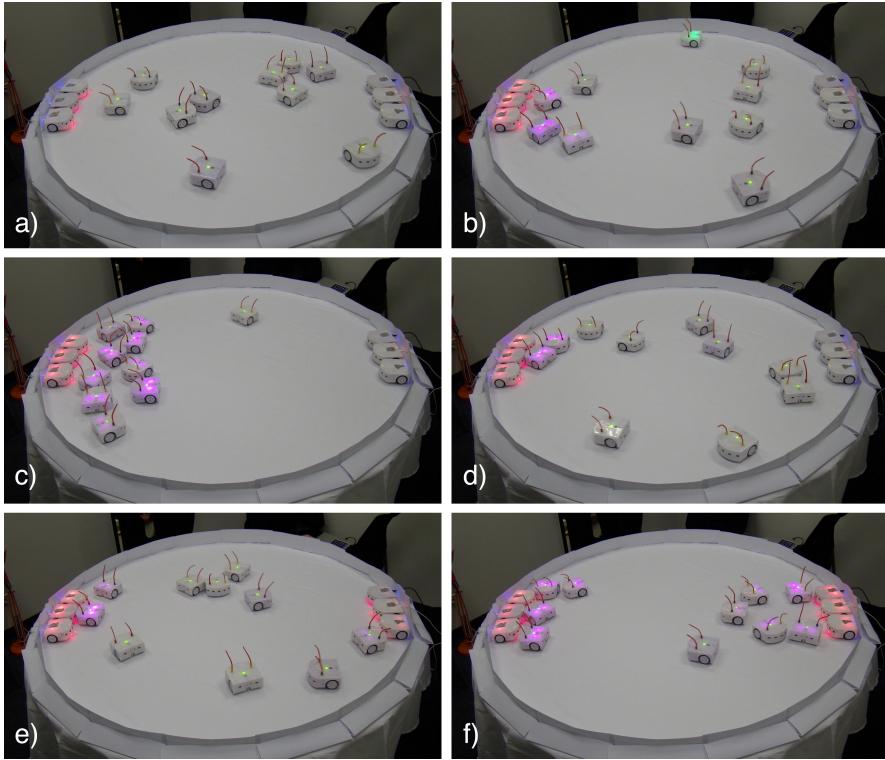


FIG. 15.5 – *Mise en œuvre de BeeClust. Les photographies ont été prises aux heures suivantes (minutes :secondes) à partir du début de l’expérience : a) 1 :40, b) 2 :30, c) 9 :40, d) 12 :40, e) 13 :20, f) 21 :10.*

d'une recherche sur le comportement des abeilles dans un nid. L'algorithme simule un groupe de jeunes abeilles dans une arène circulaire froide, où deux sources de chaleur virtuelles sont placées à droite et à gauche de l'arène. Au départ, une seule de ces sources de chaleur est active. Les sources de chaleur sont simulées par deux groupes de trois robots situés sur les bords de l'arène (Fig. 15.5(a)).

Les trois robots émetteurs de chaleur sur le côté gauche transmettent leur température. Les robots-abeilles qui se trouvent à proximité détectent ce signal et s'arrêtent pendant un temps proportionnel à la température. Pendant leur séjour dans cette zone, ils transmettent également un signal de température. En outre, si les robots émetteurs de chaleur détectent des robots proches, ils augmentent leur chaleur et transmettent la nouvelle température. La figure 15.5 montre l'évolution du comportement des robots : (a) l'état initial lorsque les robots commencent à se déplacer et que seule la source de température de gauche est allumée ; (b) les robots commencent à se

regrouper autour de la source de gauche ; (c) le niveau de regroupement le plus élevé se produit. Les robots-abeilles peuvent quitter la grappe, explorer l'environnement et revenir à la grappe, comme le montre la figure (d). Ce phénomène est dû à la nature aléatoire de l'algorithme et est essentiel pour éviter que le système ne soit piégé dans des minima locaux. La figure 15.5 (e) montre le moment où la bonne source est également activée. Après environ 21 minutes, les robots forment deux groupes plus petits (f).

#### Activity 15.1: algorithme BeeClust

- Mettre en œuvre l'algorithme BeeClust pour amener un groupe de robots à se regrouper à l'endroit le plus lumineux d'une pièce.
- Utiliser trois capteurs : un capteur de lumière pour mesurer la lumière ambiante, un capteur pour détecter les autres robots et un capteur pour détecter les limites de l'arène.
- Solution 1 : Utiliser des capteurs de proximité pour détecter les autres robots et des capteurs au sol pour détecter une ligne définissant les limites de l'arène.
- Solution 2 : Définissez les limites de l'arène à l'aide d'un mur et utilisez les communications entre robots pour faire la distinction entre les robots et le mur. Pour éviter toute confusion, n'utilisez pas le même capteur pour détecter les autres robots et le mur.

### 15.3 Robotique en essaim basée sur les interactions physiques

Dans la Sect. 7.2, nous avons étudié un exemple typique de comportement efficace en essaim : une colonie de fourmis trouvant un chemin entre leur nid et une source de nourriture. Cet exemple utilisait des communications indirectes sous la forme de phéromones déposées sur le sol. Dans cette section, nous examinons une autre forme de comportement en essaim qui est médiée par des interactions physiques. Nous commençons par des fourmis collaborant à l'extraction d'un bâton du sol et à une version robotisée de cette tâche. Nous examinerons ensuite la manière dont les forces exercées par plusieurs robots peuvent être combinées, comme le montre un algorithme simple mais astucieux appelé "poussée collective basée sur l'occlusion".

#### 15.3.1 Collaboration à une tâche physique

La figure 15.6 montre deux fourmis extrayant un bâton du sol pour l'utiliser dans la construction d'un nid. Le bâton est enfoncé si profondément qu'une fourmi ne peut pas l'extraire seule. Nous voulons concevoir un système robotique pour accomplir cette tâche (Fig. 15.7). Chaque robot cherche au hasard jusqu'à ce qu'il trouve un

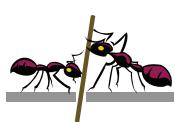


FIG. 15.6 – *Ants tirant un bâton du sol*

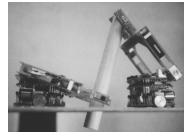


FIG. 15.7 – *Les robots tirent un bâton du sol*

bâton. Il tire alors sur le bâton aussi fort que possible. S'il réussit à extraire le bâton, il le ramène dans un nid; sinon, comme il n'a que partiellement extrait le bâton, il attend de sentir qu'un autre robot tire plus fort et relâche le bâton. Si aucun robot ne lui vient en aide pendant un certain temps, le robot relâche sa prise et retourne à une recherche aléatoire. Ainsi, s'il y a plus de bâtons que de robots, le système ne se bloquera pas, chaque robot essayant d'extraire un bâton et attendant indéfiniment.

La machine à états finis de cet algorithme est présentée dans la Fig. 15.8. Bien que ce comportement soit simple et local, lorsqu'il est appliqué par deux robots, il permet d'extraire le bâton du sol. Le robot de droite sur la Fig. 15.7 tire une partie du bâton aussi loin que possible du sol en utilisant le mouvement maximal de son bras. Lorsqu'il détecte qu'un autre robot a trouvé le même bâton et commence à tirer, le premier robot relâche le bâton pour permettre au second robot de l'extraire. En combinant les capacités physiques de deux robots avec une simple spécification de comportement, nous obtenons un résultat qu'aucun des robots ne pourrait atteindre seul.

### 15.3.2 Combiner les forces de plusieurs robots

15.9 montre un robot à entraînement différentiel se déplaçant à reculons (de gauche à droite). Il exerce une force  $F_r$  qui peut être utilisée pour tirer un objet. La figure 15.10 montre deux robots connectés ensemble de sorte qu'ils exercent une force  $F_{subtotal}$  lorsqu'ils se déplacent de gauche à droite.

Quelle est la relation entre  $F_r$  et  $F_{total}$ ? Il y a trois possibilités :

- $F_{total} < 2F_r$  : Les robots connectés perdent en efficacité car la force qu'ils exercent est inférieure à celle exercée par les deux robots tirant séparément.
- $F_{total} = 2F_r$  : Les robots connectés ont la même efficacité que deux robots séparés.
- $F_{total} = 2F_r$  : Les robots connectés atteignent la même efficacité que deux robots séparés.

Les robots peuvent atteindre  $F_{total} > 2F_r$ , où la force totale est supérieure à la somme des forces exercées par les robots individuels, car dans certaines configurations mécaniques, les robots connectés sont plus stables car leur centre de gravité est mieux placé.

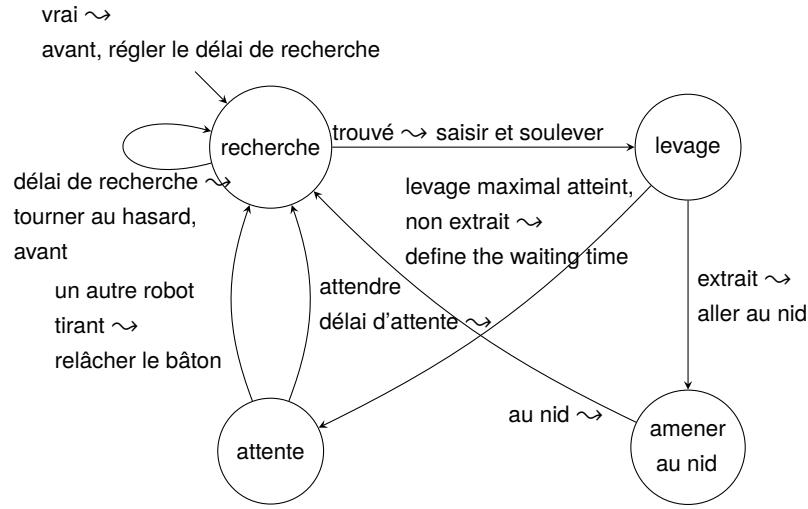
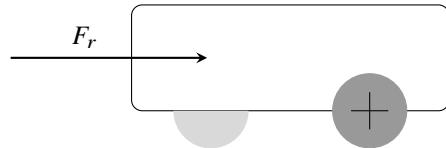
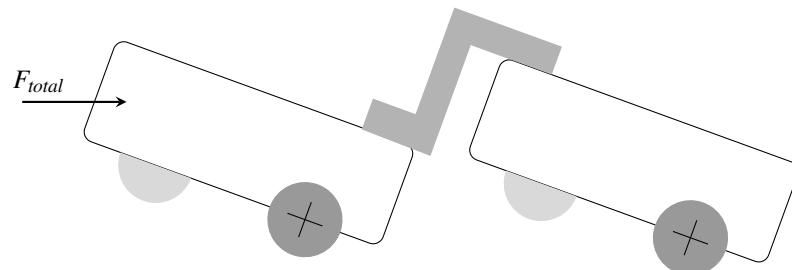


FIG. 15.8 – Algorithme pour le tirage distribué de bâtons

FIG. 15.9 – Un robot qui tire avec une force donnée  $F_r$ FIG. 15.10 – Deux robots connectés se tirent avec une force donnée  $F_{total}$

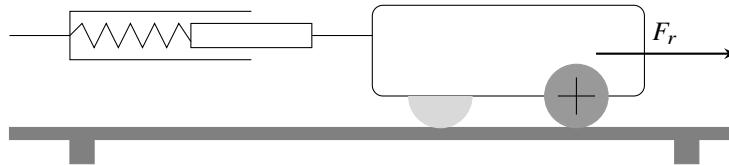


FIG. 15.11 – Mesure de la force à l'aide d'un dynamomètre

**Activity 15.2: Force de traction de plusieurs robots.**

- Connectez deux robots et vérifiez s'ils exercent une force inférieure, égale ou supérieure au double de celle exercée par un seul robot. Vous pouvez relier les robots à l'aide d'une connexion rigide comme sur la Fig. 15.10 ou à l'aide d'une connexion flexible en utilisant une ficelle.
- Mesurer  $F_r$ , la force exercée par un seul robot, puis mesurer  $F_{subtotal}$  la force exercée par les robots connectés.
- Un *dynamomètre* (Fig 15.11) est le meilleur instrument pour mesurer les forces. Si vous n'en disposez pas, vous pouvez utiliser un câble, une poulie et des poids (ou une balance) comme indiqué sur la Fig. 15.12.
- Changez l'orientation des robots : tirez vers l'avant au lieu de tirer vers l'arrière. Cela change-t-il le résultat ?
- Faire des expériences sur différentes surfaces (sol dur, moquette, papier) et déterminer l'effet de la surface sur la force résultante.

**15.3.3 poussée collective basée sur l'occlusion**

Considérons un autre exemple de combinaison de la force de plusieurs robots. Au lieu d'un lien physique comme dans la Fig. 15.10, nous utilisons de nombreux robots simples pour pousser un objet, imitant un groupe de fourmis. Une fois encore,

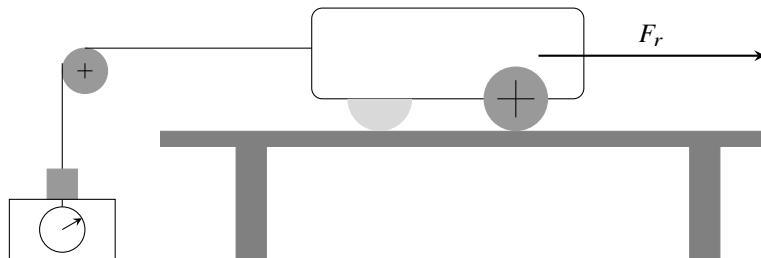


FIG. 15.12 – Mesure de la force à l'aide d'un poids et d'une balance

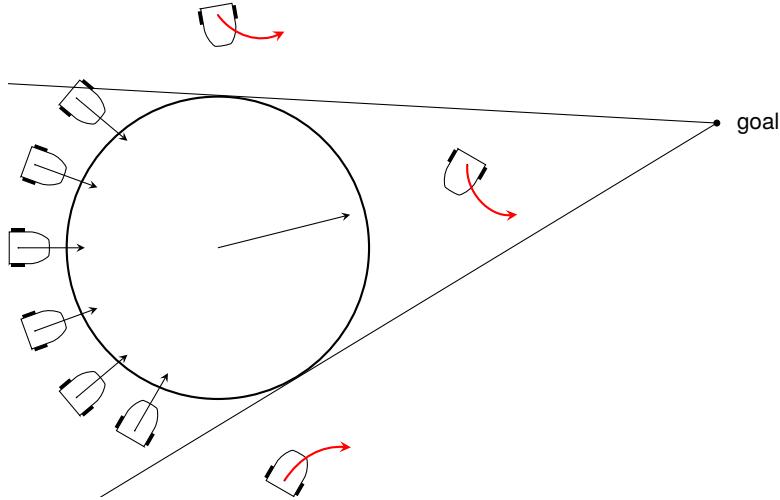


FIG. 15.13 – Coordination basée sur l’occlusion : flèches noires droites pour les robots qui poussent l’objet et flèches rouges courbes pour les robots qui recherchent une position occluse.

les avantages d'un système distribué sont la flexibilité - l'engagement des seules ressources nécessaires à la tâche - et la robustesse - si un robot tombe en panne, l'objet peut se déplacer un peu plus lentement, mais il n'y a pas d'échec total de la tâche. Ces avantages s'accompagnent de ressources supplémentaires et de la complexité de la mise en œuvre de la coordination entre les robots.

La figure 15.13 montre un groupe de petits robots poussant un grand objet représenté par un cercle vers un objectif. Une approche consisterait à déterminer la direction de l'objectif, à partager cette information entre tous les robots et à demander à chaque robot de calculer la direction dans laquelle il doit pousser. Ce n'est pas aussi simple qu'il n'y paraît, car un simple robot peut ne pas être en mesure de déterminer sa position et son cap absolu.

Une approche de robotique en essaim est appelée *poussée basée sur l’occlusion*, qui suppose que les robots n'ont qu'une connaissance locale, et non globale, de ce que font les autres robots. Les robots sont en mesure de déterminer s'ils peuvent ou non détecter l'objectif. Par exemple, une lumière vive peut être installée sur l'objectif et les robots équipés d'un capteur de lumière.

Figure 15.14 shows the finite state machine for this algorithm. The robots search for the object; when they find it they place themselves perpendicular to the surface and push (straight black arrows). This could be implemented using a touch sensor or a proximity sensor. The robots continue to push as long as they *do not* detect the goal. If they do detect the goal, they stop pushing, move away (curved red

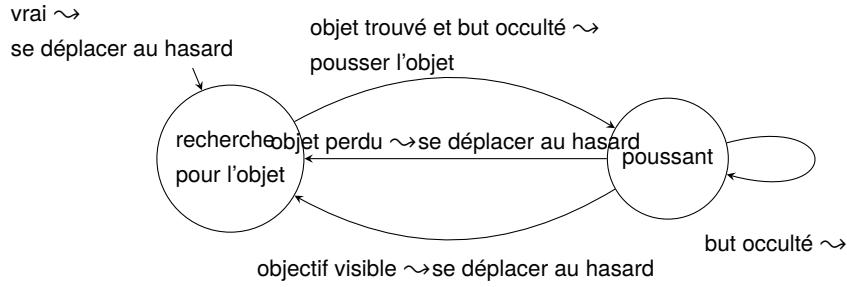


FIG. 15.14 – Algorithme pour la coordination basée sur l’occlusion

arrows) and commence a new search for an occluded position where they resume pushing. The result is that the vector sum of the forces exerted by the robots is in a direction that moves the object toward the goal. The occlusion algorithm leads to the task being performed without a central control unit and even without inter-robot communications.

**Activity 15.3: Force totale**

- Considérons la configuration montrée sur la Fig. 15.15. Le robot 1 pousse un objet à un angle de  $45^\circ$  avec la force  $f_1$ , le robot 2 pousse horizontalement avec la force  $f_2$  et le robot 3 pousse verticalement avec la force  $f_3$ . Montrer que  $f_{total}$ , la force totale sur l’objet, a une magnitude :

$$\sqrt{\left(f_2 + \frac{f_1}{\sqrt{2}}\right)^2 + \left(f_3 - \frac{f_1}{\sqrt{2}}\right)^2}.$$

dans la direction :

$$\alpha = \tan^{-1} \frac{2f_3 - \sqrt{2}f_1}{2f_2 + \sqrt{2}f_1}.$$

- Calculer l’amplitude et la direction de  $f_{total}$  pour différentes valeurs des forces individuelles. Si  $f_1 = f_2 = f_3 = 1$ , la magnitude est  $\sqrt{3}$  et la direction est  $9.7^\circ$ . Si  $f_1 = f_2 = 1$ , quelle doit être la valeur de  $f_3$  pour que  $\alpha = 45^\circ$  ?
- Utilisez trois robots pour mettre en oeuvre cette configuration et vérifiez que l’objet se déplace dans la direction calculée.

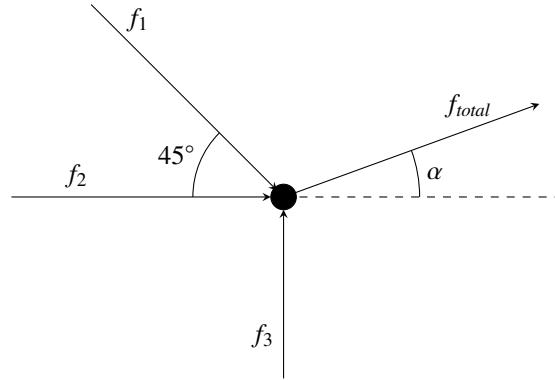


FIG. 15.15 – La force totale de trois robots

**Activity 15.4: poussée basée sur l'occlusion**

- Placez trois robots autour d'un objet et placez un objectif à une certaine distance du robot. Mettez en place un mécanisme permettant aux robots de déterminer la direction vers le but, par exemple en attachant une lumière au but ou en plaçant le but au point le plus bas d'une surface inclinée.
- Mettre en œuvre un mécanisme permettant aux robots de faire la distinction entre l'objet et la limite de la surface sur laquelle ils se déplacent. Par exemple, utiliser un ruban noir sur la surface comme limite et détecter l'objet à l'aide de capteurs de proximité.
- Mettre en place un mécanisme pour que les robots ne se poussent pas les uns les autres. Une méthode consisterait à utiliser un capteur de couleur et à attacher du ruban adhésif de couleur aux robots.
- Mettre en œuvre l'algorithme de poussée basé sur l'occlusion.
- Discuter de la possibilité d'utiliser la poussée basée sur l'occlusion en trois dimensions, par exemple, des robots sous-marins poussant un objet.

**15.4 Résumé**

La robotique en essaim utilise plusieurs robots dans une architecture distribuée pour effectuer une tâche. Avec une architecture distribuée, le système est résistant aux défaillances des robots individuels et flexible dans sa capacité à ajouter ou supprimer des robots lorsque l'échelle de la tâche change. Une architecture distribuée peut effectuer des tâches que les robots individuels ne peuvent pas réaliser, comme nous l'avons vu dans les exemples de robots combinant leurs forces. Enfin, plusieurs

robots peuvent agir simultanément à différents endroits éloignés les uns des autres. Ces avantages ont un prix : l'augmentation du coût des robots multiples, la complexité des mécanismes de coordination à mettre en œuvre et, dans certains cas, la perte de performance due au chevauchement entre les actions des différents robots.

### 15.5 Lecture complémentaire

Un aperçu de la robotique collective est donné dans [26] et la collection [13] se concentre sur la robotique en essaim. Pour les projets spécifiques présentés dans ce chapitre, voir :

- Karel le robot [36].
- BeeClust [6] et [41].
- ASSISIbf [40] et <http://assisi-project.eu>.
- Interaction physique (tirer un bâton) [24].
- Transport basé sur l'occlusion [9] et [8].



## Chapitre 16

# Kinématique d'un Manipulateur Robotique

Notre présentation s'est concentrée sur les robots mobiles. La plupart des robots éducatifs sont des robots mobiles et vous avez peut-être déjà rencontré des robots mobiles commerciaux tels que des aspirateurs robotisés. Vous n'avez probablement pas rencontré de *manipulateurs robotiques*, mais vous avez vu des images d'usines qui assemblent des circuits électroniques ou qui soudent des framed de voitures (Fig. 1.3). La différence la plus importante entre les robots mobiles et les robots fixes est l'environnement dans lequel ils travaillent. Un robot mobile se déplace dans un environnement comportant des obstacles et un sol irrégulier, de sorte que l'environnement n'est pas entièrement connu à l'avance. Un aspirateur robotisé ne vous demande pas de lui fournir un plan de votre appartement avec l'emplacement de chaque meuble, ni de le reprogrammer chaque fois que vous déplacez un canapé. Au lieu de cela, le robot détecte de manière autonome la disposition de l'appartement : les pièces et la position des meubles. Si les cartes et l'odométrie sont utiles pour déplacer un robot vers une position approximative, des capteurs doivent être utilisés pour localiser précisément le robot dans son environnement.

Un manipulateur robotique dans une usine est fixé à un sol en béton stable et sa construction est robuste : l'émission répétée des mêmes commandes déplacera le manipulateur exactement à la même position. Dans ce chapitre, nous présentons des algorithmes pour la cinématique des manipulateurs : comment les commandes données à un manipulateur et le mouvement du robot sont liés. La présentation se fera en termes de bras avec deux liens dans un plan dont les articulations peuvent tourner.

La cinématique comporte deux tâches complémentaires :

- *Cinématique avant* (Sect.16.1) : Compte tenu d'une séquence de commandes, quelle est la position finale du bras robotique ?
- *cinématique inverse* (Sect. 16.2) : Etant donné la position souhaitée du bras robotique, quelle séquence de commandes l'amènera à cette position ?

La cinématique avant est relativement facile à calculer, car le calcul du changement de position résultant du déplacement de chaque articulation fait appel à une simple trigonométrie. S'il y a plus d'une liaison, la position finale est calculée en effectuant les calculs pour une articulation après l'autre. La cinématique inverse est très difficile, car on part d'une position souhaitée et on doit chercher une séquence de commandes pour atteindre cette position. Un problème de cinématique inverse peut avoir une solution, plusieurs solutions ou même aucune solution.

Les calculs cinématiques sont effectués en termes de cadres de coordonnées. Un

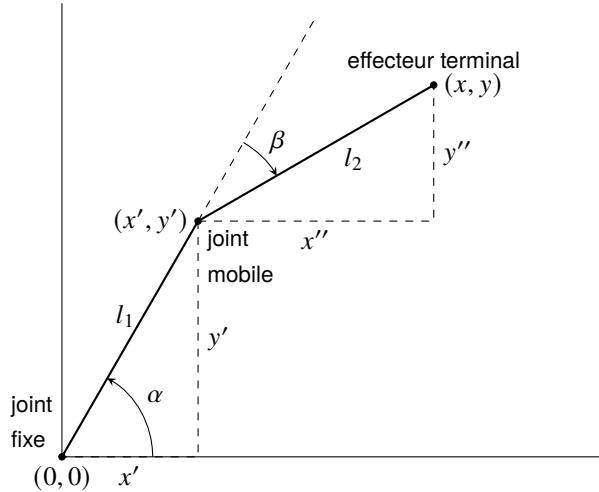


FIG. 16.1 – Cinématique avant d'un bras à deux articulations

cadre est attaché à chaque articulation du manipulateur et le mouvement est décrit comme des transformations d'un cadre à un autre par des rotations et des translations. La transformation des cadres de coordonnées en deux dimensions est présentée dans les sections 16.3–16.4. La plupart des robots manipulateurs sont tridimensionnels. Le traitement mathématique des mouvements en 3D dépasse le cadre de ce livre, mais nous espérons vous inciter à étudier ce sujet en vous présentant un aperçu des rotations en 3D dans les sections 16.5–16.6.

### 16.1 Cinématique de l'avant

Nous développons la cinématique d'un bras robotique bidimensionnel avec deux liens, deux articulations et un *effecteur final* tel qu'une pince, une soudeuse ou un pulvérisateur de peinture (Fig. 16.1). La première articulation peut tourner, mais elle est montée sur une base fixée à une table ou au sol. La liaison  $l_1$  relie cette articulation à une deuxième articulation qui peut se déplacer et tourner; une deuxième liaison  $l_2$  relie cette articulation à l'effecteur fixe.

Un système de coordonnées à deux dimensions est attribué avec la première articulation à  $(0,0)$ . Les longueurs des deux liens sont  $l_1$  et  $l_2$ . Faites pivoter la première articulation de  $\alpha$  pour déplacer l'extrémité de la première liaison avec la deuxième articulation à  $(x', y')$ . Faites maintenant pivoter la deuxième articulation de  $\beta$ . Quelles sont les coordonnées  $(x, y)$  de l'extrémité du bras, en fonction des deux constantes  $l_1, l_2$  et des deux paramètres  $\alpha, \beta$ ?

Projeter  $(x', y')$  sur les axes  $x$ - et  $y$ -; par trigonométrie ses coordonnées sont :

$$\begin{aligned} x' &= l_1 \cos \alpha \\ y' &= l_1 \sin \alpha. \end{aligned}$$

Prenons maintenant  $(x', y')$  comme origine d'un nouveau système de coordonnées et projetons  $(x, y)$  sur ses axes pour obtenir  $(x'', y'')$ . La position de l'effecteur dans le nouveau système de coordonnées est la suivante :

$$\begin{aligned} x'' &= l_2 \cos(\alpha + \beta) \\ y'' &= l_2 \sin(\alpha + \beta). \end{aligned}$$

Dans la figure 16.1,  $\beta$  est négatif (rotation dans le sens des aiguilles d'une montre), de sorte que  $\alpha + \beta$  est l'angle entre le deuxième maillon et la ligne parallèle à l'axe  $x$ .

En combinant les résultats, on obtient

$$\begin{aligned} x &= l_1 \cos \alpha + l_2 \cos(\alpha + \beta) \\ y &= l_1 \sin \alpha + l_2 \sin(\alpha + \beta). \end{aligned}$$

**Exemple** Soit  $l_1 = l_2 = 1$ ,  $\alpha = 60^\circ$ ,  $\beta = -30^\circ$ . Then :

$$\begin{aligned} x &= 1 \cdot \cos 60 + 1 \cdot \cos(60 - 30) = \frac{1}{2} + \frac{\sqrt{3}}{2} = \frac{1 + \sqrt{3}}{2} \\ y &= 1 \cdot \sin 60 + 1 \cdot \sin(60 - 30) = \frac{\sqrt{3}}{2} + \frac{1}{2} = \frac{1 + \sqrt{3}}{2}. \end{aligned}$$

Vérifions si ce résultat a un sens. La figure 16.2 montre un triangle formé par l'ajout d'une ligne entre  $(0, 0)$  et  $(x, y)$ . Le complément de l'angle  $\beta$  est  $180 - 30 = 150$  et le triangle est isocèle puisque les deux côtés sont 1, donc les autres angles du triangle sont égaux et leurs valeurs sont  $(180 - 150)/2 = 15$ . L'angle que forme la nouvelle droite avec l'axe  $x$  est  $60 - 15 = 45$ , ce qui est compatible avec  $x = y$ .

#### Activity 16.1: Cinématique vers l'avant

- Programmez votre robot de manière à ce qu'il suive la trajectoire du bras de la Fig. 16.1 : tournez à gauche  $60^\circ$ , avancez d'une unité (1m ou une autre distance convenable), tournez à droite  $30^\circ$ , avancez d'une unité.
- Mesurer les distances  $x$ - et  $y$ - du robot par rapport à l'origine et les comparer aux valeurs calculées par les équations de la cinématique d'avancement.

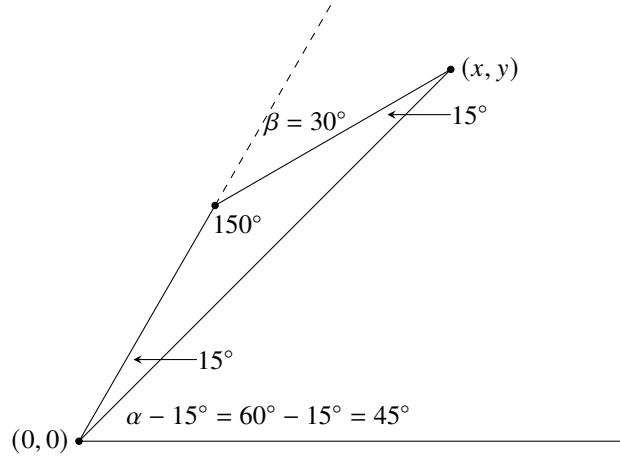


FIG. 16.2 – Calcul des angles

## 16.2 Cinématique inverse

L’anneau gris de la Fig. 16.3 montre le *workspace* du bras à deux bras, l’ensemble des positions que l’effecteur peut atteindre (nous supposons que  $l_2 < l_1$ ). (Nous supposons que  $l_2 < l_1$ .) L’espace de travail est circulairement symétrique puisque nous supposons qu’il n’y a pas de limites à la rotation des articulations sur un cercle complet entre  $-180^\circ$  et  $180^\circ$ . Tout point tel que  $a$  sur la circonférence du cercle extérieur est la position la plus éloignée du bras par rapport à l’origine; elle est obtenue lorsque les deux maillons sont alignés de manière à ce que la longueur du bras soit  $l_1 + l_2$ . Les positions les plus proches de l’origine dans l’espace de travail sont des points comme  $b$  sur la circonférence du cercle intérieur; elles sont obtenues lorsque le deuxième maillon est replié sur le premier maillon, ce qui donne une longueur de  $l_1 - l_2$ . Une autre position atteignable  $c$  est indiquée; il existe *deux* configurations (rotations des articulations) qui permettent au bras d’être positionné à  $c$ .

Sous l’hypothèse que  $l_2 < l_1$ , aucune séquence de rotations ne peut positionner l’extrémité du bras plus près de l’origine que  $l_1 - l_2$  et aucune position à une distance supérieure à  $l_1 + l_2$  de l’origine n’est accessible. La figure nous apprend qu’un problème de cinématique inverse, c’est-à-dire la recherche de commandes pour atteindre un point spécifié, peut avoir zéro, une ou plusieurs solutions.

Le calcul de la cinématique inverse utilise la *loi des cosinus* (Fig. 16.4) :

$$a^2 + b^2 - 2ab \cos \theta = c^2 .$$

Dans un triangle rectangle  $\cos 90^\circ = 0$  et la loi se réduit au théorème de Pythagore.

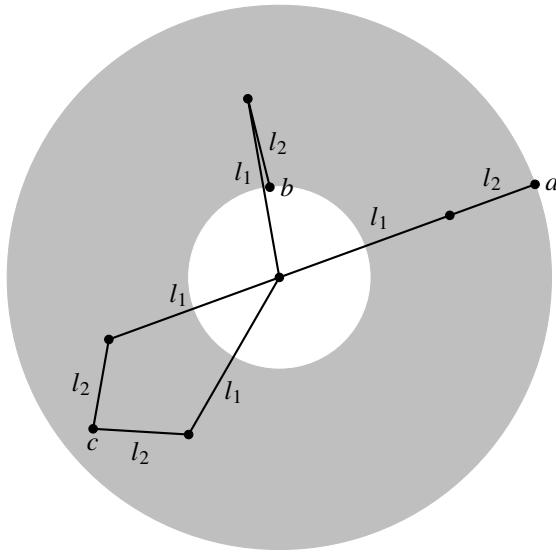


FIG. 16.3 – Espace de travail d'un bras à deux leviers

Supposons maintenant que l'on nous donne un point  $(x, y)$  et que nous voulions des valeurs pour  $\alpha, \beta$  (s'il en existe) qui amèneront le bras à ce point. Figure 16.5 similaire à la Fig. 16.2 sauf que les valeurs spécifiques sont remplacées par des angles et des longueurs arbitraires.

Par le théorème de Pythagore,  $r = \sqrt{x^2 + y^2}$ .

La loi des cosinus donne

$$l_1^2 + l_2^2 - 2l_1l_2 \cos(180^\circ - \beta) = r^2,$$

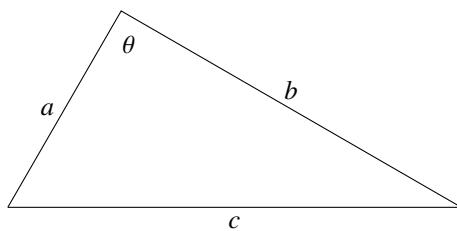


FIG. 16.4 – Loi des cosinus

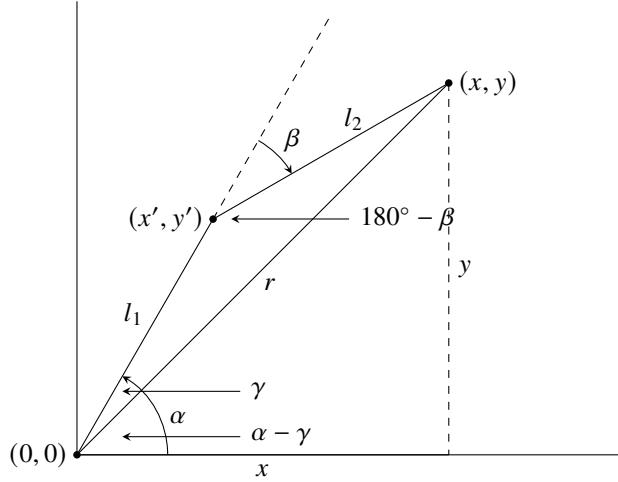


FIG. 16.5 – Cinématique inverse d'un bras à deux articulations

qui peut être résolu pour  $\beta$  :

$$\begin{aligned}\cos(180^\circ - \beta) &= \frac{l_1^2 + l_2^2 - r^2}{2l_1l_2} \\ \beta &= 180^\circ - \cos^{-1} \left( \frac{l_1^2 + l_2^2 - r^2}{2l_1l_2} \right).\end{aligned}$$

Pour obtenir  $\gamma$  puis  $\alpha$ , utiliser la loi des cosinus avec  $\gamma$  comme angle central :

$$\cos \gamma = \frac{l_1^2 + r^2 - l_2^2}{2l_1r}.$$

Du triangle rectangle formé par  $(x, y)$  on a :

$$\begin{aligned}\tan(\alpha - \gamma) &= \frac{y}{x} \\ \alpha &= \tan^{-1} \frac{y}{x} + \gamma,\end{aligned}$$

donc :

$$\alpha = \tan^{-1} \frac{y}{x} + \cos^{-1} \left( \frac{l_1^2 + r^2 - l_2^2}{2l_1r} \right).$$

**Exemple** Supposons à nouveau que  $l_1 = l_2 = 1$  et que l'effecteur se trouve au point calculé à partir de la cinématique avant :

$$(x, y) = \left( \frac{1 + \sqrt{3}}{2}, \frac{1 + \sqrt{3}}{2} \right).$$

Tout d'abord, calculez  $r^2$  :

$$r^2 = x^2 + y^2 = \left(\frac{1+\sqrt{3}}{2}\right)^2 + \left(\frac{1+\sqrt{3}}{2}\right)^2 = 2 + \sqrt{3},$$

et l'utiliser dans le calcul de  $\beta$  :

$$\begin{aligned}\beta &= 180^\circ - \cos^{-1} \left( \frac{1^2 + 1^2 - (2 + \sqrt{3})}{2 \cdot 1 \cdot 1} \right) \\ &= 180^\circ - \cos^{-1} \left( -\frac{\sqrt{3}}{2} \right) \\ &= 180^\circ \pm 150^\circ \\ &= \pm 30^\circ,\end{aligned}$$

puisque  $330^\circ = -30^\circ (\text{mod } 360^\circ)$ . Il y a deux solutions car il y a deux façons de déplacer le bras vers  $(x, y)$ .

Calculons ensuite  $\gamma$  :

$$\gamma = \cos^{-1} \left( \frac{1^2 + r^2 - 1^2}{2 \cdot 1 \cdot r} \right) = \cos^{-1} \left( \frac{r}{2} \right) = \cos^{-1} \left( \frac{\sqrt{2 + \sqrt{3}}}{2} \right) = \pm 15^\circ. \quad (16.1)$$

Le cosinus inverse peut être obtenu numériquement sur une calculatrice ou algébriquement comme indiqué dans l'Annexe B.7.

Puisque  $x = y$ , le calcul de  $\alpha$  est facile :

$$\alpha = \tan^{-1} \frac{y}{x} + \gamma = \tan^{-1} 1 + \gamma = 45^\circ \pm 15^\circ = 60^\circ \text{ or } 30^\circ.$$

La solution  $\alpha = 60^\circ, \beta = -30^\circ$  correspond à la rotation des articulations de la Fig. 16.1, tandis que la solution  $\alpha = 30^\circ, \beta = 30^\circ$  correspond à une rotation des deux articulations de  $30^\circ$  dans le sens inverse des aiguilles d'une montre.

Dans ce cas simple, il est possible de résoudre l'équation de la cinématique directe pour obtenir les formules de la cinématique inverse. En général, cela n'est pas possible, c'est pourquoi des solutions numériques approximatives sont utilisées.

#### Activity 16.2: Cinématique inverse

- Utilisez les formules de la cinématique inverse pour programmer votre robot afin qu'il se déplace vers une coordonnée donnée.
- Mesurer les distances  $x$ - et  $y$ - du robot par rapport à l'origine et les comparer aux coordonnées spécifiées.
- Si l'ordinateur de votre robot n'a pas la capacité de calculer les formules, calculez-les hors ligne et entrez ensuite les commandes au robot.

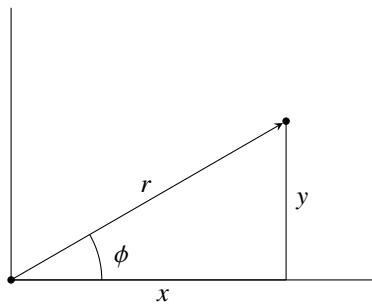


FIG. 16.6 – Un vecteur

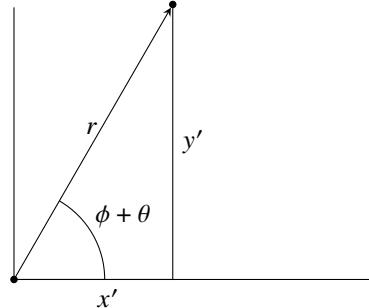


FIG. 16.7 – Le vecteur tourné par θ

### 16.3 Rotations

Le mouvement d'un manipulateur robotique est décrit en termes de *cadres de coordonnées*. Trois cadres sont associés au bras de la Fig. 16.1 : un cadre est associé à l'articulation à l'origine (que nous supposons fixée à une table ou au sol), un deuxième cadre est associé à l'articulation entre les deux liens, et un troisième cadre est associé à l'effecteur à l'extrémité du deuxième lien.

Dans cette section, nous décrivons comment le mouvement de rotation d'un bras robotique peut être modélisé mathématiquement à l'aide de *matrices de rotation*. Les liens des bras robotiques introduisent des *translations* : la deuxième articulation est décalée d'une distance linéaire de  $l_1$  par rapport à la première articulation, et l'effecteur est décalé d'une distance linéaire de  $l_2$  par rapport à la deuxième articulation. Le traitement mathématique des translations utilise une extension des matrices de rotation appelée *transformations homogènes*.

Les rotations peuvent prêter à confusion car une matrice de rotation peut avoir trois interprétations qui sont décrites dans les sous-sections suivantes : rotation d'un vecteur, rotation d'un cadre de coordonnées et transformation d'un vecteur d'un cadre de coordonnées à un autre.

#### 16.3.1 Rotation d'un vecteur

Considérons un vecteur avec des coordonnées cartésiennes  $(x, y)$  et des coordonnées polaires  $(r, \phi)$  (Fig. 16.6). On fait maintenant pivoter le vecteur d'un angle  $\theta$  (Fig. 16.7). Ses coordonnées polaires sont  $(r, \phi + \theta)$ . Quelles sont ses coordonnées cartésiennes ?

En utilisant les identités trigonométriques pour la somme de deux angles et la

conversion de  $(r, \phi)$  en  $(x, y)$  nous avons :

$$\begin{aligned} x' &= r \cos(\phi + \theta) \\ &= r \cos \phi \cos \theta - r \sin \phi \sin \theta \\ &= (r \cos \phi) \cos \theta - (r \sin \phi) \sin \theta \\ &= x \cos \theta - y \sin \theta, \end{aligned}$$

$$\begin{aligned} y' &= r \sin(\phi + \theta) \\ &= r \sin \phi \cos \theta + r \cos \phi \sin \theta \\ &= (r \sin \phi) \cos \theta + (r \cos \phi) \sin \theta \\ &= y \cos \theta + x \sin \theta \\ &= x \sin \theta + y \cos \theta. \end{aligned}$$

Ces équations peuvent être exprimées comme la multiplication d'une matrice appelée *matrice de rotation* et d'un vecteur :

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

**Exemple** Soit  $p$  le point situé à l'extrémité d'un vecteur de longueur  $r = 1$  qui forme un angle de  $\phi = 30^\circ$  avec l'axe positif  $x$ . Les coordonnées cartésiennes de  $p$  sont  $\left(\frac{\sqrt{3}}{2}, \frac{1}{2}\right)$ . Supposons que le vecteur subisse une rotation de  $\theta = 30^\circ$ . Quelles sont les nouvelles coordonnées cartésiennes de  $p$ ? En utilisant la multiplication des matrices :

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{\sqrt{3}}{2} \end{bmatrix}.$$

Le résultat est logique car la rotation d'un vecteur dont l'angle avec l'axe  $x$  est  $30^\circ$  par  $30^\circ$  devrait donner un vecteur dont l'angle avec l'axe  $x$  est  $60^\circ$ .

Supposons que le vecteur subisse une rotation supplémentaire de  $30^\circ$ ; ses nouvelles coordonnées sont :

$$\begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \left( \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix} \right) = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \frac{1}{2} \\ \frac{\sqrt{3}}{2} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \quad (16.2)$$

Ce résultat est également logique. La rotation d'un vecteur dont l'angle est de  $30^\circ$  deux fois par  $30^\circ$  (pour un total de  $60^\circ$ ) devrait donner  $90^\circ$ . Le cosinus de  $90^\circ$  est 0 et le sinus de  $90^\circ$  est 1.

La multiplication matricielle étant associative, la multiplication peut également être effectuée comme suit :

$$\left( \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \right) \begin{bmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix}. \quad (16.3)$$

### Activity 16.3: Matrices de rotation

- Démontrer l'associativité de la multiplication matricielle en montrant que la multiplication dans Eq. 16.3 donne le même résultat que la multiplication dans Eq. 16.2.
- Calculer la matrice pour une rotation de  $-30^\circ$  et montrer que la multiplication de cette matrice par la matrice pour une rotation de  $30^\circ$  donne la matrice pour une rotation de  $0^\circ$ .
- Cette multiplication est-elle commutative ?
- La multiplication des matrices de rotation à deux dimensions est-elle commutative ?

#### 16.3.2 Rotation d'un cadre de coordonnées

Réinterprétons les figures 16.6–16.7. La figure 16.8 montre un cadre de coordonnées (bleu) défini par deux vecteurs unitaires orthogonaux :

$$\vec{x} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \vec{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

La figure 16.9 montre le *bâtiment de coordonnées* tourné de  $\theta$  degrés (rouge). Les nouveaux vecteurs unitaires  $\vec{x}'$  et  $\vec{y}'$  peuvent être obtenus en les multipliant par la matrice de rotation dérivée ci-dessus :

$$\begin{aligned} \vec{x}' &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix} \\ \vec{y}' &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\sin \theta \\ \cos \theta \end{bmatrix}. \end{aligned}$$

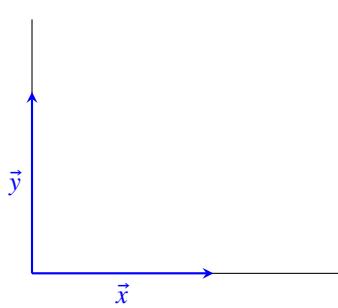


FIG. 16.8 – Cadre de coordonnées original (bleu)

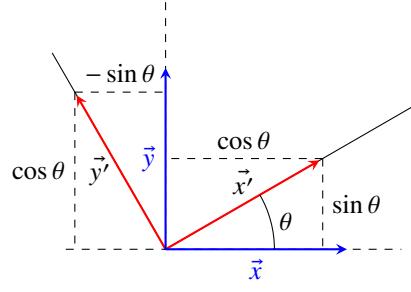


FIG. 16.9 – Nouveau cadre de coordonnées (rouge) obtenu en faisant pivoter le cadre de coordonnées original (bleu) de  $\theta$

**Exemple** Pour les vecteurs unitaires de la Fig. 16.8 et une rotation de  $30^\circ$  :

$$\vec{x}' = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix}$$

$$\vec{y}' = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} \\ \frac{\sqrt{3}}{2} \end{bmatrix}.$$

### 16.3.3 Transformation d'un vecteur d'un repère à un autre

L'origine d'un repère de coordonnées  $b$  (bleu) représente l'articulation d'un effecteur tel qu'une soudeuse et le point  $p$  est l'extrémité de la soudeuse (16.10). Par convention en robotique, le cadre de coordonnées d'une entité est désigné par un exposant " pré ".<sup>1</sup> Dans le framed  $b$ , le point  ${}^b p$  a des coordonnées polaires  $(r, \phi)$  et des coordonnées cartésiennes  $({}^b x, {}^b y)$  reliées par les formules trigonométriques habituelles :

$${}^b p = ({}^b x, {}^b y) = (r \cos \phi, r \sin \phi).$$

Supposons que l'articulation (*avec son cadre de coordonnées*) subisse une rotation de l'angle  $\theta$ . Les coordonnées du point par rapport à  $b$  restent les mêmes, mais le cadre de coordonnées a été déplacé. Nous posons donc la question suivante : quelles sont les coordonnées  ${}^a p = ({}^a x, {}^a y)$  du point dans le cadre de coordonnées avant qu'il n'ait été déplacé ? Dans la figure 16.11, le cadre original  $b$  est représenté tourné vers une nouvelle position (et toujours en bleu), tandis que le cadre de coordonnées  $a$

1. La convention est d'utiliser des lettres majuscules à la fois pour le cadre et les coordonnées, mais nous utilisons des minuscules pour plus de clarté.

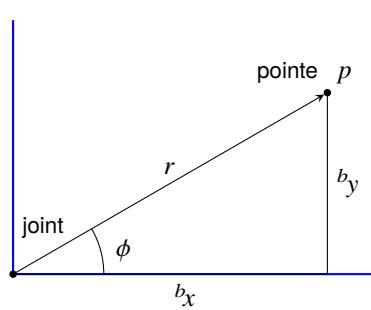


FIG. 16.10 – Point  $p$  à l'extrémité d'un effecteur dans le repère de coordonnées  $b$  (bleu)

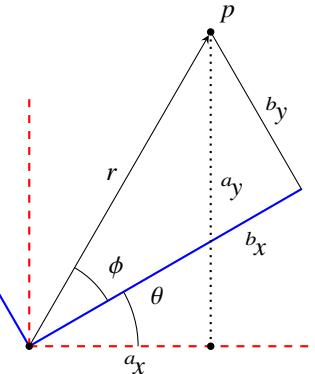


FIG. 16.11 – Point  $p$  dans les cadres de coordonnées  $a$  (rouge) et  $b$  (bleu)

se trouve dans l'ancienne position de  $b$  et est représenté par des lignes pointillées rouges. Dans la section précédente, nous avons demandé comment transformer un cadre de coordonnées en un autre ; ici, nous demandons comment transformer les coordonnées d'un point dans un cadre en ses coordonnées dans un autre cadre.

En ce qui concerne le bras robotique, nous connaissons  $({}^b_x, {}^b_y)$ , les coordonnées de l'extrémité de l'effecteur par rapport au framed de l'effecteur, et nous demandons maintenant ses coordonnées  ${}^a_p = ({}^a_x, {}^a_y)$  par rapport à la base fixe. Ceci est important car si nous connaissons  ${}^a_p$ , nous pouvons calculer la distance et l'angle entre l'extrémité de la soudeuse et les parties de la voiture qu'elle doit maintenant souder.

Nous pouvons répéter le calcul utilisé pour la rotation d'un vecteur :

$$\begin{aligned} {}^a_x &= r \cos(\phi + \theta) \\ &= r \cos \phi \cos \theta - r \sin \phi \sin \theta \\ &= {}^b_x \cos \theta - {}^b_y \sin \theta, \end{aligned}$$

$$\begin{aligned} {}^a_y &= r \sin(\phi + \theta) \\ &= r \sin \phi \cos \theta + r \cos \phi \sin \theta \\ &= {}^b_x \sin \theta + {}^b_y \cos \theta, \end{aligned}$$

pour obtenir la matrice de rotation :

$$\begin{bmatrix} {}^a_x \\ {}^a_y \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} {}^b_x \\ {}^b_y \end{bmatrix}. \quad (16.4)$$

La matrice est appelée matrice de rotation *de l'image b à l'image a* et est notée  ${}^a_b R$ . La prémultiplication du point  ${}^b p$  dans le repère  $b$  par la matrice de rotation donne à  ${}^a p$  ses coordonnées dans le repère  $a$  :

$${}^a p = {}^a_b R {}^b p .$$

**Exemple** Soit  ${}^b p$  le point du repère  $b$  situé à l'extrémité d'un vecteur de longueur  $r = 1$  qui forme un angle de  $\phi = 30^\circ$  avec l'axe positif  $x$ . Les coordonnées de  ${}^b p$  sont  $\left(\frac{\sqrt{3}}{2}, \frac{1}{2}\right)$ . Supposons que le cadre de coordonnées  $b$  (ainsi que le point  $p$ ) subisse une rotation de  $\theta = 30^\circ$  pour obtenir le cadre de coordonnées  $a$ . Quelles sont les coordonnées de  ${}^a p$ ? En utilisant l'Eq. 16.4 :

$${}^a p = \begin{bmatrix} {}^a_x \\ {}^a_y \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{\sqrt{3}}{2} \end{bmatrix} .$$

Si le cadre  $a$  subit une rotation de  $30^\circ$ , on obtient les coordonnées du point dans un troisième cadre  $a1$ . Pré-multiplier  ${}^a p$  par la matrice de rotation pour  $30^\circ$  pour obtenir  ${}^{a1} p$  :

$${}^{a1} p = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \left( \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix} \right) = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \frac{1}{2} \\ \frac{\sqrt{3}}{2} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} .$$

Le produit des deux matrices de rotation :

$$\begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & \frac{1}{2} \end{bmatrix}$$

donne la matrice de rotation pour faire pivoter le cadre de coordonnées original  $b$  de  $60^\circ$ :

$$\begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} .$$

Étant donné une séquence de rotations, la prémultiplication de leurs matrices de rotation donne la matrice de rotation pour la rotation équivalente à la somme des rotations individuelles.

## 16.4 Rotation et translation d'un cadre de coordonnées

Les articulations des manipulateurs robotiques sont reliées par des liens, de sorte que les systèmes de coordonnées sont liés non seulement par des rotations, mais

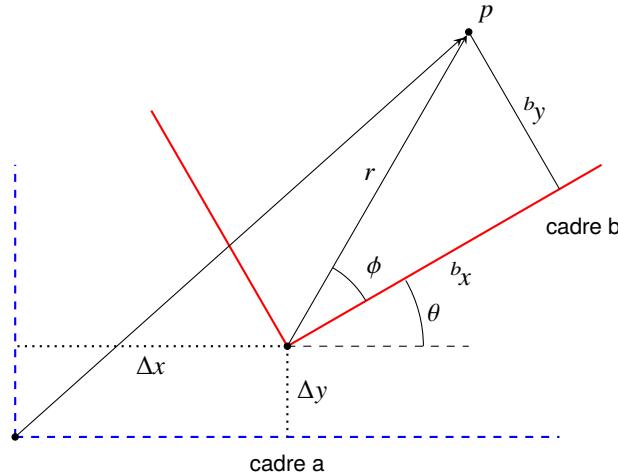


FIG. 16.12 – La frame *b* subit une rotation et une translation vers la frame *a*

aussi par des translations. Le point *p* de la Fig. 16.12 représente un point dans le repère de coordonnées (rouge) *b*, mais par rapport au repère de coordonnées (bleu en pointillés) *a*, le repère *b* subit une rotation de l'angle  $\theta$  et son origine est translatée de  $\Delta x$  et  $\Delta y$ . Si l'on connaît  ${}^b p = ({}^b x, {}^b y)$ , les coordonnées du point dans le repère *b*, quelles sont ses coordonnées  ${}^a p = ({}^a x, {}^a y)$  dans le repère *a* ?

Pour effectuer ce calcul, nous définissons un repère intermédiaire (vert) *a1* qui a la même origine que *b* et la même orientation que *a* (Fig. 16.13). Quelles sont les coordonnées  ${}^{a1} p = ({}^{a1} x, {}^{a1} y)$  du point dans le cadre *a1*? Il s'agit simplement de la rotation par  $\theta$  que nous avons effectuée précédemment :

$${}^{a1} p = \begin{bmatrix} {}^{a1} x \\ {}^{a1} y \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} {}^b x \\ {}^b y \end{bmatrix}.$$

Maintenant que nous avons les coordonnées du point dans *a1*, il est facile d'obtenir les coordonnées dans le framed *a* en ajoutant les décalages de la translation. Sous forme de matrice :

$${}^a p = \begin{bmatrix} {}^a x \\ {}^a y \end{bmatrix} = \begin{bmatrix} {}^{a1} x \\ {}^{a1} y \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}.$$

*transformations homogènes* sont utilisées pour combiner une rotation et une translation en un seul opérateur. Le vecteur bidimensionnel donnant les coordonnées

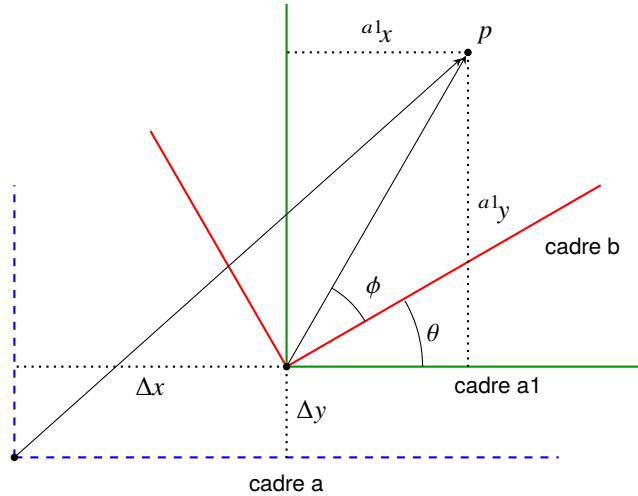


FIG. 16.13 – Le cadre *b* subit une rotation jusqu’au cadre *a1*, puis une translation jusqu’au cadre *a*

d’un point est étendu avec un troisième élément qui a une valeur fixe de 1 :

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

La matrice de rotation est étendue à une matrice  $3 \times 3$  avec un 1 dans le coin inférieur droit et des zéros ailleurs. Il est facile de vérifier que la multiplication d’un vecteur dans le cadre *b* par la matrice de rotation donne le même vecteur qu’auparavant, à l’exception de l’élément 1 supplémentaire :

$$\begin{bmatrix} a^1_x \\ a^1_y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ 1 \end{bmatrix}.$$

Le résultat est les coordonnées du point dans le framed intermédiaire *a1*. Pour obtenir les coordonnées dans le cadre *a*, on multiplie par une matrice qui effectue la

translation :

$$\begin{bmatrix} a_x \\ a_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_1^x \\ a_1^y \\ 1 \end{bmatrix}.$$

En multipliant les deux transformées, on obtient une seule transformée homogène qui peut effectuer à la fois la rotation et la translation :

$$\begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & \Delta x \\ \sin \theta & \cos \theta & \Delta y \\ 0 & 0 & 1 \end{bmatrix}.$$

**Exemple** Étendons l'exemple précédent en ajoutant une translation de (3, 1) à la rotation de  $30^\circ$ . La transformée homogène de la rotation suivie de la translation est :

$$\begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & 3 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

Les coordonnées du point dans le cadre  $a$  sont :

$$\begin{bmatrix} a_x \\ a_y \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & 3 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} + 3 \\ \frac{\sqrt{3}}{2} + 1 \\ 1 \end{bmatrix}.$$

#### Activity 16.4: transformations homogènes

- Dessinez le diagramme d'une rotation de  $-30^\circ$  suivie d'une translation de  $(3, -1)$ .
- Calculer la transformée homogène.

## 16.5 Un goût de rotations tridimensionnelles

Les concepts de transformation des coordonnées et de cinématique en trois dimensions sont les mêmes qu'en deux dimensions, mais les mathématiques sont plus complexes. En outre, beaucoup d'entre nous ont du mal à visualiser un mouvement tridimensionnel lorsque tout ce qui nous est montré est une représentation

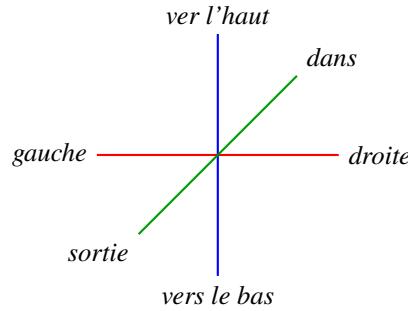


FIG. 16.14 – Cadre de coordonnées tridimensionnelles

bidimensionnelle d'objets tridimensionnels. Dans cette section, nous donnerons un avant-goût de la robotique tridimensionnelle en examinant les rotations en trois dimensions.

### 16.5.1 Rotations autour des trois axes

Un repère de coordonnées bidimensionnel  $x$ - $y$  peut être considéré comme étant encadré dans un repère de coordonnées tridimensionnel en ajoutant un axe  $z$  perpendiculaire aux axes  $x$  et  $y$ . La figure 16.14 montre une représentation bidimensionnelle du cadre tridimensionnel. L'axe  $x$  est tracé à gauche et à droite sur le papier et l'axe  $y$  est tracé de haut en bas. La ligne diagonale représente l'axe  $z$  qui est perpendiculaire aux deux autres axes. Le cadre de coordonnées "standard"  $x$ - $y$ - $z$  a les directions positives de ses axes définies par la règle de la main droite (voir ci-dessous). Les directions positives sont *droite* pour l'axe  $x$ , *haut* pour l'axe  $y$  et *sortie* (du papier vers l'observateur) pour l'axe  $z$ .

Faire pivoter le cadre de coordonnées dans le sens inverse des aiguilles d'une montre autour de l'axe  $z$ , de sorte que l'axe  $z$  reste inchangé (Figs. 16.15–16.16). La nouvelle orientation du framed est (*ver l'haut*, *gauche*, *sortie*). Considérons maintenant une rotation de  $90^{\circ}rc$  autour de l'axe  $x$  (Figs. 16.17–16.18). L'axe  $y$  sort alors du papier et l'axe  $z$  tombe sur le papier, ce qui donne l'orientation (*droite*, *sortie*, *ver le bas*). Enfin, considérons une rotation de  $90^{\circ}$  autour de l'axe  $y$  (Figs. 16.19–16.20). L'axe  $x$  "tombe" dans le papier et l'axe  $z$  "tombe" sur le papier. La nouvelle position du framed est (*dans*, *ver l'haut*, *right*).

### 16.5.2 La règle de la main droite

Il y a deux orientations pour chaque axe, soit  $2^3 = 8$  orientations au total. Ce qui compte, c'est l'orientation relative d'un axe par rapport aux deux autres; par exemple, une fois que les axes  $x$ - et  $y$ - ont été choisis pour se situer dans le plan du

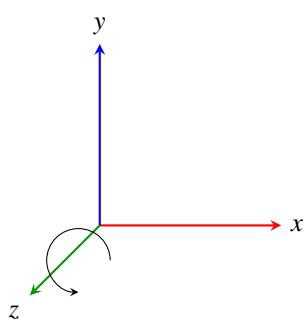


FIG. 16.15 – *x-y-z cadre de coordonnées*

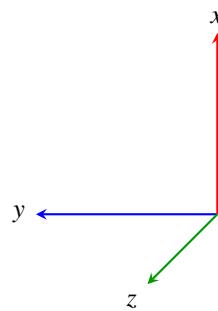


FIG. 16.16 – *x-y-z cadre de coordonnées après rotation de 90° autour de l'axe z*

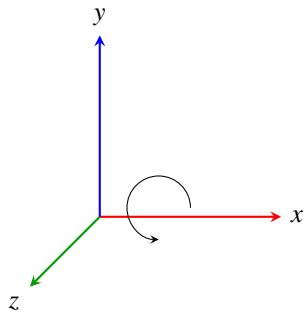


FIG. 16.17 – *cadre de coordonnées x-y-z*

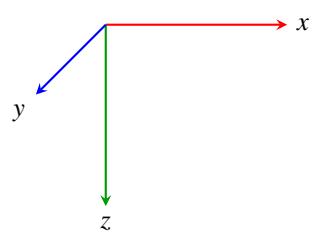


FIG. 16.18 – *base de coordonnées x-y-z après une rotation de 90° autour de l'axe x*

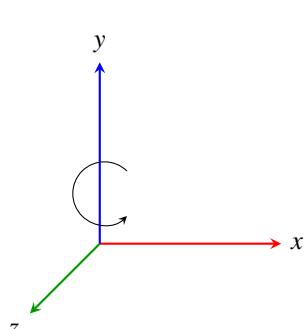


FIG. 16.19 – *cadre de coordonnées x-y-z*

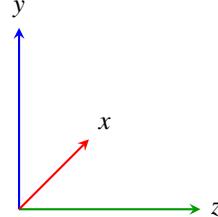


FIG. 16.20 – *cadre de coordonnées x-y-z après une rotation de 90° autour de l'axe y*

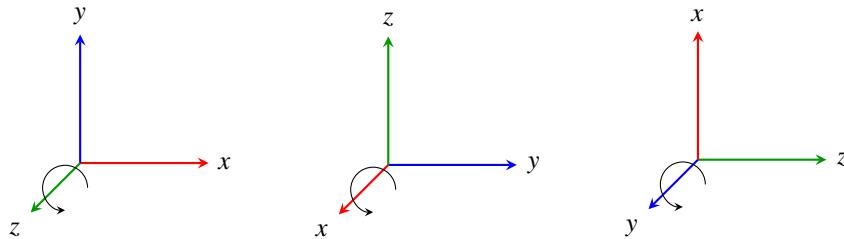


FIG. 16.21 – La règle de droite

papier, l'axe  $z$  peut avoir sa direction positive pointant vers l'extérieur du papier ou vers l'intérieur du papier. Le choix doit être cohérent. La convention en physique et en mécanique est la *règle de la main droite*. Courbez les doigts de votre main droite de manière à ce qu'ils passent d'un axe à un autre. Votre pouce pointe maintenant dans la direction positive du troisième axe. Pour les axes familiers  $x$ - et  $y$ - sur le papier, recourbez vos doigts sur le chemin allant de l'axe  $x$  à l'axe  $y$ . Ce faisant, votre pouce pointe vers l'extérieur du papier, ce qui est considéré comme la direction positive de l'axe  $z$ . La figure 16.21 montre le système de coordonnées de la main droite avec chacun des trois axes pointant *hors du papier*. Selon la règle de la main droite, les trois rotations sont :

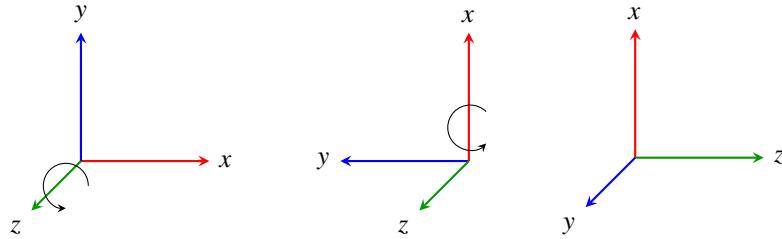
- Rotation de  $x$  à  $y$  autour de  $z$ ,
- Rotation de  $y$  à  $z$  autour de  $x$
- Rotation de  $z$  à  $x$  autour de  $y$ .

### 16.5.3 Matrices pour les rotations tridimensionnelles

Une matrice de rotation tridimensionnelle est une matrice 3 fois 3 car chaque point  $p$  d'un frame a trois coordonnées  $p_x, p_y, p_z$  qui doivent être déplacées. Commençons par une rotation de  $\psi$  autour de l'axe  $z$ , suivie d'une rotation de  $\theta$  autour de l'axe  $y$  et enfin d'une rotation de  $\phi$  autour de l'axe  $x$ . Pour la première rotation autour de l'axe  $z$ , les coordonnées  $x$  et  $y$  sont tournées comme en deux dimensions et la coordonnée  $z$  reste inchangée. Par conséquent, la matrice est :

$$R_{z(\psi)} = \begin{bmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Pour la rotation par  $\theta$  autour de l'axe  $y$ , la coordonnée  $y$  est inchangée et les coordonnées  $z$  et  $x$  sont transformées "comme si" elles étaient les coordonnées  $x$  et

FIG. 16.22 – Rotation autour de l'axe  $z$  suivie d'une rotation autour de l'axe  $x$ .

y d'une rotation autour de l'axe  $z$  :

$$R_{y(\theta)} = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix}.$$

Pour la rotation par  $\phi$  autour de l'axe  $x$ , la coordonnée  $x$  est inchangée et les coordonnées  $y$  et  $z$  sont transformées "comme si" elles étaient les coordonnées  $x$  et  $y$  d'une rotation autour de l'axe  $z$  :

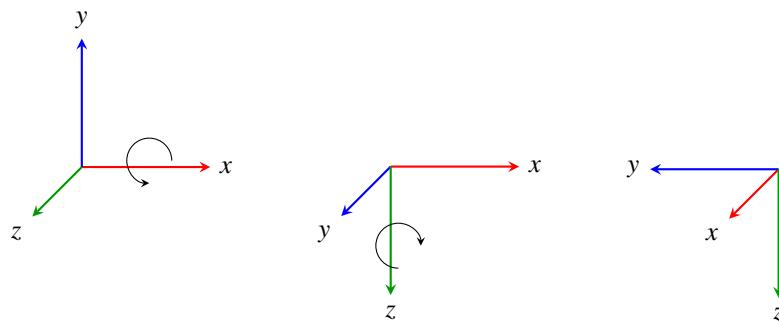
$$R_{x(\phi)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi & \cos \phi \end{bmatrix}.$$

Il peut sembler étrange que dans la matrice pour la rotation autour de l'axe  $y$  les signes de la fonction sinus aient changé. Pour vous convaincre que la matrice pour cette rotation est correcte, redessinez le diagramme de la Fig. 16.11, en remplaçant  $z$  par  $x$  et  $x$  par  $y$  et effectuez le calcul trigonométrique.

#### 16.5.4 Multiples rotations

Il y a une mise en garde à propos de la composition des rotations : comme pour la multiplication matricielle, rotations tridimensionnelles *ne pas* commuter. Démontrons-le à l'aide d'une simple séquence de deux rotations. Considérons une rotation de  $90^\circ$  autour de l'axe  $z$ , suivie d'une rotation de  $90^\circ$  autour de la (nouvelle position de) l'axe  $x$  (Fig. 16.22). Le résultat peut être exprimé par (*up, out, droite*).

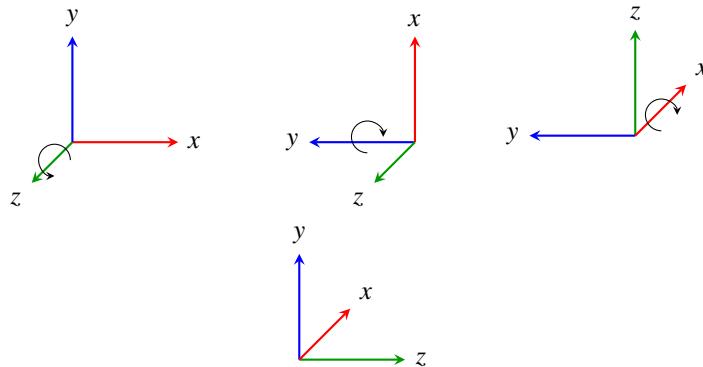
Considérons maintenant l'opération commuée : une rotation de  $90^\circ$  autour de l'axe  $x$ , suivie d'une rotation de  $90^\circ$  autour de l'axe  $z$  (Fig. 16.23). Le résultat peut être exprimé comme (*out, gauche, bas*), ce qui n'est pas la même chose que l'orientation précédente.

FIG. 16.23 – Rotation autour de l’axe  $x$  suivie d’une rotation autour de l’axe  $z$ 

### 16.5.5 angles d’Euler

Une rotation arbitraire peut être obtenue par trois rotations individuelles autour des trois axes, de sorte que la matrice pour une rotation arbitraire peut être obtenue en multipliant les matrices pour chaque rotation individuelle. Les angles des rotations sont appelés *angles d’Euler*. Les formules sont quelque peu complexes et peuvent être trouvées dans les références citées à la fin du chapitre. Nous démontrons ici les angles d’Euler à l’aide d’un exemple.

**Exemple** La figure 16.24 montre un cadre de coordonnées tourné séquentiellement  $90^\circ$  autour de l’axe  $z$ , puis de l’axe  $y$  et enfin de l’axe  $x$ . C’est ce qu’on appelle une rotation d’angle d’Euler  $zyx$ . L’orientation finale est (*in, up, right*).

FIG. 16.24 – Angles d’Euler  $zyx$  de  $(90^\circ, 90^\circ, 90^\circ)$

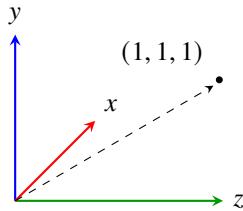


FIG. 16.25 – Vecteur après la rotation finale

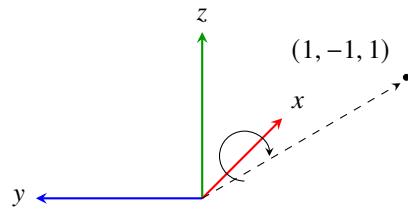


FIG. 16.26 – Vecteur avant rotation autour de l'axe x

Considérons un manipulateur robotique composé d'une seule articulation qui peut tourner autour des trois axes. Une séquence de rotations est effectuée comme le montre la Fig. 16.24. Considérons le point aux coordonnées (1, 1, 1) par rapport à l'articulation (Fig. 16.25). Après les rotations, quelles sont les coordonnées de ce point dans le framed original ?

On peut le calculer en laissant le vecteur fixe et en tenant compte des rotations des cadres de coordonnées. Pour atteindre la position finale indiquée sur la Fig. 16.25, le cadre a subi une rotation autour de l'axe  $x$  à partir de l'orientation indiquée sur la Fig. 16.26. En examinant la figure, nous voyons que les coordonnées dans ce framed sont (1, -1, 1). En passant par les deux framed précédents (Figs. 16.27, 16.28), les coordonnées sont (1, -1, -1) et (1, 1, -1).

Ces coordonnées peuvent être calculées à partir des matrices de rotation pour les rotations autour des trois axes. Les coordonnées du cadre de coordonnées final sont (1, 1, 1), donc dans le cadre avant la rotation autour de l'axe  $x$  les coordonnées

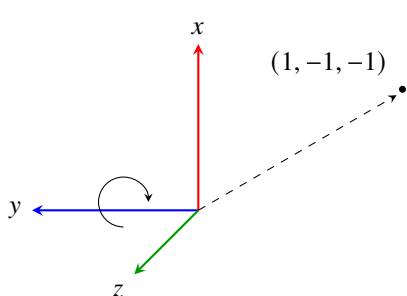


FIG. 16.27 – Vecteur avant rotation autour de l'axe y

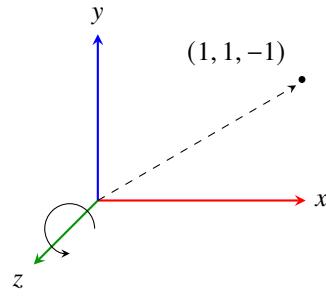


FIG. 16.28 – Vecteur dans le framed fixe avant la rotation autour de l'axe z

étaient :

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}.$$

Les coordonnées dans le framed avant la rotation autour de l'axe  $y$  étaient :

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}.$$

Enfin, les coordonnées dans le cadre fixe avant la rotation autour de l'axe  $z$  étaient :

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}.$$

Pour trois rotations arbitraires de l'angle d'Euler  $zyx$  :  $\psi$  autour de l'axe  $z$ , puis  $\theta$  autour de l'axe  $y$  et enfin  $\phi$  autour de l'axe  $x$ , la matrice de rotation est la suivante :  $\psi$  autour de l'axe  $y$ , puis  $\theta$  autour de l'axe  $x$  :

$$R = R_{z(\psi)}R_{y(\theta)}R_{x(\phi)}.$$

Il peut sembler étrange que l'ordre de la multiplication de la matrice (qui se fait toujours de droite à gauche) soit opposé à l'ordre des rotations. Cela s'explique par le fait que nous prenons un vecteur dans le cadre de coordonnées final et que nous le retrouvons dans le cadre fixe pour déterminer ses coordonnées dans le cadre fixe.

#### Activity 16.5: Angles d'Euler multiples

- Multiplier les trois matrices pour obtenir une seule matrice qui transforme directement les coordonnées de  $(1, 1, 1)$  en  $(1, 1, -1)$ .
- Effectuer le même calcul pour d'autres rotations, en changeant la séquence des axes et les angles de rotation.

#### 16.5.6 Le nombre de rotations distinctes de l'angle d'Euler

Il y a trois axes, donc il devrait y avoir  $3^3 = 27$  séquences d'angles d'Euler. Cependant, il est inutile de tourner autour du même axe deux fois de suite car le

même résultat peut être obtenu en tournant une fois par la somme des angles. Il n'y a donc que  $3 \cdot 2 \cdot 2 = 12$  séquences d'angles d'Euler différentes. L'activité suivante vous demande d'explorer différentes séquences d'angles d'Euler.

#### Activity 16.6: Angles d'Euler distincts

- Pour expérimenter les rotations tridimensionnelles, il est utile de construire un cadre de coordonnées à partir de trois crayons ou pailles perpendiculaires entre eux.
- Dessinez les cadres de coordonnées pour une rotation d'angle d'Euler de  $zyz$ , où chaque rotation est de  $90^\circ$ .
- Quelle rotation de  $zyz$  donne le même résultat que la rotation de  $zyx$  montrée sur la Fig. reffig.euler ?
- Expérimentez avec d'autres séquences de rotation et avec des angles autres que  $90^\circ$ .

## 16.6 Thèmes avancés sur les transformations tridimensionnelles

Maintenant que vous avez goûté aux rotations tridimensionnelles, nous passons en revue les étapes suivantes de l'apprentissage de ce sujet que vous pouvez étudier dans les manuels cités dans les références.

Il existe 12 angles d'Euler et le choix de celui à utiliser dépend de l'application envisagée. De plus, il existe une manière différente de définir les rotations. Les angles d'Euler sont des transformations *moving axes*, c'est-à-dire que chaque rotation se fait autour de la *nouvelle* position de l'axe après la rotation précédente. Dans la figure 16.24, la deuxième rotation se fait autour de l'axe  $y$  qui pointe maintenant vers la gauche, et non autour de l'axe  $y$  original qui pointe vers le haut. Il est également possible de définir des rotations *axes fixes* dans lesquelles les rotations suivantes se font autour des axes originaux du système de coordonnées. En trois dimensions, les transformations homogènes qui incluent des translations en plus des rotations peuvent être efficacement représentées sous la forme de matrices  $4 \times 4$ .

Les angles d'Euler sont relativement inefficaces à calculer et souffrent d'instabilités de calcul. On peut y remédier en utilisant *quaternions*, qui sont une généralisation des nombres complexes. Les quaternions utilisent trois nombres "imaginaires"  $i, j, k$ , où :

$$i^2 = j^2 = k^2 = ij = -1.$$

Rappelons qu'un vecteur dans le plan à deux dimensions peut être exprimé comme un nombre complexe  $x + i y$ . La rotation du vecteur d'un angle  $\theta$  peut être effectuée en multipliant par la valeur  $\cos \theta + i \sin \theta$ . De même, en trois dimensions, un vecteur peut être exprimé comme un *quaternion pur* avec une composante réelle nulle :  $p = 0 + x\vec{i} + y\vec{j} + z\vec{k}$ . Étant donné un axe et un angle, il existe un quaternion  $q$

qui fait tourner le vecteur autour de l'axe de cet angle en utilisant la formule  $qpq^{-1}$ . Ce calcul est plus efficace et plus robuste que le calcul équivalent avec les angles d'Euler et est utilisé dans une variété de contextes tels que le contrôle des avions et l'infographie.

### 16.7 Résumé

La cinématique est la description du mouvement d'un robot. Dans la cinématique avant, nous recevons un ensemble de commandes pour le robot et nous devons calculer sa position finale par rapport à sa position initiale. Dans la cinématique inverse, on nous donne une position finale souhaitée et nous devons calculer les commandes qui amèneront le robot à cette position. Ce chapitre a démontré les calculs cinématiques pour un bras manipulateur robotique bidimensionnel simple. Dans la pratique, les manipulateurs se déplacent en trois dimensions et les calculs sont plus difficiles. Il est généralement impossible de trouver des solutions exactes pour calculer la cinématique inverse et des solutions numériques approximatives sont utilisées.

Il existe de nombreuses façons de définir et de calculer des rotations arbitraires. Nous avons mentionné les angles d'Euler où une rotation arbitraire est obtenue par une séquence de trois rotations autour des axes de coordonnées. Les quaternions, une généralisation des nombres complexes, sont souvent utilisés dans la pratique parce qu'ils sont plus efficaces et plus robustes en termes de calcul.

### 16.8 Lecture complémentaire

Les manuels avancés sur la cinématique robotique et les sujets connexes sont ceux de Craig [12] et de Spong et al. [44]. Voir également le chapitre 3 de Correll [11]. L'annexe B de [12] contient les matrices de rotation pour toutes les séquences d'angles d'Euler. Les cours vidéo d'Angela Sodemann sont très utiles :

<https://www.youtube.com/user/asodemann3>,  
<http://www.robogrok.com/Flowchart.html>.

Bien qu'il ne s'agisse pas d'un livre sur la robotique, la monographie de Vince sur les quaternions [48] donne une excellente présentation des mathématiques des rotations.



## Annexe A

### Unités de mesure

Les tableaux A.1 et A.2 présentent les unités de mesure et leurs abréviations.

#### Exemples :

- $20 \text{ kHz} = 20 \text{ kilohertz} = 20,000 \text{ hertz}$
- $15 \text{ cm} = 15 \text{ centimeters} = \frac{15}{100} \text{ meters}$
- $50 \text{ ms} = 50 \text{ milliseconds} = \frac{50}{1000} \text{ seconds}$
- $10 \mu\text{s} = 10 \text{ microseconds} = \frac{10}{1000} \text{ milliseconds} = \frac{10}{1000000} \text{ seconds}$

Angles, such as the heading of a robot and the direction to an object, are measured in degrees or radians (Fig. A.1). By convention, angles are positive in the counter-clockwise direction and negative in the clockwise direction as measured from the front of the robot.

TAB. A.1 – *Unités de mesure*

| Propriété    | Variable | Unité                      | Abréviation      |
|--------------|----------|----------------------------|------------------|
| Distance     | $s$      | mètre                      | m                |
| Temps        | $t$      | seconde                    | s                |
| Vélocité     | $v$      | mètre/seconde              | m/s              |
| Accélération | $a$      | mètre/seconde <sup>2</sup> | m/s <sup>2</sup> |
| Fréquence    | $f$      | hertz                      | Hz               |
| Angle        | $\theta$ | radian                     | rad              |
|              |          | degré                      | °                |

TAB. A.2 – *Prefixes*

| Préfixe | signification | abréviation |
|---------|---------------|-------------|
| kilo-   | millièmes     | k           |
| centi-  | centièmes     | c           |
| milli-  | millièmes     | m           |
| micro   | millionièmes  | $\mu$       |

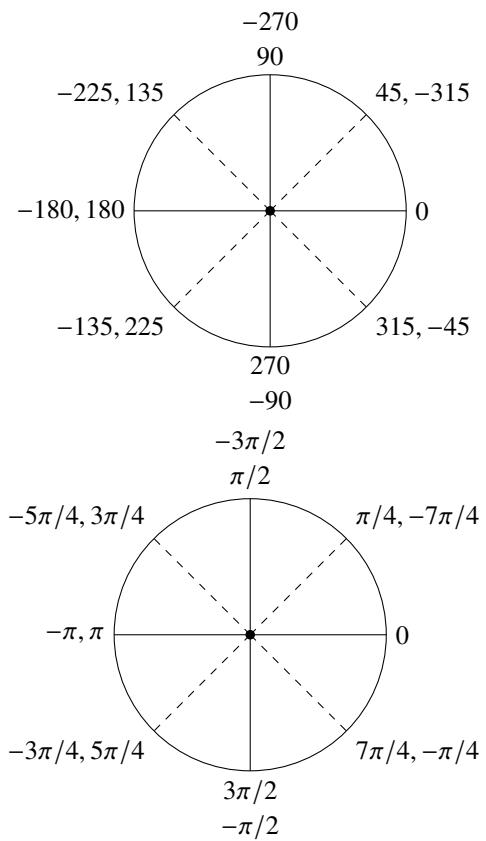


FIG. A.1 – Angles en degrés (à gauche) et en radians (à droite)

## Annexe B

# Dérivations mathématiques et tutoriels

Cette annexe rassemble les dérivations mathématiques utilisées dans le texte ainsi que de courts tutoriels sur des concepts qui peuvent ne pas être familiers.

### B.1 Probabilité conditionnelle et règle de Bayes

Compte tenu d'une lecture  $z$  du capteur, quelle est la probabilité que nous nous trouvions à la position  $x_i$ ? Cette probabilité est exprimée sous la forme d'une *probabilité conditionnelle*  $p(x_i | z)$ . Les données dont nous disposons sont la probabilité *actuelle*  $p(x_i)$  que nous nous trouvons à  $x_i$ , et  $p(z | x_i)$ , la probabilité conditionnelle que le capteur lise  $z$  si nous nous trouvons en fait à  $x_i$ . Multiplions ces deux probabilités :

$$p(z | x_i) p(x_i).$$

Qu'est-ce que cela signifie? L'événement  $x_i$  se produit avec la probabilité  $p(x_i)$  et une fois qu'il s'est produit, l'événement  $z$  se produit avec la probabilité  $p(z | x_i)$ . Il s'agit donc de la probabilité que  $z$  et  $x_i$  se produisent, appelée *probabilité conjointe* des deux événements :

$$p(z \cap x_i) = p(z | x_i) p(x_i).$$

La probabilité conjointe peut également être obtenue en multipliant la probabilité conditionnelle de  $x_i$  étant donné  $z$  par la probabilité de  $z$ :

$$p(x_i \cap z) = p(x_i | z) p(z).$$

La probabilité conjointe est commutative, donc en égalant les deux expressions, nous avons :

$$p(x_i | z) = p(x_i \cap z) = p(z \cap x_i) = p(z | x_i) p(x_i),.$$

En divisant par  $p(z)$ , on obtient :

$$p(x_i | z) = \frac{p(z | x_i) p(x_i)}{p(z)},$$

ce qui est connu sous le nom de *règle de Bayes*.

Si nous connaissons  $p(z | x_i)$  et  $p(x_i)$  pour chaque  $i$ ,  $p(z)$ , la *probabilité totale* de l'événement  $z$ , peut être calculée en additionnant les probabilités individuelles connues :

$$p(z) = \sum_i p(z | x_i) p(x_i).$$

**Exemple** Faisons le calcul pour l'exemple de la Sect. 8.4. Soit  $x_i$  l'événement que nous sommes à la position  $i$  et  $z$  l'événement que le robot détecte une porte. Initialement,  $p(x_i) = .125$  pour toutes les positions  $x_i$ , et, si le robot se trouve devant une porte, la probabilité que le capteur la détecte correctement est de .9, tandis que la probabilité qu'il détecte incorrectement une porte est de  $1 - .9 = .1$ . La probabilité de détecter une porte,  $p(z)$ , est obtenue en additionnant les probabilités à chaque position, où la probabilité est de  $.125 \text{ fois } .9 = .1125$  à une position avec une porte et  $.125 \text{ fois } .1 = .0125$  à une position sans porte :

$$p(z) = .1125 + .1125 + .0125 + .0125 + .1125 + .1125 + .0125 = .575.$$

Par la règle de Bayes, la probabilité d'être à la position  $i$  avec une porte *si* une porte est détectée est :

$$p(x_i | z) = \frac{p(z | x_i) p(x_i)}{p(z)} = \frac{.9 \times .125}{.575} = .196,$$

tandis que la probabilité d'être à la position  $i$  sans porte, mais incorrectement une porte est détectée est :

$$p(x_i | z) = \frac{p(z | x_i) p(x_i)}{p(z)} = \frac{.1 \times .125}{.575} = .022.$$

## B.2 Normalisation

L'ensemble des probabilités des résultats possibles d'un événement doit être égal à 1 puisque l'un des résultats doit se produire. Si une porte est détectée, le robot doit se trouver à l'une des 8 positions possibles, mais la somme sur toutes les positions  $i$  de la probabilité que le robot se trouve à la position  $i$  a été montrée ci-dessus comme étant :

$$.1125 + .1125 + .0125 + .0125 + .1125 + .1125 + .0125 = .575.$$

Les probabilités doivent être *normalisées* en les divisant par la somme .575 afin que la somme soit de 1. Les probabilités normalisées sont  $.1125/.575 \approx .19$  et  $.0125/.575 \approx .02$ , ce qui donne 1 :

$$.19 + .19 + .02 + .02 + .19 + .19 + .19 + .02 \approx 1.$$

## B.3 Moyenne et variance

La *moyenne*<sup>1</sup>  $\mu$  d'un ensemble de valeurs  $\{x_1, \dots, x_n\}$  est :

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i.$$

---

1. *Moyenne* est le terme technique pour moyenne.

Considérons cinq personnes gagnant respectivement 8, 9, 10, 11, 12 milliers d'euros par an. Leur salaire moyen est de :

$$\mu = \frac{8 + 9 + 10 + 11 + 12}{5} = \frac{50}{5} = 10.$$

La moyenne ne nous dit pas grand-chose car la même moyenne peut être obtenue à partir de données très différentes :

$$\mu = \frac{5 + 6 + 10 + 14 + 15}{5} = \frac{50}{5} = 10.$$

La moyenne est fortement influencée par les *outliers* : les valeurs qui sont beaucoup plus élevées ou plus basses que le reste des valeurs. Si la personne gagnant 10 mille euros a soudainement reçu une prime de 90 mille euros, le salaire moyen est maintenant de :

$$\mu = \frac{8 + 9 + 100 + 11 + 12}{5} = \frac{140}{5} = 28.$$

Un politicien sauterait sur l'occasion pour affirmer que, durant son mandat, le salaire moyen a augmenté de 180 !

*Variance* est une mesure de la dispersion d'un ensemble de valeurs. Plus les valeurs sont proches les unes des autres, plus la variance est faible. Les mesures telles que la moyenne sont plus fiables si les valeurs sont regroupées et ont donc une faible variance. La formule de la variance d'un ensemble de valeurs  $\{x_1, \dots, x_n\}$  est la suivante :<sup>2</sup>

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \mu)^2.$$

Chaque terme  $x_i - \mu$  mesure la distance de la valeur  $x_i$  par rapport à la moyenne ; la variance est la moyenne des carrés de ces distances. Les distances sont élevées au carré afin que les valeurs situées de part et d'autre de la moyenne ne s'annulent pas. Par exemple, pour des valeurs de 100 et 300, la moyenne est de 200 ; si nous calculons la variance comme  $(100 - 200) + (200 - 300)$ , le résultat sera de 0 même si les valeurs sont réparties. En utilisant la définition ci-dessus, la variance est  $(100 - 200)^2 + (200 - 300)^2 = 20000$ .

Pour l'ensemble de données  $\{8, 9, 10, 11, 12\}$  la variance est :

$$s^2 = \frac{(-2)^2 + (-1)^2 + 0 + 1^2 + 2^2}{5-1} = \frac{10}{4} = 2.5,$$

tandis que pour l'ensemble de données  $\{5, 6, 10, 14, 15\}$  la variance est :

$$s^2 = \frac{(-5)^2 + (-4)^2 + 0 + 4^2 + 5^2}{4} = 20.5.$$

---

2.  $n - 1$  compte les *degrés de liberté*. Comme la moyenne est calculée avant la variance, nous ne pouvons pas choisir les  $n$  valeurs arbitrairement ; la dernière valeur choisie est contrainte d'être la valeur qui fait que le calcul produit la moyenne donnée.

Comme 20.5 est beaucoup plus grand que 2.5, les données de la deuxième série sont réparties sur une plage plus large que les données de la première série. Après réception du bonus, la variance est de :

$$s^2 = \frac{20^2 + 19^2 + 72^2 + 17^2 + 16^2}{4} = \frac{6490}{4} = 1622.5.$$

Il est clair qu'il ne faut pas interpréter le salaire moyen comme significatif s'il existe des valeurs aberrantes.

#### B.4 Covariance

Considérons un groupe de dix personnes percevant les salaires suivants :

$$x_1 = \{11, 12, 13, 14, 15, 16, 17, 18, 19, 20\},$$

en milliers d'euros. Le salaire moyen est de :

$$\mu_1 = \frac{1}{10}(11 + 12 + 13 + 14 + 15 + 16 + 17 + 18 + 19 + 20) = 15.5.$$

Nous supposons que les personnes ayant des salaires élevés achètent des voitures plus chères que celles ayant des salaires faibles. Supposons que deux modèles de voitures soient vendus dans cette zone, l'un pour 10 mille euros et l'autre pour 20 mille euros. L'ensemble de données suivant montre les voitures achetées par ce groupe de personnes, où le  $i$ 'th élément est le coût de la voiture achetée par la  $i$ 'th personne :

$$x_2 = \{10, 10, 10, 20, 10, 20, 10, 10, 20, 20\}.$$

Pour voir s'il existe un lien entre les salaires et les coûts des voitures, on calcule la covariance  $cov(x_1, x_2)$  entre les ensembles de données  $x_1$  et  $x_2$ . Le calcul est similaire à celui de la variance, sauf qu'au lieu d'élever au carré la différence entre une valeur d'un seul ensemble et la moyenne de cet ensemble, nous multiplions la différence entre une valeur du premier ensemble et sa moyenne par la différence entre une valeur du deuxième ensemble et sa moyenne :

$$cov(x_1, x_2) = \frac{1}{n-1} \sum_{i=1}^n (x_{1,i} - \mu_1)(x_{2,i} - \mu_2).$$

La covariance des ensembles de valeurs  $x_1$  et  $x_2$  est de 7,8, une valeur positive qui indique que les salaires et le coût des voitures augmentent ensemble, c'est-à-dire que les personnes qui gagnent plus d'argent ont tendance à acheter des voitures plus chères. Si les cinq premières personnes achètent des voitures d'une valeur de 10 et les cinq suivantes des voitures d'une valeur de 20, la covariance devient 13,9, ce qui indique un lien plus fort entre le salaire et le coût d'une voiture. Inversement, si

les cinq premiers achètent des voitures chères et les cinq suivants des voitures bon marché, la covariance est de  $-13,9$ , ce qui signifie que plus le salaire augmente, plus le coût d'une voiture diminue. Enfin, si tout le monde achète la même voiture, la covariance est de 0 et nous concluons, comme prévu, qu'il n'y a pas de lien entre le salaire et la voiture achetée.

La covariance est symétrique car la multiplication des nombres réels est commutative :

$$\begin{aligned} \text{cov}(x_1, x_2) &= \frac{1}{n-1} \sum_{i=1}^n (x_{1,i} - \mu_1)(x_{2,i} - \mu_2) \\ &= \frac{1}{n-1} \sum_{i=1}^n (x_{2,i} - \mu_2)(x_{1,i} - \mu_1) \\ &= \text{cov}(x_2, x_1). \end{aligned}$$

La matrice de covariance combine les variances et les covariances :

$$\begin{bmatrix} s^2(x_1) & \text{cov}(x_1, x_2) \\ \text{cov}(x_2, x_1) & s^2(x_2) \end{bmatrix}.$$

$\text{cov}(x_1, x_2) = \text{cov}(x_2, x_1)$ , il n'y a donc que trois valeurs différentes dans la matrice.

## B.5 Multiplication des vecteurs et des matrices

La multiplication d'une matrice (bidimensionnelle)  $\vec{M}$  par un vecteur  $\vec{v}$  donne un nouveau vecteur :

$$\vec{M}\vec{v} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}.$$

La multiplication de deux matrices s'effectue en multipliant les lignes de la matrice gauche séparément avec chaque vecteur colonne de la matrice droite pour obtenir les vecteurs colonnes de la matrice résultante :

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x & u \\ y & v \end{bmatrix} = \begin{bmatrix} ax + by & au + bv \\ cx + dy & cu + dv \end{bmatrix}.$$

La matrice identité est :

$$\vec{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

et il est facile de vérifier que pour toute matrice  $\vec{M}$ ,  $\vec{M} \vec{I} = \vec{I} \vec{M} = \vec{M}$ . Pour une matrice  $\vec{M}$ , son inverse  $\vec{M}^{-1}$  est la matrice qui donne  $\vec{I}$  lorsqu'elle est multipliée par  $\vec{M}$  :

$$\vec{M} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad \vec{M}^{-1} = \frac{1}{\det(\vec{M})} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix},$$

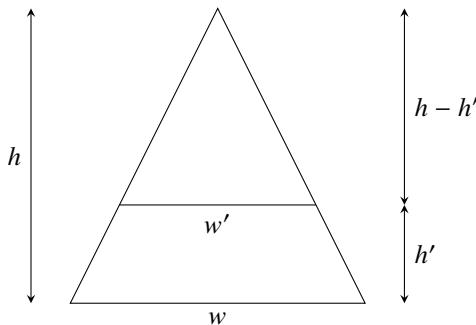
où  $\det(\vec{M})$ , le *déterminant* de  $\vec{M}$ , est  $ad - bc$ . On peut le vérifier en multipliant :

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = \begin{bmatrix} ad - bc & -ab + ba \\ cd - dc & -bc + da \end{bmatrix} = \begin{bmatrix} ad - bc & 0 \\ 0 & ad - bc \end{bmatrix}.$$

Ceci n'est valable que pour les matrices dont le déterminant est non nul, car les matrices *singular* - celles dont le déterminant est nul - n'ont pas d'inverse.

## B.6 L'aire d'un trapèze à la base d'un triangle

Le diagramme suivant montre un triangle de largeur  $w$  et de hauteur  $h$  avec une ligne parallèle à la hauteur  $h'$  qui crée un trapèze :



On veut trouver une formule pour l'aire du trapèze en utilisant les valeurs  $w, h, h'$ . L'aire  $a$  est la différence entre les aires des deux triangles :

$$a = \frac{wh}{2} - \frac{w'(h-h')}{2}.$$

Par triangles semblables :

$$\frac{h}{h-h'} = \frac{w}{w'},$$

donc :

$$w' = \frac{w(h-h')}{h}.$$

Substitution :

$$\begin{aligned}
 a &= \frac{wh}{2} - \frac{w(h-h')(h-h')}{2h} \\
 &= \frac{w(h^2 - (h-h')^2)}{2h} \\
 &= \frac{w(h^2 - h^2 + 2hh' - h'^2)}{2h} \\
 &= \frac{w(2hh' - h'^2)}{2h} \\
 &= wh'\left(1 - \frac{h'}{2h}\right).
 \end{aligned}$$

### B.7 Formules algébriques pour $\cos 15^\circ$

Eq. 16.1 affirme que :

$$\cos^{-1}\left(\frac{\sqrt{2+\sqrt{3}}}{2}\right) = \pm 15^\circ.$$

En utilisant la formule du cosinus de la différence de deux angles, nous avons :

$$\begin{aligned}
 \cos 15^\circ &= \cos(45^\circ - 30^\circ) \\
 &= \cos 45^\circ \cos 30^\circ + \sin 45^\circ \sin 30^\circ \\
 &= \frac{\sqrt{2}}{2} \cdot \frac{1}{2} + \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{3}}{2} \\
 &= \frac{\sqrt{2} + \sqrt{6}}{4}.
 \end{aligned}$$

Nous calculons maintenant :

$$\left(\frac{\sqrt{2} + \sqrt{6}}{4}\right)^2 = \left(\frac{8 + 2\sqrt{2}\sqrt{6}}{16}\right) = \frac{2 + \sqrt{3}}{4} = \left(\frac{\sqrt{2+\sqrt{3}}}{2}\right)^2.$$



## Bibliographie

- [1] Karl Johan Åström and Richard M. Murray. *Feedback Systems : An Introduction for Scientists and Engineers*. Princeton University Press, 2008. The draft of a second edition is available online at [http://www.cds.caltech.edu/~murray/amwiki/index.php/Second\\_Edition](http://www.cds.caltech.edu/~murray/amwiki/index.php/Second_Edition).
- [2] Harold Abelson and Andrea diSessa. *Turtle Geometry : The Computer as a Medium for Exploring Mathematics*. MIT Press, 1986.
- [3] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- [4] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping : Part ii. *IEEE Robotics & Automation Magazine*, 13(3) :108–117, 2006.
- [5] M. Ben-Ari. *Principles of Concurrent and Distributed Programming (Second Edition)*. Addison-Wesley, 2006.
- [6] Michael Bodi, Ronald Thenius, Martina Szopek, Thomas Schmickl, and Karl Crailsheim. Interaction of robot swarms using the honeybee-inspired control algorithm beeclust. *Mathematical and Computer Modelling of Dynamical Systems*, 18(1) :87–100, 2012.
- [7] Valentino Braatenberg. *Vehicles : Experiments in Synthetic Psychology*. MIT Press, 1984.
- [8] J. Chen, M. Gauci, W. Li, A. Kolling, and R. Groß. Occlusion-based cooperative transport with a swarm of miniature mobile robots. *IEEE Transactions on Robotics*, 31(2) :307–321, 2015.
- [9] Jianing Chen, Melvin Gauci, and Roderich Groß. A strategy for transporting tall objects with a swarm of miniature mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 863–869, 2013.
- [10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (Third Edition)*. MIT Press, 2009.
- [11] Nikolaus Correll. *Introduction to Autonomous Robots*. CreateSpace, 2014. Available for download at <https://github.com/correll/Introduction-to-Autonomous-Robots/releases/download/v1.9/book.pdf>.
- [12] John J. Craig. *Introduction to Robotics : Mechanics and Control, Third Edition*. Pearson, 2005.
- [13] Erol Şahin and William M. Spears, editors. *Swarm Robotics : SAB 2004 International Workshop*, chapter Swarm Robotics : From Sources of Inspiration to Domains of Application, pages 10–20. Springer, 2005.

- [14] Gregory Dudek and Michael Jenkin. *Computational Principles of Mobile Robotics (Second Edition)*. Cambridge University Press, 2010.
- [15] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping : Part i. *IEEE Robotics & Automation Magazine*, 13(2) :99–110, 2006.
- [16] H.R. Everett. *Sensors for Mobile Robots*. A.K. Peters, 1995.
- [17] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (Third Edition)*. Pearson, 2008.
- [18] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4) :31–43, 2010.
- [19] Peter Harrington. *Machine learning in action*, volume 5. Manning Greenwich, CT, 2012.
- [20] Simon O. Haykin. *Neural Networks and Learning Machines (Third Edition)*. Pearson, 2008.
- [21] Suzana Herculano-Houzel. The human brain in numbers : a linearly scaled-up primate brain. *Frontiers in human neuroscience*, 3 :31, 2009.
- [22] David W. Hogg, Fred Martin, and Mitchel Resnick. Braitenberg creatures. Technical Report E&L Memo No. 13, MIT Media Lab, 1991. [http://cosmo.nyu.edu/hogg/lego/braitenberg\\_vehicles.pdf](http://cosmo.nyu.edu/hogg/lego/braitenberg_vehicles.pdf).
- [23] John Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (Third Edition)*. Pearson, 2007.
- [24] Auke Jan Ijspeert, Alcherio Martinoli, Aude Billard, and Luca Maria Gambardella. Collaboration through the exploitation of local interactions in autonomous collective robotics : The stick pulling experiment. *Autonomous Robots*, 11(2) :149–171, 2001.
- [25] Alan Julian Izenman. *Modern multivariate statistical techniques*. Springer, 2008.
- [26] Serge Kernbach. *Handbook of collective robotics : Fundamentals and challenges*. CRC Press, 2013.
- [27] A.D. King. Inertial navigation—forty years of evolution. *GEC Review*, 13(3) :140—149, 1998. [http://www.imar-navigation.de/downloads/papers/inertial\\_navigation\\_introduction.pdf](http://www.imar-navigation.de/downloads/papers/inertial_navigation_introduction.pdf).
- [28] David Kriesel. A brief introduction to neural networks. [http://www.dkriesel.com/en/science/neural\\_networks](http://www.dkriesel.com/en/science/neural_networks), 2007.
- [29] Miroslav Kubat. *An introduction to machine learning*. Springer, 2015.
- [30] Deepak Kumar. *Learning Computing with Robots*. Lulu, 2011. Download from [http://calicoproject.org/Learning\\_Computing\\_With\\_Robots](http://calicoproject.org/Learning_Computing_With_Robots).

- [31] Jean-Claude Latombe. *Robot Motion Planning*. Springer, 1991.
- [32] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.
- [33] Ralf Mayet, Jonathan Roberz, Thomas Schmickl, and Karl Crailsheim. Ant-bots : A feasible visual emulation of pheromone trails for swarm robots. In Marco Dorigo, Mauro Birattari, Gianni A. Di Caro, René Doursat, Andries P. Engelbrecht, Dario Floreano, Luca Maria Gambardella, Roderich Groß, Erol Şahin, Hiroki Sayama, and Thomas Stützle, editors, *Swarm Intelligence : 7th International Conference, ANTS 2010, Brussels, Belgium, September 8-10, 2010. Proceedings*, pages 84–94. Springer Berlin Heidelberg, 2010.
- [34] Oxford Technical Solutions. What is an (INS) inertial navigation system ? <http://www.oxts.com/what-is-inertial-navigation-guide/>.
- [35] Kevin M. Passino and Stephen Yurkovich. *Fuzzy Control*. Addison-Wesley, 1998.
- [36] Richard E. Pattis, Jim Roberts, and Mark Stehlík. *Karel the Robot : A Gentle Introduction to the Art of Programming (Second Edition)*. Wiley, 1995.
- [37] Raúl Rojas. *Neural Networks : A Systematic Introduction*. Springer, 1996.
- [38] John C. Russ. *The Image Processing Handbook (Sixth Edition)*. CRC Press, 2011.
- [39] Stuart Russell and Peter Norvig. *Artificial Intelligence : A Modern Approach (Third Edition)*. Pearson, 2009.
- [40] Thomas Schmickl, Stjepan Bogdan, Luís Correia, Serge Kernbach, Francesco Mondada, Michael Bodi, Alexey Gribovskiy, Sibylle Hahshold, Damjan Miklic, Martina Szopek, et al. Assisi : mixing animals with robots in a hybrid society. In *Conference on Biomimetic and Biohybrid Systems*, pages 441–443. Springer, 2013.
- [41] Thomas Schmickl, Ronald Thenius, Christoph Moeslinger, Gerald Radspieler, Serge Kernbach, Marc Szymanski, and Karl Crailsheim. Get in touch : Cooperative decision making based on robot-to-robot collisions. *Autonomous Agents and Multi-Agent Systems*, 18(1) :133–155, 2009.
- [42] Jiwon Shin, Roland Siegwart, and Stéphane Magnenat. Visual programming language for Thymio II robot. In *Proc. of the 2014 Conference on Interaction Design and Children (IDC)*, 2014.
- [43] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots (Second Edition)*. MIT Press, 2011.
- [44] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. Wiley, 2005.
- [45] Richard Szeliski. *Computer Vision : Algorithms and Applications*. Springer, 2011.

- [46] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.
- [47] James J. Trobaugh and Mannie Lowe. *Winning LEGO MINDSTORMS Programming*. Apress, 2012.
- [48] John Vince. *Quaternions for Computer Graphics*. Springer, 2011.
- [49] Shiling Wang, Francis Colas, Ming Liu, Francesco Mondada, and Stéphane Magnenat. Localization of inexpensive robots with low-bandwidth sensors. In *Distributed Autonomous Robotic Systems (DARS)*. IEEE, 2016.
- [50] Alan Winfield. *Robotics : A Very Short Introduction*. Oxford University Press, 2012.
- [51] B. Yamauchi. A frontier-based approach for autonomous exploration. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 146–151, 1997.
- [52] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3) :338—353, 1965.

# Index

- A\* algorithm, 214
- accuracy, 53
- actionneur, 107
- activity
  - A\* algorithme, 219
  - Acceleration, 89
  - Accuracy, 53
  - algorithme BeeClust, 316
  - Algorithme de Dijkstra pour les cartes continues, 214
  - Algorithme de Dijkstra sur une carte quadrillée, 211
  - algorithme de frontière, 188
  - Algorithme de Pledge, 147
  - Amélioration d'image : lissage, 236
  - Amélioration d'image : manipulation de l'histogramme, 238
  - Angles d'Euler distincts, 352
  - Angles d'Euler multiples, 351
  - ANN avec mémoire, 259
  - ANN pour l'attraction des obstacles, 256
  - ANN pour l'évitement d'obstacles : conception, 255
  - ANN pour l'évitement d'obstacles : mise en œuvre, 256
  - ANN pour le filtrage spatial, 262
  - ANNs multicouches, 258
  - Apprentissage hébraïque pour l'évitement d'obstacles, 270
  - Apprentissage par un perceptron, 306
  - Artificial neurons for logic gates, 252
  - Attractive et répulsive, 61
  - Caméléon robotique, 279
  - Caméléon robotique avec LDA, 294
  - Carte probabiliste des obstacles, 181
  - Changement de vitesse, 88
  - Cinématique inverse, 332
  - Cinématique vers l'avant, 328
  - Codage de roue, 102
  - Combinaison de la planification de la trajectoire et de l'évitement des obstacles, 220
  - Computer la distance en accélérant, 91
  - Configuration du capteur, 68
  - Consistants, 78
  - Contrôleur désactivé, 130
  - Contrôleur PI, 136
  - Contrôleur PID, 137
  - Correction des erreurs d'odométrie, 101
  - Différentes configurations de ligne, 68
  - Distance d'un capteur de distance, 46
  - Distance à partir de la vitesse et du temps, 93
  - Dogged, 61
  - Dogged (stop), 61
  - Driven, 64
  - Définir la période de contrôle, 127
  - Déetecter un coin, 244
  - Détection d'un blob, 246
  - Détection d'un bord, 241
  - Détection des zones de haute densité, 152
  - Détermination de la position par triangulation, 164
  - Détermination de la position à partir d'un angle et d'une distance, 163
  - Effet combiné des erreurs d'odométrie, 100
  - Evitement d'obstacles avec deux cap-

- teurs, 294
- Expressions conditionnelles pour suivre un mur, 142
- Force de traction de plusieurs robots., 319
- Force totale, 322
- Grue robotique, 115
- Grue robotique (alternatives), 115
- Indécisif, 61
- Insecure, 64
- Jouer avec le jeu des repères, 162
- La présentation des véhicules par Braitenberg, 74
- Ligne suiveuse avec un capteur, 69
- Line following with proportional correction, 70
- Linéarité, 54
- Localisation avec incertitude dans le mouvement, 174
- Localisation avec incertitude dans les capteurs, 173
- Localiser le nid, 152
- Localiser le robot à partir des perceptions calculées, 199
- Localiser le robot à partir des perceptions mesurées, 201
- Logique floue, 226
- Matrices de rotation, 335
- Mesure de l'attitude à l'aide d'accéléromètres, 51
- Mesure du mouvement à accélération constante, 91
- Mouvement holonomique et non-holonomique, 120
- Multilayer ANN for obstacle avoidance, 258
- Neurones artificiels analogiques, 252
- Odométrie en deux dimensions, 97
- Ordres d'odométrie, 100
- Paranoïde, 63
- Paranoïde (droite-gauche), 63
- Paranoïde (sens alterné), 82
- Persistant, 82
- poussée basée sur l'occlusion, 323
- Précision et résolution, 52
- Recherche et approche, 82
- Reconnaitre une porte, 246
- Reprendre la ligne après l'avoir perdue, 68
- Robot qui peut seulement tourner, 108
- Réflectivité, 47
- Régulateur proportionnel, 134
- Simple suivi de mur, 144
- Suivi de ligne avec deux capteurs, 66
- Suivi de ligne circulaire lors de la lecture d'un code, 149
- Suivi de ligne en pratique, 72
- Suivi de ligne sans gradient, 71
- Suivi du mur avec direction, 146
- Suivre un objet, 295
- Suivre une ligne tout en lisant un code, 148
- Thresholds, 47
- Timid, 60
- Tondeuse à gazon robotisée, 191
- transformations homogènes, 343
- Triangulation, 48
- Vélocité sur une distance fixe, 87
- algorithm
  - Algorithme de Dijkstra sur une carte quadrillée, 207
  - Algorithme de frontière, 187
  - Analyse discriminante linéaire (phase d'apprentissage), 289
  - Analyse discriminante linéaire (phase de reconnaissance), 290
  - ANN for obstacle avoidance, 267
  - Application de la règle de Hebbian, 269
  - Classification par un perceptron (phase

- d'apprentissage), 302

Classification par un perceptron (phase de reconnaissance), 303

Contrôleur désactivé, 129

Contrôleur proportionnel, 131

Contrôleur proportionnel-intégral, 135

Contrôleur proportionnel-intégral-différentiel, 138

Distinguer les classes (phase d'apprentissage), 280

Distinguer les classes (phase de reconnaissance), 280

Détection d'un blob, 247

Feedback on the robot's behavior, 269

Multiplication des nombres entiers, 30

Paranoïde, 62

Persistant, 81

Recherche et approche, 83

Schéma de l'algorithme de contrôle, 126

Simple suivi du mur, 144

SLAM, 203

Suivi de ligne avec deux capteurs, 67

Suivi de ligne avec un capteur, 70

Suivi des murs, 146

Timid, 59

Timid avec while, 60

Algorithme control-on-off, 128

algorithme control, 127

algorithme de classification, 272

Algorithme de Dijkstra, 205

grille avec coûts variables, 209

Algorithme de Dijkstra sur une carte quadrillée à coût constant, 206

algorithme de localisation et de cartographie simultanées (SLAM), 202

algorithme de Markov, 167

algorithmes d'Euler nombre de, 352

angles d'Euler, 349

aperture, 71

apprentissage machine, 272

analyse discriminante linéaire, 281

discriminant basé sur les moyennes, 274

cadres de coordonnées, 333

calibration, 53

caméra, 48

Capteur linéaire, 54

ultrasonique, 39

capteur de proximité, 40

distance, 39

exteroceptif, 38

infrarouge, 40

laser, 45

optique, 41

proprioceptif, 38

sol, 27

tactile, 50

triangulation, 43

capteurs de proximité, 26

cartographie algorithme frontière, 183

avec connaissance de l'environnement, 189

encodage, 179

exploration de l'environnement, 181

graphique de visibilité, 213

classification des robots, 13

Codeur de roue, 101

comportement réactif, 58

control, 123

- closed loop, 124
- integrator windup, 136
- open loop, 124
- controler
  - algorithme
  - proportionnel, 130
- Contrôleur
  - algorithme
  - proportionnel-intégral, 134
  - proportionnel-intégral-derivé, 137
- degré de liberté, 107
- degré de mobilité, 116
- diagrammes d'états, 76
- Dijkstra's algorithm
  - continuous map, 211
- Direction d'Ackermann, 25
- discriminant, 274
- dispositif à couplage de charge (CCD), 48
- effecteur final, 326
- environnement de développement logiciel, 21
- event handler, 28
- fuzzy logic
  - linguistic variable, 223
- gain, 130
- gamme, 51
- inertial navigation system, 103
  - accelerometer, 103
- interpolation, 56
- Karel le robot, 312
- kinématique, 325
- ligne suivante, 64
- lignes de repère, 161
- localisation
  - angle et distance, 162
  - probabiliste, 166
  - triangulation, 164
- localisation et cartographie simultanées (SLAM)
  - exemple numérique, 192
- logique floue, 222
  - conséquence, 223
  - règle, 223
- machine learning
  - discriminant based on the means and the variances, 277
  - linear discriminant analysis example, 288
- machine à états finis, 76
  - nondéterminisme, 80
  - état final, 78
- map
  - continuous, 178
  - grid, 178
- mapping, 177
- matrice
  - rotation
  - tridimensionnelle, 346
- matrice de rotation
  - bidimensionnelle, 334
- microelectromechanical systems, 103
- motorisation différentielle, 24
- mouvement holonomique, 116, 118
- multiplication matricielle, 362
- mur suivant, 143
- navigation
  - basée sur la carte, 205
- neural network
  - Hebbian rule, 264
  - learning in, 263
  - multilayer topology, 256
  - reinforcement learning, 264
- neurone, 249
- nonlinearity, 53

- odométrie, 91
  - linéaire, 93
  - avec tours, 94
- path following and obstacle avoidance, 219
- perceptron, 297
  - classification, 300
  - learning, 301
- pixel, 48
- polling, 29
- pose, 94
- precision, 52
- probabilité
  - conditionnelle, 357
  - conjointe, 357
  - covariance, 360
  - moyenne, 359
  - normalisation, 358
  - règle de Bayes, 358
  - variance, 359
- probabilité d'occupation de la carte, 182
- pseudocode, 29
- Période de l'algorithme de contrôle, 126
- quaternions, 353
- rayon de virage, 62
- Recherche de chemin
  - modèle probabiliste des fourmis, 153
- recherche de chemin
  - fourmis, 149, 157
- resolution, 52
- robot
  - generic, 22
  - humanoid, 19
  - industriel, 15
  - mobile, 17
  - éducatif, 19
- robotique en essaim, 308
  - algorithme BeeClust, 313
  - architecture distribuée, 309
- collaboration physique, 317
- combiner les forces, 317
- communications à l'aide d'objets, 311
- par communications électroniques, 311
- Mise en œuvre de l'algorithme Bee-Clust par ASSISIbf, 314
- poussée collective basée sur l'occlusion, 320
- par échange d'informations, 310
- rotation, 333
  - d'un repère de coordonnées, 336
  - transformation d'un vecteur d'un re-  
père de coordonnées à un autre, 337
  - tridimensionnelle, 343, 352
  - d'un vecteur, 333
- roues suédoises, 117
- règle de la main droite, 344
- réseau neuronal, 249
  - apprentissage non supervisé, 264
  - apprentissage supervisé, 263
  - artificiel, 251
  - biologique, 249
  - filtre spatial, 260
  - mise en œuvre du véhicule de Brai-  
tenberg, 253
  - avec mémoire, 259
  - topologie, 256
- sensor
  - accelerometer, 50
  - microphone, 50
- simultaneous localization and mapping  
(SLAM), 177
  - fermer la boucle, 189
  - overlap, 190
- suivi de mur
  - algorithme de gage, 147
  - avec direction, 144

suivre une ligne avec un code, 148  
système de navigation inertielles  
gyroscope, 104  
système de positionnement global, 165  
systèmes redondants, 111  
temps écoulé du capteur, 43  
threshold, 28, 47  
timer, 28  
traitement d'image  
filtre de boîte, 234  
filtre de Sobel, 240  
filtre pondéré, 235  
segmentation, 232  
traitement d'images  
détection des coins, 242  
détection des contours, 239  
filtre spatial, 233  
manipulation d'histogrammes, 237  
optique, 229  
traitement de l'image  
amélioration, 231, 232  
traitement des images, 228  
couleur, 230  
résolution, 229  
transformations homogènes, 341  
trapézoïde, 363  
triangulation, 164  
unités de mesure, 355  
variance, 277  
vitesse, 86  
instantanée, 90  
Véhicule Braitenberg, 58  
dogged, 61  
driven, 64  
indécis, 61  
insecure, 64  
paranoïaque, 63, 82  
persistant, 77  
timid, 59

Véhicule de Braitenberg  
Mise en œuvre d'un ANN, 253  
Véhicule de Braitenberg  
attractif et répulsif, 61  
consistent, 78  
paranoid, 62  
réseau neuronal, 253  
évitement d'obstacles, 253  
workspace, 329  
évitement des obstacles, 142