INSTITUT UNIVERSITAIRE DE TECHNOLOGIE DE LA ROCHELLE
Département Informatique

Stage de fin d'études

Romain Douteau

Spécialité Informatique Embarquée

Développeur robotique

Développement d'un programme de navigation pour un robot.

Période de stage du 09/04/2018 au 15/06/2018

Beuth-Hochschule für Technik Berlin

Luxemburger Straße 10, 13353 Berlin, Allemagne

Enseignant tuteur : M. DOUCET Antoine

Maître de stage : M. DÖPKENS Andreas

Non confidentiel

Année Universitaire 2017-2018

INSTITUT UNIVERSITAIRE DE TECHNOLOGIE DE LA ROCHELLE
Département Informatique

Stage de fin d'études

Romain Douteau

Spécialité Informatique Embarquée

Développeur robotique

Développement d'un programme de navigation pour un robot.

Période de stage du 09/04/2018 au 15/06/2018

Beuth-Hochschule für Technik Berlin

Luxemburger Straße 10, 13353 Berlin, Allemagne

Enseignant tuteur : M. DOUCET Antoine

Maître de stage : M. DÖPKENS Andreas

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Non confidentiel

Année Universitaire 2017-2018

Remerciements

J'aimerais adresser mes sincères remerciements à M. Antoine DOUCET chargé des relations internationales à l'IUT[1] de La Rochelle et également mon professeur référent pour mon stage, de m'avoir permis d'effectuer ce stage à l'étranger.

Je remercie également les membres du SRI[2] de la Rochelle pour l'aide qu'ils ont pu m'apporter en ce qui concerne la préparation de ce stage à l'étranger.

Ensuite, je remercie l'université allemande Beuth-Hochschule für Technik de Berlin pour l'accueil au sein de leur structure en fin d'année scolaire et de l'aide qu'ils m'ont apportés concernant les démarches administratives sur place.

Je me dois également de remercier M. Andreas DÖPKENS mon maître de stage pour la confiance et la liberté qu'il m'a accordé tout au long de ces 10 semaines. En effet, son aide en ce qui concerne la compréhension des différents phénomènes physiques liés au domaine de la robotique ainsi que dans la conception des différentes phases de mon projet m'a permis d'avancer dans mon travail de façon conséquente.

De plus, je souhaite adresser ma reconnaissance à M. Brian SCHÜLER pour ses conseils en ce qui concerne la réalisation et l'optimisation de différents programmes.

Enfin, je remercie les autres collègues et étudiants du laboratoire PSE-Lab[3] pour leur accueil et leur bonne humeur.

[1] IUT : Institut Universitaire Technologie
[2] SRI : Service des Relations Internationales
[3] PSE-Lab : Laboratory for Persasive Sytems and Engineering

Sommaire

Introduction

Le monde de l'entreprise se complexifiant de jour en jour, il est désormais nécessaire de développer une expérience professionnelle pratique en plus d'une formation théorique. C'est pourquoi pour valider mes deux années de formation au sein de l'IUT de la Rochelle, j'ai effectué un stage d'une durée de 10 semaines du 9 avril 2018 au 15 juin 2018.

Ayant eu la possibilité d'effectuer ce stage au sein de l'université de Beuth-Hochschule für Technik à Berlin en Allemagne et étant passionné de voyage, c'est tout naturellement que j'ai accueilli ce choix positivement. J'ai donc été affecté à l'équipe du laboratoire de recherche en robotique PSE-Lab du département informatique de l'université de Beuth Hochschule für Technik de Berlin.

Ce stage constitue donc pour moi une opportunité de découvrir un secteur prometteur en lien avec ma spécialité tout en mettant mes compétences acquises dans les domaines techniques et linguistiques au profit de la résolution d'un problème concret dans une situation professionnelle.

L'objectif de ma mission est d'aider les membres du laboratoire à faire évoluer le programme acaBot[4] permettant aux lycéens et étudiants de découvrir le secteur de la robotique. Pour ce faire, j'ai été affecté au projet Wall Follower s'inscrivant dans la liste des nombreux projets du laboratoire. La réalisation de ce projet devra permettre de démontrer comment un robot peut-il naviguer tout en adaptant son comportement en fonction des différents éléments qu'il rencontre dans son environnement.

Dans les prochaines pages de ce rapport, je détaillerai les éléments m'ayant permis de répondre à cette problématique. Dans un premier temps, je présenterai le contexte de mon stage, l'entreprise d'accueil, mon rôle au sein du laboratoire, le matériel mis à ma disposition ainsi que mon organisation tout au long de ces 10 semaines. Ensuite, je détaillerais le travail réalisé de la phase d'analyse du sujet jusqu'à la réalisation en passant par la conception. J'appuierai ma réflexion sur des schémas théoriques, des résultats d'expériences ainsi que d'autres travaux scientifiques réalisés au préalable. Dans une troisième partie, je parlerai de la partie gestion de projet ayant guidé ce stage suivi d'un point sur l'avancement de mon projet puis, je détaillerai les différents problèmes auxquels j'ai dû faire face. Enfin, dans une dernière partie, je présenterai les apports de ce stage d'un point de vue personnel en expliquant ce qui m'a permis d'évoluer personnellement et professionnellement au cours de ces 10 semaines. Je ferai finalement un lien entre le contenu de ce stage et les enseignements dispensés au cours de ma formation m'ayant été utiles à la mise en œuvre d'un projet dans un contexte professionnel.

[4] acaBot : Academic Robots

Partie 1 : Contexte du stage

Dans cette première partie, nous allons découvrir le contexte général de ce stage. D'abord, je vous présenterai la structure d'accueil de mon stage suivi de mon environnement de travail. Ensuite, je parlerai de mon organisation tout au long de ces 10 semaines. Enfin, je détaillerai les objectifs de ma mission puis les outils mis à ma disposition pour la mener à bien.

1.1 - Université de Beuth Hochschule

L'université Beuth Hochschule für Technik se situe dans le district de Wedding de la ville de Berlin en Allemagne. Cette université publique est l'une des plus grandes d'Allemagne ; divisée en huit départements répartis sur cinq campus, elle propose neufs parcours d'études scientifiques possibles. Elle accueille chaque année plus de 12 000 étudiants encadrés par 290 professeurs. La politique de l'Université de Beuth est d'offrir une chance de réussite professionnelle égale à tous les étudiants quel que soit le milieu social duquel ils sont issus. Pour offrir des conditions optimales d'apprentissages, l'université permet à ses étudiants de travailler sur des projets concrets en partenariats avec plus de 90 entreprises. L'université, tend à se faire connaitre au fil des années à l'internationale grâce à la mise en place de plus de 130 partenariats avec d'autres écoles à travers le monde.

Figure 1 : Devanture de l'Université de Beuth.

Figure 2 : Campus principal de l'université de Beuth.

1.2 - Laboratoire PSE-Lab

Le PSE-Lab est un laboratoire de recherche dans lequel travaillent chercheurs et étudiants sur divers projets dans le domaine de la robotique. On peut y trouver de nombreux types de robots expérimentaux chacun conçu pour remplir une tâche bien définie. Dans le laboratoire, on trouve également plusieurs tables avec différentes configurations permettant de tester le fonctionnement d'algorithmes implémentés sur les différents robots.

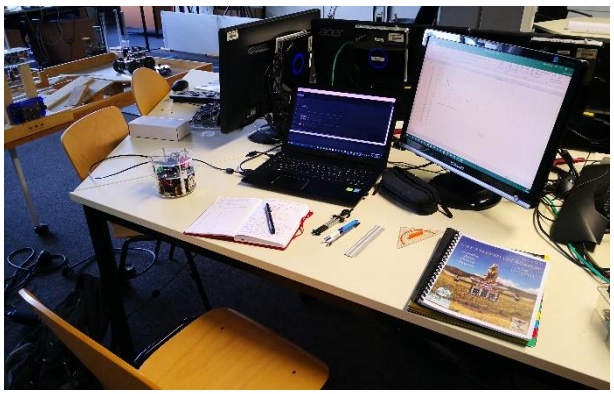

Figure 3 : Mon poste de travail au sein du laboratoire.

Figure 4 : Entrée du laboratoire.

Dès l'entrée, on trouve du matériel scientifique servant à étudier ou modéliser le fonctionnement de différents systèmes. On peut également y voir divers robots entreposés dans une étagère sur le côté droit ainsi que tous les systèmes permettant de recharger les batteries des différents robots. Plus loin, on trouve les postes de travail informatique des étudiants situés sur le côté gauche ainsi que les postes de travail informatique de M. Andreas DÖPKENS et M. Brian SCHÜLER tous deux chercheurs en robotique. Le laboratoire dispose également de postes de travail permettant d'effectuer des opérations de maintenance sur les robots.

Globalement, le laboratoire est un espace de travail assez encombré bien qu'il soit possible de circuler librement. En effet tous les systèmes robotiques et postes de travail occupent une grande partie de la surface disponible. De plus, les murs sont recouverts d'affiches concernant les projets robotiques mis en place par l'équipe du laboratoire ainsi que les étudiants. Cependant, le cadre de travail est assez calme et l'équipe ainsi que les étudiants toujours de bonne humeur.

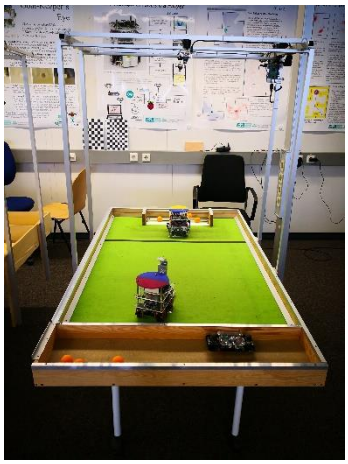

Figure 5 : Système de robots tireur et gardien de but.

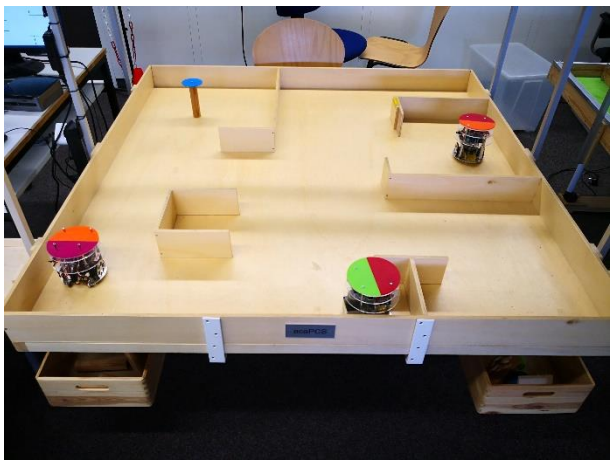

Figure 6 : Système de robots solveurs de labyrinthes.

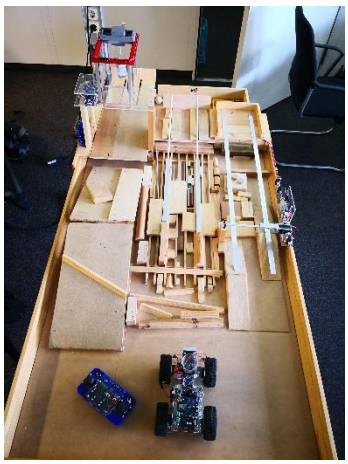

Figure 7 : Système de robots testeurs de parcours.

1.3 - L'équipe

Au sein du laboratoire, travaillent deux personnes à plein temps. M. Andreas DÖPKENS chercheur en robotique (également mon maître de stage pour ces 10 semaines), se charge de concevoir et mettre en place les divers systèmes robotique. Pour résumer, son métier consiste à designer les différents robots ainsi que mettre en place les environnements de tests correspondant à chacun d'entre eux. M. Brian SCHÜLER quant à lui est développeur en robotique. Il s'occupe de développer des programmes pour les différents robots. C'est également lui qui s'occupe du parc informatique du laboratoire, il gère la maintenance matérielle et logicielle des différents postes.

D'autres étudiants viennent travailler dans le laboratoire pour leurs études. Ils travaillent sur différents projets tels qu'un système de robot tireur et gardien de but (voir Figure 5), un système de robot solveur de labyrinthe (voir Figure 6) ainsi qu'un système de robot testeur de parcours (voir Figure 7). De façon générale, chaque personne travaille sur son projet ce qui permet à tout le monde d'avoir un espace de travail suffisamment grand pour travailler confortablement.

1.4 - Organisation

Pour ce qui est de l'organisation de mes journées de travail au sein du laboratoire, celles-ci étaient assez flexibles. En fonction des journées, j'arrivais entre 9h et 10h pour repartir entre 16h et 18h pour une durée de travail variant entre 7h et 8h par jour. En outre, une des différences de la culture allemande est qu'ils prennent peu de temps pour se restaurer le midi, je prenais donc une pause le midi entre 13h et 14h pour aller déjeuner avec l'équipe du laboratoire et d'autres étudiants.

Cette particularité de la culture allemande de ne pas s'infliger des horaires de travail fixes permet à mon avis de travailler plus sereinement, d'être plus productif et de rester centré sur ses objectifs tout en éliminant le stress.

1.5 - Sujet du stage

1.5.1 - Mission

Tout au long de ces 10 semaines, ma mission principale consistait à aider l'équipe du laboratoire à faire évoluer le programme acaBot permettant aux lycéens et étudiants de découvrir le secteur de la robotique le temps d'un stage. Ce programme permet d'apprendre les bases de la programmation et de l'électronique par le biais de différents programmes sur des cartes Arduino.

1.5.2 - Projet Wall-Follower

Pour mener à bien ma mission, j'ai été affecté au projet Wall Follower inclu dans le programme acaBot. Ce projet a pour objectif de permettre à un robot de se déplacer dans son environnement en suivant le mur se trouvant à sa droite tout en calculant sa trajectoire en fonction de la situation dans laquelle il se retrouve afin d'éviter de toucher le mur ou de se retrouver bloqué devant un obstacle. Une fois ce travail accompli, je devais créer une application graphique permettant de visualiser les déplacements du robot ainsi que les obstacles rencontrés par ce dernier.

1.6 - Outils utilisés

1.6.1 - Informatique

Un poste informatique (disposition allemande) est mis à ma disposition au sein du laboratoire, mais du fait qu'il me soit possible d'utiliser mon ordinateur portable j'ai préféré cette dernière option, car elle me permet d'avoir tous mes fichiers de travail sur le même poste et de continuer à travailler cher moi.

Pour ce qui est de la connexion à internet, un accès au réseau Wi-Fi du laboratoire m'a été accordé.

Afin de travailler plus efficacement et ne me servant pas de l'ordinateur mis à ma disposition, j'ai demandé l'autorisation d'utiliser l'écran en tant que deuxième écran sur mon ordinateur portable. Cela m'a permis de gagner en productivité.

1.6.2 - Robotique

Tous les robots du laboratoire sont équipés de cartes Arduino ou Raspberry Pi suivant les fonctionnalités qu'ils embarquent. Ces deux architectures accessibles financièrement permettent à la fois de créer des robots expérimentaux fiables et simples à programmer. Le robot mis à ma disposition pour ce stage est un modèle aCTino (voir Annexe 1) fonctionnant grâce à une carte Arduino Léonardo (modèle de la carte Arduino). Ce robot dispose pour se repérer dans l'espace, de trois capteurs de distance infrarouges ainsi que deux moteurs chacun couplés à une roue pour se déplacer. Le tout est alimenté par une batterie de 9V.

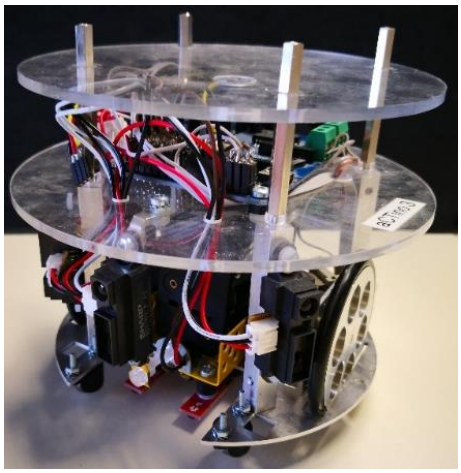

Figure 8 : Robot de modèle aCTino.

1.6.3 - Logiciels

Pour développer un programme destiné à être exécuté sur une carte Arduino, il est nécessaire de télécharger l'IDE Arduino qui intègre à la fois un compilateur et un outil permettant de flasher les cartes Arduino connectées à l'un des ports USB de l'ordinateur par le biais d'un convertisseur USB-Série implémenté directement dans la carte Arduino. Cet IDE est open source et gratuit, il peut donc être téléchargé librement depuis le site officiel d'Arduino.

1.6.4 - Tests

Les tests de fonctionnement, sont particulièrement importants dans le domaine de l'informatique et encore plus dans le secteur de la robotique. En effet, chaque robot avant sa mise en service dans un milieu humain doit être minutieusement testé afin d'éviter tout accident. C'est pourquoi je dispose au sein du laboratoire d'un labyrinthe assez simplifié permettant d'évaluer le comportement de mon robot face à différents cas. La particularité de ce labyrinthe est qu'il dispose d'une configuration modulable permettant de tester le fonctionnement du robot dans presque tous les cas.

Figure 9 : Labyrinthe de test pour le projet Wall Follower

Dans cette première partie, je vous ai présenté la structure d'accueil et l'environnement de travail dans lequel s'est déroulé ce stage de 10 semaines. J'ai également annoncé le sujet et la mission de mon stage. Enfin, nous avons vu les différents outils et moyens mis à ma disposition par le laboratoire afin d'effectuer ma mission dans les meilleures conditions.

Partie 2 : Mission et travail réalisé

Je vais à présent vous parler du travail effectué durant ce stage. Je commencerai par présenter la prise en main des différents systèmes qui constitue un prérequis au développement d'algorithmes sur le robot aCTino. Je parlerai ensuite du travail effectué en ce qui concerne le projet Wall Follower de la phase d'analyse en passant par les phases de recherche d'algorithmes, d'étude des différents cas possibles et phénomènes physiques à prendre en compte dans le développement d'un tel système. Je finirai par développer les différentes phases de la réalisation du projet de la conception à la réalisation sans oublier les tests de fonctionnement.

2.1 - Prise en main de la plateforme Arduino

Mon travail durant la première semaine de ce stage consistait à me familiariser avec la plateforme Arduino ainsi qu'à revoir différents principes du domaine de l'électronique. Du fait de ma spécialité Informatique Embarquée, cela n'était pour moi que de simples révisions mais qui restaient importantes pour la réalisation de ma mission. Pour les 5 premiers jours, un document nommé « Get started on acaBot for interns, a 10-day training program » coécrit par M. Andreas DÖPKENS et M. Brian SCHÜLER a été mis à ma disposition. Ce document, comme son nom l'indique est un programme permettant aux stagiaires de découvrir le secteur de la robotique en 10 jours.

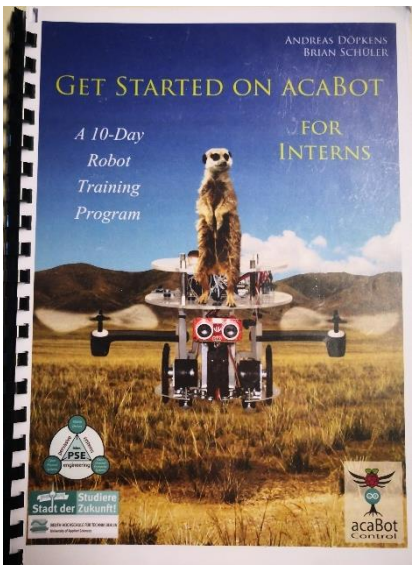

Figure 10 : Couverture du livret "Get stated on acaBot for interns"

A l'intérieur, on trouve en premier lieu, des informations sur le fonctionnement de la plateforme Arduino. On y apprend que Arduino est une plateforme de prototypage open source permettant de créer des systèmes électroniques divers grâce à des cartes électroniques dont les plans sont disponibles librement sur le site officiel. Une carte Arduino possède des entrées et sorties électronique permettant l'interaction entre la carte Arduino ainsi que différents capteurs et actionneurs.

De plus, comme précédemment annoncé, les cartes Arduino sont programmables grâce à l'IDE Arduino également disponible sur le site officiel. L'IDE Arduino se veut simple d'utilisation (voir Annexe 2), mais il reste très complet en laissant aux développeurs la possibilité d'inclure les bibliothèques logicielles dont ils ont besoin de ce qu'ils souhaitent développer. Les cartes Arduino sont programmables en langage C++ grâce à la bibliothèque logicielle Arduino inclue dans l'IDE fournissant une surcouche complète qui simplifie grandement le travail des développeurs. De ce fait, les cartes Arduino sont beaucoup utilisées pour le prototypage, car elles permettent de créer des systèmes simples à prendre en main, fiables et très accessibles financièrement.

2.1.1 - Fonctionnement d'un programme en Arduino

Lorsque l'on développe pour la plateforme Arduino, il est indispensable de comprendre la structure élémentaire d'un programme propre à cette dernière.

Comme on peut le voir sur le programme disponible en annexe (voir Annexe 3), on distingue trois grandes parties dans un programme Arduino.

1. La première est la déclaration des variables qui se fait au début du programme selon les bonnes pratiques du laboratoire. Cela évite de détruire et recréer en permanence les mêmes variables à chaque passage dans la boucle et donc de consommer du temps CPU inutilement.

2. La deuxième partie d'un programme correspond à la définition permettant d'initialiser l'état du système : la fonction *setup*. C'est la première fonction qui sera exécutée (une seule fois) au moment de la mise sous tension de la carte Arduino.

3. La troisième et dernière partie est la définition de la fonction principale du programme : la fonction *loop*. Celle-ci sera exécutée en boucle par le programme à la suite de la fonction *setup*. Ces deux fonctions sont disponibles nativement à la plateforme Arduino, elles n'ont en aucun cas besoin d'être appelées pour fonctionner et leur nom est réservé par le système. Il est possible au besoin d'ajouter des fonctions à un programme Arduino ; il faudra cependant les appeler dans l'une des deux fonctions *setup* ou *loop* pour les utiliser.

2.1.2 - Gestion du temps

Grâce à l'horloge interne des cartes Arduino, il est possible d'implémenter dans chaque programme un algorithme permettant de gérer le temps avec une précision de 1 ms.

L'algorithme de gestion du temps aussi appelé Powerfull Millis-Timer Algorithm (voir Annexe 4) développé par M. Andreas DÖPKENS et M. Brian SCHÜLER répond à ce besoin. Il se base sur la fonction *millis* disponible nativement sur Arduino et qui permet de connaitre le nombre de millisecondes écoulées depuis la mise sous tension du système. Son principe de fonctionnement est simple, le programme récupère la valeur retournée par la fonction *millis* a chaque passage dans la fonction *loop*, si cette valeur est supérieure ou égale à celle de la variable nextEvent correspondant au nombre de millisecondes du prochain évènement depuis la mise sous tension de la carte, le programme effectue les actions contenues dans la boucle principale du timer. Le programme incrémente ensuite la valeur du prochain évènement par celle de la variable timeBase correspondant au nombre de milliseconde d'attente avant le prochain évènement. De ce fait, le programme exécutera les actions contenues dans la boucle du timer à intervalle de temps régulier défini par la variable timeBase. Il est donc important de comprendre qu'il ne faut en aucun cas mettre des actions critiques (devant être effectuées le plus rapidement possible) dans la boucle du timer au risque de compromettre la fiabilité et la sécurité du système.

2.1.3 - Programmation machine à état

La machine à état est un paradigme de programmation visant à représenter un système sous la forme d'une machine pouvant prendre un nombre fini d'état. Une machine à état ne peut être que dans un seul état à un instant précis. Chaque état défini un comportement du système, ce dernier fonctionne donc en passant d'état en état. Chaque changement d'état d'une machine à état est provoqué par un évènement particulier comme la fin d'une période de temps définie ou une donnée physique mesurée par un capteur.

Pour implémenter une machine à état dans un programme Arduino (voir Annexe 5), il est tout d'abord nécessaire de déclarer les différents états que peut prendre le système. Ensuite, il suffit de créer une variable ayant pour valeur l'état actuel du programme. Dans la fonction *setup*, il nous faut initialiser l'état du système avant le premier passage dans la fonction *loop*. Le reste de la machine à état se situe dans la fonction *loop*. Afin de rendre le code du programme plus lisible et donc maintenable, nous allons

définir chaque état de la machine dans une itération de type switch-case ce qui permettra de différencier visuellement les différentes parties du programme pour chaque cas. Si l'on considère que certains états peuvent amener à certains autres, mais que tous ne se succèdent pas nécessairement, il est indispensable de vérifier indépendamment dans chaque état, les conditions de transition vers chaque autre état possible depuis ce premier. Enfin, pour passer d'un état à l'autre une fois la condition de transition validée, il faut tout d'abord opérer les changements nécessaires sur le système avant de modifier la valeur de la variable définissant l'état actuel du système par la valeur du nouvel état.

Pour la suite du projet, chaque programme destiné au robot devra être conçu et implémenté sous la forme d'une machine à état.

2.2 - Prise en main du robot

Lors de ma deuxième semaine de stage, j'ai pu prendre en main le robot mis à ma disposition par mon maître de stage. J'ai dû étudier son fonctionnement par le biais de différents programmes présents à la fin du livret « *Get started on acaBot for interns* » ; cela m'a permis de maitriser la programmation du robot afin de créer des programmes plus complexes sur celui-ci.

2.2.1 - Fonctionnement du robot

Dans le but de comprendre le fonctionnement du robot, je me suis en premier lieu familiarisé avec la bibliothèque logicielle permettant de programmer le robot grâce à l'IDE Arduino. Dans cette bibliothèque, on trouve toutes les différentes variables et fonctions permettant de faire fonctionner le robot.

Plusieurs fonctionnalités utiles que nous allons voir juste après sont implémentées dans cette bibliothèque logicielle.

2.2.1.1 - Déplacement du robot

Le déplacement du robot depuis l'IDE Arduino est rendu possible grâce à la fonction *vOmega* disponible dans la bibliothèque logicielle du robot. Son principe de fonctionnement est simple, elle prend en paramètre deux variables que nous appellerons v et ω. La variable v (exprimée en cm/s) représente la vitesse linéaire du robot et la variable ω (exprimée en degrés/s) représente sa vitesse angulaire. La combinaison de ces deux vitesses permet de gérer le déplacement complet du robot.

De ce fait, il est possible au robot d'aller tout droit, en avant ou en arrière. Cependant, il est dangereux de faire reculer le robot, car il est impossible de détecter d'éventuels obstacles à l'arrière du fait que les capteurs de distance se trouvent à l'avant de ce dernier. Faire reculer un robot de ce type représente donc un risque de créer un accident dans un environnement humain. La bonne pratique dans ce genre de situation est de faire tourner le robot de 180 degrés autour de son centre (sur lui-même) pour faire demi-tour. Cette action prend plus de temps, mais est moins dangereuse. Il est également possible que le robot effectue une rotation autour de son centre dans le sens horaire ou dans le sens antihoraire. Enfin, il est possible de combiner des valeurs positives ou négatives des variables v et ω afin que le robot se déplace en effectuant des courbes comme le montre le tableau suivant.

Tableau 1 : Valeurs des variables v et w en fonction des actions souhaitées.

Action	Vitesse linéaire	Vitesse angulaire
Arrêt	$v = 0$	$\omega = 0$
Rotation dans le sens antihoraire	$v = 0$	$\omega > 0$
Rotation dans le sens horaire	$v = 0$	$\omega < 0$
Avancer tout droit	$v > 0$	$\omega = 0$
Courbe vers la gauche	$v > 0$	$\omega > 0$
Courbe vers la droite	$v > 0$	$\omega < 0$

La fonction *vOmega* constitue une interface entre le l'IDE Arduino et les deux moteurs contrôlant le robot, elle permet de s'affranchir du contrôle séparé des deux moteurs ce qui permet un gain de temps conséquent et une meilleure compréhension du programme. Elle facilite également la lecture du programme ce qui le rend plus maintenable.

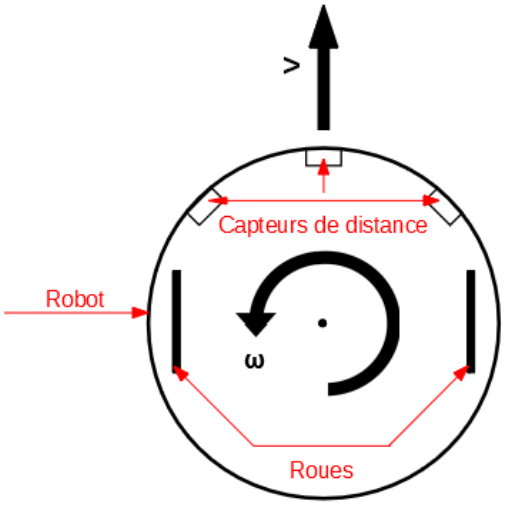

Figure 111 : Schéma explicatif de la fonction vOmega.

2.2.1.2 - Contrôle de la vitesse des moteurs

Le robot, évoluant dans un environnement physique, consomme de l'énergie pour se déplacer. En se déplaçant, il perd également de l'énergie par l'effet du frottement de ses roues sur le sol. De plus, s'il se déplace sur un sol en pente, suivant le sens de celle-ci, si la même quantité d'énergie est fournie aux moteurs, sa vitesse réelle sera différente de celle initialement définie en paramètre de la fonction *vOmega*. S'il veut se déplacer à la bonne vitesse, le robot doit en permanence calculer sa vitesse réelle et réguler l'apport d'énergie dans ses moteurs en conséquence. De cette manière, si la pente est descendante, il faudra ralentir le robot en fournissant moins d'énergie aux moteurs alors que si cette pente est montante, un apport d'énergie plus important sera nécessaire. Il faut donc prendre en compte l'apport d'énergie à fournir pour compenser le frottement des roues du robot sur le sol en plus de la régulation de la vitesse des moteurs en fonction du sens de la pente sur laquelle évolue le robot.

Pour pallier à ce phénomène, chacun des moteurs du robot dispose d'un système d'encodage rotatif. Ce système est composé de deux encodeurs écartés de 90 degrés ainsi qu'une roue d'encodage comportant des segments alternatifs noirs et blancs. Chacun des encodeurs détecte les variations de couleur lors de la rotation des moteurs. Grâce à ce dispositif, il est possible de connaitre le sens et la vitesse de rotation des moteurs pour réguler leur vitesse. De ce fait, un de deux encodeurs de chaque moteur, est connecté à une broche supportant les interruptions externes ; cela permet au robot de comparer en temps réel sa vitesse réelle à sa vitesse définie en comptant le nombre de changements de couleurs de la roue d'encodage en un temps donné. Si le nombre de segments comptés pendant un temps donné est inférieur au nombre de segments correspondant à la vitesse de rotation attendue, le robot est trop lent ; la carte Arduino doit donc fournir plus d'énergie aux moteurs. Si le nombre de segments comptés est supérieur au nombre de segments attendus, le robot est trop rapide ; la carte Arduino doit cette fois-ci fournir moins d'énergie aux moteurs.

2.2.2 - Programmation du robot

Les premiers programmes que j'ai dû réaliser afin de rendre possible les mouvements du robot étaient relativement élémentaires. En effet, dans un premier temps, j'ai dû réaliser un programme permettant au robot d'avancer tout droit sur une longueur de 1 mètre. Dans un deuxième temps, il m'a fallu programmer le robot de façon à ce qu'il effectue un se déplace en formant un cercle de 50 cm de diamètre. Pour terminer, j'ai réalisé un programme permettant au robot d'effectuer un cercle de 50 cm de diamètre avec un pentagramme de même centre à l'intérieur. Pour vérifier la qualité de mon travail, nous avons accroché un stylo au robot puis l'avons disposé sur une grande feuille de papier déployée sur le sol avant de mettre le système sous tension.

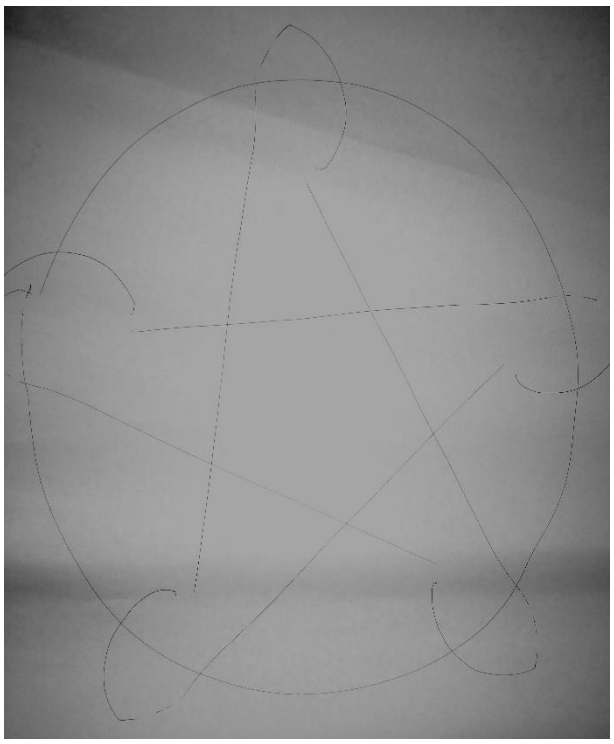

Figure 12 : Dessin du cercle ainsi que du pentagramme effectué par le robot.

Les arcs de cercle non désirés que l'on retrouve sur le dessin sont dû au fait que le stylo soit attaché à l'arrière du robot. Ce dernier ayant un diamètre de 14 cm, lorsqu'il effectue une rotation autour de son centre, on retrouve inévitablement des tracés correspondants à des portions de cercle de même diamètre.

Tous ces programmes m'ont vraiment permis de prendre en main le robot et d'appréhender toutes les possibilités de ce dernier en termes de déplacements. J'ai également pu mettre en pratique tous les concepts vus précédemment ce qui m'a permis d'en savoir plus sur les différents phénomènes à prendre en compte lors de la conception d'un programme d'un tel système.

2.3 - Projet Wall-Follower

Après m'être exercé à la programmation du robot aCTino, j'ai été affecté au projet Wall-Follower qui consiste à programmer le robot afin qu'il suive en permanence le mur situé sur sa droite. Pour ce faire, le robot devra se fier aux données de son capteur de distance droit, ce qui lui permettra de déterminer en temps réel la distance qui le sépare du mur se trouvant sur sa droite. Nous ferons en sorte que le robot se déplace à une vitesse ainsi qu'une distance du mur bien définies.

2.3.1 - Etude de l'existant

Pour mener à bien ce projet, je me suis dans un premier temps inspiré du travail précédent d'un autre stagiaire nommé Jonas effectué en 2014. Ce programme simple (voir Annexe 6) permet au robot de suivre le mur se trouvant à sa droite en lui affectant des vitesses fixes en fonction du cas qu'il rencontre dans son environnement.

Le principe de fonctionnement de ce programme est de définir des vitesses fixes pour chaque cas rencontré par le robot. On définit ensuite la vitesse linéaire (v) du robot à 10 cm/s, sa vitesse de rotation (ω) lorsqu'il arrive à la fin du mur qu'il suit à -70 degrés /s et sa vitesse de rotation lorsqu'il rencontre un obstacle à 130 degrés /s. Pour ce qui est de l'algorithme, le robot commence à avancer tout droit jusqu'à ce qu'il détecte un mur grâce à l'un de ses capteurs de distance. Ensuite, le robot effectue une rotation pour se positionner parallèlement au mur se trouvant à sa droite. Après cela, si le robot ne rencontre aucun obstacle durant le suivi du mur, il avance tout droit ($v = 10, \omega = 0$) ; pour continuer de suivre le mur. Si le robot arrive à la fin du mur, il effectue une courbe dans le sens horaire ($v = 10, \omega = 70$) afin de toujours rester parallèle à celui-ci. Enfin, si un obstacle bloque sa progression, le robot devra effectuer une courbe dans le sens horaire tout en réduisant sa vitesse linéaire ($v = 6, \omega = 130$) afin d'éviter l'obstacle.

Le problème de ce programme est qu'il n'est pas suffisamment précis pour être utilisé sur un robot de ce type. En effet, tous les cas ne sont pas pris en compte et les courbes effectuées par le robot sont approximatives et brusques. On constate également que le robot, lorsqu'il arrive à la fin du mur de droite, vient toucher le mur, chose que l'on veut à tout prix éviter. De plus, le robot se déplace par saccades ce qui, à terme, use ses moteurs prématurément et fragilise sa structure. Enfin, on peut constater qu'il est injustifié d'affecter une vitesse de rotation aussi important que 130 degrés/s au robot sachant que sa vitesse angulaire maximale théorique est de 90 degrés/s.

2.3.2 - Différents cas

Au cours de son évolution dans son environnement, le robot pourra être amené à rencontrer différents cas que nous allons voir en suivant un scénario précis. Chaque cas rencontré fait l'objet d'un état dans la machine à état implémentée dans les programmes du robot.

Lorsque le robot est mis sous tension, à l'initialisation de la carte, il devra se trouver en l'état de recherche d'un mur. De ce fait, il avancera tout droit jusqu'à ce que l'un de ses trois capteurs de distance détecte la présence d'un mur (voir figure 13).

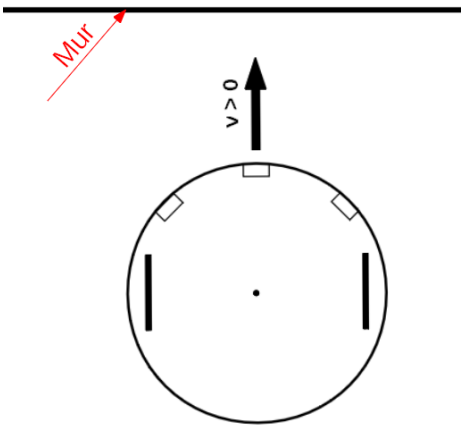

Figure 13 : Schéma explicatif de l'état "Recherche d'un mur".

Ensuite, le robot devra s'arrêter et effectuer une rotation autour de son centre dans le sens antihoraire afin de se positionner parallèlement au mur se trouvant à sa droite (voir figure 14).

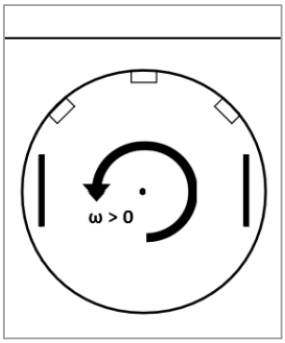

Figure 14 : Schéma explicatif de l'état "Tourner le robot sur lui-même".

Après cela, le robot devra passer à l'état lui permettant de suivre le mur se trouvant à sa droite en avançant tout droit (voir figure 15).

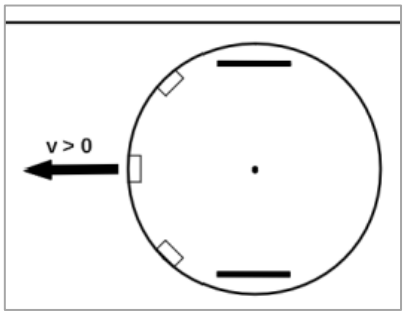

Figure 15 : Schéma explicatif de l'état "Suivre le mur de droite".

Si lors de son évolution, le robot rencontre un mur en face lui, il devra pivoter de 90 degrés dans le sens antihoraire afin d'éviter le mur (voir figure 16). Il devra de plus se trouver à la même distance du mur avant et après avoir pivoté.

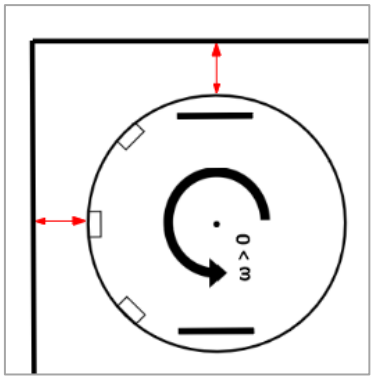

Figure 16 : Schéma explicatif de l'état "Tourner à gauche (sens antihoraire)".

Si le robot arrive à la fin du mur de droite qu'il suivait, il devra dans ce cas précis avancer plus loin que la fin du mur (voir figure 17) afin de l'éviter puis pivoter dans le sens horaire de 90 degrés (voir figure 18) avant d'avancer tout droit pour continuer à suivre le mur de droite (voir figure 19).

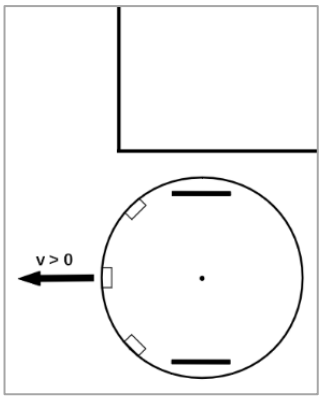

Figure 17 : Schéma explicatif de l'état "Avancer tout droit après la fin d'un mur"

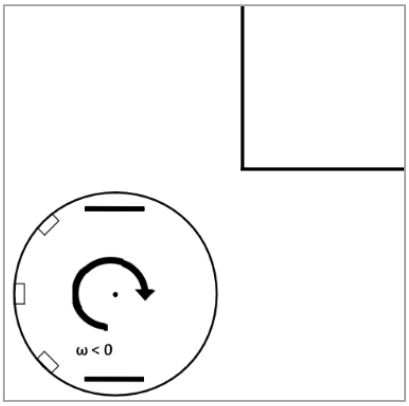

Figure 18 : Schéma explicatif de l'état "Tourner à droite (sens horaire)"

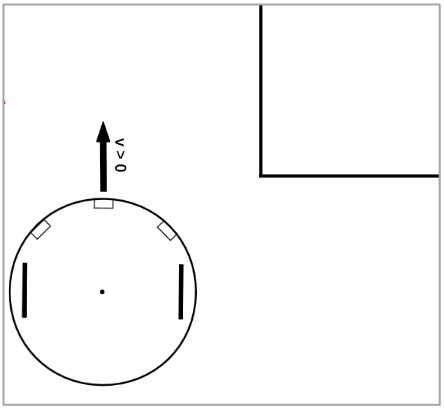

Figure 19 : Schéma explicatif de l'état "Suivre le mir de droite"

Si après cette première rotation aucun mur ne se trouve à sa droite, le robot devra alors avancer tout droit pendant une période définie (voir figure 20), pivoter une seconde fois de 90 degrés dans le sens horaire afin de faire demi-tour (voir figure 21) pour à nouveau suivre le mur se trouvant à sa droite (voir figure 22).

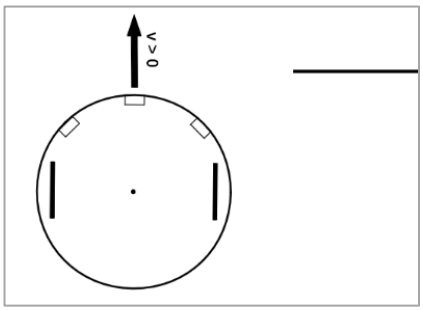

Figure 20 : Schéma explicatif de l'état "Avancer tout droit pendant une période définie"

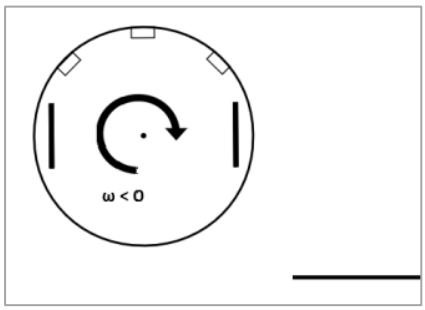

Figure 21 : Schéma explicatif de l'état "Faire demi-tour (sens antihoraire)"

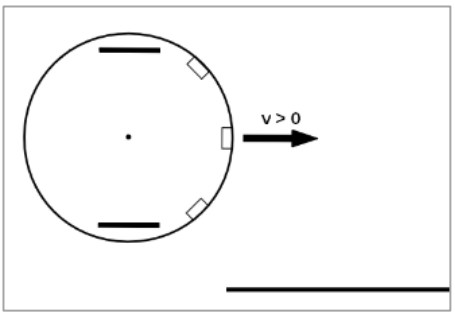

Figure 22 : Schéma explicatif de l'état "Suivre le mur de droite"

Il existe deux autres cas que le robot pourrait être amené à rencontrer. Le premier est le cul-de-sac. En effet, si par hasard le robot se trouve à un moment donné dans un cul-de-sac, ce dernier devra d'abord s'arrêter avant le mur se trouvant en face lui pour ensuite pivoter de 180 degrés dans le sens horaire afin de faire demi-tour (voir figure 23).

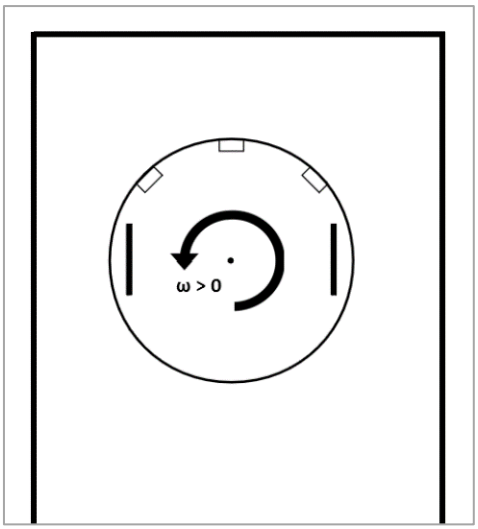

Figure 23 : Schéma explicatif de l'état "Cul-de-sac"

Le second cas est celui ou le robot est en train de tourner autour d'un même obstacle ; cela se repère au nombre de fois à la suite ou le robot tourne à droite. En effet, si le robot tourne plus de trois fois à droite, il est certain qu'il est en train de tourner en rond autour d'un obstacle. Afin d'éviter cela, le robot devra au lieu de tourner une quatrième fois à droite, prendre une trajectoire linéaire et se mettre à nouveau à l'état de recherche d'un mur (voir figure 24).

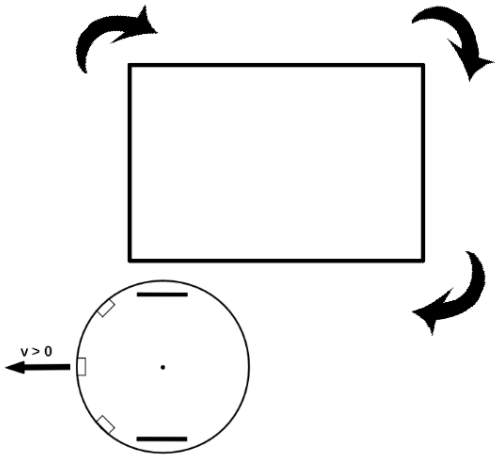

Figure 24 : Schéma explicatif de l'état "Tourner autour d'un obstacle"

2.3.3 - Recherche d'algorithmes

Après avoir défini tous ces cas, il était nécessaire de rechercher des algorithmes existants pour répondre au besoin définit précédemment. Cependant, après plusieurs jours de recherche, j'ai décidé de stopper celles-ci ne trouvant aucun algorithme s'appliquant à une telle situation. De plus, le robot aCTino à des particularités que d'autres n'ont pas, ce qui a rendu mes recherches beaucoup plus difficiles en plus du fait que l'algorithme devait remplir un but très précis, à savoir suivre le mur de droite. J'ai donc décidé, sur les conseils de mon maître de stage, de concevoir moi-même un algorithme permettant au robot de naviguer en suivant le mur se trouvant sur sa droite. De plus, créer un algorithme de la sorte était dans

mes capacités, cela représentait donc, au vu du résultat de mes recherches, un gain de temps mais potentiellement plus de travail de réflexion préalable.

2.3.4 - Calibration des capteurs de distance infrarouge

Afin de garantir une fiabilité optimale du robot, il est nécessaire de calibrer les capteurs de distance infrarouge. La courbe des valeurs des tensions de sortie des capteurs en fonction de la distance à laquelle ils se trouvent de l'objet qu'ils détectent est disponible en annexe (voir Annexe 7). Sur cette courbe, on constate que le capteur de distance infrarouge est utilisable pour mesurer des distances allant de 3 cm à 40 cm.

Les capteurs, étant reliés aux entrées analogiques de la carte Arduino, doivent fournir une tension de sortie comprise entre 0 V et 5V. Or, on constate que les tensions de sortie fournies sur la courbe, dans la documentation technique du capteur, sont comprises entre 0 V et 3,3 V. Il est donc possible de relier directement les capteurs de distance sur la carte Arduino. La carte Arduino, convertira automatiquement les valeurs de tension d'entrées en signal numérique sur 10 bits à l'aide d'un ADC, ce qui nous permettra d'exploiter ces valeurs dans nos futurs programmes.

Chaque capteur, comme n'importe quel instrument de mesure, a besoin d'être calibré afin d'éviter d'utiliser des valeurs erronées dans notre programme ; il est donc nécessaire de procéder à leur calibration en suivant le protocole suivant :

1- Placer le capteur (robot) dans un endroit fixe et dégager l'espace à l'avant de ce capteur sur au moins 40 cm.

2- Placer une règle graduée droite devant le capteur à partir de celui-ci.

3- Implanter un programme dans la carte Arduino permettant d'afficher la valeur numérique correspondant à la tension de sortie (convertie sur 10 bits) du capteur de distance infrarouge.

4- Prendre plusieurs points de mesure en plaçant un objet plat et détectable par le capteur en veillant à chaque fois à noter la distance à laquelle se trouve l'objet du capteur ainsi que la valeur numérique associée renvoyée par le capteur.

5- Entrer les valeurs sur un tableur et créer un nuage de points en faisant correspondre les valeurs numériques obtenues à la distance associée à chaque valeur en cm.

6- Déduire du nuage de points la courbe de tendance associée ainsi que l'équation de cette courbe (on sait que c'est une hyperbole donc un polynôme de degré 3).

7- Implémenter l'équation de la courbe dans le programme du robot.

Grâce à nos mesures, nous avons déduis l'équation de la courbe de tendance des capteurs de distance du robot :

On pose :

x : la valeur de la tension de sortie du capteur de distance convertie en valeur numérique (sur 10 bits).

y : la distance (en cm) séparant le capteur de distance de l'objet qu'il détecte.

On obtient :

$$y = -1E-07x^3 + 0{,}0003x^2 - 0{,}1787x + 45{,}707$$

Cependant, les recherches préalables de M. Andreas DÖPKENS et M. Brian SCHÜLER ont permis de définir une équation plus précise. Cette équation est la suivante :

$$y = \frac{2076}{0.545x - 11} - 1$$

Pour le reste du projet Wall-Follower, nous utiliserons donc cette seconde équation bien qu'elle ne corresponde pas à mes propres résultats.

2.3.5 - Contrôle de la trajectoire en ligne droite

Lorsque le robot se déplace en ligne droite en suivant le mur se trouvant sur sa droite, l'expérience montre que le robot à parfois tendance à s'approcher ou s'éloigner un peu trop du mur. Il est donc nécessaire de corriger en temps réel la trajectoire de ce dernier. Pour ce faire, il faut en permanence mesurer la distance qui sépare le robot du mur se trouvant à sa droite et modifier légèrement la vitesse angulaire de ce dernier de façon à ce que si le robot s'approche trop du mur, sa vitesse angulaire soit positive afin qu'il pivote dans le sens antihoraire pour s'éloigner. Si en revanche le robot venait à se trouver trop loin du mur, il sera alors nécessaire d'affecter une valeur négative à sa vitesse angulaire pour qu'il se rapproche à nouveau du mur de droite.

Le capteur de distance situé sur la droite du robot est celui qui permettra à ce dernier de mesurer la distance de ce dernier par rapport au mur dans le but de la corriger. Cependant, le capteur du côté droit est incliné de 45 degrés sur la droite par rapport au capteur central. Or nous souhaitons que le robot se déplace à une distance précise correspondant à la longueur séparant le bord du mur de l'extrémité droite du robot. Le capteur de distance droit étant incliné de 45 degrés par rapport au mur lorsque le robot est parallèle à ce dernier, nous fournira une valeur différente de celle attendue. Il est possible de faire le lien entre cette la distance mesurée par le capteur droit et la distance attendue entre le robot et le mur par une simple formule trigonométrique.

On pose :

x : la distance attendue entre le robot et le mur de droite.

y : la distance attendue entre le capteur droit et le mur.

On obtient :

$$y = \frac{x}{\cos\frac{\pi}{4}}$$

Il nous faut ensuite utiliser cette valeur dans un algorithme de correction de trajectoire du robot.

On pose :

z : la distance mesurée par le capteur droit.

k : coefficient multiplicateur.

$$\omega = (y - z) * k$$

L'algorithme de correction de la trajectoire du robot disponible en annexe (voir Annexe 8) se base sur l'équation ci-dessus. De plus, l'algorithme de correction de trajectoire est bridé, c'est-à-dire que la valeur la vitesse angulaire pour cet état du robot sera nécessairement supérieure à -30 degrés/s et inférieur à 30 degrés/s cela permet de rendre plus fluides les mouvements du robot. La valeur du coefficient a été choisie empiriquement suite à de nombreux essais effectués. De ce fait, la correction de la trajectoire est presque imperceptible ce qui est plus agréable pour la structure du robot mais également plus économe en énergie, car le robot n'a pas à compenser sans arrêt ses propres mouvements s'ils sont trop brusques.

2.3.6 - Protocole de test

Le labyrinthe de test mis à ma disposition pour ce projet à la particularité d'être modulable c'est-à-dire qu'il est possible d'ajouter ou de retirer, au besoin, certains éléments en fonction du type de robot le parcourant. Pour ce qui est de notre robot, étant donné la configuration de ses capteurs, il nous sera nécessaire d'ajouter un morceau de bois permettant d'allonger un des coins du labyrinthe. En effet, sans cet ajout, le robot ne serait pas capable de détecter ce coin du labyrinthe, car ce dernier est trop court.

Un des autres avantages du labyrinthe est qu'il contient tous les cas précédemment évoqués, ce qui permet au robot de mettre en pratique tous ces cas. De plus, pour valider un programme, celui-ci doit être testé sur le robot qui doit alors être capable de parcourir tout le labyrinthe sans échec au moins trois fois de suite.

Différentes versions du programme de navigation ont donc été mises en place ; je vais, à présent, vous présenter trois versions du programme de navigation qui me paraissent intéressantes.

2.3.7 - Navigation du robot (angles droits)

Tout d'abord, pour bien me rendre compte des éventuels problèmes et obstacles pouvant être rencontrés par le robot, je devais commencer par concevoir et développer un programme permettant à ce dernier de se déplacer en suivant le mur de droite (but du projet Wall Follower) de façon à ce que le robot, chaque fois qu'il rencontre un obstacle ou qu'il est confronté à un des cas précédemment décris, effectue une rotation autour de son centre pour changer de direction.

2.3.7.1 - Conception

Pour la conception de cette version du programme Pour le projet Wall-Follower, j'ai tenu compte de tous les différents précédemment annoncés auxquels pouvait être confronté le robot.

Le robot devant être programmé sous la forme d'une machine à état, l'outil le plus adéquat était le diagramme machine à état. Voici donc le diagramme machine à état du programme permettant au robot de se déplacer en effectuant des angles droits pour changer de direction.

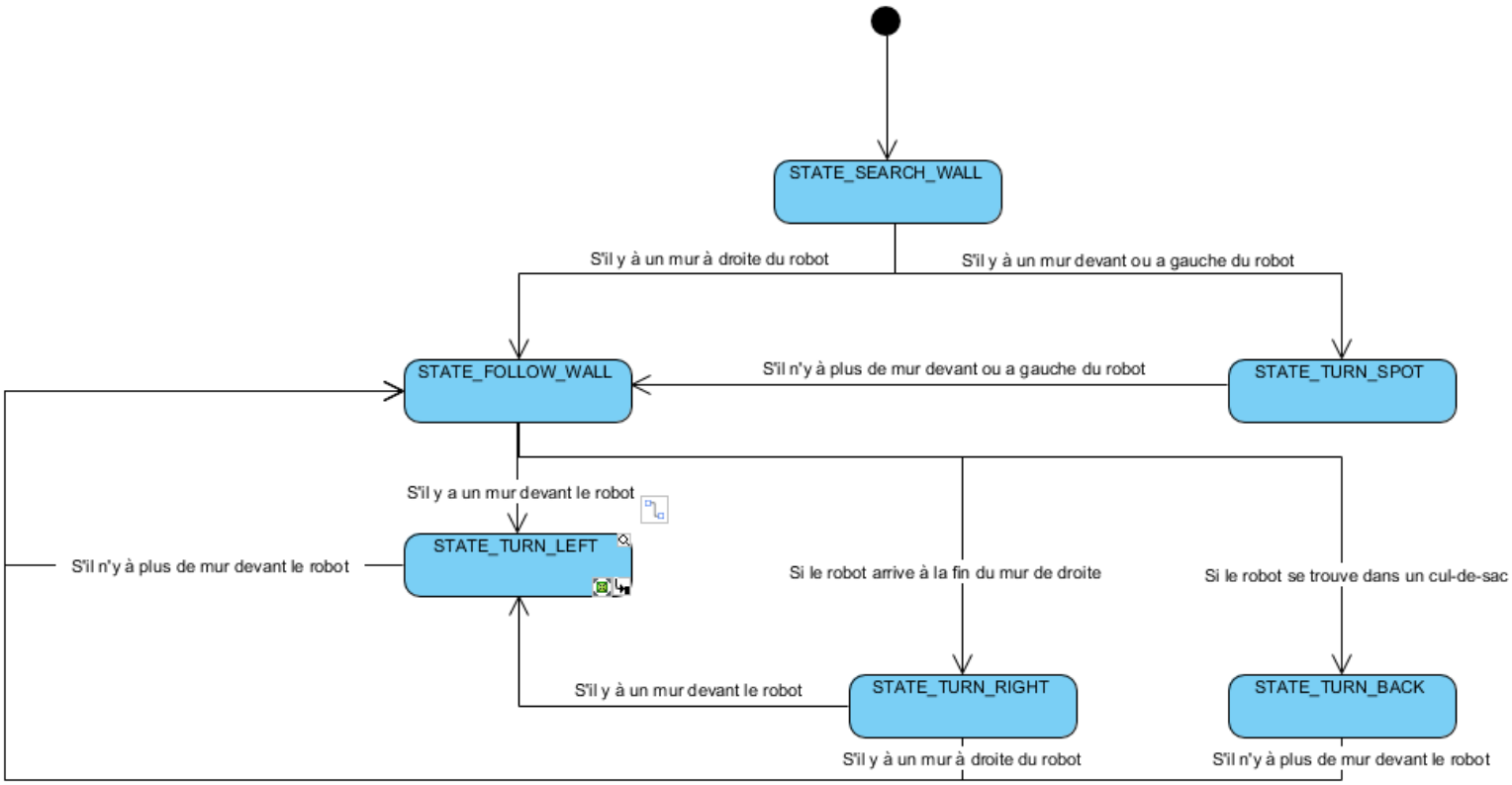

Figure 252 : Diagramme machine à état du programme Wall-Follower

2.3.7.2 - Implémentation

Afin d'implémenter le programme correspondant au diagramme machine à état précédent, il était nécessaire de déclarer les variables suivantes :

- **nextEvent**: Instant de début du prochain événement à partir du début du programme (en ms)
- **timeBase**: Durée entre la fin d'un événement et l'événement suivant (en ms)
- **robotState**: État actuel de la machine d'état
- **STATE_SEARCH_WALL**: Recherche d'un mur
- **STATE_FOLLOW_WALL** : Suivre le mur à droite du robot
- **STATE_TURN_SPOT** : Tournez le robot autour de son centre
- **STATE_TURN_LEFT**: Tourner à gauche
- **STATE_TURN_RIGHT**: Tourner à droite
- **STATE_TURN_BACK**: Faire demi-tour
- **distWallSet**: Distance attendue entre le robot et le mur (en cm)
- **distWallAngleSet**: Distance attendue entre le robot et le mur pour le capteur de distance droit (en cm)
- **botForward_V :** Vitesse linéaire du robot pendant le suivi du mur (en cm/s)
- **botOmega**: Vitesse angulaire du robot (lors de la rotation)
- **kp**: Coefficient d'ajustement du correcteur de trajectoire
- **distanceL**: Valeur du capteur de distance gauche (en cm)
- **distanceM**: Valeur du capteur de distance central (en cm)
- **distanceR**: Valeur du capteur de distance droit (en cm)
- **omega**: Vitesse angulaire du robot après correction de la trajectoire (lors du suivi du mur)
- **curveRightCounter**: Comptez le nombre de virages à droite à la suite.
- **stateMachineEntries**: Compter le nombre d'entrées dans la machine à états.

De plus, nous utiliserons les fonctions suivantes :

- **setup()** : Fonction d'initialisation du système
- **loop()** : Fonction principale du système
- **mesureDistance()** : Cette fonction permet de récupérer les données des capteurs de distance infrarouge du robot. Les données des trois capteurs de distance (gauche, milieu et droite) sont stockées et converties en cm dans les variables distanceL, distanceM et distanceR respectivement.

- **pController()** : Cette fonction ne doit être utilisée que lorsque le robot est à l'état *STATE_FOLLOW_WALL*. Elle permet au robot de mesurer et de corriger en temps réel la distance entre lui et le mur se trouvant à sa droite. Cette fonction permet de recalculer la trajectoire du robot en modifiant en permanence la valeur de la variable oméga.

2.3.7.2 - Tests et conclusion

Les résultats obtenus lors du test du programme sont plutôt concluants et montrent que cette première version du programme, permet au robot de se déplacer dans le labyrinthe de test de façon plus précise que le programme existant. En effet, on constate que le robot reste à la bonne distance du mur grâce à son algorithme de correction de trajectoire en ligne droite. De plus, ce dernier ne vient plus toucher le mur du labyrinthe ce qui permet de le définir comme programme fonctionnel pour ce robot.

2.3.8 - Navigation du robot (quarts de cercles)

La deuxième version fonctionnelle du projet Wall-Follower consiste à développer un programme permettant au robot de suivre le mur se trouvant à sa droite cette fois en effectuant des courbes correspondant à des quarts de cercle afin d'éviter les obstacles. L'objectif est de rendre le robot plus rapide en ne le faisant pas s'arrêter chaque fois qu'il rencontre un cas différent. Cette deuxième version, rend également les mouvements du robot plus fluides et économise l'énergie des batteries du fait qu'il ne soit plus nécessaire d'arrêter le robot pour le remettre en mouvement une fois la rotation effectuée.

2.3.8.1 - Conception

Dans cette version améliorée du programme précédent, nous distinguerons deux types de virage ayant chacun leurs caractéristiques. Le virage à gauche et le virage à droite.

2.3.8.1.1 - Virage à gauche

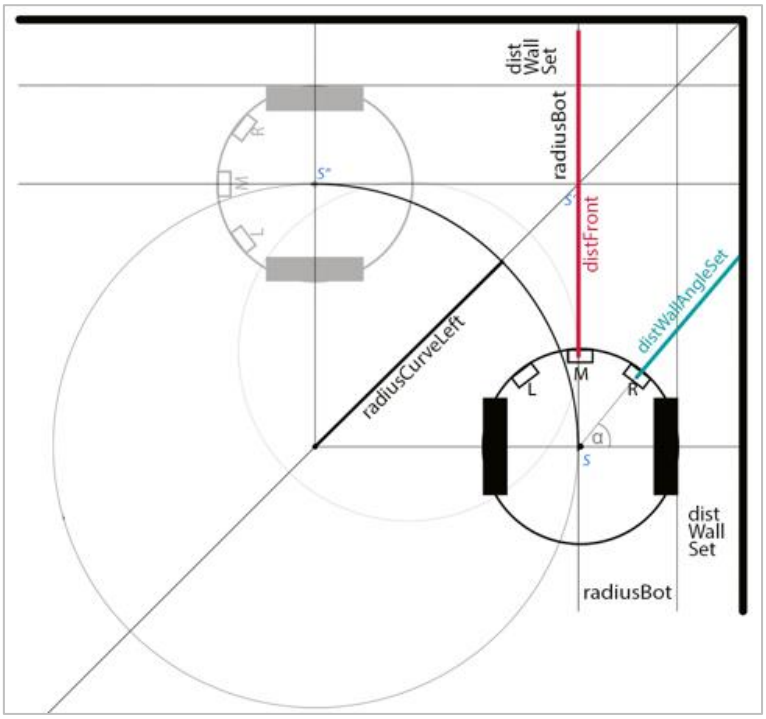

Figure 26 : Schéma explicatif du virage en quart de cercle à gauche

Comme on peut le voir sur la figure ci-dessus (voir figure 13), il nous faut choisir un rayon de virage (radiusCurveLeft) sachant que la valeur minimale de ce rayon est de 0 cm. Dans ce cas, les points S et S'' sont confondus et le robot se déplace jusqu'au point S, s'arrête, tourne sur lui-même de 90 degrés

vers la gauche, puis continue tout droit pour continuer à suivre le mur de droite. Or, l'intérêt de cette version du programme est de faire déplacer le robot de façon à ce qu'il effectue un virage correspondant à un quart de cercle, il nous faut donc choisir une valeur plus élevée pour le rayon du virage. Dans la figure ci-dessus, le rayon est deux fois plus grand que le rayon du robot. La taille du rayon doit être choisie de manière raisonnable et permettre au robot d'explorer tout le labyrinthe sans se bloquer dans une partie du labyrinthe. Nous avons donc dû effectuer des tests de fonctionnement avec différentes valeurs de rayon de virage afin de déterminer le meilleur rayon de virage possible pour un robot de ce type.

En fonction du rayon de virage choisis et la position du robot dans le labyrinthe, le virage à gauche démarrera au moment où le capteur de distance infrarouge situé au milieu de celui-ci affichera une valeur inférieure ou égale à la valeur de la variable *distFront*.

Sachant que :

$$distFront = radiusCurveLeft + ROBOT\_RADIUS + distWallSet - DISTANCE\_SENSOR\_OFFSET$$

2.3.8.1.2 - Virage à droite

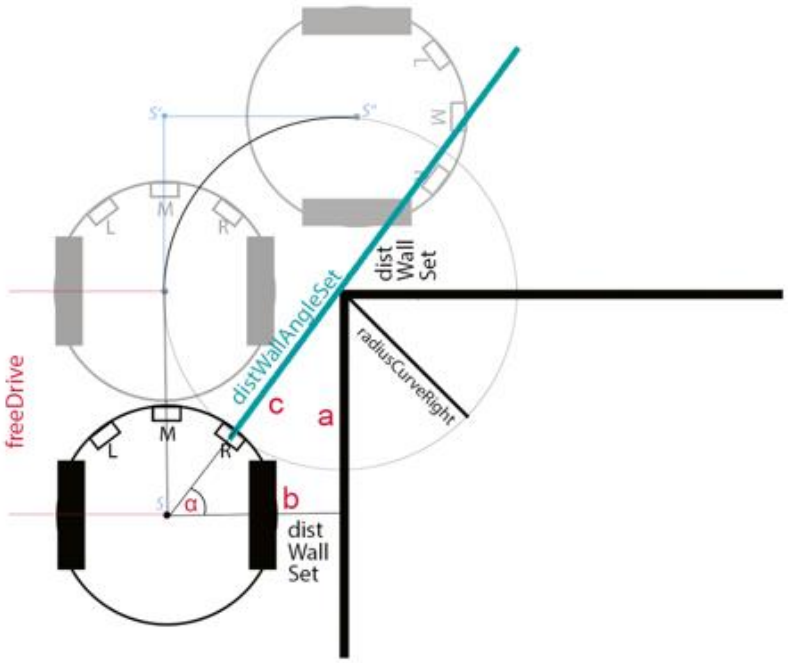

Figure 27 : Schéma explicatif du virage en quart de cercle à droite

Grâce à la figure ci-dessus, on constate qu'il n'est, dans ce cas pas possible de faire varier le rayon (*radiusCurveRight*) de la courbe du virage à droite étant donné que le robot doit se trouver à égale distance du mur de droite avant et après le virage. La seule valeur possible pour le rayon de virage correspond à la somme du rayon du robot (*ROBOT_RADIUS*) et de la distance définie entre le robot et le mur (*distWallSet*).

De ce fait :

$$radiusCurveRight = ROBOT\_RADIUS + distWallSet$$

Dans cette version du programme, le robot devra calculer et adapter sa vitesse à chaque situation rencontrée. Pour ce qui est de la machine à état du programme, celle-ci est la même que pour le programme précédent. En effet, il est important de comprendre que la structure du programme pour cette deuxième version de l'algorithme de déplacement du robot est sensiblement la même que celle de la version précédente. La seule différence entre les deux versions est que dans cette deuxième version, le robot devra se déplacer en formant des courbes correspondant à des quarts de cercles de rayons bien définis. Le diagramme machine-à-état de la version précédente s'applique également à cette version du programme.

2.3.8.2 - Implémentation

Pour implémenter cette deuxième version du programme, nous avons ajoutés les variables suivantes au programme précédent :

leftCurveRadius : Rayon du virage à gauche (en cm)

radiusCurveRight : Rayon du virage à droite (en cm)

freeDrive : Distance à parcourir avant d'effectuer le virage à droite (en cm)

distFront : Distance à partir de laquelle le robot détectera le mur situé devant lui

botLeft_V : Vitesse linéaire du robot durant le virage à gauche (en cm/s)

botRight_V : Vitesse linéaire du robot durant le virage à droite (en cm/s)

freeDrive_V : Vitesse linéaire du robot dans l'étape précédant le virage à droite (en cm/s)

ROBOT_RADIUS : Rayon du robot (en cm)

DISTANCE_SENSOR_OFFSET : Distance entre le centre du robot et le capteur de distance droit (en cm)

2.3.8.3 - Tests et conclusion

Après le test du robot dans le labyrinthe et quelques ajustements au niveau des valeurs de certaines variables, on constate que le programme fonctionne parfaitement. Le robot se dirige comme prévu à l'étape de conception. Ce programme, étant une amélioration du précédent, permet au robot de se déplacer plus efficacement dans le labyrinthe du fait qu'il n'a pas à s'arrêter à chaque obstacle.

2.3.9 - Navigation du robot (clothoïdes)

La neuvième semaine de ce stage, après avoir effectué la majeure partie du projet Wall-Follower, mon maître de stage, m'a suggéré une nouvelle possibilité quant à l'algorithme de parcours du robot. Cette version permet au robot de se déplacer en effectuant des courbes dynamiques en évitant au robot de changer de direction d'un seul coup. Cette version du projet Wall-Follower se base sur l'implémentation des clothoïdes ; elles permettent de modifier la direction du robot de façon progressive. Les clothoïdes permettant de sécuriser les changements de direction dans le domaine des transports à grande vitesse comme les autoroutes ou encore les voies de chemin de fer et offrent également un plus grand confort aux utilisateurs en ne leur faisant pas subir des changements brusques de direction qui se traduisent par l'ajout d'une force attirant tout corps en mouvement vers l'extérieur du virage : la force centrifuge. Les clothoïdes sont donc le meilleur moyen possible connu à ce jour pour protéger la structure matérielle des corps en mouvement dans le domaine des transports.

2.3.9.1 - Conception

Une clothoïde est une courbe ou la courbure est proportionnelle à la longueur de cette même courbe. De ce fait, la courbure augmente linéairement avec la longueur de l'arc de la courbe.

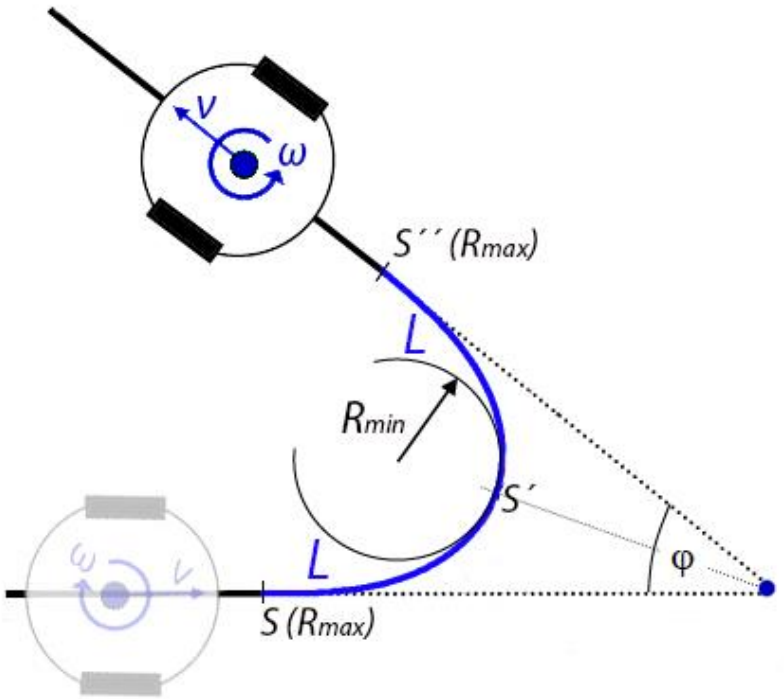

Figure 283 : Schéma explicatif du déplacement du robot en forme de clothoïde

Au début de la courbe (S), la valeur de la courbure est nulle, le robot se déplace donc en ligne droite. C'est à cet endroit que le rayon de la courbe du robot est le plus grand (*Rmax*). Ensuite, le rayon de la courbe tend à se rétrécir pour atteindre sa valeur minimale (*Rmin*) au point S'. Ensuite, du point S' jusqu'au point S", l'effet inverse se produit et le rayon de la courbe augmente de nouveau pour atteindre sa valeur maximale (*Rmax*) au point S". Ce processus crée une courbe lisse connue sous le nom de spirale d'Euler.

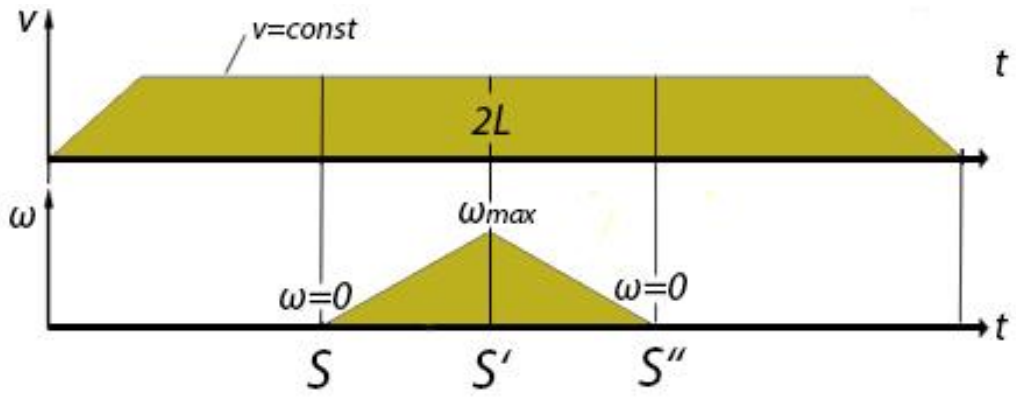

Figure 29 : Evolution de la du rayon de la courbe en fonction du temps

Sur la figure ci-dessus, *v* correspond à la vitesse linéaire du robot, *ω* à sa vitesse angulaire et *t* correspond au temps.

Pour obtenir une trajectoire fidèle à celle de la figure ci-dessous, il est nécessaire de ne plus fixer une valeur précise en ce qui concerne la vitesse angulaire du robot. Il faut en effet faire varier cette valeur au cours du temps. De cette façon, la valeur de la variable ω augmentera du point S au point S' et

diminuera ensuite jusqu'au point S". Cela permet au robot de se déplacer de façon plus fluide sans modifier brusquement sa direction.

Dans la version précédente du programme, nous avons déjà considéré les deux types de virages possibles et déclaré les variables nécessaires pour effectuer ces derniers. Il ne nous reste plus qu'à adapter le programme précédent afin de faire changer dynamiquement la valeur de la vitesse angulaire du robot pour qu'elle puisse correspondre au modèle théorique définit précédemment.

2.3.9.1.1 – Virage à gauche

Voici ci-dessous, la courbe clothoïde du robot pour le virage à gauche (voir figure 30). On remarque qu'elle ressemble beaucoup à la courbe de la version précédents, mais elle se base sur un principe théorique très différent et plus adapté pour permettre le déplacement d'un robot de ce type.

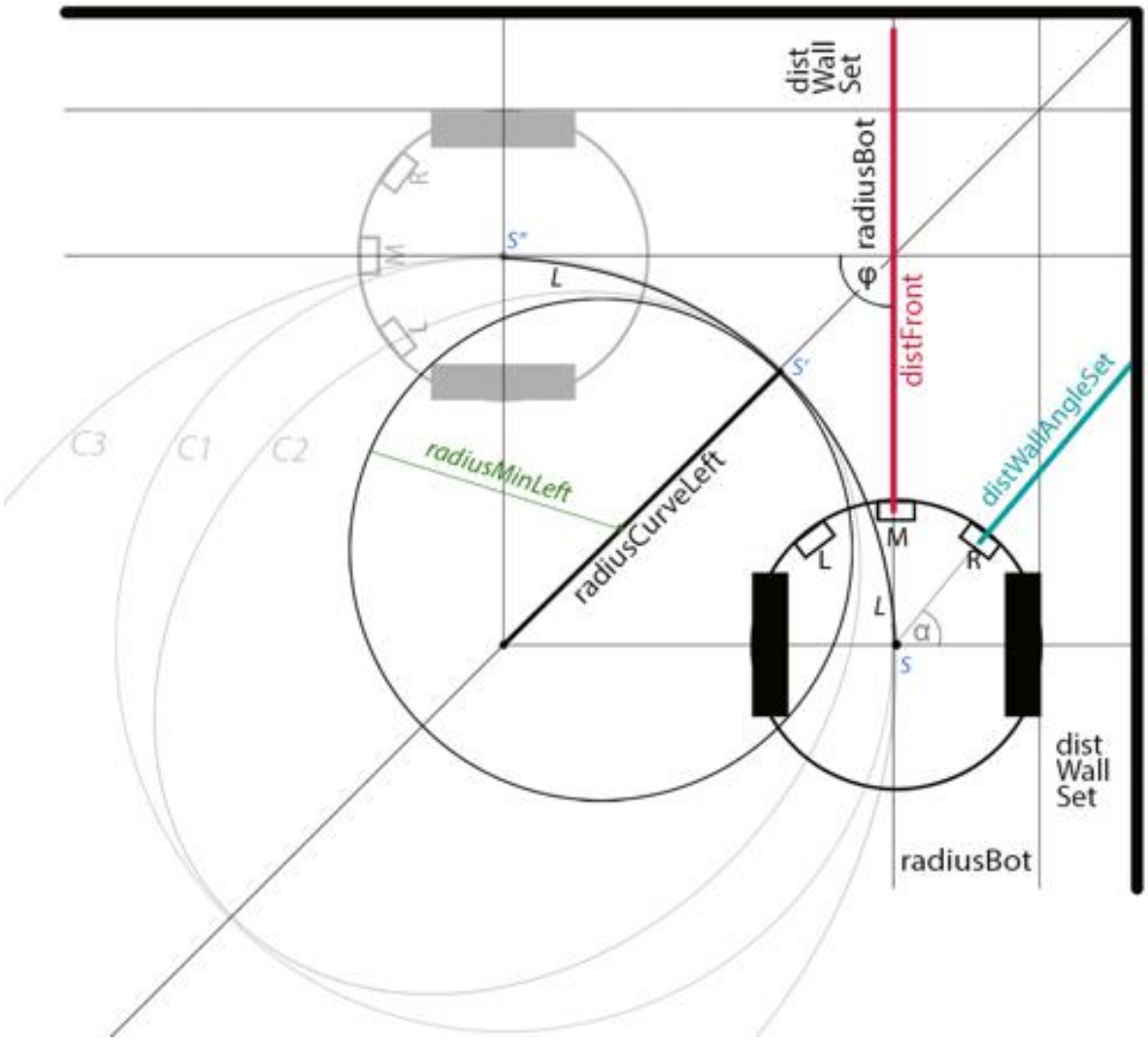

Figure 30 : Schéma explicatif de la courbe clothoïde pour le virage à gauche

2.3.9.1.2 – Virage à droite

La courbe correspondant au virage à droite (voir figure 31), comme dis précédemment ressemble également à celle de la version du programme précédente.

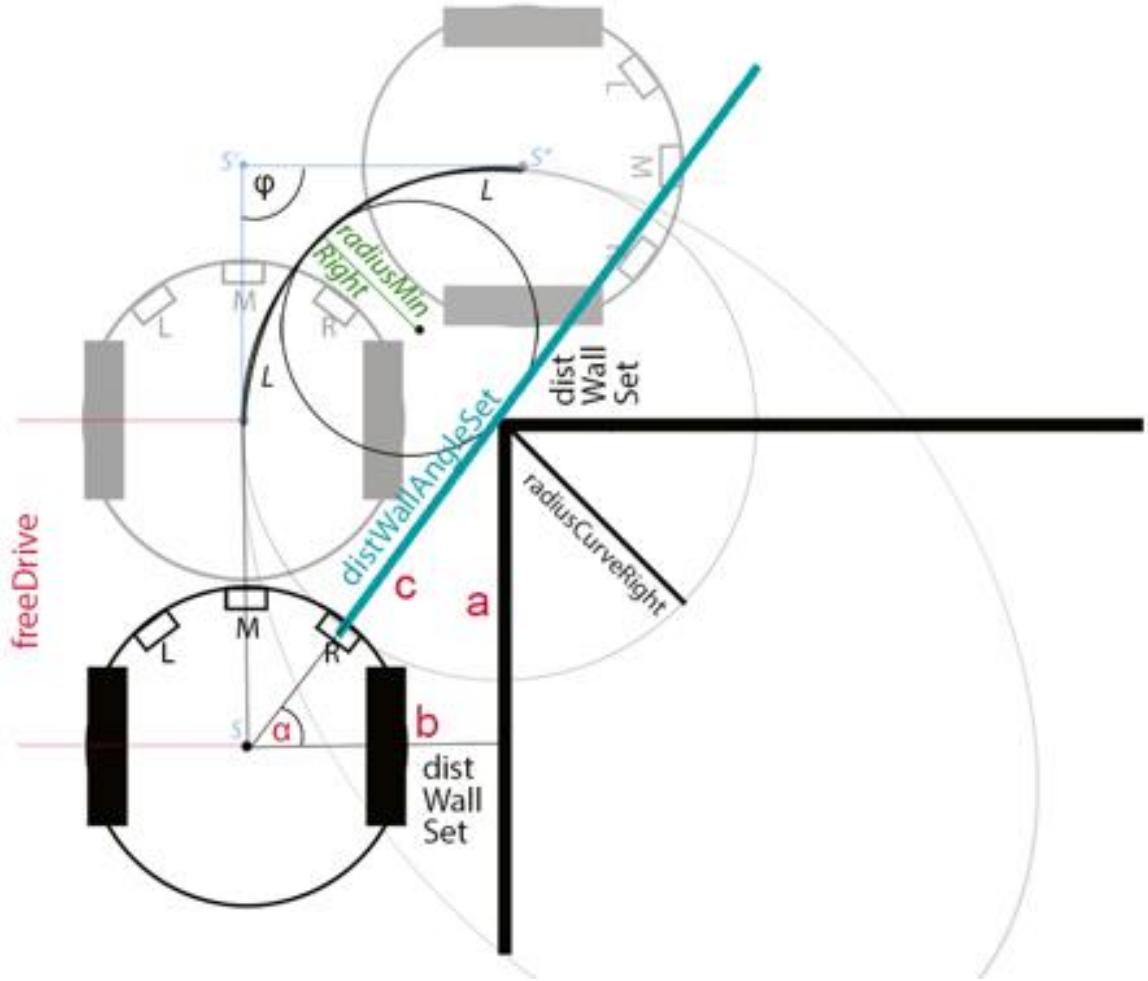

Figure 31 : Schéma explicatif de la courbe clothoïde pour le virage à droite

2.3.9.1.3 – Adaptation du programme

Dans cette version du programme, il nous faut également définir une valeur pour les rayons minimaux des courbes vers la droite ($R_{minRight}$) vers la gauche ($R_{minLeft}$). On a donc :

$$R_{minRight} = 0.5 * radiusCurveRight$$

$$R_{minLeft} = 0.5 * radiusCurveLeft$$

De plus, si nous supposons l'angle φ (voir figure) est dans le cas de notre labyrinthe toujours égal à 90 degrés, alors :

$$\vartheta_L = 90° - 0.5\varphi = 45° = \frac{\pi}{4}$$

Nous avons également besoin de définir les vitesses de rotation maximale du robot en ce qui concerne le virage à droite et le virage à gauche. Ces deux valeurs peuvent être calculées à partir de la vitesse linéaire du robot (*botForward_V*) ainsi que la valeur minimale du rayon du virage pour chaque cas. De ce fait :

$$R_{min} = \frac{\text{botForward\_V}}{\omega_{max}}$$

$$\omega_{maxRight} = \frac{\text{botForward\_V}}{R_{minRight}}$$

$$\omega_{maxLeft} = \frac{\text{botForward\_V}}{R_{minLeft}}$$

De plus, il nous faut prendre en compte le temps nécessaire au robot pour effectuer la courbe. Ce temps est calculé à partir de la longueur de la clothoïde (L) et de la vitesse linéaire du robot (*botForward_V*) ce qui nous donne :

$$L = 2 * \vartheta_L * R_{min}$$

$$L_{right} = 2 * \vartheta_L * R_{minRight}$$

$$L_{left} = 2 * \vartheta_L * R_{minLeft}$$

$$\text{tcL} = \frac{2L}{\text{vForward}}$$

$$\text{tcL}_{left} = \frac{2L_{left}}{\text{vForward}}$$

$$\text{tcL}_{right} = \frac{2L_{right}}{\text{vForward}}$$

Enfin, il est nécessaire de diviser la durée totale mise par le robot pour effectuer la courbe en nombre de segments d'égale durée définie par la variable $\text{DeltaL}_{\text{timeticks}}$.

Par exemple, si l'on a :

$$\omega_{max} = 45°$$

$$Delta\text{L}_{timeticks} = 1s$$

$$tcL = 6s$$

On peut calculer le nombre de segments total de la courbe :

$$\text{NoL}_{\text{timeticks}} = \frac{tcL}{DeltaL_{timeticks}} = \frac{6s}{1s} = 6$$

Dans cet exemple volontairement simplifié, nous obtenons un total de six segments de courbe ce qui signifie que la courbe clothoïde sera divisée en six parties égales. Cela signifie également que le robot adaptera sa vitesse six fois durant son déplacement le long de la courbe.

La valeur de ω variera alors plus ou moins entre chaque pas en fonction du nombre de pas :

$$\omega_{stepWidth} = \frac{\omega_{max}}{NoL_{timeticks}/2} = \frac{45°}{3} = 15°$$

Tableau 2 : Valeur de ω en fonction du temps pour la courbe clothoïde.

$NoL_{timetick}$	0	$1 * \frac{\Delta t}{6}$	$2 * \frac{\Delta t}{6}$	$3 * \frac{\Delta t}{6}$	$4 * \frac{\Delta t}{6}$	$5 * \frac{\Delta t}{6}$	0
Omega ω	0	15°	30°	45°	30°	15°	0

$$\Delta t = DeltaLtimeticks$$

Cet exemple est volontairement simplifié pour illustrer le fonctionnement de cette version du programme, mais en réalité, pour une précision optimale, il nous faudrait choisir une valeur de $DeltaL_{timeticks}$ de quelques ms au maximum.

2.3.9.2 - Conclusion

Contrairement aux deux premières versions du programme pour le projet WallFollower, je n'ai pu implémenter cette dernière par manque de temps. En effet, celle-ci, bien que prometteuse, nécessitait un peu plus de temps à implémenter dans un programme. Cette version du programme n'est donc pas encore fonctionnelle car non terminée.

Dans cette partie, nous avons pu voir le travail effectué pour mener à bout ma mission, nous avons pour commencer parlés de la prise en main ainsi que du fonctionnement de la plateforme Arduino et du robot aCTino. Ensuite, je vous ai présenté l'existant concernant le projet Wall-Follower puis mes résultats concernant la recherche d'algorithmes existants pouvant permettre de faire fonctionner le robot. Ensuite, je me suis concentré sur le travail effectué durant ce stage en présentant les différents cas auxquels le robot pouvait être confronté durant ses déplacements dans le labyrinthe de test. J'ai, par la suite, détaillé les différentes solutions mises en place afin que le robot puisse naviguer dans son environnement. Enfin, j'ai présenté différentes versions du programme permettant au robot de se déplacer de façon autonome dans le labyrinthe.

Partie 3 : Gestion de projet

Pour réaliser tout le travail présenté précédemment dans le temps imparti, j'ai dû élaborer une stratégie efficace de gestion de projet. Cette troisième partie est donc consacrée à la gestion de projet. Je commencerai par présenter la planification prévisionnelle mise en place les premières semaines de ce stage suivi de la planification effective correspondant aux tâches réalisées. J'expliquerai ensuite les différences entre ces deux plannings par le biais des difficultés rencontrées tout au long de ces 10 semaines. Enfin, je finirais par parler des axes d'amélioration possibles en ce qui concerne mon travail effectué.

3.1 - Planification

3.1.1 - Planning prévisionnel

Dans le but de mieux gérer mon temps et après m'être familiarisé avec le travail à réaliser, j'ai mis en place un planning prévisionnel. Ce dernier m'a permis de ne pas perdre de vue l'objectif principal de ma mission et également d'être plus productif en me définissant chaque jour, une ou plusieurs tâches permettant d'avancer dans cet objectif final.

Semaines	*Tâches prévues*
9 avril au 13 avril	Prise en main de la plateforme Arduino. Introduction aux différents phénomènes physiques liés à la robotique.
16 avril au 20 avril	Etude de l'architecture et prise en main du robot. Elaboration d'un programme test afin de permettre au robot de se déplacer.
23 avril au 27 avril	Elaboration d'un programme permettant au robot de naviguer dans son environnement. Ajout d'une fonctionnalité de détection d'obstacles à l'algorithme précédent.
2 mai au 4 mai	Elaboration d'un programme permettant de réguler la vitesse du robot.
7 mai au 11 mai	Recherche d'algorithmes permettant de corriger des problèmes physiques liés à la robotique.
14 mai au 18 mai	Ecriture d'un nouvel algorithme de déplacement plus complet pour le robot.
21 mai au 25 mai	Ecriture d'un nouvel algorithme de déplacement plus complet pour le robot. Test de fonctionnement du nouvel algorithme.
28 mai au 1 juin	Conception d'un programme permettant de visualiser graphiquement les obstacles rencontrés par le robot. Implémentation d'un algorithme permettant de visualiser graphiquement les obstacles rencontrés par le robot.
4 juin au 8 juin	Implémentation d'un algorithme permettant de visualiser graphiquement les obstacles rencontrés par le robot.
11 juin au 15 juin	Implémentation d'un algorithme permettant de visualiser graphiquement les obstacles rencontrés par le robot. Corrections et améliorations.

3.1.2 - Planning effectif

Après avoir établi le planning prévisionnel précédent, je me suis rendu compte au moment où j'ai démarré l'implémentation du premier programme de navigation pour le robot, qu'il était impossible de prévoir tout le travail à faire dès le début. En effet, de nombreuses tâches imprévues dont la réalisation était nécessaire se sont glissées dans mon programme. De plus, étant donné la difficulté de créer un programme de navigation pour le robot sans expérience préalable, le risque d'échouer était important. C'est pourquoi, j'ai décidé sur les conseils de mon maître de stage, de séparer le travail de création de l'algorithme du robot en plusieurs parties (présentées précédemment).

Semaines	*Tâches prévues*
9 avril au 13 avril	Révisions sur les bases de la programmation. Prise en main de la plateforme Arduino. Calibration des capteurs de distance infrarouge. Initiation à la robotique et aux phénomènes physiques liés. Etude de l'architecture et prise en main du robot. Conception et implémentation d'un algorithme permettant de réguler la vitesse des moteurs.
16 avril au 20 avril	Elaboration de programmes simples permettant au robot de se déplacer. Ajout d'une fonctionnalité de détection d'obstacles sur le robot. Etude de l'existant. Transformation du programme existant sous la forme d'un programme en machine-à-état. Recherche d'algorithmes. Conception et implémentation d'un algorithme permettant au robot de se déplacer dans le labyrinthe en prenant en compte différents cas. Tests de fonctionnement.
23 avril au 27 avril	Améliorations et optimisations du programme de navigation du robot. Conception et implémentation d'un algorithme permettant au robot de rester à une distance bien définie du mur de droite.
2 mai au 4 mai	Conception et implémentation d'un algorithme permettant au robot de rester à une distance bien définie du mur de droite. Tests de fonctionnement.
7 mai au 9 mai	Intégration des différentes fonctionnalités du robot dans un seul et même programme. Améliorations et optimisations du programme de navigation du robot.

14 mai au 18 mai	Conception d'un algorithme permettant au robot de se déplacer en effectuant des angles droits pour changer de direction.
21 mai au 25 mai	Implémentation d'un algorithme permettant au robot de se déplacer en effectuant des angles droits pour changer de direction. Tests de fonctionnement Améliorations et optimisations du programme de navigation du robot.
28 mai au 1 juin	Conception et implémentation d'un algorithme permettant au robot de se déplacer en effectuant des courbes en forme de quarts de cercle pour changer de direction. Tests de fonctionnement.
4 juin au 8 juin	Améliorations et optimisations du programme de navigation du robot. Conception d'un algorithme permettant au robot de se déplacer en effectuant des courbes en forme de clothoïde pour changer de direction.
11 juin au 15 juin	Conception d'un algorithme permettant au robot de se déplacer en effectuant des courbes en forme de clothoïde pour changer de direction. Documentation.

3.2 - Difficultés et problèmes rencontrés

Dans un premier temps, j'ai dû apprendre à communiquer de façon efficace en anglais, chose qui n'a pas été simple au début de ce stage. En effet, mon manque de vocabulaire en langage technique pour le domaine de la robotique a été l'un des principaux obstacles à la communication.

Pour commencer, mon manque d'expérience dans le domaine de la robotique m'a demandé beaucoup d'énergie et de temps au début de mon stage. Effet, il m'a fallu acquérir de nombreuses connaissances de base pour ensuite pouvoir manipuler le robot.

De plus, le fait d'avoir séparé l'étape de développement de l'algorithme du robot en trois parties différentes (angles droits, courbes circulaires et clothoïdes), m'a également pris plus de temps que prévu. Cependant, ces tâches intermédiaires m'ont permis de découvrir des phénomènes physiques à prendre en compte avant de créer un algorithme plus complet. Par ces contraintes rencontrées, je n'ai pas eu le temps de m'intéresser à l'application graphique permettant de visualiser les déplacements du robot ainsi que les obstacles rencontrés par ce dernier.

3.3 - Travail restant

Comme mentionné dans la partie décrivant ma mission pour ce stage, la création d'une application graphique permettant de visualiser les déplacements du robot était prévue. Cependant, par manque d'expérience dans le domaine de la robotique, il m'était impossible de réaliser toutes ces tâches en 10 semaines. Cette partie est donc manquante en plus de l'implémentation de la version du programme permettant au robot de se déplacer en effectuant des clothoïdes pour changer de direction.

Cependant, les deux versions fonctionnelles du programme pour le projet Wall Follower sont utilisables par le laboratoire et permettant de définir ma mission comme accomplie. En effet, mon travail à bien permis à l'équipe de chercheurs d'améliorer le programme acaBot qui était l'objectif principal de ma mission.

Actuellement, le déplacement du robot avec les courbes clothoïdes sont les plus efficaces que j'ai pu trouver, mais il existe peut-être d'autres types de déplacement ayant échappé à mes recherches.

Dans cette partie, j'ai d'abord présenté la partie gestion de projet : relative à mon stage. J'ai présenté l'organisation de mon travail tout au long de ces 10 semaines en comparant les tâches prévues dans le planning prévisionnel, aux tâches réalisées dans le planning effectif. J'ai enfin parlé des difficultés et problèmes rencontrés ainsi que du travail restant.

Partie 4 : Apports du stage et de la formation

Grâce à tout le travail réalisé au sein du laboratoire PSE-Lab, j'ai pu découvrir un nouveau secteur et développer mes compétences en informatique. Dans cette partie, je présenterais dans un premier temps les apports personnels et professionnels de ce stage. Je parlerai ensuite des différents aspects sur lesquels ce dernier m'a permis de gagner en expérience. Ensuite, je montrerai en quoi le temps passé à travailler au sein du laboratoire PSE-Lab m'a permis de préciser mon projet professionnel futur. Enfin, je ferai le lien entre ce stage et les cours dispensés lors de mes deux ans de formation au sein de l'IUT de La Rochelle.

4.1 - Apport personnel

Ce stage a grandement contribué à mon développement personnel et m'a permis de progresser dans des domaines qui me tiennent à cœur tels que :

- **Le voyage** : Passionné de voyage, l'immersion dans une culture différente de la mienne m'a permis de m'ouvrir sur le monde. J'ai, durant ces 10 semaines, pu élargir mon ouverture d'esprit à force de rencontres avec des personnes en provenance des quatre coins du globe.
- **La langue anglaise** : Bien que difficile au départ, le fait d'avoir travaillé deux mois et demi en communiquant uniquement en anglais m'a permis de franchir un cap en ce qui concerne l'apprentissage de cette langue. Grâce à l'immersion, je suis maintenant à même de tenir une conversation en anglais ; seul mon accent reste perfectible.
- **La communication** : De tempérament assez réservé, j'ai pu durant ce stage m'ouvrir aux autres ce qui m'a permis de mieux communiquer.

4.2 - Apport professionnel

Ce stage a contribué à mon développement également sur le plan professionnel sur plusieurs points :

- **La robotique** : La découverte de ce nouveau secteur d'activité très prometteur, m'a permis d'élargir mon champ de compétences par la résolution d'un projet concret aux exigences professionnelles.
- **L'expérience** : Ces 10 semaines de travail au sein du laboratoire m'ont permis de me forger une première expérience dans le domaine de l'informatique ; cela permet de se démarquer auprès des employeurs lors des demandes de recrutement.
- **La communication** : Le travail en milieu professionnel m'a obligé à communiquer de façon plus efficace afin d'éviter de perdre du temps.

4.3 - Projet professionnel futur

S'il est une chose qui m'intéresse dans le secteur de la robotique, c'est le fait de pouvoir élargir le champ d'actions de l'être humain et de permettre à ce dernier de s'affranchir des tâches répétitives et ennuyeuses. La robotique permet donc à l'humain d'utiliser le temps passé à effectuer des tâches automatisables d'une autre façon pour s'adonner à d'autres tâches participant à son développement personnel. A mon avis, la robotique est en train de faire évoluer notre société de façon conséquente en nous simplifiant la vie au fil des années par le biais de différents systèmes s'étant imposés dans notre quotidien (aspirateurs automatiques, pilotage automatique, trains et voitures automatiques). De cette façon, l'humain doit apprendre à accepter l'arrivée des robots et vivre en harmonie avec ces derniers comme il l'a fait pour l'arrivée d'internet et des smartphones. Toutes ces raisons, sont celles qui font du secteur de la robotique un secteur prometteur à exploiter à bon escient.

Pour ce qui est de mon projet professionnel futur, je n'ai, malgré ce stage, pas encore eu le temps de le définir clairement. En effet, il me faudrait pour ce faire découvrir d'autres secteurs afin de me rendre compte au mieux des possibilités qu'offre le domaine de l'informatique.

4.4 - Lien avec enseignements dispensés

Plusieurs des enseignements dispensés tout au long de ces deux années de formation au sein de l'IUT de La Rochelle, m'ont été utiles pour mener à bien ma mission :

- **L'anglais** : Indispensable aujourd'hui, l'anglais à l'IUT m'a permis d'acquérir le vocabulaire de base du domaine de l'informatique ainsi que la grammaire de la langue.
- **L'expression-communication** : Les différentes méthodes acquises en cours m'ont permis de m'exprimer de façon plus claire et plus efficace.
- **Architecture Embarquées** : Les robots étant des systèmes embarqués, les connaissances acquises durant ce cours m'ont permis de mieux comprendre le fonctionnement de ces derniers. De plus, j'ai pu faire en sorte d'optimiser le programme du robot grâce à mes connaissances sur les systèmes embarqués.
- **Algorithmique et programmation** : Tous les cours d'algorithmique et programmation m'ont permis d'acquérir la logique de base de la programmation indispensable à ce stage. De plus, un bon nombre de bonnes pratiques enseignées à l'IUT de La Rochelle m'ont permis de rendre mes programmes plus lisibles donc plus maintenables.
- **L'expérience** : De façon générale, l'expérience acquise au cours de nombreux travaux pratiques et projets réalisés au cours de ces deux ans, m'a permis de comprendre et m'adapter plus rapidement au contexte de ce stage. L'expérience m'a de plus permis de mieux évaluer la durée de chacune de mes tâches.

Dans cette dernière partie, j'ai présenté les apports de mon stage d'un point de vue personnel. Ensuite, je me suis concentré sur les apports professionnels puis sur l'impact de ce stage sur mon futur projet professionnel. Enfin, j'ai fait le lien entre les différents enseignements dispensés au cours de ma formation avec ce stage.

Conclusion

Ce stage de 10 semaines au sein du laboratoire PSE-Lab de l'Université Beuth Hochschule für Technik de Berlin m'a fait découvrir en profondeur le secteur d'activité prometteur de la robotique. Le projet Wall-Follower auquel j'ai été affecté m'a permis de mettre à profit bon nombre de connaissances acquises au cours de ma formation de deux ans au sein de l'IUT de La Rochelle. La réalisation d'un projet de cette envergure constituait pour moi un vrai challenge du fait des exigences et des responsabilités qu'il implique.

Cette première expérience à l'étranger a débuté par une phase d'organisation. En effet, il m'était indispensable de prendre en main mes responsabilités en ce qui concerne les démarches annexes à ce stage comme ma demande de bourse, la recherche d'un appartement ainsi que d'autres démarches administratives sur place.

Ma mission sur ces 10 semaines, consistait à aider l'équipe du laboratoire PSE-Lab à améliorer le programme acaBot, permettant aux étudiants et lycéens de découvrir le secteur de la robotique et les bases de la programmation ; cela par le biais de différents programmes sur la plateforme Arduino.

Les deux premières semaines de mon stage, ont été très riche en découvertes. En effet, il me fallait me familiariser avec les différents outils utilisés par le laboratoire ainsi que comprendre le fonctionnement du robot et de la plateforme Arduino sur laquelle ce dernier fonctionne. J'ai pu découvrir le secteur de la robotique avec le guide « Get Started on acaBot » écrit par M. Andreas DÖPKENS et M. Brian SCHÜLER. Par le biais de différents programmes, j'ai dû pour commencer apprendre à utiliser le robot mis à ma disposition pour ensuite développer des programmes plus complexes sur ce dernier.

Dans un deuxième temps, j'ai été affecté au projet Wall Follower qui consistait à développer un programme permettant au robot de suivre le mur se trouvant sur sa droite. Cependant, au début de ce stage, mes connaissances dans le secteur de la robotique étaient limitées ; j'ai donc, sur les conseils de mon maître de stage, divisé cette tâche en trois sous-tâches. La première consistait à implémenter un programme permettant au robot de suivre le mur se trouvant à sa droite en effectuant des angles droits afin de changer de direction. La deuxième tâche devait également permettre au robot de suivre le mur de droite mais en effectuant des courbes en quarts de cercle pour changer de direction. La troisième étape consistait à faire changer le robot de direction en remplaçant les courbes en quarts de cercle par des courbes clothoïdes.

Contraint en temps, je n'ai pu aller au terme de cette troisième étape. Pour ce qui est des deux premières, elles disposent chacune d'un programme fonctionnel ce qui m'a permis de définir ma mission comme accomplie. En effet, mon travail à bien permis aux chercheurs du laboratoire de faire évoluer le programme acaBot.

L'accomplissement de cette mission a été rendu possible par la mise en place d'une stratégie efficace de gestion de projet basée sur un planning prévisionnel ; j'ai pu grâce à cela, me rendre compte de l'avancement de mon travail chaque jour, mais surtout, de rester centré sur l'objectif principal de ma mission.

Finalement, ce stage a contribué à mon développement intellectuel sur les plans personnels et professionnels. Cela a été rendu possible par les efforts que j'ai effectué afin de mieux communiquer en langue anglaise. En effet, l'anglais n'étant pas ma langue d'origine et n'ayant jamais eu de réelle occasion de parler dans cette langue, je me suis vite rendu compte de mes lacunes principalement en ce qui concerne l'expression orale. J'ai pu combler par des efforts ces difficultés ; cela m'a permis de m'ouvrir sur le monde, de clarifier l'état de ma pensée et de rencontrer d'autres étudiants en provenance des quatre coins du globe.

Glossaire

IDE : Environnement de programmation, souvent présenté sous la forme d'un logiciel, qui contient un ensemble d'outils permettant au développeur de gagner en productivité.

Arduino : Plateforme de prototypage open source permettant de créer des systèmes électroniques divers grâce à des cartes électroniques dont les plans sont disponibles librement sur le site officiel.

Compilateur : Programme qui transforme un code écrit en un langage de programmation, en code exploitable par un ordinateur ou une carte électronique programmable.

Convertisseur USB-Série : Un convertisseur USB-Série permet de programmer une carte électronique depuis un ordinateur par le biais d'un port USB. Cet outil sert à faire communiquer deux périphériques n'utilisant pas le même protocole.

Open source : Logiciel ou programme dont le code source est accessible, modifiable et redistribuable librement.

Algorithme : Suite finie d'instructions itératives permettant de résoudre un problème ou d'obtenir un résultat.

Capteurs : En électronique, un capteur est un dispositif qui permet de transformer une grandeur physique observée en une tension électrique exploitable.

Actionneurs : En électronique, un actionneur est un dispositif permettant de modifier physiquement l'état d'un système.

C++ : Langage de programmation compilé inventé au début des années 80. Il est aujourd'hui réputé pour ses performances.

Bibliothèque logicielle : Fichiers informatiques contenant déjà des instructions propres à un système particulier et contient toutes les fonctions nécessaires à la programmation de ce système.

CPU : Composant électronique exécutant les instructions machines d'un programme informatique. C'est l'élément principal de tous les ordinateurs et cartes électroniques.

Timer : En informatique, un timer est un dispositif physique et logiciel permettant de contrôler le temps dans un programme.

Switch-case : Type d'itération en langage C++ permettant de tester l'égalité d'une variable.

Tension : En électronique, une tension correspond à la circulation d'un champ magnétique mesuré dans un circuit. Elle se mesure en Volt.

Analogique : L'électronique analogique est la discipline traitant des systèmes électroniques sur des grandeurs à variation continue.

ADC : Un Convertisseur Analogique Digital est un dispositif permettant de convertir une information d'entrée analogique (souvent en provenance d'un capteur) en information de sortie numérique exploitable par un programme informatique.

Bibliographie

- A Clothoïdal Wall Follower — Andreas DÖPKENS & Brian SCHÜLER
- An improved Wall Follower — Andreas DÖPKENS & Brian SCHÜLER
- Get started on ACABOT for interns — Andreas DÖPKENS & Brian SCHÜLER
- Pentagram Route — Andreas DÖPKENS & Brian SCHÜLER
- The proportional-motor-controller — Andreas DÖPKENS & Brian SCHÜLER
- The wall Follower — Andreas DÖPKENS & Brian SCHÜLER
- Virer sans déraper — Jean-Marie DE KONINCK & Frédéric GOURDEAU

Table des annexes

Résumé

La robotique est aujourd'hui un domaine occupant une part importante dans notre société. C'est grâce aux robots que l'humain peut s'affranchir de différentes tâches, le plus souvent pénibles, auxquelles ils sont confrontés. De plus, les robots augmentent significativement le champ d'action des humains en permettant des actions qui semblaient impossible sans leur aide. C'est pourquoi le laboratoire de recherche en robotique de l'Université de Beuth-Hochscule für Technik de Berlin a développé le programme acaBot permettant aux étudiants d'apprendre les bases de la robotique par le biais de différents programmes. Tout au long de ce stage, j'avais pour rôle d'aider l'équipe du laboratoire à faire évoluer le programme acaBot par la création de différents programmes permettant le déplacement d'un robot de modèle aCTino fonctionnant grâce à une carte Arduino Léonardo.

Pour commencer, j'ai donc dû étudier le fonctionnement de la plateforme Arduino ainsi que les bases de la robotique. Ensuite, j'ai dû concevoir et implémenter un algorithme permettant au robot de se déplacer dans un labyrinthe de test conçu spécialement pour lui. Pour ce faire, il m'a été nécessaire de rechercher différents algorithmes possibles et de sélectionner celui le plus adapté. J'ai donc développé un programme permettant au robot de se déplacer en suivant le mur se trouvant à sa droite tout en prenant en compte la gestion des obstacles et le calcul des trajectoires. J'ai finalement terminé ce stage par le test concluant de l'algorithme de déplacement du robot. Je n'ai cependant pas eu le temps d'aller au terme de la dernière étape qui consistait à créer une interface permettant de visualiser les obstacles rencontrés par le robot. Ce stage m'a finalement permis de découvrir le secteur de la robotique, tout en mettant à profit mes connaissances acquises durant ces deux années de formation à l'IUT ainsi qu'en améliorant mon niveau d'anglais, et ce, en me constituant une expérience professionnelle solide.

Mots clés : Robotique, Découverte, Apprentissage, Arduino, Navigation, Trajectoires, Obstacles

Abstract

Nowadays, robotics is an important part of our society. Thanks to robots, humans can free themselves from different arduous tasks they confront every day. Moreover, robots significantly increase the field of action of humans by allowing actions impossible without their help. That is why the Robotics Research Laboratory at the University of Beuth-Hochscule für Technik in Berlin has developed the acaBot program, which enables students to learn the basics of robotics through various programs. Throughout this course, my role was to help the laboratory team to develop the acaBot program by creating different programs allowing the movement of an aCTino model robot using an Arduino Leonardo card.

I started studying the Arduino platform and the basics of robotics. Then, I had to design and implement an algorithm allowing the robot to move through a test maze, designed especially for it. To do this, I had to look for different possible algorithms and select the most suitable one. So, I developed a program that allows the robot to move along the wall to its right while considering obstacle management and trajectory calculation. I finally finished this internship with the conclusive test of the robot's motion algorithm. However, I didn't have time to go to the end of the last step which consisted in creating an interface allowing to visualize the obstacles encountered by the robot. This internship finally allowed me to discover the robotics sector, to use my knowledge acquired during these two years of training at the IUT as well as to improve my level of English while gaining solid professional experience.

Keywords: Robotics, Discovery, Learning, Arduino, Navigation, Trajectories, Obstacles

Annexes

Annexe 1: aCTino Robot architecture

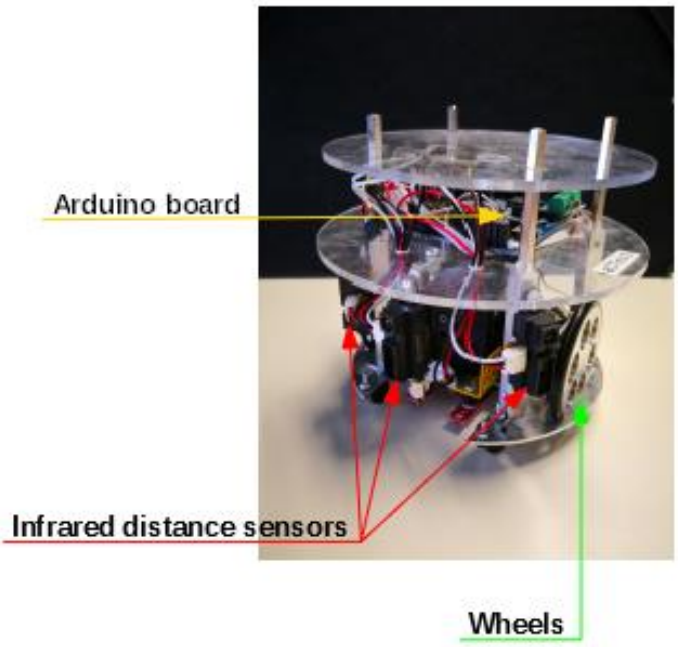

Figure 32 : Schéma explicatif de l'architecture du robot aCTino.

Annexe 2: Arduino IDE

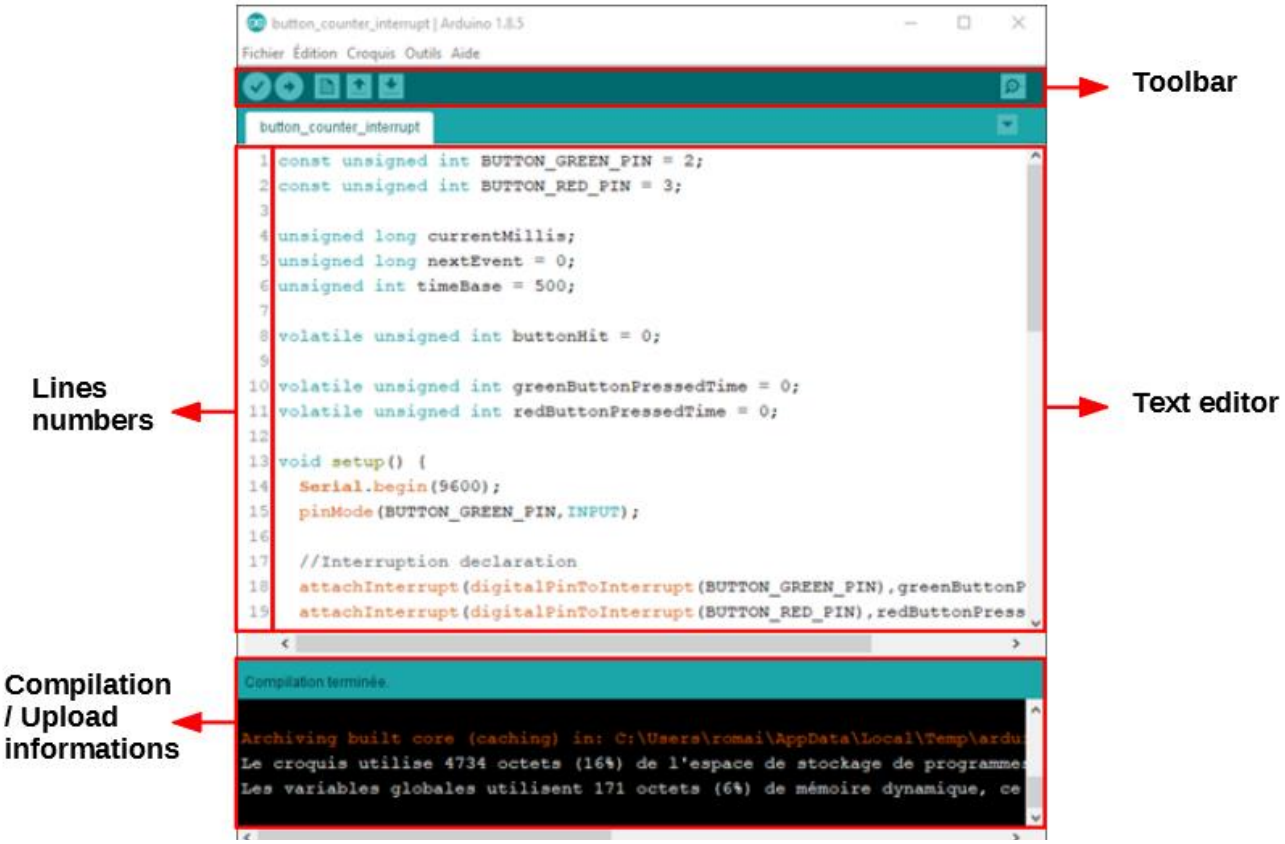

Figure 33 : Schéma explicatif de la disposition de l'IDE Arduino

Annexe 3: Arduino program structure

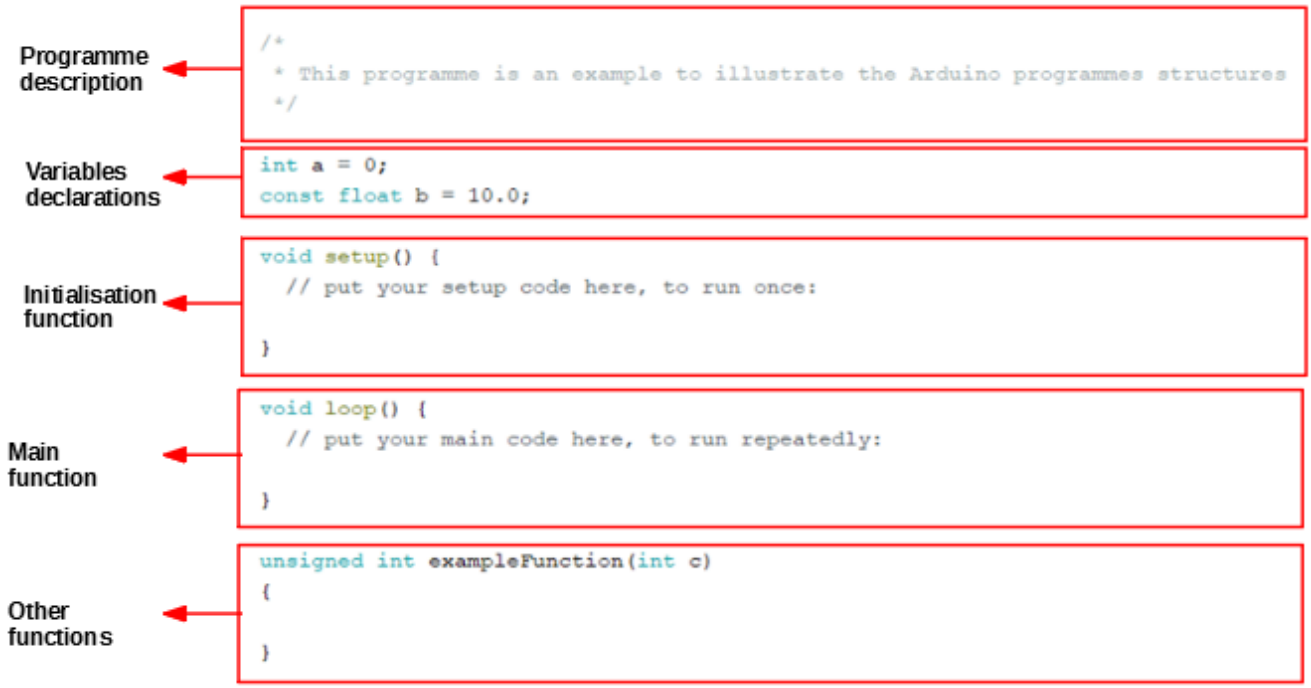

Figure 34 : Structure de base d'un programme Arduino.

Annexe 4: Powerfull Millis-Timer Algorithm

Figure 35 : Algorithme de gestion du temps "Powerfull Millis-Timer Algorithm" élaboré par M. Andreas DÖPKENS et M. Brian SCHÜLER.

Annexe 5: State machine structure program

State machine variables

```
//State machine states
#define STATE_SEARCH_WALL 1
#define STATE_FOLLOW_WALL 2
#define STATE_DRIVE_CURVE 3
#define STATE_AVOID_OBSTACLE 4


//State machine current state
unsigned int robotState;
```

State machine program

```
switch (robotState)
{
  case STATE_SEARCH_WALL    : if (distSensR > 400 || distSensM > 400 || distSensL > 400)
                              {
                                robotState = STATE_FOLLOW_WALL;
                                vOmega(10.0, 0.0);
                              }
                              break;

  case STATE_FOLLOW_WALL    : if (distSensL < 400 && distSensM < 450)
                              {
                                robotState = STATE_DRIVE_CURVE;
                                vOmega(10.0, 70.0);
                              }
                              else if (distSensL > 600 || distSensM > 450 || distSensR > 500)
                              {
                                robotState = STATE_AVOID_OBSTACLE;
                                vOmega(6.0, -130.0);
                              }
                              break;

  case STATE_DRIVE_CURVE    : if (distSensL > 400 && distSensL < 600 && distSensM < 450)
                              {
                                robotState = STATE_FOLLOW_WALL;
                                vOmega(10.0, 0.0);
                              }
                              else if (distSensL > 600 || distSensM > 450 || distSensR > 500)
                              {
```

Figure 36 : Structure d'un programme machine à état.

Annexe 6: Existing program for the Wall-Follower project

```
if (!wallFound)
{
  vOmega(10.0, 0.0);
  wallFound = (distSensR > 400 || distSensM > 400 || distSensL > 400);
}
else
{
  //Driving along the wall
  if (distSensR > 400 && distSensR < 610 && distSensL < 500 && distSensM < 450)
  {
    vOmega(10.0, 0.0);
  }

  //Driving a curve
  else if (distSensR < 400 && distSensM < 450 && distSensL < 500)
  {
    vOmega(10.0, -70.0);
  }

  //Avoiding obstacles and walls
  else if (distSensR > 650 || distSensM > 450 || distSensL > 500)
  {
    vOmega(6.0, 130.0);
  }
}
```

Figure 37 : Programme existant pour le projet Wall-Follower écrit par Jonas (2014)

Annexe 7: Infrared distance sensors curve

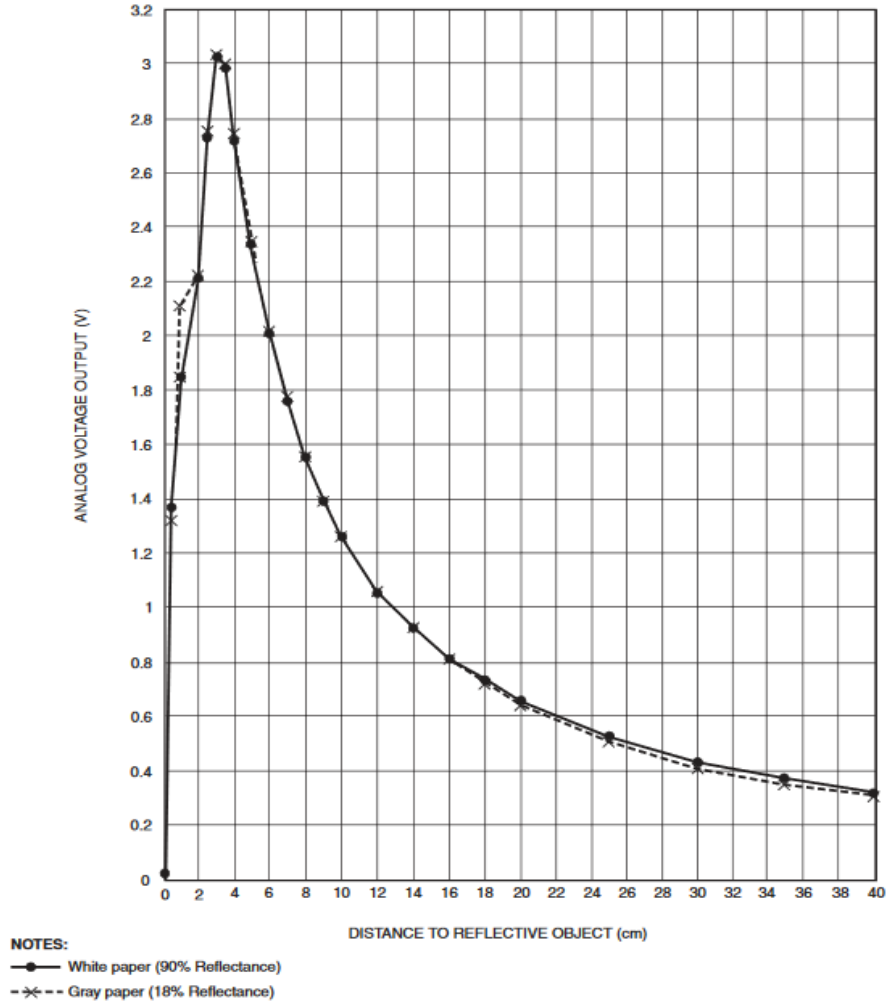

Figure 38 : Courbe représentant les valeurs renvoyées par le capteur de distance infrarouge en fonction de la distance.

Annexe 8: Trajectory correction algorithm

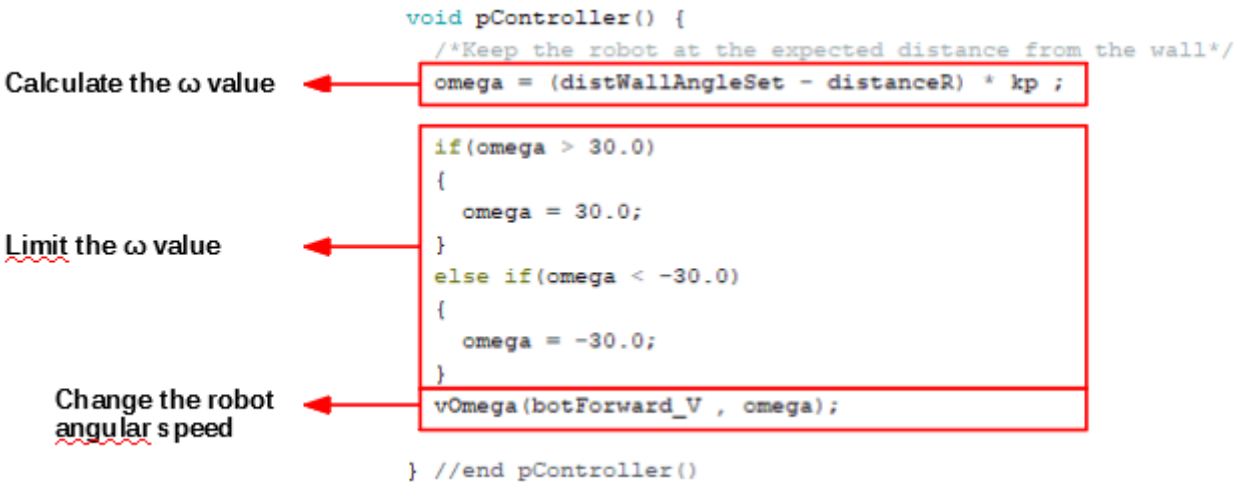

Figure 39 : Algorithme de correction de la trajectoire du robot en ligne droite.