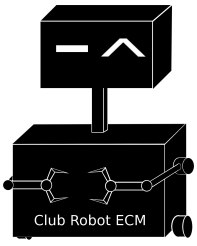
RAPPORT FINAL
Projet Transverse E=M6

TUTEUR
Alain Kilidjian

CHEF DE PROJET
Sara El Afia

MEMBRES
Rodolphe Bouley
Jean Feutrie
Jeremy Grand
Axel Jardin
Adrien Kvaternik
Guillaume Marin
Yuan Tian



Nous tenons à remercier M. Dethoor pour et M. Kilidjan pour leur précieuse aide.

Table des matières

Introduction

Le projet transverse E=M6 a comme but de participer à la Coupe de France de Robotique en concevant deux robots qui affronteront d'autres équipes lors de matchs, et devront respecter le cahier des charges du règlement de la coupe. Cet évènement se tiendra du 8 au 11 Mai 2013 à La Ferte-Bernard, conjointement organisé par la commune et par l'association « Planète Sciences ». Notre équipe y représentera l'Ecole Centrale Marseille auprès des autres écoles françaises et des associations de robotique participantes. Ainsi, nous présenterons un dossier technique aux organisateurs et assurerons notre communication auprès des autres équipes de robotique. Durant les matchs de la coupe, les différentes équipes se défient selon des règles qui nous ont été communiquées fin septembre. D'après les règles récurrentes qui sont retrouvées chaque année, les robots autonomes devront respecter des contraintes de dimensionnement, éviter les collisions avec les robots adverses, et s'arrêter automatiquement à la fin des matchs (durée de 90s).

Le thème de l'édition 2013 sera « Happy Birthday » pour la coupe, dont les règles de jeu et les nombreuses actions à réaliser rendent ce projet transverse plus complexe et ambitieux d'année en année. Pour collecter de précieux points pour leur équipe pendant les matchs, les deux robots de chaque camp tenteront de pousser des cadeaux, collecter et lancer des cerises sur un gâteau, empiler des verres, gonfler un ballon, et éteindre des bougies.

Notre projet est donc de construire 2 robots homologables, de participer à la coupe en obtenant le meilleur classement possible, puis de transmettre au mieux nos acquis à la prochaine promotion.

Chapitre 1

Présentation du PT E=M6

1.1 Contexte, historique de la demande

Participer à la coupe de France de robotique nécessite de nombreuses compétences, notamment en informatique et en électronique, non enseignées à Centrale Marseille. C'est pourquoi nous avons rencontré l'équipe précédente pour comprendre le travail qu'ils avaient effectué, et les aider à finaliser leur robot en apprenant auprès d'eux. Nous avons pu ainsi acquérir plus rapidement les connaissances nécessaires et réutiliser une partie de leur travail pour notre projet, comme l'algorithme du calcul de chemin.

L'équipe 2011 du projet E=M6 a participé à l'édition de Mai 2012 de la coupe en homologuant et qualifiant un robot sur les deux construits et a terminé 111e avec 31 points sur l'ensemble de ces trois matchs.

1.2 Parties prenantes, acteurs du projet

Le projet transverse E=M6 est un projet ambitieux au budget important, commandé et financé par l'Ecole Centrale Marseille. Il est lié depuis maintenant trois ans à l'association régie par la loi 1901 « Club Robotique de l'école Centrale Marseille », composée d'un Bureau restreint et de membres actifs. Après l'Assemblée Générale de passation, le nouveau bureau est :

Présidente : Sara El Afia

Vice-Président : Guillaume Marin

Trésorier : Axel Jardin

Vice-Trésorier : Jean Feutrie

Secrétaire : Adrien Kvaternik

Vice-Secrétaire : Rodolphe Bouley

Membres : Yuan Tian, Jérémy Grand

Cette équipe constitue ainsi les acteurs du projet. L'Ecole et en particulier notre tuteur Monsieur Alain Kilidjan veillent sur l'avancée de notre projet. Les trésoriers sont les relais entre Monsieur Gallais et le projet pour faire les diverses commandes de matériel. La Présidente et Chef de Projet Sara El Afia s'occupe de l'organisation des tâches et de la recherche de partenariats.

1.3 Identification de la problématique

Nous avons accompagné l'ancienne équipe à l'édition 2012 de la coupe de France de robotique afin de mieux définir les enjeux de notre projet. Ceci nous a permis de collecter diverses informations : nous avons ainsi noté que la plupart des équipes utilisaient des matériaux bon marché et facilement usinables tels que le plexiglas, PVC, le bois... Une grande partie d'entre eux possédait également un nombre de cartes électroniques réduit par rapport à nous. Nous avons pris ces éléments en compte, nos nouvelles cartes et matériaux nous permettront un gain de place non négligeable dans le robot et des coûts réduits. Nous avons aussi pu établir une liste des différentes entreprises partenaires des autres associations de robotique afin de pouvoir faire notre recherche de sponsors.

1.4 Objectifs, livrable

Notre objectif est de participer à la coupe de France de robotique en tant que représentants de l'Ecole Centrale Marseille. Pour passer avec succès les homologations et participer aux matchs, nous devrons livrer un robot fonctionnel répondant au cahier des charges de la coupe.

Au vu du travail conséquent à réaliser et du nombre d'heures consacrées au projet, nous ne pouvons produire et présenter un robot prêt et fonctionnel pour le 11 janvier ; notre objectif principal est donc d'achever la conception du robot et l'usinage des pièces dans le temps imparti. L'assemblage du robot, les tests et sa finalisation se feront en vue de la Coupe de France le 20 Mai.

Chapitre 2

Les règles de la coupe

2.1 Introduction aux règles

La coupe de France de robotique a pour particularité de changer le règlement pour chaque édition. En effet, bien que la base des règles, comme les conditions minimum de qualification, les limitations matérielles ou les règles de sécurité, soit récurrente, le « but du jeu » et le système de gain de point change chaque année, en accord avec le thème de l'année en question.

Plus en détails, les règles invariables sont :

- La limite d'âge pour les participants est de 30 ans, toutefois, chaque équipe peut disposer d'un unique encadrant auquel cette limite ne s'appliquera pas.
- Chaque match se joue entre deux équipes, et dure 90 secondes.
- Chaque robot doit être parfaitement autonome et aucune intervention humaine n'est autorisée pendant les 90 secondes. Il va sans dire que télécommander son robot est interdit.
- Les robots n'ont pas le droit de se heurter, d'endommager le décor ou le robot adverse, ni d'entraver de quelque façon que ce soit le fonctionnement de ce dernier.
- Pour se qualifier, chaque équipe doit présenter un robot capable de sortir de sa zone de départ et d'effectuer une action rapportant des points. (Et bien sûr, pas d'intervention humaine autorisée pendant l'épreuve d'homologation).
- Chaque robot doit respecter des limitations de taille. Une limite au départ, et une autre, un peu plus large, pendant le fonctionnement (le robot a donc une petite marge pour pouvoir se déployer et utiliser des outils comme des bras ou autres)

Le thème de l'édition 2013, pour commémorer la vingtième coupe de France, est « Happy Birthday ! ». Ainsi les robots seront conviés à l'anniversaire de la coupe et gagneront des points en se comportant en invités exemplaires.

Règles particulières à cette édition : chaque équipe peut utiliser deux robots : 1 principal, dont le périmètre ne doit pas dépasser 1000mm non déployé et 1400mm déployé, et 1 secondaire dont le périmètre doit être inférieur à 600mm non déployé et 800mm déployé.

2.2 Table de jeu

La table de jeu se présentera comme ce qui suit : Soit une table rectangulaire de 3m par 2m avec des bordures. Les dimensions précises et les références des couleurs seront disponibles en annexe. Avec :

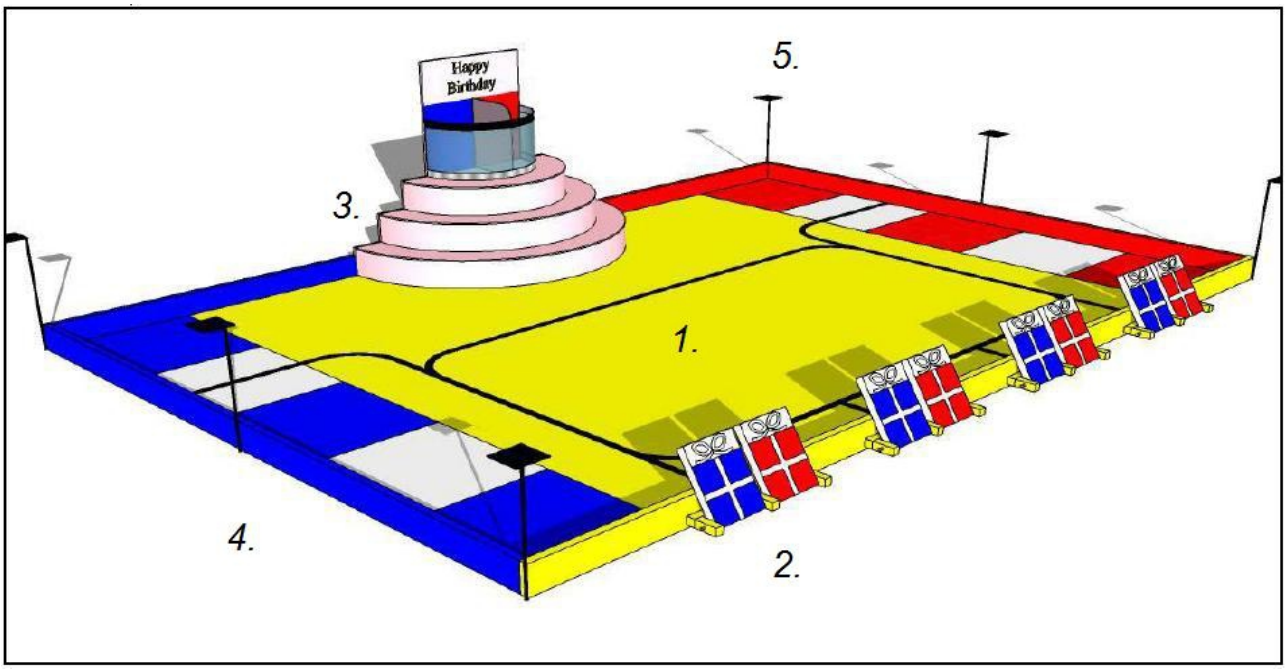


FIGURE 2.1 – Table de jeu

en 1.(zone jaune) : La zone centrale, où seront disposés les verres. A noter qu'une bande noire sera tracée au sol afin de permettre aux robots, s'ils disposent des capteurs et de la programmation adaptée, de se repérer.

En 2. : Les cadeaux, qui sont représentés par des planches pouvant basculer.

En 3. : Le gâteau, qui comporte 3 étages.

Sur les deux premiers étages s'alternent les bougies de chaque couleur, il s'agit en fait de balles de tennis posées sur des tubes et retenues par des élastiques ;« Souffler la bougie » consiste en réalité à faire tomber la balle dans le tube. Sur le troisième étage se trouve le bol

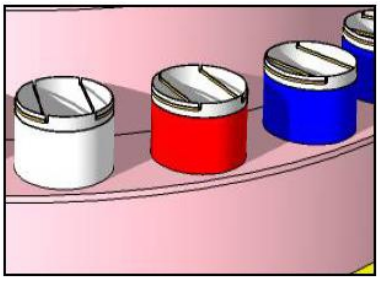


FIGURE 2.2 – Les Bougies

dans lequel le robot doit lancer les cerises, qui sont en réalité des balles de ping pong.

En 4. (zones bleue et rouge) : Les zones de départ : chaque robot commence dans une des 5 cases que comporte sa zone, il doit être en contact avec le rebord et ne pas dépasser de celle-ci. Les robots devront également ramener leurs verres dans leur zone respective.Dans ces zones seront également placées des assiettes d'où les robots prendront les cerises.

En 5. : Les 6 piliers que l'on remarque sont destinés à recevoir des balises de repérage, à raison de 3 piliers par équipe, que les équipes auraient pu préparer.

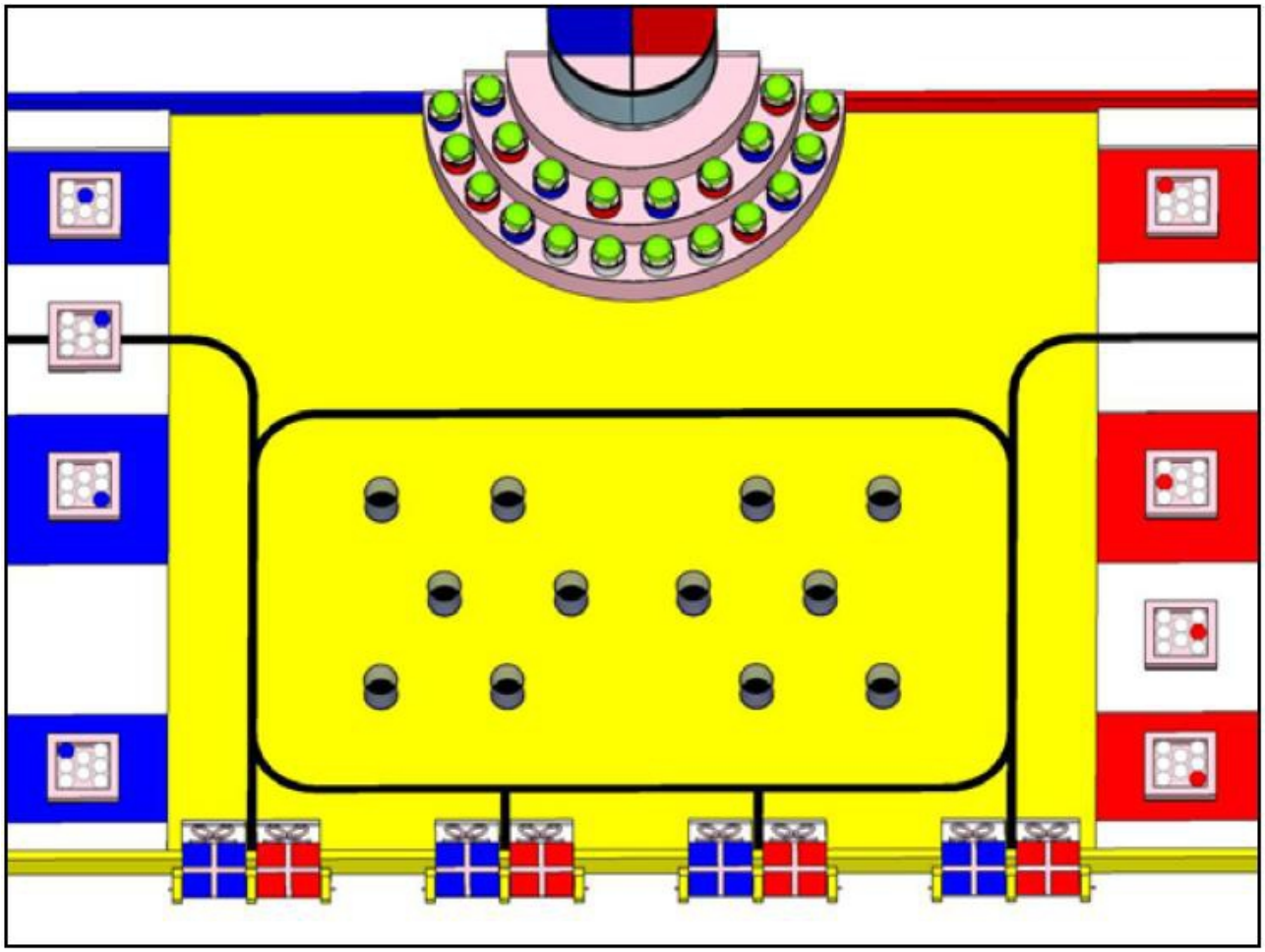


FIGURE 2.3 – Vue de dessus de la table avec tous ses éléments

Déroulement d'un match : Lorsqu'un match est annoncé, chaque équipe se voit attribuée aléatoirement un côté du terrain (et la couleur correspondante). A partir de ce moment, chaque équipe dispose de 3 minutes afin de mettre leurs robots en place et d'effectuer les derniers ajustements. Lorsque le top départ sera lancé, la seule intervention humaine autorisée sera l'activation d'un interrupteur au moyen d'un fil tiré

Gagner des points : A la fin des 90 secondes, les juges inspecteront la table de jeu et attribueront les points. Ils seront attribués selon ces règles :

- Ouvrir un cadeau : Ouvrir un cadeau, à savoir faire basculer la planche où est peint un cadeau, rapporte 4 points à l'équipe correspondante à la couleur du cadeau.
- Souffler les bougies : souffler une bougie, c'est à dire enfoncer complètement la balle dans le tube, rapporte 4 points à l'équipe de la même couleur que la bougie ou, dans le cas d'une bougie blanche (dite de coopération), aux deux équipes. Note : Si toutes les bougies blanches ont été soufflées, chaque équipe gagne 20 points bonus.
- Mettre des cerises sur le gâteau : les robots peuvent récupérer des cerises disponibles dans des assiettes dans les zones de départ et les lancer dans les paniers attribués à chaque équipe au sommet du gâteau. Chaque cerise blanche dans un panier rapporte 2 points à l'équipe de la même couleur. Toutefois, chaque assiette comporte une cerise pourrie,

reconnaissable car colorée, et la présence d'une (ou plusieurs) cerises pourries dans le panier divisera les points des cerises par 2. Note : une cerise pourrie d'une équipe ne peut pas être envoyée dans le panier de son adversaire.

- Apporter des verres au buffet : Le robot doit ramasser les verres dans la zone centrale et les rapporter dans sa zone. Chaque verre dans sa zone rapporte 4 points à l'équipe.Note : Les robots sont encouragés à réaliser des pyramides avec les verres. Chaque verre voit ses points multipliés en fonction de l'étage auquel il se trouve : un verre posé sur le sol rapportera 4 points, tandis qu'un verre au premier étage en rapportera 8, qu'un au deuxième étage 12 et ainsi de suite. Par exemple, la pyramide ci-dessous rapportera à son équipe 2*4 points pour la base, 8 points pour le premier étage, 2*12 pour le deuxième et 16 pour le dernier ; soit 56 points au total pour 6 verres (qui n'auraient valu que 24 points si tous posés au sol).
- Funny Action : à la fin des 90 secondes de jeu, les robots disposent de 10 secondes supplémentaires pour réaliser leur « Funny Action », ils doivent gonfler un ballon (préalablement embarqué) et le maintenir gonflé. Si cette action est validée, elle rapporte 12 points.Note : La Funny Action doit être réalisée uniquement après la fin des 90 secondes de jeu, et le robot n'est pas autorisé à se déplacer pendant la réalisation de celle-ci.

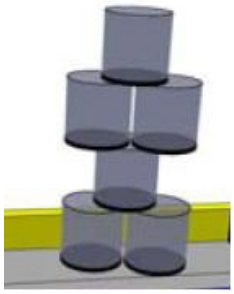


FIGURE 2.4 – Exemple de pyramide

Pénalités :

- Une équipe est déclarée disqualifiée (ou forfait) si aucun de ses robots ne parvient à quitter sa zone de départ durant la totalité du match. Son score pour ce match sera considéré 0 point (peu importe les points donnés par l'adversaire).
- Les arbitres se réservent le droit d'attribuer, à tout instant, une pénalité de -30 points à une équipe en cas d'incident découlant du non-respect des règles (comme par exemple, heurter et renverser le robot adverse).

Conclusion des matchs : Lors des phases qualificatives (comportant entre 3 et 5 matches selon la rapidité de l'équipe à passer l'homologation de ses robots), l'équipe qui aura plus de points que son adversaire sera déclarée gagnante.Ensuite, pour le classement global, seront attribués :

- 5 points bonus à l'équipe gagnante
- 3 points bonus à chaque équipe en cas d'égalité
- 1 point bonus à l'équipe perdante
- 0 point à une équipe déclarée forfait

A l'issue des phases qualificatives, les 16 premières équipes au classement global participeront à un tournoi dans les phases finales. Ce tournoi sera à élimination directe. En cas d'égalité ou de double défaite (lorsqu'aucun des robots ne parvient à quitter sa zone de départ), le match

est rejoué immédiatement. Si la situation se reproduit, le gagnant sera désigné selon les points accumulés lors des phases qualificatives. La finale sera jouée en deux manches gagnantes.

2.3 Inscription à la coupe

La validation de notre inscription à la coupe de France de robotique se fait en plusieurs étapes tout au long de l'année. Après avoir pris connaissance du thème 2013 « Happy Birthday » en fin d'année scolaire 2012 et reçu le règlement et les conditions de participation pour l'édition 2013 publiés vers fin septembre 2012, plusieurs jalons doivent être validés comme indiqués sur le site de la coupe :

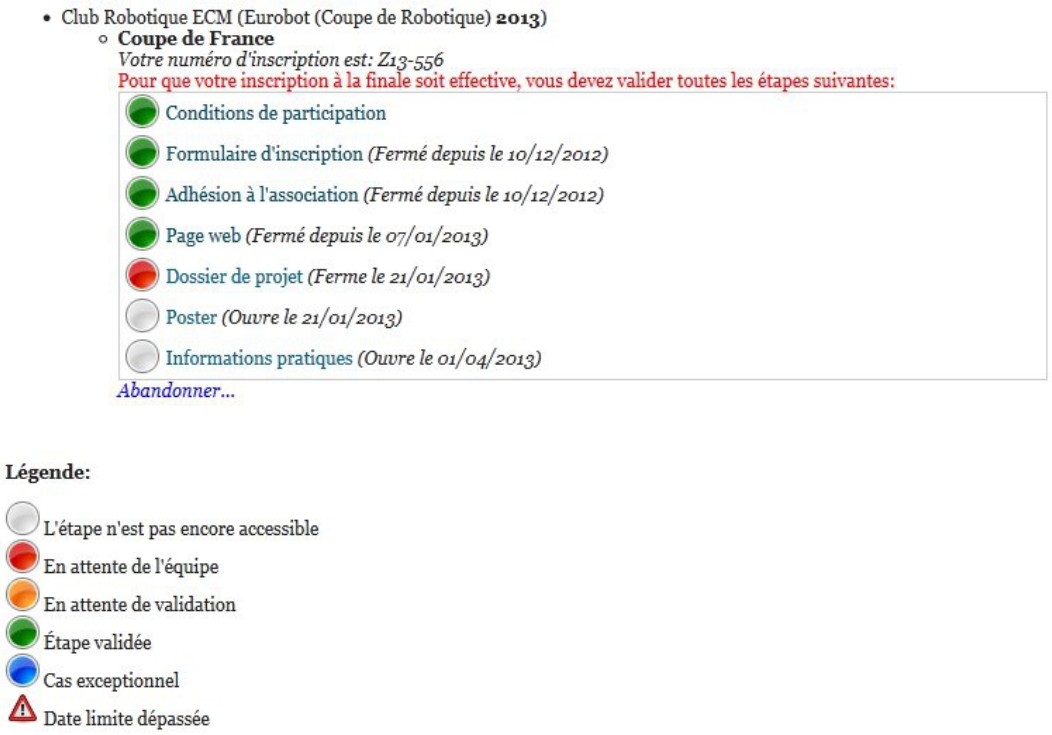


FIGURE 2.5 – Inscription à la coupe

Il convient d'abord d'accepter les conditions de participation, de remplir un formulaire d'inscription détaillant les informations de notre équipe et d'adhérer à l'association (cotisation et formulaire adressés par courrier) avant le 10 décembre.

Nous créons ensuite une page Web implémentée directement sur le site de la coupe - validée pour le 07 janvier, permettant de nous présenter au public et aux autres équipes. Notre page contient une présentation de notre équipe, une présentation de la structure encadrante (Projet Transverse et Association de l'école), de nos robots, de nos idées et de nos choix retenus. Nous rendrons également un dossier présentant les éléments techniques du projet (dimensions, alimentations, capteurs, actionneurs) pour permettre aux organisateurs de vérifier le respect des contraintes imposées par le jeu. Enfin, un poster sera réalisé et imprimé pour présenter notre stand le jour de la coupe.

Chapitre 3

Organisation du PT

3.1 Stratégie de jeu

Avant de concevoir notre robot, et une fois que l'on a identifié clairement les moyens de remporter des points et donc de progresser dans la coupe, il faut mettre en place une stratégie de jeu et développer un robot qui puisse la suivre.

Comme on l'a vu dans la partie précédente, on dispose d'une variété de façons différentes de gagner des points. A vrai dire, un peu trop pour un seul robot considérant qu'il ne disposera qu'un de 90 secondes pour s'exécuter. C'est pour quoi, et puisque l'utilisation de deux robots est autorisée, notre groupe a décidé de concevoir deux robots, et de leur répartir les tâches. Dans un esprit d'optimisation des gains de points et de réduction des éventuels problèmes techniques, il a été décidé que chaque robot s'occupera d'une partie du terrain et sera indépendant de l'autre.

Ainsi, le robot principal aura pour tâche de : (par ordre de priorité)

1. Ouvrir les cadeaux
2. Ramasser des verres et les apporter dans la zone de départ
3. Empiler les verres

La priorité est mise sur « Ouvrir les cadeaux » car c'est la tâche la plus simple à réaliser au niveau des ressources à mettre en jeu, en effet, pour l'accomplir, il suffit de pousser une plaque en bois. Le robot peut donc s'en acquitter en se servant uniquement de ses capacités à se mouvoir et à se repérer dans l'espace, autrement dit, le minimum pour toutes les autres actions possibles (et même pour l'homologation).

Ensuite, « Ramasser les verres » demande de les localiser, de les embarquer, de les transporter et de les redéposer ailleurs. C'est l'action qui rapporte le plus de points, à condition d'empiler les verres. Cette action est bien entendu beaucoup plus ardue car, contrairement aux cadeaux, qui sont des points gagnés définitivement, une pyramide, qui demande déjà beaucoup de doigté à assembler peut s'écrouler (surtout si un robot adverse visite notre zone). Le groupe a opté pour une solution technique qui associe les deux actions, à savoir, déposer les verres directement en pyramide, afin de gagner du temps et de réduire le risque de mauvaise manipulation résultant en l'écroulement de la structure. Cette solution sera détaillée plus amplement dans une partie ultérieure. Cette solution sera effectuée à l'aide d'une pompe à air qui pourra également servir à gonfler le ballon pour accomplir la Funny Action (gonfler le ballon).

Le robot secondaire concentrera quant à lui ses efforts autour du gâteau et aura pour tâche de :

- souffler les bougies
- lancer les cerises dans le panier

Ces tâches, ayant lieu au même endroit, seront effectuées simultanément. Les solutions techniques retenues pour les réaliser seront détaillées dans une autre partie.

3.2 Planification et répartition du travail

Afin de gérer au mieux le temps imparti pour la réalisation de notre projet, nous avons réalisé un diagramme de GANTT en avant projet, et avons tenté de nous y tenir depuis. Nous avons cependant accumulé un retard dans la modélisation et usinage des robots, dû à la complexité des actions à effectuées dans les règles, et qui nous ont demandés un certain temps de réflexion et de prise de décision. Nous pouvons tout de même prévoir de finir les robots dans les délais de la coupe grâce à la marge de manoeuvre prévue que nous laisse notre planning. Nos dates butoirs de la phase 2 : prise de connaissances des règles fin décembre 2012, rapport final et soutenance les 9 et 11 janvier 2013, inscription complète à la coupe en Avril et participation à la coupe du 8 au 11 Mai.

La répartition du travail au sein de notre groupe de PT se fait comme suit : La CAO, la partie mécanique et programmation bas niveau - cartes Arduino - sont assurées par Sara El Afia, Yuan Tian ainsi que Axel Jardin. La programmation haut niveau - carte BeagleBone,et la communication entre les différents niveaux et entre les robots sont sous la responsabilité de Rodolphe Bouley, Jean Feutrie et Jérémy Grand. Enfin, Guillaume Marin et Rodolphe Bouley s'occupent de l'IA, ou intelligence artificielle et du calcul de chemin Adrien Kvaternik des capteurs.

3.3 Ressources du projet et budget

Nous nous assurons les compétences utiles à la réalisation de notre projet grâce à notre formation de base à Centrale Marseille, aux acquis transmis par l'ancienne équipe, et par nos propres travaux et recherches. Sur le plan matériel, l'Ecole Centrale Marseille nous fournit les ressources nécessaires. Nous pouvons ainsi jouir d'un local réservé tout au long de l'année scolaire pour y travailler et y stocker notre matériel.

Le projet E=M6 est donc actuellement entièrement financé par l'école. Nous projetons tout de même la recherche de sponsors dès que notre premier robot sera assemblé. Ci-contre le budget du projet pour cette année 2012-2013 : Le budget présenté en Avant-projet prévoyait 4600 euros de financement, soit 1000 euros de plus. En effet, 3000 euros était prévus pour l'usinage des deux robots. Des solutions bon marché ont été trouvées permettant de le réduire à 800 euros. Cependant, des dépenses supplémentaires, en équipement technique pour le local et en cartes électroniques réduisent cet écart.

BUDGET PT 2012-2013	
Ressources	Coût
Table de jeu	
Bois (dont Gâteau)	100
Peinture	70
Divers (verres, balles)	150
Usinage	
Grand Robot	500
Petit Robot	300
Capteurs	700
Batteries Ni-MH	200
Cartes électroniques	700
Equipement technique	500
Autre composants	400
TOTAL	**3620**

FIGURE 3.1 – Budget

Chapitre 4

La CAO

4.1 La CAO du grand robot

Les tâches du Grand Robot sont de déballer les cadeaux, prendre les verres, et gonfler le ballon à la fin de la partie. Le choix de ces tâches se base sur la répartition de l'espace du plateau de jeu entre les deux robots de telle sorte qu'ils puissent fonctionner indépendamment l'un de l'autre et occuper chacun un espace différent, pour aller chercher le plus de points et éviter les accidents : chocs, « bouchon » à certains niveaux du plateau, etc...

4.1.1 Récupération des verres et formation de colonnes

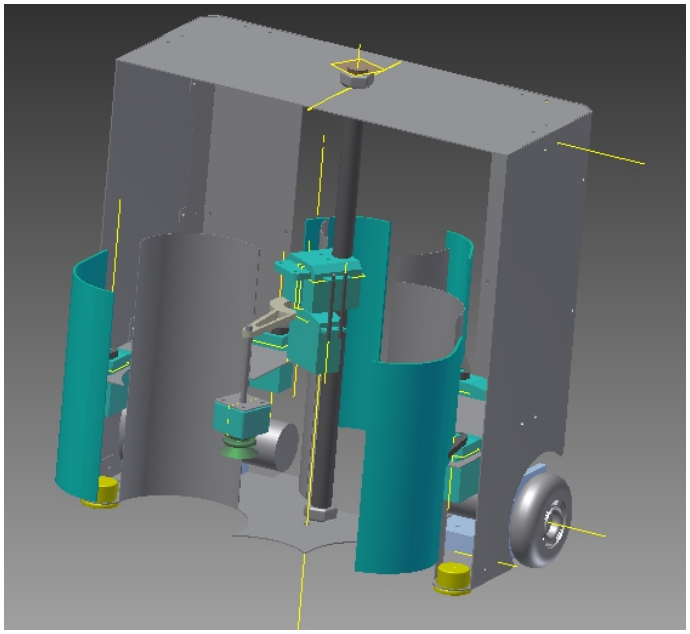


FIGURE 4.1 – Grand robot - vue de face

Pour gagner un maximum de points, on a décidé de poser les verres en les alignant les uns sur les autres sous forme de colonne. Il y a 12 verres sur le terrain du match, 6 verres de chaque côté du plateau (côté proche du joueur et côté proche de l'adversaire).

Dans le grand robot, il existe trois structures cylindriques pour mettre trois verres dans chaque cylindre les uns sur les autres. Si on est suffisamment rapide pour récupérer les 9 verres

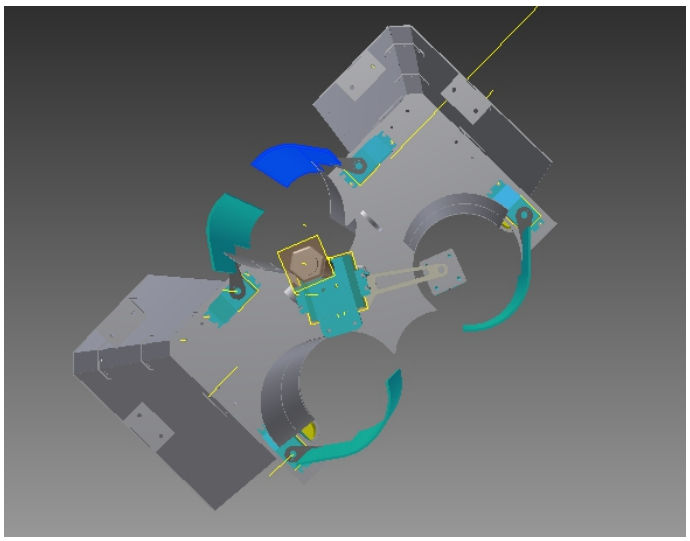


FIGURE 4.2 – Robot - vue de haut

6 de notre côté et 3 de l'autre côté de l'adversaire), on va gagner 3*(4+4*2+4*3)=72 points et surtout en ne faisant qu'un aller pour récupérer les verres et un retour vers la base pour libérer les 3 colonnes ainsi formées dans les cylindres du robot.

On a deux cylindres en avant et un en arrière. L'ouverture et la fermeture de ces 3 cylindres sont contrôlées par les 4 servomoteurs (2 en avant et 2 en arrière). Pour récupérer les verres, on

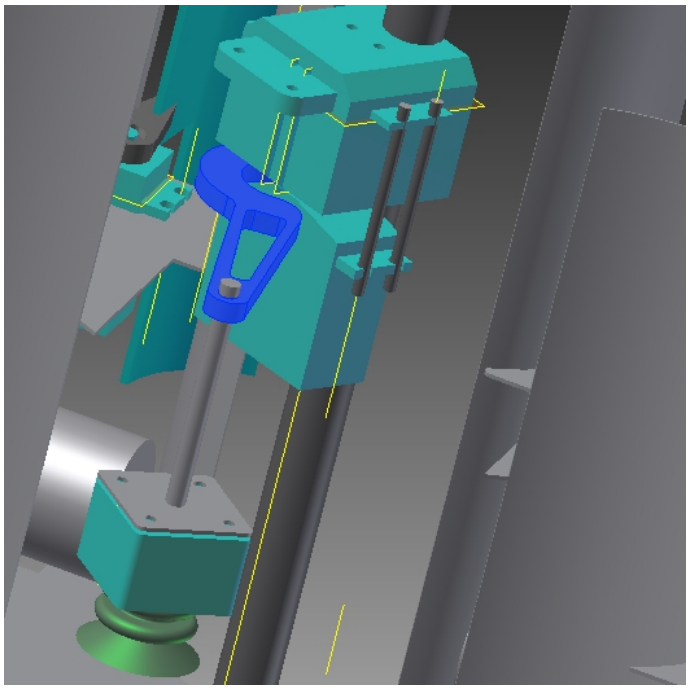


FIGURE 4.3 – Structure de ventouse

utilise une ventouse alimentée par une pompe à air en vidant l'air entre la ventouse et le fond en bois du verre. Cette structure de ventouse a deux degrés de liberté (translation et rotation en direction z) réalisés par deux moteurs. Chaque fois que la ventouse aspire le verre, le module ventouse/moteur va monter et tourner avec le verre et le poser dans l'un des cylindres. Après

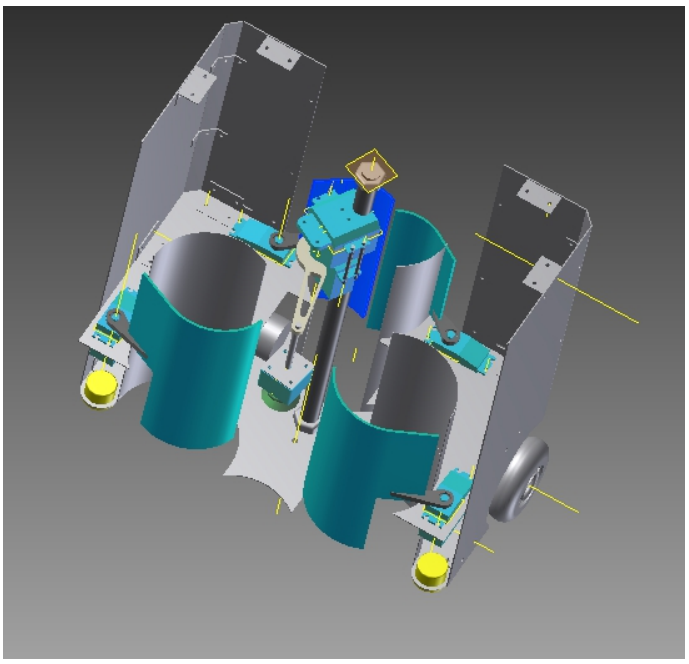


FIGURE 4.4 – Robot avec les cylindres fermés

avoir récupéré les verres, les 4 servomoteurs vont serrer les verres récupérés en fermant les portes. Le robot se dirige ensuite vers la zone de construction, ouvre les portes et dépose les verres en colonnes.

4.1.2 Ouverture des cadeaux

Pour déballer les cadeaux, le robot va marcher parallèlement au bord du terrain où sont alignés les cadeaux. Chaque fois qu'il passe près d'un cadeau de sa couleur, il ouvre la porte contrôlée par le servomoteur et pousse le cadeau. La pièce en bois représentant le cadeau basculera et révèlera la face « cadeau ouvert ».

4.1.3 Réalisation de la Funny Action : gonfler un ballon

A la fin des 90 secondes de la durée de match, il suffira d'alimenter la pompe à air, pour gonfler le ballon connecté à la pompe par un tube. Avant ce moment, le ballon ne pouvait pas se gonfler puisque la pompe était utilisée pour pomper l'air sous la ventouse. Elle libérait ensuite cette petite quantité d'air en relâchant la ventouse à la libération du verre. Cette action réalisable avec le même module que l'action de récupération des verres rapporte 12 points, c'est le maximum qu'on peut marquer en une seule action.

4.2 La CAO du petit robot

Les tâches du Petit Robot sont de rassembler les cerises, de les lancer au-dessus du gâteau dans le panier de points et éteindre les bougies de la couleur de notre zone de jeu et les bougies blanches qui rapportent des points aux deux équipes et un bonus non négligeable de 20 points si elle sont toutes éteintes à l'issue du match. Le choix de ces tâches se base sur la répartition de l'espace du plateau de jeu, comme pour le grand robot, en effet ces tâches-ci se concentrent autour du gâteau et près de la base ce qui permet de commencer à les exécuter très vite au lancement de la partie et sans grands déplacements du robot.

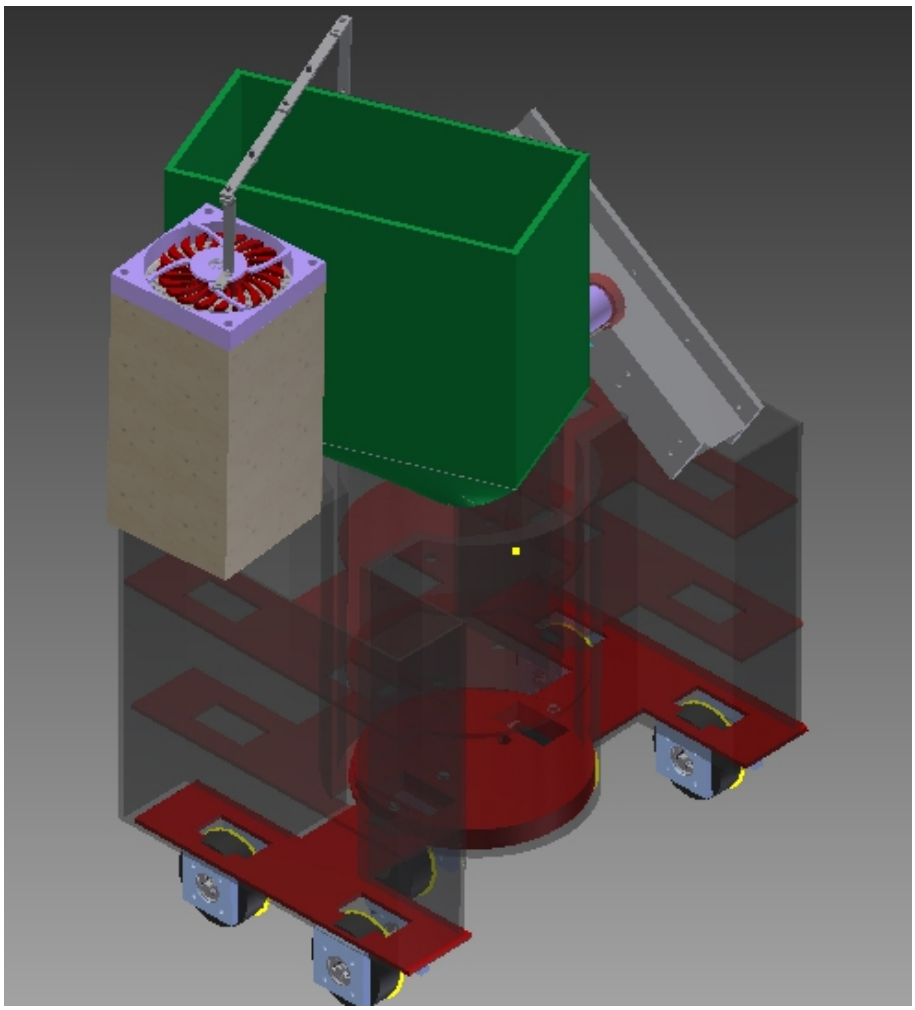


FIGURE 4.5 – Petit robot - vue de biais

4.2.1 Récupération des cerises : balles de ping pong dans des cadres carrés

Pour réaliser cette tâche, nous utiliserons un système d'aspiration qui permettra de recueillir toutes les cerises d'un cadre avec un minimum de mouvement. Le bras mécanique du cylindre d'aspiration est équipé d'un seul servomoteur à grande amplitude angulaire : 320°. Ce qui permettra de baisser le module d'aspiration puis de le remonter au-dessus de la cuve de récupération des cerises/balles de Pingpong. Le fonctionnement est simple : le ventilateur DC sans balai est actionné par un moteur CC à brosse. Dès que le moteur se met en route la rotation des pales du ventilateur aspire tout ce qui ce trouve devant l'ouverture du cylindre. Et de même dès que le moteur s'éteint tout tombe par pesanteur.

Remarque importante : Nous avons pris le parti de récupérer toutes les balles de pingpong en un seul geste pour gagner du temps ,car ce système d'aspiration générale permet de recueillir jusqu'à 12 balles en même temps et il reste plus sûr qu'un sytème à ventouse/aspiration unitaire qui risque de ne pas attraper la balle à certains moments. Cette décision est faite avec une concession stratégique : le fait d'aspirer tout le cadre à cerises fait que nous aspirerons aussi la balle rouge « cerise pourrie ». Mais sachant qu'il n'y ait une ou qu'il y ait plusieurs « cerises pourries » dans le panier à la fin, la pénalité est la même : le total est divisé par deux et sachant que même un système de repérage et extraction des « cerises pourries », (ex : une ventouse avec

FIGURE 4.6 – Servomoteur HerkuleX pour le petit robot

capteur de couleur) a quasiment 1/5 de récupérer une balle pourrie avec les 5 bacs à disposition, nous trouvons la concession justifiée et raisonnable.

4.2.2 Lancer des cerises

Une fois le cylindre d'aspiration plein : en ayant parcouru un cadre et demi de cerises, le petit robot se dirige vers le gâteau et se place face notre panier de points sans éteindre le moteur CC de l'aspiration pour ne pas perdre les balles. Une fois positionné face au gâteau, le robot fait un angle obtus avec le bras du module d'aspiration pour le ramener au-dessus d'un récipient mis en haut du robot pour recueillir les balles. Ce récipient a été conçu avec un trou vers le bas excentré pour favoriser l'« écoulement des balles ». La balle qui passe le trou roule le long de deux fils métalliques épais puis monte le début d'une pente en PVC grâce à son inertie. Cette pente est munie à son entrée d'un tout petit moteur réducteur où une roue tournant haute vitesse la propulse jusqu'en haut de la pente où elle amorce sa chute balistique dans le panier des points.

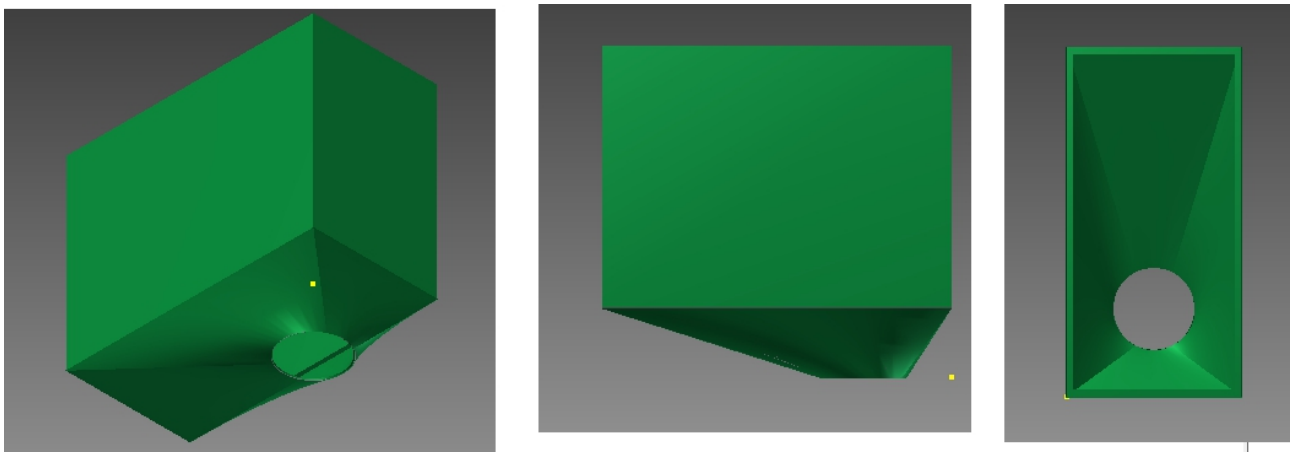


FIGURE 4.7 – Détail du bac d'écoulement des balles de Pingpong

4.2.3 Eteindre les bougies

Pour éteindre les bougies sur le gâteau, nous utiliserons un membre articulé de servomoteurs avec un capteur de couleur pour n'éteindre que les bougies de notre couleur Rouge ou Bleu et les bougies communes blanches. Pour le membre articulé, nous utiliserons un modèle pouvant servir de pied pour robot marcheur pour sa robustesse. Le fonctionnement du capteur utilisé DFRobot TCS3200 (pour les couleurs bleu, rouge et verte) est détaillé en Annexe.

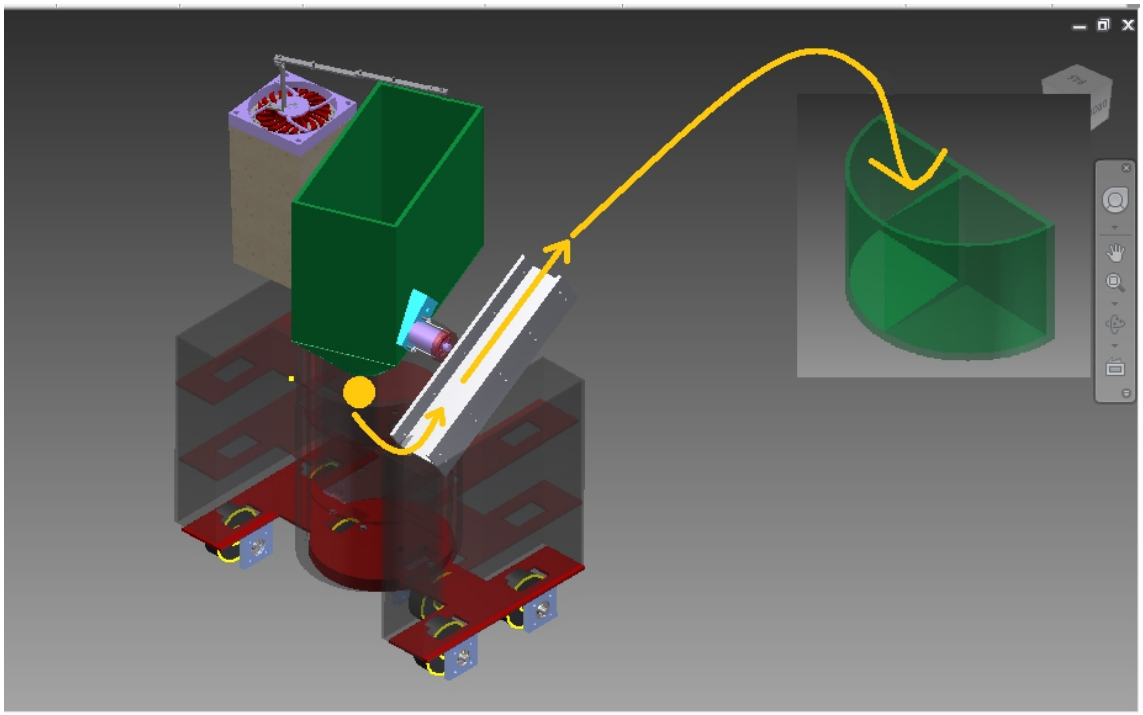


FIGURE 4.8 – Dispositif de lancer de cerises du petit robot

FIGURE 4.9 – Membre articulé motorisé pour éteindre les bougies (jambe Lynxmotion THLEG3-BLK)

Chapitre 5

Alimentation

Cette année, l'alimentation des robots se fait sous 12v via une batterie Rechargeable Ni-MH (BAT-01 12V / 1600 mAH) car nous avons dû renoncer aux batteries Lithium Polymère des anciens groupes car elles sont interdites à la coupe de France de robotique de la Ferté Bernard .

Cependant les cartes doivent être alimentées sous 5v. Pour abaisser la tension d'entrée de 12V à 5V et alimenter les cartes d'asservissement des moteurs et les servomoteurs nous utilisons la Motor Shield Arduino qui se connecte directement à l'Arduino AT Mega 2560, nous permettant ainsi de gagner de l'espace. L'Arduino Motor Shield repose sur un pilote de double pont en H,

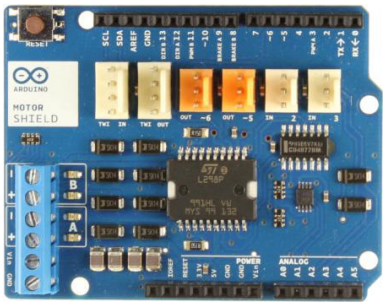


FIGURE 5.1 – Vue d'ensemble du Motor Shield pour l'Arduino AT Mega 2560

le L298, conçu pour alimenter des charges inductives comme relais, solénoïdes, DC et moteurs pas à pas. Il permet de conduire les deux moteurs à courant continu et d'asservir la vitesse et la direction de chacun d'eux indépendamment.

Resumé

- Tension de service : 5V à 12V
- Contrôleur de moteur : L298P Moteurs à courant continu de 2 disques ou 1 moteur pas à pas
- Max courant : 2 A par canal ou 4 a max (avec alimentation externe)
- Détection du courant : 1,65 / A
- Exécution fonction stop et frein

Notre alimentation externe est montée sur la carte en connectant les bornes de la batterie aux les fils qui conduisent l'alimentation aux bornes à vis-Vin et GND, en prenant soin de respecter les polarités.

Entrée et sortie : L'Arduino Motor Shield a deux canaux séparés, nommées A et B, chacun utilise 4 pins de l'Arduino pour contrôler le moteur. Au total, il y a 8 pins en cours d'utilisation

sur cette carte. Nous utilisons chaque canal séparément pour conduire deux moteurs pas à pas droit et les broches supplémentaires sur l'Arduino Motor Shield sont décrites comme suit :

- Borne à vis pour connecter les moteurs et leur alimentation.
- 2 TinkerKit connecteurs pour deux entrées analogiques (en blanc), relié à l'A2 et A3, que nous utilisons pour deux capteurs .
- 2 TinkerKit connecteurs pour deux sorties analogiques (en orange au milieu), connecté au PWM sorties sur bornes D5 et D6, que nous utilisons pour deux servomoteurs.
- 2 TinkerKit connecteurs pour la DHT interface (en blanc à 4 broches), l'un pour l'entrée et l'autre pour la sortie, pour la communication avec notre carte BeagleBone.
- Pour le reste des connections, nous avons accès aux Pins de l'arduino AT MEGA 2560 branchée à l'Arduino Motor Shield car les broches se correspondent sur les deux cartes.
- La carte dispose d'un capteur analogique d'intensité intégré (+++) (0V + 1,65V/A) pour chaque phase/moteur

Connexions des moteurs : Pour conduire nos deux moteurs pas à pas, nous connectons les deux fils de chacun d'eux dans le signe (+) et (-) des bornes pour chacun des canaux A et B vis borniers. Nous contrôlons leurs orientations en affectant HIGH ou LOW aux broches DIR A/DIR B. Nous pouvons contrôler la vitesse en faisant varier les valeurs de cycle PWM A et PWM B aux broches correspondantes. Les broches frein Brake A et frein Brake B mises à l'état HIGH arrêtent net les moteurs plutôt que de laisser ralentir en coupant l'alimentation. Nous pouvons mesurer le courant traversant le moteur à courant continu en lisant les broches SNS0 et SNS1.

Chapitre 6

Asservissement des robots

Pour pouvoir envisager n'importe quelle stratégie de jeu, il est indispensable d'asservir les moteurs en vitesse et position afin de maîtriser le déplacement des robots. Dans cette partie nous allons voir nos outils de mise en place de cet asservissement.

6.1 Matériel mécanique et électronique utilisé et principe de fonctionnement

Nous avons gardé le matériel de l'année dernière à savoir :
- deux moteurs pas à pas
- deux odomètres
- un microcontrôleur Arduino AT MEGA 2560

FIGURE 6.1 – Odomètre utilisé (celui de l'année dernière)

Mais nous avons remplacé les deux hacheurs par notre carte Arduino Motor Shield présenté à la partie 5.

6.2 Odométrie pour deux roues codeuses

6.2.1 Principe

L'odométrie (dead reckoning en anglais) est une technique qui consiste à calculer la position du robot, en cumulant tous les déplacements effectués depuis la dernière position connue. Il s'agit d'une technique de positionnement relative qui est donc peu précise puisque les erreurs de mesure s'accumulent au cours du temps. Par sa nature, on l'associe généralement à l'asservissement du robot, en effet, comme l'asservissement, l'odométrie nécessite de lire les encodeurs moteurs le plus régulièrement possible. On effectue donc les calculs au même moment.

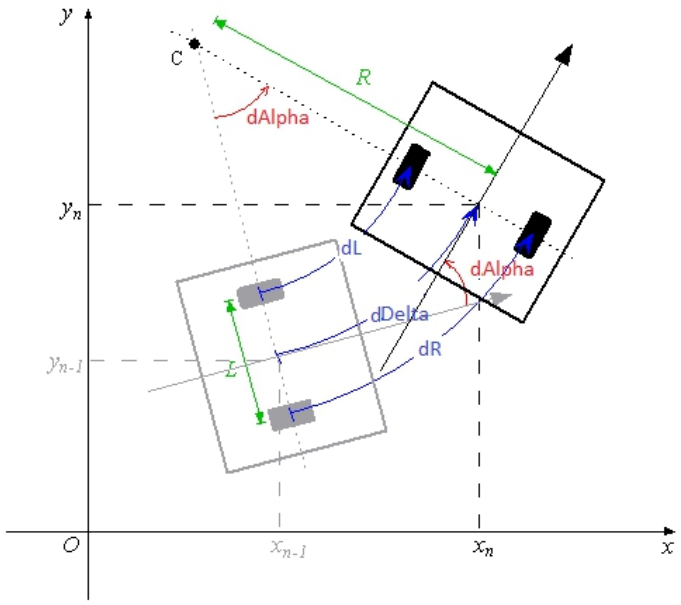


FIGURE 6.2 – Données d'un déplacement unitaire [pas d'échelle : norlement (dL,dR)«L]

6.2.2 Réglage des paramètres

Pour utiliser notre système d'odométrie à d'encodeur sur arbre moteur, il nous faudra calibrer avec soin les paramètres ci-dessous : *Périmètre des roues
Distance entre les deux roues (entraxe)

- Le périmètre des roues influe sur la précision de tous les déplacements du robot (ligne droite et rotation), il faut donc le calibrer en premier. Pour cela, nous réaliserons une série de lignes droites et nous calculerons l'écart entre valeur théorique du déplacement attendue et valeur mesurée.
- La distance entre les deux roues influe sur les rotations. On procède donc de la même manière à mesurer l'écart entre valeur théorique et valeur mesurée lors de rotations sur place du robot.

6.2.3 Risque de patinage du robot

Comme nous n'utilisons pas de balises pour rafraîchir la position du robot par odométrie externe car ces balises restent brouillables comme tout système externe et car les temps de mesure sont souvent excessifs et incompatibles avec les déplacements rapides durant un match de 90 secondes, nous avons donc besoin de pouvoir corriger les erreurs de l'odométrie interne lors des glissements du robot.
Nous envisageons pour cela deux solutions possibles :

- La première solution est de repérer les glissements du robot avec la mise en place d'une nouvelle méthode pour pouvoir néanmoins repérer quand le robot patine : en couplant deux types de capteurs d'odométrie différents, le premier comme l'année dernière monté sur l'axe des roues motrices et le deuxième monté sur une roue folle pour donner l'avancement du robot. Cela peut aussi permettre de repérer un contact avec un obstacle.
- La deuxième solution est de tenter de réduire les erreurs dues glissements en mettant deux roues folles dans l'axe des 2 roues motrices et de les utiliser pour contrôler la trajectoire du robot à la place des roues motrices. Les codeurs seront donc couplés aux roues folles et non aux roues motrices qui peuvent glisser. Nous testons actuellement cette deuxième solution.

6.2.4 Programme en C

Avec l'approximation des petits angles (micro-déplacements),nous pouvons avoir une idée des variations de l'angle et de la position du robot :

```
dAlpha = (dRight-dLeft)/2;   //variation de l'angle
dDelta = (dRight+dLeft)/2;   //variation de l'avancement

//conversion en radian
alpha += dAlpha / entraxeEnTick;

//calcul des décalages selon X et Y
dX = cosf(alpha) * dDelta;
dY = sinf(alpha) * dDelta;

//conversion de la position en mètre
X += dX ;
Y += dY ;
```

FIGURE 6.3 – Programme en C

Pour optimiser ce calcul, nous évitons de recalculer cosinus et sinus à chaque coup pour ne pas surcharger le microcontrôleur. On utilise à la place une approximation avec décomposition de Taylor :

6.3 Asservissement de la vitesse et de la position

6.3.1 Asservissement de la vitesse

Pour l'asservissement de la vitesse nous utilisons la correction PI mise en place par nos prédécesseurs et que l'on retrouve en annexe.

6.3.2 Asservissement de la position

Comme la boucle de position de nos prédécesseurs destabilisée la boucle de vitesse, et que nous mettons en place une nouvelle disposition des roues codeuses pour réduire les erreurs de patinage, nous avons cherché à mettre en place un nouveau modèle d'asservissement de la position.
Pour la disposition des roues codeuses : ce sont deux roues folles disposées sur l'axe des roues motrices, les codeurs ne sont plus couplés directement aux roues motrices (cf.6-2-3 2ème solution). Nous envisageons deux modèles d'asservissements de position, inspirés des travaux de l'équipe de robotique RCVA.
Remarques : dans ce qui suit, Les coordonnées du robot sont prises au milieu de l'essieu des roues, l'orientation représentant l'angle orienté formé par l'axe longitudinal du robot par rapport à l'axe des Y.

Asservissement de position principe 1

Chaque roue codeuse est asservie à une consigne de position, sa position réelle étant contrôlée par le codeur incrémental associé, suivant le schéma fonctionnel suivant :

La consigne représente la valeur désirée de la position, le codeur mesurant la position réelle. L'écart entre ces 2 valeurs est appliquée à un correcteur PID (Proportionnel Intégral et Dérivé) qui calcule la commande appliquée au moteur par l'intermédiaire de l'amplification de puissance.

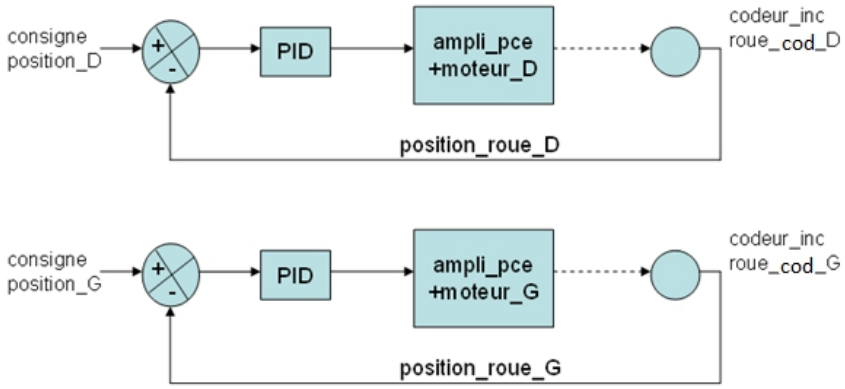


FIGURE 6.4 – Schéma bloc des deux boucles fermées (source RCVA)

Bien entendu l'asservissement est échantillonné car géré numériquement : pour une fréquence d'échantillonnage de 200 Hz par exemple, il y a contrôle d'asservissement toutes les 5 ms. Pour les valeurs des coefficients du correcteur nous nous basons sur les valeurs de Ziegler et Nichols cf. Annexe. Imaginons les 3 figures imposées suivantes :

- Robot à l'arrêt : Il suffit d'appliquer des consignes de position constantes. Le robot est alors immobile et asservi en position, quelle que soit la perturbation qui lui est appliquée (par exemple la poussée d'un robot adverse, avec quand même certaines limites)
- Robot se déplaçant en ligne droite : Il suffit d'appliquer 2 consignes égales en forme de rampe c'est-à-dire proportionnelles au temps. (On ignore pour l'instant les phases d'accélération et de freinage)
- Robot en rotation symétrique : si on ignore également les phases d'accélération et de freinage, il suffit d'appliquer 2 consignes de signes opposés en forme de rampe.

Le rôle du processeur, en fait du programme associé est donc double : Il doit générer les consignes de position et assurer l'asservissement avec une correction PID (toutes les 5 ms).Jusque là, rien d'original et une solution matérielle classique consiste à utiliser le composant spécialisé LM629 Precision Motion Controller qui permet de décharger le processeur central de cette double tache. Solution qui a le mérite de la simplicité avec toutefois des limites en performance.

Asservissement de position, principe 2

Dans ce cas de figure les 2 consignes de position sont remplacées par une consigne de distance et une consigne d'orientation conformément au schéma fonctionnel suivant : On impose une consigne de distance et une consigne d'orientation au robot La ligne en pointillé représente la trajectoire curviligne suivie par le robot depuis sa position initiale A.

- position_D=reset du codeur roue droite
- position_G=0, reset du codeur roue gauche
- $\theta = \theta_0$, orientation du robot

L représente la distance parcourue depuis la position de départ jusqu'à l'instant présent. $L = \frac{1}{2}.(position\_D + position\_G)$ θ représente l'orentation du robot à l'instant présent. $\theta = \theta_0 + position\_D - postition\_G$ Comparons les 2 principes d'asservissement :

- Principe1 : Les 2 grandeurs position_D et position_G sont asservies à leurs grandeurs de consigne respectives, consigne_position_D et consigne_position_G.
- Principe2 : Les 2 grandeurs L et θ sont asservies à leurs grandeurs de consigne respectives, consigne_distance et consigne_orientation.

Imaginons de nouveau quelques figures imposées mais cette fois avec le principe 2 :

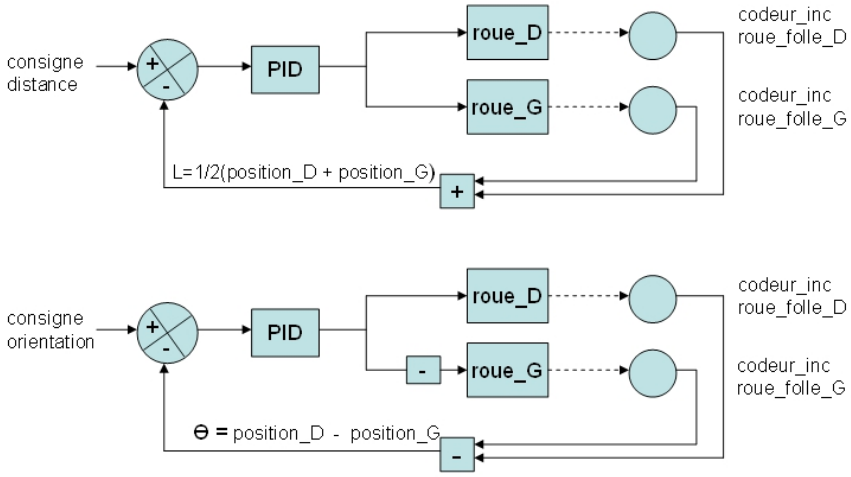


FIGURE 6.5 – Schéma bloc des deux nouvelles boucles fermées (source RCVA)

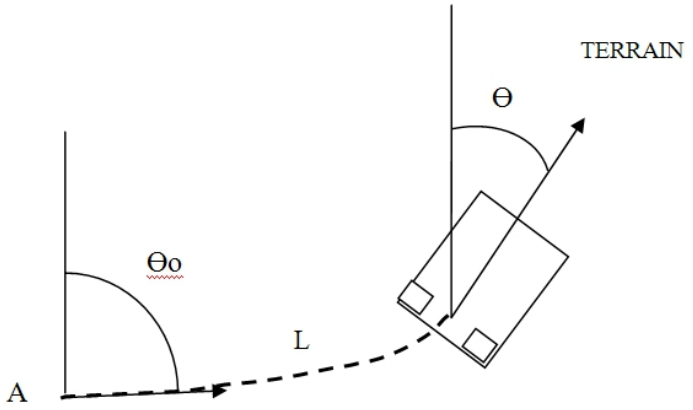


FIGURE 6.6 – Représentation d'une trajectoire curviligne quelconque (source RCVA)

Mis à part une gestion simplifiée des trajectoires curvilignes à rayon constant (Virage), ici nous ne voyons pas encore vraiment l'intérêt du principe 2 par rapport au principe 1. Mais l'intérêt apparait dans l'exemple suivant : On propose de se déplacer du point A vers le point B puis vers C, puis D, ... avec une orientation initiale θ_0 quelconque.

Le robot présente une orientation initiale θ_0
1ère solution : Alignement vers le point B par rotation $(\beta_0 - \theta_0)$Puis déplacement rectiligne noté d_0 suivant le segment de droite AB
2ème solution : Trajectoire curviligne de A vers B (pointillé) obtenue par l'algorithme suivant :
repérer (tant que d non égal à 0)
calcul de d et β;
$consigne\_distante = k1.t$;
$consigne\_orientation = \beta$;
$asservissement\_vers\_valeurs\_de\_consigne()$;
L'idée est d'asservir à chaque instant d'échantillonnage (point M sur la figure) l'orientation θ du robot à l'orientation β du point B. Ainsi le robot décrit une trajectoire curviligne en semblant attiré comme par aimantation vers le point B. On n'impose pas la forme de la trajectoire mais juste le passage par le point B puis en généralisant par le point C puis D, etc.
Remarque : Les calculs de la distance restante d et de l'orientation β du point destination

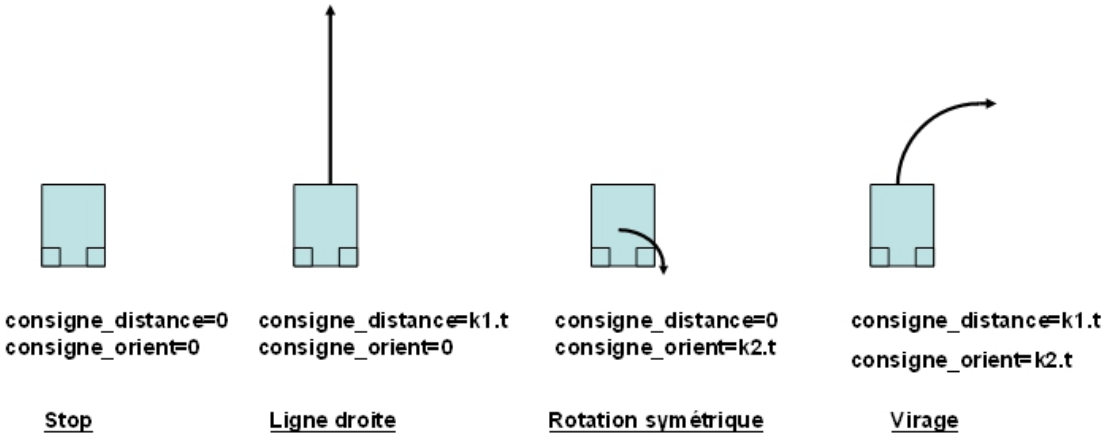


FIGURE 6.7 – Données de déplacements simples (source RCVA)

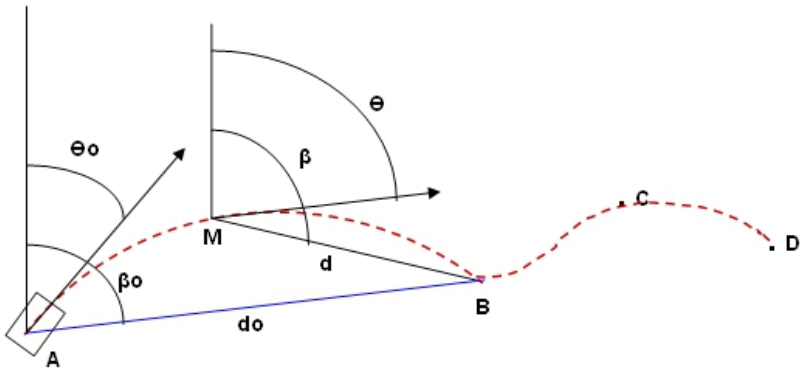


FIGURE 6.8 – Exemple d'une trajectoire plus complexe (source RCVA)

supposent que le robot connaisse à tout instant sa position sur le terrain c'est-à-dire les valeurs de ses coordonnées X,Y,θ (cf.annexe Localisation)

6.3.3 conclusion

Avec l'asservissement de la vitesse du robot et de sa position, ainsi que la connaissance de sa position par calcul de localisation, nous avons en main tous les éléments nécessaires afin que le robot puisse exécuter toutes les tâches stratégiques données par la carte Maître à l'Arduino.

Chapitre 7

Les capteurs

Nous disposons cette année de deux types de capteurs : les capteurs infrarouges et les capteurs ultrason. Ces capteurs jouent tous les deux les rôles d'émetteur et de récepteur. Nous avons effectué une étude plus poussée que l'an dernier concernant les possibilités qu'offrent ces capteurs, ce qui nous permettra de les utiliser de façon optimale.

7.1 Capteurs infrarouges

Les capteurs infrarouges GP2D12 sont dotés d'un émetteur et d'un récepteur infrarouge. Le principe est le suivant : l'émetteur envoie une onde IR qui est réfléchie sur un obstacle puis récupérée par le récepteur. En fonction du temps de retour de l'onde, il en sort une tension qui nous permet de déterminer à quelle distance se situe l'obstacle.

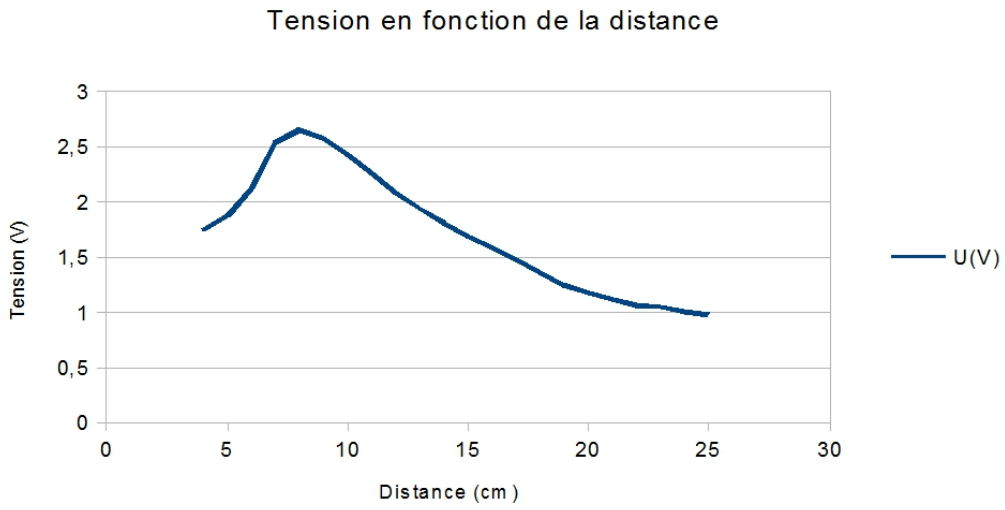


FIGURE 7.1 – Tension en fonction de la distance

On voit que la tension n'est pas une fonction monotone de la distance. Ceci pose un problème car nous utilisons la tension pour en déduire la distance. Nous verrons l'impact de ceci dans le paragraphe sur la détection des obstacles.

Nous avons effectué des tests pour déterminer où le capteur était sensible, c'est-à-dire dans quelle zone il était capable de voir un obstacle. Pour cela, on a placé un obstacle fin (tige

métallique) à différentes coordonnées prises par rapport au centre du capteur. On a ensuite mesuré la tension de sortie du capteur. La figure ci-dessous illustre ces résultats.

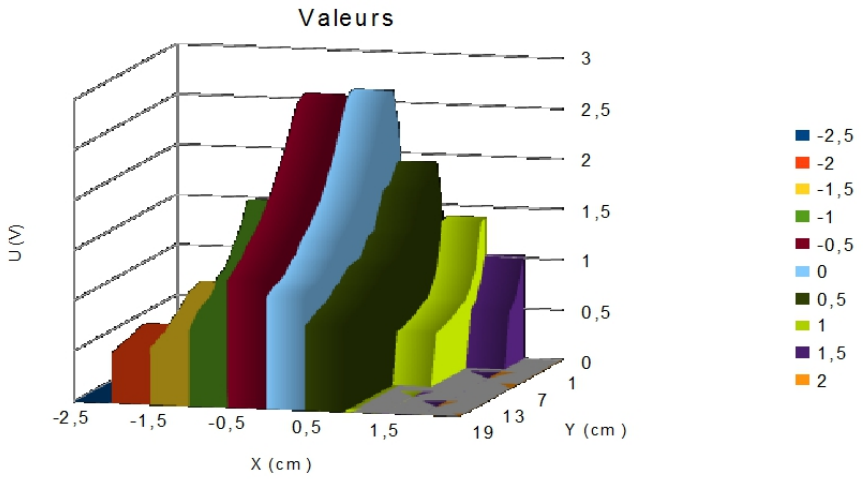


FIGURE 7.2 – Tension en fonction des coordonnées

La conclusion que l'on peut tirer de ces données est que le capteur IR est très directif. En effet, il ne détecte un obstacle seulement si celui-ci est placé directement face à lui. Toutefois, ceci ne constitue pas un inconvénient majeur, car les obstacles que nous rencontrerons seront beaucoup plus larges que la tige métallique utilisée pour les mesures.

FIGURE 7.3 – Capteur infrarouge GP2D12

7.2 Capteurs ultrasons

Les capteurs ultrasons SRF05 fonctionnent sur le même principe que les capteurs IR. Cependant, la tension de sortie est une impulsion de durée proportionnelle à la distance de l'obstacle.

Il suffit de diviser cette durée exprimée en microseconde par 58 pour obtenir la distance en cm.

Contrairement aux capteurs IR, ils sont capables de détecter un obstacle situé à quelques centimètres (environ 3cm), avec une précision beaucoup plus appréciable.

Cependant, de même que pour les capteurs IR, ils sont très directifs. Pour des raisons de dimension des obstacles, ceci n'est pas un problème.

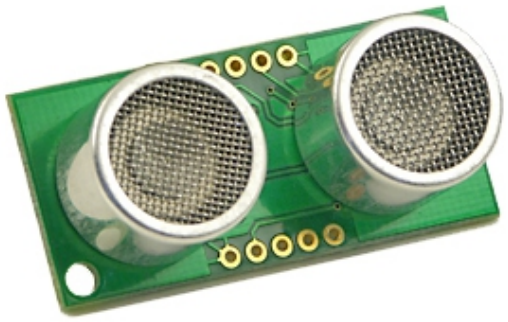


FIGURE 7.4 – Capteur ultrason SRF05

7.3 Détection des obstacles

Pour décider de la présence d'un obstacle, il est nécessaire de se baser sur un critère préalablement choisi. Ce critère est du type 'si le capteur renvoie une tension supérieure à une tension seuil, il y a un obstacle' pour les capteurs IR et 'si la distance renvoyée par le capteur est inférieure à une distance seuil, il y a un obstacle' pour les capteurs ultrasons.

Le capteur IR pose ici un problème, dû à la non-monotonie de la courbe Tension=f(Distance). En effet, si l'on prend une tension seuil de 2.4V, l'obstacle n'est détecté qu'entre 7 et 10cm (voir Figure 3). Il faut garder en tête que l'on veut éviter des obstacles, mais tout de même être capable de les approcher. L'idéal serait de les approcher suffisamment tout en ayant une distance minimale faible. Malheureusement, ceci n'est pas possible, il est nécessaire de faire un compromis entre les distances maximale et minimale de détection, dû à la forme de la courbe. On règlera donc la distance maximale à 10cm et il restera un risque, celui que l'obstacle soit trop près du capteur pour être détecté (distance inférieure à 7cm). Il faudra donc tout faire pour éviter ce cas lors des matchs.

Le cas des capteurs ultrasons est beaucoup plus simple, car on a directement accès à la distance séparant l'obstacle du capteur. Ici donc, pas de compromis, on se retrouve dans le cas du capteur IR, avec la distance minimale ramenée vers 0cm : il n'y a pas de risque que l'obstacle soit trop près pour être détecté. On choisira la distance seuil à 10cm.

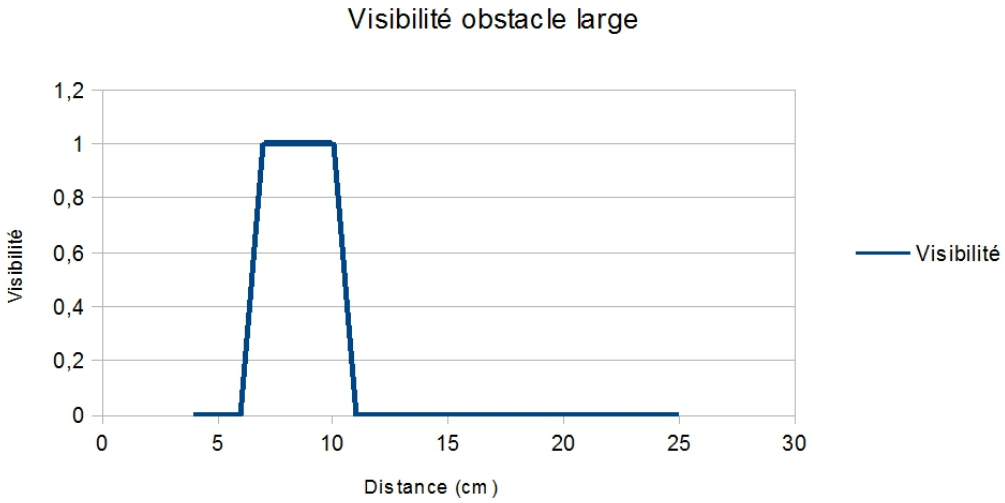


FIGURE 7.5 – Visibilité du capteur IR (seuil à 2.4V

Compte-tenu de ces observations, nous allons placer 2 capteurs ultrasons à l'avant et un capteur IR à l'arrière pour chaque robot. Les côtés ne posent pas de problèmes à priori. Le code déterminant la présence d'obstacle a été bien avancé et des tests sur le terrain permettront de placer les capteurs de façon plus précise.

Chapitre 8

Cartes et Communication

Pour contrôler les actions du robot, la partie électronique programmée du robot doit gérer : la commande des moteurs et des actionneurs, la réception des informations transmises par les capteurs, et l'Intelligence Artificielle pour le traitement des données et la prise de décision. Le robot nécesite donc des cartes électroniques (Arduino) servant de microcontrôleur et d'une carte gérant l'Intelligence Articielle.

8.1 La BeagleBone : le maître

La communication entre tous les composants du robot est nécessaire au bon fonctionnement du robot et au respect des règles. Le robot a donc besoin d'un cerveau pour gérer l'ensemble des composants et des cartes Arduinos. Nous avons choisi d'utiliser une BeagleBone. Cette carte a l'avantage d'offrir une rapidité de calcul très performante et une connectivité très diverse : Ethernet, USB, Série... C'est l'équivalent d'un micro-ordinateur. La distribution Linux Angstrom est installée et permet d'installer tous types d'applications. Elle permet donc d'installer une intelligence artificielle où les données traitées pourront automatiquement être envoyées aux Arduinos. Ci-dessous ses principales caractéristiques :

1. Processeur
   - 720MHz super-scalar ARM Cortex-A8
   - 3D graphics accelerator
   - ARM Cortex-M3 for power management
   - 2x Programmable Realtime Unit 32-bit RISC CPUs
2. Connectivité
   - USB client : power, debug and device
   - USB host
   - Ethernet
   - 2x 46 pin headers
   - 2x I2C, 5x UART, I2S, SPI, CAN, 66x 3.3V GPIO, 7x ADC
3. Software
   - Cloud9 IDE on Node.JS with Bonescript library
   - 4GB microSD card with Angstrom Distribution

La communication entre la BeagleBone et les Arduinos se fait par le biais des ports séries appelés UART dans la description. Afin d'établir une bonne communication entre ces cartes, il était nécessaire de configurer la BeagleBone et d'installer un compilateur C/C++ afin de pouvoir programmer la communication en C. Avec le sytème Linux, la programmation de la communication en C est assez facile. On associe le nom du port série sous Linux à un fichier, l'envoie et la réception de données s'effectue alors avec les commandes classiques d'écriture et de lecture d'un fichier.

Dans le but d'obtenir un robot totalement autonome lorsqu'il est alimenté, nous avons modifié le fichier nommé 'profile' dans /etc du système d'exploitation Angstrom afin que la BeagleBone conserve la configuration pour la communication entre elle et la Arduino. Voici les lignes ajoutées : La première ligne permet de configurer les paramètres du port de réception. La deuxième

```
echo 28 > /sys/kernel/debug/omap_mux/uart1_rxd
echo 0 > /sys/kernel/debug/omap_mux/uart1_txd
stty -raw -F /dev/ttyO1
stty cbreak -F /dev/ttyO1
./start
```

FIGURE 8.1 – Code ajouté dans le fichier /etc/profile

ligne configure les paramètres du port de transmision. Les deux lignes suivantes permettent de configurer le fichier ttyO1 utilisé pour la réception et l'émission des caractères en C. Enfin la commande ./start permet de lancer le programme au démarrage.

FIGURE 8.2 – Beaglebone

8.2 L'esclave : Arduino - structure d'automate

8.2.1 Arduino

Nous utilisons comme microcontrôleur, pour contrôler les actionneurs et les capteurs, la carte Arduino ATmega 2560 qui convient parfaitement en termes de rapidité de calcul et d'entrées/sorties nécessaires.

- Microcontrôleur ATmega2560
  - 16MHz
  - 256KB de mémoire FLASH
  - 8KB de SRAM

  - 4KB de EEPROM
- Connectivité
  - Interface USB ATmega8U2
  - 54 broches d'entrées/sorties numériques, dont 14 avec sortie PWM
  - 16 broches d'entrées/sorties analogiques
- Dimensions : 101.6 mm * 53.3 mm

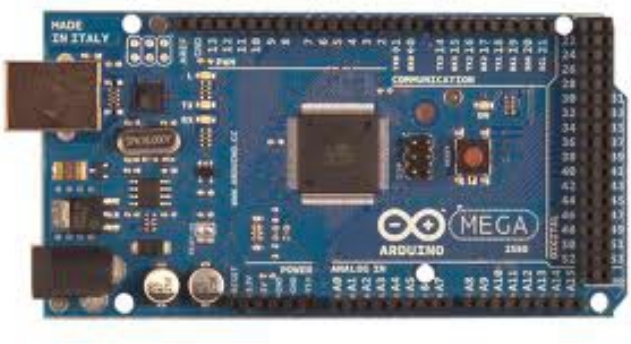


FIGURE 8.3 – Arduino

Du point de vue de la programmation de la carte Arduino, la communication avec la BeagleBone se fait grâce à une structure d'automate.

8.2.2 La structure d'automate

Un automate correspond à une machine virtuelle décomposée en états, et pouvant passer successivement d'un état à un autre selon certaines conditions préalables.

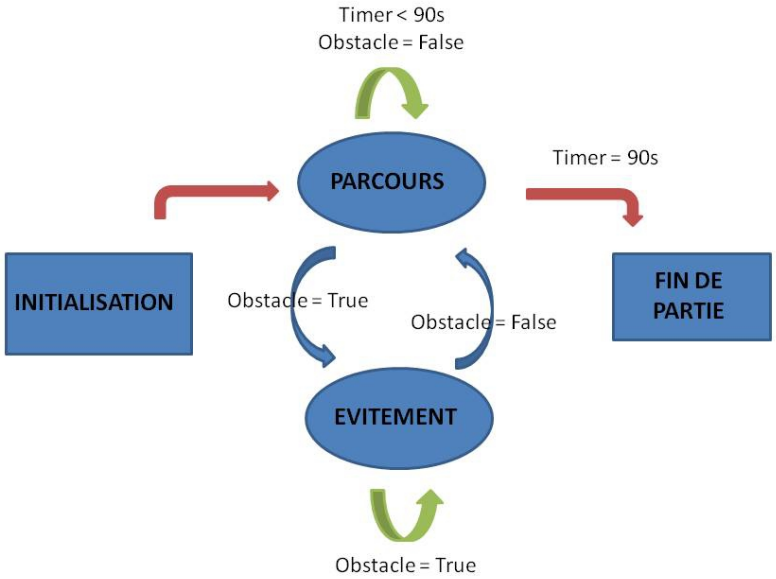


FIGURE 8.4 – Schema automate Arduino

Dans notre cas, l'état d'initialisation permet de configurer tous les ports et toutes les variables. Puis, l'automate se met dans l'état «Esclave» et écoute les instructions de la BeagleBone

tout en lui transmettant les données de positionnement et de capteur. Lorsqu'il détecte un obstacle, il passe dans l'état «Evitement», n'obéit plus à la BeagleBone et tente un évitement. Il revient ensuite dans l'état «Esclave». Un timer le place dans l'état «Arrêt» au bout de 90 secondes, en fin de partie (ou ballon s'il s'agit de la carte qui contrôle le gonflement du ballon).

8.3 Protocole de communication

La BeagleBone et les Arduino communiquent via leurs ports séries : le port de réception de l'Arduino branché sur un port de transmission de la BeagleBone et vice-versa. Le nombre de messages à transmettre étant relativement limité («Avance », « Recule », « Tourne à droite », « Prends le verre », « Stop », ...), nous avons choisi de communiquer à l'aide de la table des caractères (variables char).

BB vers A		A vers BB	
Message	Caractère	Message	Caractère
Avance	a	J'ai avancé	A
Recule	r	J'ai reculé	R
Tourne à droite	d	j'ai tourné à droite	D
Tourne à gauche	g	j'ai tourné à gauche	G
Stop !	s	je me suis arrêté	S

FIGURE 8.5 – Tableau récapitulatif du protocole de déplacement

Un accusé de réception est systématiquement renvoyé lorsque la BeagleBone transmet un ordre à une Arduino pour s'assurer de la transmission des informations.

Chapitre 9

Calcul de chemin

9.1 Compréhension

On a étudié le code du calcul de chemin de l'année précédente seul et avec l'aide du créateur du code (Gabriel Plassard 3A ancien membre du PT Robot) pour comprendre le fonctionnement de cet algorithme. On utilise donc le calcul de chemin basé sur la méthode A*. Le terrain de jeu est modélisé par un quadrillage avec des cases de 1cm de côté. Le robot peut virtuellement se déplacer de case en case.

9.2 Traçage des obstacles

Le traçage des obstacles permet de simuler les obstacles pour le calcul de chemin, pour cela on définit les zones où le robot peut aller ou non, grâce au quadrillage. On a réalisé l'adaptation de la fonction traceObstacles() permettant de tracer les obstacles fixes en modifiant le code pour prendre en compte le terrain de jeu de cette année. On a aussi réalisé la création d'une fonction trace(Plateau) qui permet de tracer les objets non fixes comme les assiettes (dépendantes du placement initial) et des verres qui peuvent bouger durant le match. De plus, dans le calcul de chemin on a pris en compte la taille du robot, ce qui change la trajectoire, comme on peut le voir sur la simulation ci-dessous.

9.3 Communication avec la carte Arduino

Une fois le calcul de chemin effectué, il convient de communiquer le résultat à la carte Arduino, qui permettra alors le déplacement réel du robot. Afin d'obtenir la communication la plus simple et efficace possible, nous avons créé un programme supplémentaire, appelé lancerConversion(). Le fonctionnement de ce programme est simple : il récupère le tableau fourni par le calcul de chemin et le traduit en un tableau de caractères.

Ce dernier est constitué de quatre caractères différents : a (signifiant avance), g (signifiant tourne à gauche puis avance), d (signifiant tourne à droite puis avance) et t (signifiant fais demi-tour et avance). Afin de faciliter ce tableau ainsi que la lecture, nous avons ajouté un code de direction, c'est-à-dire que nous avons inclus les quatre points cardinaux de façon arbitraire. Par défaut, le Nord sera la direction dans laquelle le robot regardera au début de chaque match.

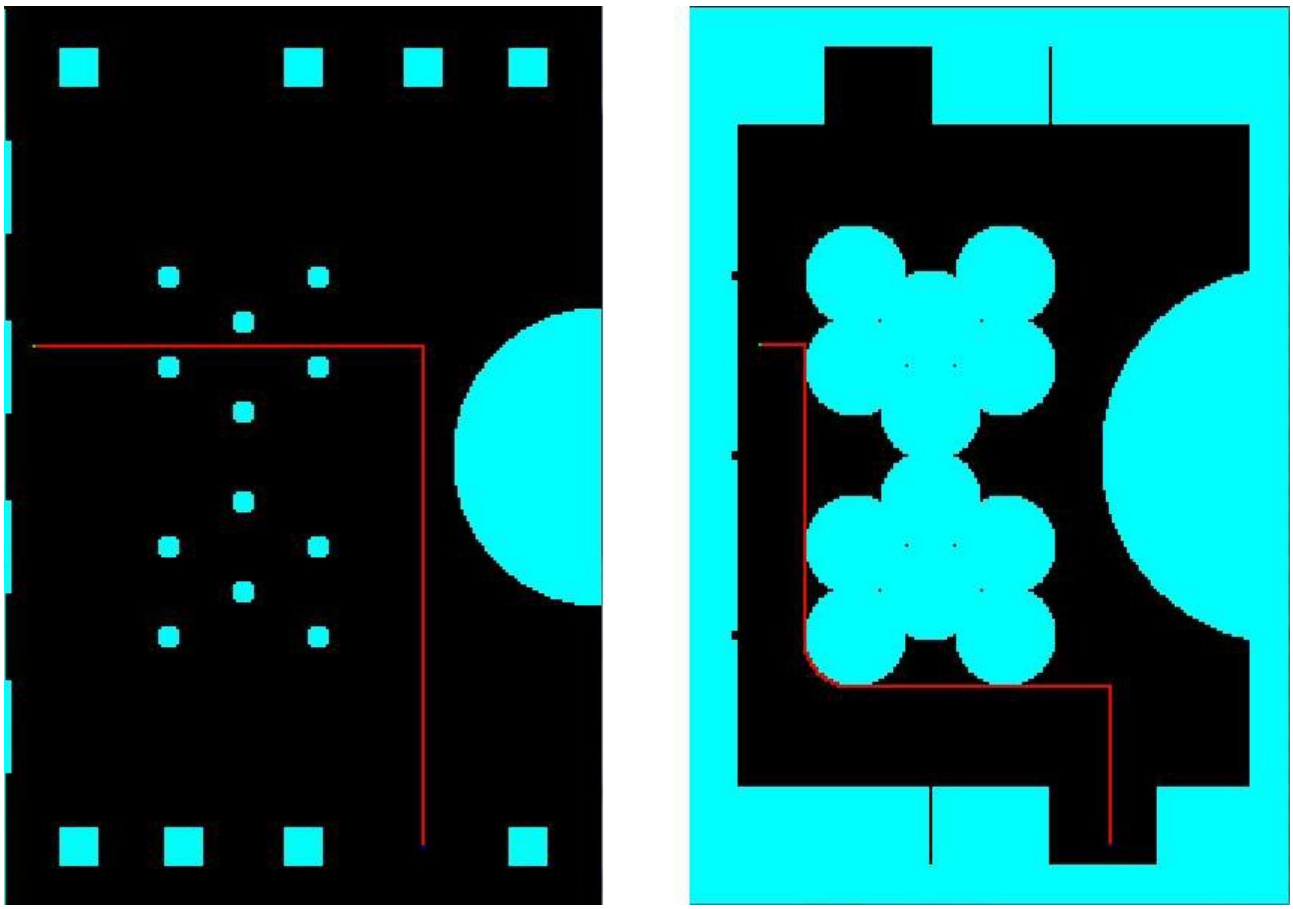


FIGURE 9.1 – Représentation du chemin calculé pour aller ouvrir un cadeau depuis une position initiale sans prendre en compte la taille du robot (à gauche) et en prenant en compte la taille du robot (à droite).

9.4 Utilisation dans l'IA

Le calcul de chemin est une partie importante de l'intelligence artificielle. Pour la simulation, elle sert à savoir quel chemin adopter pour aller d'un point A à un point B de la façon la plus rapide en évitant les obstacles, avec la fonction passage() qui ressort les points par lequel le robot doit passer pour atteindre son objectif. De plus, le calcul de chemin est utile dans la prise de décision de l'intelligence artificielle, comme expliqué ci-après.

Chapitre 10

L'Intelligence Artificielle

La réalisation de l'Intelligence Artificielle est divisée en 3 étapes :

10.1 Modélisation du terrain de jeu

Cette étape consiste à créer les différentes parties du terrain dans l'intelligence du robot. Pour cela on crée des classes pour chaque type d'objets avec des caractéristiques qui leur sont propres. Par exemple, on a créé une classe Cadeau représentant les cadeaux avec comme caractéristique un point pour sa position, un booléen pour savoir s'il est ouvert ou non et une chaîne de caractère pour modéliser sa couleur. De même, on crée les classes suivantes : Verre, Cerise, Assiette, Bougie, Panier, Robot, Gros Robot, Petit Robot, Zone et Plateau. Ainsi, tout le terrain sera modélisé.

Ensuite, on modélise les actions basiques (avancer et tourner pour les robots par exemple) et les relations entre les objets (par exemple, un robot ramasse un verre) avec des fonctions.

10.2 Réalisation de l'IA avec un Système de classeurs

Principe : Pour chaque action on réalise une fiche action dans laquelle on met les informations pour réaliser l'action, les informations de réalisabilité et un score associé à chaque fiche permettant d'estimer la valeur de l'action. Dans un premier temps, on teste toutes les fiches pour voir si elles sont réalisables ou non. On en récupère un ensemble de fiches actions activables. Avec ces fiches, on calcule le score associé dépendant de plusieurs paramètres comme la distance, le temps d'action, le nombre de points gagnés... Puis on sélectionne la fiche avec le plus grand score (étant la fiche estimée la meilleure dans la situation actuelle). Enfin, on lance la meilleure action, que l'on peut recalculer au cas où il y ait un imprévu (comme un nouvel obstacle (robot adverse), la disparition des objectifs (verre ramassé par un autre robot,...)...).

Réalisation : Création d'une classe mère FicheAction (avec comme paramètre le robot sur lequel porte l'IA ainsi que le Plateau. Elle contient aussi des fonctions virtuelles qui seront communes à toutes les classes filles :
- une fonction Estactivable() qui dit si l'action est réalisable ou non
- une fonction Score() qui donne le score associé à la fiche
- une fonction Lanceaction() qui lance l'action (déplacement robot, etc...)

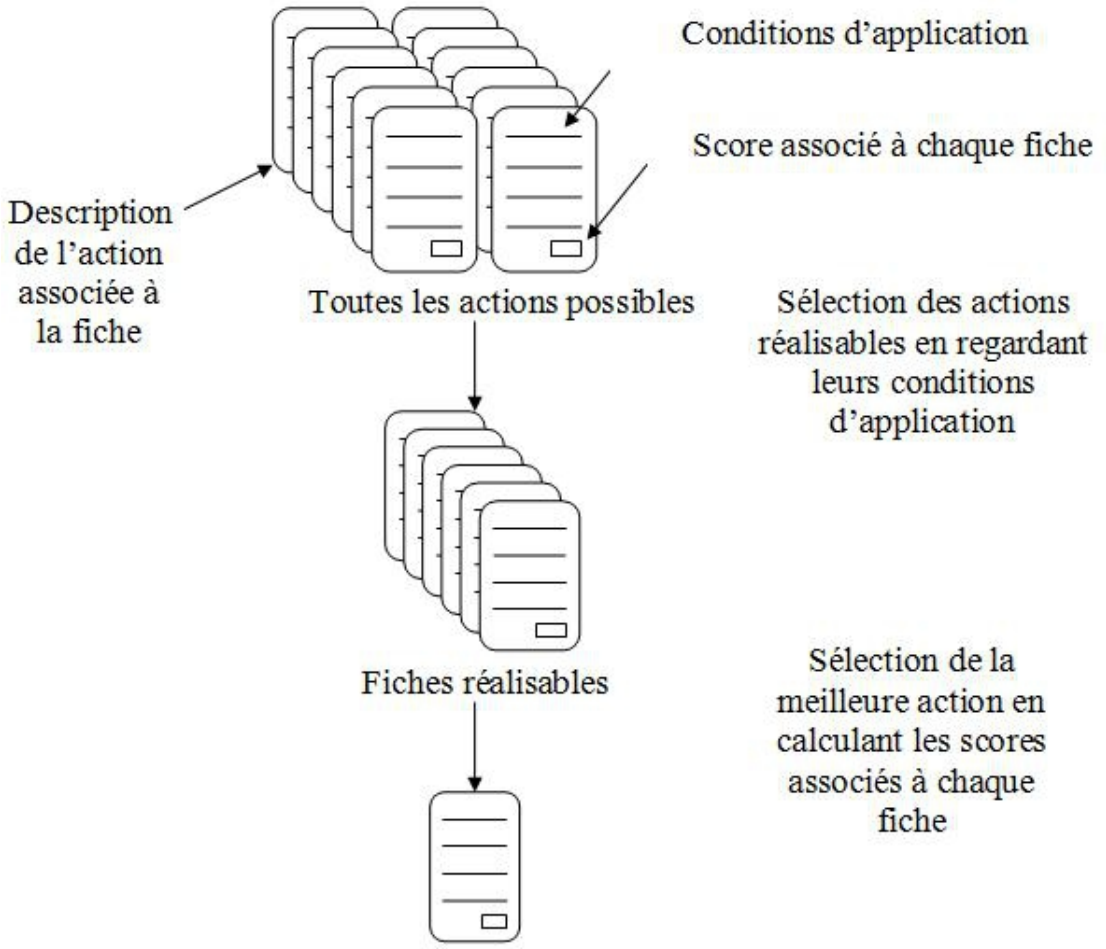


FIGURE 10.1 – Schéma du principe du système de classeurs

Pour chaque type d'action, on crée une classe avec les fonctions virtuelles héritées de la classe mère et des attributs qui leur sont propres. Par exemple, FAOuvrirCadeau contient les même fonctions que la classe mère, avec un attribut supplémentaire un Cadeau, correspondant au Cadeau à ouvrir. Cependant, les fonctions récupérées de la classe mère sont différentes dans chaque fiche action. En effet, par exemple, les conditions pour réaliser la fiche Ouvrir un Cadeau ne sont pas les mêmes que pour réaliser la fiche Ramasser un Verre.

Enfin, on crée une classe IA dans laquelle on met en attribut un tableau de FicheAction, qui permet de regrouper toutes les actions existantes. Cette classe est composée des fonctions suivantes :

- Ajout de FicheAction qui permet d'ajouter des fiches actions dans l'IA lors de la création du terrain.
- Sélection des actions réalisables qui permet de récupérer un tableau de fiches actions réalisables.
- Sélection de la meilleure action qui choisit à partir d'un tableau d'action la meilleure action en fonction des scores de chaque fiche.
- Lancer meilleure Action qui permet de déclencher l'action.

On retrouve les étapes du schéma précédent dans les fonctions de la classe IA, avec les différentes sélections.

10.3 Optimisation des fonctions de calcul des scores

La forme générale d'une fonction de calcul de score est un nombre proportionnel aux points que l'on gagne en réalisant l'action et inversement proportionnel au temps qu'il faut pour réaliser l'action. Pour optimiser l'intelligence artificielle, on améliore ces fonctions de calcul de score en testant et simulant des matchs de 90 secondes pour déterminer les coefficients pour augmenter le nombre de points obtenus à la fin et ainsi les chances de victoires.

Conclusion

L'édition 2013 de la coupe de robotique à laquelle nous participons est une édition particulière qui fête les 20 ans de cette prestigieuse manifestation scientifique. Les tâches à réaliser pour marquer des points sont particulièrement ardues et très variées, constituant un Best Of des épreuves de toutes les autres éditions. Ceci nous a poussé à réfléchir à des solutions mécaniques originales et polyvalentes étant donné que les dimensions des robots de cette édition sont beaucoup plus petites que celles des années précédentes. Cette édition spéciale constitue un challenge intéressant pour nous et nous a amené à chercher de nouvelles solutions techniques afin d'optimiser l'exécution des tâches de base des robots et de leur laisser assez de réactivité pour marquer des points. Nous avons commencé à donner vie à ces différentes solutions mécaniques et électroniques. Nous comptons éprouver nos modèles avec davantage d'expériences et de tests et continuer à nous préparer pour la coupe du 08 Mai.

Notre groupe de projet transverse apprécie particulièrement de travailler sur cette coupe de robotique. La singularité de ce sujet nous a permis d'acquérir des connaissances dans des domaines techniques variés, de développer de nouvelles compétences et surtout de faire appel sans cesse à notre créativité et notre esprit scientifique pour répondre aux exigences de cette grande compétition. Nous avons également appris au fur et à mesure à nous organiser en tant que groupe soudé alors que nous n'avions pas les mêmes affinités et centres d'intérêt au départ. Notre équipe grandit en même temps que les robots et nous espérons arriver à la coupe avec des robots perfomants et une équipe compétitive.

Annexes

Annexe :

Repérer les couleurs à l'aide de la DFRobot TCS3200 et Arduino

Le **Capteur de couleur TCS3200 de DFRobot** est un détecteur complet de couleur, incluant une puce capteur RVB TCS3200 TAOS et 4 DEL blanches.

Comment ça marche :

Figure 11 : le capteur de couleur TCS3200 DFRobot présenté en Annexe

Comme nous pouvons le voir qu'il y a des rangées de photodiodes avec filtres de couleur sur le dessus. Il existe quatre types de filtres : un filtre clair, vert, rouge et bleu.

À l'exception de l'alimentation, TCS3200 a seulement 1 sortie et le reste des fils sont ses entrées. La sortie est un signal carré avec une fréquence proportionnelle à l'intensité de la lumière (rayonnement solaire) sur le filtre sélectionné. Il y a deux broches pour diviser la fréquence de la place des vagues S0 et S1, et deux épingles pour sélectionner la couleur du filtre S2 et S3. La puce fonctionne de 2, 7V à 5, 5V, donc cela va marcher très bien avec des microcontrôleurs 3, 3V ou 5V.

Câblage :

Lors de l'utilisation de la base de données TCS3200 avec Arduino : connecter la Vdd, V, et + 5v à ligne 5V de l'Arduino, Gnd se connecte à la masse de l'Arduino. Tous les autres fils sont présentés dans le programme, ils sont à connecter aux digital inputs de l'Arduino exemple pin 8 à 13.

Fonctionnement avec l'Arduino :

Le programme doit déterminer s'il un objet devant le capteur. S'il y a un objet présent, il doit deviner de quelle couleur il s'agit. Nous cherchons au front montant de la sortie de la TCS3200 pour le front descendant, donc le moins de temps s'est écoulé signifie qu'il n'y a plus de lumière. Il y aura quelques lectures redondantes, mais il s'agit d'assurer la détection correcte si quelque chose est en mouvement.

***Calibrage :** Prendre une mesure avec la LED éteinte, puis comparer que d'une mesure avec la LED (ligne 53), toutes les deux avec un filtre clair. S'il y a une différence significative, celasignifie qu'il existe un objet devant le capteur. S'il n'y a pas de différence significative alors juste retourner un zéro (ligne 61-64).

***Principe de détection de couleur :** Prendre une lecture du filtre clair et la comparer avec les lectures de chaque filtre de couleur, les deux avec la LED (ligne 67-72). Puis faire un peu d'un rapport inverse, depuis plus petits moyens plus légers et choisir la plus grande lecture (lignes 70 – 96).

On utilise la fonction detectColor (broche de TCS3200′s sortie signal carré) qui renvoie un 0 si rien n'est en face du capteur, 1 rouge, 2 bleu et 3 pour les verts. Après chaque lecture, il équipera la puce TCS3200 et LED.

***Exemple de programme en C inspiré:**

```
On connecte les pins du capteur (Taos pins) aux Digital pins i/o de l'Arduino pins 8-13.

int S0 = 8;//pinB
int S1 = 9;//pinA
int S2 = 12;//pinE
int S3 = 11;//pinF
int out = 10;//pinC
int LED = 13;//pinD

void setup() {
TCS3200setup();
Serial.begin(115200);
Serial.print("\n\n\nready\n\n\n");
delay(100);
}
void loop() {
Serial.print(detectColor(out));
Serial.print("\n\n\n");
delay(1000);
}
int detectColor(int taosOutPin){
//isPresentTolerance devra être quelque chose de petit, comme nous serons dans un
environnement très éclairé.
//La détection de couleur fonctionnera dans les deux cas, mais est le plus isPresentTolerance,
plus l'objet devra être en face du capteur
double isPresentTolerance = 5;
double isPresent = colorRead(taosOutPin,0,0)/colorRead(taosOutPin,0,1//le nombre augmente
quand quelque chose est en face du capteur.

Serial.print("estPresent:");
Serial.println(estPresent,2);
Serial.print("estPresentTolerance a actuellement la valeur:");
Serial.println(estPresentTolerance,2);
if(estPresent < estPresentTolerance){
Serial.println("rien n'est face au capteur");
return 0;
}
double red,blue,green;
double white = colorRead(taosOutPin,0,1);
red = white/colorRead(taosOutPin,1,1);
blue = white/colorRead(taosOutPin,2,1);
green = white/colorRead(taosOutPin,3,1);
Serial.print("red");
Serial.println(red);
Serial.print("blue");
Serial.println(blue);
Serial.print("green");
Serial.println(green);
if(red > blue && red > green){
Serial.println("couleur rouge en face");
return 1;
}
if(blue > green && blue > red){
Serial.println("couleur bleue en face");
return 2;
}
if(green > blue && green > red){
Serial.println("couleur verte en face");
return 3;
}
}
/*
```

```
/*
Cette méthode retourne la lecture pulseIn de la couleur sélectionnée.
Étant donné que la fréquence est proportionnelle à l'intensité lumineuse du filtre couleur
sélectionnée,
Plus pulseIn est petit, plus la lumière est de la couleur du filtre sélectionné.
On initialise le capteur avec taosMode(1) et on l'éteint à la fin taosMode(0)
couleur: 0 = blanc, 1 = rouge, 2 = bleu, 3 = vert
Si LEDstate est 0, la LED sera éteinte. A 1, la LED sera allumée.
taosOutPin est la sortie de la TCS3200.*/

double colorRead(int taosOutPin, int color, boolean LEDstate){ //s'assurer que notre pin est
en entrée (input)
pinMode(taosOutPin, INPUT);//allume le capteur avec rglage à haute fréquence
taosMode(1); //delay à laisser le capteur avant chaque lecture.
int sensorDelay = 1;//Set les broches pour sélectionner la couleur
 if(color == 0){//blanc
digitalWrite(S3, LOW); //S3
digitalWrite(S2, HIGH); //S2
                        // Serial.print("Blc");
}else if(color == 1){//rouge
digitalWrite(S3, LOW); //S3
digitalWrite(S2, LOW); //S2
                        // Serial.print(" r");
}else if(color == 2){//bleu
digitalWrite(S3, HIGH); //S3
digitalWrite(S2, LOW); //S2
                        // Serial.print(" b");
} else if(color == 3){ //vert
digitalWrite(S3, HIGH); //S3
digitalWrite(S2, HIGH); //S2
                         // Serial.print(" v");
}
double readPulse;
if(LEDstate == 0){
digitalWrite(LED, LOW);
}
if(LEDstate == 1){
digitalWrite(LED, HIGH);
}
delay(sensorDelay);
readPulse = pulseIn(taosOutPin, LOW, 80000);
                                          //if le pulseIn arrive à expiration, il renvoie la
valeur 0 et met en déroute les numéros.
if(readPulse < .1){
readPulse = 80000;
}
                  //éteint le capteur de couleur et la LED blanche pour
taosMode(0);
return readPulse;
}                 //A zéro, le taos est en mode faible consommation ;
void taosMode(int mode){
if(mode == 0){
                  //power OFF
digitalWrite(LED, LOW);
digitalWrite(S0, LOW); //S0
digitalWrite(S1, LOW); //S1
                    // Serial.println("mOFFm");
}else if(mode == 1){
                     //this will put in 1:1
digitalWrite(S0, HIGH); //S0
digitalWrite(S1, HIGH); //S1
                   // Serial.println("m1:1m");
}else if(mode == 2){
                       //this will put in 1:5
digitalWrite(S0, HIGH); //S0
digitalWrite(S1, LOW); //S1
                     //Serial.println("m1:5m");
}else if(mode == 3){
                       //this will put in 1:50
digitalWrite(S0, LOW); //S0
digitalWrite(S1, HIGH); //S1
//Serial.println("m1:50m");
}
return;
}
void TCS3200setup(){//initialisation des broches
pinMode(LED,OUTPUT); //LED pinD
                     //choix du mode de couleur
pinMode(S2,OUTPUT); //S2 pinE
pinMode(S3,OUTPUT); //S3 pinF
pinMode(S1,OUTPUT); //S1 pinA
return;}
```

```
// ----------------------------------------------------------------
// ANNEXE CHAPITRE 7 - ARDUINO ATMEGA2560 - CODE DE L'AUTOMATE
// Programme de l'automate de la carte Arduino ATmega2560 :
// Initialisation, Parcours, évitement d'obstacle, fin de partie.
// ----------------------------------------------------------------


char valeurConsole = 0; // valeur communiquée sur Serial via la console

int temps_de_la_partie = 10; // en secondes (retard avec le temps de calcul...)
int vitesse = 60;
int temps_de_recul = vitesse*16;
int temps_de_rotation = vitesse*12;
int temps_contourne = vitesse*16;

// ****************************************************************
// DEFINITION DES VARIABLES
// ****************************************************************

// Timer de la partie en ms
int timer = temps_de_la_partie*1000;
boolean trig_timer = false;

// Capteur d'obstacle
#define capteur A3
boolean obstacle;

// Communication BeagleBone
int led = 13; // port de la diode led
char valeurEntree; // definition variable pour recuperer RT1

// Définition des états du super automate
int etat;
#define INITIALISATION 0
#define PARCOURS 1
#define EVITEMENT 2

// Définition des états de l'automate d'évitement
int etat_evitement = 0;
#define INITIAL 0
#define AVANCE 1
#define RECUL 2
#define TOURNE_D 3
#define TOURNE_G 4

// Variables pour le schéma des états
long tempo; // temps d'attente, alternative au delay()
boolean trig_tempo; // true = enclenche le tempo
boolean trig_contourne; // true = enclenche le contournement

// Ports de commande moteurs
#define  DIRA  12
#define  DIRB  13
#define  PWMA  3
#define  PWMB  11
#define  BRAKEA  9
#define  BRAKEB  8

// Choix vitesse des moteurs
#define  MOTOR_LEFT_NORMAL_SPEED  vitesse  // OUTB
#define  MOTOR_RIGHT_NORMAL_SPEED  vitesse  // OUTA
```

```
// ****************************************************************
// SETUP
// ****************************************************************

void setup() {

  pinMode(capteur, INPUT); // port du capteur en mode "entrée"

  // Ports de communication
  pinMode(led, OUTPUT);
  Serial.begin(19200); // ouverture + débit du port série (usb) en bauds
  Serial1.begin(9600);  // ouverture + débit du port série 1

  pinMode(DIRA,OUTPUT);
  pinMode(DIRB,OUTPUT);
  digitalWrite(DIRA,HIGH);
  digitalWrite(DIRB,HIGH);

  pinMode(PWMA,OUTPUT);
  pinMode(PWMB,OUTPUT);
  analogWrite(PWMA, 0);
  analogWrite(PWMB, 0);

  pinMode(BRAKEA,OUTPUT);
  pinMode(BRAKEB,OUTPUT);
  digitalWrite(BRAKEA,LOW);
  digitalWrite(BRAKEB,LOW);

  // Message début de programme
  Serial.println("Console du programme du super automate :");
  Serial.println("----------------------------------------- ");
  Serial.println(" ");
}


// ****************************************************************
// FONCTIONS
// ****************************************************************

void avancer(){
  analogWrite(PWMA,MOTOR_RIGHT_NORMAL_SPEED);
  analogWrite(PWMB,MOTOR_LEFT_NORMAL_SPEED);
  digitalWrite(DIRA,HIGH);
  digitalWrite(DIRB,LOW);
}

void stopper(){
  analogWrite(PWMA,0);
  analogWrite(PWMB,0);
  delay(1);
  digitalWrite(BRAKEA,HIGH);
  digitalWrite(BRAKEB,HIGH);
  delay(1);
  digitalWrite(BRAKEA,LOW);
  digitalWrite(BRAKEB,LOW);
}

void reculer(){
  digitalWrite(DIRA,LOW);
  digitalWrite(DIRB,HIGH);
  analogWrite(PWMA,MOTOR_RIGHT_NORMAL_SPEED);
  analogWrite(PWMB,MOTOR_LEFT_NORMAL_SPEED);
}
```

```
void tourner_d(){
  digitalWrite(DIRA,LOW);
  digitalWrite(DIRB,LOW);
  analogWrite(PWMA,MOTOR_RIGHT_NORMAL_SPEED);
  analogWrite(PWMB,MOTOR_LEFT_NORMAL_SPEED);
}

void tourner_g(){
  digitalWrite(DIRA,HIGH);
  digitalWrite(DIRB,HIGH);
  analogWrite(PWMA,MOTOR_RIGHT_NORMAL_SPEED);
  analogWrite(PWMB,MOTOR_LEFT_NORMAL_SPEED);
}


// ****************************************************************
// LOOP
// ****************************************************************

void loop(){

  if (timer == 0){
    // FIN DE PARTIE
    if (trig_timer==0){
    stopper();
    Serial.println("****** FIN DE PARTIE ******");
    trig_timer = 1;
    }
    else{}
  }

  else
  {
    timer--;
    delay(1); // délai en ms
    // Détection d'obstacle
    if (analogRead(capteur) >= 400){
      obstacle = true;
    }
    else {
      obstacle = false;
    }

    if(Serial.available())
  {
    valeurConsole = Serial.read();
    Serial1.print(valeurConsole);

    Serial.print("Message envoye : ");
    Serial.write(valeurConsole);
    Serial.println("");
  }

    switch(etat) {
      case INITIALISATION:
      {
        // Faire l'initialisation
        etat_evitement = INITIAL;
        trig_tempo = false;
        tempo = 0;
        trig_contourne = false;
        obstacle = false;
        // Changement d'état
        etat = PARCOURS;
```

```
      break;
    }

    case PARCOURS:
    {
      // Mettre l'automate du parcours
if (Serial1.available()) {

  Serial.println("**************************");

  // lecture du port d'entrée RX1:
  valeurEntree = Serial1.read();

  // message de reception affiché à l'écran:
  Serial.print("message recu : ");
  Serial.write(valeurEntree);
  Serial.println(" ");

  // Faire avancer le moteur si la commande est "h"
  if (valeurEntree=='A'){
    avancer();
    Serial.println("j'avance");
    Serial1.write('a');
  }
  if (valeurEntree=='R'){
    reculer();
    Serial.println("je recule");
    Serial1.write('r');
  }
  if (valeurEntree=='D'){
    tourner_d();
    Serial.println("je tourne a droite");
    Serial1.write('d');
  }
  if (valeurEntree=='G'){
    tourner_g();
    Serial.println("je tourne a gauche");
    Serial1.write('g');
  }
  if (valeurEntree=='S'){
    stopper();
    Serial.println("je stoppe");
    Serial1.write('s');
  }
  // Vider le cache des ports séries pour le prochain message
  Serial1.flush();
  Serial.flush();
}

      // Conditions de changement d'état
      if (obstacle == 1){
        etat = EVITEMENT;
      }
      break;
    }

    case EVITEMENT:
    {
      // Automate pour l'évitement

      // Décompte du tempo
if(trig_tempo){
  tempo--;
}
```

```
// SCHEMA DES ETATS
switch (etat_evitement){

  case INITIAL:
  {
    stopper();
    // Conditions de changement d'état
    if (obstacle) {
      etat_evitement = RECUL;
    }
    else {
      etat_evitement = AVANCE;
    }
    break;
  }

  case AVANCE:
  {
    avancer();
    // Enclenchement du contournement
    if (trig_contourne == true){
      if (trig_tempo == false){
        tempo = temps_contourne;
        trig_tempo = true;
      }
      if (tempo == 0){
        trig_tempo = false;
        etat_evitement = TOURNE_G;
       trig_contourne = false;
      }
    }
    // Conditions de changement d'état
    if (obstacle){
      stopper();
      trig_contourne = false;
      trig_tempo = false;
      etat_evitement = RECUL;
    }
    break;
  }

  case RECUL:
  {
    // Enclenchement de l'état
    if(trig_tempo == false){
      reculer();
      //Initialisation du décompte
      tempo = temps_de_recul;
      trig_tempo = true;
    }
    else {}
    // Conditions de changement d'état
    if(tempo == 0){
      stopper();
      etat_evitement = TOURNE_D;
      trig_tempo = false;
    }
    break;
  }

  case TOURNE_D:
  {
    // Enclenchement de l'état
    if(trig_tempo == false){
```

```
        tourner_d();
        // Initialisation du décompte
        tempo = temps_de_rotation;
        trig_tempo = true;
      }
      else {}
      // Conditions de changement d'état
      if(tempo == 0){
        stopper();
        trig_tempo = false;
        etat_evitement = AVANCE;
        trig_contourne = true;
      }
      break;
    }

   case TOURNE_G:
   {
     // Enclenchement de l'état
     if(trig_tempo == false){
        tourner_g();
        // Initialisation du décompte
        tempo = temps_de_rotation;
        trig_tempo = true;
      }
      else {}
      // Conditions de changement d'état
      if(tempo == 0){
        stopper();
        etat_evitement = AVANCE;
        trig_tempo = false;
      }
      break;
    }
  }

        // Conditions de changement d'état
        if (obstacle == 0){
          etat = PARCOURS;
        }
        break;
      }
    }
  }
}
```

Annexe 1 du Chapitre 6

Méthode pratique de réglage d'un correcteur P, P.I ou P.I.D - Méthode de Ziegler Nichols

- **Principe**

Les correcteurs PI et P.I.D sont parmi les correcteurs analogiques les plus utilisés. Le problème principal réside dans la détermination des coefficients Kp, Ti, Td du correcteur. Plusieurs méthodes expérimentales ont été développées pour déterminer ces coefficients

La méthode développée par Ziegler et Nichols n'est utilisable que si le système étudié supporte les dépassements.

La méthode consiste à augmenter progressivement le gain d'un correcteur proportionnel pur jusqu'à la juste oscillation. On relève alors le gain limite (Klim) correspondant et la pulsation des oscillations ω_{osc}.

À partir de ces valeurs Ziegler et Nichols propose nt des valeurs permettant le réglage des correcteurs P, P.I et P.I.D.

Correcteur	P	P.I	P.I.D
Kp	$0.5 \cdot K_{\lim}$	$0.45 \cdot K_{\lim}$	$0.6 \cdot K_{\lim}$
Ti	∞	$0.83 \cdot T_{osc}$	$0.5 \cdot T_{osc}$
Td	0	0	$0.125 \cdot T_{osc}$

Figure 1 :_ Tableau des coefficients de Ziegler et Nichols

Annexe 2 du Chapitre 6 : Localisation du Robot (source RCVA)

Pour connaître la position du robot sur le terrain nous distinguons : la localisation échantillonnée sur le même rythme que l'asservissement (200Hz par exemple) de la localisation ponctuelle qui consiste à recaler le robot à divers moments stratégiques du match. Nous n'envisageons pas dans ce qui suit ce 2ème type de localisation par divers moyens comme par exemple l'utilisation des bordures, de balises fixes, de lignes tracées sur le terrain.
Pour la localisation échantillonnée c'est-à-dire le calcul à chaque période l'échantillonnage des coordonnées X,Y,Θ du robot.

Les coordonnées du robot sont prises au milieu de l'essieu des roues, l'orientation Θ représentant l'angle orienté formé par l'axe longitudinal du robot par rapport à l'axe des Y. (Θ négatif sur la figure). Comme pour le calcul de l'asservissement, on utilise l'information fournie par odométrie.

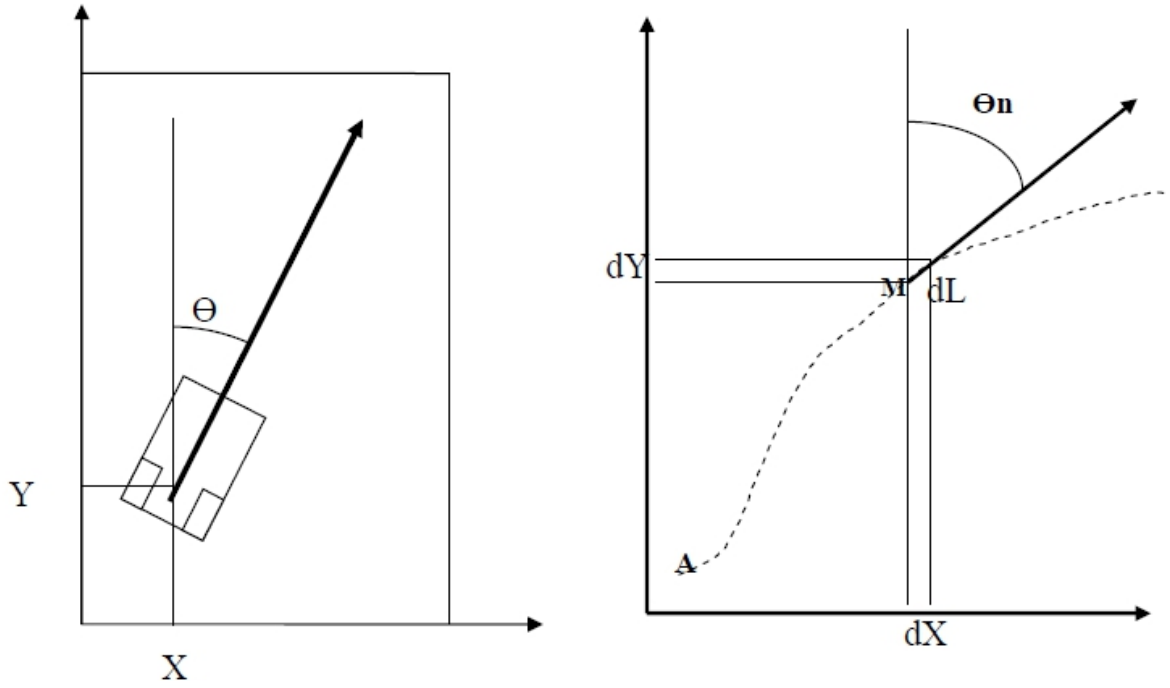


Figure 1 : Coordonnées du robot lors d'un déplacement

Le principe de la localisation:

L'odométrie permet de calculer la distance parcourue sur la trajectoire, donc la valeur de la vitesse dérivée de la distance. L'odométrie permettant par ailleurs de connaître l'orientation du robot Θ, c'est-à-dire l'orientation du vecteur vitesse, celui-ci est par conséquent connu (représenté par exemple par le vecteur en trait gras sur la figure droite). Il suffit alors d'intégrer le vecteur vitesse à chaque période d'échantillonnage pour connaître la position du robot. Pas de panique. Nous sommes en numérique. La dérivation s'obtient par une simple soustraction et l'intégration par une simple addition. Représentons en pointillé la trajectoire du robot sur le terrain (figure droite). Soit Te la période d'échantillonnage. Plaçons nous à l'instant présent n.Te (point M sur la figure). Entre l'instant présent n.Te et l'instant suivant (n+1).Te, le robot s'est déplacé de la distance dL sur sa trajectoire en suivant son orientation Θn. On rappelle (voir dossier asservissement) que :

Ln=1/2.(position_roue_D + position_roue_G)

Θn= Θo + (position_roue_D - position_roue_G)

Θo représentant l'orientation initiale du robot (au point A sur la figure).
Par définition la vitesse est égale à la distance parcourue pendant l'unité de temps égale à Te.
Soit : **Vn = dL = Ln+1 - Ln (vitesse à l'instant nTe)**

En projetant dL sur les 2 axes :

$$dX = -Vn \cdot \sin(\Theta n)$$
$$dY = Vn \cdot \cos(\Theta n)$$

En intégrant X et Y :

$$Xn+1 = Xn + dX$$
$$Yn+1 = Yn + dYr$$

L'algorithme d'actualisation de X et Y devient:

Fonction localisation échantillonnée

```
{
X=Xo ; // initialisation du robot
Y=Yo ;
Θ= Θo ;
Répéter à chaque Te
{saisie_codeurs_incrementaux() ;
L=1/2.(position_roue_D + position_roue_G) ;
Θ= Θo + (position_roue_D - position_roue_G) ;
vitesse=L – Lprecedent ;                    // dérivation
Lprecedent = L ;
deltaX= -vitesse*sin (Θ) ;
deltaY= vitesse*cos(Θ) ;
X=X + deltaX ;                              //intégration
Y=Y + deltaY ;}}
```

Annexe : Asservissement de la vitesse de l'ancien groupe

1- Théorie de la boucle de vitesse

Le but de la carte arduino est d'effectué le positionnement du robot. On note que suite à la
phase précédente, le robot est capable d'atteindre la vitesse souhaité. Mais, au delà de cela, on note
que l'asservissement est brouillon, il convint donc que nous allons calculer la fonction de transfert
de la boucle ouverte en vitesse pour permettre un calcul des constantes du correcteur.

Nous avons pris un schéma bloc typique et avons déterminé la fonction :

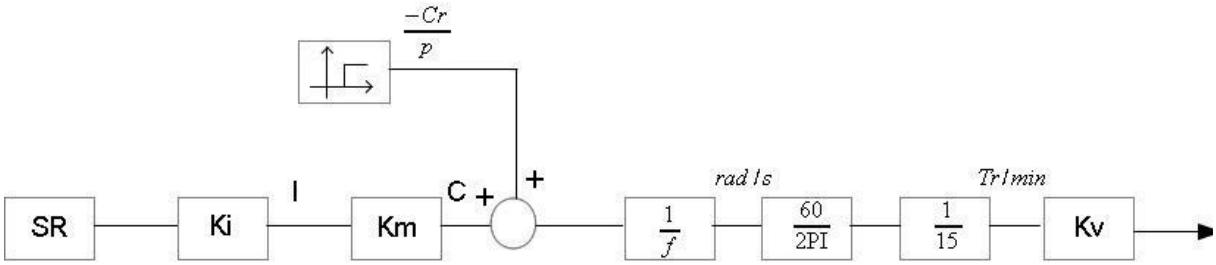


Figure 1 – Schéma bloc typique

KM est une donnée constructeur que nous avons trouvé dans la datasheet du moteur. Pour déterminer
le gain KI, nous avons tracé de manière expérimentale et en étude statique I en fonction

de Sr (vitesse demandée à l'entrée).

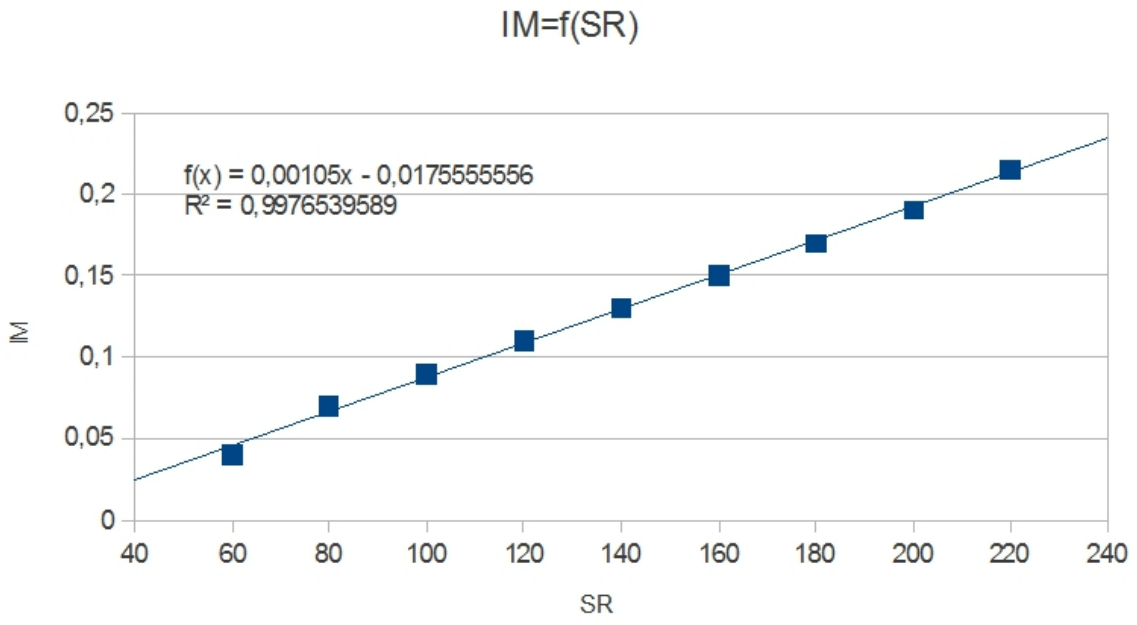


Figure 2 – Graphique IM=f(SR)

Kl est donc le coefficient directeur de la courbe ci dessus. Ensuite, nous avons tracé la courbe
V=f(Sr). Son coefficient directeur nous a permis de retrouver la valeur de f et sont ordonnée à

l'origine donne la valeur de Cr(modélisation des frottements statiques.

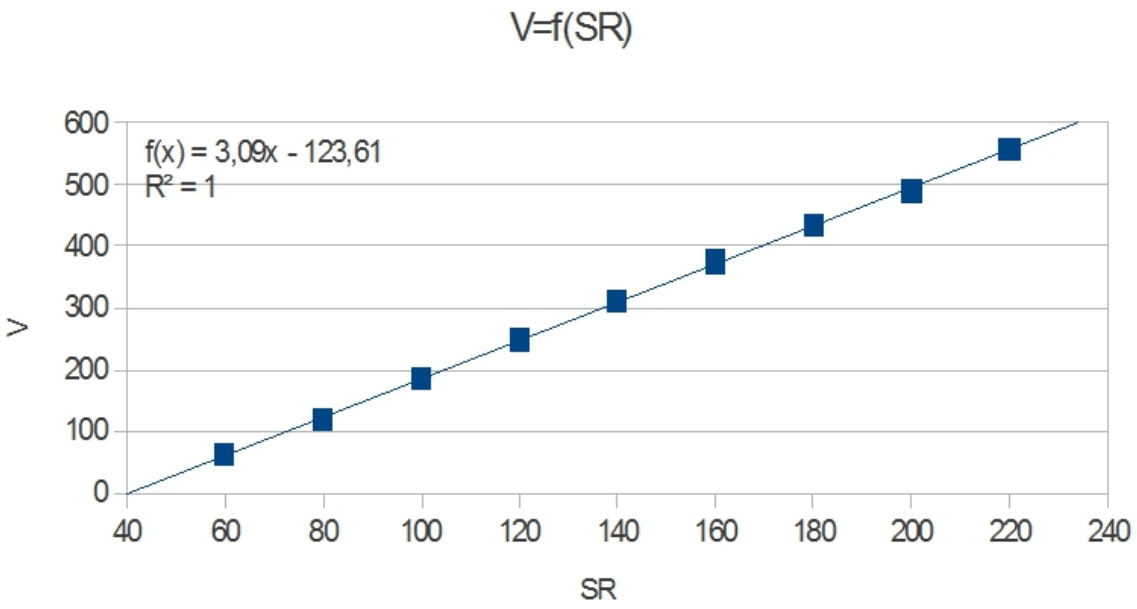


Figure 3 – Graphique V=f(SR)

Pour finir, Kv est un coefficient qui transforme la vitesse en tour par minute et est déterminé
en comptant le nombre de tour de roue en une minute à vitesse connu et constante.
Grâce à ces deux courbes, nous avons l'expression de la fonction de transfert moteur + pont en h
en étude statique.
Maintenant, il faut trouver la constante de temps du système (celui-ci étant d'ordre 1). Pour
cela, on lui donne une vitesse en dehors du domaine de saturation du système et on trouve la constante de temps en traçant la tangente à l'origine (elle coupe la valeur final en T0).

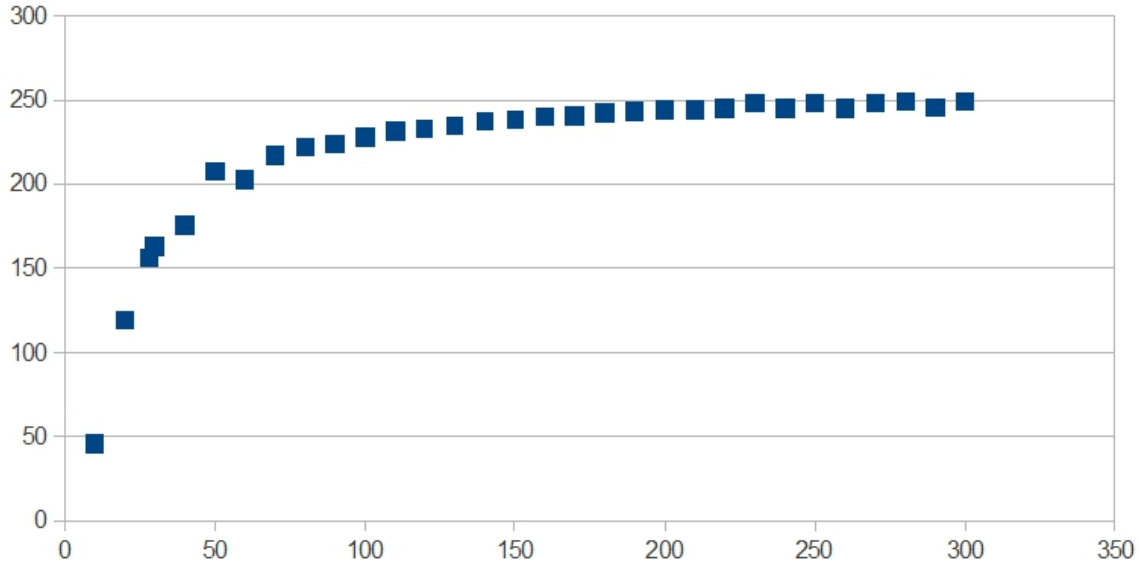


Figure 4 – Réponse du système
On en déduit donc la valeur 28e-5s pour T0 et on l'introduit dans le schéma bloc sous forme

d'un premier ordre.

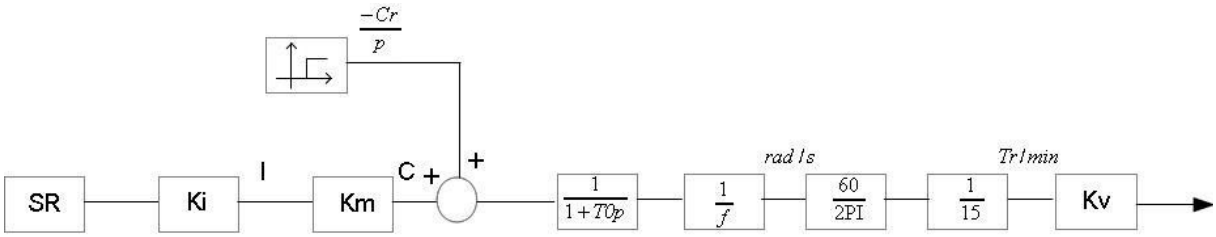


Figure 5 – Schéma bloc du moteur réel
Nous avons ensuite bouclé le système et déterminé le gain critique grâce à un diagramme de
black du système grâce à la modélisation de la boucle ouverte avec Acsyde. Nous avons déterminé
les coefficients du correcteur (PI au début) grâce à la méthode de Ziegler-Nichols. Pour cela, nous
avons dut introduire, et dans les frottements statiques, et dans la boucle directe un retard pur de

valeur la demi période d'échantillonnage. Ci-après, le schéma bloc de la boucle fermé :

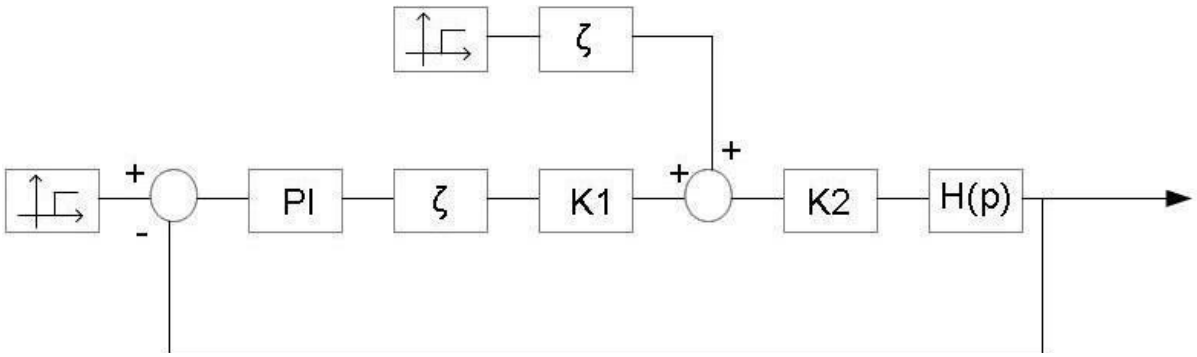


Figure 6 – Schéma bloc de la boucle fermé

2- Test de la boucle de vitesse

Avant cette étape, nous avons reçu un premier robot, les tests ont donc eu lieu sur celui-ci.

Pour les premiers tests, nous avons utilisé les coefficients donnés par la méthode de Ziegler-
Nichols, nous avons obtenu la réponse qui suit avec une condition en vitesse de 300 :

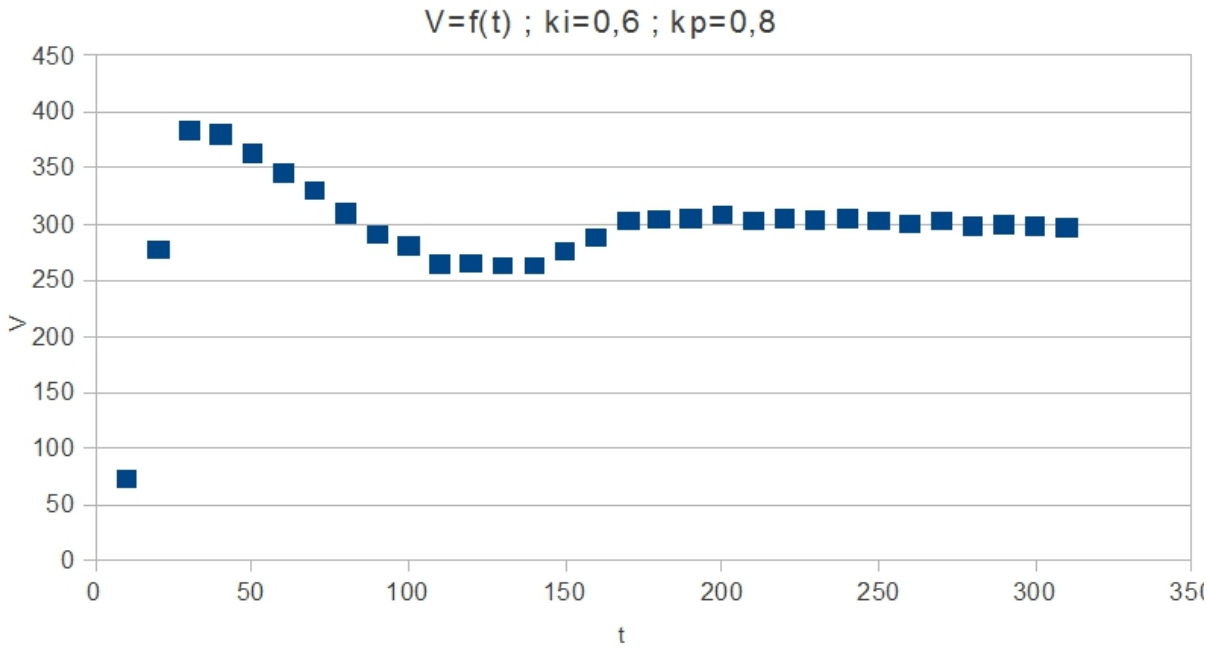


Figure 7 – Réponse V=f(t) pour une condition de vitesse de 300 avec ki=0.6 et kp=0.8

Cette courbe montre que l'asservissement est correct. En effet, le système tend relativement
vite vers la commande. Cependant, il y a un trop dépassement et le système oscille avant d'atteindre
la valeur souhaité. Il convient donc de rectifier légèrement les coefficients du PI pour obtenir
quelque chose de plus optimal que ceux calculer par Ziegler-Nichols. Après seulement 4 essaies,
nous obtenons ce résultat.

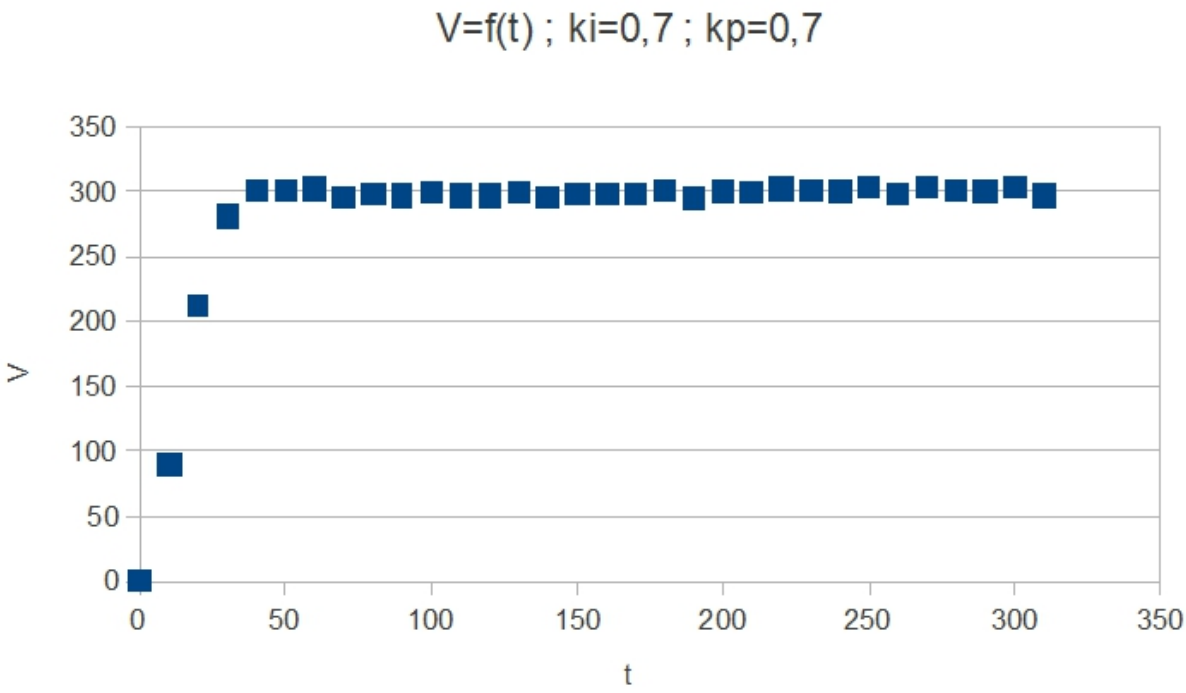


Figure 8 – Réponse V=f(t) pour une condition de vitesse de 300 avec ki=0.7 et kp=0.7

Sur cette courbe, le temps de réponse à 5% est de l'ordre de 30ms et il n'y a aucun dépassement.

C'est tellement bien que nous n'allons même pas essayer un correcteur PID car il serait difficile de

faire mieux.

Annexe 3 du Chapitre 6 : Mesure du déplacement angulaire de l'axe d'un moteur avec une carte Arduino – Par M. Laurent Dethoor

Cette article traite de la méthode à suivre pour mesurer un déplacement angulaire (axe d'un moteur ou d'un potentiomètre) avec une carte Arduino.

Principe de mesure du déplacement angulaire de l'axe d'un moteur.

Pour pouvoir mesurer le déplacement angulaire de l'axe d'un moteur, le moteur doit être équipé d'un « codeur rotatif » fixé sur son arbre.

Codeur rotatif seul

Codeur rotatif monté sur un moteur à courant continu

Nous traiterons dans cet article des **codeurs rotatifs incrémentaux**.

Le codeur rotatif incrémental de type optique est constitué d'un disque avec des fines fentes sur la périphérie, fixé sur l'axe en rotation.

Vue d'un codeur rotatif optique

Signaux de sorties d'un codeur rotatif incrémental

Une lumière émise par une Led traverse les fentes et éclaire des capteurs optiques (photo-transistors) . Deux capteurs optiques sont disposés de tels manières qu'ils produisent 2 signaux électriques **A et B impulsionnels en quadrature** l'un de l'autre (déphasage de 90°). Une impulsion représente le **pas angulaire**. Si le codeur rotatif génère 120 impulsions par tours, le pas angulaire sera de 360°/120 = 3°.

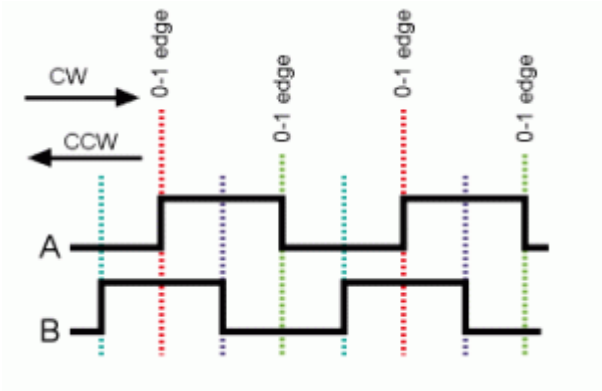
Les codeurs rotatifs incrémentaux permettent de mesurer le déplacement angulaire mais aussi le sens de rotation.

Pour mesurer le déplacement angulaire, on incrémente un compteur (+1) à chaque impulsion si la rotation est dans un sens, et on décrémente ce même compteur (-1) à chaque impulsion si la rotation est dans le sens opposé.

Mise en œuvre de la mesure du déplacement angulaire à l'aide d'une carte Arduino.

Cette partie de l'article a été écrite à partir des informations présentes sur le site Arduino.

La méthode utilisée ici est la suivante :



Chronogramme des signaux de sortie du codeur rotatif

- Un des deux signaux du codeur rotatif (par exemple A) est connecté sur une broche de la carte Arduino sensible au changement d'état ; à chaque changement d'état, le micro-contrôleur de l'Arduino exécutera une routine d'interruption.
- Cette routine d'interruption lit l'état sur la broche reliée à l'autre signal du codeur rotatif (par exemple B) et incrémente une variable de comptage si le signal A et le signal B sont au même état, ou décrémente celui-ci si ils sont à des états différents.
- La variable de comptage contiendra donc le nombre de pas angulaire relatif effectué par le codeur rotatif depuis le début de l'exécution du programme Arduino.

Exemple de programme sur Arduino :

```
/* Lecture des signaux d'un codeurs rotatifs et utilisation d'une interruption externe. Le signal encoder0PinA est connecté à la broche 2 (entrée d'interruption externe).Le signal encoder0PinB est connecté à la broche 4. Il n'y a aucune importance si les deux signaux A et B du codeur rotatif sont échangés.Dans cet exemple, le nombre maximal de pas angulaire est fixé à 65535. Pour augmenter la capacité de comptage, vous pouvez déclarer la variable encoder0Pos en "unsigned long"*/

#define encoder0PinA  2

#define encoder0PinB  4

volatile unsigned int encoder0Pos = 0;

void setup() {

        pinMode(encoder0PinA, INPUT);

        pinMode(encoder0PinB, INPUT);

        attachInterrupt(0, doEncoder, CHANGE);  // L'interruption externe 0 est affectée à la broche 2 de l'Arduino

  Serial.begin (9600);

 Serial.println("ça commence");}
```

```
void loop(){

if (encoder0Pos!=encoder0Pos_prev){

Serial.println(encoder0pos,DEC);

encoder0Pos_prev = encoder0Pos;}}}

void doEncoder() {

   /* Si le signal A et le signal B sont tous les deux à 1 ou à 0, * la rotation a lieu dans un sens (sens horaire par exemple).* Si ils sont à des états différents, la rotation est inversée (sens anti-horaire).*/

  if (digitalRead(encoder0PinA) == digitalRead(encoder0PinB)) {

    encoder0Pos++;  }

else {  encoder0Pos--;  }

}
```

```
// initialisation des variables


int etat_gauche_droite;
boolean obstacle_avant;
#define GAUCHE 0
#define DROITE 1
long tempo_gauche;
long tempo_droite;
boolean trig_tempo_gauche;
boolean trig_tempo_droite;

const int numOfReadings = 10;
int readings1[numOfReadings];
int readings2[numOfReadings];
int arrayIndex1 = 0;
int arrayIndex2 = 0;
int total1 = 0;
int total2 = 0;
int averageDistance1 = 0;
int averageDistance2 = 0;


int echoPin1 = 2;
int initPin1 = 3;
int echoPin2 = 4;
int initPin2 = 5;
unsigned long pulseTime1 = 0;
unsigned long distance1 = 0;
unsigned long pulseTime2 = 0;
unsigned long distance2 = 0;


// setup

void setup() {

  pinMode(initPin1, OUTPUT);                 // définit init pin 3 comme une sortie
  pinMode(echoPin1, INPUT);                 // définit echo pin 2 comme une entrée
  pinMode(initPin2, OUTPUT);                 // définit init pin 5 comme une sortie
  pinMode(echoPin2, INPUT);                 // définit echo pin 4 comme une entrée

  // remise à zéro des listes

  for (int thisReading = 0; thisReading < numOfReadings; thisReading++) {
readings1[thisReading] = 0;
readings2[thisReading] = 0;
 }

// initialise le port série, pour permettre l'affichage

    Serial.begin(9600);
```

```
// initialisation des variables

    etat_gauche_droite = GAUCHE;
    trig_tempo_gauche = false;
    tempo_gauche = 0;
    trig_tempo_droite = false;
    tempo_droite = 0;
 }

// execute
void loop() {

delay(1);
if(trig_tempo_gauche){                  // décrémente tempo_gauche
   tempo_gauche--;
}
if(trig_tempo_droite){                  // décrément tempo_droit
   tempo_droite--;
}
switch (etat_gauche_droite){
   case GAUCHE:
   {
    if(trig_tempo_gauche == false){
      digitalWrite(initPin1, HIGH);                 // envoie une impulsion de 10 microsecondes
      delayMicroseconds(10);                // attend 10 microsecondes avant de s'arrêter
      digitalWrite(initPin1, LOW);                  // arrête d'envoyer l'impulsion
      pulseTime1 = pulseIn(echoPin1, HIGH);           // mesure le temps de l'impulsion retour
      distance1 = pulseTime1/58;                    // Distance = pulse time / 58 (pour avoir des cm)
       total1= total1 - readings1[arrayIndex1];         // soustrait la dernière distance
      readings1[arrayIndex1] = distance1;             // ajoute la distance lue à la liste
      total1= total1 + readings1[arrayIndex1];          // ajoute la valeur au total
      arrayIndex1 = arrayIndex1 + 1;                  // va au prochain élément de la liste

      // Remise à zéro de l'index

      if (arrayIndex1 >= numOfReadings)  {
          arrayIndex1 = 0;
        }

        averageDistance1 = total1 / numOfReadings;      // calcule la distance moyenne
      tempo_gauche = 10;
      trig_tempo_gauche = true;
    }
    if(tempo_gauche ==0){
      etat_gauche_droite = DROITE;          // passe au capteur suivant
      trig_tempo_gauche = false;
    }
    break;
   }

   case DROITE:
```

```
    {
     if(trig_tempo_droite == false){
       digitalWrite(initPin2, HIGH);
       delayMicroseconds(10);
       digitalWrite(initPin2, LOW);
       pulseTime2 = pulseIn(echoPin2, HIGH);
       distance2 = pulseTime2/58;
        total2= total2 - readings2[arrayIndex2];
       readings2[arrayIndex2] = distance2;
       total2= total2 + readings2[arrayIndex2];
       arrayIndex2 = arrayIndex2 + 1;

       if (arrayIndex2 >= numOfReadings)  {
           arrayIndex2 = 0;
         }

         averageDistance2 = total2 / numOfReadings;
       tempo_droite = 10;
       trig_tempo_droite = true;
     }
     if(tempo_droite ==0){
       etat_gauche_droite = GAUCHE;
       trig_tempo_droite = false;
       Serial.print("(");
       Serial.print(averageDistance1, DEC);         // affiche la distance moyenne gauche
       Serial.print(",");
       Serial.print(averageDistance2, DEC);         // affiche la distance moyenne droite
       Serial.println(")");
       if((averageDistance1<10)||(averageDistance2<10)){   // décide de la présence d'obstacle
         obstacle_avant = true;
       }
       else{
         obstacle_avant = false;
       }
       Serial.println(obstacle_avant);              // affiche s'il y a un obstacle
     }
     break;
    }
}

}
```

```
int sensorpin = 0;                // pin analogique utilisé pour connecter le capteur
float val = 0;                 // variable pour garder les valeurs du capteur
float voltage = 0;
float seuil = 2.5;
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  val = analogRead(sensorpin);  // lit la valeur du capteur
  voltage = val*5/1024;

  if (voltage>seuil){
    Serial.println("Obstacle");
  }
  else {
    Serial.println("");
  }
  delay(100);
}
```

Fichier LancerConversion.cpp

```
InverserListe();
   char* liste_variable;
   int TAILLE_FERMEE = Get_tailleFermee()*sizeof(char);
   liste_variable = malloc(TAILLE_FERMEE);

   int absp = Getx_listeInverse(0);                    // Coordonnées de l'abscisse précédente
   int ordp = Gety_listeInverse(0);                    // Coordonnées de l'ordonnée précédente

   char orientation;                                   // Permet de connaitre l'orientation du robot
                                                          (Nord, Est, Sud, Ouest)
   int i = 1;
   int condition = 1;                                  // Condition d'arrêt (1=marche / 0=arrêt)
   int aAjouter;


   void ajouterDansListeVariable()
   {
      if (aAjouter == 1){liste_variable[i]='t';}
      if (aAjouter == 2){liste_variable[i]='d';}
      if (aAjouter == 3){liste_variable[i]='a';}
      if (aAjouter == 4){liste_variable[i]='g';}
   }




   void lancerConversion()
   {
      while (condition == 1)
      {
         int abs = Getx_listeInverse(i);                  // abscisse suivante
         printf("abs = %d ",abs);
         int ord = Gety_listeInverse(i);                  // ordonnée suivante
         printf("ord = %d ",ord);
         printf(" -- absp = %d",absp);
         printf(" -- ordp = %d",ordp);
         if (i==1) orientation = 'N';                     // initialisation à N si premier déplacement
```

```
if(orientation == 'N')                          // Orientation NORD
{
   if((absp == abs) && (ordp > ord))                  // le robot va vers le sud
   {
      //demi-tour + avance
      printf(" Dans N ");
      aAjouter = 1;
      //liste_variable[i] = 't';
      ajouterDansListeVariable();
      orientation='S';
      printf("puis dans S ");
   }
   if((absp == abs) && (ordp < ord))                  // le robot va vers le nord
   {
      //tout droit
      printf(" Dans N ");
      aAjouter = 3;
      //liste_variable[i] = 'a';
      ajouterDansListeVariable();
      printf("          ");
   }
   if((absp > abs) && (ordp == ord))                  // le robot va vers l'ouest
   {
      //tourne gauche + avance
      printf(" Dans N ");
      aAjouter = 4;
      //liste_variable[i] = 'g';
      ajouterDansListeVariable();
      orientation = 'O';
      printf("puis dans O ");
   }
   if((absp < abs) && (ordp == ord))                  // le robot va vers l'est
   {
      //tourne droite + avance
      printf(" Dans N ");
      aAjouter = 2;
      //liste_variable[i] = 'd';
      ajouterDansListeVariable();
      orientation = 'E';
      printf("puis dans S ");
   }
}
```

```
else if(orientation == 'E')                    // Orientation EST
{
   if((absp == abs) && (ordp > ord))                    // le robot va vers le sud
   {
      //tourne droite + avance
      printf(" Dans E ");
      aAjouter = 2;
      //liste_variable[i] = 'd';
      ajouterDansListeVariable();
      orientation = 'S';
      printf("puis dans S ");
   }
   if((absp == abs) && (ordp < ord))                    // le robot va vers le nord
   {

      //tourne gauche + avance
      printf(" Dans E ");
      aAjouter = 4;
      //liste_variable[i] = 'g';
      ajouterDansListeVariable();
      orientation = 'N';
      printf("puis dans N ");
   }
   if((absp > abs) && (ordp == ord))                    // le robot va vers l'ouest
   {
      //demi-tour + avance
      printf(" Dans E ");
      aAjouter = 1;
      //liste_variable[i] = 't';
      ajouterDansListeVariable();
      orientation = 'O';
      printf("puis dans O ");
   }
   if((absp < abs) && (ordp == ord))                    // le robot va vers l'est
   {
      //tout droit
      printf(" Dans E ");
      aAjouter = 3;
      //liste_variable[i] = 'a';
      ajouterDansListeVariable();
      printf("          ");
   }
}
```

```
else if(orientation == 'S')                    // Orientation SUD
{
   if((absp == abs) && (ordp > ord))                    // le robot va vers le sud
   {
      //tout droit
      printf(" Dans S ");
      aAjouter = 3;
      //liste_variable[i] = 'a';
      ajouterDansListeVariable();
      printf("          ");
   }
   if((absp == abs) && (ordp < ord))                    // le robot va vers le nord
   {
      //demi-tour + avance
      printf(" Dans S ");
      aAjouter = 1;
      //liste_variable[i] = 't';
      ajouterDansListeVariable();
      orientation = 'N';
      printf("puis dans N ");
   }
   if((absp > abs) && (ordp == ord))                    // le robot va vers l'ouest
   {
      //tourne droite + avance
      printf(" Dans S ");
      aAjouter = 2;
      //liste_variable[i] = 'd';
      ajouterDansListeVariable();
      orientation = 'O';
      printf("puis dans O ");
   }
   if((absp < abs) && (ordp == ord))                    // le robot va vers l'est
   {
      //tourne gauche + avance
      printf(" Dans S ");
      aAjouter = 4;
      //liste_variable[i] = 'g';
      ajouterDansListeVariable();
      orientation = 'E';
      printf("puis dans E ");
   }
}
```

```
else if(orientation == 'O')                    // Orientation OUEST
{
   if((absp == abs) && (ordp > ord))                    // le robot va vers le sud
   {
      //tourne gauche + avance
      printf(" Dans O ");
      aAjouter = 4;
      //liste_variable[i] = 'g';
      ajouterDansListeVariable();
      orientation = 'S';
      printf("puis dans S ");
   }
   if((absp == abs) && (ordp < ord))                    // le robot va vers le nord
   {
      //tourne droite + avance
      printf(" Dans O ");
      aAjouter = 2;
      //liste_variable[i] = 'd';
      ajouterDansListeVariable();
      orientation = 'N';
      printf("puis dans N ");
   }
   if((absp > abs) && (ordp == ord))                    // le robot va vers l'ouest
   {
      //tout droit
      printf(" Dans O ");
      aAjouter = 3;
      //liste_variable[i] = 'a';
      ajouterDansListeVariable();
      printf("          ");
   }
   if((absp < abs) && (ordp == ord))                    // le robot va vers l'est
   {
      //demi-tour + avance
      printf(" Dans O ");
      aAjouter = 1;
      //liste_variable[i] = 't';
      ajouterDansListeVariable();
      orientation = 'E';
      printf("puis dans E ");
   }
}
```

```
        printf(" --> %c <--\n",liste_variable[i]);

        absp = abs;
        ordp = ord;
        i++;
        if (i==Get_tailleFermee()) {condition = 0; printf("Le robot est arrive a destination : [%d;
%d]",abs,ord);}

      }
   }
```

Fichier Point.h

```
#include <iostream>

class Point{

   public:
   Point(int abscisse=0, int ordonnee=0);                    //Constructeur du Point
   friend std::ostream& operator<<(std::ostream& sortie,Point P);     // Permet l'affichage
                                                                        d'informations sur le Point
   int getabs();                                             // Permet de récupérer l'abscisse du Point
   int getord();                                             // Permet de récupérer l'ordonnée du Point
   void setabs(double ab);                                   // Permet de modifier l'abscisse du Point
   void setord(double ordo);                                 // Permet de modifier l'ordonnée du Point
   double distance( Point p);                                // Renvoie la distance jusqu'au Point p
   double distance();                                        // Renvoie la distance par rapport à l'origine
   Point plus(Point p);                                      // Permet l'addition de Point
   Point moins(Point p);                                     // Permet la soustraction de Point
   double getdirect(Point p);                                // Permet de connaitre l'orientation pour
                                                             aller vers le Point p
   double getangle();                                        // Renvoie l'angle du Point
   double produitscalaire(Point p);                          // Renvoie le produit scalaire avec le Point p


   private:
   int abs;                  // Représente l'abscisse du Point
   int  ord;                 // Représente l'ordonnée du Point

};
```

Fichier Cadeau.h

```
#include "Point.h"
#include "calculChemin.h"

class Cadeau
{
  public:
    Cadeau(Point c, std::string co);          //Constructeur de cadeau
    Point getcentre();                        // Retourne le centre du Cadeau (Point de référence)
    std::string getcouleur();                 // Retourne la couleur du Cadeau
    bool estouvert();                         // Retourne vrai si le cadeau est ouvert, faux sinon
    void ouvre();                             // Passe le booleen ouvert de faux à vrai
    friend std::ostream& operator<<( std::ostream &sortie, Cadeau c);   //Permet l'affichage des
                                                                          informations sur le cadeau
    Point getdest();                          // Retourne le point où doit se placer le robot pour ouvrir le
                                              cadeau

  private:
    Point centre;                             // Point de référence du cadeau
    bool ouvert;                              // Dit si le cadeau est ouvert ou non
    std::string couleur;                      // Couleur du cadeau (rouge ou bleu)
};
```

Fichier Bougie.h

```
#include "Point.h"

class Bougie {
  public:
  Bougie(Point p, std::string c ,int e=0);            // Constructeur de Bougie
  Point getcentre();                                  // Retourne le centre de la Bougie
  int getetage();                                     // Retourne l'étage de la bougie
  std::string getcouleur();                           // Retourne la couleur de la bougie
  bool estallume();                                   // Retourne vrai si la bougie est allumée, faux sinon
  void eteint();                                      // Passe le booleen allumee de vrai à faux
  friend std::ostream& operator<<( std::ostream &sortie, Bougie b);      // Permet l'affichage des
                                                                         informations de la bougie

  private:
    Point centre;                                     // Point de référence au centre de la bougie
    int etage;                                        // Etage symbolise le niveau de la bougie sur le gateau
    bool allumee;                                     // Permet de savoir si la bougie est allumée ou non
    std::string couleur;                              // Couleur de la bougie (bleu, rouge ou blanc)

};
```

Fichier Verre.h

```
#include "Point.h"
#include "Zone.h"
#include "calculChemin.h"

class Verre
{
   public:
   Verre(Point p);                          //Constructeur de Verre
   bool estdansrobot();                     // Retourne vrai si le verre est dans un robot, faux sinon
   int getetage();                          // Retourne l'étage du verre
   Point getcentre();                       // Retourne le centre du verre
   void entredansrobot();                   // Passe le booleen dansrobot de faux à vrai
   void changeetage(int i);                 // Change l'étage par la valeur i donné en paramètre
   friend std::ostream& operator<<( std::ostream &sortie, Verre v);   //Permet l'affichage des
                                                                       informations sur le verre
   void setcentre(Point p);                 // Permet de modifier le centre du verre
   void sortrobot();                        // Passe le booleen dansrobot de vrai à faux
   bool estdanszone(Zone Z);                // Permet de savoir si le verre est dans la Zone Z ou non
   Point getdest();                         // Retourne le point où se rendre pour ramasser le verre

   private:
   Point centre;                  // Point de référence centre du verre
   int etage;                     // Etage du verre s'il est dans une pyramide
   bool dansrobot;                // Permet de savoir si le verre est dans un robot ou non

};
```

Fichier Cerise.h

```
#include "Point.h"

class Cerise
{
   public:
   Cerise(Point c, std::string coul);        // Constructeur de Verre
   bool estdansrobot();                      // Permet de savoir si la cerise est dans un robot ou non
   std::string getcouleur();                 // Retourne la couleur de la cerise
   Point getcentre();                        // Retourne le centre de la cerise
   void setcentre(Point p);                  // Modifie le centre de la cerise
   void entredansrobot();                    // Modifie le booleen dansrobot de faux à vrai
   bool estdanspanier();                     // Permet de savoir si la cerise est dans un panier ou non
   void entredanspanier();                   // Modifie le booleen danspanier de faux à vrai
   void sortrobot();                         // Modifie le booleen dansrobot de vrai à faux
   friend std::ostream& operator<<( std::ostream &sortie, Cerise c);   //Permet l'affichage
                                                                        d'informations sur la cerise

   private:
   Point centre;                  // Point de référence centre de la cerise
   std::string couleur;           // Couleur de la cerise
   bool dansrobot;                // Permet de savoir si la cerise est dans un robot ou non
   bool danspanier;               // Permet de savoir si la cerise est dans un panier ou non
};
```

Fichier Robot.h

```
#include "Point.h"

class Robot
{
   public:
   Robot(Point ci, double orientation,std::string c);        //Constructeur du robot
   void deplacement(Point p);                    // Permet de se déplacer d'un cran vers le Point p (si bonne orientation au préalable)
   void rotation(double anglearret);             // Permet de tourner d'un cran vers l'anglearret
   void arretrotation();                         // Modifie le booleen tourne de vrai à faux
   double getangle();                            // Retourne l'angle pour connaitre l'orientation du robot
   Point getpoint();                             // Retourne le centre du robot
   std::string getcouleur();                     // Retourne la couleur du robot
   friend std::ostream& operator<<( std::ostream &sortie, Robot r);      // Permet l'affichage d'informations sur le robot
   bool estgonfle();                             // Permet de savoir si le ballon du robot est gonflé
   void gonfle();                                // Modifie le booleen ballongonfle de faux à vrai
   void va(Point p);                             //Permet le déplacement complet du robot vers le Point p (rotation inclus)

   protected:
   Point centre;                        // Point de référence du robot(centre de gravité)
   double angle;                        // Angle permettant de connaitre l'orientation du robot
   bool tourne;                         // Est vrai si le robot est entrain de tourner
   bool avance;                         // Est vrai si le robot est entrain d'avancer
   double vitesse;                      // Permet de simuler la vitesse du robot
   double vitesserotation;              // Permet de simuler la vitesse de rotation du robot
   std::string couleur;                 // Permet de connaitre la couleur du robot
   bool ballongonfle;                   // Est vrai si le ballon est gonflé

};
```

Fichier GrosRobot.h

```
#include "Robot.h"
#include <vector>
#include "Verre.h"
#include "Cadeau.h"


class GrosRobot :public Robot
{
   public:
   GrosRobot(Point p,double orientation,std::string coul);        // Constructeur du Gros Robot
   void ramasseverre(Verre* v);                                   // Fonction permettant le ramassage
                                                                  d'un verre (ajout au tableauverre)
   void deplacement(Point p);                                     // Permet le deplacement vers un
                                                                  Point P d'un cran
   void poseverre();                                               // Permet de déposer les verres la où
                                                                  est le robot
   void deballe(Cadeau* C);                                       // Permet de déballer un cadeau
   void va(Point p);                                              // Permet le deplacement complet
                                                                  (rotation plus avancement) en ligne
                                                                  droite


   private:
   std::vector<Verre*> tableauverre;                  // Représente les verres contenus dans le robot

};
```

Fichier PetitRobot.h

```
#include "Robot.h"
#include "Bougie.h"
#include "Panier.h"
#include "Assiette.h"


class PetitRobot: public Robot{
  public:
  PetitRobot(Point p,double orientation,std::string coul);          // Constructeur du Petit Robot
  void deplacement(Point p);                                        // Permet le deplacement vers un
                                                                    Point P d'un cran
  void ramassecerise(Assiette* A);                                   // Permet le ramassage des cerises
                                                                    d'une Assiette (ajout dans
                                                                    tableaucerise)
  void ramassecerise(Cerise *C);                                    // Permet le ramassage d'une cerise
                                                                    (ajout dans tableaucerise)
  void jettecerise(Panier *P);                                       // Permet la sortie de cerise du
                                                                    robot
  void va(Point p);                                                  // Permet le déplacement complet
                                                                    du robot
  void eteintbougie(Bougie *B);                                      // Permet d'éteindre une bougie

  private:
  std::vector<Cerise*> tableaucerise;                 // Représente les cerises contenues dans le robot

};
```

Fichier Panier.h

```
#include <vector>
#include "Cerise.h"

class Panier
{
   public:
   Panier(Point p, std::string c);          //Constructeur du Panier
   void ajoutecerise(Cerise* c);            // Permet la reception d'une cerise dans le panier (ajout dans
                                            tableaucerise)
   std::string getcouleur();                // Renvoie la couleur du Panier
   Point getcentre();                       // Renvoie le centre du Panier
   std::vector<Cerise*> gettab();           // Renvoie le tableau de cerises présentes dans le panier

   private:
   Point centre;                                // Représente le centre du Panier
   std::string couleur;                         // Permet de connaitre la couleur du panier
   std::vector<Cerise*> tableaucerise;          // Représente les cerises présentes dans le panier
};
```

Fichier Zone.h

```
#include "Point.h"

class Zone
{
   public:
   Zone(Point hg,Point hd,Point bg,Point bd,std::string c);        //Constructeur de la Zone
   Point getbg();                                                  // Renvoie le point bas gauche de la
                                                                   Zone
   Point gethd();                                                  // Renvoie le point haut droit de la
                                                                   Zone
   std::string getcouleur();                                       // Renvoie la couleur de la Zone

   private:
   Point hautgauche;                          // Représente le point haut gauche
   Point hautdroit;                           // Représente le point haut droit
   Point basgauche;                           // Représente le point bas gauche
   Point basdroit;                            // Représente le point bas droit
   std::string couleur;                       // Représente la couleur de la Zone

};
```

Fichier Assiette.h

```
#include "Cerise.h"
#include <vector>

class Assiette
{
 public:
 Assiette(Point p);                                  //Constructeur de l'Assiette
 Point getcentre();                                  // Renvoie le centre de l'Assiette
 void ajoutecerise(Cerise* c);                       // Ajoute la Cerise dans l'Assiette
 void enlevecerise();                                // Enlève la Cerise de l'Assiette
 std::vector<Cerise*> gettableau();                  // Renvoie les Cerises dans l'Assiette
 friend std::ostream& operator<<( std::ostream &sortie, Assiette a);     //Permet l'affichage
                                                                         d'informations sur l'Assiette

 private:
  Point centre;                                      // Représente le centre de l'Assiette
  std::vector<Cerise*> tableaucerise;                // Représente les Cerises présentes dans l'Assiette

};
```

Fichier Plateau.h

```
#include "GrosRobot.h"
#include "PetitRobot.h"


class Plateau
{
   public:
   Plateau();                                   // Constructeur du Plateau
   void ajouteverre(Verre* V);                  // Permet l'ajout d'un verre sur le terrain(ajout
                                                dans  tableauverre)
   void ajoutecadeau(Cadeau* C);                // Permet l'ajout d'un cadeau sur le terrain(ajout
                                                dans  tableaucadeau)
   void ajouteGrobot(GrosRobot* GR);            // Permet l'ajout d'un GrosRobot sur le terrain(ajout
                                                dans tableauGrobot)
   void ajouteProbot(PetitRobot* PR);           // Permet l'ajout d'un PetitRobot sur le terrain(ajout
                                                dans tableauProbot)
   void ajoutecerise(Cerise* C);                // Permet l'ajout d'une Cerise sur le terrain(ajout
                                                dans tableaucerise)
   void ajouteassiette(Assiette* A);            // Permet l'ajout d'une Assiette sur le terrain(ajout
                                                dans tableauassiette)
   void ajoutepanier(Panier* P);                // Permet l'ajout d'un Panier sur le terrain(ajout dans
                                                tableaupanier)
   void ajoutezone(Zone* Z);                    // Permet l'ajout d'un Zone sur le terrain(ajout dans
                                                tableauzone)
   void ajoutebougie(Bougie* B);                // Permet l'ajout d'une Bougie sur le terrain(ajout
                                                dans tableaubougie)
   std::vector<Verre*> gettableauverre();           // Renvoie le tableau des Verres
   std::vector<Cadeau*> gettableaucadeau();         // Renvoie le tableau des Cadeaux
   std::vector<Cerise*> gettableaucerise();         // Renvoie le tableau des Cerises
   std::vector<Assiette*> gettableauassiette();     // Renvoie le tableau des Assiettes
   std::vector<Zone*> gettableauzone();             // Renvoie le tableau des Zones
   std::vector<Bougie*> gettableaubougie();         // Renvoie le tableau des Bougies
   std::vector<Panier*> gettableaupanier();         // Renvoie le tableau des Paniers
   double score(std::string couleur);               // Renvoie le score de l'équipe de la couleur
                                                    donnée en paramètre

   private:
   std::vector<Verre*> tableauverre;            // Représente les Verres sur le terrain
   std::vector<Cadeau*> tableaucadeau;          // Représente les Cadeaux sur le terrain
   std::vector<GrosRobot*> tableauGrobot;       // Représente les Gros Robots sur le terrain
   std::vector<PetitRobot*> tableauProbot;      // Représente les Petits Robots sur le terrain
   std::vector<Cerise*> tableaucerise;          // Représente les Cerises sur le terrain
   std::vector<Assiette*> tableauassiette;      // Représente les Assiettes sur le terrain
```

```
    std::vector<Panier*> tableaupanier;                 // Représente les Paniers sur le terrain
    std::vector<Zone*> tableauzone;                     // Représente les Zones sur le terrain
    std::vector<Bougie*>tableaubougie;                  // Représente les Bougies sur le terrain

};
```

Fichier fonctionsutiles.h

```
#include "Plateau.h"
#include "calculChemin.h"

double recupangle(double angle);        // Permet de ramener un angle en radians entre -Pi et Pi
void trace(Plateau* Pl);                // Permet de tracer les cadeaux, les verres et les assiettes
float gradendegre(double rad);          // Permet la conversion des angles de radians en degrés
std::vector<Point> passage();           // Ressort un tableau de points de passage calculé à partir du
                                        calcul de chemin
```

Fichier FicheActionGR.h

```
#include "fonctionsutiles.h"

//Classe mère de toutes les fiches actions du Gros Robot

class FicheActionGR
{
   public:
   FicheActionGR(GrosRobot* Gr,Plateau* Pla);          //Fonction de création de la FicheActionGR
   virtual bool estactivable();                        // Fonction permettant de savoir si une
                                                       action est réalisable ou non
   virtual double score();                             // Fonction permettant de calculer le score
                                                       associé à chaque fiche
   virtual void affiche();                             // Fonction permettant d'afficher des
                                                       informations sur la ficheaction (pour vérifier
                                                       le bon fonctionnement)
   virtual void lanceaction();                          // Fonction permettant de lancer l'action

   protected:
   GrosRobot* pr;              // Pointe vers le GrosRobot correspondant à l'IA
   Plateau* Pl;                // Pointe vers le Plateau de jeu
```

Fichier FAOuvrirCadeau.h

```
#include "FicheActionGR.h"

//Pour chaque cadeau on a une fiche Action pour ouvrir un cadeau associé

class FAOuvrirCadeau : public FicheActionGR
{
 public :
   FAOuvrirCadeau(GrosRobot* rob, Plateau *pl, Cadeau* C);
   virtual bool estactivable();                  // Fonction permettant de savoir si une action est
                                                 réalisable ou non
   virtual double score();                       // Fonction permettant de calculer le score associé à
                                                 chaque fiche
   virtual void affiche();                       // Fonction permettant d'afficher des informations
                                                 sur la ficheaction (pour vérifier le bon
                                                 fonctionnement)
   virtual void lanceaction();                   // Fonction permettant de lancer l'action

 private:
   Cadeau* pc;              // Pointe vers le cadeau que l'on veut ouvrir

};
```

Fichier FARamasse1Verre.h

```
#include "FicheActionGR.h"

class FARamasse1Verre: public FicheActionGR
{
   public :
   FARamasse1Verre(GrosRobot* rob, Plateau *pl, Verre *V,Zone *Z);    // Constructeur de la
                                                                       FicheAction Ramasse1Verre
   bool estactivable();                          // Permet de savoir si elle est réalisable ou non
   double score();                               // Renvoie le score de la fiche
   void lanceaction();                           // Lance l'action associée à la fiche
   void affiche();                               // Permet l'affichage d'informations sur la fiche


  private:
   Verre* pv;                                  // Représente le verre à ramasser
   Zone* pz;                                   // Représente la zone où il faut ramener le verre

};
```

Fichier IA.h

```
#include "FAOuvrirCadeau.h"
#include "FARamasse1Verre.h"


class IA
{
   public:
   IA();                                          //Constructeur de l'IA
   void ajouteFA(FicheActionGR* FA);              // Ajoute une FicheAction à l'ensemble
   std::vector<FicheActionGR*> FArealisable();    // Renvoie les FicheActions réalisables
   FicheActionGR* meilleureFA();                  // Renvoie la meilleure FicheAction
   void lanceMFA();                               // Lance la meilleure FicheAction


   private:
   std::vector<FicheActionGR*> tabFA;              // Représente l'ensemble des FicheActions
                                                  existantes  pour le Gros Robot.
};
```