Projet de mécanique

Construction d'un robot autonome en vue de participer au concours EUROBOT 2008

Promoteur : Service de Génie Mécanique

Etudiants : Allard Paul-Henri
Antoine Vincent
Brine Stéphanie
Leclercq Stéphane
May Nicolas
Van Ruymbeke Denis

5e Mécanique

Année académique 2007 – 2008

Table des matières

1 Introduction **1**

2 Règlement & stratégie **5**
2.1 Présentation des règles . . . 5
2.2 Présentation de la stratégie . . . 7
2.3 Phase de jeu optimale . . . 8
2.4 Phase d'homologation . . . 11
2.5 En guise de conclusion . . . 11

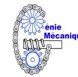
I Mécanique **13**

3 Introduction **14**

4 Cahier des Charges **16**

5 Propulsion et odométrie **17**
5.1 Odométrie . . . 17
5.2 Transmission . . . 19
5.2.1 Choix d'une transmission par renvois d'angle ou par courroies . . . 19
5.2.2 Dimensionnement de la transmission par courroies . . . 20
5.2.3 Switches . . . 22

6 Système d'avalement **24**
6.1 Etude des différents systèmes . . . 24
6.1.1 Pince . . . 24
6.1.2 Rouleau . . . 24
6.1.3 Soufflerie . . . 25
6.1.4 Pattes contrarotatives . . . 25
6.1.5 Courroie . . . 25
6.1.6 Simple chaîne . . . 26
6.1.7 Double chaîne . . . 26
6.2 Motorisation . . . 27
6.3 Bras de guidage . . . 29
6.3.1 Bras gauche . . . 29
6.3.2 Bras droit . . . 32
6.4 Adaptation au concours . . . 33
6.4.1 Guidage du bras gauche par tôles cintrées . . . 33



6.4.2 Languette . . . 34

7 Ascenceur **36**
7.1 Dimensionnement de la courroie . . . 37
7.2 Rouleaux . . . 38
7.3 Amorce de la montée des balles . . . 40
7.4 Positionnement . . . 41
7.5 Motorisation . . . 42
7.6 Interface A/C & A/T . . . 42

8 Éjection des balles **46**
8.1 Catapulte . . . 46
8.1.1 Comparaison des différentes possibilités . . . 46
8.1.2 Motorisation . . . 48
8.2 Trémie d'éjection . . . 49
8.2.1 Dimensionnement de l'espace disponible . . . 49
8.2.2 Tuyau . . . 49
8.2.3 Fixation . . . 50
8.2.4 Porte de sortie . . . 51
8.2.5 Amélioration et fiabilisation . . . 52

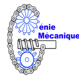
9 Conclusions de la partie Mécanique **53**

II Électronique **56**

10 Introduction **57**

11 Microswitches **60**
11.1 Introduction . . . 60
11.2 Principe . . . 60
11.3 Applications . . . 61
11.3.1 Microswitches de contact . . . 61
11.3.2 Microswitch de démarrage . . . 63
11.3.3 Carte de gestion des switches . . . 64
11.4 Carte d'extension . . . 65
11.4.1 Position du problème . . . 65
11.4.2 Principe . . . 66
11.4.3 Réalisation . . . 66
11.5 Blindage des fils . . . 67

12 Laser **70**
12.1 Introduction . . . 70
12.2 Principe . . . 70
12.3 Applications . . . 72
12.3.1 Traitement du signal . . . 72
12.3.2 Implantation dans le robot . . . 73



13 Plaque de démarrage **76**
13.1 Introduction . . . 76
13.2 Alimentation . . . 76
13.2.1 Première version . . . 76
13.2.2 Seconde version . . . 77
13.3 Choix de la stratégie . . . 79
13.4 Bouton RESET . . . 80
13.5 Plaque complète . . . 80

14 Évitement **83**
14.1 Position du problème . . . 83
14.2 Technologies à disposition . . . 83
14.2.1 La technologie infrarouge . . . 83
14.2.2 La technologie ultrason couplée à un signal radio . . . 86
14.2.3 Le BUMPER . . . 89

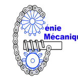
15 Conclusions de la partie Électronique **92**

III Informatique **93**

16 Introduction **94**
16.1 Introduction . . . 94
16.2 Généralités . . . 95

17 Programmation **96**
17.1 Introduction . . . 96
17.2 Code Microb Technology . . . 96
17.2.1 Spécifications du code . . . 97
17.2.2 Asservissement polaire . . . 97
17.2.3 Schématique de l'asservissement . . . 99
17.2.4 Description du code . . . 101
17.2.5 Fonctionnement du code sous interruption . . . 104
17.3 Code Nucléo . . . 105
17.3.1 Les machines d'état . . . 106
17.3.2 Les commandes associées aux machines d'état . . . 109
17.3.3 La fonction *Activate()* . . . 110
17.3.4 Remarque . . . 111
17.3.5 La gestion du déplacement . . . 111
17.4 Choix du code . . . 111

18 Asservissement & Régulation **113**
18.1 Asservissement . . . 113
18.1.1 Commande . . . 114
18.1.2 Consignes - Génération de la trajectoire . . . 114
18.1.3 Odométrie . . . 114
18.1.4 Odométrie dépendante et indépendante . . . 117
18.1.5 Asservissement . . . 118



18.1.6 Paramètres à régler . . . 118
18.2 Régulation . . . 121
18.2.1 Choix des moteurs . . . 122
18.2.2 Actions des PID's . . . 123
18.2.3 Réglage des paramètres du PID de vitesse . . . 125
18.2.4 Réglage des paramètres du PI de position . . . 128
18.2.5 Tuning . . . 133

19 Capteurs & Actionneurs **139**
19.1 Capteurs . . . 139
19.2 Actionneurs . . . 141
19.2.1 Moteurs DC . . . 141
19.2.2 Servomoteurs . . . 143

20 Stratégie **144**
20.1 Introduction . . . 144
20.2 Grafcets . . . 144
20.3 Codage de la stratégie et trajectoire de match . . . 145
20.3.1 *RobotGoToColorDistributor* . . . 145
20.3.2 *RobotPickUpBalls* . . . 150
20.3.3 *RobotGoToFrozenContainer* . . . 152
20.3.4 *RobotCatapult* . . . 152
20.4 Evitement . . . 153
20.4.1 La fonction *ManageOdoStates()* modifiée . . . 153
20.4.2 La manoeuvre d'évitement . . . 156

21 Compléments **160**
21.1 La connexion par port série . . . 160
21.2 Quelques notions de *C* . . . 161
21.2.1 Les instructions de préprocesseur . . . 161
21.2.2 Les énumérations . . . 162
21.3 L'interface *Debug* . . . 163
21.4 Plate-forme SolidEdge des déplacements . . . 164

22 Conclusion de la partie informatique **166**

23 Conclusions générales **167**

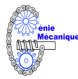
A Aide-mémoire en électronique **168**

B Détection du signal infrarouge **170**

C Datasheet du module ultrasonique SRF05 **173**

D Datasheet de l'émetteur ultrasons utilisé **175**

E Couverture des émetteurs ultrasons **177**



F Caractéristiques des moteurs et réducteurs MAXON utilisés pour les différents systèmes mécaniques 179

G Plans de l'ascenseur 182

H Plans du système d'avalement 184

I Plans des bras 186

J Plans des parois de finition du robot 188

K Plans des plate-formes de la struture 190

Bibliographie 193

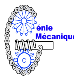
Table des figures

1.1 Mons Polytech Team 2008 4

2.1 Logo Eurobot 5
2.2 Thème du concours 5
2.3 Présentation générale de la table de jeu et de ses accessoires 6
2.4 Périmètre maximal du robot, déployé ou non 7
2.5 Position de départ du robot dans son carré bleu, touchant deux des bords de la table 8
2.6 Sortie du carré de départ 9
2.7 Rotation du robot pour aller vers le distributeur 9
2.8 Approche du robot vers le distributeur et avalement 10
2.9 Déplacement du robot vers le conteneur réfrigéré et tir des balles 10
2.10 Déplacement du robot jusqu'au conteneur standard et dépôt des balles 11

5.1 Précision des odomètres 17
5.2 Odomètre 18
5.3 Encombrement et montage des odomètres sur glissière 18
5.4 Transmission par renvois d'angle 19
5.5 Transmission par courroies 19
5.6 Ensemble roue/palier/pignon 20
5.7 Vue de l'encombrement moteur/courroie 21
5.8 Dimensionnement de la courroie 22
5.9 Pièce de fixation des switches en attaque indirecte - face avant 23
5.10 Pièce de fixation des switches fixée sur la plate-forme 23

6.1 Pince 24
6.2 Rouleau 25
6.3 Soufflerie 25
6.4 Courroie crantée 26
6.5 Maillons d'une chaîne 26
6.6 Système d'avalement à double chaîne 27
6.7 Mécanisme de réglage de la tension dans les chaînes 27
6.8 Mécanisme de réglage de la tension dans les chaînes (modélisation Solid Edge) . 28
6.9 Vue d'ensemble du système d'avalement 28
6.10 Vue d'ensemble du système d'avalement (modélisation Solid Edge) 29
6.11 Usinage des bras pour la fixation des switches en attaque indirecte 30
6.12 Bras gauche en position ouverte 30
6.13 Bras gauche en position fermée 31



6.14 Pièce de fixation du câble 31
6.15 Etude du cintrage du bras gauche 31
6.16 Bras "porte" en position fermée 32
6.17 Bras "porte" en position ouverte 32
6.18 Forme du bras droit 33
6.19 Description du principe du bras droit/Position fermée 33
6.20 Description du principe du bras droit/Position déployée 34
6.21 Bras droit non déployé 34
6.22 Languette poussant les balles à l'intérieur du robot 35

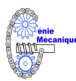
7.1 Ascenceur 2004 36
7.2 Ensemble ascenceur 37
7.3 Réglage de la tension dans la courroie à l'aide de vis qui fixent la hauteur de l'axe du rouleau bas dans ses lumières 39
7.4 Tiges "flipper" qui coincent les balles contre la courroie 39
7.5 Vue éclatée du rouleau côté moteur 39
7.6 L'ensemble rouleau moteur dans l'assemblage 40
7.7 Vue éclatée rouleau du bas 40
7.8 Rouleau du bas dans l'ensemble ascenseur 41
7.9 Dimensionnement de la languette d'amorce de montée des balles 41
7.10 Moteur, renvoi d'angle et fixation 42
7.11 Interface Ascenceur/Catapulte et Ascenceur/Trémie 43
7.12 Pièce d'interface, couplée avec la trémie d'éjection 44
7.13 Pièce d'interface Ascenceur/Trémie 44
7.14 Pièce d'interface Ascenceur/Catapulte 45

8.1 Vue d'ensemble de la catapulte, avec godet 47
8.2 Vue d'ensemble de la catapulte avec tôle cintrée 48
8.3 Module trémie d'éjection 49
8.4 Place disponible pour placer la trémie dans le robot (zone rosée) 50
8.5 Tuyau de la trémie d'éjection 51
8.6 Porte de la trémie, sens d'ouverture schématisé par la flèche 52
8.7 Vue du module trémie inséré dans le robot 52

9.1 Vue de côté du robot 54
9.2 Vue d'ensemble du robot, face avant 54
9.3 Vue d'ensemble du robot, bras en position fermée 55

10.1 Connecteurs JST 3 logements 58
10.2 Connecteur JST 3 logements câblé 58

11.1 Microswitch 60
11.2 Montage didactique 61
11.3 Montage en attaque indirecte du microswitch, vue du bras 62
11.4 Deux possibilités de câblage des microswitches 62
11.5 Boîtier de démarrage 63
11.6 Deux possibilités de câblage du microswitch de démarrage 64
11.7 Schéma de la carte de gestion des switches 65
11.8 Carte de gestion des switches 65



11.9 Principe de la carte d'extension pour les entrées I/O 66
11.10(a) Pin configuration du PCF8574 - (b) Modèle de dipswitch utilisé pour effectuer l'adressage du composant (le quatrième interrupteur n'étant connecté à rien) . . 67
11.11Board et schematic de la carte d'extension des switches 68
11.12Câble blindé utilisé pour éviter tout parasitage des signaux 69

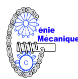
12.1 Phototransitor monté avec lentille 70
12.2 Balise laser recouverte de bande réfléchissante 71
12.3 Principe de détection grâce au rayon laser 71
12.4 Emplacements possibles des balises 72
12.5 Circuit électronique permettant le traitement du signal laser détecté par le phototransistor 73
12.6 Carte électronique de traitement du signal laser 74
12.7 Arrière du laser : carte de traitement de l'information et potentiomètres 75
12.8 Extrait du règlement concernant les dimensions des supports de balise 75

13.1 Batterie Ni-Cd 12 Volts 77
13.2 Première version du montage d'alimentation 78
13.3 Bouton d'arrêt d'urgence et bloc associé 79
13.4 Seconde version du montage d'alimentation 80
13.5 Interrupteur tripolaire 81
13.6 Montage relatif au switch de stratégie 81
13.7 Plaque complète intégrée dans le robot 82

14.1 Télémètre SHARP - principe de fonctionnement 84
14.2 Balise Infrarouge - principe de fonctionnement 85
14.3 Stratégie de déplacement du robot vers le goal à l'aide de la balise IR 85
14.4 Mat sous lequel sont placés les capteurs tels que le capteur IR, le module US et la cellule laser 85
14.5 Principe de fonctionnement de la balise de détection radio-ultrasons 87
14.6 Modélisation 3D de la balise d'évitement placée sur le robot adverse 88
14.7 Régulateur de tension de type 7805 89
14.8 Balise placée sur le robot adverse 89
14.9 Board et schématique de la carte électronique incluse dans la balise 90
14.10Bumper placé à l'avant du robot 91

17.1 Schéma pour l'envoi des PWM en distance et en angle vers les moteurs 98
17.2 Schéma pour la lecture des codeurs et la conversion en distance et en angle . . . 100
17.3 Schéma automatique en boucle fermée avec contrôleur PID et filtrage en rampe quadratique sur la position 101
17.4 Schéma de l'architecture du programme concernant l'asservissement 102
17.5 RobotState = RobotStopped 107
17.6 RobotState = RobotGoToColorDistributor 107
17.7 RobotState = RobotPickUpBalls 107
17.8 RobotState = RobotGoToFrozenContainer 108
17.9 RobotState = RobotCatapult 108

18.1 Commande classique d'un moteur DC en robotique 113
18.2 Profil de vitesse trapézoïdal 115



18.3 Profil de position assoscié au profil de vitesse trapézoïdal 115
18.4 Codeur impulsionnel incrémental 116
18.5 Disque interne d'un codeur impulsionnel 116
18.6 Signaux des voies A et B en quadrature dans un codeur 117
18.7 Principe simplifié de l'asservissement 118
18.8 Courbes caractéristiques $\omega = f(U)$ des quatre moteurs *Maxon* en possession du service 122
18.9 Asservissement minimaliste avec PID 123
18.10Réponse type d'un procédé stable avec contrôleur en boucle fermée pour une consigne de vitesse 124
18.11Représentation de la vitesse au cours du temps de chacun des moteurs pour un gain du PID de vitesse de 0,19 126
18.12Représentation de la vitesse au cours du temps de chacun des moteurs pour un gain du PID de vitesse de 0,21 127
18.13Représentation de la vitesse au cours du temps de chacun des moteurs pour un gain du PID de vitesse de 0,3 127
18.14Représentation de la vitesse au cours du temps de chacun des moteurs pour un coefficient intégral du PID de vitesse de 3,5 128
18.15Représentation de la vitesse au cours du temps de chacun des moteurs pour un coefficient intégral du PID de vitesse de 4,5 129
18.16Représentation de la vitesse au cours du temps de chacun des moteurs pour un coefficient intégral du PID de vitesse de 10 129
18.17Représentation de la vitesse au cours du temps de chacun des moteurs pour un coefficient dérivé du PID de vitesse de 0,005 130
18.18Représentation de la vitesse au cours du temps de chacun des moteurs avec un PID de vitesse et un P de position 131
18.19Effet du gain dans un contrôleur PI en position 132
18.20Représentation de la vitesse au cours du temps de chacun des moteurs avec un PID de vitesse et un PI de position 132
18.21Effet du coefficient intégrateur dans un contrôleur PI en position 133
18.22Représentation de la vitesse des moteurs de propulsion gauche et droit au cours du temps pour une tension de 12 Volt aux bornes de ces moteurs 134
18.23Profil des déplacements avec une décélération de 500 137
18.24Profil des déplacements avec une décélération de 4000 137

20.1 Grafcet relatif au temps 145
20.2 Grafcet relatif aux déplacements 146
20.3 Grafcet relatif aux actions à effectuer 147

21.1 Câble de connexion série 161
21.2 Plate-forme 2D SolidEdge pour la détermination des trajectoires 164
21.3 Plate-forme 2D SolidEdge avec dimensions pour la détermination des trajectoires 165

E.1 Calcul de la distance minimale à partir de laquelle les émetteurs US assurent une couverture totale, compte tenu de leur cône d'émission 177
E.2 Détermination de la couverture des signaux ultrason 178

Liste des tableaux

14.1 Codage ASCII des caractères envoyés respectivement par les trois balises 86
14.2 Vitesse de propagation des ondes dans l'air 87

Chapitre 1

Introduction

L'ingénieur actuel ne peut plus se contenter de travailler seul. Il est de plus en plus souvent confronté à des problèmes qu'il ne peut résoudre sans collaboration. Son métier ne peut plus se faire sans communication.

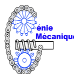
Dans le cadre des projets de 5eMécanique, la Faculté Polytechnique de Mons nous a offert la formidable opportunité de participer au concours EUROBOT 2008. Ce projet met à l'honneur bien évidemment la robotique, fusion parfaite de la mécanique, de l'électronique et de l'informatique ; mais aussi le travail d'équipe et la poursuite d'objectifs concrets.

Cette compétition se veut novatrice, notamment en favorisant l'ingéniosité et l'originalité mais aussi collégiale par son soucis de créer une *entité* Robotique en Belgique mais en aussi dans le monde entier. Elle incite au partage d'information et au dialogue entre les équipes qui ne sont finalement adversaires que lors des matches officiels. De nombreux forums et site internet sont mis à disposition des participants et sont régulièrement mis à jour.

Nous sommes donc 6 étudiants mécaniciens à s'être lancé dans cette formidable aventure avec pour seuls acquis nos connaissances scolaires et notre motivation. Cette expérience était avant tout, et ce pour chacun de nous, un défi de taille et une manière à la fois sympathique et valorisante de terminer notre cursus à la F.P.Ms.

Le présent rapport est divisé en trois parties relatives chacunes à un des trois pôles. Avant d'entrer dans le vif du sujet, nous développerons les principales contraintes du réglement ; contraintes qui ont bien sûr guidé notre raisonnement pour choisir la stratégie qui fera également l'objet d'un développement préliminaire. Et pour terminer, nous tirerons les conclusions de notre travail.

Les annexes contiennent quelques fiches *mode d'emploi* qui ont été écrites afin de faciliter l'apprentissage des différentes techniques que nous avons pu maîtriser cette année et pour faciliter l'emploi des blocs modulaires que nous avons conçus. Il est fortement conseillé à nos successeurs de les lire.



Membres de l'équipe

La Mons Polytech Team 2008 est composée de six étudiants mécaniciens entourés par toute une série de professeurs, techniciens, collègues et amis. Chaque étudiant était responsable d'une partie que nous appelerons *technique* et d'une partie *administrative* pour mener à bien ce projet.

Paul-Henri Allard était responsable mécanique et délégué aux relations extérieures, il s'est principalement occupé de la conception et de la réalisation des systèmes mécaniques ainsi que des inscriptions et des recherches de sponsorings.

Vincent Antoine était responsable informatique et trésorier, il s'est principalement occupé de l'asservissement, du réglage des PIDs et de l'agencement du câblage dans le robot. Il a également tenu la comptabilité à jour durant toute la durée du projet.

Stéphane Brine était responsable mécanique et déléguée règlement/stratégie. Elle s'est principalement occupé de la conception/réalisation/mise en plan des systèmes mécaniques et a été la personne de référence pour tout ce qui concerne le cahier des charges et le choix de la stratégie.

Stéphane Leclercq était responsable électronique et coordinateur de projet. Il s'est principalement occupé du design et de la réalisation des cartes électroniques ainsi que du câblage. Il a également été le coordinateur de projet et s'est occupé du grafcet de la stratégie.

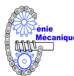
Nicolas May était responsable électronique et secrétaire. Il s'est occupé du design et la réalisation des cartes électroniques et de l'implantation de celles-ci dans le robot. Il a également rédigé les procès verbaux lors des réunions.

Denis Van Ruymbeke était responsable informatique et webmaster. Il s'est principalement occupé de l'écriture des machines d'états et de la mise en place de la stratégie. De plus, il a tenu à jour le site internet de l'équipe tout au long du projet.

Remarques

Il est évident que ces postes ne sont que théoriques. La plupart des décisions ont été prises de manières collégiales et résultent de l'expérience de chacun. Le pôle électronique a travaillé main dans la main avec le pôle informatique afin de permettre le transfert le plus aisé possible des informations. Le pôle mécanique a fourni les plate-formes de test au pôle informatique et a effectué les réparations nécessaires. Le pôle mécanique a dimensionné ses logements de cartes en fonction des desiderata des deux autres pôles.

Ce projet est avant tout un partage des connaissances de chacun et une fusion de nos expériences respectives.



Remerciements

Participer au concours EUROBOT est avant tout une aventure formidable faite de rencontres, de discussions, de franches rigolades ainsi que de moments de stress intenses. Nous tenions tout particulièrement à remercier les personnes suivantes :

La FPMs en général et tout particulièrement M. le Recteur Calogero Conti et M. le Doyen Paul Lybaert pour nous avoir fourni le budget nécessaire à cette épopée de même que pour leur soutien lors de la compétition.

M. Enrico Filippi pour nous avoir soutenus tout au long de cette année et pour nous avoir fait confiance jusqu'à la dernière minute.

M. Pierre Dehombreux pour ses conseils et son soutien lors de la compétition.

M. Edouard Rivière-Lorphèvre pour son expérience en la matière, ses conseils toujours éclairés et sa bonne humeur.

M. Christophe Chariot pour son expérience, sa sympathie, ses goûts musicaux discutables et son aide précieuse apportée en électronique.

M. Marc Delhaye pour ses conseils avisés, son aide au niveau du grafcet et pour son engouement perpétuel.

M. Marcel Rémy pour ses explications claires et ses connaissances tendant vers l'infini en matière de PID.

M. Pierre Lecomte et M. Frédéric Coquelet pour leur aide précieuse apportée en électronique, pour leur soutien lors du concours et leur disponibilité.

M. Ludovic Dufranne pour sa disponibilité, son aide plus que nécessaire au niveau du programme informatique et sa bonne humeur.

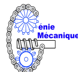
M. Laurent Pinchart pour sa patience, ses explications toujours limpides et son ouverture d'esprit.

L'équipe des Kamikazes, pardon Kadroïds, pour les conseils fournis en flux continu et avec qui il aurait été plus qu'amusant de partir en coupe d'Europe mais ceci est une autre histoire. Un merci particulier à Steve et J-F ainsi qu'à Philippe et Benja.

Les 5 Zoulous de l'écoshell avec qui ce fut un véritable plaisir de partager la halle de Génie Mécanique aussi bruyants soient-ils. Merci à eux d'être venus nous soutenir au PASS.

Les membres des Services Généraux pour leur aide et disponiblité. Un merci particulier à Fernand pour l'usinage, et Fredo pour les commandes.

Aurore, notre maman à tous, pour son soutien et ses crêpes !



Laura et Pauline pour l'inspiration électronique.

Toutes les personnes venues nous supporter et nous ayant supportés tout au long de cette formidable année.

And last but not least, un tout grand merci à l'immense Laurent Vergari, l'homme du soir, pour sa disponibilité, sa bonne humeur perpétuelle, sa rigueur mécanique, ses conseils avisés et son aide lors des nocturnes de dernière semaine. Mais avant tout, merci à lui de nous avoir appris à nous servir de nos mains.

Fig. 1.1 – Mons Polytech Team 2008

Chapitre 2

Règles de jeu, stratégie et homologation

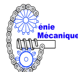
2.1 Présentation des règles de jeu

Le thème du concours 2008 est "Mission to Mars". Le but du concours est de construire un robot entièrement autonome capable de prélever et de ramener dans les meilleures conditions possibles des preuves de vie de Mars, le tout dans la joie et dans la bonne humeur caractérisant la compétition d'Eurobot Open. L'idée est la suivante : deux robots sont sur la table de jeu.

Fig. 2.1 – Logo Eurobot

Fig. 2.2 – Thème du concours

Chaque robot appartient à une équipe différente, la bleue ou la rouge. Sur la table et disponible en périphérie, des balles : bleues, rouges et blanches. Le but du jeu est que le robot de notre équipe marque le plus de points possibles en déposant les balles aux bons endroits, en fonction de sa couleur et de la couleur des balles, sans déborder sur le temps imparti, 90 secondes par match.
A la figure 2.3, on peut voir la table de jeu. Elle mesure 2100 sur 3000 mm. Différents accessoires sont attachés à la table : les distributeurs verticaux, le distributeur horizontal, les goals et la rigole à l'avant de la table. Tous ces éléments ont des dimensions et une géométrie précise. Pour



plus de précision, le plus simple est de consulter le règlement. Brièvement, les distributeurs verticaux contiennent chacun cinq balles d'une couleur unique ; un de balles bleues à côté du carré de départ bleu côté du fond de la table, un de balles blanches côté latéral et symétriquement du côté rouge. La zone de départ est un carré de 500 mm de côté, dans lequel le robot doit se situer entièrement au départ. De plus, le robot doit toucher les deux côtés du carré représenté par les bordures de la table. Des balles sont aussi présentes dans le conteneur horizontal ; il ne sera pas décrit car on ne l'utilise pas (nous devrions disposer de capteur de couleur pour cela car il contient des balles bleues et rouges qui se dispersent aléatoirement sur la table après actionnement de l'ouverture du distributeur, ce que nous n'utilisons pas). Pour marquer des

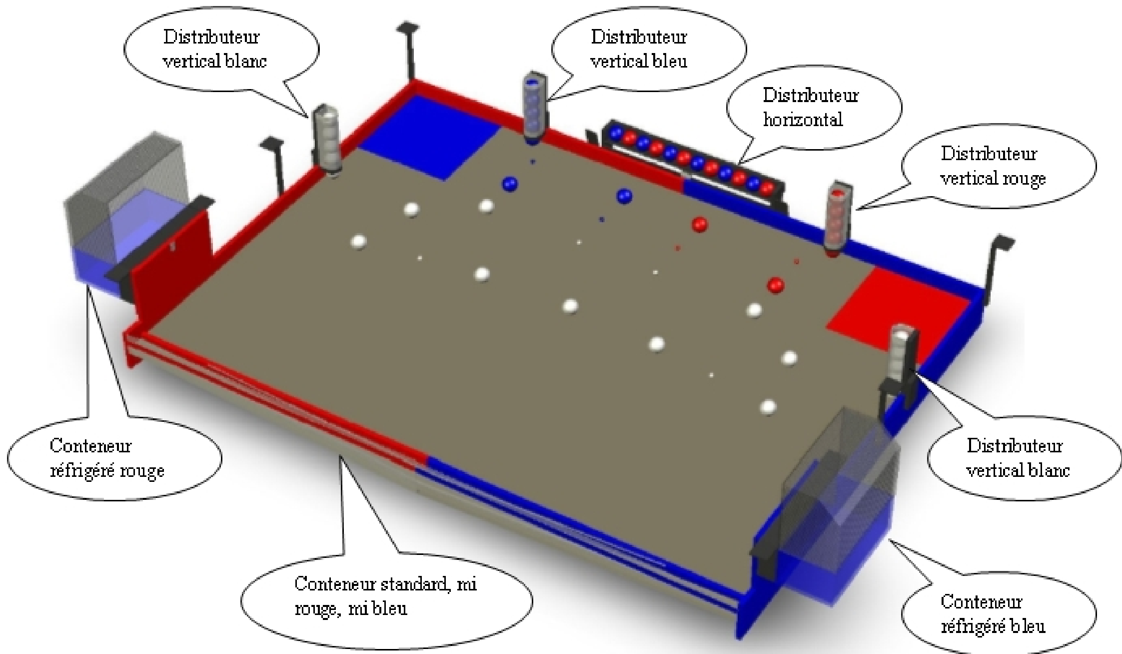


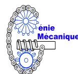
Fig. 2.3 – Présentation générale de la table de jeu et de ses accessoires

points, les possibilités sont les suivantes, si on considère appartenir à l'équipe bleue pour plus de facilité de présentation :

- Sortir du carré de départ : 1 point ;
- Placer une balle dans le conteneur réfrigéré bleu : 2 points ;
- Placer un balle bleue n'importe où dans le conteneur standard : 2 points ;
- Placer un balle blanche dans le conteneur standard côté bleu : 1 point ;
- Réussir une combinaison dite de "conservation d'un échantillon" : *blanc – bleu – blanc* n'importe où dans le conteneur standard : 3 points (si la combinaison est réalisée du côté rouge du conteneur, le bonus nous revient ainsi que les deux points pour la balle bleue mais pas les points des balles blanches) ;
- Terminer un match égalité avec le robot adverse : 1 point ;
- Terminer un match en étant gagnant : 3 points.

Des pénalités peuvent aussi être distribuées :

- Si on place un échantillon rouge ou blanc dans le conteneur réfrigéré bleu ;
- Si on garde des balles de l'adversaire dans notre robot (prévenu par un avertissement la première fois que la situation se présente, pénalisé les fois suivantes) ;
- Si on transporte ou déplace volontairement plus de cinq balles simultanément dans le robot (avertissement la première fois, pénalisé les fois suivantes) ;



- Si on entre en collision avec le robot adverse (avertissement la première fois, pénalisé les fois suivantes) ;
- Si on enfreint une quelconque règle, que l'on reçoit un avertissement de la part d'un arbitre, et qu'on réitère la même action.

Le robot à construire doit respecter certains impératifs. Notamment pour ne citer que les plus importants : il doit être totalement autonome, respecter des dimensions maximales imposées (cfr figure 2.4), ne pas être dangereux que ce soit vis-à-vis du public ou en risquant d'endommager les accessoires de jeu, etc. Il doit aussi disposer d'une limitation dans la possibi-

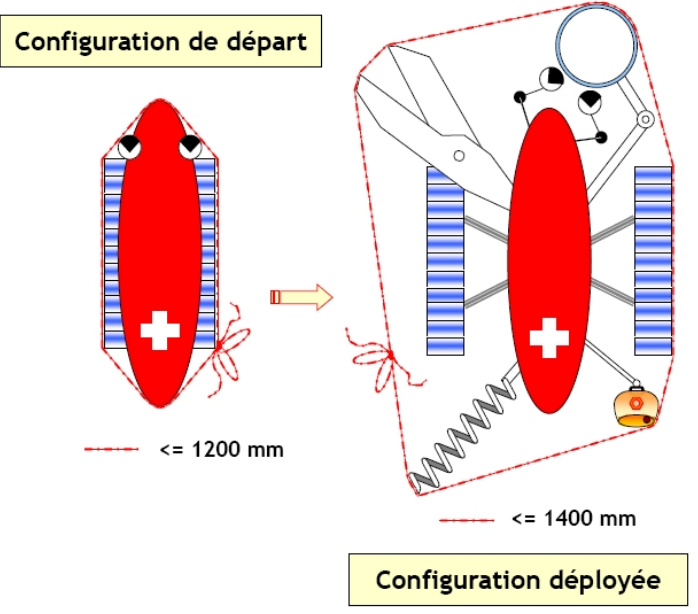


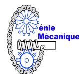
FIG. 2.4 – Périmètre maximal du robot, déployé ou non

lité de transporter plus de cinq balles, limitation mécanique (place maximale dans le robot pour cinq balles) ou non (système de comptage électronique par exemple). Le robot doit disposer d'un mât au centre du robot, avec une plate forme de 80 × 80 mm située à 42 cm du sol afin de pouvoir recevoir une balise adverse dans le cas où l'équipe que l'on affronte en a besoin pour son système d'évitement. Le robot doit être constitué d'une seule et même partie solidaire, et ne pas laisser de morceaux sur la table en cours de jeu. Le robot doit disposer d'un bouton d'arrêt d'urgence permettant de rendre le robot inactif. Le démarrage du robot doit être possible en tirant une corde dont la longueur est imposée.

Pour toute information complémentaire, consultez directement le règlement disponible en annexe.

2.2 Présentation de la stratégie

Pour la stratégie, l'idée la plus importante à garder à l'esprit est la simplicité des déplacements. Notre point fort est la mécanique, et il faudra donc que notre mécanique s'adapte à l'électronique et l'informatique, plus capricieuses, afin de simplifier leur travail, et d'assurer une fiabilité maximale. Pour cela, l'option la plus adéquate est de minimiser les déplacements ainsi que tout ce qui demande de la précision, le déplacement du robot n'étant pas suffisamment



maîtrisé.
La stratégie développée est la suivante : l'objectif principal est de prendre les cinq balles de notre couleur situées dans le distributeur vertical attenant au carré de départ, et d'aller les placer dans le conteneur réfrigéré de la bonne couleur ou dans la rigole (conteneur standard) à n'importe quel endroit afin de marquer dix points. Cette stratégie simple nécessite très peu de déplacement, et nous permet d'accéder à dix points quasi certainement. Mais où aller placer les balles ?

- Les placer dans le conteneur réfrigéré permet d'assurer les points. Il est impossible que le robot adverse vienne les enlever. Mais ce choix nécessite de les porter en hauteur et d'utiliser le tir.
- Les placer dans le conteneur standard à l'avant de la table nous permet de marquer aussi dix points, quelque soit l'endroit où l'on viendra les déposer. Mais l'inconvénient majeur est le risque que l'on encoure face à un adversaire capable de retirer les balles de la rigole. Les points ne sont pas garantis face à une équipe très bien armée. Par contre, le dépôt des balles est plus aisé car on vient les déposer vers le bas.

Dès le départ, nous avons opté pour le tir dans les goals. Les raisons majeures sont la sûreté des points ainsi que la technologie de la catapulte, déjà utilisée lors d'autres concours, notamment dans le robot de 2004.

2.3 Présentation de la phase de jeu optimale

Pour faciliter la description des règles du jeu, considérons que nous appartenons à l'équipe bleue. La stratégie est illustrée aux figures ci-dessous.

1. Le robot est dans sa zone de départ, nous disposons de trois minutes afin de placer le robot dans la zone de départ bleue, au contact des deux bords. Si l'une des équipes n'est pas prête au bout des trois minutes, elle peut être déclarée forfait (figure 2.5).

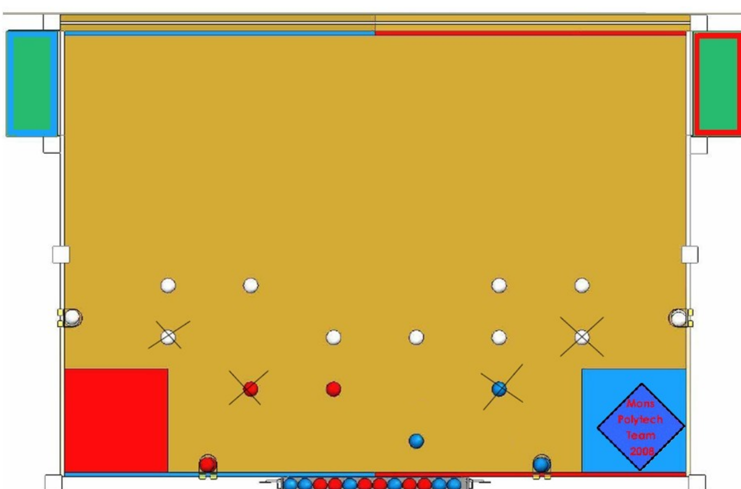
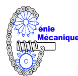


FIG. 2.5 – Position de départ du robot dans son carré bleu, touchant deux des bords de la table

2. Le robot démarre. Il sort du carré de départ, ce qui permet de marquer le premier point. L'angle de départ est fixé par le pôle informatique en fonction de leur préférence pour la programmation de la trajectoire (figure 2.6).



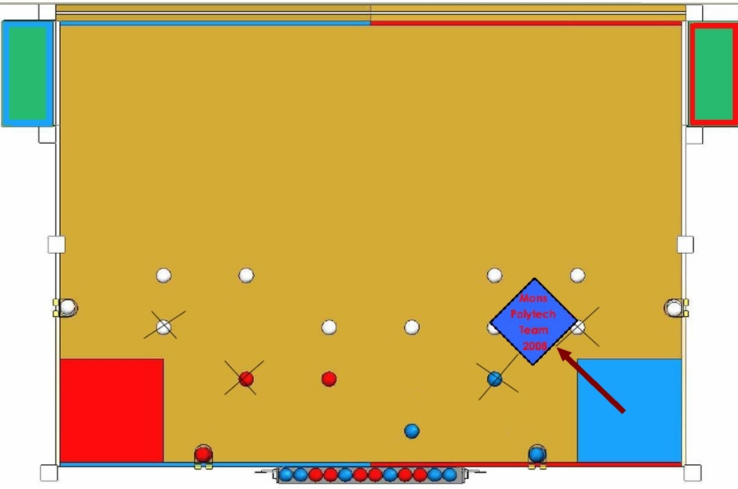


FIG. 2.6 – Sortie du carré de départ

3. Le robot effectue une rotation pour que, lors du déplacement suivant, il puisse s'approcher du distributeur de balles bleues. Les bras s'ouvrent pendant cette phase ou à un autre moment, en fonction des préférences du pôle informatique (figure 2.7).

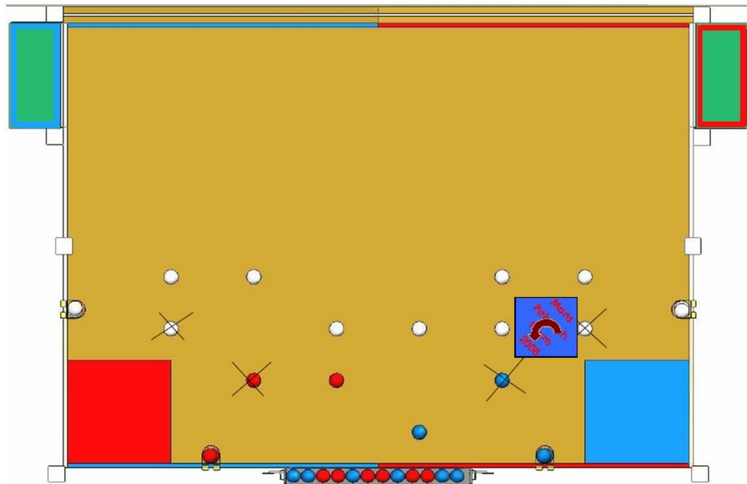
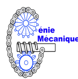


FIG. 2.7 – Rotation du robot pour aller vers le distributeur

4. Le robot avance vers le distributeur, l'encadre avec ses bras déployés. Le système d'avalement peut se mettre en route afin de capter les cinq balles de couleur du distributeur (figure 2.8).
5. Le robot peut aller vers le goal pour y catapulter les balles. Pour cette manoeuvre, il doit traverser la table afin de se rendre vers le conteneur réfrigéré bleu. Cette phase est la plus critique car elle nécessite une ligne droite assez précise car elle est assez longue. Le choix de la trajectoire jusqu'au conteneur est laissé à l'appréciation de l'informatique, en fonction de leur capacité à gérer les déplacements. Durant ce mouvement, le risque de croiser le robot adverse est lui aussi accru (figure 2.9).

   Suite à un problème technique survenu peu de temps avant le concours, la stratégie a dû être revue. En effet, le moteur de la catapulte a rendu l'âme ; nous avons dû remplacer la



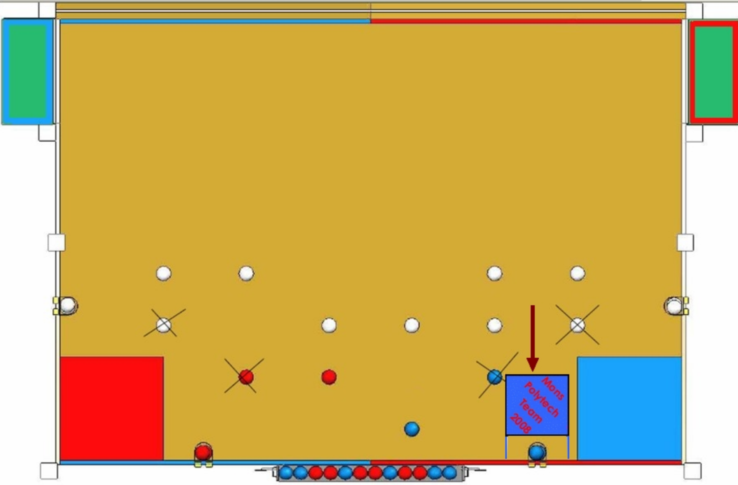


FIG. 2.8 – Approche du robot vers le distributeur et avalement

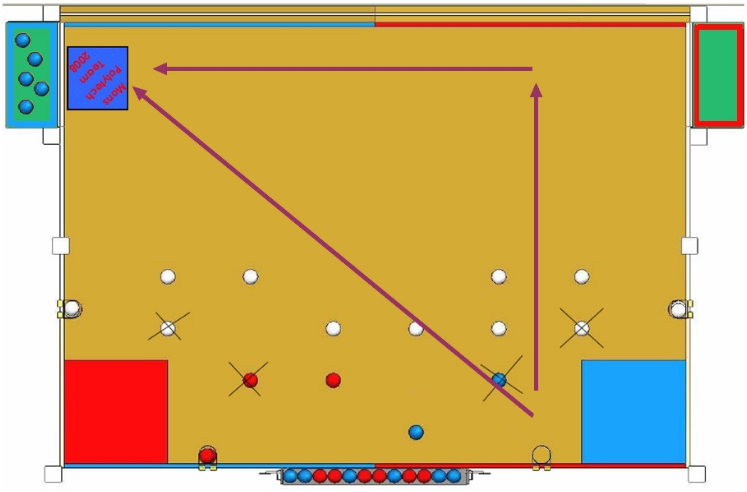
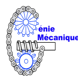


FIG. 2.9 – Déplacement du robot vers le conteneur réfrigéré et tir des balles

catapulte vers une sortie différente, et plus simple, vers la rigole à l'avant de la table. Le point (5) de la stratégie est alors remplacé par le (6).

6. Le robot se déplace en ligne droite jusque le conteneur standard et y déverse ses balles. L'avantage de ce déplacement est que moins de précision y est requise, car la rigole à atteindre est présente tout le long de la table de jeu. La procédure d'évitement sera aussi plus simple, puisqu'on peut éviter l'adversaire et se rendre à un tout autre point du conteneur standard. Le seul point négatif est que les points ne sont pas assurés, l'adversaire peut retirer nos points s'il en est technologiquement capable (figure 2.10).

Pour ces deux stratégies qui permettent de marquer idéalement un nombre de points équivalent, il n'est pas nécessaire de disposer d'accessoires électroniques tel qu'un capteur de couleur pour distinguer des balles. Les déplacements sont minimisés, à la fois en distance parcourue et en précision, ce qui devrait faciliter le travail de programmation et de réglage de paramètres.



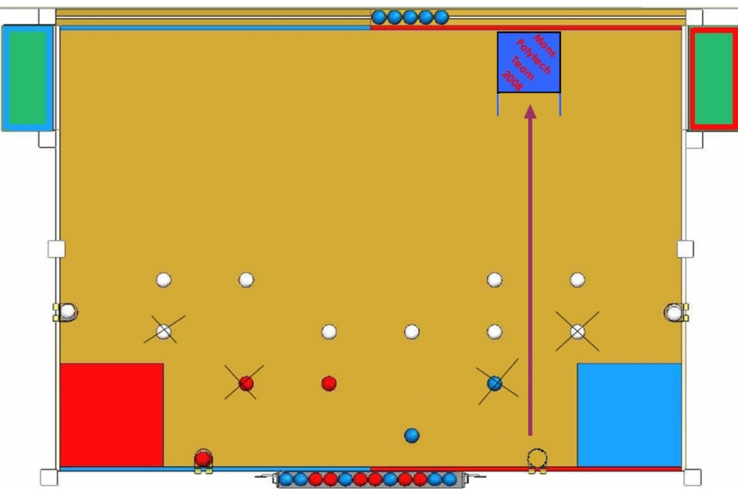


FIG. 2.10 – Déplacement du robot jusqu'au conteneur standard et dépôt des balles

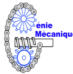
2.4 Phase d'homologation

La phase d'homologation, dont la réussite est l'objectif du projet, se compose de différentes étapes :

1. Le robot ne peut pas être dangereux, pour le public et pour les éléments de jeu ou comporter d'accessoires interdits (une liste d'exemple est disponible dans le règlement du concours).
2. Le robot ne doit pas excéder les dimensions imposées, périmètre non déployé et déployé, hauteur.
3. Démontrer à l'arbitre que le robot peut sortir de son carré de départ. Cette vérification est liée au point suivant.
4. Démontrer à l'arbitre lors d'une phase de démonstration que le robot, seul sur l'aire de jeu, peut marquer un point. Pour cela, la stratégie est pleinement tentée devant l'arbitre.
5. Démontrer que le robot ne peut pas volontairement transporter ou déplacer à son avantage plus de cinq balles. A cet effet, nous avons prévu d'expliquer à l'arbitre que notre stratégie ne nous permet pas de le faire, et que nous disposons d'un mécanisme pour compter les balles entrant dans le robot. Après la cinquième, le système qui avale les balles est arrêté.
6. Démontrer à l'arbitre que le robot peut éviter un adversaire. Pour ce test, l'arbitre place un robot factice sur la trajectoire de notre robot. L'arbitre s'attend à voir une réaction de la part de notre robot, qui doit éviter ou au moins s'arrêter face au robot factice, sans risquer de le faire dévier de sa trajectoire. En effet, il est par exemple permis de toucher le robot factice à condition de ne pas le déplacer. Dans notre cas, nous avons mis au point un système d'évitement détaillé dans le chapitre consacré à l'électronique.

2.5 En guise de conclusion

Finalement, nous pouvons conclure que nos choix s'avérèrent payants. En effet, l'idée de déposer les balles dans les conteneurs standards plutôt que de les catapulter dans les conteneurs réfrigérés était la meilleure. Premièrement, nous avons pu constater que c'était là la solution



choisie par la majorité des équipes, tant en Coupe de Belgique qu'à Eurobot en Allemagne. Deuxièmement, différents facteurs physiques (telle la tension des filets des goals) diminuaient fortement la précision des tirs. Nous avons pu constater a posteriori que les équipes qui avaient optés pour la solution du tir ne furent que très peu récompensées.

Un autre problème rencontré durant la Coupe de Belgique fût la difficulté d'avaler les balles au bas des distributeurs verticaux. Les tables du concours ne respectaient en effet pas le cahier des charges. Grâce à notre capacité de réaction en situation de crise, nous pûmes réagir à temps que pour nous qualifier pour les huitièmes de finale.

Première partie

Mécanique

Chapitre 3

Introduction

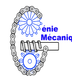
Dans cette partie du rapport, nous détaillerons les étapes successives de la construction mécanique de notre robot. Au départ d'une stratégie claire et précise, nous élaborerons les différents systèmes inhérents à celle-ci. Comment développer un robot le plus modulable et le plus fiable possible ? Telle était la question qui nous préoccupait sans cesse. Comme dans toute construction mécanique qui se respecte, un cahier des charges a été élaboré. Lors du développement du projet, celui-ci a dû être scrupuleusement respecté, parce que répondant parfaitement aux attentes de la stratégie.

Le but du jeu cette année restant relativement simple, nous avons tenté de ne pas nous égarer dans d'interminables élaborations au fonctionnement non démontré. De par notre formation de mécanicien, nous avons concentré nos forces afin d'élaborer un robot le plus **fiable** possible. En effet, nous ne pouvions nous permettre la moindre erreur mécanique, des fiabilisations devant être réalisées dans les deux autres domaines : électronique et informatique.
Malgré ces restrictions, nous n'avons cependant pas pu déroger à certains organes vitaux pour notre robot. La mise en place de capteurs de type *micro-switches* s'avéra plus que nécessaire pour la détection des bords de la table d'une part, et le comptage des balles entrant dans le robot d'autre part.

Chacune des différentes parties a fait l'objet de la même démarche :

1. Réflexion et choix de la meilleure solution possible ;
2. Modélisation sous Solid Edge afin d'imaginer l'encombrement et l'interaction avec les autres éléments, et vérifier les possibilités d'association ;
3. Élaboration de plans cotés détaillés à partir de la modélisation Solid Edge ;
4. Réalisation pratique ;
5. Validation du système sur l'aire de jeu ;
6. Implémentation de celui-ci sur le prototype et mise en oeuvre avec les autres procédés ;
7. Optimisation et réglages définitifs.

En appliquant cette méthodologie avec rigueur, nous créons notre robot en développant nos capacités à mener à bien un projet. Car plus qu'un simple concours, la Coupe de Robotique a été pour nous un réel projet d'apprentissage.



Remerciements

Nous souhaitions exprimer toute notre reconnaissance à M. Laurent Vergari, sans qui la construction de notre robot n'aurait pas été possible. Sa disponibilité, son expérience et sa rigueur nous furent d'une aide précieuse. Nous remercions également les opérateurs de l'atelier, et plus particulièrement Fernand, pour les pièces réalisées avec grand brio. Enfin, un grand merci à M. Edouard Rivière-Lorphèvre pour ses conseils judicieux et son expérience en la matière. Puissent-t-ils trouver ici toutes les éloges qui leur sont dues.

Chapitre 4

Cahier des Charges

Afin de réaliser notre stratégie, le robot devra être capable de :

1. Se déplacer sur la table, de manière autonome ;
2. Avaler les balles se trouvant dans les distributeurs verticaux ;
3. Stocker les balles dans le robot en exploitant la hauteur disponible par un système qui permet également l'élévation de celles-ci, dans le cas de l'utilisation de la catapulte ;
4. Expulser les balles du robot.

Il s'agit donc d'un cahier des charges très précis. Les différents systèmes devront être modulables et le robot final le plus fiable possible.

Compléments sur le règlement

Le règlement nous impose certaines contraintes d'un point de vue mécanique. Le périmètre, dans la configuration de départ du robot, ne peut excéder **1200 mm**. *"Le périmètre du robot est défini comme l'enveloppe convexe qui est comprise dans la projection verticale du robot sur le terrain."* [1] Il s'agit donc de prendre des précautions. Nous nous fixerons donc comme base une plaque de 280 par 280 mm afin d'obtenir un facteur de sécurité convenable. Dans sa configuration déployée, le robot ne peut dépasser les **1400 mm**.
La hauteur du robot n'excédera pas 350 mm, en excluant le mât du support de balise, d'éventuels capteurs intégrés sur le mât du support de balise et le bouton d'arrêt d'urgence. Aucune contrainte de dimension n'est donnée concernant le déploiement du robot en-dessous du niveau de la table. Tous les autres systèmes, y compris des systèmes obligatoires, doivent être contenus dans le volume spécifié ci-dessus. Toutes les parties du robot doivent rester physiquement connectées, donc le robot ne peut pas déposer certaines de ses parties sur l'aire de jeu.
Par conséquent, toutes ces restrictions nous limitent dans la modélisation de notre robot.

[1] Mission to Mars, Eurobot, Règlement 2008.

Chapitre 5

Propulsion et odométrie

Le système de propulsion est composé de deux roues motrices, placées sur un axe moteur. Pour une plus grande aisance au déplacement et à la rotation, l'axe moteur doit être centré au mieux sur la plate-forme de base du robot. Pour assurer une rotation efficace, les roues doivent être écartées autant que possible.

5.1 Odométrie

Les odomètres sont présents sur l'axe moteur, car ils doivent être alignés sur les roues afin de pouvoir les suivre sans décalage en rotation. Pour des raisons de précision, leur position doit être la plus extérieure possible. En effet, la mesure d'une rotation se fera alors sur un plus grand nombre de tours de roue si le rayon de l'arc de cercle effectué est plus important (voir figure 5.1), le périmètre de la roue de l'odomètre étant fixe. Les odomètres sont les modèles

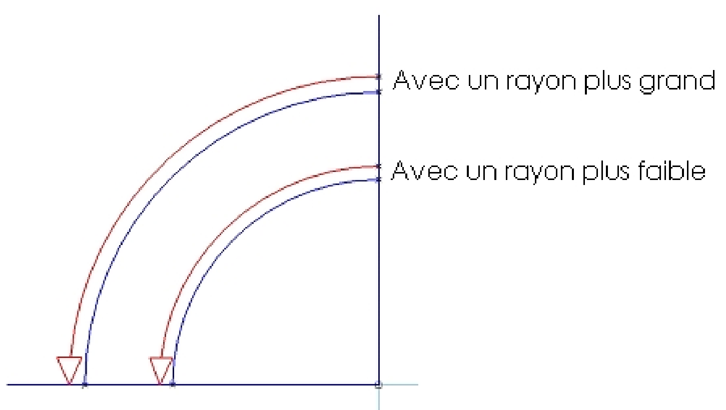
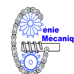


Fig. 5.1 – Précision des odomètres

présentés à la figure 5.2. Comme nous disposons de ces modèles, le montage va être adapté à leur géométrie. Les odomètres sont des odomètres de roue libre, c'est-à-dire qu'ils disposent de leur propre roue pour la mesure. Pour assurer une mesure la plus proche possible des déplacements réellement effectués, les odomètres de roues libres sont plus fiables que les odomètres présents dans les moteurs. En effet, nous avons placé les odomètres sur une glissière verticale avec un ressort qui plaque la roue au sol. Grâce à ce mécanisme simple et peu encombrant, on assure que l'odomètre enregistre un déplacement uniquement quand le robot est vraiment en train d'avancer ; s'il est coincé contre un bord par exemple, et que la roue motorisée patine,



l'odomètre associé à la roue motorisée indiquerait un déplacement, ce qui n'est pas le cas dans la réalité. Ce système prend néanmoins de la place. Comme vous pouvez le voir sur la figure 5.3,

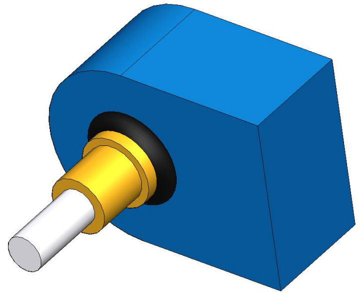


FIG. 5.2 – Odomètre

page 18, l'encombrement dépend de la roue libre, de l'odomètre et de la glissière, qui est la seule liaison entre le système et la plate forme. Le réglage du système et le suivant : la glissière doit permettre de rattraper une différence par rapport à la hauteur nominale à laquelle on place la plate-forme. Le ressort doit fixer la position de repos de l'odomètre légèrement plus bas que les roues motrices. On assure donc de forcer la roue de l'odomètre à rouler sur la table de jeu même si les roues motrices patinent. Il ne faut pas non plus que le ressort soit trop fort, il soulèverait alors le robot qui perdrait toute adhérence. L'encombrement total du système odomètre sur

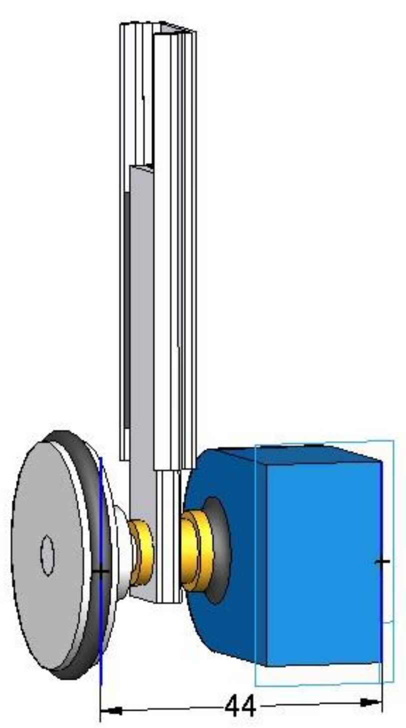
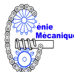


FIG. 5.3 – Encombrement et montage des odomètres sur glissière

glissière est de 44 mm (cfr figure 5.3, page 18). Il reste donc 280 mm -2×44 mm = 192 mm pour placer les deux roues motrices sur l'axe moteur. Il n'y a donc pas assez de place pour placer les moteurs sur le même axe que les roues.



5.2 Transmission

5.2.1 Choix d'une transmission par renvois d'angle ou par courroies

Pour réaliser la transmission, deux solutions s'offrent à nous :

1. Utiliser des renvois d'angle (figure 5.4, page 19), ce qui permettrait de placer les moteurs soit vers l'arrière de l'axe moteur, soit au-dessus de la plate-forme, mais toujours perpendiculairement à l'axe moteur. Après analyse d'encombrement, seule la solution de placer les moteurs au-dessus de la plate-forme serait possible.
2. Utiliser une transmission par courroie crantée, ce qui permettrait de placer les moteurs où la place est disponible, parallèlement à l'axe moteur.

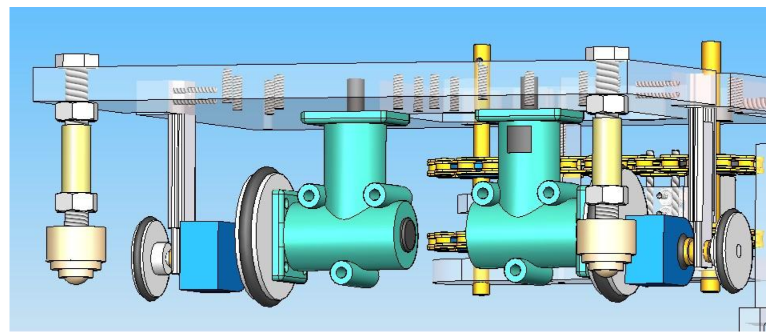


FIG. 5.4 – Transmission par renvois d'angle

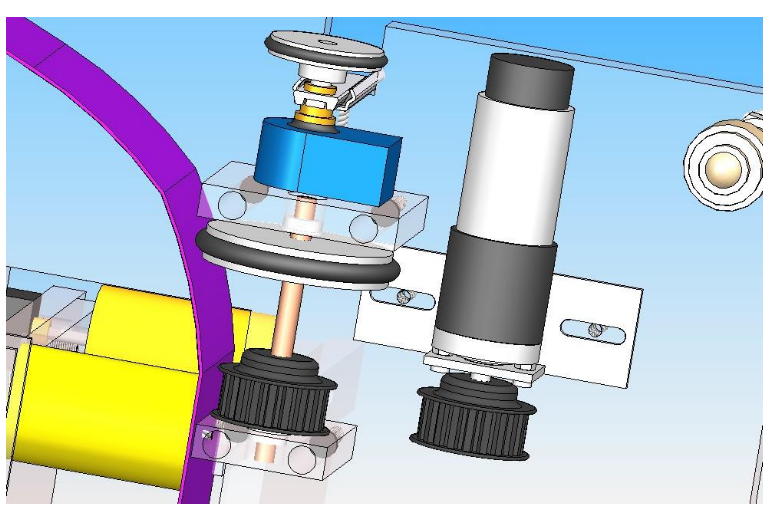
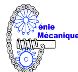


FIG. 5.5 – Transmission par courroies

La question est restée posée assez longtemps, car différents paramètres entrent en jeu. Notamment, la facilité de mettre en place les renvois d'angles qui font aussi office de palier pour les roues, ce qui permettrait une facilité de conception et un gain de temps quant à la réalisation. C'est pour ces raisons que nous avons initialement opté pour cette solution pendant la phase de développement des systèmes mécaniques.
L'autre argument à discuter est l'encombrement et le positionnement relatif des éléments. Les courroies permettent une disposition beaucoup plus modulable. C'est pour ces raisons que nous avons définitivement choisi d'utiliser des courroies.



En effet, grâce aux courroies, l'ensemble du système concernant le déplacement sera placé sous le robot, ce qui permet d'optimiser l'encombrement, et de faciliter une éventuelle intervention mécanique sur les éléments concernant la propulsion sans devoir déranger les autres systèmes, la propulsion étant maintenant complètement accessible par le dessous du robot ! Cet agencement a d'ailleurs été profitable à de nombreuses reprises, notamment lors du desserrage d'une vis de pression, ou d'une désolidarisation de la roue et de l'O-ring l'entourant.
Le choix final est d'utiliser des courroies, pour lesquelles un dimensionnement est nécessaire, un choix du type de pignons, ainsi qu'une conception des paliers pour supporter les roues.

5.2.2 Dimensionnement de la transmission par courroies

La plateforme est placée à 80 mm du sol. Cela laisse un espace suffisant pour que les balles de 72 mm de diamètre puissent se déplacer sous la plateforme, sans non plus perdre de la place au-dessus. En effet, la partie supérieure de la plate-forme accueillera l'ascenseur, la catapulte ou la trémie, les cartes électroniques et les batteries.
Il faut donc respecter les critères suivants :

1. Placer sur l'axe moteur : 2 × 2 paliers, 2 roues, 2 pignons, 2 systèmes "odomètre de roue libre" ;
2. Fixer la plateforme à 80 mm du sol ;
3. Aligner l'ensemble pour faciliter le travail de programmation. Il est impératif de veiller à la précision lors de la réalisation pratique.

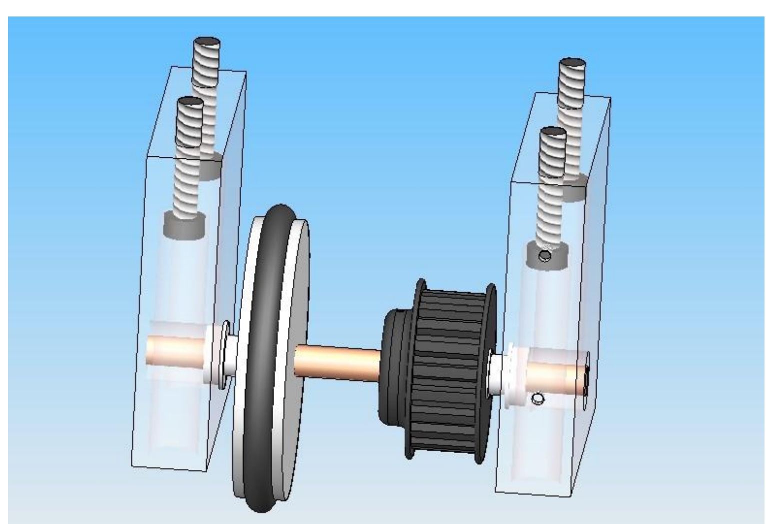
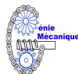


FIG. 5.6 – Ensemble roue/palier/pignon

La figure 5.6, page 20 représente l'assemblage de la roue sur son axe moteur, entrainé par le pignon qui soutient la courroie, l'axe étant maintenu à la hauteur voulue grâce à deux paliers qui contiennent des roulements à collerette.
La roue est fixée axialement et en rotation sur l'axe à l'aide d'une vie de pression à tête plate. Il en est de même pour le pignon. Par conséquent, des méplats ont dû être usinés préalablement sur l'axe rectifié que nous avons utilisé. Les vis de pression sont des M3.
La position du pignon est déterminée axialement par son homologue relatif au moteur. En effet,



il faut placer le moteur qui a une dimension imposée et qui ne doit pas dépasser du bord ! (cfr figure 5.7, page 21).
Après avoir respecté la position du pignon, les paliers sont placés extérieurement, en optimisant

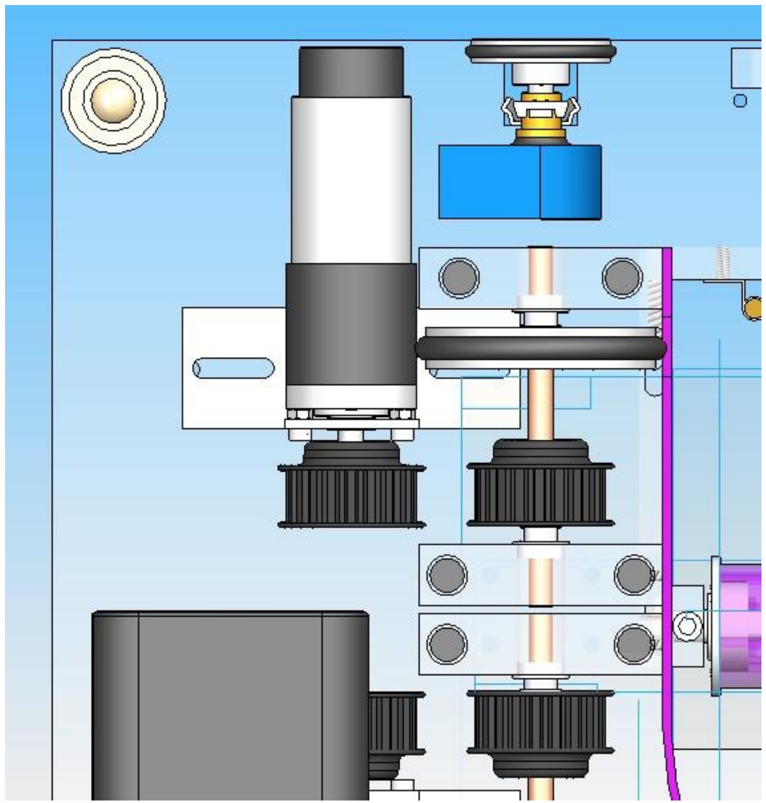
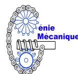


Fig. 5.7 – Vue de l'encombrement moteur/courroie

la place (un palier au plus près des odomètres en gardant une distance de sécurité, et un palier de l'autre côté du pignon). La roue est placée au plus près du palier voisin de l'odomètre.
L'axe moteur est un axe rectifié en acier de 6 mm de diamètre.
Les roulements à collerette sont placés sur l'axe et dans le palier en plexiglas. La bague intérieure est serrée sur l'axe car c'est elle qui voit l'effort tourner. Les roulements sont placés en opposition pour permettre de placer des entretoises afin de les bloquer axialement dans les deux sens.
Les paliers en plexiglas ont une dimension étudiée en fonction de :

1. L'effort subi : une plaque de plexiglas de 15 mm d'épaisseur est exigée ;
2. La hauteur est fixée par les 80 mm exigé pour la position de la plateforme ;
3. La largeur est minimisée, mais pour que l'entièreté du système du déplacement soit montable et démontable par le bas, la fixation sera assurée par le bas. Il faut un perçage suffisant pour laisser passer la tête de la vis.

Choix de la courroie : Maintenant que le montage est fixé, il reste à choisir la courroie. Il en existe de deux types : à profil de dents curviligne (ω = courroies "HTD") ou trapézoïdal ("ZR" ou en "T"). Nous avons opté pour les courroies ω car elles sont plus performantes, plus résistantes, elles adhèrent mieux aux poulies, et en plus nous en avions en réserve ! Pour calculer la dimension de la courroie, l'outil Solid Edge a été d'une grande utilité. En fonction de l'encombrement du moteur et de la position imposée des roues, on peut estimer l'entraxe



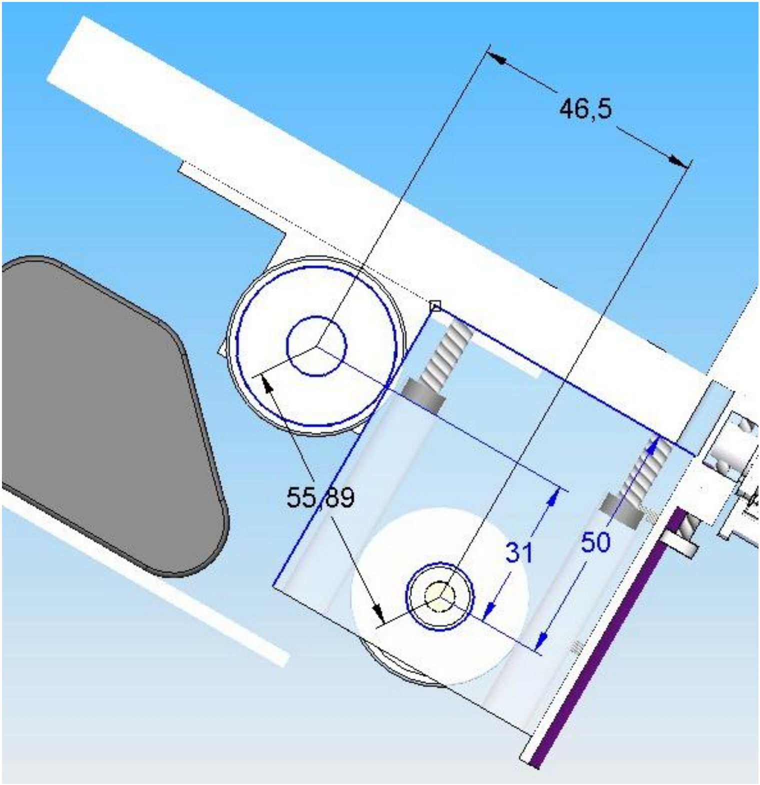
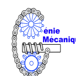


FIG. 5.8 – Dimensionnement de la courroie

optimal entre les deux poulies. Comme on peut le voir sur la figure 5.8, page 22, l'entraxe est de 56 mm, ce qui donne pour la longueur de la courroie $56 \times 2 + 2 \times \pi \times r_{poulie} = 225$ mm
Remarque : l'entraxe a été trouvé égal à une valeur différente, mais a été adapté pour permettre le choix d'une courroie d'une valeur de catalogue.

5.2.3 Switches

Des switches ont dû être placés sur le robot afin de détecter les bords de la table et le robot adverse. Les critères mécaniques étaient la hauteur bien précise de ceux-ci, afin de détecter les différents bords de table, et de ne pas dépasser le périmètre autorisé. Plus d'informations concernant les switches sont disponibles dans la partie électronique de ce rapport. Toutefois, signalons que le principe utilisé était celui d'une *attaque indirecte* afin de ne pas endommager le switch au contact. Une tige, filetée en ses deux extrémités, sert de relais pour la détection du contact. Côté extérieur, un écrou à tête ronde avec une rondelle sont fixés (afin d'empêcher *d'embrocher* une balle). Côté intérieur, une plaque en acier de 1 mm d'épaisseur est serrée entre deux écrous sur la tige. En position de repos, le switch est donc enclenché. Lors d'un contact, la tige s'enfonce dans la pièce, la plaque d'acier recule et le switche se déclenche.
La figure 5.9, page 23, présente la pièce de fixation des switches en attaque indirecte sur la face avant du robot. Le switche du bas sert à détecter le bord de 22 mm de hauteur qui nous signale donc notre présence face au conteneur standard dans lequel nous déposerons les balles. Nous remarquerons la complexité de la pièce : la découpe en arc de cercle au-dessus est destinée à laisser passer le moteur, la découpe en bas à droite permet quant à elle à la roue de l'odomètre de roue libre de rouler sans interférences.
La pièce montée sur le robot est présentée à la figure 5.10.



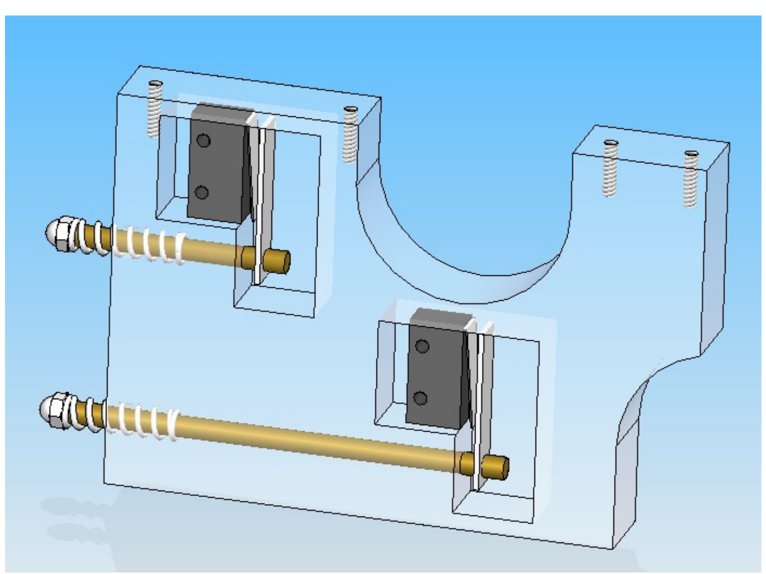


Fig. 5.9 – Pièce de fixation des switches en attaque indirecte - face avant

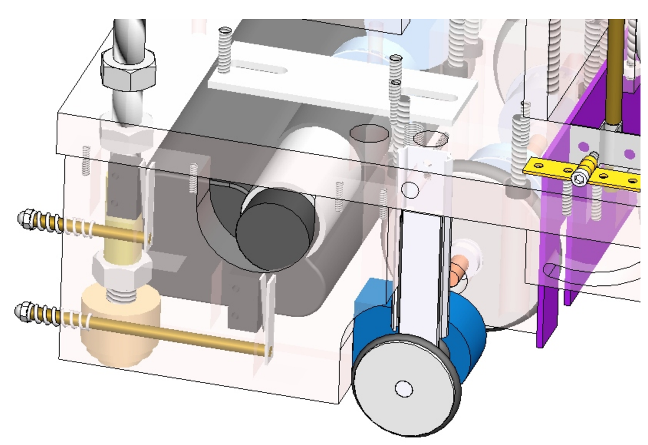


Fig. 5.10 – Pièce de fixation des switches fixée sur la plate-forme

Chapitre 6

Système d'avalement

Selon la stratégie établie, ce système devra être capable d'avaler un maximum de 5 balles. Pour ce faire, plusieurs possibilités ont été imaginées. Cette partie a pour but d'en détailler les principales et d'expliquer nos choix.
De manière générale, tous les systèmes ont été imaginés afin de venir se caler contre le bord de la table avec deux bras se déployant aux extrémités du robot. Ces bras, munis de switches en leur extrémité, détectent donc automatiquement le premier contact aux bords. Cette information sera utilisée dans le code.

6.1 Etude des différents systèmes

6.1.1 Pince

Dans un premier temps, il a été envisagé d'utiliser un système de pince (figure 6.1, page 24) permettant d'aller chercher de manière certaine les balles au bas du distributeur vertical. Le désavantage principal, et celui pourquoi l'idée n'a pas été retenue, est que le fonctionnement n'est pas continu, chose que nous voulions éviter.

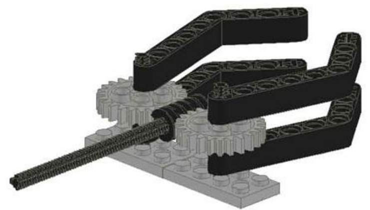


Fig. 6.1 – Pince

6.1.2 Rouleau

L'avantage principal dans ce cas précis est le fonctionnement en continu, une amélioration donc par rapport au cas précédent. De plus, il s'agit d'un prototype déjà validé par nos prédécesseurs (figure 6.2, page 25). Toutefois, l'incapacité totale à aller agripper les balles en-dessous du distributeur nous oblige à abandonner cette solution.

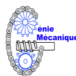


Fig. 6.2 – Rouleau

6.1.3 Soufflerie

L'utilisation d'une soufflerie (figure 6.3, page 25), et son adaptation à notre cas, présenterait l'avantage énorme d'avoir une vitesse très élevée, favorisant de ce fait un gain de temps en match. Les raisons de notre abandon vis-à-vis de cette dernière sont :

1. Les balles sont trouées ;
2. Un encombrement excessif du système ;
3. Et enfin, l'interdiction formelle de son emploi par le nouveau règlement !

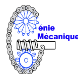
Fig. 6.3 – Soufflerie

6.1.4 Pattes contrarotatives

L'utilisation de deux pattes contrarotatives nous rapproche de notre but. Nous arrivons à l'élaboration d'un système fonctionnant en continu cette fois. Toutefois, une simple réflexion quant à l'encombrement réduit de notre robot et la place pour laisser passer les balles nous oblige une fois de plus à rendre les armes. Cependant, la solution finale est toute proche !

6.1.5 Courroie

La solution serait la suivante : entraîner une courroie crantée (figure 6.4, page 26) par un pignon moteur, courroie sur laquelle on fixerait une languette permettant d'éjecter les balles en un minimum de temps. Cependant, la fixation de cette languette pose problème ! Un collage serait insuffisant pour maintenir le tout au passage sur les pignons. De même, un rivetage engendrerait des interférences aux mêmes endroits. Comble de malchance, un phènomène de "vissage" de la courroie, dû au moindre défaut de parallélisme des deux axes, est susceptible d'apparaître. Ce phénomène étant renforcé par le fait que les axes sont verticaux. La courroie



pourrait alors se dérober après quelques cycles. Ne prenons pas le moindre risque et venons-en alors à la solution optimale pour notre application : la chaîne!

Fig. 6.4 – Courroie crantée

6.1.6 Simple chaîne

Un système d'avalement à chaîne résoud nos problèmes de fixation de languette. Comme le montre la figure 6.5, page 26, un axe monté serré entre deux maillons peut aisément être remplacé par une petite tigé filetée, celle-ci servant alors de support à la languette. Un problème,

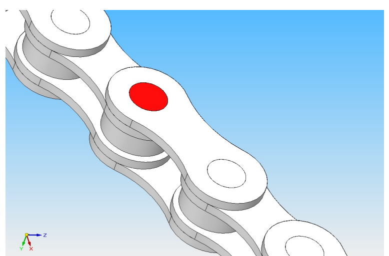


Fig. 6.5 – Maillons d'une chaîne

et non le moindre, subsiste toutefois : la torsion! Quelle sera l'action d'une balle sur cette languette? Inévitablement, cette dernière se tordra jusqu'à faire dérailler la chaîne. Problème réglé si nous envisageons maintenant un système à double chaîne.

6.1.7 Double chaîne

Finalement, nos pérégrinations nous mènent à un système d'avalement dont un système à double chaîne entraîne une languette qui éjecte les balles en prenant le contact sur leur diamètre. Avantages :

1. Fonctionnement en continu;
2. Rigidité face à la torsion;
3. Tension des chaînes réglable.

Inconvénients :

1. Difficulté d'assurer le parallélisme des axes, ce qui entraîne une précision de montage parfaite;
2. Des logements pour les roulements à billes devront être prévus. Ils nécessitent un usinage qui ne permet aucune erreur d'alignement;
3. Réglage parfait de la tension des chaînes :

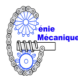


FIG. 6.6 – Système d'avalement à double chaîne

(a) Si trop tendu, on force sur les pignons.
(b) Si trop lâche, déraillement assuré.

Afin de veiller à la tension des chaînes, un petit mécanisme a été mis au point. Comme le montre les figures 6.7 et 6.8, celui-ci permet de déterminer la position de la vis par rapport au blocage axial en venant serrer un écrou contre une butée.

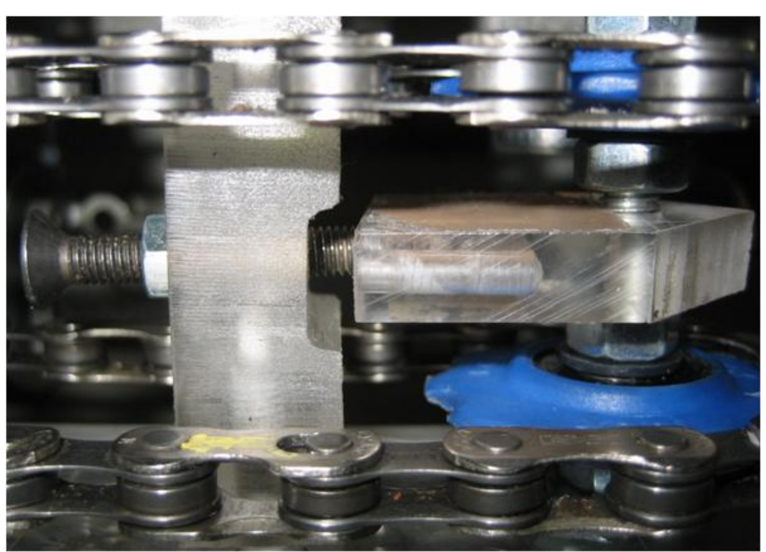


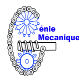
FIG. 6.7 – Mécanisme de réglage de la tension dans les chaînes

Enfin, deux vues d'ensemble (figures 6.9 et 6.10) nous permettent de visualiser le système mis en place.

Il est important de noter que celui-ci a été conçu et testé sur table. Il nous a permis d'éjecter les 5 balles en 5 passages de languette. Le système est donc validé !
Ajoutons enfin qu'un *switch à moustache* a été placé à l'entrée du robot afin de compter les 5 balles réglementaires.

6.2 Motorisation

Le moteur qui anime le système est un Maxon Brushless (référence 200 863), avec un réducteur 128 :1 (référence 143 988).



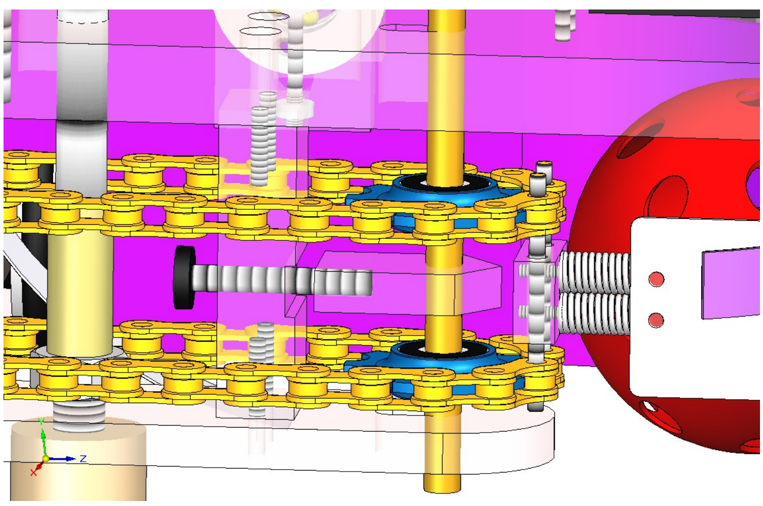


Fig. 6.8 – Mécanisme de réglage de la tension dans les chaînes (modélisation Solid Edge)

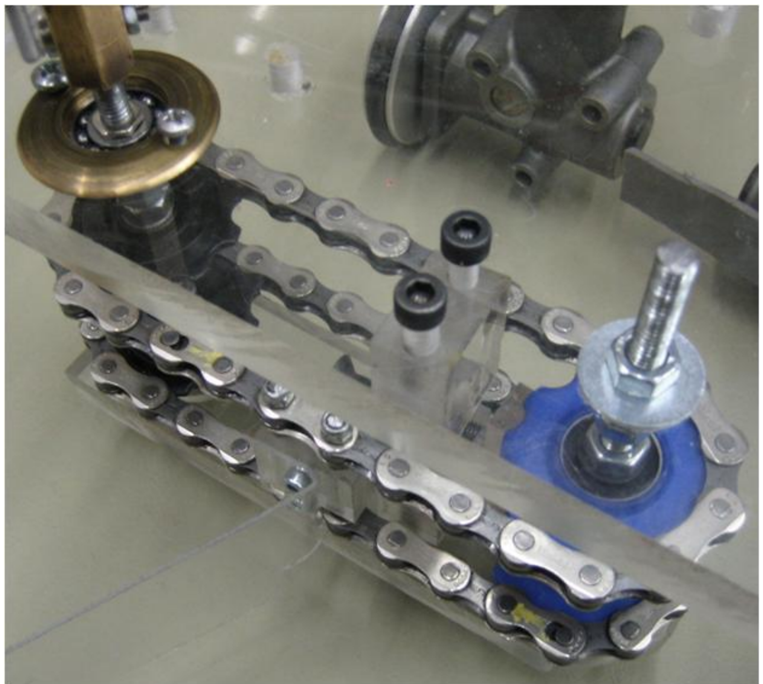
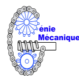


Fig. 6.9 – Vue d'ensemble du système d'avalement

Les caractéristiques du motoréducteur sont les suivantes :
- 24 V
- 20 W
- 130 à 160 $\frac{tr}{min}$
- 1,2 $N \times m$

Les datasheets relatives aux moteurs et réducteurs Maxon sont disponibles en annexe F.



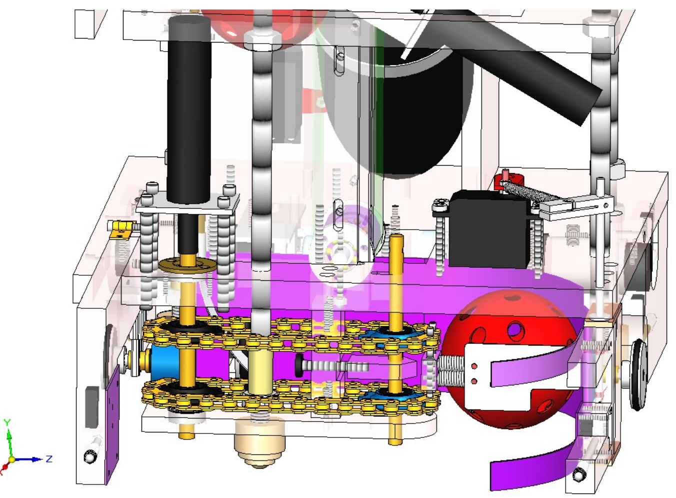


FIG. 6.10 – Vue d'ensemble du système d'avalement (modélisation Solid Edge)

6.3 Bras de guidage

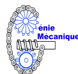
Des bras de guidage vont devoir être placés de part et d'autre du système d'avalement. Ils auront pour but de :

1. Fixer la position du robot par rapport au bord de la table ;
2. Guider les balles à l'intérieur du robot pour l'un des bras ;
3. Bloquer le passage des balles pour l'autre bras ;
4. Encadrer le distributeur vertical.

Ces bras devront respecter plusieurs contraintes. En effet, le périmètre déployé du robot ne peut excéder les 1400 mm. Ayant déjà un robot de 1200 mm de périmètre (compte tenu d'un facteur de sécurité), chaque bras pourra donc faire un maximum de 10 cm (switches compris). Pour fixer les idées, appelons bras gauche celui du côté duquel les balles entrent dans le robot, et bras droit l'autre.
Pour des raisons évidentes de résistance aux chocs, ces bras seront usinés dans de l'aluminium de 15 mm d'épaisseur. En effet, il s'agit des seules parties du robot qui seront déployées lors du match. Il convient donc d'éviter tout risque de rupture. De plus, ces bras seront usinés afin de venir y placer les switches à l'intérieur, d'où le choix d'une épaisseur minimale requise. L'usinage des bras est présenté à la figure 6.11, page 30.

6.3.1 Bras gauche

Avant toute chose, il s'agit de prendre en compte les contraintes que ce bras devra respecter. En position ouverte (voir figure 6.12, page 30), celui-ci ne peut avoir une longueur supérieure à 10 cm, switch compris. En position fermée (voir figure 6.13, page 31), il doit être capable de laisser passer la languette du système d'avalement qui tournera en continu, d'où sa forme creuse. Il doit être capable de s'ouvrir en début de match, et de se refermer lorsque 5 balles auront été prises. En effet, le règlement stipule que le robot ne peut pas transporter plus de 5 balles. Par conséquent, notre bras devra pouvoir s'ouvrir et se refermer. Nous avons fait choix d'un bras pivotant autour d'un axe. Un servomoteur relié par un câble à cet axe permet de passer d'une position à l'autre, tandis qu'un ressort de torsion le contraint à se mettre en position



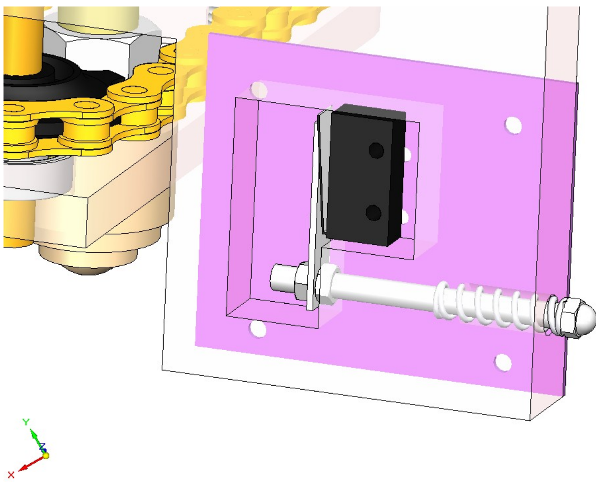


Fig. 6.11 – Usinage des bras pour la fixation des switches en attaque indirecte

ouverte. Pour résumer, en position fermée, le bras est maintenu par la tension du câble relié au servomoteur. Lors de l'ouverture, le servomoteur relâche la tension de ce câble et le ressort de torsion le pousse à prendre la position ouverte. Un blocage en rotation sera également prévu pour que le bras ne pivote pas de plus de 90 degrés.
Le fait d'utiliser un système de câble et de ressort est motivé par un aspect important : la collision. En effet, si l'adversaire en match venait à percuter le bras (fixé rigidement au servomoteur), le servomoteur devrait reprendre l'effort lié au choc. En utilisant notre méthode, un éventuel choc serait encaissé par le ressort de torsion. La pièce de fixation du câble à l'axe

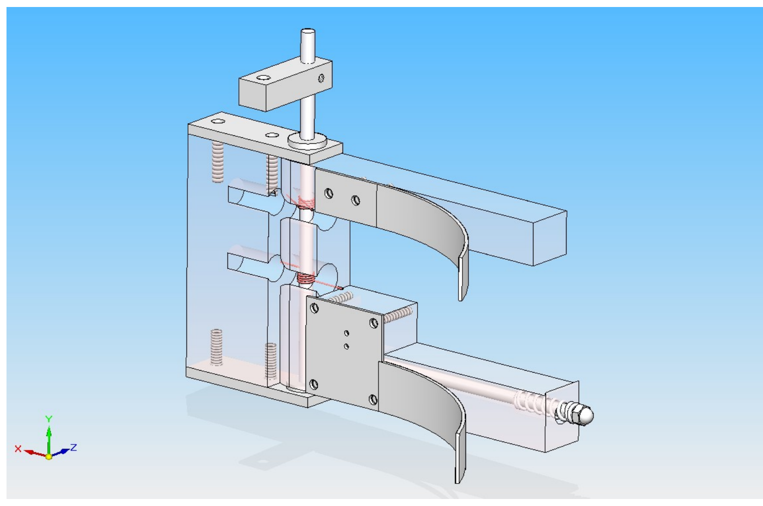
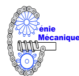


Fig. 6.12 – Bras gauche en position ouverte

tournant est présentée à la figure 6.14. Elle est inspirée de la fixation des câbles de frein sur les vélos classiques, pour que le câble ne soit pas gêné en rotation. La figure 6.17, page 24 nous montre le montage définitif.

Guidage : En ce qui concerne le guidage des balles, un raisonnement assez théorique peut être tenu. Les balles doivent être guidées de manière optimale et nous devons donc prévoir un certain cintrage du bras. Prenons comme référence l'axe autour duquel tourne la languette. Un



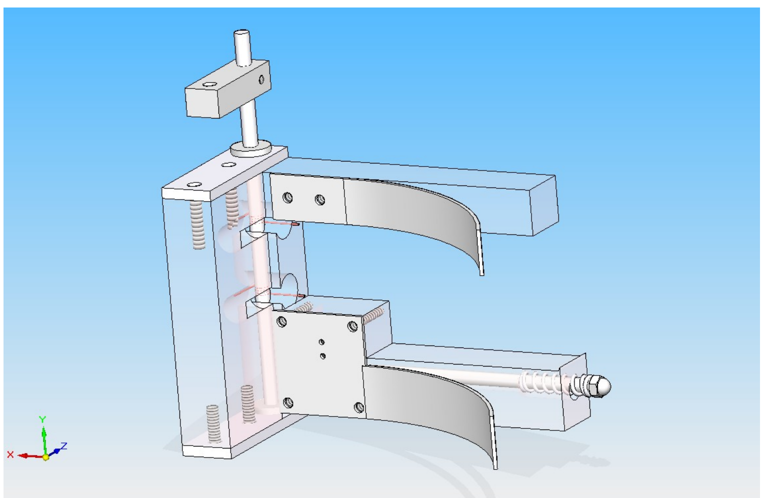


FIG. 6.13 – Bras gauche en position fermée

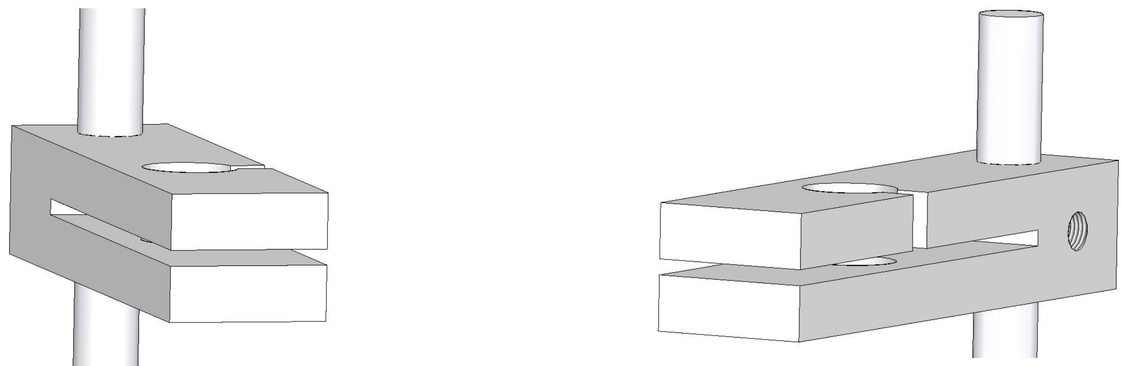


FIG. 6.14 – Pièce de fixation du câble

premier diamètre est fixé par l'encombrement du pignon et de la chaîne. Connaissant alors le diamètre de la balle (72 mm), nous obtenons le diamètre extérieur du chemin des balles. Ces dernières se déplaceront donc entre ces deux diamètres extrêmes.

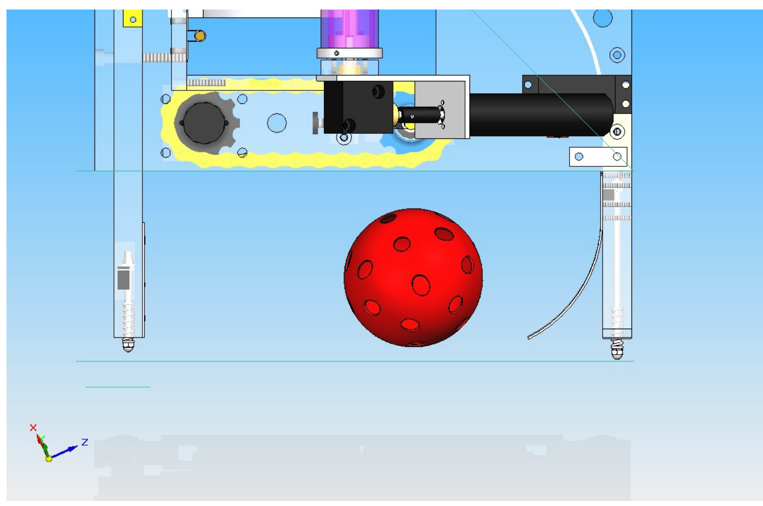
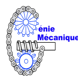


FIG. 6.15 – Etude du cintrage du bras gauche



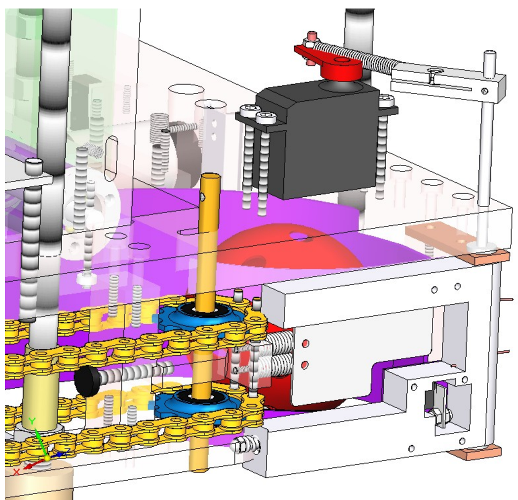


FIG. 6.16 – Bras "porte" en position fermée

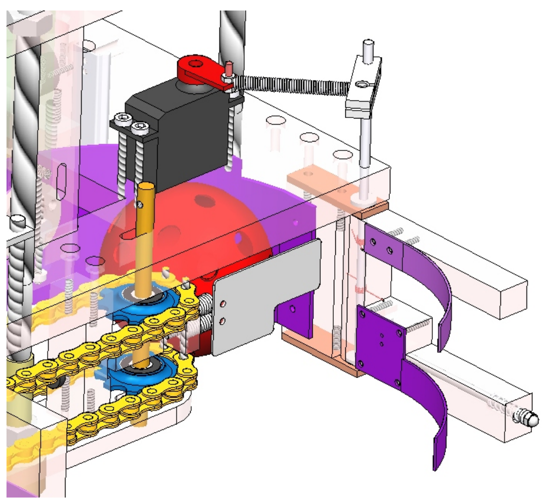
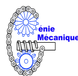


FIG. 6.17 – Bras "porte" en position ouverte

6.3.2 Bras droit

De la même manière que précédemment, voyons les contraintes qui nous sont imposées pour ce bras. switch compris, sa longueur ne peut excéder les 10 cm. L'avantage de ce dernier est qu'il pourra rester en position déployée durant tout le match, car il n'influence pas le parcours des balles. L'idée de départ fut de créer un bras qui se déploie sous l'effet de la pesanteur, en lui donnant la forme requise (voir figure 6.18, page 33). Afin de comprendre son utilisation, voyons la figure 6.19, page 33. Il est articulé autour d'une charnière qui fixe donc son axe de rotation. Une lumière circulaire usinée dans le bras est destinée à le guider lors de son mouvement de rotation. En position fermée, le bras se situe à l'intérieur du périmètre autorisé. Nous remarquons toutefois qu'il aura tendance à s'ouvrir vers l'extérieur. Sa position initiale sera maintenue par une goupille. Lors du démarrage du robot, la goupille sera tirée. Le simple effet de la pesanteur obligera le bras à prendre sa position déployée comme l'indique la figure 6.20, page 34. Notons enfin que sa forme en "L" se justifie aisément : un micro-switch y est incorporé en son extrémité et doit pouvoir détecter les bords. Mais ce bras ne doit pas s'opposer au passage de la languette dans sa rotation par rapport aux pignons. D'où l'intérêt d'affiner un



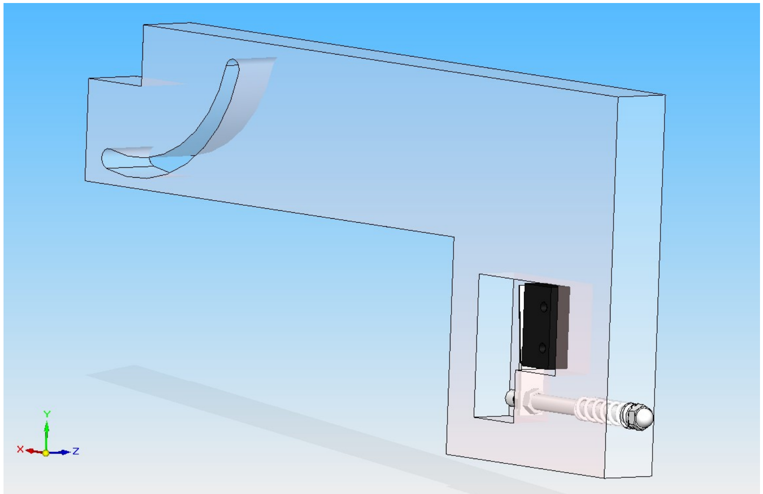


FIG. 6.18 – Forme du bras droit

maximum ce bras dans ces conditions. Enfin, une mince épaisseur de Teflon appliquée sur la tranche du bras lui permettra de glisser très facilement sur la table.

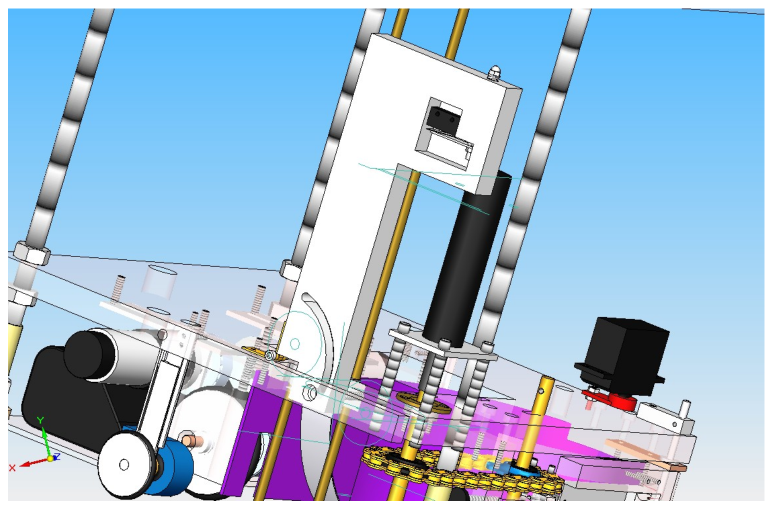
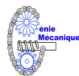


FIG. 6.19 – Description du principe du bras droit/Position fermée

6.4 Adaptations en fonction des tables du concours

6.4.1 Guidage du bras gauche par tôles cintrées

Nous avons vu précédemment que le guidage des balles par le bras gauche se fait au moyen de tôles cintrées. La rigidité de ce système posait problème. Celui-ci a été résolu en remplaçant ces tôles par de la mousse, en conservant la forme du chemin des balles. L'avantage principal réside dans le fait que les balles, poussée par la languette, s'enfoncent dans la mousse avant d'être projetées à l'intérieur du robot. Avec les tôles, les balles avaient tendance à tourner sur elles-mêmes.



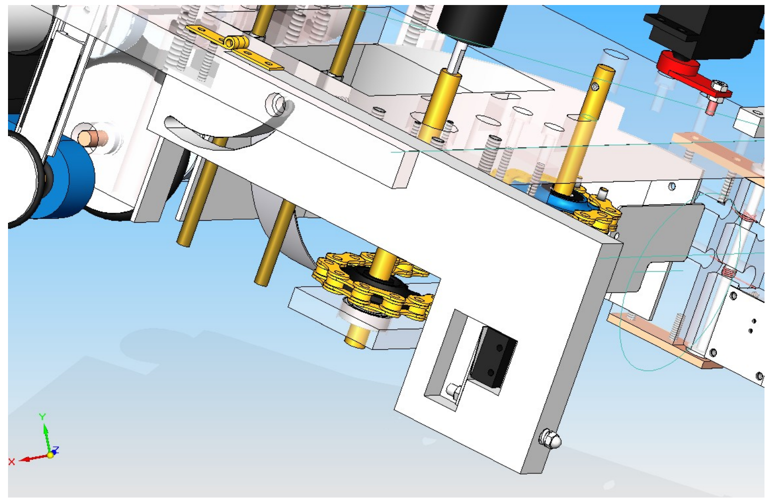


FIG. 6.20 – Description du principe du bras droit/Position déployée

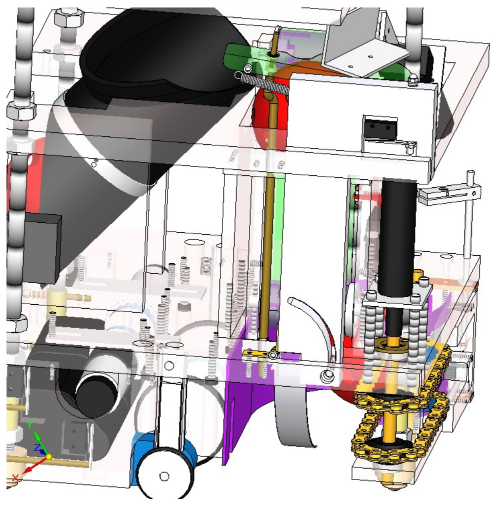
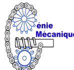


FIG. 6.21 – Bras droit non déployé

6.4.2 Languette

Les balais au bas des distributeurs verticaux ayant été remplacés par une sorte de caoutchouc visqueux, à l'état de surface relativement collant, nous devions vaincre cet effort. Il fallait donc une languette à l'état de surface plus collant que cette matière. La languette fut reconstituée par un morceau de caoutchouc, entouré d'un morceau de papier-collant double-face, afin d'obtenir une meilleure adhérence sur la languette que sur le bas du distributeur !
Le montage de cette languette est présenté à la figure 6.22.



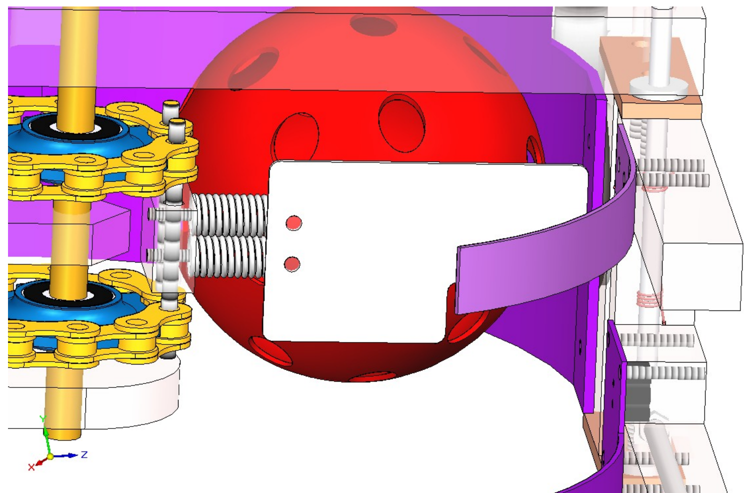


Fig. 6.22 – Languette poussant les balles à l'intérieur du robot

Chapitre 7

Ascenceur

Le but de l'ascenseur est à la fois de servir de stockage ainsi que de monter les balles soit vers la catapulte, soit vers la trémie d'éjection en fonction du système utilisé. Pour monter les balles, plusieurs solutions se sont présentées. Les caractéristiques principales à retenir devaient être :

- la continuité, pour pouvoir facilement être synchronisé avec le système d'avalement ;
- la possibilité de stockage tandis que le système est toujours en marche ;
- la hauteur à laquelle les balles doivent être amenées.

Le système retenu est basé sur l'idée de l'ascenseur du robot pour la coupe de 2004. Pour rappel, il s'agissait du système présenté sur la figure 7.1, page 36. Le système ne pourra pas

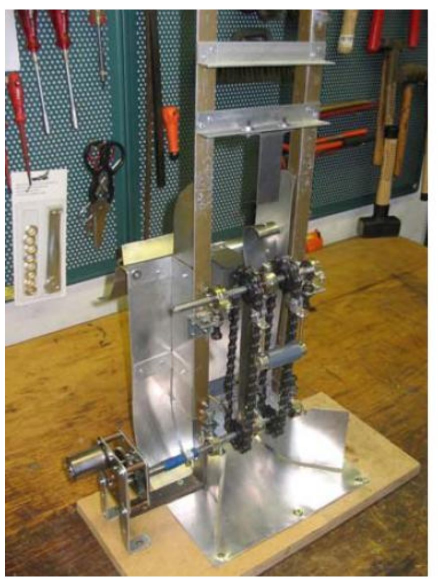
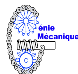


Fig. 7.1 – Ascenceur 2004

être réutilisé tel quel pour notre robot ; la différence la plus importante étant la forme et la matière des balles. Cette année, les balles sont dures et sphériques, en 2004 il s'agissait de balles de rugby en mousse. Il faut donc prévoir une structure sensiblement différente. A la figure 7.2, page 37, est présenté le système de cette année. Il s'agit d'une courroie tendue entre deux rouleaux dont les axes sont horizontaux. Le tout a été précisément dimensionné, comme développé au point suivant. Notons enfin que lorsque la première balle entre dans l'ascenceur, elle s'élève jusqu'au point le plus haut. Arrivée au dessus, elle attend d'être chargée par la



catapulte ou d'être versée dans la trémie d'éjection.
Cette dernière idée amène donc deux réflexions :

1. Une interface Ascenceur/Catapulte et Ascenceur/Trémie doit être développée ;
2. Les balles qui suivent la première doivent pouvoir glisser sur les rails, alors que la courroie entraînée par le moteur tourne en continu. Ce principe a été validé lors des séances de tests.

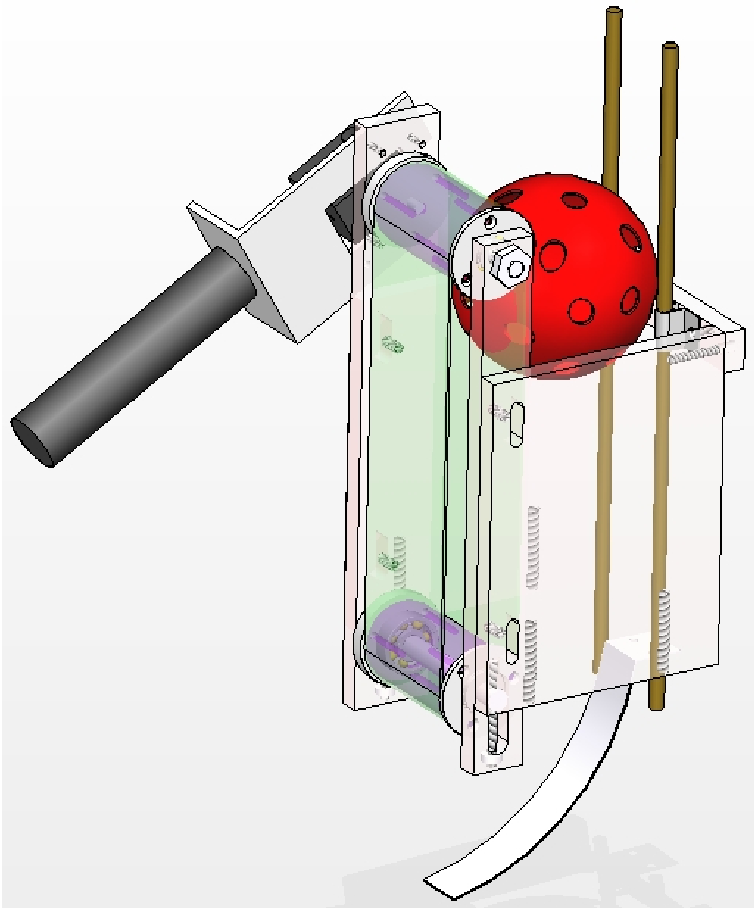
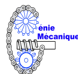


FIG. 7.2 – Ensemble ascenceur

7.1 Dimensionnement de la courroie

L'ascenseur doit prendre les balles qui arrivent, posées sur le sol, dans le chemin des balles. La courroie doit toucher tangentiellement les balles à maximum 72 mm du sol pour les prendre en charge sans les écraser. Afin de pouvoir réajuster la hauteur de la courroie avec plus de précision, l'assemblage est monté sur lumières. Ensuite, les balles doivent être montées le plus haut possible tout en étant sous la hauteur maximale autorisée. En effet, d'après le règlement (qui a dû être précisé sur le Forum par un arbitre), il est ressorti que les balles stockées dans le robot ne pouvaient pas sortir des dimensions autorisées. L'ensemble robot et balles doit rester sous les 35 cm, hauteur extrême autorisée. Pour estimer la hauteur maximale du système ascenseur, plusieurs paramètres entrent en compte. On peut considérer que le diamètre de la balle quitte le système un peu au-dessus du diamètre du rouleau haut du système. Afin de conserver une



certaine marge de sécurité avant de ne pas risquer qu'une balle ne dépasse du robot, nous dirons en première approximation que la balle quitte le système tangentiellement au rouleau du haut. On se libère alors du paramètre "diamètre des rouleaux" pour déterminer la géométrie générale de l'ascenseur. La hauteur maximale du système est donc de :

$$H = 350\ mm - 2 \times 72\ mm = 206\ mm$$

La longueur de la courroie dépend maintenant du diamètre des rouleaux. Précisons directement que les rouleaux sont de diamètres identiques. Cela permet de simplifier le système qui ne nécessite pas ce genre d'optimisation.

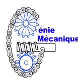
$$L = (H - 2 \times R) \times 2 + 2 \times \pi \times R \tag{7.1}$$

Avec :
- L, la longueur de la courroie ;
- H, la hauteur totale de l'ascenseur ;
- R, le rayon des rouleaux.

Pour des raisons pratiques évidentes, et surtout pour ne pas devoir patienter le temps de la livraison, nous avons tenté de récupérer une des courroies que nous avions à disposition, notamment une courroie de 435 mm.
Dans ce cas, par (7.1), le rayon minimum pour les rouleaux devrait être de 10 mm.
Finalement le rayon que nous avons choisi est de 15 mm, pour différentes raisons :
- la courroie de stock est utilisable ;
- nous garantissons une marge de sécurité vis-à-vis de la hauteur maximale autorisée, sachant que d'autres systèmes devront encore être mis en place ;
- la hauteur de l'ascenseur est donc de 200 mm ;
- cela permet de stocker deux à trois balles dans l'ascenseur.

7.2 Rouleaux

Afin de pouvoir placer la courroie ainsi que de gérer sa tension, le rouleau du bas est placé dans des lumières comme présenté à la figure 7.3, page 39. Le perçage dans l'axe du rouleau est fileté, ce qui permet de régler la hauteur du rouleau. En effet, la courroie a tendance à maintenir le rouleau haut, tandis qu'en vissant les deux vis dans les lumières, le rouleau va être forcé de descendre, les vis étant fixes dans les lumières. Ce système permet de régler assez finement la tension dans la courroie, et sans grande difficulté. Ensuite, afin de permettre aux balles d'être maintenues, nous avons placé deux barres pour les guider de manière optimale. Grâce à ce système, il suffit de venir placer la balle vers les rails, sans grande précision, et elle viendra directement se placer correctement. Dans le but de gérer la force avec laquelle les rails vont presser les balles contre la courroie, il sera possible de venir interposer des entretoises. Sur la figure 7.4, page 39, on peut voir la position des éventuelles entretoises, précisée par les flèches bleues. L'élaboration des rouleaux a demandé une étude détaillée qui a abouti sur le tracé de plans. Grâce à ces plans, nous avons pu commander les différentes pièces à l'atelier central. Une vue éclatée de la totalité du rouleau moteur est présentée à la figure 7.5, page 39. Pour bien comprendre la vue, il faut imaginer que l'axe du renvoi d'angle vers le moteur pénètre dans le flasque et puis dans le rouleau du côté droit sur la vue (voir figure 7.6, page 40). L'axe, sur lequel on a préalablement usiné un méplat, est maintenu à l'aide d'une vis de pression présente dans le flasque. Le flasque est ensuite rendu solidaire du rouleau à l'aide de trois vis à tête fraisée. De



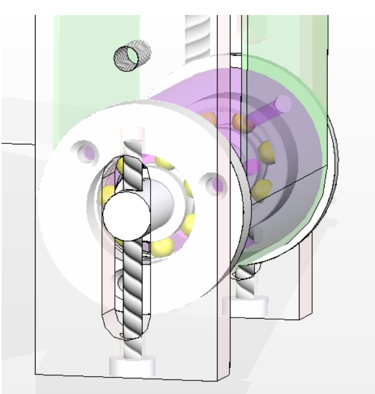


FIG. 7.3 – Réglage de la tension dans la courroie à l'aide de vis qui fixent la hauteur de l'axe du rouleau bas dans ses lumières

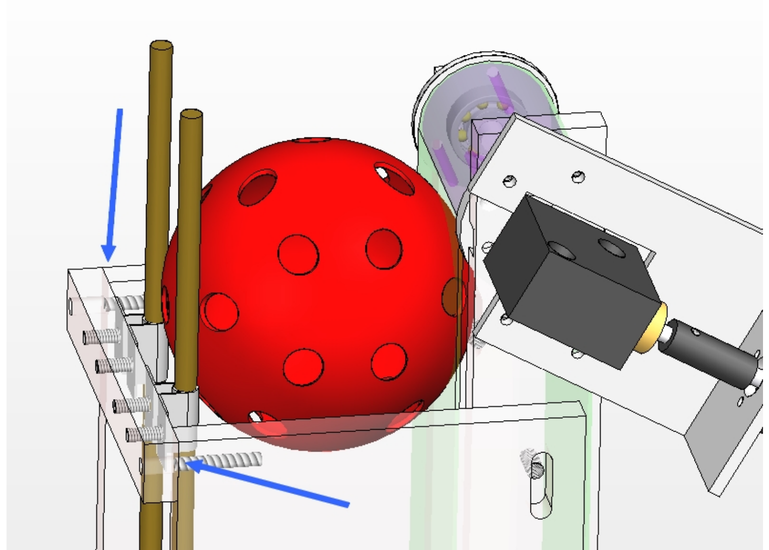


FIG. 7.4 – Tiges "flipper" qui coincent les balles contre la courroie

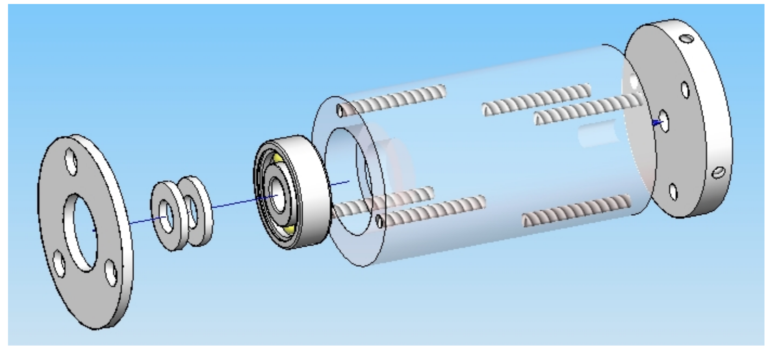
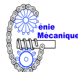


FIG. 7.5 – Vue éclatée du rouleau côté moteur

l'autre côté, le rouleau a été taillé de façon à pouvoir recevoir un roulement en son sein. Vu que c'est le rouleau qui tourne, il a été monté serré sur sa bague extérieure. Le flasque de ce côté maintien le roulement en bloquant axialement sa bague extérieure. Le flasque est fixé comme le premier et accompagne le rouleau dans sa rotation. Afin de maintenir un écartement fixe, des



rondelles sont présentes entre le morceau de plexiglas qui maintien l'axe et la bague intérieure du roulement. L'ensemble est présenté à la figure 7.6, page 40. Ensuite, pour le rouleau du bas,

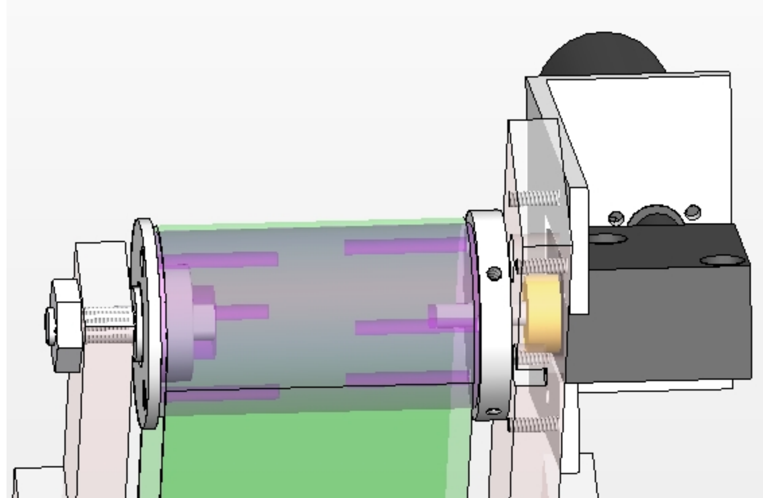


FIG. 7.6 – L'ensemble rouleau moteur dans l'assemblage

la structure est similaire (cfr figure 7.7, page 40). Le montage est tout à fait ressemblant au rouleau moteur, du côté de son roulement. Vu que ce rouleau suit le mouvement de la courroie imposé par le rouleau moteur, il y a un roulement de chaque côté.

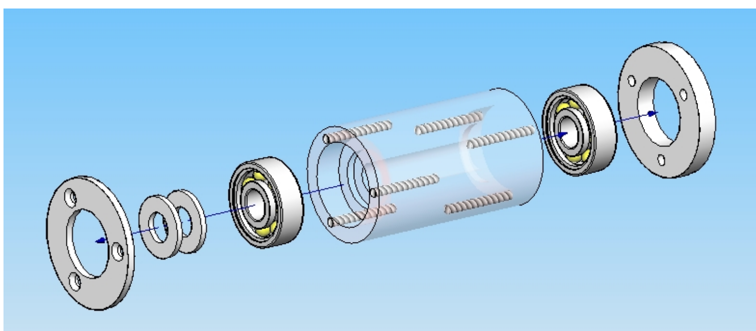
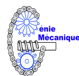


FIG. 7.7 – Vue éclatée rouleau du bas

7.3 Amorce de la montée des balles

Afin d'amorcer la montée des balles, une languette d'amorce de montée est indispensable. En effet, les balles sont poussées dans le chemin des balles par la languette de préhension des balles de l'avalement. Cette poussée les force à passer sous le rouleau bas de l'ascenseur. La balle est alors collée à la languette d'amorce de montée selon un rayon bien défini. Grâce à la précision du système, la courroie entraîne la balle vers la montée dans son mouvement. La languette d'amorce de montée a dû être réalisée à la cintreuse, et en acier, afin de conserver sa forme, quels que soient les efforts qu'on lui soumette. Comme décrit à la figure 7.9, page 41, la balle de 72 mm de diamètre ne peut plus être laissée libre à partir du moment où elle touche l'ensemble ascenseur. Sans cette précision, la balle resterait à l'entrée du système, en tournant sur elle-même. En considérant que le rouleau du bas fait 30 mm de diamètre, en ajoutant à cela 1 mm



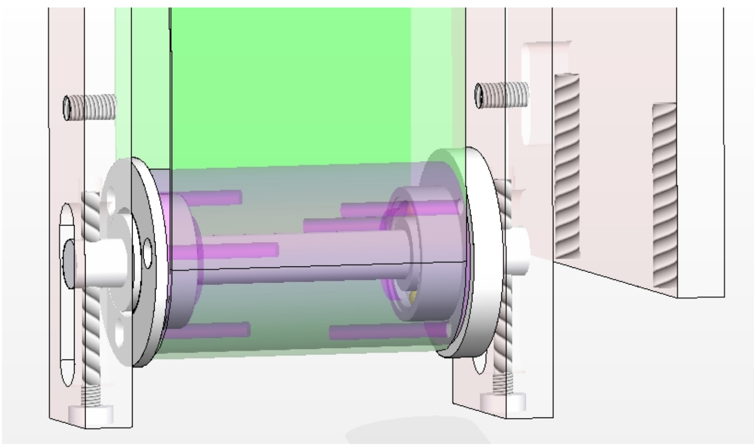


FIG. 7.8 – Rouleau du bas dans l'ensemble ascenseur

d'épaisseur de courroie, on obtient un diamètre de 32 mm qui vient toucher tangentiellement la balle. En laissant juste l'espace pour que la balle puisse passer, nous fixons le rayon de la languette d'amorce par : $\frac{32}{2} + 72 = 88mm$.
Précisons aussi que l'ensemble rouleaux + courroie est placé dans des lumières afin de pouvoir

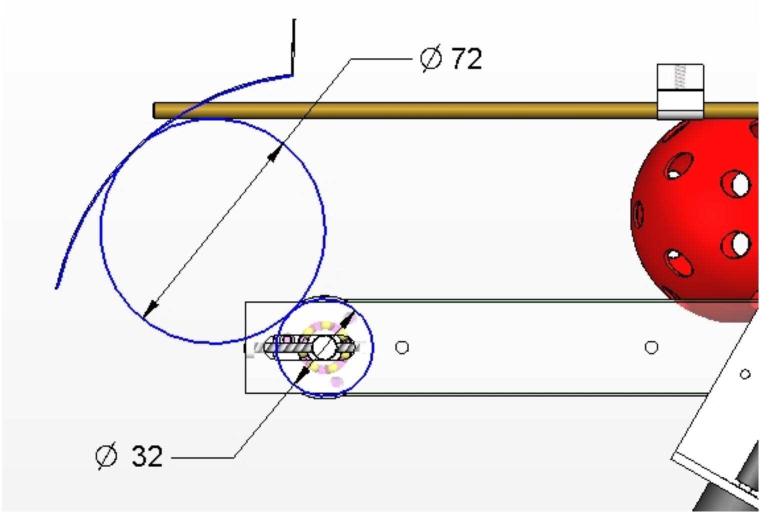
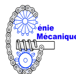


FIG. 7.9 – Dimensionnement de la languette d'amorce de montée des balles

régler au mieux la position du rouleau du bas par rapport au sol. Ce réglage permet d'optimiser la captation des balles par l'ascenseur.

7.4 Positionnement

Afin de positionner le module dans le robot, la plaque de base a dû être découpée. La découpe a une forme rectangulaire, et est entourée de quatre lumières. Leur but est de pouvoir jouer sur la position en translation du module ascenseur afin de le positionner au mieux à la fois par rapport au chemin des balles et par rapport à la position de la catapulte ou de la trémie d'éjection.



7.5 Motorisation

Enfin, le moteur qui anime le système est un Maxon Brushless (référence 200 863), avec un réducteur 84 :1 (référence 143 984).
Les caractéristiques du motoréducteur sont les suivantes :
- 24 V
- 20 W
- 200 à 250 $\frac{tr}{min}$
- 1,2 $N \times m$

Le moteur a été choisi de telle façon que la montée des balles soit plus rapide que l'avalement, dans le but d'éviter tout risque "d'embouteillage" à l'entrée de l'ascenseur. Le moteur a nécessité un renvoi d'angle afin de minimiser l'encombrement du module, les moteur Brushless de chez Maxon étant assez fins et allongés. Le moteur est fixé par trois vis sur une pièce d'assemblage assez originale, comme présentée à la figure 7.10, page 42, qui est elle-même fixée à la structure du module. Les datasheets relatives aux moteurs et réducteurs Maxon sont disponibles en

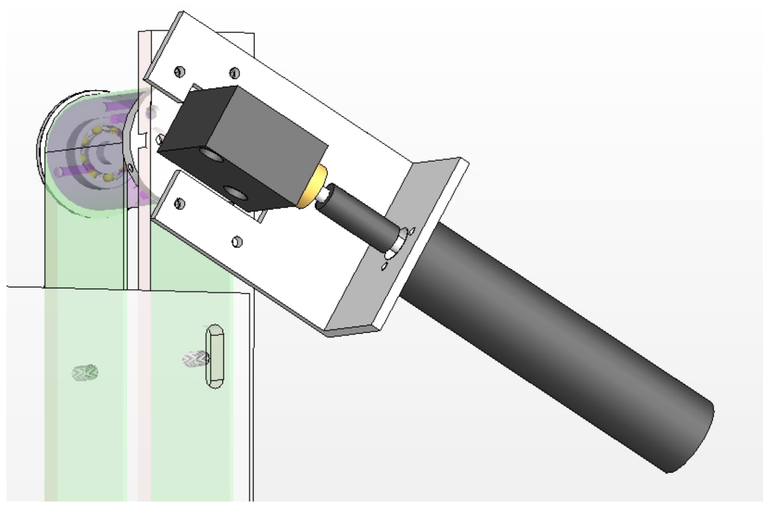


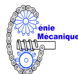
Fig. 7.10 – Moteur, renvoi d'angle et fixation

annexe F.

7.6 Interface Ascenceur/Catapulte et Ascenceur/Trémie

Afin d'éviter tout *bourrage* lors du chargement de la catapulte, une pièce supplémentaire d'interface a dû être conçue. La pièce d'interface aurait plusieurs fonctions :
- Laisser passer la première balle et lui faire occuper la place de *balle-tampon*, tant que la catapulte ne l'a pas chargée ;
- Retenir les autres balles et les stocker dans l'ascenceur, qui tourne en continu.

Dans le cas de la trémie, elle laisserait les balles passer une à une, jusqu'à atteindre la limite de stockage de la trémie (3 balles). Nous comprenons également toute l'importance de cette pièce qui permet à la **dernière balle** de passer de l'ascenceur vers le système d'éjection. En effet, la dernière balle, arrivant au sommet de l'ascenceur, n'est plus poussée par la suivante. Elle resterait bloquée en tournant sur elle-même. La pièce d'interface, montée sur ressorts, permet donc ce passage de manière *naturelle*. Résumons donc pour la trémie :



- Faire passer les balles une à une de l'ascenceur vers la trémie, éviter que plusieurs balles n'entrent **simultanément** dans le système d'éjection ;
- Propulser naturellement la dernière balle, qui n'est pas poussée par une suivante.

Après maints et maints essais, la pièce finalement obtenue est celle présentée à la figure 7.11, page 43. La pièce est percée (2) d'un trou dans lequel est fixé un roulement. Ce roulement est

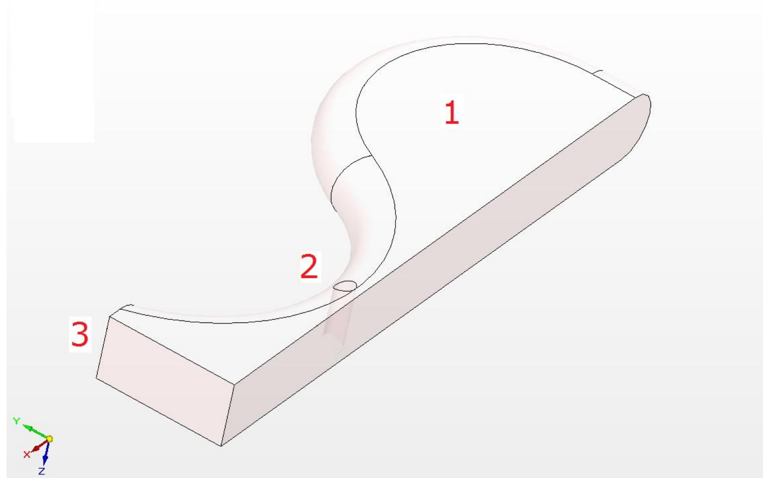
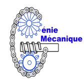


Fig. 7.11 – Interface Ascenceur/Catapulte et Ascenceur/Trémie

quant à lui en rotation autour d'un axe. En position de repos, la partie (1) de la pièce se trouve au-dessus de l'ascenceur. Cette position est maintenue par des ressorts. Imaginons maintenant la première balle qui se présente dans l'ascenceur : elle est montée par la courroie. Au contact de la pièce d'interface, la partie (1) est poussée dans le sens horlogique et la porte s'ouvre. La balle continue son chemin, la pièce tournant alors dans le sens anti-horlogique, et se retrouve par conséquent dans la position (2), dite de *balle-tampon*. Elle y reste tant que la catapulte ne s'est pas armée pour la laisser entrer. Le fait qu'elle reste alors à cet endroit bloque en rotation la pièce d'interface. Les autres balles arrivant alors dans l'ascenceur, qui tourne en continu, glissent donc sur les rails et attendent que la balle tampon soit chargée. Lorsque la catapulte s'arme, la première balle est accueillie dans le godet, en (3), elle quitte la position (2) et ouvre donc la porte à la balle suivante ! Le procédé se répétera autant de fois que nécessaire. Les ressorts devront être suffisamment raides que pour obliger la pièce à revenir aussi vite que possible dans sa position de repos, mais pas trop de manière à permettre à la dernière balle, qui elle n'est plus poussée par les autres, de pouvoir vaincre l'effort. Enfin, notons que l'avantage principal de ce système est qu'il ne nécessite pas de commande extérieure !
A la figure 7.12, la description du processus est explicitée par une image qui vaut mieux qu'un long discours. Pour la configuration Ascenceur/Trémie, la figure 7.13 nous éclaire. Dans la configuration Ascenceur/Catapulte, la figure 7.14 décrit le fonctionnement du processus.



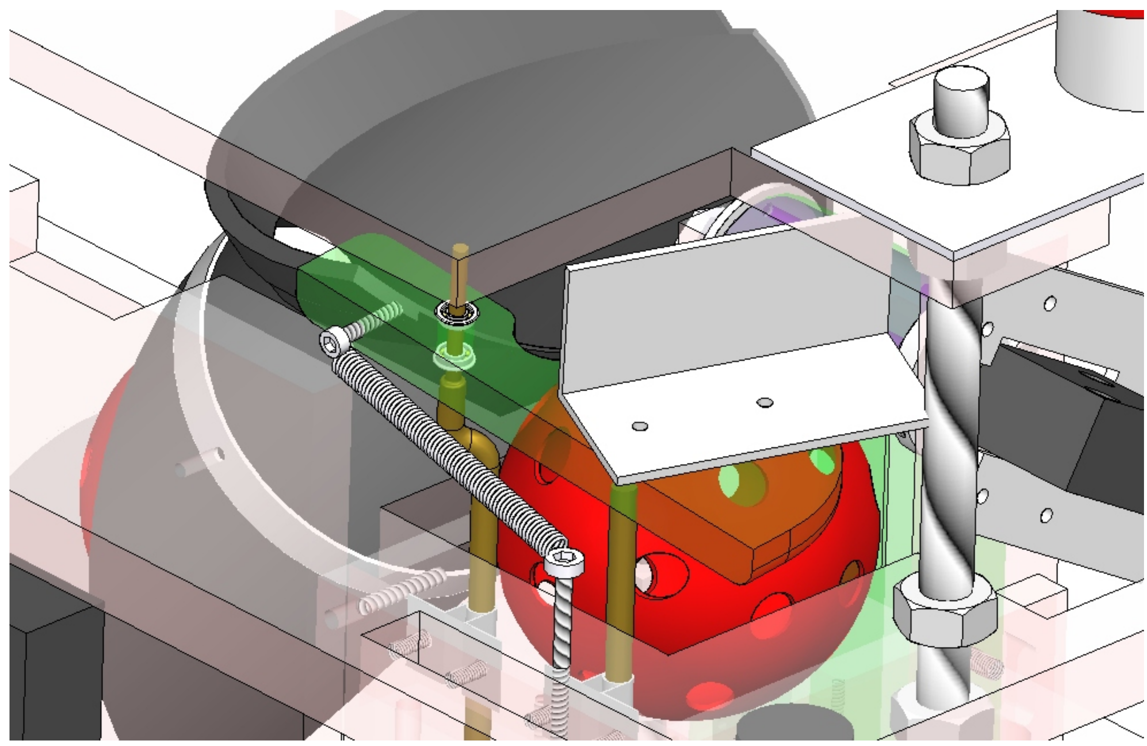


FIG. 7.12 – Pièce d'interface, couplée avec la trémie d'éjection

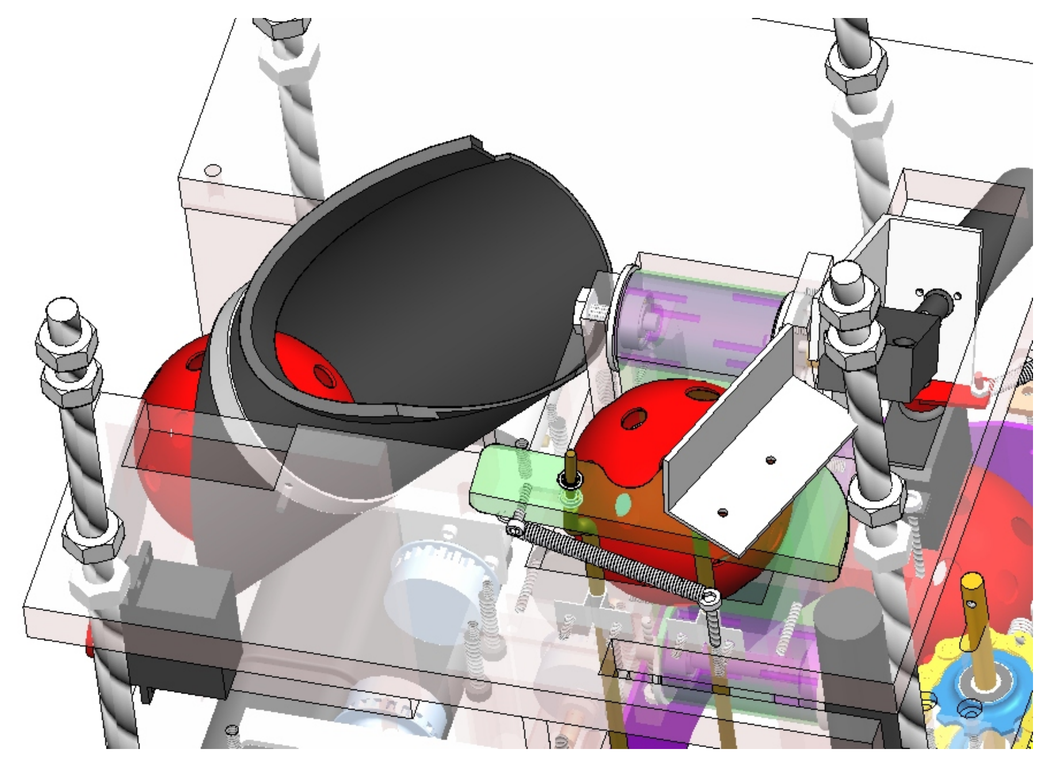
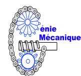


FIG. 7.13 – Pièce d'interface Ascenceur/Trémie



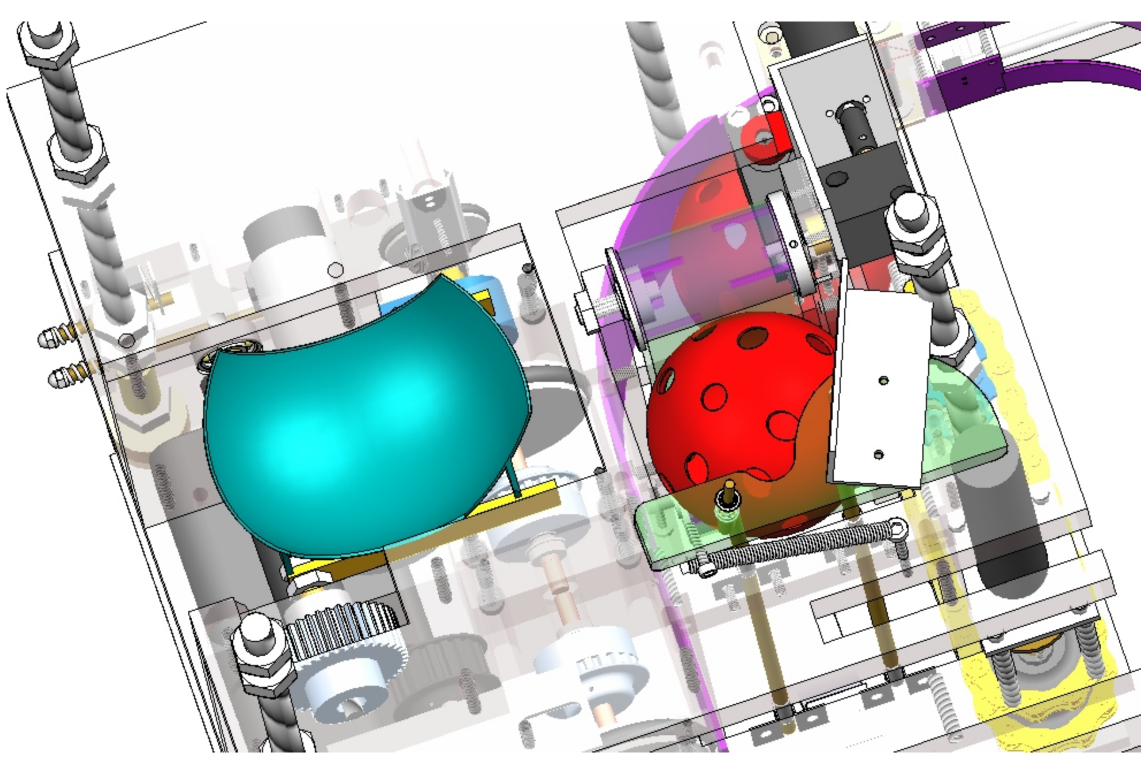


FIG. 7.14 – Pièce d'interface Ascenceur/Catapulte

Chapitre 8

Éjection des balles

Selon la stratégie établie au départ du concours, nous devons être capable d'éjecter les balles du robot vers le conteneur réfrigéré adéquat (goal bleu ou rouge). Dans un premier temps, et ce tout au long de l'année, nous avions planifié de catapulter ces balles. Mais la perte d'un moteur peu de temps avant le concours nous a obligé à passer à la deuxième solution : la trémie d'éjection.

8.1 Catapulte

Le système de catapultage des balles a été réalisé après mûres réflexions quant aux différentes solutions plausibles.
Plusieurs possibilités ont été envisagées, à savoir une catapulte à air comprimé, un système à came ou encore une catapulte à engrenages. Après une rapide analyse des différentes possibilités, nous verrons que cette dernière solution sera adoptée, car ayant déjà fait ses preuves auparavant. Le système de catapulte à engrenages avait en effet été validé par l'équipe de nos prédécesseurs en 2004.

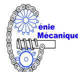
8.1.1 Comparaison des différentes possibilités

Catapulte à air comprimé

Citons brièvement les avantages et inconvénients d'un tel système.
Avantages :
- Puissance de lancement des balles ;
- Facilité de conception (un axe de rotation avec bras de levier, un piston, un godet, un réservoir d'air comprimé, une butée, un système de commande pneumatique).

Inconvénients :
- Encombrement important du mécanisme complet ;
- Réservoir d'air comprimé nécessite un espace supplémentaire dans le robot ;
- Nécessité de commander le robot par la commande pneumatique ;
- Obligation d'avoir toujours un compresseur à disposition.

Toutes ces considérations, et particulièrement celles concernant l'encombrement sont autant d'éléments qui nous feront choisir une autre solution.



Système à came

Ce système serait constitué d'un bâti, d'une plaque (la catapulte en elle-même), d'une came entraînée par un moteur à courant continu, d'une butée et de ressorts. Au repos, la catapulte serait en position "non armée" et maintenue par des ressorts. Le mouvement en continu de la came armerait la catapulte.
Une telle solution n'offre cependant pas assez de précision en terme de distance et de direction du tir. Ce système ne sera donc pas adopté, même s'il offre un encombrement réduit par rapport à la catapulte à air comprimé. Enfin, nous verrons que la catapulte à engrenages offre des performances bien meilleures face à ces deux premières propositions.

Catapulte à engrenages

Le principe sur lequel se base la conception de la catapulte est très simple, puisqu'il met en oeuvre un système d'engrenages et un ressort de torsion.
Un moteur brushless tourne en continu afin d'entraîner un pignon dont une partie des dents ont été limées. Ce pignon moteur engrène quant à lui sur une roue dentée, fixée sur l'axe de rotation de la catapulte. Un ressort de torsion solidaire de cet axe contraint la catapulte à rester à sa position de repos, c-à-d la position "non armée". Lorsque les dents du pignon moteur engrènent sur la roue dentée de la catapulte, celle-ci s'arme jusqu'à ce que les dents disparaissent. A ce moment, la catapulte est en position armée. Le ressort de torsion contraint alors la catapulte à revenir à sa position de repos et la balle contenue dans le godet peut être éjectée. Une butée en rotation sera prévue afin de limiter la course de la catapulte et de donner un angle de catapultage adéquat.
Le nombre de dents limées correspond à une fourchette d'angles suffisamment précise pour le chargement de la catapulte. En plus du ressort de torsion, un ressort de rappel en traction a été ajouté afin d'éliminer les rebonds sur la butée et d'éviter que la catapulte ne charge à une position autre que celle du repos. Enfin, le rapport entre les roues dentées a influencé le choix du moteur dans les catalogues MAXON. Le nombre de tirs/minute pouvait alors être déduits.
Un prototype a été créé afin de valider le principe. Les essais s'avérèrent concluant. Nous avons

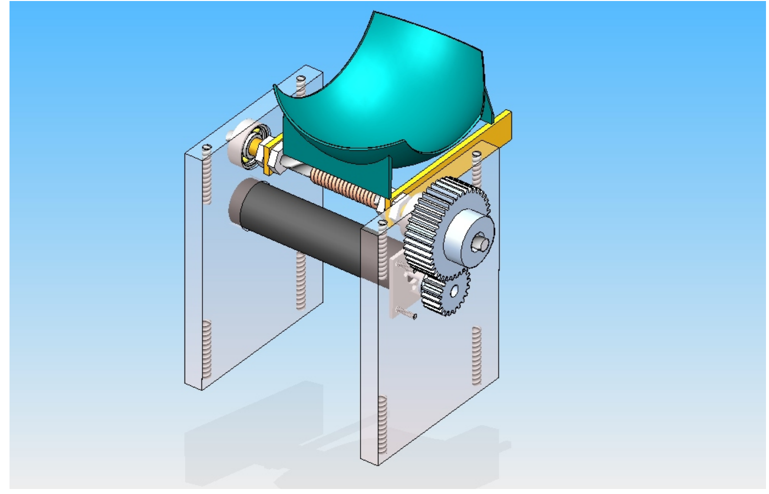
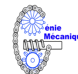


FIG. 8.1 – Vue d'ensemble de la catapulte, avec godet

pu remarquer que la distance de tir était correcte (possibilité de tirer à plus d'un mètre) et que la trajectoire des balles était parabolique. De plus, un simple réglage de la position de la butée



permet de régler la puissance de tir et la trajectoire de la balle.
Enfin, le godet a été réalisé afin d'épouser au mieux la forme de la balle, augmentant de cette façon la précision de tir. De plus, une tôle cintrée a été ajoutée au dos de la catapulte, pour que les balles sortant de l'ascenseur ne tombent dans le robot avant d'être chargées. Le rayon de cette tôle cintrée a donc été tout simplement déduit à partir de l'axe de rotation de la catapulte. Une vue de cette catapulte montée sur le robot est présentée à la figure 8.2, page 48. Nous distinguons la tôle en question, entourée d'un cercle rouge.

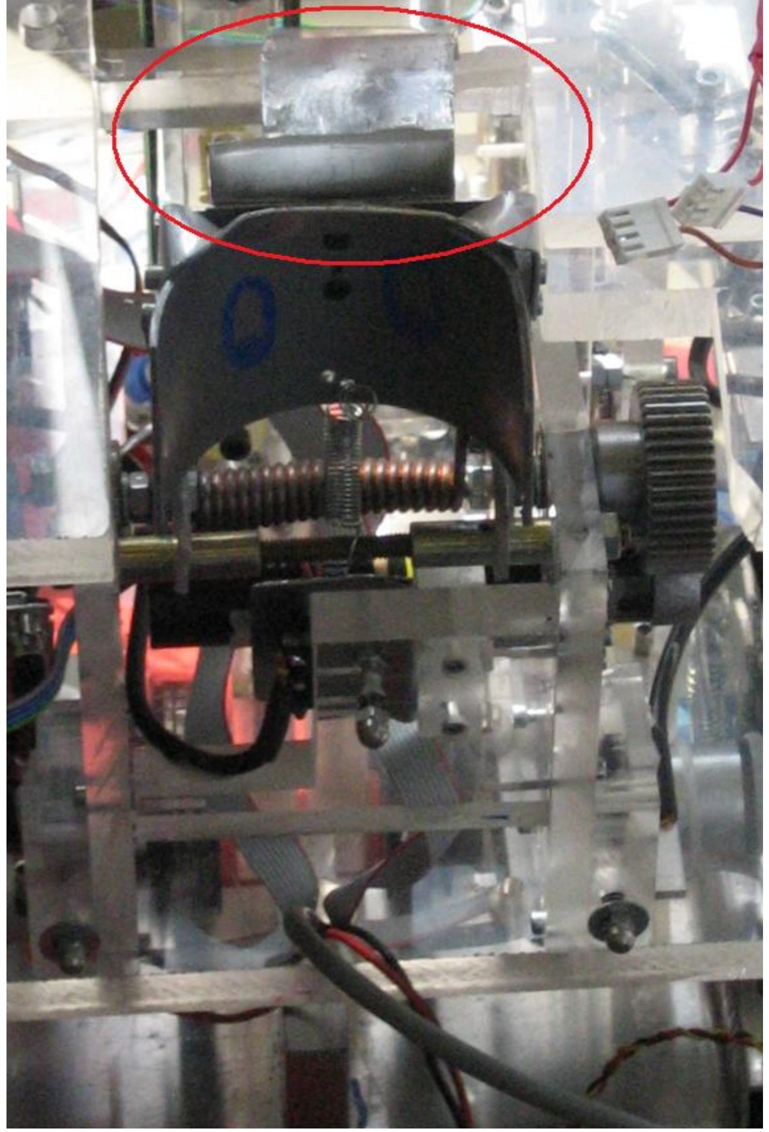
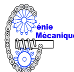


Fig. 8.2 – Vue d'ensemble de la catapulte avec tôle cintrée

8.1.2 Motorisation

Le moteur est un Maxon Brushless (référence 200 863), avec un réducteur 370 :1 (référence 143 996).
Les caractéristiques du motoréducteur sont les suivantes :
- 24 V



- 20 W
- 45 à 55 $\frac{tr}{min}$
- 1,2 N $\times m$

Les datasheets relatives aux moteurs et réducteurs Maxon sont disponibles en annexe F.

8.2 Trémie d'éjection

Une fois les balles arrivées en hauteur, plusieurs possibilités s'offrent à nous. On peut soit les lancer vers le goal en passant par une catapulte (décrite précédemment), soit les lâcher dans la rigole à l'avant de la table. La trémie d'éjection est le module qui permet de poser les balles dans la rigole.
Le principe est simple : les balles sont stockées et libérées lorsque le robot se place face au conteneur standard. Le système est visible à la figure 8.3, page 49.

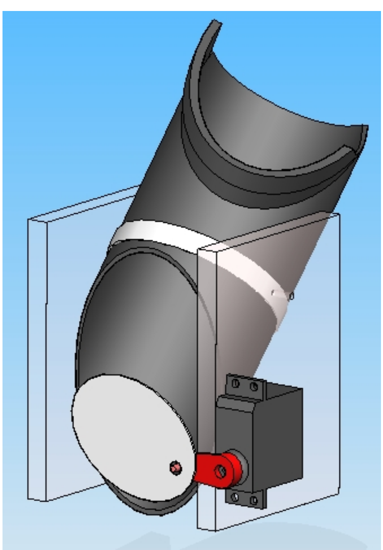


Fig. 8.3 – Module trémie d'éjection

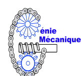
8.2.1 Dimensionnement de l'espace disponible

La place disponible pour la trémie d'éjection a été estimée sur base de l'encombrement de la catapulte. En effet, la trémie remplace le module catapulte, et doit donc avoir des dimensions similaires afin de s'imbriquer au mieux dans le système. La place disponible est représentée par le parallélépipède rosé à la figure 8.4, page 50. Ses dimensions sont de 110 $\times$ 140 $\times$ 180 mm.

8.2.2 Tuyau

Pour mettre en place ce système très simple, nous avons utilisé un morceau de tuyau en PVC de 80 mm de diamètre comme pièce de base. Plusieurs raisons à cela :

- Guider les balles dans toutes les directions. Même si elles s'accumulent, elles resteront dans le tube. En effet, un simple "toboggan" ouvert aurait permis aux balles de s'accumuler et de se placer de manière chaotique, risquant alors un blocage lors de leur libération, voire même de sortir du robot de manière inopportune ;



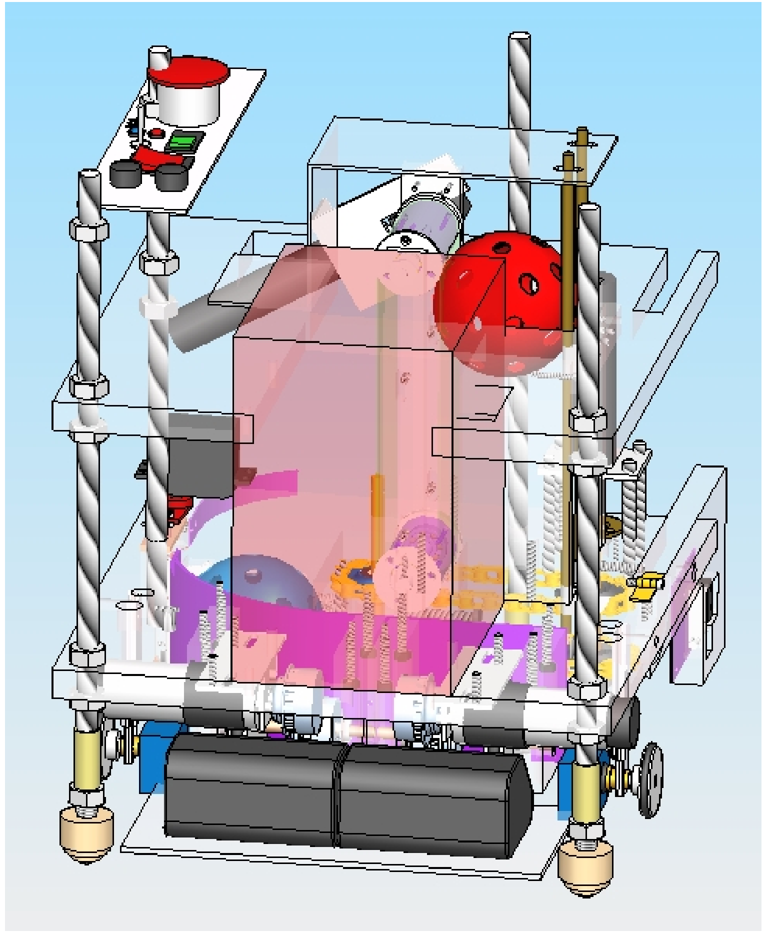


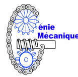
FIG. 8.4 – Place disponible pour placer la trémie dans le robot (zone rosée)

- Les balles sont stockées dans la trémie, car la "porte" de sortie est en bas, à l'extrême fin du tube. La porte est détaillée et argumentée plus loin ;
- Permettre aux balles de tomber le plus facilement possible en réduisant les frottements. Le diamètre du tube a été choisi suffisamment proche du diamètre des balles, tout en restant lâche en suffisance ;
- Amener les balles au plus près de la rigole sans toucher à la plate-forme de base, déjà fortement fragilisées par les fixations de tous les autres systèmes qui s'y imbriquent ;
- Pouvoir réutiliser le système de fixation de la catapulte, pour la même raison : éviter de fragiliser le système par de nouveaux perçages.

Au vu de la place disponible, le morceau de tuyau a comme dimension générale de 200 mm, duquel nous avons retiré différents parties afin d'obtenir le profil désiré. Dans le but de récolter au mieux les balles à la sortie de l'ascenseur, le début du tuyau a été transformé en entonnoir par déformation après traitement thermique.

8.2.3 Fixation

Pour fixer le tuyau en PVC aux parois de soutien, un simple collier de plomberie a été utilisé. Le tuyau est alors suffisamment fixé, sa forme le positionnant de manière unique. Les soutiens du tuyau sont des morceaux rectangulaires en plexiglas. Des trous filetés sont présents dans l'épaisseur, sur la face du bas, afin de fixer l'ensemble sur la plate-forme de base en réutilisant



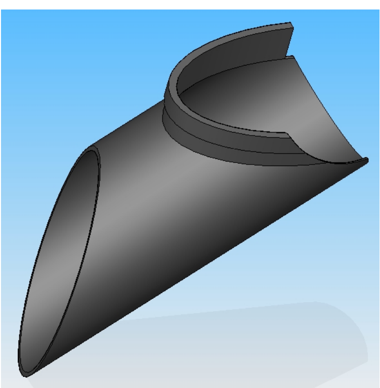


FIG. 8.5 – Tuyau de la trémie d'éjection

les mêmes perçages que pour la catapulte.

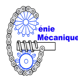
8.2.4 Porte de sortie

Différentes solutions techniques étaient possibles afin de libérer les balles. Par exemple :

1. Pour le positionnement de la porte : en haut du tuyau, pour que les balles restent uniquement stockées dans l'ascenseur. Cette solution présente l'avantage de ne pas devoir supporter le poids des balles. Elle n'a pas été retenue pour deux raisons majeures :
   (a) Le poids des balles n'est pas important, elles sont très légères et ne risquent pas d'endommager le système ;
   (b) Le stockage des 5 balles autorisées n'aurait été possible que dans l'ascenseur et le chemin des balles (au sol). Or, la dernière balle se serait trouvée juste à l'entrée du robot. Lors du déplacement de ce dernier vers la rigole, la dernière balle aurait pu ressortir du chemin des balles, ce qui nous aurait immédiatement privés de deux précieux points.

   Il est donc impératif que les balles soient stockées plus à l'intérieur du robot, idéalement dans l'ascenseur et dans la trémie. La position de la porte est donc préférable en bas.
2. Le choix de la motorisation : immédiatement notre choix s'est porté vers un servomoteur, technologie aisément maîtrisée par l'électronique et l'informatique, nécessitant très peu de place et très simplement fixé. Une motorisation était obligatoire car le système ne sait pas facilement être couplé avec un autre module.
3. Pour la géométrie de la porte : la porte la plus légère possible a été l'objectif. En effet, plus elle sera légère et plus elle sera facilement manoeuvrée par le servomoteur qui l'actionne. La porte la plus légère sera la porte la plus petite possible, ce qui ne pose pas de problème car le diamètre du tube est un peu supérieur au diamètre des balles.

La solution finale est présentée à la figure 8.6, page 52. Très simplement, un servomoteur à droite de la sortie ouvre la porte dès que le robot s'est correctement positionné face à la rigole (ou en fin de match, quelle que soit la position du robot, afin de tenter le tout pour le tout). Dès l'actionnement du servomoteur, la porte se lève dans une rotation dans le sens horlogique (l'utilisateur doit être face vers la sortie), ce qui libère instantanément les balles.



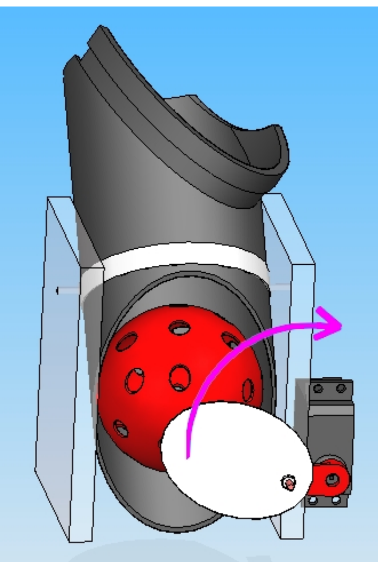


FIG. 8.6 – Porte de la trémie, sens d'ouverture schématisé par la flèche

8.2.5 Amélioration et fiabilisation

Lors de différents essais, différents problèmes ont été constatés :

1. Les balles sortent trop vite de la trémie, rebondissent dans la rigole, et parfois en ressortent. Pour résoudre ce problème, la solution a été de ralentir les balles à la sortie, en créant des pertes de charges. Pratiquement, en plaçant un morceau de caoutchouc très souple à la sortie du tuyau, et en lui donnant une forme optimale après différents tests, nous avons efficacement ralenti les balles sans risquer de les bloquer.
2. Une balle restait parfois bloquée en haut du tuyau, à l'entrée de l'entonnoir. Cela était dû à la forme de l'entonnoir, un peu trop grande à la première conception. En effet, deux balles pouvaient se positionner de front dans le tuyau, ce qui bloquait parfois les deux dernières. La forme de l'entonnoir a été réadaptée, en réduisant un peu son amplitude d'ouverture à l'entrée afin que le problème ne se reproduise plus.

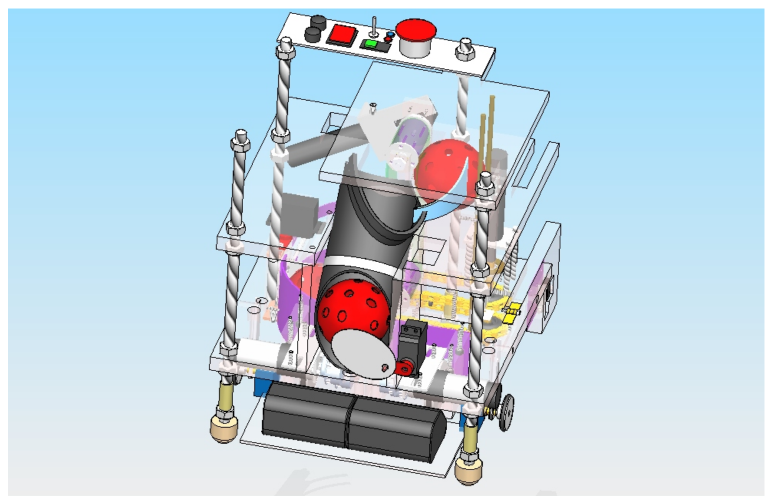


FIG. 8.7 – Vue du module trémie inséré dans le robot

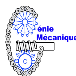
Chapitre 9

Conclusions de la partie Mécanique

En guise de conclusion à cette Partie Mécanique, nous pouvons affirmer que le robot a été conçu correctement, nous en voulons pour preuve sa belle prestation à la Coupe de Belgique, où aucune défaillance mécanique importante n'a été à déplorer.

Nous avons également pu constater qu'il est indispensable de mener en parallèlle **le développement de prototypes à valider** avec **la représentation du robot sous Solid Edge**. En effet, celle-ci est plus que nécessaire afin de toujours avoir une idée de l'encombrement de chaque sous-système constituant le robot. De plus, des plans ont été créés sur base de cette modélisation, afin de commander suffisamment tôt les pièces nécessaires à l'atelier.
La **modularité** de notre robot fut un élément payant. En effet, cela nous a permis de développer des systèmes en parallèlle, de les valider séparément. Cette modularité nous a permis également d'intervenir rapidement lors des tests sur table. Chaque sous-système du robot pouvait être réparé ou modifié indépendamment.

Ensuite, lorsque la propulsion du robot a été conçue, il a été important de dédoubler celle-ci afin de pouvoir travailler en parallèlle avec le pôle informatique. La programmation d'une trajectoire, aussi simple puisse-t-elle sembler, n'est pas chose aisée.
Dès le départ, nous avons pris des marges de sécurité importantes en ce qui concerne les dimensions du robot, spécifiées par le règlement. Il aurait en effet été très dommageable de présenter à l'homologation un robot surdimensionné.
Des vues d'ensemble du robot sont présentées aux figures 9.1, 9.2 et **??**.



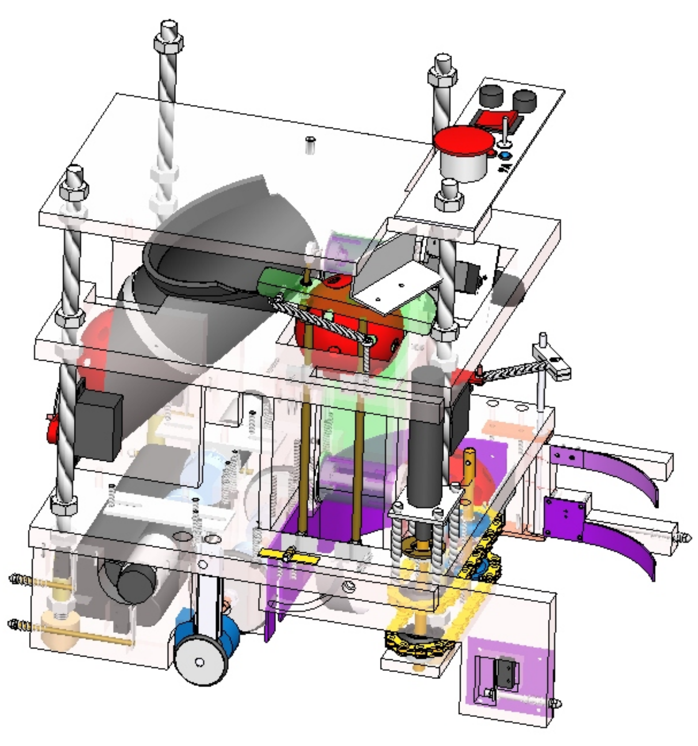


FIG. 9.1 – Vue de côté du robot

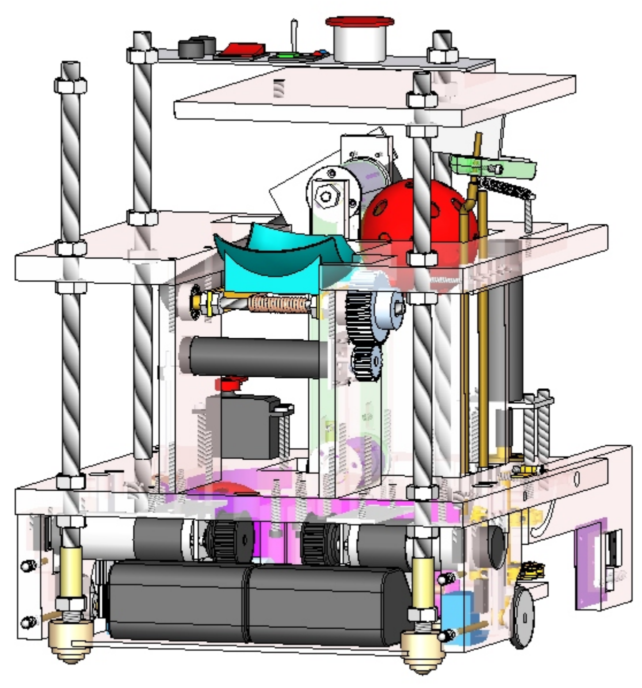
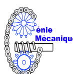


FIG. 9.2 – Vue d'ensemble du robot, face avant



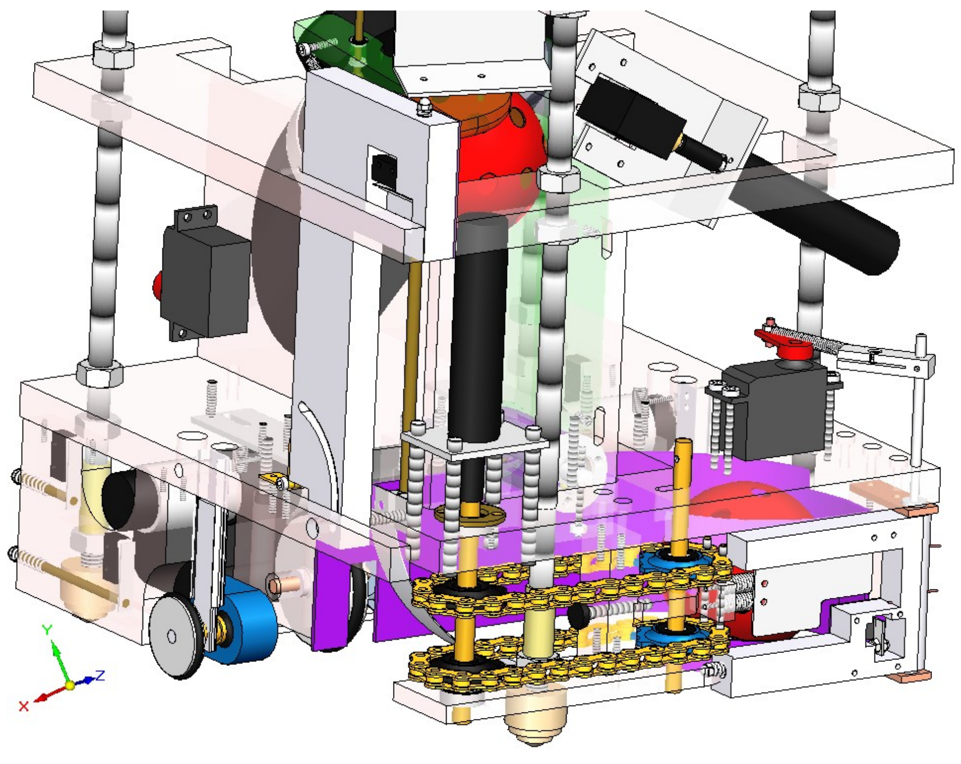


FIG. 9.3 – Vue d'ensemble du robot, bras en position fermée

Deuxième partie

Électronique

Chapitre 10

Introduction

L'électronique constitue un des trois piliers de la science que nous appelons *Robotique*. Elle complète parfaitement la Mécanique et l'Informatique. Elle s'intéresse aux circuits électriques, aux microcontroleurs, aux composants les plus variés tels que les quartzs, les leds, les résistances,... Tout ceci en se basant sur le transport du courant grâce aux pistes de cuivre reliant les composants entre eux ou aux fils conducteurs.

Dans le cadre de ce projet, l'électronique est utilisée comme outil permettant le traitement d'informations. Elle nous permettra, à partir de données entrantes (*inputs*) de fournir des données sortantes (*outputs*).

Comme nous le verrons par la suite, nous avons essayé de simplifier au maximum ce transfert d'informations. En effet, partant de la philosophie selon laquelle plus il y a de composants, plus la probabilité d'erreurs, de dysfonctionnements et d'incompréhensions est grande, nous avons créé des *objets* recevant une et une seule information et en transmettant également une et une seule.

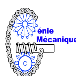
Par exemple, la carte électronique traitant le signal ultrason ne recevait qu'une seule information à savoir l'arrivée du signal ; et ne transmettait qu'une seule information à la carte principale à savoir est-ce que oui ou non, le robot adverse se situe dans la zone critique. De cette façon, tous les inputs et les outputs pouvaient être interprêtés sous la forme d'un seul "bit". Soit le courant passe et une action peut se réaliser, soit le courant ne passe pas et le robot continue ses tâches habituelles.

Le *pôle* électronique s'est également occupé de tout ce qui concerne le câblage. Les interrupteurs de stratégie, d'alimentation, ainsi que les différents *boutons* ont à chaque fois été réalisés/commandés après reflexion. Nous avons également opté pour une standardisation des connecteurs et des couleurs de fils. Leur solidité, leur petite taille et leur aisance au niveau du montage nous ont fait choisir les connecteurs de la série JST (voir figure 10.1). Remarquons également que l'achat d'une pince à sertir haute technologie à fait pencher la balance du côté de ce fabricant.

Pour la couleur des fils, nous avons ratifié la convention suivante dans un espoir de qualification européenne :

- Noir : masse (GND)
- Jaune : signal (SIGN)
- Rouge : alimentation (5V)

Toujours dans une idée de standardisation, l'emplacement des fils dans les connecteurs sera toujours le même. Nous avons choisi la combinaison Noir-Rouge-Jaune lorsque le connecteur



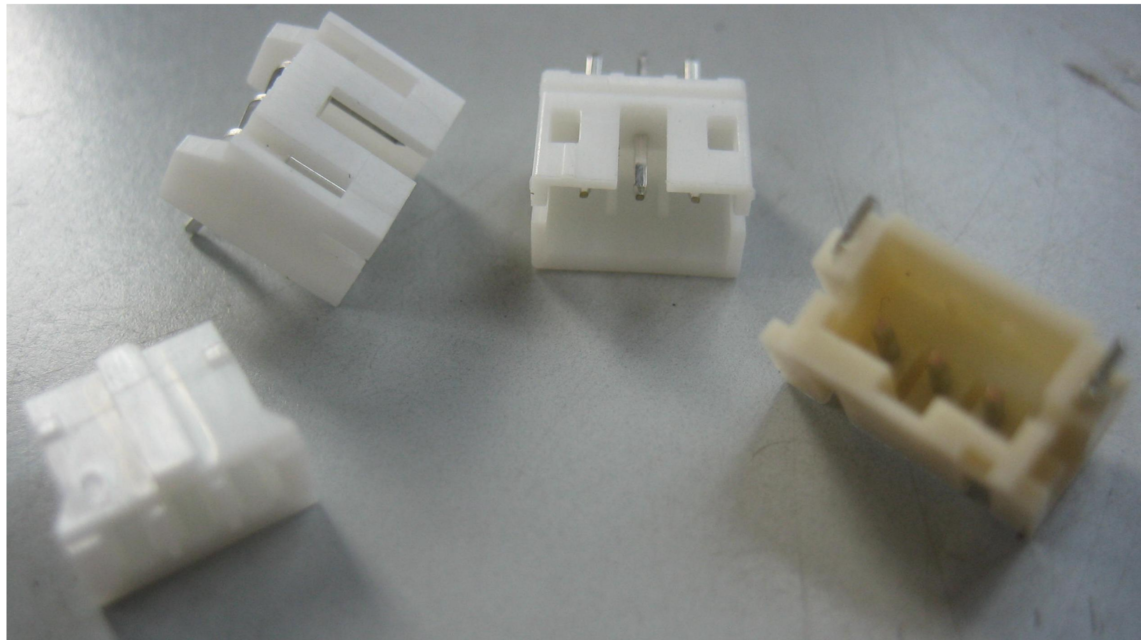


FIG. 10.1 – Connecteurs JST 3 logements

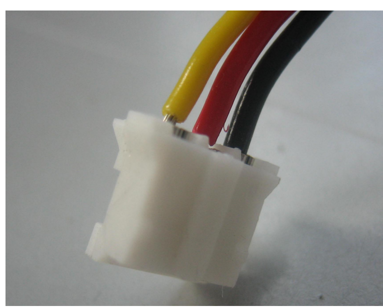
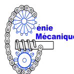


FIG. 10.2 – Connecteur JST 3 logements câblé

est posé sur un plan horizontal, l'excroissance vers le haut, de gauche à droite (voir figure 10.2).

Cette partie du rapport s'étalera sur plusieurs chapitres. Tout d'abord, et ceci dans un but tout à fait didactique, nous parlerons de la plaque de test que nous avons designée et réalisée au cours du premier semestre. Les trois chapitres suivants seront consacrés aux trois technologies que nous avons développées afin de permettre au robot d'éviter un obstacle ou de se repérer sur la table de jeu. Ensuite, nous nous intéresserons aux microswitches et nous verrons en quoi ils ont permis au pôle informatique d'appliquer la stratégie. Enfin, nous fournirons les schémas des montages électriques des différents interrupteurs avant de tirer les conclusions qui s'imposent.

L'entièreté des réalisations du pôle l'ont été dans un but d'être réutilisées les années suivantes. Le boitier de démarrage, le module ultra-son, le module infra-rouge, la balise ainsi que le laser (datant de l'année passée), ont été concus de manière modulaire. Cette technique présentait également l'avantage de pouvoir très simplement remplacer un composant défectueux ou inadapté. Pour exemple, on peut citer le changement du microcontroleur PIC réalisé en quelques secondes entre deux matches lors du concours.



Prise en main

Ce projet de robotique est un travail multidisciplinaire. En tant que mécaniciens dans l'âme, il a donc fallu nous plier, le temps d'une année, aux exigences de la capricieuse dame électronique. C'est pourquoi les premiers mois de nos travaux se sont essentiellement articulés autours de la prise en main des technologies gravitant autour de l'électronique. Nous avons donc passé beaucoup de temps autour d'exemples didactiques qu'il est, à nos yeux, peu utile de détailler dans ce rapport. En effet, ces applications n'ont pas eu d'impact direct sur des développements effectués sur le robot. Cependant, sans eux, il nous aurait été impossible de comprendre le fonctionnement de l'électronique à bord d'un robot et d'y apporter la moindre contribution. Nous allons cependant les citer ci-après sans entrer dans les détails spécifiques.
Notre premier contact avec le monde de l'électronique s'est effectué au travers la manipulation d'une *bread board* ou plaque de développement. Son utilisation nous a permis de nous familiariser avec le fonctionnement de divers composants de base. Nous avons par ailleurs effectué notre premier câblage du microcontrôleur grâce à ce type d'équipement. Ceci étant fait, nous avons appris comment programmer ces derniers. Cela est passé, notamment, par la (re)prise en main du language C. Nous avons, entre autres, appris à traîter un signal infrarouge et à utiliser les diodes afin de débugger notre montage. Nous avons également fait connaissance avec quelques usages pratiqués en électronique, par exemple, le fait de placer des diodes pour signaler une alimentation ou le câblage des microswitches avec seulement 2 fils(voir chapitre 11). Par après, il nous a fallu apprendre comment intégrer les systèmes étudiés sur plaque de développement dans notre machine. Ceci passe par la gravure de cartes électroniques. Nous avons alors pris en main un logiciel dédié à cet effet, EAGLE. Il nous a alors fallu acquérir le savoir-faire inhérent au processus de gravure proprement dit. Ceci est passé par la maîtrise du temps d'insolation et de trempe dans l'acide, le perçage à l'aide de mèches de précision et le soudage des composants sur la carte. Toutes ces techniques étant acquises, nous avons pu les mettre en application pour résoudre les problèmes concrets se présentant lors du développement des solutions utilisées sur le robot.

Remerciements

Nous tenons particulièrement à remercier M. Christophe Chariot et M. Laurent Pinchart pour leurs explications claires et leur patience. Concernant la balise ultrason, nous ne serions jamais parvenu à la mettre au point sans les conseils avisés de M. Pierre Lecomte et M. Frédéric Coquelet du service de Physique Générale.

Chapitre 11

Microswitches

11.1 Introduction

Un microswitch est un composant électronique à 3 bornes et un levier pouvant actionner un interrupteur sous une force très faible (voir figure 11.1). Il permet de faire passer le courant d'un circuit à un autre suivant l'état physique dans lequel se trouve l'interrupteur. L'emploi le plus courant de ce genre de composant est le "click de souris". Il a été inventé en 1932 par Peter McGall, Illinois.

11.2 Principe

Supposons que la borne C amène le courant. Celui-ci peut ensuite sortir du microswitch par la borne NC (*normally closed*) ou par la borne NO (*normally opened*) mais jamais par les deux bornes en même temps. Le choix de la borne de sortie dépend de la position du *levier*. Si celui-ci est à l'état de repos (*original state*), le courant entrera par la borne C et sortira par la borne NO. La borne NC est alors flottante. Si le levier est enfoncé, il appuie sur un interrupteur permettant la commutation. Le courant passe alors de la borne C à la borne NC, laissant la borne NO flottante.

Remarquons que si le passage de l'état de repos à l'état enclenché se fait sous une très

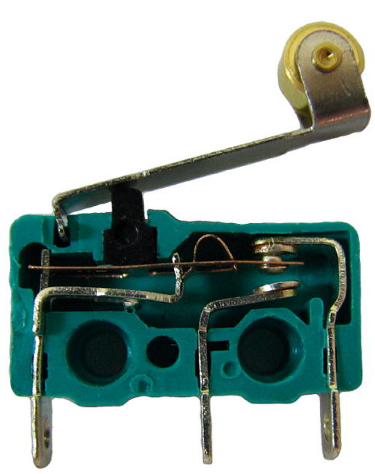
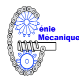


Fig. 11.1 – Microswitch

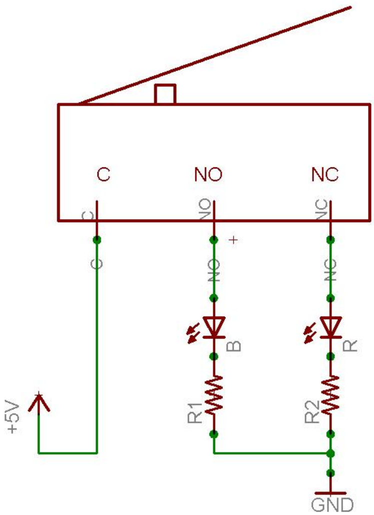


FIG. 11.2 – Montage didactique

faible pression, le repositionnement à la position d'origine nécessite lui une absence quasi totale d'effort sur le levier. On parle de contact *franc*.

Si nous réalisons le montage didactique de la figure 11.2, on voit que si le levier est à l'état de repos, c'est la led R qui va s'allumer. En effet, le courant provient de l'alimentation 5V, arrive dans le microswitch par la borne C et continue son chemin vers la borne NC. "Normally closed" traduisant ici le fait que, à l'état normal (c-a-d l'état de repos), le circuit est fermé. Si le circuit est fermé, le courant peut donc passer. A contrario, la led B ne s'allumera pas. En effet, aucun courant ne passant dans la branche contenant la led B, celle-ci restera éteinte.

Par contre, si le levier appuie sur l'interrupteur, c'est la led B qui s'allumera et non la R. En effet, la situation est exactement l'inverse de la précédente.

11.3 Applications

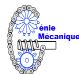
11.3.1 Microswitches de contact

Dans le cadre de ce projet, les microswitches sont utilisés pour permettre au robot de distinguer deux situations distinctes

- situation 0 : rien à signaler, tout se déroule comme prévu dans la stratégie de base
- situation 1 : le microswitch est enclenché, un évènement s'est produit, le robot est susceptible de changer d'état.

Remarquons qu'au point de vue mécanique, les microswitches sont insérés dans leur logement d'une manière telle que l'on parle d'*attaque indirecte*. En effet, comme le montre la figure 11.3, l'interrupteur du microswitch est enclenché lorsque le robot est en situation 0. C'est seulement si une pression est exercée sur l'écrou borgne situé au bout de la tige filetée que l'interrupteur est déclenché. Comme expliqué dans la partie mécanique, la tige filetée permet de maintenir le levier du microswitch en position basse.

Au point du vue du montage électrique, plusieurs solutions s'offraient à nous. Les schémas de la figure 11.4 présentent les deux possibilités de câblage les plus simples qui permettent la différenciation des deux situations (0 et 1). Ils se basent tous sur le principe de la *pull-up* relative à l'entrée SIGN.



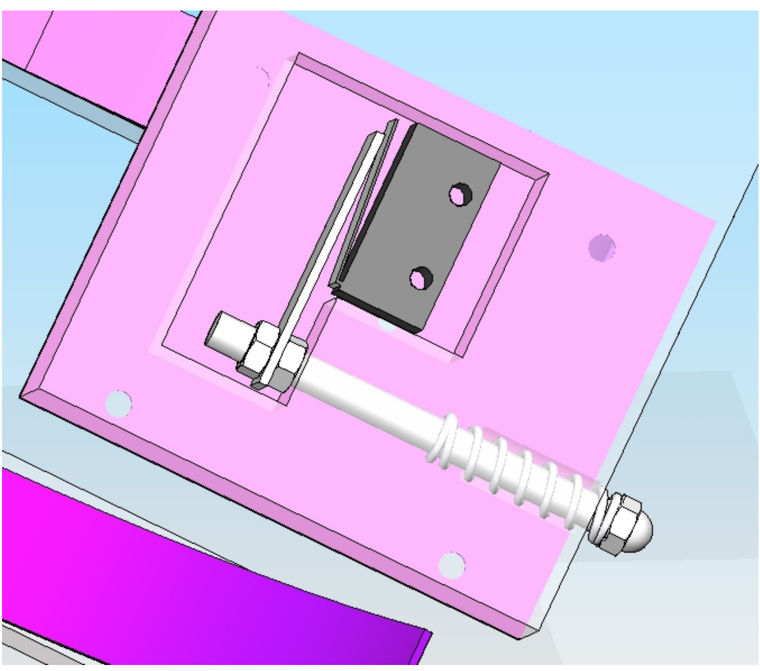


Fig. 11.3 – Montage en attaque indirecte du microswitch, vue du bras

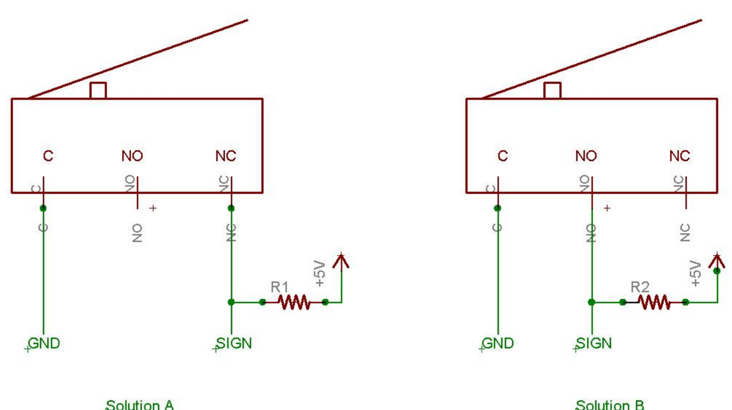
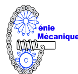


Fig. 11.4 – Deux possibilités de câblage des microswitches

On constate qu'en fait, seulement deux bornes sont utilisées.

Pour la solution A, on voit que lorsque le microswitch sera en position de repos (donc, lorsque l'écrou borgne de la tige filetée sera sollicité). le courant circulera à travers le microswitch pour aller vers la masse. L'information à la borne SIGN sera alors 0. En effet, si le courant passe à travers la résistance, il s'en suit une chute de potentiel et donc, l'entièreté du montage situé en aval de cette résistance sera au potentiel de la masse c'est-à-dire zéro. Par contre, si aucune force n'est exercée sur l'écrou borgne, alors aucun courant ne circulera dans le microswitch. La branche SIGN sera au même potentiel (5V) et l'information sera alors 1. Nous avons donc bien distingué les deux situations.

La solution B est tout à fait similaire à la solution A. La seule différence est que l'on aura l'information 0 si l'écrou borgne n'est pas sollicité et vice-versa. Le choix de la solution A est arbitraire, elle présente l'avantage d'écarter au maximum les fils électriques et donc de diminuer les risques de mauvais contact.



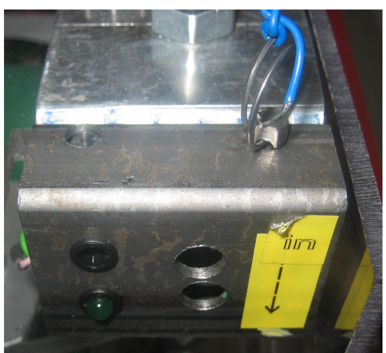


FIG. 11.5 – Boîtier de démarrage

11.3.2 Microswitch de démarrage

Le règlement du concours nous obligeait à démarrer notre robot en "tirant sur une ficelle". Dans cette optique, nous nous sommes démarqués des robots des années précédentes qui utilisaient un système de colson afin de faire passer un microswitch de l'état enclenché à l'état de repos. Cette technique, bien que simple et très facilement mise en oeuvre nous semblait incertaine. Nous avons donc décidé de concevoir un *boîtier de démarrage*.

Principe mécanique

L'idée principale était de fixer solidement un microswitch à un morceau d'acier et d'utiliser une tige rectifiée en acier également pour passer de l'état enclenché à l'état de repos. La figure 11.5 nous éclaire.

La difficulté était ici de parfaitement percer les trous parlesquels passera la tige rectifiée afin de s'assurer du changement d'état du microswitch.

Un méplat a été réalisé à l'extrémité de la tige rectifiée afin de percer un trou dans lequel on passera la ficelle de démarrage. Si la tige rectifiée est placée verticalement, il suffit de tirer la ficelle pour que le microswitch passe à l'état de repos. En effet, la tige rectifiée ne maintiendra alors plus le levier du microswitch sur l'interrupteur. Comme précédemment, on distingue donc deux situations :

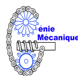
- situation 0 : l'interrupteur est libéré, la tige vient d'être enlevée, le robot a l'autorisation d'appliquer la stratégie.
- situation 1 : l'interrupteur est enclenché, la tige est dans son logement, le robot reste fixe.

Montage électrique

Nous avons en réalité réalisé plusieurs montages électriques. Notre première idée était de placer des leds rouge et verte en série dans le circuit afin de vérifier si le changement d'état s'était bien réalisé au niveau du microswitch. Ce besoin de vérification se justifiait par le fait qu'une fois le boîtier placé dans le robot, le léger son émis par l'interrupteur lorsque celui-ci est libéré est tout à fait inaudible. Il nous fallait donc une vérification visuelle.

Nous avons tout d'abord réalisé le montage illustré à la figure 11.6 A. On voit tout de suite apparaître deux problèmes.

On voit qu'en aval des leds, les résistances font doublons. En effet, il suffit de placer une seule résistance en aval des leds mais après la jonction des deux branches. De cette façon, on



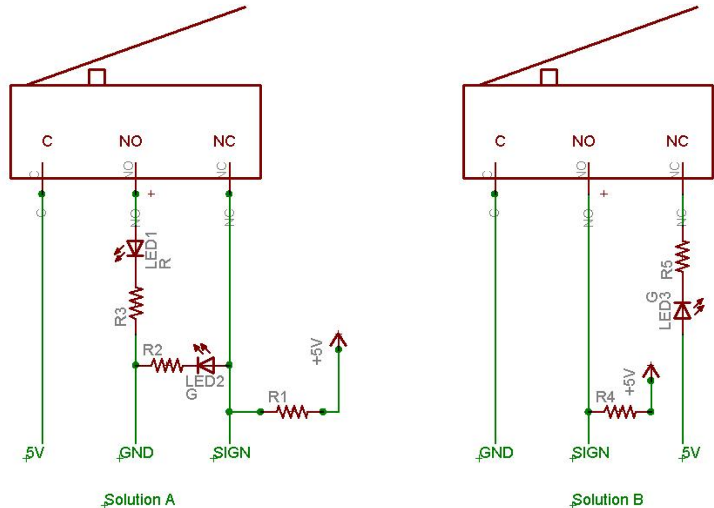


FIG. 11.6 – Deux possibilités de câblage du microswitch de démarrage

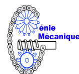
gagne en nombre de composants (et donc en coût) mais également en nombre de soudures (et donc de risque de faux contact) à réaliser.

La deuxième erreur est plus subtile. Après avoir réalisé ce montage et l'avoir implanté dans le robot, nous nous sommes rendus compte que, parfois, le robot démarrait sans que la tige de démarrage n'ait été retirée ! En analysant plus en détail le montage, il s'avère en fait que le signal peut être considéré comme *flottant*. En effet, si l'on considère que la tige se trouve dans son logement, l'interrupteur du microswitch est donc enclenché et la courant passe de la borne C qui amène le 5V à la borne NO qui alimente la led R. Mais de part la nature du SIGN reposant sur le principe de pull-up, on voit qu'un courant peut néanmoins circuler entre le 5V de la pull-up et le GND via la led G. Si ce courant circule, il y a donc chute de potentiel après la résistance de pull-up (qui rappelons-le est de haute impédance) et l'information signal passe à 1 alors qu'elle devrait rester à zéro.

Cette erreur de conception résulte de notre méconnaissance du principe de pull-up. Afin de palier ce manquement, nous avons opté pour un autre montage (voir figure 11.6 B). Cette solution nous permet simplement de voir si le microswitch est bien passé à son état de repos grâce à la led placé en série sur le 5V. Au niveau du SIGN, aucun doute n'est plus possible. Lorsque la tige est dans son logement, le courant passe via la pull-up vers la borne C et donc la masse. L'information du signal sera alors 0. Une fois la tige retirée, le courant passera depuis le 5V jusqu'à la masse via la led G. Au niveau du signal, l'information sera toujours 1 car aucun courant ne passe dans le branche SIGN.

11.3.3 Carte de gestion des switches

Les deux figures suivantes illustrent la carte de gestion des switches que nous avons réalisé. Cette carte, à laquelle sont reliés tous les câbles "noir-jaune-rouge" permet de servir d'intermédiaire entre les microswitches disposés partout dans le robot et la carte principale qui enverra les informations dans le programme. Les logements sont bien sûr disposé sur l'extérieur de la carte pour faciliter la fixation des connecteurs.



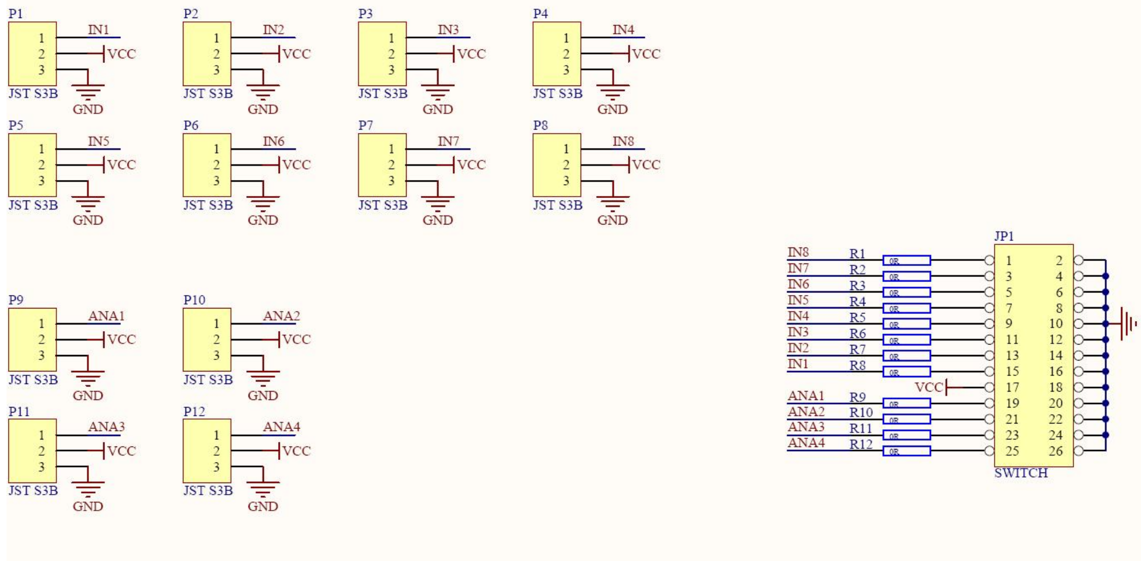


FIG. 11.7 – Schéma de la carte de gestion des switches

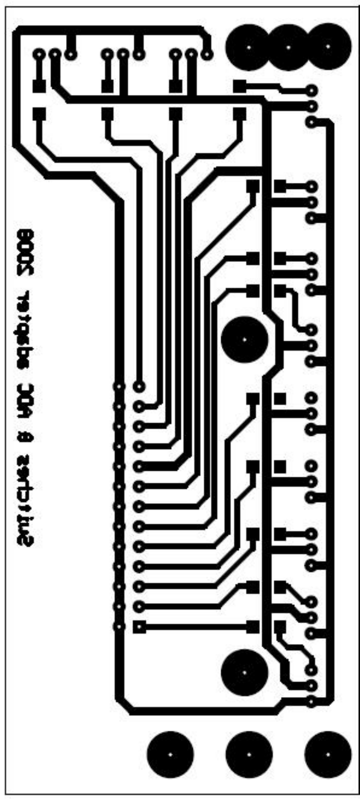
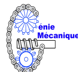


FIG. 11.8 – Carte de gestion des switches

11.4 Carte d'extension

11.4.1 Position du problème

Le système de gestion informatique fourni par M. Ludovic Dufranne possède des entrées pour seulement 8 switches. Or, la stratégie choisie impose un nombre plus important d'entrées



I/O. Ainsi, il nous est nécessaire de concevoir une carte d'extension. Cette dernière est réalisable grâce à la connexion par bus I^2C disponible sur la carte mère. Par ailleurs, comme expliqué au paragraphe 14.2.1 du chapitre 14, nous avons également besoin d'une carte d'extension afin de traiter le signal infrarouge. Pour des raisons d'encombrement minimum tant au point de vue de la taille de la carte qu'au point de vue de la longueur des fils, nous fusionnons ces deux cartes en une seule qui pourra effectuer les 2 fonctions.

11.4.2 Principe

Le principe de la carte d'extension est très simple. Il s'agit d'une carte sur laquelle viennent se brancher 8 microswitches. Le signal émis par chaque microswitch est envoyé sur le bus I^2C [1]. Chacun des branchements est évidement affecté d'une adresse liée à l'adresse de la carte d'extension de manière à ce que la carte mère puisse détecter d'où provient l'information.

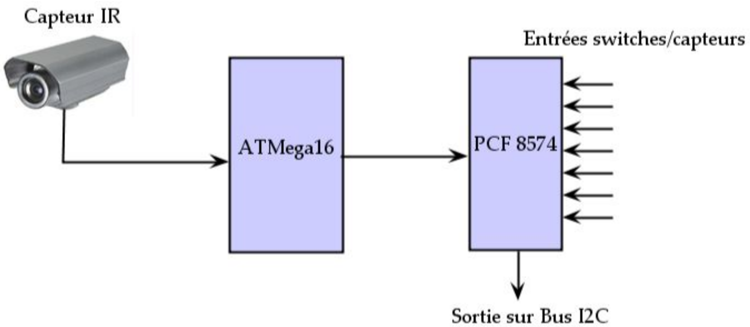
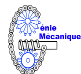


FIG. 11.9 – Principe de la carte d'extension pour les entrées I/O

11.4.3 Réalisation

En pratique, la réalisation de ce type de carte est assez simple. Afin d'envoyer sur le bus les informations collectées issues des différents capteurs, nous avons besoin d'un composant classique dont les références sont PCF8574. Ce composant possède 8 ports d'entrée/sortie (P0 à P7) sur lesquels viennent se brancher les capteurs (voir figure 11.10(a)). Il est aisé de communiquer en lecture ou en écriture avec ce composant via le protocole I^2C en lui affectant une adresse à l'aide d'un autre élément appelé dipswitch. Il s'agit en fait d'une série d'interrupteurs (3 pour notre application) que l'on branche sur les ports A0 à A2 du PCF8574 afin de lui attribuer une adresse (voir figure 11.10(b)). Ainsi, il est possible d'ajouter 8 cartes d'extension sur le bus I^2C étant donné que l'adressage se fait sur 3 bits ($2^3 = 8$ adresses). La schématique et le board de cette carte sont fournis en figure 11.11.

[1] Le bus I^2C est un protocole de communication série, c'est-à-dire que toutes les données sont envoyées à la suite l'une de l'autre à chaque coup d'horloge. Les informations circulent donc par un seul fil, contrairement à la liaison parallèle avec laquelle plusieurs informations sont envoyées en même temps, mais sur différents fils. De par ces considérations, la communication série est plus lente que la parallèle mais reste tout à fait acceptable pour l'application qui nous occupe. [1]



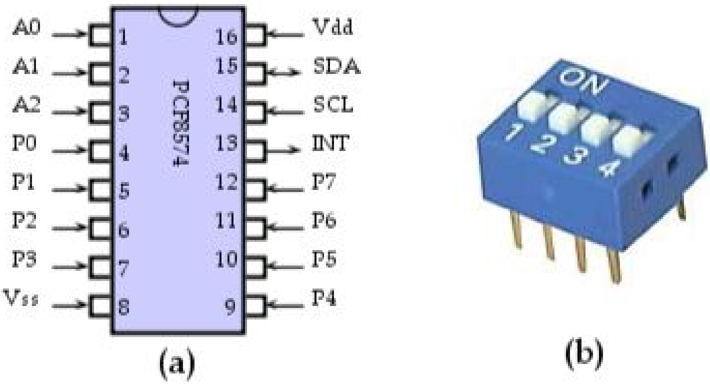
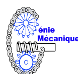


FIG. 11.10 – (a) Pin configuration du PCF8574 - (b) Modèle de dipswitch utilisé pour effectuer l'adressage du composant (le quatrième interrupteur n'étant connecté à rien)

11.5 Blindage des fils

Notre robot, comme toute machine électrique, est parcouru par de nombreux courants. Ceux-ci sont, tantôt des courants de commande de moteurs, tantôt des courants issus de l'activation de l'un ou l'autre switch, tantôt des courants provenant des codeurs des roues. On peut affirmer, sans se tromper que tous varient régulièrement au cours du temps. Or, nous savons que tout déplacement de charge électrique entraine la création d'un champ électromagnétique. Ce dernier, agissant sur une portion de conducteur entraîne l'apparition d'un courant à l'intérieur de celui-ci ; courant qui peut *polluer* le signal initialement attendu dans la portion de conducteur concernée. Par ailleurs, les moteurs utilisés pour la propulsion sont des moteurs *à balais et collecteurs*. Il se crée donc sans arrêt des petits arcs électriques à l'extrémité de ces balais qui induisent la création de champs électromagnétiques à très haute fréquence. Ceux-ci traversent un grand nombre de matériaux et sont donc succeptibles de polluer considérablement les signaux parcourant les conducteurs à l'intérieur du robot. Il pourrait s'en suivre des comportements anormaux de notre machine.
Afin de nous prémunir de tout problème de ce type, et sous les conseils avisés du Prof. M. Delhaye, nous avons opté pour un blindage des fils conducteurs de signaux de commande. Le câble utilisé est celui présenté à la figure 11.12. Il s'agit de matériel utilisé essentiellement pour la connection des alarmes domestiques. Il va donc sans dire que son efficacité est remarquable. Il est composé de 5 fils conducteurs protégés par une tresse de blindage en cuivre que nous relions à la masse du robot. De cette manière, lorsque ce câble est soumis à un champ électromagnétique, les électrons présents dans la tresse de blindage se mettent en mouvement et créent un champ à l'intérieur du blindage tendant à annuler celui qui a fait naître leur mouvement. Accessoirement, ce type de fils nous permet, par ailleurs, d'obtenir un câblage beaucoup plus propre et ainsi une plus grande facilité à détecter la source d'éventuels problèmes. Une perspective intéressante à envisager pour les années futures serait, dans le cas où l'on utilise encore le même type de moteurs, d'effectuer un capotage de ceux-ci. Le matériau utilisé devrait être un alliage possédant une très haute perméabilité magnétique, idéalement du μ-métal.



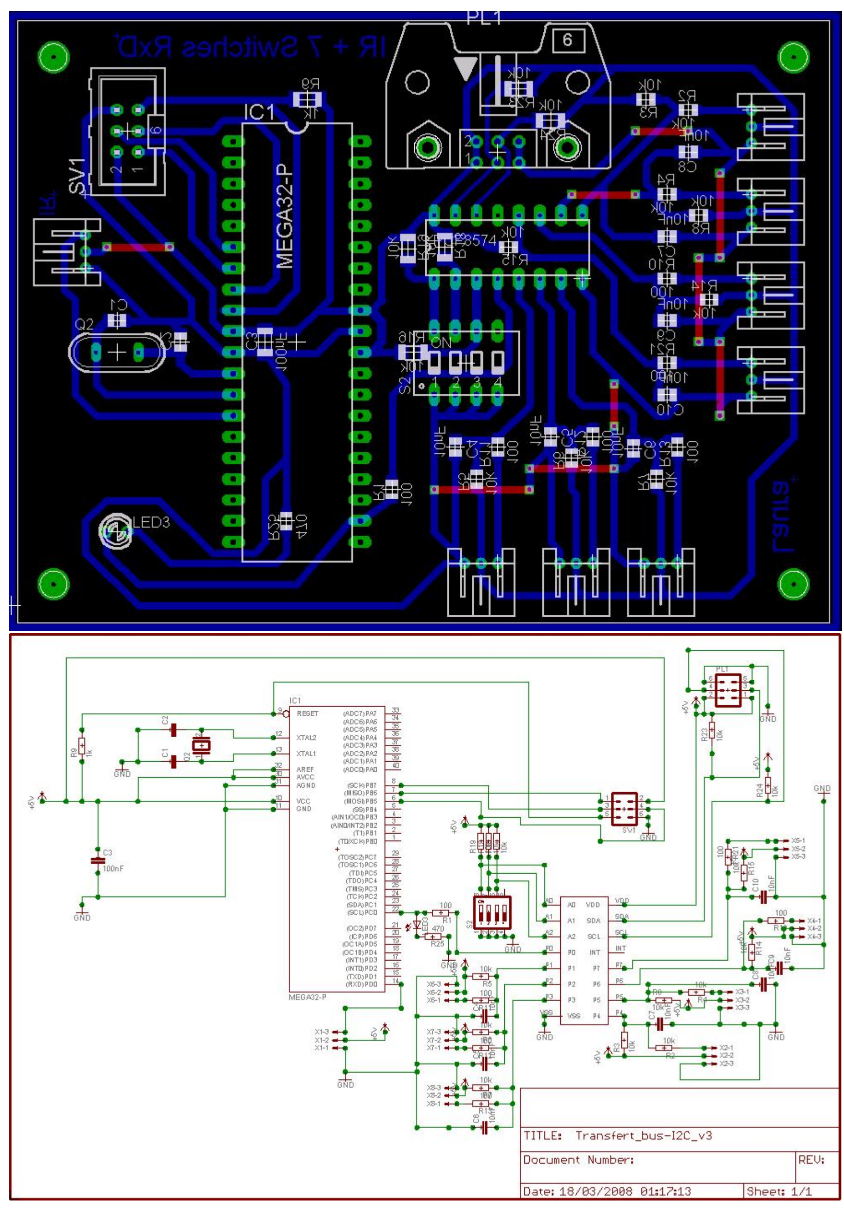
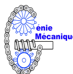


Fig. 11.11 – Board et schematic de la carte d'extension des switches



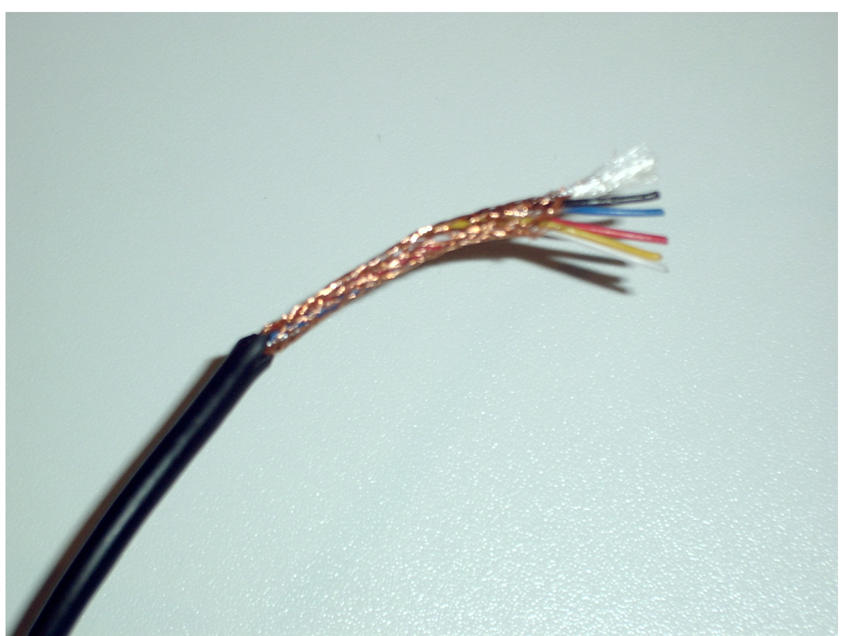


FIG. 11.12 – Câble blindé utilisé pour éviter tout parasitage des signaux

Chapitre 12

Laser

12.1 Introduction

Afin de nous situer sur la table de jeu, nous avons (re-)développé une technologie en parallèle à celle du traitement d'infra-rouges. En effet, les étudiants ayant participé au projet EUROBOT l'année passée avait réalisé de manière très efficace un laser permettant de recaler le robot dans une direction bien précise.

Nous avons donc décidé de reprendre cette technologie en apportant les quelques modifications nécessaires propres au règlement de cette année.

12.2 Principe

Le principe du laser est rigoureusement le même que celui de l'année passée.

Le *laser*, considéré sous la forme du boîte noire, envoie un rayon lumineux, émis par un phototransistor (voir figure 12.1), dans une seule direction et de manière très localisée. En un endroit fixe de la table, on dispose une *balise* cylindrique recouverte d'une bande réfléchissante (voir figure 12.2). Le rayon laser peut donc être réfléchi par la balise. On dispose alors un capteur infra-rouge près de la diode laser afin de récolter l'éventuel rayon réfléchi.

Le phototransistor est monté accompagné d'une petite lentille. Celle-ci permet de focaliser les rayons diffus réfléchis par la balise sur le phototransistor situé au point focal de cette lentille. Le schéma de la figure 12.3 nous éclaire.

FIG. 12.1 – Phototransitor monté avec lentille

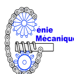


FIG. 12.2 – Balise laser recouverte de bande réfléchissante

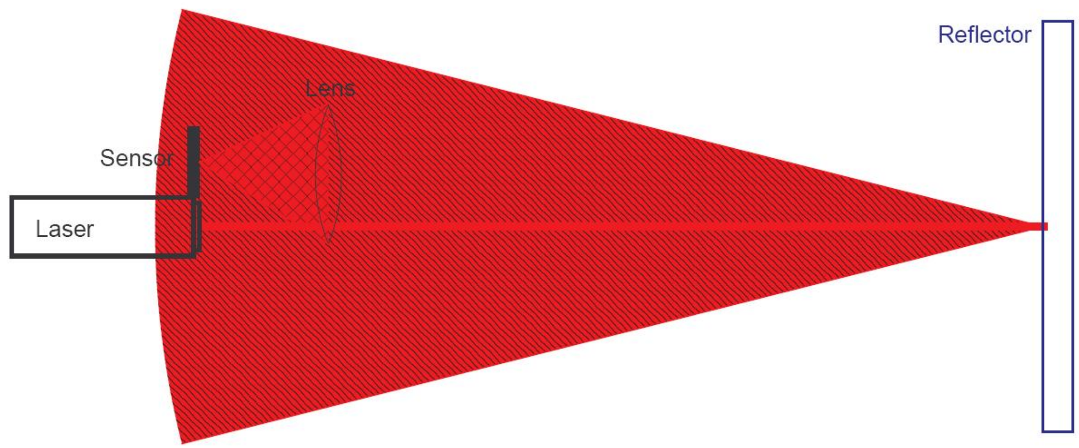
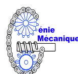


FIG. 12.3 – Principe de détection grâce au rayon laser



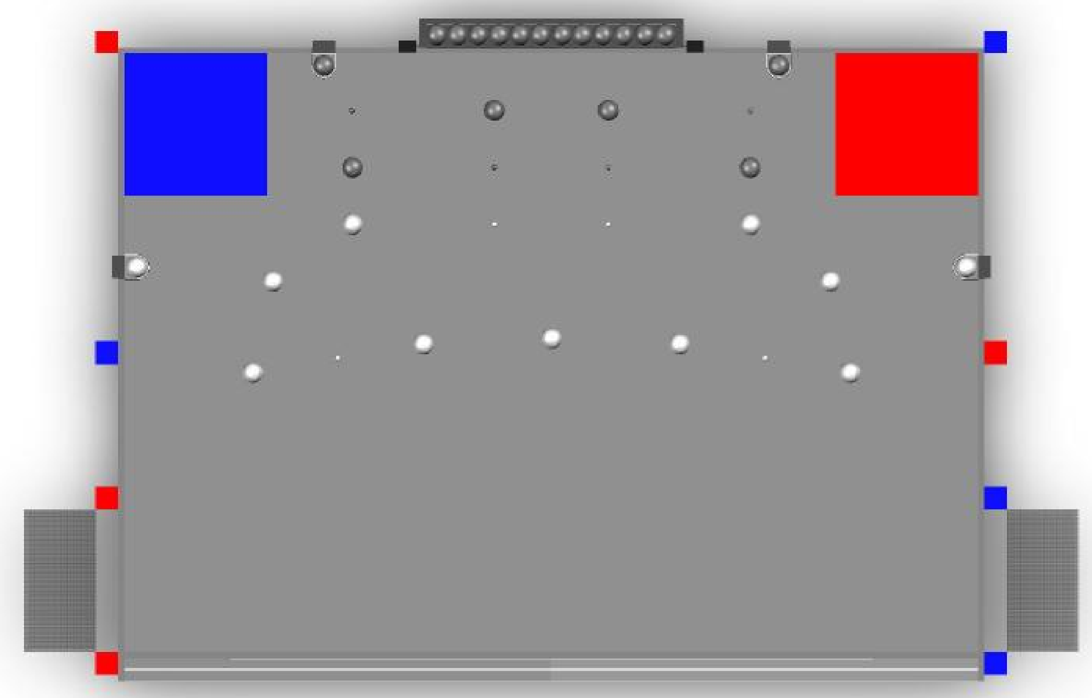


FIG. 12.4 – Emplacements possibles des balises

12.3 Applications

Notre idée était d'utiliser le laser afin que le robot puisse se diriger vers le conteneur réfrigéré. En effet, comme le montre la figure 12.4, nous étions autorisés à disposer des balises à côté de ces conteneurs à une hauteur de 350 *mm* au dessus du niveau de la table.

Afin de mettre le plus de chances de notre côté, nous avons choisi de disposer la balise sur la plate-forme la plus éloignée du point de départ, dans le coin opposé. Ainsi, le robot, après avoir avalé les balles, effectuerait une rotation jusqu'à ce qu'il recoive le rayon réfléchi. Il pourrait alors se diriger en ligne droite vers la balise sur une distance fixée. Une fois cette distance parcourue, il lui suffirait de pivoter afin de s'aligner face au conteneur réfrigéré, de s'en approcher et de catapulter les balles.

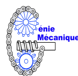
Notre objectif était de faire fonctionner en continu le laser. En effet, vu notre stratégie minimaliste en termes de déplacements, le rayon réfléchi ne pouvait normalement pas être détecté lors de la phase d'approche du réservoir vertical. Le seul moment où la détection était possible était lors de la traversée selon la diagonale de la table, ce qui était le but recherché.

Un autre avantage du fonctionnement continu était de polluer l'environnement pour l'adversaire !

12.3.1 Traitement du signal

Afin d'exploiter le phototransistor, il est nécessaire de réaliser un petit circuit électronique composé de deux parties. La première (à gauche sur la figure 12.5) permet de traiter le signal. La deuxième (à droite) définit la tension de référence à partir de laquelle le signal passera de 0 à 1 traduisant le fait que le phototransistor reçoit le rayon réfléchi par la balise.

Le phototransistor est connecté en X2-2 (collecteur) et X2-3 (émetteur). Le signal à analyser



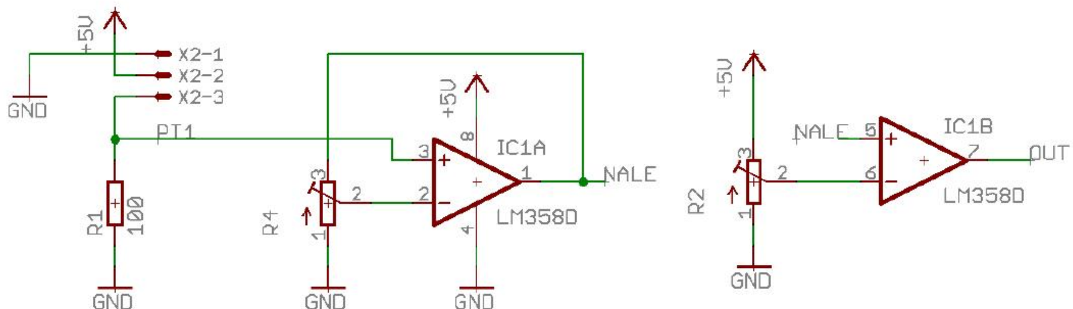


Fig. 12.5 – Circuit électronique permettant le traitement du signal laser détecté par le phototransistor

est la tension aux bornes de la résistance R1 (faible impédance). Si cette tension est nulle, le signal sera nul ; si cette tension est supérieure au seuil fixé, le signal sera unitaire et l'information sera transmise à la carte principale afin de faire réagir le robot.

Comme la tension aux bornes de R1 est relativement faible, on ajoute un amplificateur opérationnel monté en gain non inverseur afin de mieux discerner les états hauts et bas selon que le phototransistor détecte ou non le rayon réfléchi. Le gain de l'amplificateur opérationnel est réglé par le potentiomètre R4 à la valeur désirée. On compare ensuite la tension aux bornes de R1 à une tension dite *de référence* prise aux bornes du potentiomètre R2. On couple donc à ce potentiètre un ampli opérationnel qui servira de comparateur de seuil. Celui-ci n'enverra un signal OUT que si le faisceau laser réfléchi est détecté.

La figure 12.5 nous éclaire.

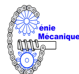
En fait, les deux potentiomètres permettent de régler la sensiblité du laser en fonction de l'environnement dans lequel il se trouve. La carte électronique que nous avons réalisée (voir figure 12.6) est presque identique à celle de l'année passée. Nous avons simplement agrandi les pistes pour éviter toute surprise.

La figure 12.7 nous montre la petite carte de traitement et les deux potentiomètres utilisés placés à l'arrière afin de pouvoir y accéder rapidement en cas de modification de l'éclairage ambiant.

12.3.2 Implantation dans le robot

Le laser (du moins le phototransitor) devait, pour bien faire, se trouver dans le même plan que la balise. Etant donné que nous n'étions autorisé à placer des balises que sur les plate-formes situées à une hauteur bien précise, nous avons donc raisonné en nous basant sur les dimensions fournies par le réglement.

Sachant que le logement pour balise se situe à une hauteur de 350 mm par rapport au niveau de la table, que le support de balise du robot se situe à une hauteur de 430 mm toujours par rapport au niveau de la table et qu'il était autorisé de disposer des éléments capteurs en-dessous de ce support, nous avons fixé le laser pour que le phototransistor soit situé à une hauteur de 390 mm au dessus du niveau de la table. La balise a donc également été désignée pour qu'à une telle hauteur, le rayon laser puisse être réfléchi par la bande réfléchissante. La figure 12.8 nous éclaire à ce sujet.



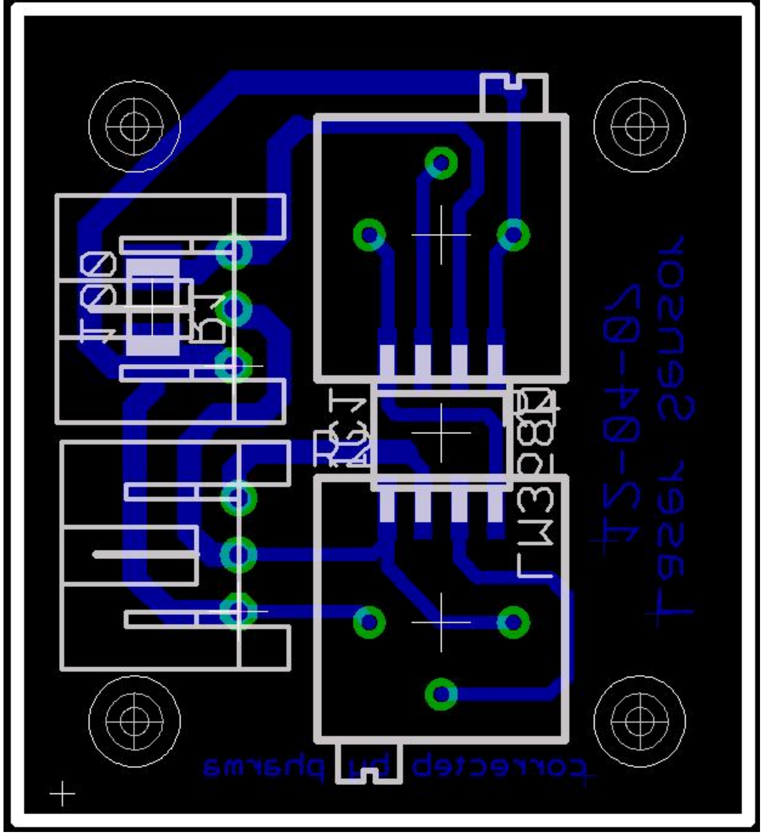
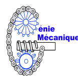


FIG. 12.6 – Carte électronique de traitement du signal laser

Remarque

Finalement, et cela suite à la défectuosité du moteur de la catapulte et donc au changement de stratégie, le laser, comme l'année passée, n'a pas été utilisé. Il reste néanmoins disponible pour les prochaines *Mons Polytech Teams* !



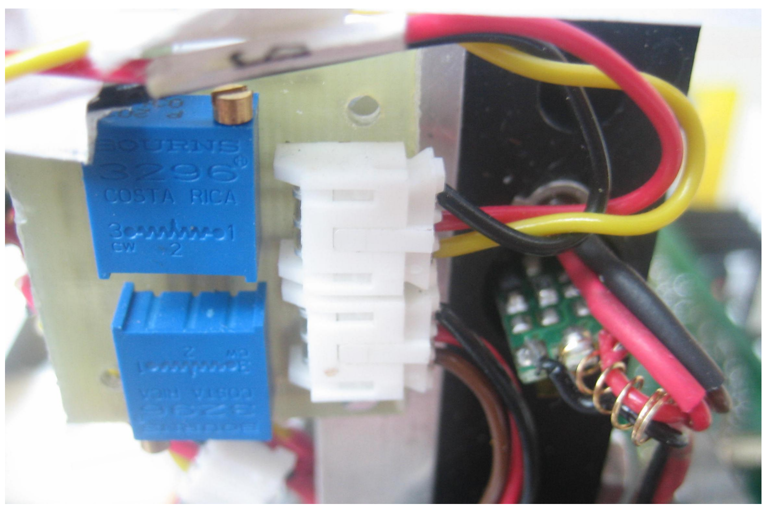


FIG. 12.7 – Arrière du laser : carte de traitement de l'information et potentiomètres

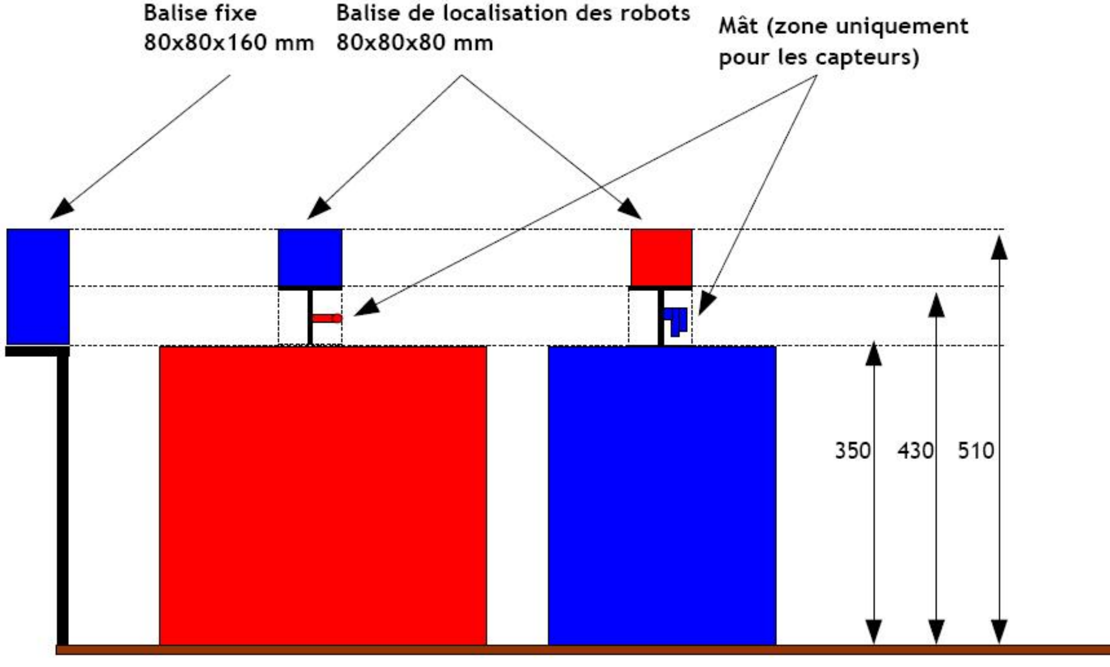


FIG. 12.8 – Extrait du règlement concernant les dimensions des supports de balise

Chapitre 13

Plaque de démarrage

13.1 Introduction

Lors de la conception du robot, il était évident que nous devions prévoir des interrupteurs afin d'alimenter le robot en énergie et de choisir la stratégie (rouge ou bleu). Il semblait tout naturel également que ces interrupteurs soient faciles d'accès et regroupés afin de minimiser le nombre de câbles à l'intérieur du périmètre du robot. Nous avons donc choisi de récupérer une plaque d'aluminium rectangulaire dans laquelle nous avons percé les logements pour les différents composants.

Notons que le réglement nous imposait de disposer d'une bouton d'arrêt d'urgence de type coup de poing qui, une fois enclenché, coupe automatiquement tous les systèmes motorisés du robot.

13.2 Alimentation

Pour alimenter le robot en énergie, nous disposions de batteries au Nickel-Cadmium 12 Volts (voir figure 13.1).

13.2.1 Première version

Deux circuits d'alimentation ont été conçus en parallèle. Le premier, en 12 V, alimente toute la partie électronique. Il conduit le courant depuis les batteries jusqu'à la carte électronique de puissance. Cette dernière alimente ensuite la carte de gestion de switches (voir chapitre 11) en 5 V qui alimentera alors les différents modules électroniques (ultrason, infra-rouge, laser, switch de démarrage, switch de stratégie, bouton reset) en 5 V. Le deuxième, en 24 V, alimente les moteurs de propulsion.

Notre tâche était donc d'insérer un interrupteur afin d'*allumer* le robot, un bouton d'arrêt d'urgence et des fusibles de sécurité afin d'éviter qu'un courant trop important ne circule dans les fils conducteurs. Nous avons donc réalisé le schéma fourni à la figure 13.2.

Nous avons choisi des fusibles déclenchant à 6 A. Ce choix est évidemment fonction du fil conducteur choisi ($2,5\ mm^2$ de section). Le trait pointillé au milieu des deux interrupteurs (S1 pour l'interrupteur ON/OFF classique et Emergency Stop pour le bouton d'arrêt d'urgence) indique la synchronisation de la coupure au niveau des deux circuits. Nous avons donc choisi un interrupteur ON/OFF à 4 bornes (deux circuits) et un bouton d'arrêt d'urgence également à 4

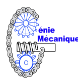


FIG. 13.1 – Batterie Ni-Cd 12 Volts

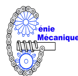
bornes (deux circuits). Nous avons acheté ce dernier auprès d'une société suisse. Il est constitué du bouton en lui même de type coup de poing et de son logement qui réalise l'opération mécanique de coupure des deux circuits. Les fils dénudés sont introduits dans les logements prévus à cet effet et sont ensuite serrés. De cette manière, on évite la présence de soudure près des composants. La figure 13.3 nous donne les caractéristiques principales du bouton "Mushroom" 04 2 NC de la marque EAO. Nous avons choisi un bouton de type 2 NC (2 fois normally closed) car à l'état de repos (original state), le courant doit passer. Par contre, une fois le bouton enclenché, les deux circuits doivent être ouverts. Remarquons que le bouton dispose d'une sécurité. En effet, une fois le bouton enclenché, les deux circuits restent ouverts. Il faut défaire la sécurité mécaniquement pour revenir à l'état de repos.

Remarquons que notre choix d'investir dans un bouton d'arrêt d'urgence et un bloc "boîte noire" se justifie par le fait que la solution pour laquelle avaient opté les équipes des années précédentes consistait en la création d'une nouvelle carte électronique avec un système relativement complexe de transitors distribuant le courant dans certains circuits et pas d'autres. Nous avons préféré rester simples et surtout fiables.

13.2.2 Seconde version

Après les premiers tests, nous nous sommes rendus compte quil serait plus aisé de réguler les moteurs de propulsion s'ils étaient alimentés en 12 V. De plus, le pôle mécanique ayant terminé les éléments motorisés auxiliaires (avalement, ascenseur et catapulte), nous avons décidé d'alimenter ces derniers en 24 V principalement pour une raison de vitesse d'exécution. Il nous fallait donc repenser complètement notre schéma d'alimentation.

La difficulté principale était ici de réutiliser le même bouton d'arrêt d'urgence (c'est à dire capable de couper deux circuits) alors que nous devions maintenant être capable de couper



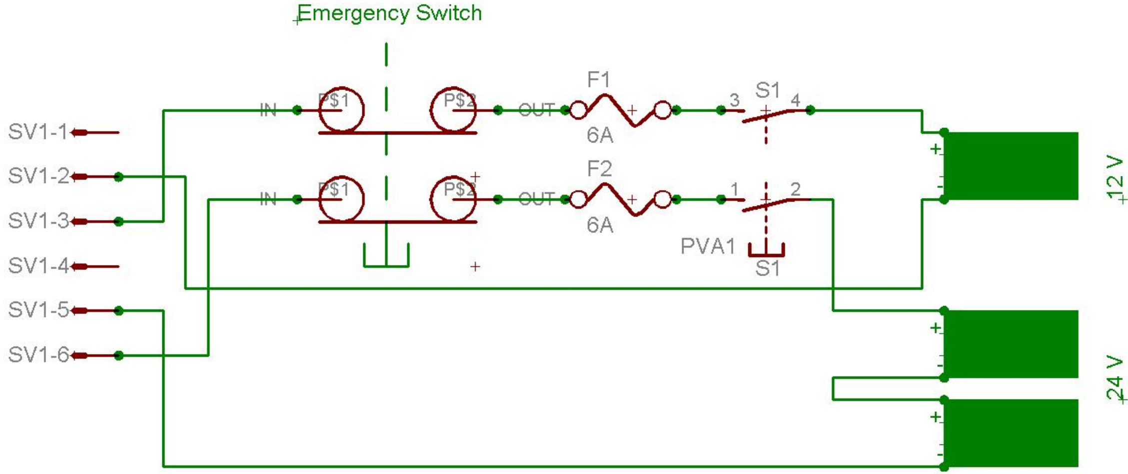


FIG. 13.2 – Première version du montage d'alimentation

3 circuits : l'alimentation électronique (12 V), l'alimention des moteurs auxiliaires (24 V) et l'alimentation des moteurs de propulsion (12 V). Au niveau emcombrement, nous voulions également ne conserver que 3 batteries.

Nous avons décidé de réaliser le câblage illustré à la figure 13.4.

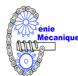
La première batterie alimente la partie électronique en 12 V comme précédemment. La deuxième batterie alimente tout d'abord les moteurs de propulsions en 12 V et une déviation a été posée de façon à ce que la borne négative de la troisième batterie soit mise en série avec la borne positive de la deuxième batterie. De cette façon, la borne positive de la troisième batterie sera à un potentiel de 24 V par rapport à la masse à 0 V. Les deux connecteurs Mollex 5566-4 permettent de distribuer le 24 V dans la carte de gestion des moteurs intermédiaires. Nous avons délibérement choisi de réaliser cette alimentation par deux entrées pour diminuer l'intensité du courant passant dans le fils. En effet, ces connecteurs ne permettent pas d'enficher des câbles de grand diamètre. Le risque de surchauffe était donc bien réel et nous avons voulu nous en prémunir.

On constate également que le jeu de batterie n'apporte que le potentiel positif (c-a-d 24 V) mais pas la masse à la carte de gestion des moteurs intermédiaires. Ceci se justifie par le fait que la masse (potentiel 0 V) est amené sur cette carte directement depuis la carte principale via une nappe. Nous avons donc "récupéré" cette masse et l'avons redistribuée via un autre connecteur (Weidmuller, en bas sur le schéma).

Cette manière de faire quelque peu bricolée nous permettait de garder le même bouton d'arrêt d'urgence. En effet, si l'on regarde sur le schéma de la figure 13.4, on voit qu'une pression sur ce bouton bloque les 3 circuits.

Nous avons également inséré un fusible sur le circuit d'alimentation des moteurs auxiliaires. Ce fusible déclenche à une intensité plus faible que les deux autres ; de cette façon, les parties electronique et propulsion peuvent continuer à fonctionner même si le fusible à 3 A a lâché. De plus, on s'assurait de cette façon de ne pas avoir de courant trop élévé dans les connecteur Mollex 5566-4.

Ce montage a été réalisé sur le robot et est toujours en fonction actuellement.



Colour	Part No., Plastic	Part No., Aluminium
Red	704.070.2	704.071.2
Yellow	704.070.4	704.071.4
Green	704.070.5	704.071.5
Black	704.070.0	704.071.0

Part No.	Contact type
704.900.1	1NO
704.900.2	1NC
704.900.3	2NO
704.900.4	2NC
704.900.5	1NO/1NC

FIG. 13.3 – Bouton d'arrêt d'urgence et bloc associé

13.3 Choix de la stratégie

A la lecture du règlement, notre robot est susceptible de jouer chaque match soit en *rouge*, soit en *bleu*. Le terrain de jeu étant parfaitement symétrique (à l'exception des balles placées de manière aléatoire), le pôle informatique a réalisé des straétgies *miroirs*. C'est l'arbitre du match qui détermine au début de la partie la couleur de l'équipe. Notre objectif était donc de réaliser un montage électrique basé sur un interrupteur tripolaire (voir figure xxx).

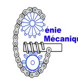
Remarquons qu'on parle plus souvent de *switch* que d'interrupteur.

Notre idée était de réaliser un montage permettant :

- de témoigner visuellement du choix de la couleur (rouge ou bleue)
- d'être exploité par l'intermédiaire de la carte de gestion des switches

Nous avons donc dressé le schéma de la figure 13.6 nous basant sur un connecteur JST et ses trois bornes, GND, 5V et SIGN. On voit que si l'interrupteur est en position haute, le courant circule depuis le 5V vers la borne GND en passant par la résistance et la led R. Le signal quand a lieu transmet l'information 1 puisque, via la pull-up interne, tout la branche depuis la borne SIGN jusqu'à la led B est au potentiel 5 V. Si par contre l'interrupteur est en position basse, le courant circule depuis le 5V vers la borne GND en passant cette par la résistance et la led B. Le signal transmet alors l'information 0 car on lui impose le potentiel de la masse c'est-à-dire 0V.

Le pôle informatique pourra alors traiter au niveau de la carte de gestions des switches l'information 1 comme correspondant à la stratégie rouge et l'information 0 comme correspondant à la stratégie bleue. Les leds sont utiles pour le manipulateur... elles permettent de voir si le choix de la stratégie est correct sans devoir passer par le programme.



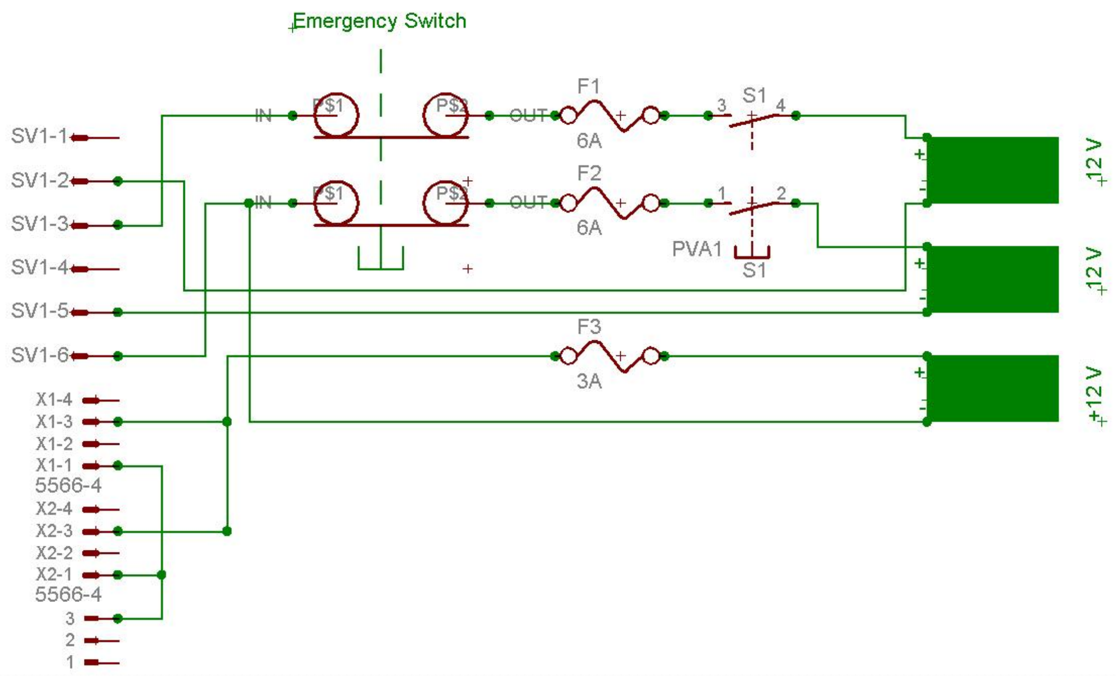


FIG. 13.4 – Seconde version du montage d'alimentation

Remarquons qu'une résistance a été placée en série des leds pour ne pas provoquer une chute de potentiel trop importante en leurs bornes.

13.4 Bouton RESET

Afin de permettre au programme de se réinitialiser, nous avons ajouter sur la plaque métallique un bouton poussoir. Ce bouton à deux bornes tout à fait classique permet le passage du courant quand il est pressé. Au niveau informatique, il suffit donc de détecter un front montant pour réaliser l'opération de réinitialisation.

13.5 Plaque complète

La figure 13.7 montre la plaque complète intégrée dans la robot.
On y distingue donc, de gauche à droite, de haut en bas :

1. le porte-fusible 3A en noir
2. le bouton d'arrêt d'urgence en rouge
3. le bouton RESET (carré vert)
4. les deux leds relatives à la stratégie (R ou B) et le switch permettant le choix
5. l'interrupteur générale ON/OFF
6. les deux porte-fusibles 6A en noir

Remarquons que la plaque a été fixée grâce à deux tiges filetées et des écrous. Les fixations ne sont pas symétriques car les tiges traversent tout le robot... Il fallait donc trouver des emplacements qui n'empêchaient pas le bon fonctionnement des autres systèmes.

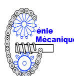


FIG. 13.5 – Interrupteur tripolaire

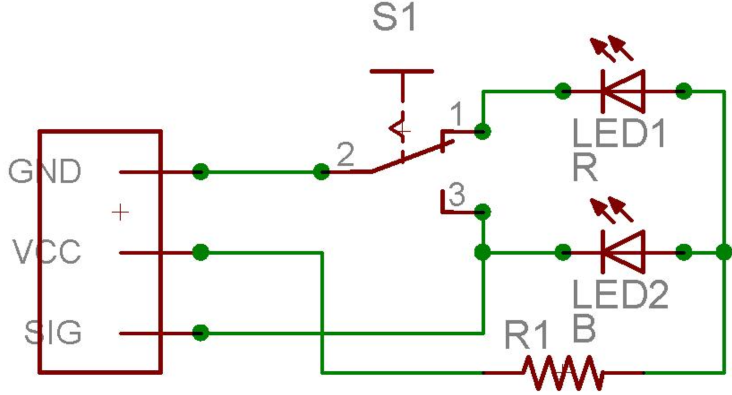
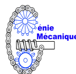


FIG. 13.6 – Montage relatif au switch de stratégie



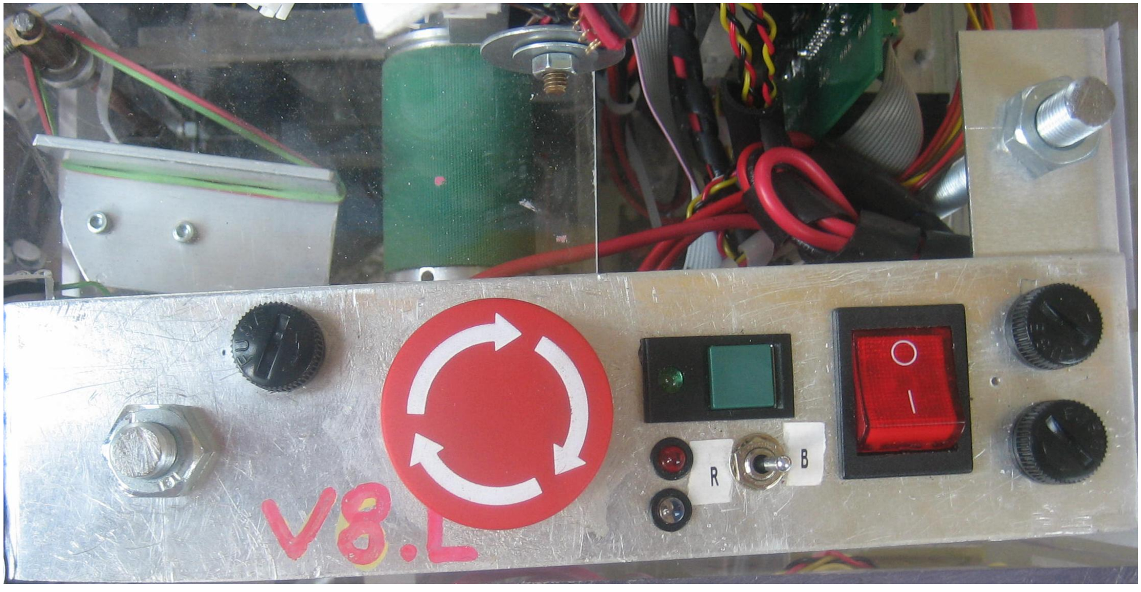


Fig. 13.7 – Plaque complète intégrée dans le robot

Chapitre 14

Évitement - détection de l'adversaire

14.1 Position du problème

D'après le cahier des charges rédigé par le comité organisateur du concours, chaque robot participant doit passer des tests d'homologation. L'un de ceux-ci consiste à prouver de manière convaincante au corps arbitral que notre machine est capable d'éviter un robot adverse qui viendrait se placer sur son chemin ou légèrement de côté. Afin de répondre au mieux à cette exigence, il est possible de placer sur l'adversaire une balise dont les dimensions sont telles qu'elle doit être comprise dans un cube de 80 mm de longueur d'arête. Cette dernière sera placée sur un mât placé au centre de la machine adverse et à une altitude de 430 mm.

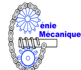
14.2 Technologies à disposition

Dans le paragraphe qui suit, nous allons présenter les diverses technologies à notre disposition et qui sont susceptibles d'être utilisées afin de résoudre ce problème.

14.2.1 La technologie infrarouge

Les télémètres

La technologie infrarouge est un outil puissant en robotique. Lors des précédentes participations de la Polytech Mons Team, les télémètres infrarouges ont été utilisés de manière assez intensive pour la détection d'obstacles, notamment. Le principe de fonctionnement du télémètre infrarouge est assez simple. Il est composé d'un émetteur IR et d'une rangée de récepteurs. L'émetteur envoit un signal modulé qui est réfléchi par l'obstacle. Selon l'angle avec lequel le rayon réfléchi arrive sur le récepteur, on peut déterminer la distance à laquelle se situe l'élément détecté (voir figure 14.1). Le domaine d'efficacité effectif de ce type de matériel est de 10 à 80 cm. Au delà, la précision devient trop faible. Ceci est simplement une conséquence du calcul trigonométrique effectué et des dimensions du télémètre. Cependant, il existe des limitations à ce type de dispositif. En effet, sous une lumière incidente directe sur le capteur, la détection ne se fait pas de manière optimale. Or, lors du concours, les lumières projetées sur la tables sont assez nombreuses et puissantes. Nous avons donc délibérément choisi de ne pas utiliser ce type d'équipement. Notre choix s'inscrit également dans la politique que nous nous sommes imposée et qui consiste à limiter au maximum l'aspect programmation dans notre



machine.

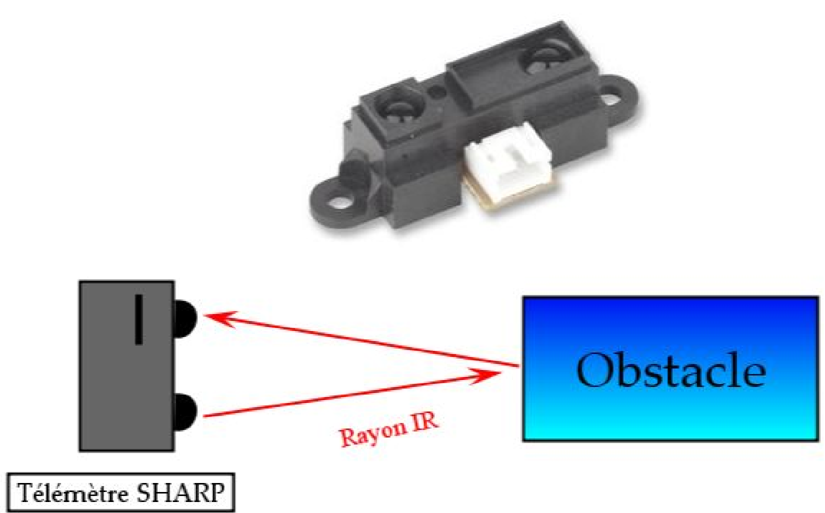


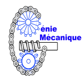
Fig. 14.1 – Télémètre SHARP - principe de fonctionnement

Les balises

Lors des précédentes participations de la *Mons Polytech Team* à la Coupe de Robotique, le service de physique générale a développé un système de balisage fonctionnant à l'aide de la technologie infrarouge. Il consiste en 3 balises émettant chacune un signal IR modulé à l'aide d'un émetteur non directionnel. Chacune de ces balises envoit un caractère (A, B ou C). Le récepteur IR possède également un cône de sensibilité proche de 180°. Il faut donc le munir d'une fenêtre de manière à sélectionner la direction dans laquelle les signaux sont détectés. Après diverses séances de tests, nous avons pu observer l'efficacité de ce type de dispositif. Malheureusement, pour l'application qui nous intéresse, il n'est pas utilisable. En effet, si on place une balise sur l'adversaire, notre machine pourra détecter quand il se déplace dans sa direction, mais il ne connaîtra jamais la distance qui le sépare de l'autre robot. Il faudrait donc coupler ce dispositif à un détecteur de présence de type Sharp, ce qui nous compliquerait considérablement la tâche.

Utilisation des balises IR. Nous avons donc délibérement décidé de garder ce dispositif comme dispositif de guidage vers le but. En effet, notre stratégie de base, commenvoitloppée au chapitre 20, consistait à prendre les balles de notre couleur au distributeur le plus proche et à se diriger par après vers le goal situé à l'opposé de la table. La portée des balises étant suffisament importante, en en plaçant une à coté du goal, à l'endroit autorisé, notre machine n'aura qu'à suivre le signal jusqu'au but et pourra alors tirer ses balles (voir figure 14.3).

Implantation. Il convient de placer le capteur de signaux infrarouges de manière à optimiser son efficacité. Comme autorisé dans le règlement, il est placé sous le support de balise adverssuffisammentnc fixé au mat de notre machine comme montré à la figure 14.4.



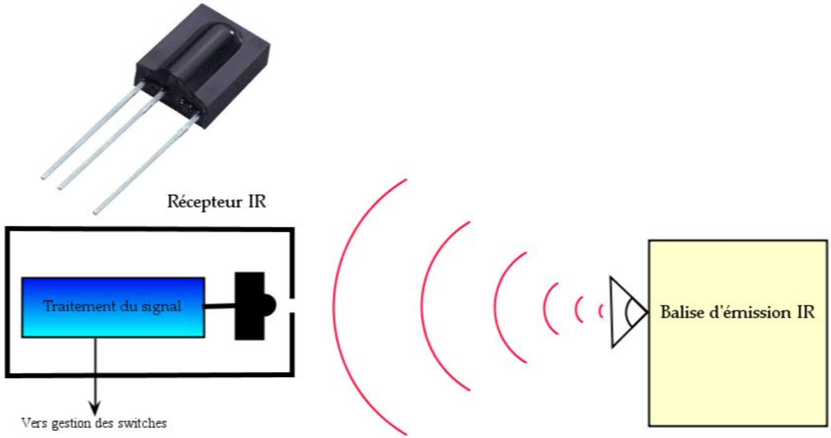


FIG. 14.2 – Balise Infrarouge - principe de fonctionnement

width=11cm]distrtogoal

FIG. 14.3 – Stratégie de déplacement du robot vers le goal à l'aide de la balise IR

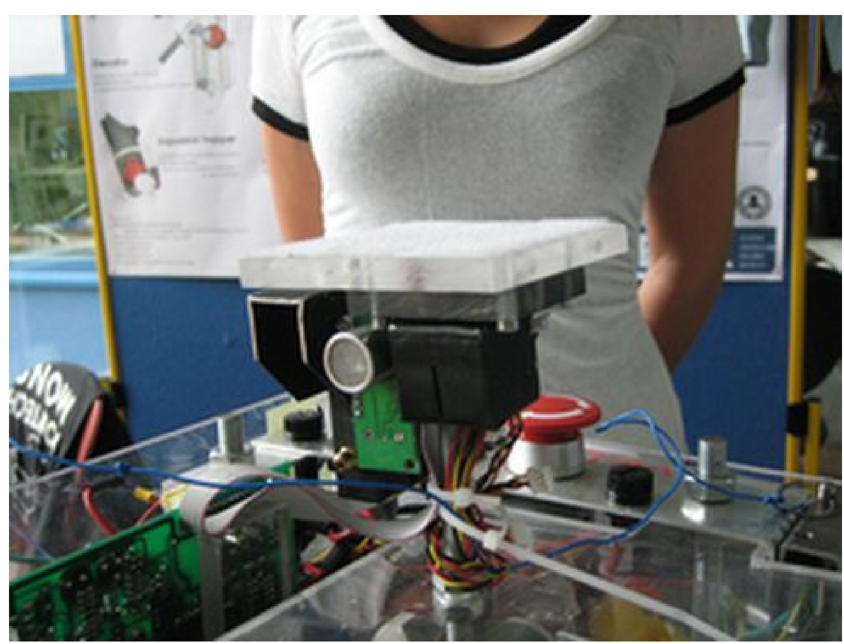
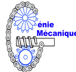


FIG. 14.4 – Mat sous lequel sont placés les capteurs tels que le capteur IR, le module US et la cellule laser

Traîtement. Pour des raisons évidentes d'encombrement, l'exploitation de l'information collectée par le capteur n'est pas effectuée à proximité de celui-ci, mais bien sur une carte dédiée située à l'intérieur du robot. Le traitement est effectué par un ATMéga 16[1] programmé par nos soins. On peut d'ailleurs trouver en Annexe B le code utilisé pour mener à bien cette application. Il s'agit en fait d'un code très simple dans lequel il suffit de programmer les interruptions dues aux changements de valeurs de la variable réprésentant le signal d'entrée. Ainsi, dès que

[1]Il s'agit d'un microcontrôleur au même titre que le PIC16F84. Sa datasheet est fournie en [2].



Codage en base				Caractère
10	8	16	2	
65	0101	41	10000001	A
66	0102	42	10000010	B
67	0103	43	10000011	C

TAB. 14.1 – Codage ASCII des caractères envoyés respectivement par les trois balises

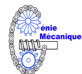
celle-ci reçoit le signal correspondant à celui émis par la balise (signal correspondant à la lettre A, B ou C traduite en code ASCII, voir tableau 14.1), L'ATMéga envoie un 1 logique (+5V) sur une entrée I/O (entrée utilisée également pour les switches). Cette opération est effectuée sans arrêt étant donné que le code entier est contenu dans une boucle infinie (`while(1)`) comme on peut l'observer dans l'annexe B.
Par ailleurs, pour des raisons d'encombrement et de clarté, le chip responsable du traitement du signal IR a été placé sur la carte d'extension des switches. En effet, ce microcontrolleur joue la même fonction qu'un microswitch puisqu'il renvoie à la carte mère une information 0 tant qu'il ne se passe rien et, lorsque le capteur détecte le signal attendu, l'information 1 est envoyée (voir le chapitre 11 pour plus de détails sur le fonctionnement des microswitches).

14.2.2 La technologie ultrason couplée à un signal radio

Toujours dans le cadre de développement de modules pouvant servir au concours de robotique, le service de Physique Générale a procédé au développement d'un système de détection mettant en oeuvre un couplage d'ondes radio et d'ultrasons. Ce système nous a été fourni sous forme d'une plaque de développement et notre rôle a donc été réparti sur plusieurs fronts. Premièrement, nous avons du trouver des composants actuels disponibles sur le marché et compatibles avec le montage afin d'être en mesure de construire plusieurs modules et de les documenter suffisament. En effet, certains des composants utilisés sur le montage de développement sont des éléments de récupération sur lesquels nous disposons de peu d'information ou qui sont carrément impossibles à obtenir de nos jours. Deuxièmement, nous avons eu la tâche consistant à choisir la configuration physique à adopter de manière à assurer une efficacité et maintenance optimale de la balise tout en respectant les exigences du réglement.

Principe

Le principe de ce montage est un peu plus compliqué que les deux précédents de par le fait qu'il met en oeuvre un couplage de deux types d'ondes.microcontrôlleurnc utiliser des émetteurs-récepteurs adaptés à chacune. Le dispositif que nous présentons ici fonctionne par *question-réponse*. Le module situé sur notre robot envoie en continu et ce, dans toutes les directions, un signal radio. Le signal choisi est, comme au paragraphe 14.2.1, un caractère traduit en code ASCII. Il s'agit ici de la lettre "E". Dès l'envoi de ce dernier, le microcontrôleur démarre un chronomètre. Dès réception, le dispositif embarqué sur le robot adverse déclanche le tir d'une salve ultrasonique dans la direction pointée par l'émetteur (en pratique, celui-ci possède un certain cône d'ouverture). Les ultrasons, plus lents que les ondes radio (voir tableau 14.2) mettent un certain temps à arriver sur le récepteur situé sur notre machine. Quand ce dernier reçoit la salve ultrasonique, il arrête le chronomètre. Par un calcul simple, il peut alors déterminer à quelle distance se situe la machine adverse. L'illustration du fonctionnement est présenté à la figure 14.5.



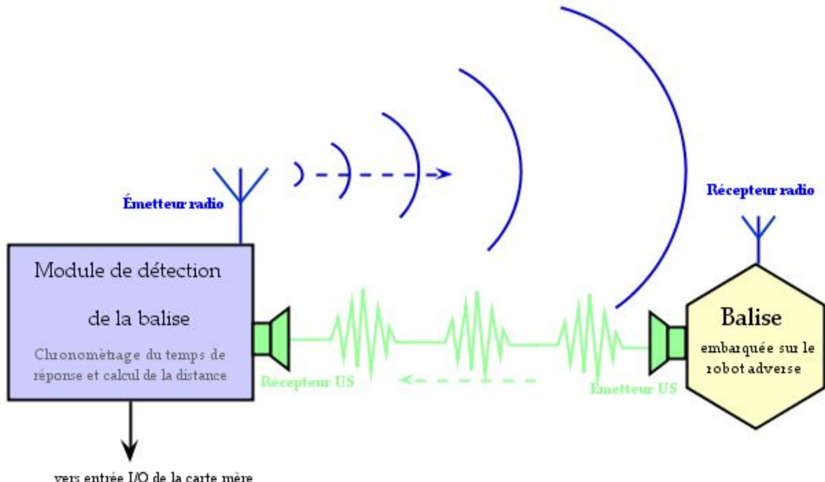


FIG. 14.5 – Principe de fonctionnement de la balise de détection radio-ultrasons

Vitesse des ondes	
Ondes radio	300 000 000 m/s
Ultrasons	300 m/s

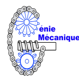
TAB. 14.2 – Vitesse de propagation des ondes dans l'air

Réalisation

La balise

Fonctionnement Comme expliqué au paragraphe 14.2.2, nous devons construire une balise capable de recevoir et d'émettre un signal dans toutes les directions. La réception du microcontrôleur ne pose pas de problème étant donné que l'antenne utilisée (une simple portion de fil conducteur) est omnidirectionnelle. Par contre, il est évident que la forme de la balise est fonction des émetteurs ultrasons utilisés et inversement et que la forme circulaire est la plus propice à une émission omnidirectionnelle. Cependant, étant donné les moyens dont nous disposons, il est plus simple pour nous d'opter pour une forme hexagonale.
Par ailleurs, le règlement nous impose un encombrement maximum correspondant à un cube de 80mm d'arête. Il est donc capital de prendre en compte cette contrainte lors de la conception de la balise. Une modélisation 3D (voir figure 14.6) de cette dernière a été effectuée afin de rendre compte des embombrements recpectifs des différents éléments constitutifs, notamment les émetteurs US et la pile de 9 volts. Ainsi, en considérant l'encombrement diamétral des émetteurs, on arrive à un hexagone régulier de 32 mm de côté. Afin de favoriser la modularité de ce système, l'élément en charge de la détection à l'intérieur du robot est une "boîte" facilement déplaçable à l'intérieur de celui-ci. Elle a pour rôle de renvoyer à la carte mère un signal 1 ou 0 selon que la balise se trouve ou pas dans la zone ou il faut détecter un évitement.

Le microcontrôlleur gérant le comportement de la détection de la balise est réglé de manière à envoyer un signal à la carte mère dès que la balise (située sur le robot adverse) est détectée à 55 cm de notre mât. Cette distance a été fixée en prenant en compte plusieurs paramètres. Le premier est la plus grande distance entre le mât et le bord de notre robot (la demi-diagonale du carré de base), 21 cm. Nous avons considéré la même distance du mât du robot adverseà son extrémité de manière à nous prémunir d'une machine qui viendrait nous heurter sur son



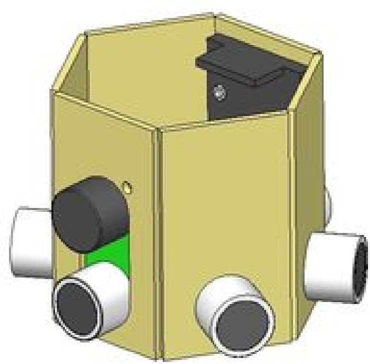


Fig. 14.6 – Modélisation 3D de la balise d'évitement placée sur le robot adverse

arrête. Enfin, étant donné l'inertie de notre machine et la criticité du test d'évitement lors de l'homologation, il nous a semblé important de prendre une marge de sécurité suffisante.

Choix des composants Afin d'assurer le rôle assigné à notre balise, il faut la doter d'un module ultrasonique de type SRF04 ou SRF05 (voir datasheet fournie en annexe C) qui, couplé à un microcontrôleur de type PIC, est en mesure d'envoyer la salve US dès réception du signal. Paencombrementserespectifsit subir une extension de ces émetteurs US afin d'obtenir une émin-versementdirectionnelle. Ceci est réalisé par simple mise en parallèle de ces émetteurs placés sur la périphérie de la balise avec celui situé sur le module SRF05. La forme de la balise étant fixée, il convient de choisir ces composants de manière à être en mesure d'assurer une couverture optimale de l'espace en signaux ultrasoniques. Ainsi, un simple raisonnement trigonométrique développé en annexe E nous permet de conclure qu'un émetteur de type 400ST120 de la firme Midas Components (datasheet fournie en annexe D) possèdant un cône d'émission de 85° suffit amplement pour notre application.
Par ailleurs, le chip en charge du traitement de l'information est ici un PIC 16F84A-04[2]. Celui-ci fonctionne grâce à une alimentation de 5 volts. Celle-ci lui est fournie par une pile de 9 volts classique que l'on connecte à un régulteur de tension de type 7805[3](voir figure 14.7). La photo présentée à la figure 14.8. La partie supérieure est recouverte de velcro©de manière à recevoir une balise de couleur indiquant la couleur pour laquelle le robot concourt lors des matches officiels. Tous ces éléments ont donc été agencés de manière judicieuse afin d'obtenir la balise la plus compacte possible. La schématique et la gravure (board) de la carte électronique de la balise est présenté à la figure 14.9.

[2]Le PIC 16F84 est un microcontroleur, c'est-à-dire un microprocesseur qui intègre des périphériques d'entrées-sorties. Le chiffre *16* signifie qu'il possède une mémoire qui gère des programmes de 14 bits. *F* signifie qu'il est équipé d'une mémoire de type *Flash* (mémoire réinscriptible non volatile). Le nombre *84* décrit le modèle de PIC (il en existe plus de 200 !). Les deux derniers chiffres indiquent la fréquence à laquelle est cadensée l'horloge interne. Notre application inclut donc une horloge cadensée à 4 Mhz. [3]

[3]Ce type de composant est courant en électronique. Les 2 premiers chiffres de la référence (78) signifient qu'il s'agit ici d'un régulateur de tension positive (79 pour le régulateur de tension négative). Les deux derniers chiffres (05) indiquent la tension de sortie. Nous avons donc bien +5V aux bornes de sortie. Il est à noter que, dans certaines applications où la dissipation d'énergie est importante dans le régulateur, il faut munir celui-ci d'un refroidisseur afin d'évacuer plus rapidement la chaleur et éviter ainsi la dégradation et destruction du composant

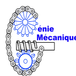


FIG. 14.7 – Régulateur de tension de type 7805

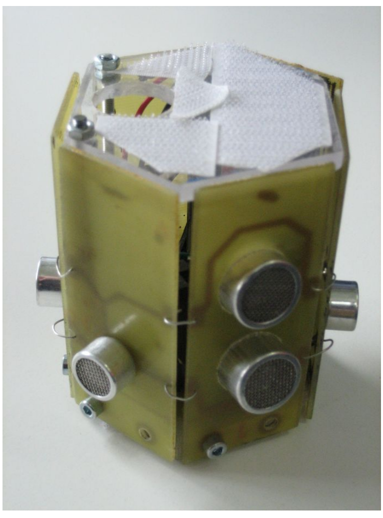
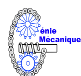
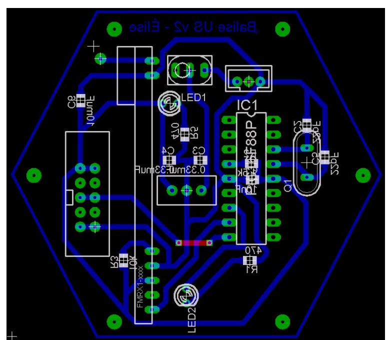


FIG. 14.8 – Balise placée sur le robot adverse

14.2.3 Le BUMPER

Lors de précedentes participations au concours (et notamment l'année passée), un BUMPER a été utilisé. Iréinscriptibleit d'un *"pare-chocs"* monté sur charnière et à la base de laquelle se situe un microswitch. À chaque contact du robot au niveau de ce bumpepossédancadencéeenclenche et on peut entamer une procédure d'évitement. Cependacadencée qu'il fonctionne correctement, il faut bien entendu que notre machine aille au contact du robot adverse. Or, cette année, le règlement impose, lors de l'homologation,régulateurvite un robot "en bois" à l'intérieur duquel serait placé un récipient rempli d'eau. Un évitement est un succès si ce récipient ne s'est pas renversé à l'issue de l'épreuve d'homologation. Par conséquent, connaissant l'inertie importante de notre robot, il y a peu de chances que celui-ci réussisse l'homologation avec ce simple équipement. Ainsi, étant donné que ce système ne demande pas un encombrement important (à peine le logement d'un microswitch), nous optons pour ce système en guise de système "de secours". En effet, en cas de défaillance du système de balise lors d'un match, on peut encore éviter le robot adverse. Dans le cas contraire, notre machine viendrait au contact de l'adversaire et, ne le détectant pas, il poursuivrait sont déplacement normal (sans évitements). Il serait alors perdu et déposerait les balles n'importe où sur la table, annihilant toutes nos chances de marquer le moindre point. On peut observer un cliché du bumper à la figure 14.10.





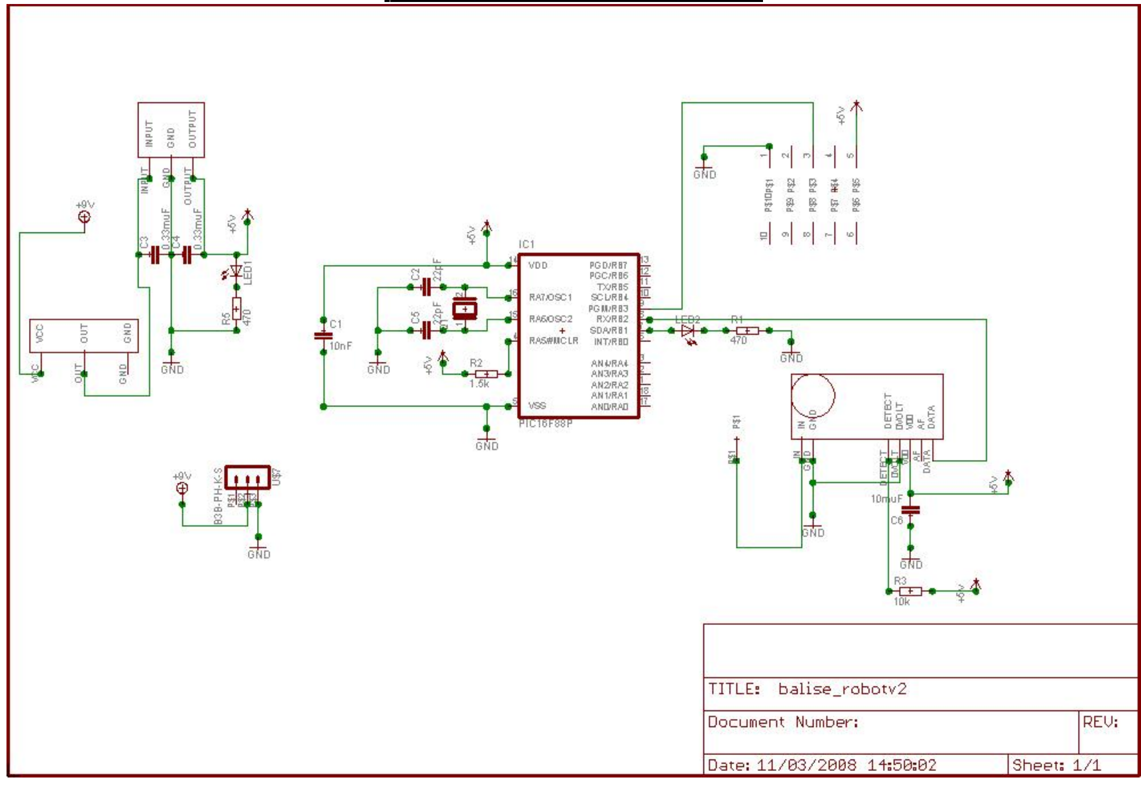


FIG. 14.9 – Board et schématique de la carte électronique incluse dans la balise

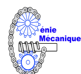


Fig. 14.10 – Bumper placé à l'avant du robot

Chapitre 15

Conclusions de la partie Électronique

La conclusion du travail effectué en électronique cette année dans le cadre du projet de robotique tient en plusieurs points.

Tout d'abord, nous voudrions insister sur la quantité impressionnante de matière et d'expérience emmagasinée grâce au travail effectué cette année. Nous, mécaniciens, nous sommes heurtés de plein fouet à la dure réalité de l'électronique et nous avons dû y faire face. Il nous a fallu apprendre l'ABC de l'électronique. Ainsi, notre premier semestre s'est essentiellement articulé autour de la confection d'exemples didactiques. De cette manière, nous avons pu appréhender d'une meilleure manière la logique de cette discipline et mettre nos connaissances acquises au service du groupe.

Par ailleurs, nous avons pu constater, à maintes reprises, tant au niveau national qu'international, l'efficacité de la balise d'évitement confectionnée en collaboration avec le service de Physique Générale. Ce module est d'ailleurs tout à fait réutilisable pour les années futures car, comme pour le reste du robot, tous les éléments ont été conçus de manière à maximiser la modularité.

En outre, il nous a semblé important de souligner un point concernant la philosophie du travail. En effet, lors de ce projet, nous avons été constamment confrontés au travail manuel. Cette expérience nous à permis de nous rendre compte de l'importance de la préparation et de la rigueur à mettre en oeuvre avant toute manipulation ou montage.

Enfin, je voudrais terminer en soulignant l'apport personnel que nous a fourni ce projet. Nous avons eu la chance de vivre au sein d'une équipe et rencontrer les joies et difficultés inhérentes à ce type d'organisation. Cette exprience fût très enrichissante et nous sera fort utile dans notre développement personnel et professionnel.

Troisième partie

Informatique

Chapitre 16

Introduction

16.1 Introduction

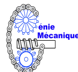
La programmation d'un robot est comme la programmation de tout programme : simple et complexe à la fois ! En effet, il ne s'agit ni plus ni moins que d'une succession de petites actions basiques ingénieusement combinées afin de créer des actions plus complexes en apparence. Chaque petite action prise séparément reste cependant simple en elle-même. La complexité se situe quant à elle au niveau de l'optimisation du regroupement de ces actions basiques dans le but de réduire les temps de calcul et par conséquent afin d'obtenir un meilleur fonctionnement.

Dans le cadre de ce projet, l'informatique permet avant toute chose la gestion du déplacement et l'actionnement des différents systèmes mécaniques embarqués, c'est-à-dire la mise en mouvement ou non de moteurs à des vitesses variables ou constantes en fournissant des tensions aux bornes des moteurs. L'informatique est également utilisé comme outil permettant de traiter les données entrantes (*inputs*) provenant des diverses cartes et contrôleurs électroniques et de faire réagir le système en conséquence en fournissant des données sortantes (*outputs*).

Afin de donner des bases solides à de tels traitements de données, un outil est primordial : un langage de programmation robuste. De nos jours, de nombreux langages ont fait leur apparition, mais en ce qui nous concerne, celui qui est le plus adapté à l'utilisation que l'on veut en faire, est le *C*. Dans un souci d'optimisation, nous avons même opté pour le *C++* en ce qui concerne certaines parties du code.

Tout au long de cette partie du travail, le langage que nous utiliserons sera donc principalement le *C*, voire le *C++*. Dans certaines parties du code, nous avons toutefois utilisé des *Instructions Orientées Octet*, ce qui nous permettait d'effectuer directement des modifications de registre, à savoir, dans notre cas, directement modifier des données rapatriées suite à des évènements extérieurs.

Dans le cas du robot de cette année, la carte utilisée est celle reprise des robots de feu l'équipe *Nucléo*. Cette carte, grâce à sa puce *Motorola*, permet une gestion totale du robot de part sa puissance de calcul. Un code avait été développé, que ce soit pour le contrôle des moteurs de propulsion et leur asservissement, pour le contrôle des autres moteurs à courant continu, des moteurs pas-à-pas ou des Servo-Moteur, ou encore la gestion des entrées et sorties. Dans un premier temps, il a été nécessaire que nous apprenions les subtilités du codage en *C*



afin de comprendre ce code. Il a ensuite été nécessaire que nous nous familiarisions avec celui-ci pour en maîtriser son fonctionnement. Enfin, nous avons pu adapter des parties du code à notre propre utilisation, voire totalement développer des parties entières de code.

Dans cette partie, nous allons décrire les différents aspects de la programmation en robotique nécessaires à la compréhension de notre code. Nous décrirons également les différents codes que nous avons étudiés et développés. Nous avons tout d'abord étudié le code implémenté à l'origine sur la carte, à savoir le code *Nucléo*. Ce code, bien qu'ayant déjà fait ses preuves au cours de plusieurs coupes de robotique, présente quelques défaillances. Un second code, adaptés de la technologie développée par *Microb Technology*
et plus performant, a alors été étudié. Puisque ce code a été prévu à l'origine pour des *ATMéga*, après avoir adapté le code pour le contrôlleur *Motorola*, nous avons pu essayer une adaptation et une utilisation de celui-ci. Malheureusement, la charge de travail restant pour cette adaptation et l'échéance avant la coupe étant trop courte, nous avons du nous résoudre à reprendre l'ancien code de l'équipe *Nucléo*.

16.2 Généralités

L'architecture d'un programme quel qu'il soit doit rester bien organisée afin de garder une certaine logique dans la succession des actions mais également afin d'optimiser les temps de calcul. L'architecture globale de tout programme de traitement de données comporte quatre parties principales :

- l'initialisation ;
- la lecture des entrées via les différents capteurs ;
- le traitement de ces données et le calcul des sorties à appliquer en fonction de celles-ci ;
- l'application des sorties sur les actionneurs ou les moteurs.

Les années précédentes, la programmation se faisait sur *ATMega*. Cela nécessitait la présence de plusieurs cartes électroniques supportant les différentes puces *ATMega*, et par conséquent, autant de codes différents. Cette année, la programmation se faisant sur un processeur plus puissant, à savoir un *Motorola*, il n'est plus nécessaire d'écrire plusieurs codes, un seul et unique suffit. Cependant, la même structure globale reste valable. Nous aurons ainsi trois catégories au sein du code :

- stratégie ;
- déplacement ;
- capteurs et actionneurs.

Il va de soi que ces trois parties ne sont pas indépendantes mais vont interagir entre elles. Nous pouvons même dire qu'elles sont imbriquées.

Chapitre 17

Programmation

17.1 Introduction

Le code Nucléo comme le code Microb Technology sont composés de plusieurs couches : la couche de *drivers*, la couche *middleware* et enfin l'*API* (pour *Application Programming Interface*).

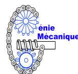
Tout en-dessous, on retrouve la couche *drivers*. C'est elle qui joue le rôle d'intermédiaire entre le reste du programme et les différents périphériques (moteurs, odomètres, servomoteurs, bus de communication I^2C, gestion du port série...).

Vient ensuite la couche *Middleware*. Celle-ci sert à faire le lien entre les *drivers* et l'*API*. Comparé à la couche *drivers*, elle intègre des fonctions de plus haut niveau. C'est notamment au sein du *Middleware* qu'on va retrouver les différents *managers* pour l'odométrie, les servomoteurs, les *sensors*... Les fonctions faisant partie du *middleware* ne sont en général pas utilisées directement pour le codage de la stratégie, mais sont néanmoins très utiles à la bonne exécution de celle-ci : par exemple, l'*OdoManager* va avoir en charge de définir quelle puissance fournir aux moteurs de propulsion tout en surveillant le retour des odomètres de telle sorte que le robot ne dépasse pas sa consigne en position.

Enfin, l'*Application Programming Interface* est la couche qui vient au-dessus des deux autres. Il s'agit en fait d'une bibliothèque de fonctions facilement exploitables pour réaliser la stratégie. Les fonctions de l'*API* sont définies de façon non ambiguë, elles ne prennent en général que quelques paramètres simples. Ce n'est pas pour autant que les actions découlant de l'appel à ces fonctions sont simples : elles vont en général appeler diverses fonctions du *middleware* qui elles-mêmes vont appeler différents *drivers* pour s'exécuter correctement. Nous sentons bien ici tout l'intérêt de l'*API* : fournir un ensemble de fonctions relativement simples pour programmer la stratégie, sans devoir se soucier de tout ce qu'il se passe derrière l'appel à ces fonctions.

17.2 Code Microb Technology

Comme nous l'avions déjà fait remarquer au chapitre 16, nous avons étudié un code développé par l'équipe *Microb Technology*. Cette équipe, composée à l'origine d'anciens des équipes de la ville d'Avray, a fait ses preuves durant de nombreuses années en coupes de France et



d'Europe. Nous avons donc opté pour une évolution du code qui avait été mis au point par feu l'équipe *Nucléo*, sur base du code de cette équipe française. Il est évident que dans un premier temps une adaptation fut nécessaire puisque nous utilisons un processur *Motorola* tandis que le code était prévu à l'origine pour des micro-contrôleurs *ATMéga*.

Le but en passant à ce nouveau code était d'améliorer les déplacements que ce soit aussi bien en temps qu'en précision ! Dans ce chapitre nous allons présenter les points forts du code mis au point par *Microb Technology*. Nous nous attarderons donc plus précisément sur la gestion des déplacements plutôt que sur le reste du code, plus classique.

17.2.1 Spécifications du code

Les points principaux de l'asservissement du code *Microb Technology* sont les suivants :

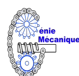
- Asservissement en position ;
- Asservissement différentiel des roues, il y a un contrôle sur l'angle et sur la distance plutôt que de contrôler chacune des roues indépendamment (cf. chapitre 17.2.2) ;
- Génération de trajectoire à profil de vitesse trapézoïdal avec anticipation sur le freinage afin de stopper en bonne position ;
- Localisation de la plate-forme selon trois coordonnées : x , y , a ;
- Deux systèmes de mesure des roues codeuses (roues motorisées et codeurs indépendants) ;
- Génération de trajectoire par points de passage et évitement d'obstacles.

En outre, le code n'a pas été implémenté tel une grande boucle algorithmique dont le temps de boucle est fonction des tâches les plus longues. En effet, ce code a été conçu pour fonctionner sous interruption grâce à son module *scheduler*. Nous reviendrons sur cela plus en détail à la section 17.2.5.

17.2.2 Asservissement polaire

Comme nous l'avons déjà signalé à la section 17.2.1, l'asservissement de ce code se fait sur le positionnement en théta-alpha. En effet, l'asservissement ne se fait pas comme dans la plupart des robots avec un PID sur chacun des moteurs de propulsion, mais avec un PID sur la différence de position des roues codeuses (angle alpha) et un PID sur la distance parcourue par la moyenne des deux roues (distance théta).

Ce type d'asservissement trouve tout son avantage dans le recalage sur une bordure quelconque : il est seulement nécessaire d'asservir la distance et non l'angle. Cela permet dès lors à la plate-forme de n'opposer aucune résistance en angle lors du recalage et donc une mise en mouvement plus fluide. Il apparaît dès lors plus logique, en ce qui concerne un asservissement de plate-forme robotique, d'utilisr un positionnement en théta-alpha par rapport aux codes dans lesquels on asservit les deux roues de propulsion indépendamment l'une de l'autre. Par ailleurs, la génération de trajectoire se verra également grandement facilitée du fait du choix de cette méthode.



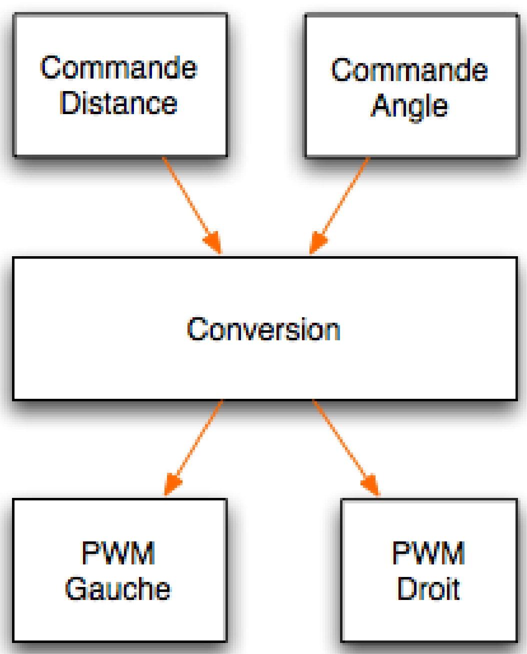


FIG. 17.1 – Schéma pour l'envoi des PWM en distance et en angle vers les moteurs

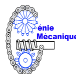
Comme nous utilisons ici un asservissement polaire, nous ne pouvons donc pas directement utiliser les données des codeurs ou directement appliquer des PWM à chacun des moteurs. Une conversion est nécessaire ! La conversion à partir des données des codeurs se fait alors simplement comme suit :

```
void rs_get_polar_from_wheels(struct rs_polar * p_dst, struct rs_wheels * w_src)
{
p_dst->distance = (w_src->right + w_src->left)/2;
p_dst->angle    = (w_src->right - w_src->left)/2;
}
```

Pour envoyer les commandes PWM aux moteurs, il suffit tout simplement d'utiliser des fonctions inverses de conversion. Les PWM d'angle et de distance n'ont bien évidemment aucune existence physique mais leur interface est la même que celle de l'interface des vraies PWM. Cela permet de les utiliser comme une fonction à part entière lors du process. Le module *robot_system* effectue cette opération simple de conversion pour l'envoi des PWM d'angle et de distance (cf figure 17.1) :

```
angle_set();
distance_set();
```

En ce qui concerne la lecture des codeurs incrémentaux, l'opération est quelque peu plus complexe puisque le code a été prévu, à la base, pour pouvoir tirer parti des données de quatre codeurs : des codeurs intégrés aux moteurs et des codeurs indépendants (deux gauches et deux



droits). L'idée est de pouvoir utiliser indépendamment les codeurs moteurs ou les codeurs indépendants et de pouvoir passer à tout moment de l'un à l'autre.

Le module *robot_system* reprend et convertit l'information des quatre codeurs afin de générer les valeurs des codeurs virtuels en angle et en distance (cf figure 17.2) :

```
angle_get();
distance_get();
```

Une fonctionnalité supplémentaire est intégrée au module pour prendre en compte des coefficients correcteurs afin de compenser des défauts mécaniques. Un coefficient multiplicateur appliqué à l'angle est également intégré au sein du module afin de pouvoir passer des codeurs indépendants ou codeurs moteurs. En effet, les deux types de codeurs ne retournent pas forcément le même nombre d'impulsions par tour. Le nombre d'impulsions renvoyé par le codeur par unité de longueur est définie dans le fichier *config.h* :

```
#define DIST_IMP_CM 433.25 * 2
```

Les différences d'impulsions entre les deux codeurs sont alors appréciées directement dans le fichier principal de stratégie *strat.cpp* :

```
rs_set_left_ext_encoder(&MyRobot.rs, 0, NULL, 9.975530179);
rs_set_right_ext_encoder(&MyRobot.rs, 0, NULL, 10.00000);

rs_set_left_mot_encoder (&MyRobot.rs, 0, NULL, 0.4212);
rs_set_right_mot_encoder(&MyRobot.rs, 0, NULL, 0.4291);
```

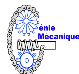
17.2.3 Schématique de l'asservissement

Au sein du code de l'asservissement, nous pouvons trouver plusieurs modules spécifiques ayant chacun un but particulier. On distingue quatre grand groupes fonctionnels (cf figure 17.4) :

- Régulation proprement dite ;
- Calcul de la position courante en x, y, a sur la table ;
- Génération des trajectoires via une interface en haut niveau ;
- Surveillance du bon fonctionnement des déplacements et détection des éventuelles erreurs (blocage, patinage...).

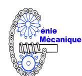
La figure 17.4 représente le schéma de l'architecture du programme concernant l'asservissement. On peut facilement comprendre grâce à ce schéma le fonctionnement de la régulation. Nous nous proposons de décrire son fonctionnement ici.

Afin que la plate-forme se déplace en un point x, y, a sur la table, le programme principal envoie la commande au module *trajectory*. Celui-ci va alors calculer des consignes de position,



Distance
Angle
Selection
En fonction des différents paramètres, les codeurs moteurs ou externes sont privilégiés et retournés.
Distance
Angle
Distance
Angle
A ce stade, dans un monde parfait sans frotements ni patinage, les couples (distances,angle) moteurs et externes sont égaux
Gain
Correction de l'angle des codeurs externes
conversion
conversion
Pour chaque codeur, on détermine leurs somme (distance) et leurs différence (angle)
Gain
Gain
Gain
Gain
Correction des erreurs sur le diamètre des roues
Codeur moteur G
Codeur moteur D
Codeur externe G
Codeur externe D
Codeurs

FIG. 17.2 – Schéma pour la lecture des codeurs et la conversion en distance et en angle



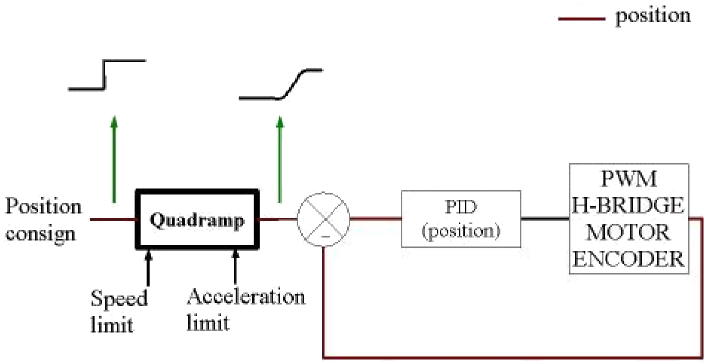


FIG. 17.3 – Schéma automatique en boucle fermée avec contrôleur PID et filtrage en rampe quadratique sur la position

qui vont être envoyées aux asservissements d'angle et de distance. Cette consigne est périodiquement réactualisée en fonction de la position courante de la plate-forme avec une période de 50 ms (période fonction du processeur *Motorola*).

C'est le module *robot_system* qui, par exécution sur interruption (cf chapitre 17.2.5) avec une période de 5 à 10 ms, lit périodiquement la valeur réelle des codeurs, et met à jour les valeurs virtuelles des capteurs d'angle et de distance. Le module *control_system_manager* envoie alors seulement la nouvelle position à l'entrée du filtre *quadramp*, et ce pour chaque asservissement. Ce dernier module permet de filtrer la consigne en limitant les deux premières dérivées du signal.

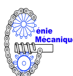
L'erreur d'angle et de distance est ensuite calculée par le module *control_system_manager* par comparaison avec les valeurs des codeurs virtuels. Ces erreurs sont alors envoyées aux correcteurs PID en angle et en distance qui renvoie des valeurs filtrées au module *robot_system* (cf figure 17.3). Ce dernier va déterminer les PWM's a envoyer aux roues et les communiquer au module *PWM* qui va modifier les signaux générés par le processeur.

Un évènement supplémentaire appelé *position_manager* est également exécuté périodiquement (toutes les 10 ms) afin de recalculer la position en x, y, a de la plate-forme sur l'aire de jeu en fonction des codeurs et de la position à la période précédente. Cette position alors déterminée sera alors utilisée lors de la prochaine exécution du module *trajectory*. L'évènement *blocking_detection_manager* est exécuté de la même manière mais avec une période de seulement 100ms. Ce dernier module est placé juste après le module *position_manager* afin de permettre un recalcul de la position si un blocage est détecté.

L'ensemble des fichiers correspondant au schéma décrit ici sont répertoriés à la section 17.2.4.

17.2.4 Description du code

Dans cette section, nous allons simplement faire un listing des fichiers prinicipaux afin de se rendre compte des fonctionnalités qu'offre le programme.



Architecture du programme

statut (fin de trajectoire)

main

points de passage (x,y)

trajectory manager

obstacle avoidance

destination finale (x,y)

gestion de la trajectoire
periode = 100ms

consigne de position en angle, distance

Position x,y,a

Blocage détecté

quadramp

consigne de position, angle et distance

consigne de position filtrée (rampe de vitesse)

Erreurs

blocking detection manager

control system manager

détection de blocage
periode = 100ms

Erreurs d'angle et distance

Erreurs filtrées par le PID

Sortie vers les PWM virtuels

Codeurs virtuels et réels

pid

position manager

codeurs virtuels angle et distance

calcul de position
periode = 10ms

robot system

Codeurs virtuels angle et distance

asservissement
periode = 10ms

PWM réels

valeur des 4 codeurs

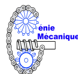
encoders

pwm

mise à jour de odomètres
periode = 128us

Moteurs, odomètres

Fig. 17.4 – Schéma de l'architecture du programme concernant l'asservissement



Au sein du *control_system*, nous pouvons trouver les modules suivants :

- Le *control_system_manager* est le module principal de l'asservissement, c'est lui qui gère l'appel de tous les autres modules ;
- Le module *pid* est un filtre proportionnel - intégral - dérivé appliqué sur la position ;
- Le filtre *ramp* permet un filtrage de la dérivée première à l'entrée de l'asservissement. Il n'est pas intégré dans notre contrôle mais pourrait l'être dans une régulation en vitesse ;
- Le filtre *quadramp* permet de limiter les dérivées première et seconde à l'entrée de l'asservissement. Celui-ci permet de générer les profils de vitesse trapézoïdaux.

Le module *trajectory* comporte deux fichiers principaux :

- Le *trajectory_manager* qui permet la définition des trajectoires en haut niveau dans le code ;
- Le fichier emphobstacle_avoidance qui s'occupe de générer les trajectoires voulues avec points de passage tout en gérant les obstacles.

Concernant les différents blocages et erreurs, un simple fichier s'occupe de tout cela. Il s'agit du fichier *blocking_detection_manager*.

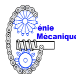
Enfin, quelques modules supplémentaires et non spécifiques à l'asservissement proprement dit sont également nécessaires au déplacement de la plate-forme :

- Le module *scheduler* s'occupe de la planification de l'appel des fonctions et de la gestion des priorités ;
- Le fichier *vect2* est spécifique aux opérations sur les vecteurs en 2D ;
- emphencoder_microb s'occupe de la gestion des codeurs (moteurs et indépendants) ;
- Le fichier emphPWM est dédicacé au calcul et à l'envoi des commandes vers les moteurs de propulsion.

Il y a bien évidemment d'autres modules supplémentaires qui ne sont pas affectés au déplacement. Ceux-ci s'occupent principalement de la gestion des I/O. Nous ne les décrivons pas ici puisque ceux-ci sont les mêmes que ceux qui seront utilisés dans le code *Nucléo*.

Le fichier *strat.cpp* est utilisé comme interface principale dans ce code. C'est lui qui permet l'appel des différentes fonctions au sein du code. Parmi celles-ci, nous pouvons trouver l'implémentation des paramètres des PID en position et des filtres quadramp, ainsi que la définition des vitesses :

```
pid_set_gains(&MyRobot.pid_d, 800, 15, 20000);
pid_set_maximums(&MyRobot.pid_d, 0, 50000, 4095);
pid_set_out_shift(&MyRobot.pid_d, 10);
pid_set_derivate_filter(&MyRobot.pid_d, 4);

quadramp_init(&MyRobot.qr_d);
quadramp_set_1st_order_vars(&MyRobot.qr_d, 50, 50);
quadramp_set_2nd_order_vars(&MyRobot.qr_d, 20, 20);
```



```
pid_set_gains(&MyRobot.pid_a, 800, 15, 30000);
pid_set_maximums(&MyRobot.pid_a, 0, 50000, 2047);
pid_set_out_shift(&MyRobot.pid_a, 10);
pid_set_derivate_filter(&MyRobot.pid_a, 4);

quadramp_init(&MyRobot.qr_a);
quadramp_set_1st_order_vars(&MyRobot.qr_a, 50, 50);
quadramp_set_2nd_order_vars(&MyRobot.qr_a, 20, 20);

trajectory_set_speed(&MyRobot.traj, 75, 50);
```

C'est également ce fichier qui s'occupe de l'initialisation de l'ensemble des variables en début de match. Les différentes machines d'état sont soit implémentées directement dans ce module, soit appelées par celui-ci via la machine d'état principale codée dans la fonction principale du programme :

```
void strat_main(void)
```

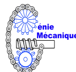
En ce qui concerne les paramètres invariants de la plate-forme (configuration des roues, positions des ServoMoteurs...) ils sont implémentés dans le fichier *config.h* :

```
#define MATCH_TIME 90000

#define MOT_TRACK_CM 13.2
#define EXT_TRACK_CM 26.2

#define DIST_IMP_CM 433.25 * 2

#define CServo1Id           0
#define CServo1Pos1         230
#define CServo1Pos2         125
```

17.2.5 Fonctionnement du code sous interruption

Comme nous l'avons déjà fait remarquer plus haut, un avantage de ce code est le temps d'exécution. En effet, plutôt que de faire des boucles d'attentes constamment actives qui augmentent considérablement le temps d'exécution du programme, la solution choisie ici est de faire tourner l'asservissement en tâche de fond. C'est le module *scheduler* qui s'occupe de cette tâche.

L'idée de base est de pouvoir planifier l'appel des fonctions de manière périodique, ou tout simplement unique. Le module *scheduler*, qui est alors un gestionnaire d'appel des fonctions sous interruption, permet de gérer les priorités des évènements tout en empêchant d'appeler de manière récursive une même fonction si cela n'est pas utile.



La définition propre de l'interruption est le fait d'interrompre temporairement et de manière périodique l'exécution normale d'une tâche afin d'en exécuter une autre, alors appelée routine ou interruption.

Lors d'une interruption, l'état du système est sauvegardé en mémoire par le processeur et celui-ci saute à la routine d'interruption la plus prioritaire. Lorsque cette dernière est traitée, la routine se termine par une instruction de retour d'interruption qui restaure le système à l'endroit où il avait été interrompu et relance alors le processeur. C'est la base du multiprocessing.

Dans notre cas, puisque nous sommes dans un système assimilé à un système multi-tâches, l'interruption est utilisée pour commuter de nombreuses tâches. Une interruption périodique est alors déclenchée par une horloge interne et c'est l'ordonnanceur qui va alors déterminer quel est la tâche suivante qui sera commutée selon l'ordre des priorités. Une interruption de priorité supérieure est prise en compte lors du traitement d'une autre interruption, mais une interruption de priorité inférieure est toujours mise en attente.

Le fait de passer sous interruption rend alors l'asservissement autonome. En effet, le calcul des déplacements une fois terminé, ce module rend la main au programme principal pour tout autre calcul (gestion des I/O par exemple).

Il est toutefois nécessaire de définir les priorités dans le code pour que celui-ci sache dans quel ordre il doit effectuer les tâches, une fois la précédente terminée. La définition des priorités est ainsi faite dans l'ordonnanceur qui est implémenté dans le fichier *config.h*. Nous avons dès lors comme listing des interruptions :

```
#define ENCODERS_PRIO      200
#define LED_PRIO           170
#define TIME_PRIO          160
#define CS_PRIO            150
#define SENSORS_PRIO       100
#define I2C_POLL_PRIO       20
```

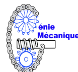
Les priorités sont définies de manière croissante entre 0 et 255.

17.3 Code Nucléo

Dans cette partie, nous n'allons pas reparler de l'architecture du code (cf. section 17.1 pour plus de détails) mais bien ici de la façon dont il s'organise : comment les fonctions sont appelées entre elles, dans quels cas elles sont appelées...

Comme tout programme écrit en *C*, l'exécution de celui-ci commence par la fonction *main()* :

```
int main(void)
{
  /************************
```



```
    // phase d'initialisation
    // (non reprise ici)
    ************************/

    do
    {
      ManageRobotState();
      ManageRobotCommand();
      Activate();
    } while (1);
}
```

Comme nous pouvons le voir, après une phase d'initialisation[1], le programme entre dans une boucle sans fin (délimitée par *do* et *while(1)* ; la condition *1* étant toujours vraie, on ne sort donc jamais de la boucle). Trois fonctions sont appelées successivement au sein de cette boucle :

- *ManageRobotState()*
- *ManageRobotCommand()*
- *Activate()*

L'existence de ces trois fonctions répond à un souci de clarté du code. En effet, elles ont toutes un rôle complémentaire. Ainsi *ManageRobotState()* a pour but de définir dans quel état est le robot et de faire les transitions entre les différents états existant. *ManageRobotCommand()* effectue certaines commandes bien précises et donne l'ordre d'en faire d'autres plus générales. Ces commandes plus générales seront exécutées par le biais de *Activate()*, en plus d'autres tâches de fond : lecture des *I/O*, supervision du déplacement, mise en position des servomoteurs...

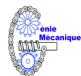
17.3.1 Les machines d'état

Comme dit précédemment, les machines d'état et les transitions entre celles-ci sont définies au sein de la fonction *ManageRobotState()*. L'état actuel est enregistré au sein de la variable *RobotState*[2]. Les différentes valeurs que peut prendre la variable *RobotState* sont les suivantes :

- RobotStopped ;
- RobotGoToColorDistributor ;
- RobotPickUpBalls ;
- RobotGoToFrozenContainer ;
- RobotCatapult.

[1]Les fonctions appelées lors de la phase d'initialisation ne sont appelées qu'une seule fois, au moment où l'on met le robot sous tension.

[2]Comme beaucoup de variables utilisées dans le code, la variable *RobotState* est une énumération, ce qui permet d'utiliser un nom complet pour chaque état lorsque nous programmons tandis qu'à la compilation le nom complet sera remplacé par une valeur numérique, assurant ainsi une économie de mémoire allouée au stockage de la variable *RobotState*. Pour plus de détails sur les énumérations, vous pouvez consultez la section 21.2.2



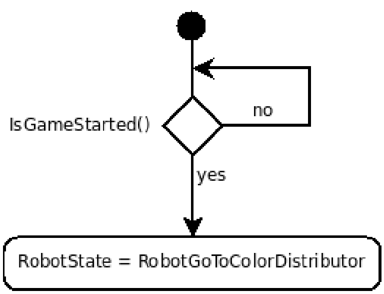


FIG. 17.5 – RobotState = RobotStopped

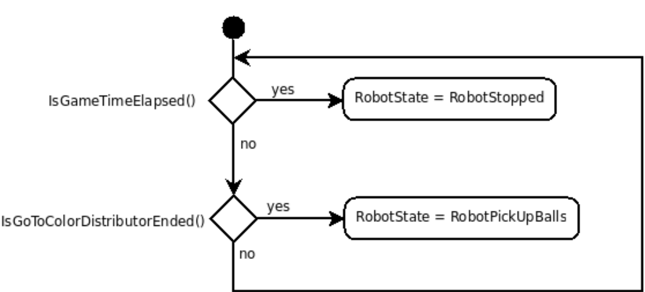


FIG. 17.6 – RobotState = RobotGoToColorDistributor

En fonction de la valeur de *RobotState*, nous pouvons dessiner les machines d'état reprises sur les figures 17.5 à 17.9.

En langage *C*, tout ceci se traduira au moyen d'un *switch* et de *cases* :

```
switch(RobotState)
  {
    case RobotStopped :
         if (IsGameStarted())
         {
    RobotState = RobotGoToColorDistributor;
         }
         break;

    case RobotGoToColorDistributor :
      if (IsGameTimeElapsed()) {
     RobotState = RobotStopped;
```

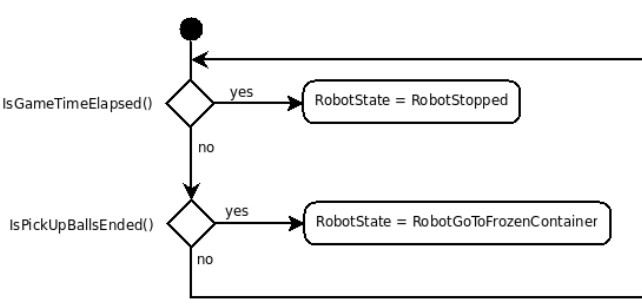
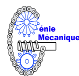


FIG. 17.7 – RobotState = RobotPickUpBalls



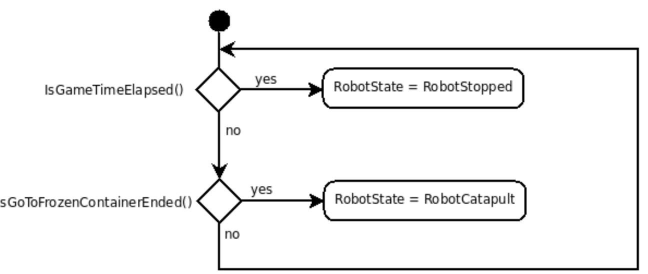


FIG. 17.8 – RobotState = RobotGoToFrozenContainer

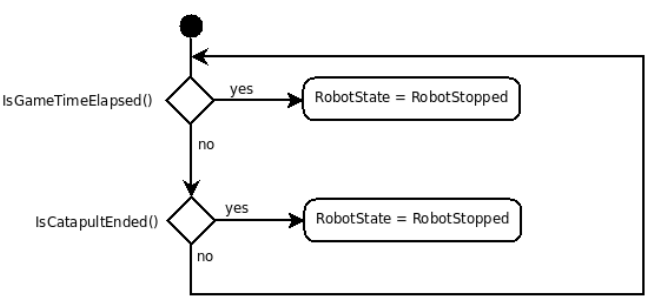
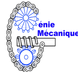


FIG. 17.9 – RobotState = RobotCatapult

```
    }
    else if (IsGoToColorDistributorEnded()) {
   RobotState = RobotPickUpBalls;
   RestartStopReason();
  }
  break;

case RobotPickUpBalls :
     if (IsGameTimeElapsed()) {
        RobotState = RobotStopped;
     }
     else if (IsPickUpBallsEnded()) {
          RobotState = RobotGoToFrozenContainer;
          RestartStopReason();
     }
     break;

case RobotGoToFrozenContainer :
     if (IsGameTimeElapsed()) {
        RobotState = RobotStopped;
     }
     else if (IsGoToFrozenContainerEnded()) {
          RobotState = RobotCatapult;
          RestartStopReason();
          PerformBackCalibration();
     }
     break;

case RobotCatapult :
```



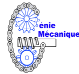
```
            if (IsGameTimeElapsed()) {
               RobotState = RobotStopped;
            }
            else if (IsCatapultEnded()) {
                 RobotState = RobotStopped;
            }
            break;
     }
```

De la même manière, nous retrouvons deux autres *switch* pour les états des servomoteurs (correspondants aux positions ouvertes et fermées du bras coté avalement et de la porte coté éjection).

17.3.2 Les commandes associées aux machines d'état

Les commandes associées aux différentes machines d'état sont appelées au moyen de la fonction *ManageRobotCommand()*. Comme dans la fonction *ManageRobotState()*, on retrouve un *switch(RobotState)*. Chaque *case* correspond à un état défini ci-dessus. Pour chaque *case* sont associés les différentes fonctions à appeler pour cet état.

```
switch(RobotState)
  {
    case RobotStopped :
         StartStop_MotorEC(0x0);
         StopDCMotors1();
         StopDCMotors2();
        Stop();
         break;

    case RobotGoToColorDistributor :
         //on commence le ramassage
         if (!IsRobotInit()){
          RobotInit();
         }
         ManageRobotGoToColorDistributorState();
         ManageRobotGoToColorDistributorCommand();
         break;

    case RobotPickUpBalls :
         ManageRobotPickUpBallsState();
         ManageRobotPickUpBallsCommand();
         break;

    case RobotGoToFrozenContainer :
         ManageRobotGoToFrozCState();
         ManageRobotGoToFrozCCommand();
         break;
```



```
        case RobotCatapult :
             ManageRobotCatapultState();
             ManageRobotCatapultCommand();
             break;
    }
```

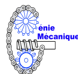
Mis à part pour l'état *RobotStopped*, nous constatons que les fonctions appelées sont souvent de la forme *ManageRobot*NomDeLEtat*State()* et *ManageRobot*NomDeLEtat*Command()*. En effet, pour chaque état défini précédemment, on va retrouver des sous-machines d'état. Nous reviendrons là-dessus au chapitre 20, lorsque nous parlerons de la stratégie, de la trajectoire de déplacement et de l'interaction entre les deux.

17.3.3 La fonction *Activate()*

La fonction *Activate()* est définie comme suit :

```
void Activate() {
  ReadSensorsStates();
  ManActivate();
  ActivateServo(ArmServo);
  ActivateServo(FrontServo);
  if (IsGameStarted()) {
      // La fonction IsGameStarted() nous renvoie un booléen:
      // --> "true" si on détecte un front descendant sur le switch
      //     de démarrage, càd quand on tire sur la ficelle de démarrage
      // --> "false" dans les autres cas
      StartTimer(GameTimer,CGameDuration);
  } else if (!GameEnded) {
   GameEnded = IsTimerElapsed(GameTimer);
  }

  if (FirstCycle) {
    if (DetectLowState(StrategySwitch,true)) {
      StrategyConfig = 0;
      // On est rouge
    }else {
      StrategyConfig = 1;
  // On est bleu
    }
    FirstCycle = false;
  }
}
```

A chaque fois qu'elle sera appelée, c'est-à-dire à chaque cycle, la fonction *Activate()* va donc lire l'état des *inputs* (*ReadSensorsStates()*) et activer différents processus : *ManActivate()* appelle le manager d'odométrie, et *ActivateServo()* va activer la mise en mouvement des servomoteurs s'ils ne sont pas dans la position requise.



Nous remarquons que c'est aussi au sein de la fonction *Activate()* qu'est initialisé le chronomètre général (90 secondes) au moment où on tire sur la ficelle de démarrage. C'est aussi ici qu'au démarrage on va enregistrer la valeur du switch de stratégie (rouge ou bleu).

17.3.4 Remarque

D'une manière générale, l'organisation du code sous forme de trois modules (la gestion des états, la gestion des actions associées à un état et l'exécution des actions) est une structure qu'on retrouvera régulièrement dans le code *Nucléo*. Par exemple, on va la retrouver au sein de la gestion de l'odométrie : la fonction *ManActivate()* va ainsi commencer par l'appel de deux fonctions tierces, *ManageOdoStates()* et *ManageOdoCommands()*, avant d'effectuer la régulation et l'odométrie proprement dits.

17.3.5 La gestion du déplacement

La gestion du déplacement au sein du code *Nucléo* passe par un ensemble de fonctions regroupées au sein de trois fichiers : *OdoManager.cpp*, *Regulation.cpp* et *CommandsFifo.cpp*. D'un point de vu général, ces fonctions appartiennent à la couche *middleware*. Elles ne seront pas appelées directement lors de l'implémentation de la stratégie. Nous passerons par des fonctions intermédiaires reprises dans l'*API* (*ReachPosition()*, *Rotate()*, *Move()*...)

Les fonctions reprises au sein du fichier *CommandsFifo.cpp* servent à gérer le buffer contenant les différentes instructions de déplacement à effectuer. La structure utilisée pour le buffer est de type *FIFO* (*First In, First Out*, une file d'attente).

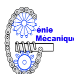
Le déplacement en cours de réalisation sera lui pris en charge par le biais des fonctions de *Regulation.cpp*. Dans le code *Nucléo*, la régulation porte sur deux points : la régulation en vitesse au moyen d'un PID, pour que le déplacement se fasse à vitesse constante, et la régulation en position au moyen d'un PI. Nous reviendrons plus en détail sur la régulation au chapitre 18.

Enfin, les fonctions du fichier *OdoManager.cpp* ont pour rôle de chapeauter tout ce qui est relatif au déplacement, y compris ce qu'on retrouve dans *CommandsFifo.cpp* et *Regulation.cpp*.

Quand on appelera une fonction de l'*API* (*ReachPosition()*, *Rotate()*, *Move()*...), l'instruction sera transmise au manager d'odométrie, qui se chargera de la rajouter à la suite du buffer. C'est aussi le manager d'odométrie qui au moment opportun (typiquement, quand le déplacement précédent est terminé) se chargera de transférer la première fonction du buffer vers les modules de régulation afin que le déplacement soit effectué.

17.4 Choix du code

Au cours de cette année, nous avons travaillé avec plusieurs plate-formes différentes. Nous avons d'abord utilisé une plate-forme de test équipée de moteurs Maxon 12V, issus du matériel



de l'équipe *Nucléo*. Nous sommes ensuite passés sur une plate-forme de test équipée des moteurs Maxon 24V prévus pour le robot final. Et enfin, nous avons travaillé avec le robot final. En parallèle, les *Kadroïds* ont développé leur robot, qui est lui équipé de moteurs Maxon 12V, mais d'un modèle différent des anciens moteurs *Nucléo*.

Les premiers tests réalisés sur la plate-forme 12V semblaient prometteurs. Mais au moment de passer sur plate-forme 24V, nous nous sommes apperçu que celle-ci ne se déplaçait pas vitesse constante, mais oscillait de manière critique autour d'une consigne de vitesse. Nous avons cherché pendant plusieurs semaines après une explication, sans grand résultat. C'est à ce moment là que nous avons pu tester le code *Microb Technology* sur le robot des *Kadroïds*. Nous avons observé que le phénomène apparaissait aussi sur leur robot, mais que la fréquence d'oscillation est moindre que sur notre plate-forme 24V.

Nous en sommes donc arrivés à la conclusion suivante : le code *Microb Technology* présente une défaillance au niveau de l'asservissement. Il ne comporte en effet pas d'asservissement en vitesse, mais seulement de l'asservissement en position[3]. Il génère même une oscillation de la consigne de vitesse, oscillation dont la fréquence est proportionnelle au nombre d'impulsions par millimètre que les codeurs renvoient, ce nombre étant relativement faible pour les moteurs Maxon 12V *Nucléo* (de l'ordre de 7 impulsions/mm), légèrement supérieur pour les moteurs *Kadroïds* (de l'ordre de 19 impulsions/mm) et largement supérieur pour les moteurs Maxon 24V que nous utilisons (de l'ordre de 88 impulsions/mm) ; ce qui explique que l'oscillation n'était pratiquement pas visible avec les moteurs *Nucléo* et très importante avec nos moteurs 24V.

Devant le peu de temps restant pour développer un nouvel asservissement en vitesse, nous avons préféré réutiliser le code *Nucléo*. Bien que moins réactif, car ne disposant pas de scheduler, ce code présente néanmoins des fonctionnalités suffisantes pour notre robot, moyennant quelques adaptations.

[3]Nous supposons que l'équipe *Microb Technology* utilise une carte électronique supplémentaire pour asservir les moteurs en vitesse ou alors qu'ils disposent de moteurs autorégulés, comme le sont par exemple les moteurs *Crouzet*.

Chapitre 18

Asservissement & Régulation

Il est nécessaire que le robot se déplace de manière intelligente, en connaissant en permanence l'emplacement sur la table auquel il se trouve ainsi que la direction dans laquelle il est dirigé. Il est évident qu'il doit donc être capable de récupérer des données de positionnement afin d'y parvenir. Dans la même optique, le robot doit pouvoir réguler sa vitesse dans le but de s'arrêter en un point précis de la table auquel il devait se rendre.

Dans un premier temps, nous nous proposons de décrire les outils qui seront utilisés afin de permettre le déplacement du robot. Ensuite, nous présenterons la régulation des paramètres permettant une optimisation de ce déplacement.

18.1 Asservissement

En robotique, le schéma le plus simple qui puisse être représenté pour l'asservissement est celui représenté à la figure 18.1.

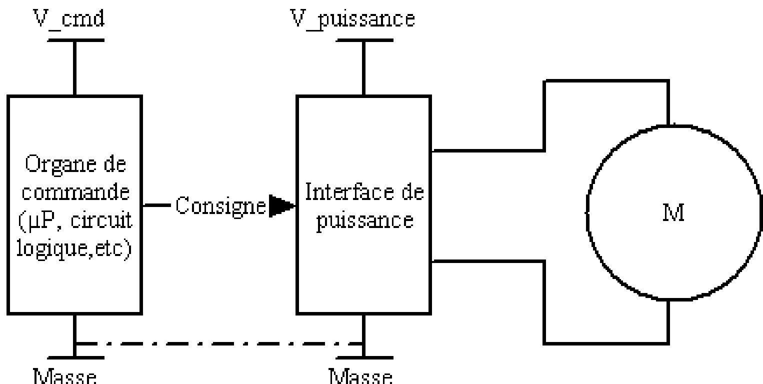
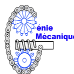


Fig. 18.1 – Commande classique d'un moteur DC en robotique

L'organe de commande envoie un signal à la carte de puissances qui va appliquer une tension, proportionnelle à la commande, aux bornes du moteur.

Si le robot doit se déplacer d'une certaine distance, son déplacement est bien évidemment entraîné par les moteurs de propulsion. L'organe de commande va alors envoyer une consigne



durant un certain lapse de temps mais, cependant, sans savoir si les moteurs auront réellement effectué ce qui leur a été demandé ou pas. L'asservissement consiste ainsi en la vérification de ce qui a été effectué et en corrigeant la commande si nécessaire. Dans le jargon automatique, il s'agit tout simplement d'une boucle de retour ! On travail donc en boucle fermée.

18.1.1 Commande

Le déplacement d'une plate-forme, bien que paraissant simple, est loin de l'être ! En effet, il ne suffit pas d'envoyer une simple suite de signaux binaires aux moteurs de propulsion et ensuite arrêter cet envoi après un certain temps pour les stopper. Afin de permettre un meilleur contrôle des moteurs pour les déplacements, une commande à modulation de largeur d'impulsion ou *Pwm* est nécessaire. Comme nous l'avons déjà décrit au chapitre 16.2, il est avant tout nécessaire d'initialiser ces *Pwm* avant de les appliquer aux moteurs (une pour le moteur gauche et une pour le moteur droit), en fonction des données entrantes. Cette partie du code se fait dans le fichier *moteur_interface.c.*

Nous ne détaillerons pas la programmation des *Pwm* ici étant donné que cela fait partie de la programmation basse. En effet, le code Nucléo que nous avons récupéré était suffisamment abouti pour que nous ne devions pas nous occuper de cette partie de la programmation.

18.1.2 Consignes - Génération de la trajectoire

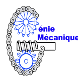
Comme nous l'avons déjà fait remarquer au chapitre 18.1.1, les deux moteurs de propulsion sont pilotés indépendement l'un de l'autre. Deux consignes différentes sont alors générées pour la mise en mouvement du robot. Cela permet, si les consignes sont identiques, de faire parcourir une ligne droite au robot, en avant ou bien en arrière. Si les consignes sont différentes, le robot peut alors faire des trajectoires non rectilignes, voir même pivoter sur lui-même lorsque les consignes sont de grandeur identique mais de signe opposé.

Toutefois, les moteurs ne peuvent pas être mis en marche et arrêtés de manière nette et brutale. Il est donc nécessaire d'introduire un facteur d'accélération au démarrage et à l'arrêt. En effet, cela evite le patinement des roues sur le sol du fait d'une accélération infinie au démarrage mais également un dérapage au moment de l'arrêt des moteurs, ce qui pourrait entraîner des erreurs de positionnement. Un profil de vitesse trapézoïdal (avec accélération et décélération) est repèsenté à la figure 18.2

A cette consigne de vitesse est associée une consigne de position (cf Figure 18.3). C'est cette consigne qui va servir à l'organe de commande pour communiquer avec la carte de puissance dans le but de générer les tensions de contrôle des moteurs de propulsion.

18.1.3 Odométrie

Comme nous l'avons déjà mentionné plus haut, il est nécessaire de mesurer ce que le moteur a réellement effectué afin de pouvoir contrôler les déplacement réels du robot. Pour ce faire, nous utilisons des capteurs impulsionnels incrémentaux, ou *Odomètres* (cf Figure 18.4),



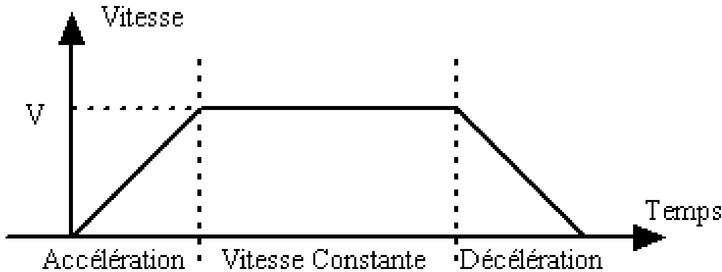


FIG. 18.2 – Profil de vitesse trapézoïdal

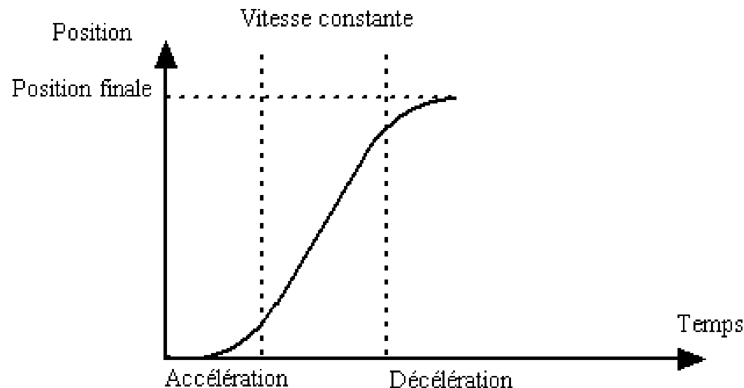


FIG. 18.3 – Profil de position assoscié au profil de vitesse trapézoïdal

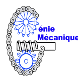
afin de récupérer une donnée proportionnelle au déplacement du robot. Ils fonctionnent par comptage et décomptage des impulsions qu'ils délivrent. Une roue codeuse est constituée d'un disque solidaire de l'axe de rotation, celui-ci étant percé de trous sur sa circonférence. Un système optique permet de compter le nombre de trous qui ont été *vus* lors de la rotation de la roue.

Le disque d'un tel codeur est composé de deux types de pistes (cf Figure 18.5). La première, la piste extérieure, est divisée en n intervalles réguliers alternativement opaques et transparents. Ce nombre n, que l'on appelle *résolution* ou encore *nombre de périodes*, correspond au nombre d'impulsions qui seront délivrées lors d'une rotation complète du disque de la roue codeuse.

Deux photodiodes (voies A et B), décalées l'une par rapport à l'autre, délivrent des signaux carrés en quadrature au travers de la piste extérieure (cf Figure 18.4). Le déphasage électrique de 90° des signaux sur les voies A et B permet une détermination du sens de la rotation de la roue codeuse (cf Figure 18.6). Une seconde piste, ou piste intérieure, est quant à elle composée d'une seule lumière (cf Figure 18.5), elle ne délivre donc qu'un seul signal impulsionnel par tour. Ce signal Z, aussi appelé *Top Zéro*, dure 90° électrique et est synchrone aux signaux A et B. Il détermine une position de référence qui permet une réinitialisation à chaque tour du disue codeur.

Ces capteurs nous permettent ainsi de connaître l'angle parcouru par la roue et le sens de la rotation. Par extrapolation, si on connaît la *résolution* de la roue codeuse, on peut facilement déterminer le déplacement réel du robot. Il existe deux types de calculs des données reprises par la roue codeuse :

– sur base de temps constante : on compte le nombre d'impulsions retournées pendant un



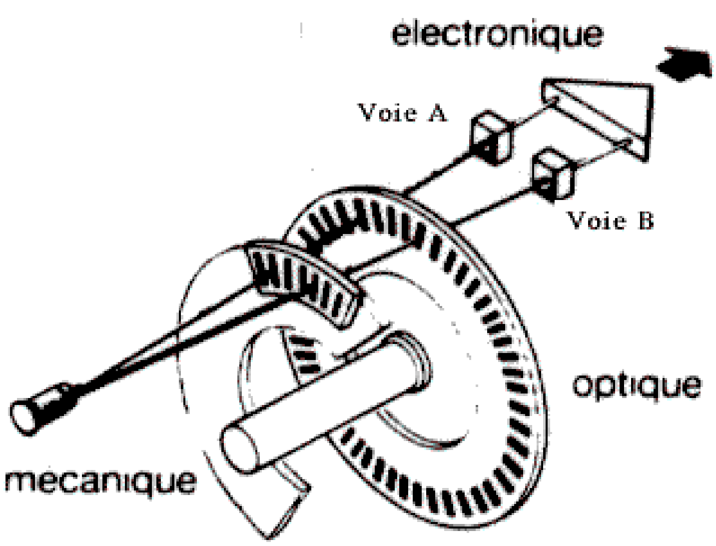


FIG. 18.4 – Codeur impulsionnel incrémental

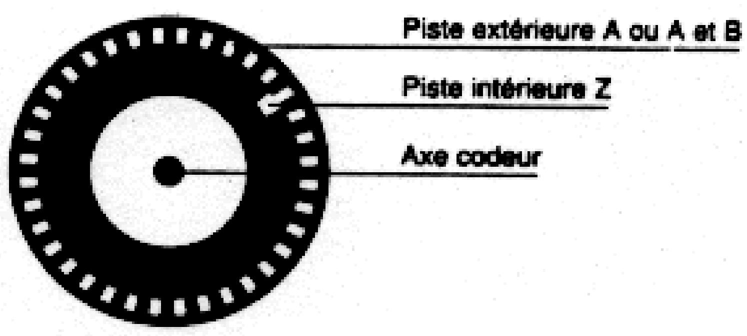
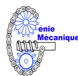


FIG. 18.5 – Disque interne d'un codeur impulsionnel

lapse de temps donné, celui-ci restant constant (cette solution n'est valable que si le temps de calcul est inférieur au temaps le plus court entre deux impulsions retournées par la roue codeuse) ;
- sur base impulsionelle : on détermine le temps entre deux impulsions retournées par la roue coudeuse.

Dans le cas de notre robot, nous prenons une base de temps constante et on compte donc le nombre d'impulsions retournées par la roue coudeuse durant ce lapse de temps. En effet, étant donné que le temps nécessaire pour effectuer une fois la boucle algorithmique de notre programme est important, il est nécessaire de choisir la seconde solution.



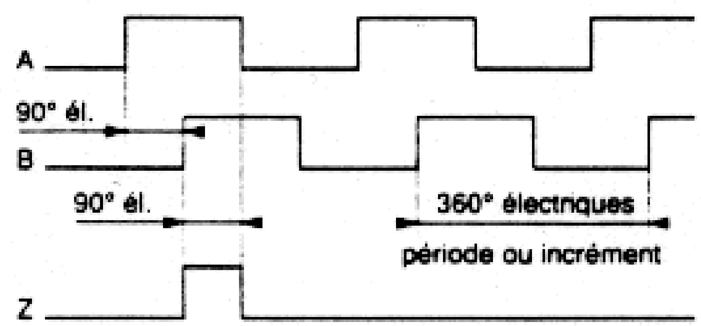


FIG. 18.6 – Signaux des voies A et B en quadrature dans un codeur

18.1.4 Odométrie dépendante et indépendante

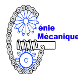
Il existe deux types de codeurs :

- les codeurs moteurs ;
- les codeurs indépendants.

Dans le premier cas, le codeur est directement monté sur l'axe du moteur. Ce type de montage soulève un problème important : si la roue dérape, la roue codeuse continue à envoyer un signal tandis que le robot ne bouge pas ! Il en résulte que l'odométrie dépendante n'est pas un choix probant pour une application en robotique puisqu'il y a de grands risques de dérapage. Dans le second cas, la roue codeuse est montée sur un axe de rotation n'ayant aucune liaison physique avec l'axe du moteur. L'indépendance des roues codeuses permet ainsi de mesurer le déplacement effectif du robot.

Il est toutefois à remarquer qu'il est primordial que les axes de rotation des roues codeuses, lorsqu'elles sont indépendantes, soient parfaitement alignés avec les axes de rotation des roues motrices. En effet, bien que cela ne pose aucun problème lors d'une trajectoire rectiligne, une erreur va apparaître durant une rotation puisque si l'alignement n'est pas respecté, il y aura patinage des roues.

Pour notre part, nous avons tout d'abord travaillé sur une programmation en utilisant des codeurs moteurs intégrés dans les moteurs *Maxon*. En effet, nous ne pouvions pas travailler sur les codeurs indépendants dès le début étant donné que nous n'étions pas encore certain de la fiabilité de nos moteurs. Nous avons ensuite développé un second programme permettant une utilisation des odomètre indépendants afin de rendre le système insensible aux variations d'adhérence. Cependant, ce code n'a pas été utilisé lors de la coupe de Belgique vu l'échéance qu'il nous restait.



18.1.5 Asservissement

Grâce à ce qui a été décrit aux points précédents, nous pouvons présenter un schéma plus complet de la gestion des moteurs (cf Figure 18.7).

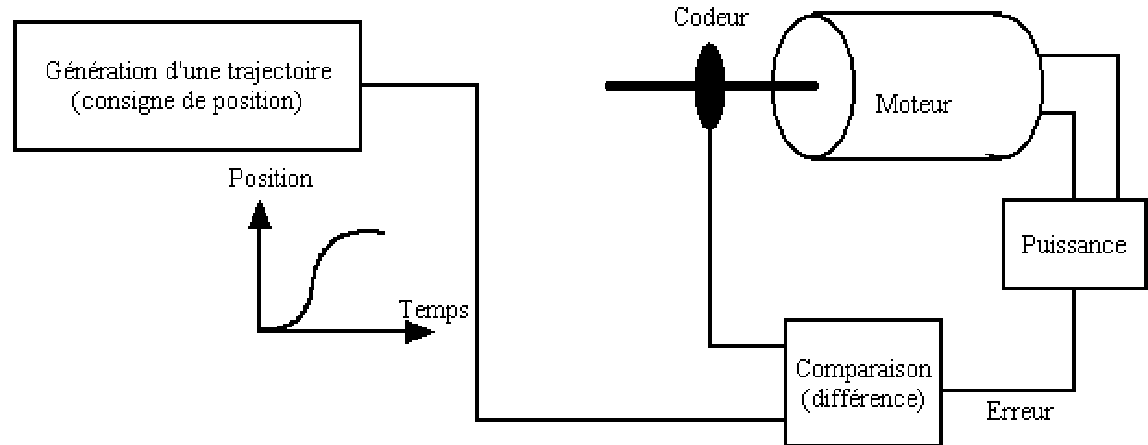


FIG. 18.7 – Principe simplifié de l'asservissement

Dans un premier temps, on détermine le déplacement à effectuer : c'est la génération de la commande. Dans un second temps, on effectue une comparaison avec ce qui a été effectué : c'est l'asservissement proprement dit de la trajectoire. Ainsi, plus la différence entre la consigne de position et la position réelle augmente, plus l'erreur va augmenter. Il va s'en suivre que la commande vers le moteur va elle-même augmenter. Ce qui va avoir pour conséquence que celui-ci va avoir tendance à diminuer l'erreur.

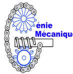
Dans notre cas, l'asservissement se fait sur deux plans. Dans un premier temps, nous avons un asservissement sur la vitesse. Celui-ci est modulable grâce à l'implémentation d'un PID informatique dans la boucle de contrôle. Dans un deuxième temps, un asservissement sur la position est également effectué. Ce dernier est contrôlé grâce à un PI. Nous reviendrons sur ces deux contrôleurs dans la partie consacrée à la régulation.

18.1.6 Paramètres à régler

Via tout ce qui a été décrit précédemment, il est simple de déduire que certains paramètres physiques doivent nécessairement être implémentés dans le programme afin de permettre l'asservissement et par conséquent le meilleur positionnement posible. Dans cette partie, nous allons décrire successivement les différents paramètres du code que nous avons du déterminer. Leur valeur fixe doit être implémentée dans le fichier *RegConstants.h*.

Entraxe des roues codeuses

Il est nécessaire de connaître l'entraxe des roues codeuses afin de pouvoir effectuer des rotations. Remarquons que plus l'entraxe sera grand (plus les roues codeuses se situeront sur l'extérieure du robot) et plus il faudra d'impulsions pour effectuer une rotation. On gagne donc



en précision ! Le paramètre, en millimètre, est le suivant :

```
#define CAxleLenght  130.95
```

Dans notre cas, il s'agit de l'entraxe des roues motrices puisque ce sont les codeurs intégrés aux moteurs *Maxon* qui sont utilisés. Notons toutefois que si nous avions utilisé les odomètres indépendants, ce paramètre aurait eu pour valeur 276,5mm.

Remarquons que plus la roue associée au codeur sera large et plus il sera difficile de déterminer le point réel de contact de celle-ci avec le sol. Afin de pallier ce problème, nous avons décidé de tourner des roues à gorges dans lesquels nous avons placés des *o-ring* dans le but de rendre le point de contact le plus ponctuel possible.

Avancée impulsionnelle des roues codeuses

Afin de pouvoir faire le lien entre les impulsions retournées par une des roues codeuses et l'angle effectué par la roue lors de sa rotation, il faut introduire un paramètre prenant en compte le nombre d'impulsion comptée par millimètre parcouru. Etant donné le fait que les paramètres physiques ne sont pas exactement identiques pour la roue gauche et la roue droite, il nous faut deux paramètres différents. Ceux-ci sont calculés en connaissant le nombre d'impulsion par tour de la roue codeuse, le taux de réduction de la transmission entre la roue touchant le sol et le codeur, ainsi que le diamètre de la roue touchant le sol. En ce qui concerne la coupe de Belgique, comme nous l'avons déjà expliqué au chapitre 18.1.4, la roue touchant le sol est la roue de propulsion puisque nous utilisons les codeurs moteurs.

Les diamètres et réductions relatifs à notre plate-forme sont les suivants :

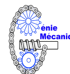
- Roue codeuse gauche : 36.4 mm
- Roue codeuse droite : 36.5 mm
- Rapport de réduction : 1/1
- Roue motrice gauche : 61.45 mm
- Roue motrice droite : 61.35 mm
- Rapport de réduction : 1/33

Le détermination théorique de chaque avancée impulsionnelle peut alors se fait comme suit :

$$CCptToDistR = \frac{D \cdot \pi}{33 \cdot 512} = 0.0114072389 \ mm/imp \quad (18.1)$$

$$CCptToDistL = \frac{D \cdot \pi}{33 \cdot 512} = 0.0114258327 \ mm/imp \quad (18.2)$$

Ces valeurs ne sont pas les valeurs réelles mais s'en approche et peuvent être utilisées lors des premiers tests effectués pour les déplacements de la plate-forme.



Les vraies valeurs sont déterminées de manière empirique. En faisant avancer la plate-forme d'une distance de un mètre, on regarde combien d'impulsions sont retournées par chaque roue codeuse. En supposant I, le nombre d'impulsions retournées pour un déplacement de un mètre, on aura :

$$CCptToDist = \frac{1000}{I}\ mm/imp \tag{18.3}$$

En effectuant cela plusieurs fois (une dizaine de fois, par exemple) et en prenant la valeur moyenne de l'ensemble des tests effectués, on obtient une valeur réelle pour chacun des paramètres d'avancée impulsionelle. On observe alors que ces valeurs sont légèrement différentes des valeurs théoriques. Cela est du à la non circularité parfaite de roues mais également à de légères erreurs dans la chaîne de mesure.

Les paramètres relatifs aux avancées impulsionnelles des codeurs droit et gauche (unité : millimètre par impulsion) sont donc implémentés dans le code comme suit (les valeurs prises en exemple étant celles déterminées empiriquement lors des tests effectués cette année) :

```
#define CCptToDistR  0.011657651
#define CCptToDistL  0.011474825
```

Concrètement, pour pouvoir effectuer ce test, il faut faire avancer la plate-forme en la poussant à la main tout en empêchant le robot de se mouvoir de par lui-même. Deux solutions sont alors possibles en ce qui concerne notre code. Soit on impose les vitesses des moteurs de propulsion gauche et droite à zéro dans le fichier *Regulation.cpp* :

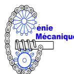
```
SpeedLeft = 0
SpeedRight = 0
```

Ou bien, on peut aussi imposer des tensions nulles aux bornes de ces mêmes moteurs :

```
PowerLeft = 0
PowerRight = 0
```

Biais constant entre les vitesses des moteurs de propulsion

Il n'est pas toujours évident d'obtenir les mêmes paramètres physiques pour la propulsion droite et la propulsion gauche. Pour des paramètres physiques différents, les efforts sur les moteurs de propulsion sont répartis de manière assymétrique (adhérence sur la table, couples résistifs de transmission, diamètre des roues, poids...). La puissance à fournir aux moteurs pour que leur vitesse de rotation soit identique est donc différente. Cela est d'autant plus vrai lorsque la transmission utilisée est, comme dans notre cas, une courroie. En effet, puisqu'il est



impossible d'obtenir exactement la même tension dans les deux courroies, et donc les mêmes couples résistifs de transmission, il faut pouvoir corriger les puissances envoyées aux moteurs.

Par ailleurs, même si deux moteurs appartiennent à la même série de fabrication, ils ne sont pas toujours identiques. Ainsi, ce n'est pas parce qu'une même tension est appliquée à leurs borenes que leur vitesses de rotation seront exactement identiques. Nous reviendrons sur ce point au chapitre 18.2.1 en ce qui concerne le choix des moteurs.

Le paramètre permettant d'atténuer, voire supprimer ces deux effets est le Biais :

```
#define CBiasRightLeft  1.09
```

Ce Biais permet ainsi de corriger la différence de puissance entre les deux systèmes de propulsion gauche et droit en faisant varier les vitesses des moteurs de propulsion. Ce terme est pris en compte dans le fichier *Regulation.cpp* :

```
SpeedLeft  = Speed - Bias;
SpeedRight = (Speed + Bias) * CBiasRightLeft;
```

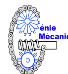
Dans notre cas, pour une valeur unitaire de ce Biais, la plate-forme décrivait une courbe vers la droite. Il a donc été nécessaire d'augmenter la valeur du Biais constant afin de corriger cette courbe et obtenir une trajectoire rectiligne pour un déplacement en ligne droite. La détermination de ce paramètre a donc été purement empirique. En somme, plus la valeur du Biais constant est grand et plus la plate-forme aura tendance à partir vers la gauche puisque la vitesse du moteur de propulsion droit augmentera.

18.2 Régulation

Comme nous l'avons déjà fait remarquer plus haut, notre plate-forme utilise des moteurs *Maxon*. A ces moteurs, sont associés des codeurs intégrés (non indépendants) qui renvoient directement l'information au processeur *Motorola*. Nous avons donc bien à faire à un asservissement en boucle fermée. Un contrôleur est donc nécessaire afin de permettre l'obtention de mouvements corrects lors des déplacements. Pour notre part, puisque nous avons opté pour le code de feu l'équipe *Nucléo*, nous avons tout d'abord un contrôleur PID (Proportionnel - Intégral - Dérivé) prévu pour la vitesse, et ensuite un PI (Proportionnel - Intégral) pour régler le positionnement.

Dans cette partie, nous allons avant tout décrire les principes de base nécessaires à la compréhension du fonctionnement d'un contrôleur PID afin de contrôler les réglages des paramètres de manière logique. Nous en viendrons ensuite à la description des différentes étapes de réglages des ces paramètres proprements dits. Dans un premier temps, la régulation que nous avons effectuée s'est faite sur un fonctionnement des moteurs en 24 Volt, ensuite nous avons opté pour un fonctionnement en 12 Volt. Nous reviendrons sur cela au chapitre 18.2.5.

Avant d'aborder ce chapitre, remarquons que dans tous les graphiques qui seront présentés ici, la commande fournie aux moteurs ne convergera jamais vers la consigne. Au mieux,



cette commande suivra une parallèle à la consigne. Cette erreur n'a rien avoir avec une erreur en régime mais est inhérente au code de la feu équipe *Nucléo*. Une des raison de ce comportement trouve son explication dans le fait que nous faisons fonctionner nos moteurs en haute vitesse. Nous n'avons, pour notre part, malheureusement pas été capables de supprimer ce bugg.

18.2.1 Choix des moteurs

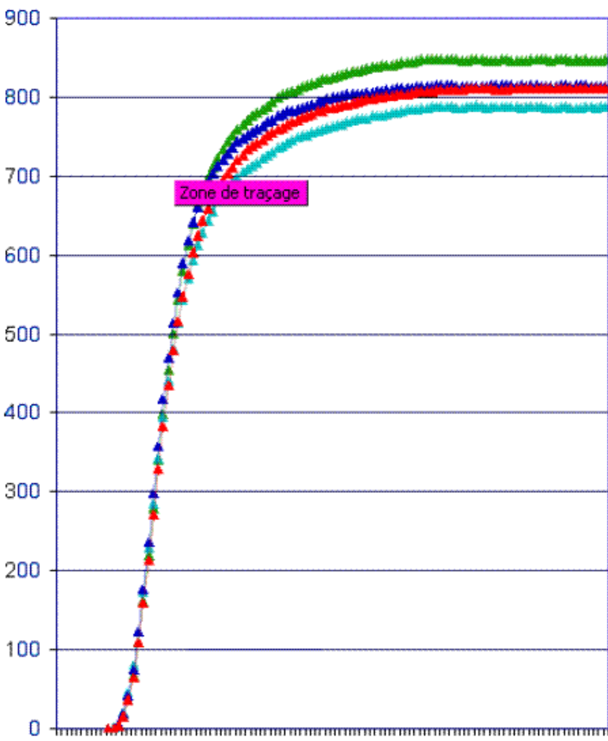
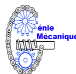


Fig. 18.8 – Courbes caractéristiques $\omega = f(U)$ des quatre moteurs *Maxon* en possession du service

Comme nous l'avons déjà vu au chapitre 18.1.6, les vitesses des roues de propulsion ne sont pas toujours identiques. Une des raisons provient du fait que les moteurs ne sont pas toujours exactement les mêmes (frottement des balais, pertes dans les roulementts...). Il est donc nécessaire de pouvoir les différencier qualitativement. Ce point est facilement observable en relevant les courbes caractéristiques $\omega = f(U)$ (figure 18.8), c'est-à-dire en imposant, au même instant, les mêmes tensions à chacun des moteurs en notre possession et en relevant les vitesses à vide de ceux-ci via les codeurs intégrés aux moteurs. L'imposition des tensions se fait évidemment avec seulement une régulation proportionnelle sur la vitesse (cf chapitre 18.2.3).

Grâce au graphique représenté à la figure 18.8, on observe que les comportements de deux des moteurs sont similaires. En choisissant ces deux moteurs pour notre propulsion, on s'assure donc une meilleure régulation. On minimise également les différences de vitesses entre les moteurs gauche et droit.



Une étude similaire avait déjà été effectuée par le pôle informatique de l'année dernière. Nos données ont confirmé celles relevées l'année dernière.

18.2.2 Actions des PID's

Un régulateur PID est un organe de contrôle qui permet d'obtenir une régulation d'un système automatique en boucle fermée (cf figure 18.9). Le choix d'un tel contrôleur a été fait été donné qu'il permet de contrôler la gande majorité des procédés. Pour notre part, les différents PID de contrôle étaient déjà implémentés dans le code, nous n'avons donc pas eu le choix de son utilisation.

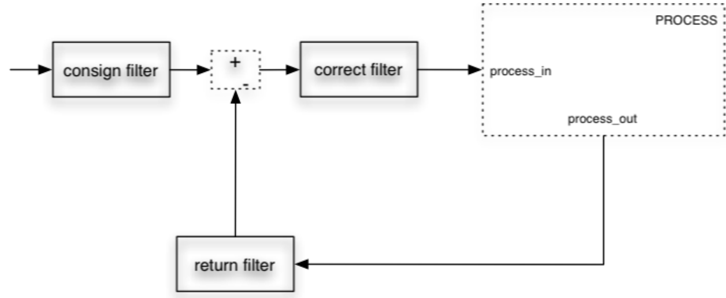


Fig. 18.9 – Asservissement minimaliste avec PID

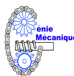
Un PID a l'avantage de permettre d'obtenir trois actions simultanées sur l'erreur de consigne :

- Une action proportionnelle qui a pour effet de multiplier l'erreur par un gain H ;
- Une action intégrale qui intègre cette erreur sur un intervalle de temps Ti ;
- Une action dérivée grâce à laquelle l'erreur est dérivée sur un temps Td.

Dans le but d'obtenir une réponse valable du procédé (stable et proche de la consigne) et de la régulation, il suffit de régler le contrôleur PID, ce qui revient à déterminer les trois coefficients H, Ti et Td. La finalité de cette manoeuvre est ainsi d'obtenir un système à la fois robuste, rapide et précis, tout en limitant les dépassements éventuels de la consigne. Nous avons dès lors trois notions fondamentales (cf figure 18.10) :

- La robustesse, qui défini le bon fonctionnement de la régulation lorsque le modèle varie autour de sa position d'équilibre. Le régulateur doit donc ainsi être capable d'accuser des petites variations de fonctions de transfert du procédé ;
- La rapidité, qui est fonction de deux paramètres temporels : le temps de montée et le temps d'établissement en régime ;
- La précision, qui est représenté par l'erreur statique.

Dans notre cas, il est intéressant de savoir comment les différents paramètres du PID influencent le système puisque notre but est de les régler. Les différents effets, dans le cas d'un asservissement en vitesse, sont les suivants :



- Le gain H permet de régler la réactivité du moteur. Lorsqu'il augmente, le temps de montée est plus rapide alors que le dépassement de la consigne se voit augmenter. L'erreur statique se voit améliorée ;
- L'action intégrale influe surtout sur la valeur en régime. On s'assure une valeur statique de la vitesse du moteur presque nulle lorsque la valeur de ce paramètre augmente. Le temps de montée est également raccourci. Toutefois, le temps d'établissement est allongé et le dépassement augmente. Remarquons que c'est cette action intégrale qui nous permet d'atteindre la vitesse maximale ;
- L'action dérivée, en augmentant, permet de minimiser les dépassements par rapport à la consigne. Le temps d'établissement se voit également amélioré.

Pour chacun des paramètres, à partir d'une certaine valeur limite, le système devient instable et le moteur s'emballe. Il y a donc des valeurs maximales qu'il ne faudra pas dépasser.

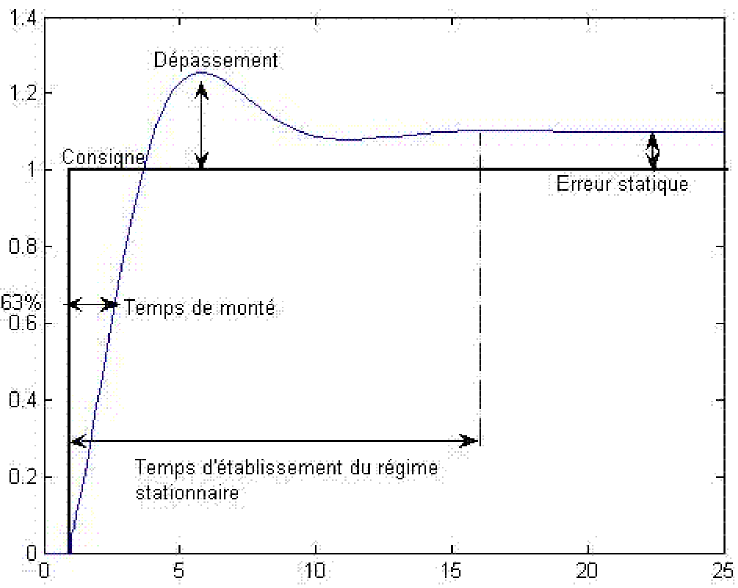
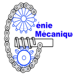


Fig. 18.10 – Réponse type d'un procédé stable avec contrôleur en boucle fermée pour une consigne de vitesse

Le paramétrage d'un PID est assez délicat étant donné qu'il n'existe pas une manière unique de réguler le système. Le contrôleur idéal n'existe pas, il parait donc évident qu'il faudra trouver des compromis afin d'obtenir la meilleure réponse possible tout en garantissant un procédé stable.

Une méthode théorique de réglage qui a déjà fait ses preuves est la méthodes de Ziegler-Nichols. Cette méthode utilise le gain et la fréquence d'oscillation de la réponse à la limite supérieure d'instabilité en boucle fermée, en l'absence de tout coefficient d'intégration ou de dérivation afin de déterminer les paramètres idéaux du PID. Cependant, dans notre cas, une méthode empirique plus performante sera utilisée.



18.2.3 Réglage des paramètres du PID de vitesse

Comme déjà mentionné au chapitre précédent, la méthode de réglage qui a été utilisée ici est une méthode purement empirique. Les différents tests sont effectués par mise en rotation des moteurs en charge, c'est-à-dire que la plate-forme n'est pas surélevée mais déposée sur ses roues. Dans un premier temps, les valeurs de chacun des paramètres des deux PID sont mises à zéro, à l'exception de la fonction proportionnelle sur la vitesse. Ce dernier est alors déterminé par augmentation progressive jusqu'à la limite inférieure d'instabilité. Une fois le premier paramètre déterminé, les actions intégrale et ensuite dérivée sont déterminées de la même manière et chacune à leur tour. Il est toutefois à noter que la valeur de chacun des paramètre influe sur celle des autres paramètres, la détermination n'est donc pas directe et définitive !

Pour la détermination des paramètres du PID de vitesse, afin d'observer une réponse du système lorsqu'on applique une consigne à celui-ci, il est nécessaire d'appliquer une vitesse constante aux moteurs. Afin de réaliser cela, dans le fichier *Regulation .cpp*, on impose les vitesses des moteurs gauche et droit à la vitesse voulue.

Réglage du paramètre proportionnel *KpS*

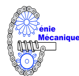
Lors des déplacements de la plate-forme, deux vitesses différentes sont utilisées. Afin d'éviter d'obtenir des dépassements démesurés, et par conséquent des instabilités lors de la mise en rotation des moteurs, il est nécessaire que le gain soit réglé pour atteindre la plus faible des vitesses désirées. Le réglage du *KpS* se fait donc à vitesse minimale (unité : millimètre par seconde), à savoir dans notre cas, pour un fonctionnement des moteurs *Maxon* en 24 Volt :

```
SpeedLeft = 250
SpeedRight = 250
```

Le coefficient proportionnel du PID de vitesse est implémenté dans le fichier *RegConstants.h* comme suit :

```
#define CBiasKpSCoef  0.63
```

En augmentant progressivement le paramètre proportionnel du PID on obtient successivement les trois situations représentées aux graphiques 18.11, 18.12 et 18.13. On voit que plus le gain augmente et plus la valeur en régime se rapproche de la consigne. Dans le dernier cas, on observe une instabilité franche du fait d'un gain trop important alors que dans le deuxième cas, une légère instabilité se fait ressentir mais n'est pas totalement visible. En réalité, la valeur optimale se situe entre le premier et le second cas. Remarquons qu'aucun dépassement n'est visible ici et que l'erreur en régime est importante, cela est notament dû au fait que les moteurs sont utilisés en 24 Volt et donc à plein régime. Nous étudierons ce problème plus en détail au chapitre 18.2.5.



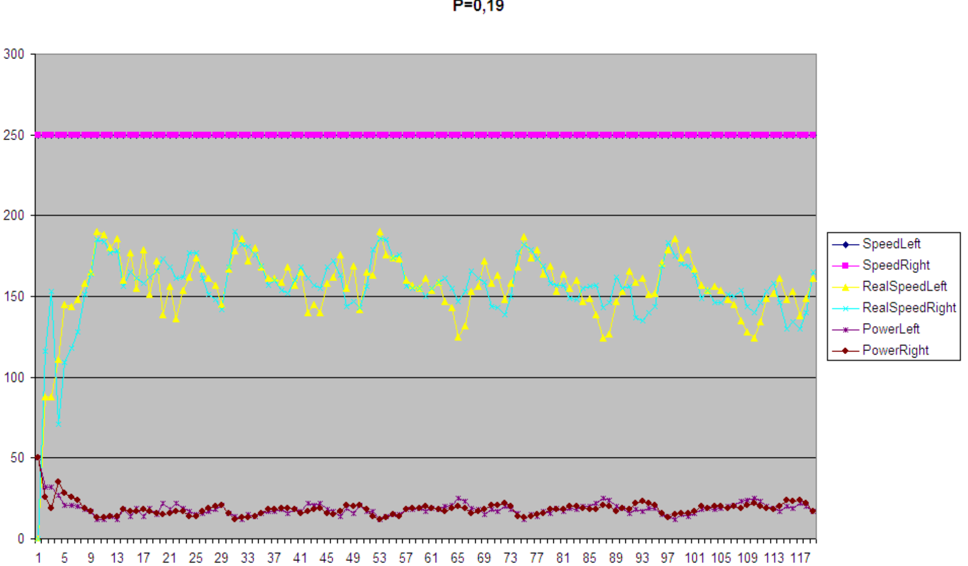


FIG. 18.11 – Représentation de la vitesse au cours du temps de chacun des moteurs pour un gain du PID de vitesse de 0,19

Réglage du paramètre intégral *KiS*

En ce qui concerne la régulation du paramètre intégral, celui-ci se fait à la vitesse maximale qui sera demandée d'atteindre aux moteurs de propulsion :

```
SpeedLeft = 400
SpeedRight = 400
```

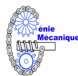
La définition du paramètre intégral, dans le fichier *RegConstants.h*, se fait via le paramètre suivant :

```
#define CBiasKiSCoef  2.6
```

Les figures 18.14, 18.15 et 18.16 représentent l'évolution de la vitesse réelle des moteurs au cours de temps lorsque l'on fait varier le paramètre intégral. On observe, par rapport à une simple régulation proportionnelle, que la valeur en régime se rapproche de la valeur de consigne. L'erreur statique se voit donc diminuée. Cependant, on observe toutefois une augmentation du dépassement.

Sur le graphique des figures 18.15 et 18.16, on observe très clairement des instabilités des moteurs du fait d'un coefficient intégral trop important. En fait, le paramètre idéal se situe légèrement en-dessous de la valeur limite d'instabilité. En fait, de manière optimale, on devrait observer un comportement similaire à celui reprit à la figure 18.14.

Toutefois, la régulation du paramètre intégral ne se fait pas purement graphiquement, elle se fait également à l'oreille et par observation de la plate-forme lors de son déplacement. Une



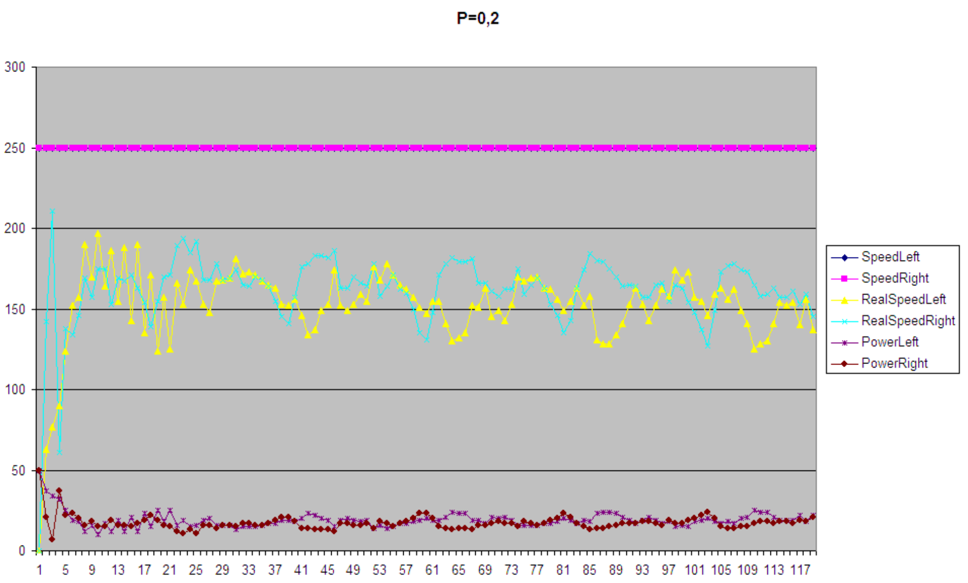


FIG. 18.12 – Représentation de la vitesse au cours du temps de chacun des moteurs pour un gain du PID de vitesse de 0,21

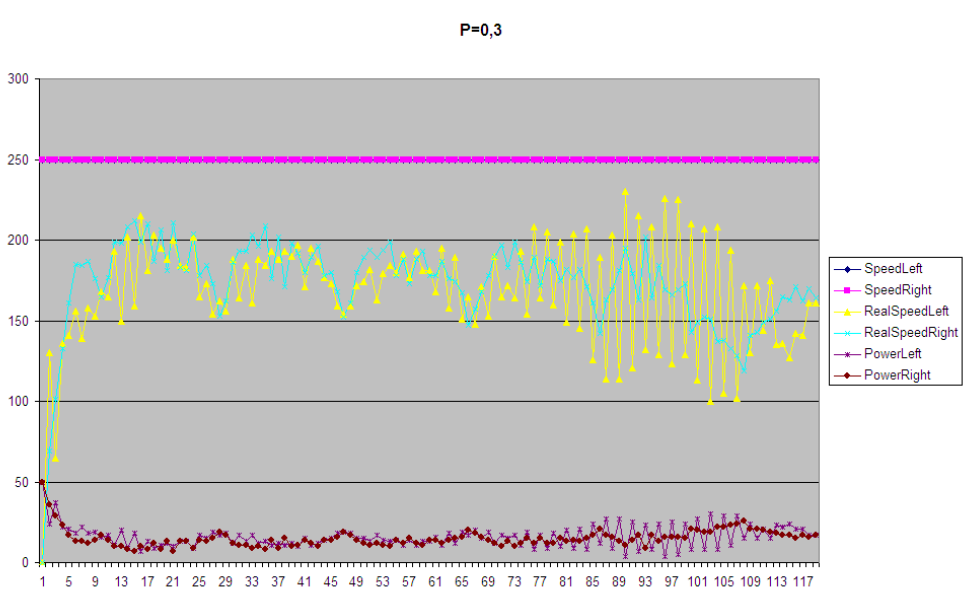
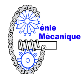


FIG. 18.13 – Représentation de la vitesse au cours du temps de chacun des moteurs pour un gain du PID de vitesse de 0,3



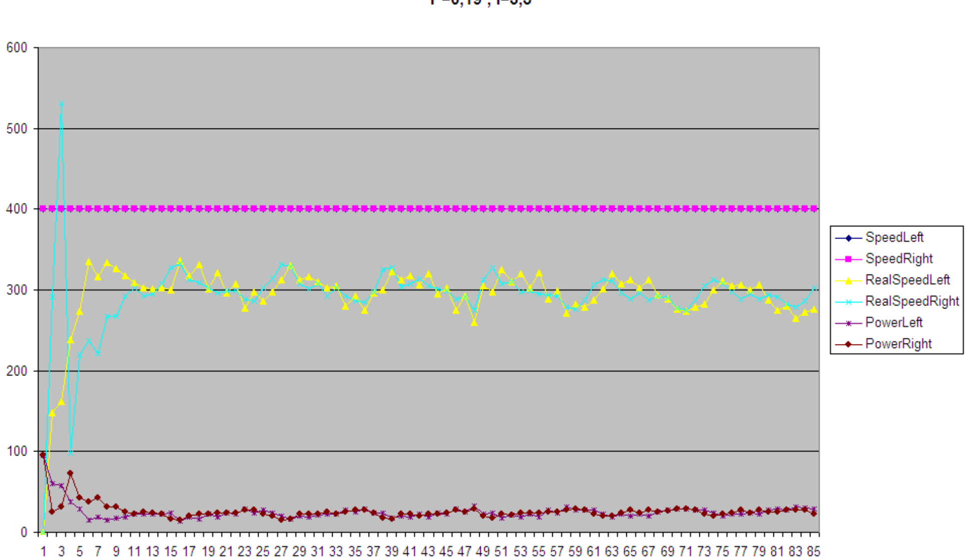


FIG. 18.14 – Représentation de la vitesse au cours du temps de chacun des moteurs pour un coefficient intégral du PID de vitesse de 3,5

vibration de celle-ci signifie une instabilité des moteurs !

Réglage du paramètre dérivé *KdS*

Le paramètre dérivé permet d'amortir le dépassement obtenu par l'augmentation des paramètres précédents. Le réglage de celui-ci se fait dans le fichier *RegConstants.h* grâce à la définition du coefficient suivant :

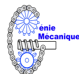
```
#define CBiasKdSCoef  0.0025
```

Notons que la régulation du coefficient dérivé est assez sensible. En effet, le comportement du dérivateur d'un PID peut être assimilé à une capacité dans un circuit électrique, c'est pourquoi la valeur du paramètre doit être relativement faible. De plus, si il est la régulation est mal effectuée, des instabilité peuvent rapidemment apparaître au sein du système.

Une représentation d'un résultat acceptable est faite à la figure 18.17. On observe clairement une atténuation du dépassement de la consigne. Elle est toutefois à noter que, comme pour le coefficient intégral, la régulation se fait également à l'oreille et par observation des vibrations de la plate-forme lors de son déplacement.

18.2.4 Réglage des paramètres du PI de position

La régulation en vitesse ayant été effectuée, il est maintenant nécessaire de s'intéresser à celle concernant le positionnement. Celle-ci ne se fait plus en imposant une vitesse constante aux deux moteurs de propulsions mais un déplacement linéaire sur une distance donnée. Dans



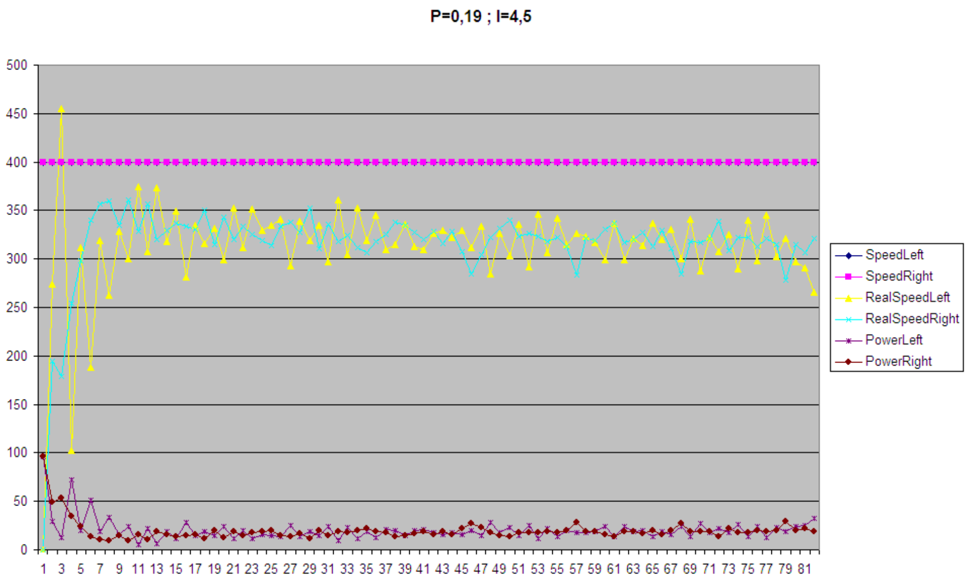


FIG. 18.15 – Représentation de la vitesse au cours du temps de chacun des moteurs pour un coefficient intégral du PID de vitesse de 4,5

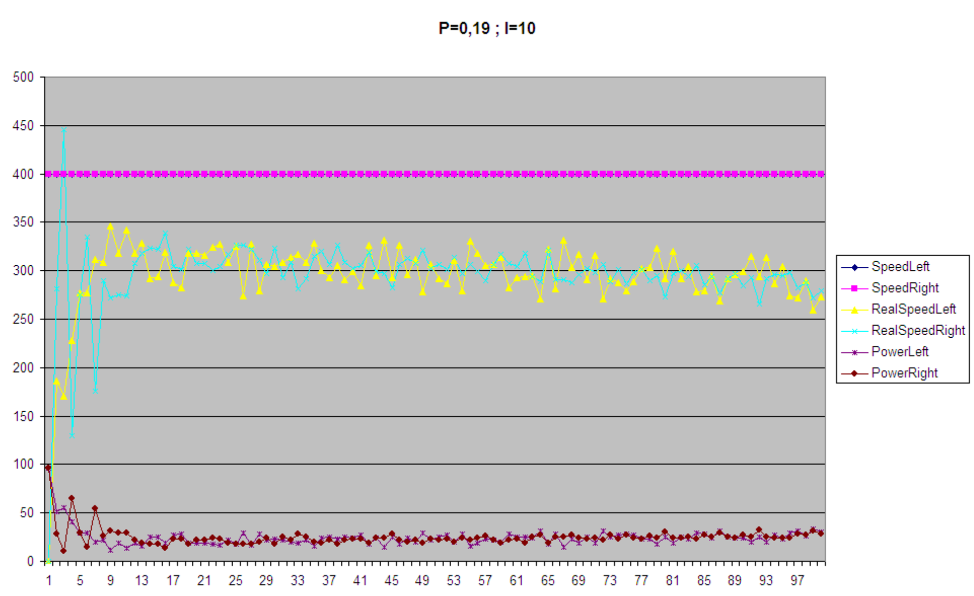
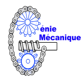


FIG. 18.16 – Représentation de la vitesse au cours du temps de chacun des moteurs pour un coefficient intégral du PID de vitesse de 10



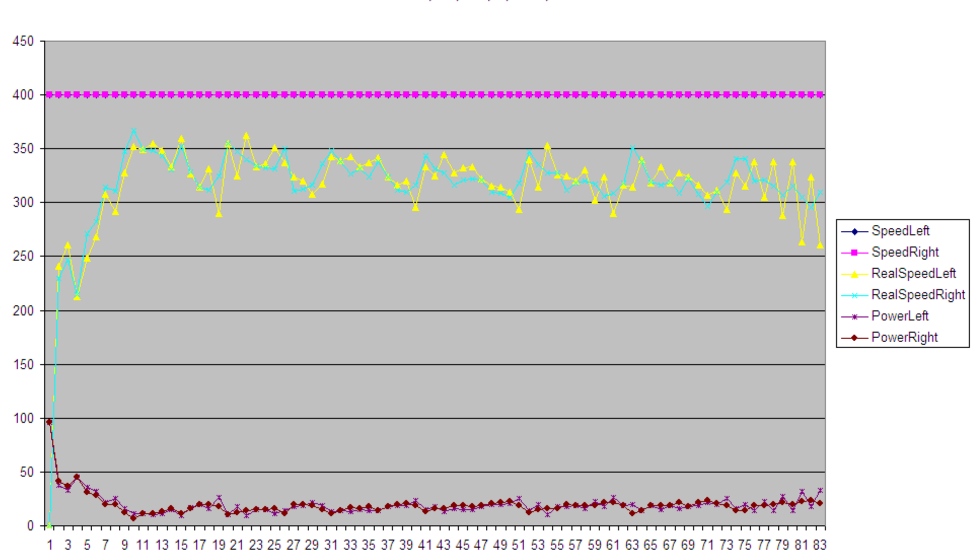


Fig. 18.17 – Représentation de la vitesse au cours du temps de chacun des moteurs pour un coefficient dérivé du PID de vitesse de 0,005

un premier temps, il est donc nécessaire de mettre en place une machine d'état de base comportant un seul et unique état. Pour faciliter les choses, le mouvement effectué sera le plus simple possible, à savoir un mètre sur une ligne droite :

```
SetMaxSpeed();
Move(1000);
```

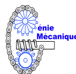
La vitesse choisie étant bien évidemment la vitesse maximale qui sera utilisée lors des déplacements :

```
#define CSpeedMax  600
```

Contrairement à la régulation en vitesse qui avait une action directe sur la vitesse en rotation des moteurs de propulsion, le contrôleur PI de position a une action sur les consignes qui sont imposées aux moteurs. Si on résonne sur un schéma automatique, le contrôleur PI en position se trouve donc avant le contrôleur PID en vitesse.

L'introduction du PI en position va nous permettre de compenser les différences de vitesses observées entre les deux moteurs de propulsion (cf figure 18.18). De ce fait, si la plate-forme dévie de sa trajectoire, la présence d'un PI bien réglé suffiera au fait que le robot compensera de lui-même cette déviation (cf figure 18.20), dans les limites de ses capacités bien évidemment.

Il est toutefois à remarquer que les erreurs de déplacement que nous pouvons typiquement relever sur les graphiques ne donnent lieu qu'à de très faibles variations de position sur la table. Sur de courtes distances et pour de simples mouvements, il est même possible que celles-ci n'auraient pu être décelées à l'oeil nu ! Cependant, lorsque la plate-forme effectue une série de



plusieurs trajectoires simples à la suite, les erreurs s'additionnent et peuvent dès lors prendre des dimensions dramatiques pour le bon déroulement d'un match. De par ce fait, plus les déplacements de la plate-forme seront complexes et plus le PI de position prendra toute son importance.

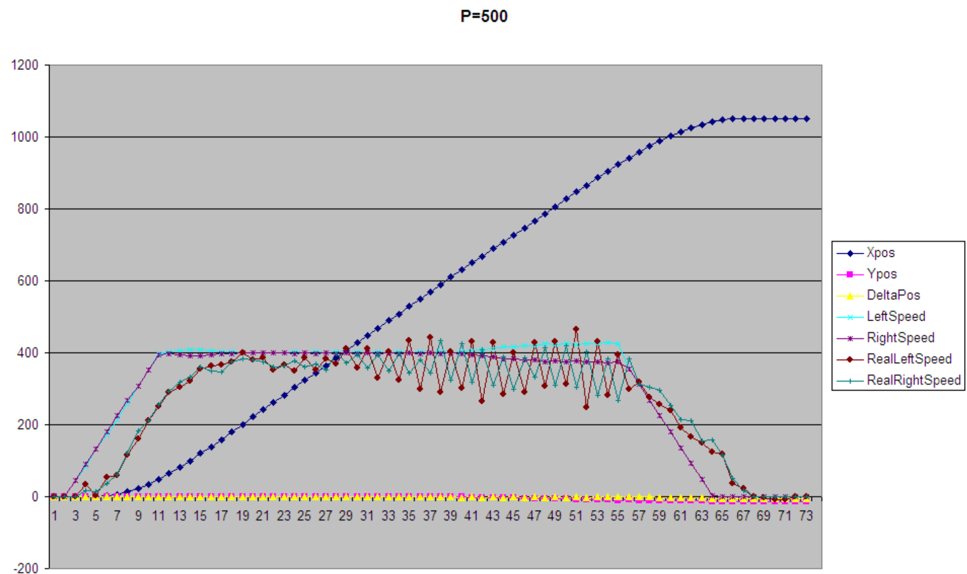


Fig. 18.18 – Représentation de la vitesse au cours du temps de chacun des moteurs avec un PID de vitesse et un P de position

La régulation des paramètres du PI en position est beaucoup moins intuitive que pour celle des paramètres du PID en vitesse. En effet, il ne suffit pas d'augmenter les coefficients de proche en proche jusqu'à la limite d'instabilités pour déterminer le réglage idéal. De nombreux tests d'observation des courbes et des mouvements sur la table de jeu est effectivement nécessaire afin d'obtenir le paramétrage voulu pour un positionnement acceptable.

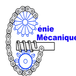
Réglage du paramètre proportionnel *KpP*

Comme nous pouvons le voir à la figure 18.19, le paramètre proportionnel va permettre de ramener la commande en direction de la consigne. Ce coefficient va donc diminuer l'erreur statique. En théorie, plus ce terme va augmenter et plus l'erreur en régime sur la position va diminuer, sous couvert qu'à partir d'une certaine valeur, des instabilités vont apparaître sur le positionnement. Dans notre cas cependant, le réglage a été un peu plus empirique.

Le terme relatif au coefficient proportionnel du PI de position est implémenté dans le fichier *RegConstants.h* tel que :

```
#define CBiasKpPCoef  100
```

On observe à la figure 18.18 que, bien qu'une boucle fermée avec un correcteur proportionnel ait été introduit, cela n'est pas suffisant. Sur la fin de la trajectoire, il persiste un écart entre



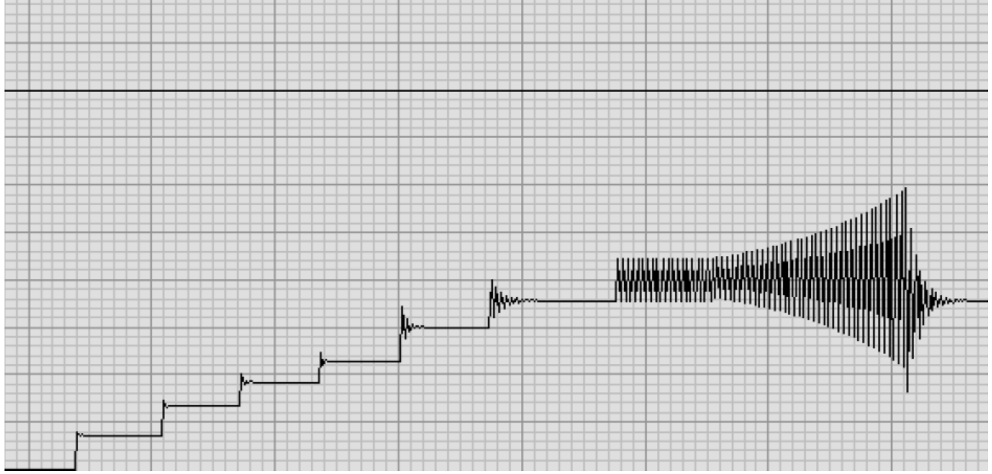


FIG. 18.19 – Effet du gain dans un contrôleur PI en position

les vitesses des deux moteurs. Remarquons qu'en l'absence de PI en position (gain unitaire en boucle ouverte pour le correcteur), l'écart aurait été d'autant plus marqué.

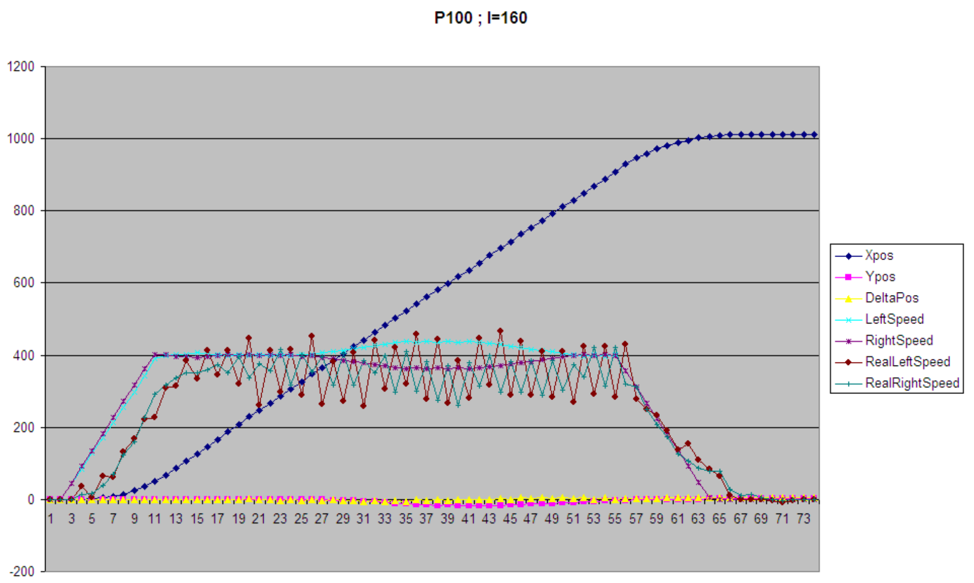
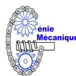


FIG. 18.20 – Représentation de la vitesse au cours du temps de chacun des moteurs avec un PID de vitesse et un PI de position

Réglage du paramètre intégral *KiP*

L'introduction d'un paramètre intégral supplémentaire va améliorer la régulation en position en permettant cette fois à la commande générée de converger vers la valeur de consigne. Une correction plus efficace de la déviation peut être observée à la figure 18.20. Théoriquement, plus ce paramètre augmente et plus la convergence vers la consigne de position est rapide (cf figure 18.21). Toutefois, à partir d'une certaine valeur du coefficient intégral, des instabilités



apparaissent lors du déplacement. Comme pour le cas de la régulation du coefficient proportionnel, notre approche a été beaucoup moins intuitive.

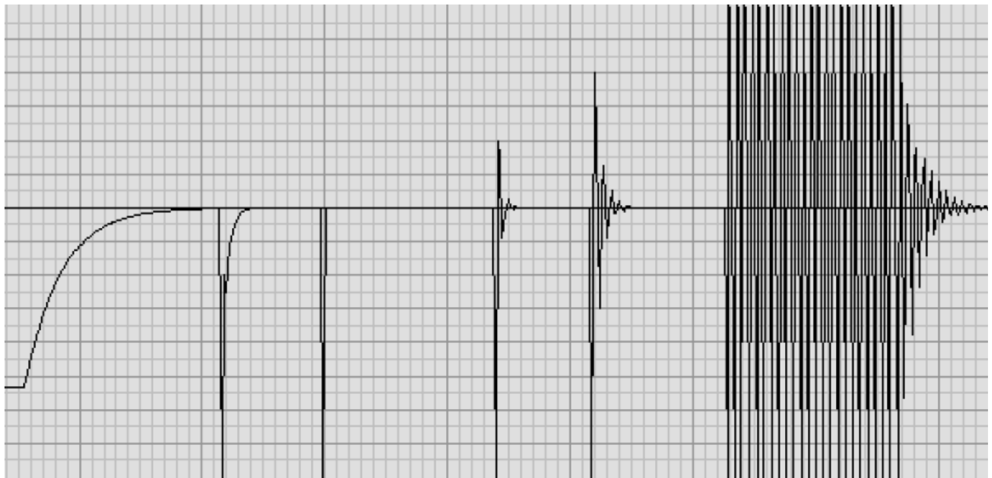


Fig. 18.21 – Effet du coefficient intégrateur dans un contrôleur PI en position

Le coefficient intégrateur relatif au PI de position est défini dans le fichier *RegConstants.cpp* tel que :

```
#define CBiasKiPCoef  20
```

Les écarts que l'on peut observer sur le graphique représenté à la figure 18.20 sont encore importants, il est donc nécessaire que le tuning du PI en position soit plus poussé. Ceci dit, comme nous le verrons au chapitre 18.2.5, d'autres paramètres entrent en compte dans la régulation. Nous y présenterons des graphiques avec une régulation plus optimale.

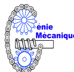
Il serait possible d'introduire un paramètre supplémentaire, à savoir un coeffiecient dérivé. Cependant, celui-ci n'est pas vraiment nécessaire puisque son rôle est moindre, mis à part qu'il a la particularité de supprimer totalement toute erreur en régime à long terme. Dans notre cas, puisque nos déplacement sont très courts, ce coefficient n'a pas vraiment son importance et on peut donc le négliger.

18.2.5 Tuning

Mis à part les différents paramètres que nous avons déjà étudiés ci-avant, d'autres peuvent encore être adaptés afin d'améliorer le comportement de la plate-forme du robot lors de sa mise en mouvement. Dans ce chapitre, nous allons décrire quelques uns de ces différents paramètres supplémentaires que nous pouvons régler.

Tension aux bornes des moteurs

Sur tous les graphiques représentés ci-avant, nous pouvons facilement observer que la puissance des moteurs de propulsion est très faible. Puisque nous désirons nous déplacer à une



vitesse relativement constante, cela entraîne bien évidemment des répercutions sur le couple moteur et donc sur la réactivité des moteurs de propulsion. Ce phénomène est dû au fait que nous utilisons nos moteurs *Maxon* en 24 Volt, c'est-à-dire à leur tension maximale.

Il est donc nécessaire d'augmenter la puissance fournie par la moteur ! Afin de régler cela, une solution est de placer les moteurs de propulsion sous une tension de 12 Volt. En effet, puisque les moteurs fonctionnent à tension constante, afin d'améliorer la réactivité de ceux-ci (c'est-à-dire leur couple moteur), la seule alternative que nous ayons est d'augmenter le rapport cyclique des pwms de commande des ces moteurs. Donc, pour ce faire, il est nécessaire de devoir baisser la tension appliquée aux bornes des moteurs de ropulsion. Dans notre cas, nous avons choisi de baisser celle-ci à 12 Volt.

Grâce à une telle modification, il nous est permis d'obtenir un profil de vitesses tel que représenté à la figure 18.22. Celui-ci représente les courbes de vitesses obtenues en imposant au système une consigne de vitesse constante sur les deux moteurs. Nous pouvons clairement voir sur ce graphique que la courbe de puissance est beaucoup plus élevée pour une utilisation en 12 Volt des moteurs qu'elle ne l'était en 24 Volt. On en conclue donc à une bien meilleure réactivité des moteurs du fait de la présence d'un couple moteur plus important.

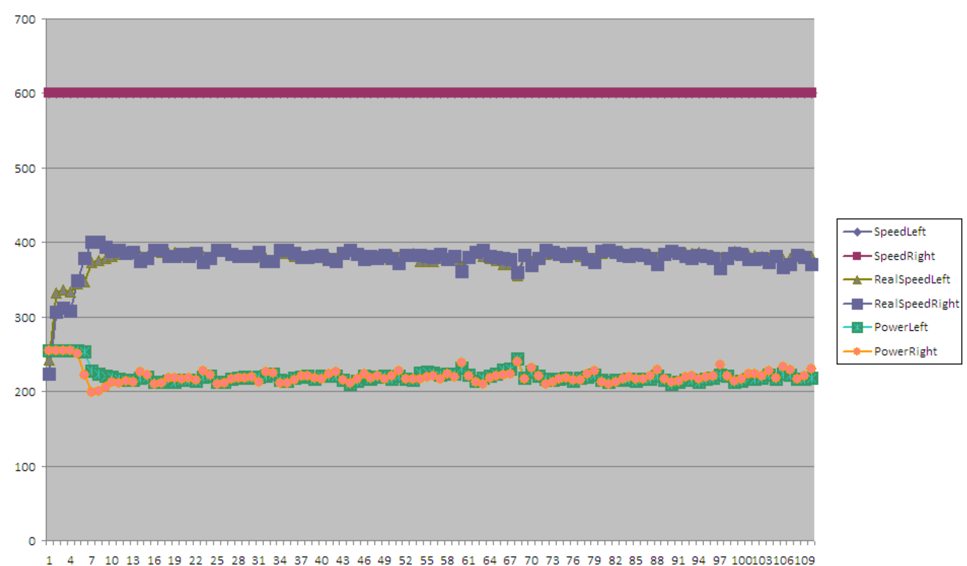
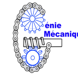


FIG. 18.22 – Représentation de la vitesse des moteurs de propulsion gauche et droit au cours du temps pour une tension de 12 Volt aux bornes de ces moteurs

Vitesse

Lors de nos différents tests, suite au passage à une tension de 12 Volt aux bornes des moteurs de propulsion, nous avons également pu remarquer que nous étions capables de nous déplacer plus rapidement tout en gardant une précision acceptable. Ainsi, nous avons défini comme nouvelle vitesse maximale pour notre robot (en millimètre par seconde) :



```
#define CSpeedMax  600
```

Pour information, une vitesse moindre est également utilisée lors de nos déplacements, par exemple durant les approches de parois ou les phases de dégagements. Cette dernière a pour valeur la moitié de la vitesse maximale, à savoir *0,3 m/s*.

Freinage

Un masse importante pour les robot de la Faculté, c'est devenu une marque de fabrication au fur et à mesure des années. Dès lors, pour stopper le robot rapidemment à la fin de sa trajectoire, une simple décélération n'est donc plus suffisante étant donné son poids important. Comme lors de la coupe de l'année dernière, trois fonctions différentes ont dès lors été nécessaires pour prendre en compte les diverses configurations possibles des moteurs. Ces fonctions ont simplement été récupérées dans l'ancien codes.

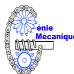
Pour la marche avant des moteurs gauche et droit, les deux fonctions suivantes ont été implémentées :

```
void set_forward_left (void)
{
quicc->pio_padat |=   PA_CLK7;
quicc->pio_padat &=   ~PA_CLK5;
}

void set_forward_right (void)
{
quicc->pio_padat &=   ~PA_CLK3;
quicc->pio_pcdat |=   PC_RTS2;
}
```

Afin de permettre une marche arrière des deux mêmes moteurs, deux autres fonctions ont alors été créées :

```
void set_reverse_left (void)
{
quicc->pio_padat |=   PA_CLK5;
quicc->pio_padat &=   ~PA_CLK7;
}

void set_reverse_right (void)
{
quicc->pio_padat |=   PA_CLK3;
quicc->pio_pcdat &=   ~PC_RTS2;
}
```

En combinant les fonctions de marche avant et de marche arrière des moteurs gauche et droit, il est évident qu'il est possibble de faire tourner le robot sur lui-même. De manière plus



complexe, il est également possible d'effectuer des courbes de Bezier.

En ce qui concerne le freinage du robot en fin de trajectoire, il s'agit tout simplement d'une mise en court-circuit des moteurs afin de les bloquer net. Les définitions des fonctions correspondantes sont les suivantes :

```
void set_brake_left (void)
{
quicc->pio_padat |=  PA_CLK7;
quicc->pio_padat |=  PA_CLK5;
}

void set_brake_right (void)
{
quicc->pio_padat |=   PA_CLK3;
quicc->pio_pcdat |=   PC_RTS2;
}
```

Le codage de ces trois configurations différentes a été effectué au sein du fichier *moteur_interface.c*.

Accélération et décélération

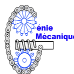
Comme nous l'avons déjà vu au chapitre 18.1.2, les profils de vitesse sont trapézoîdaux, c'est-à-dire que nous avons des accélérations en début et en fin de trajectoire dans le but d'éviter des sursauts de la plate-forme, du fait de la mise en mouvement de celle-ci et de son arrêt, mais également parce qu'il est physiquement impossible d'obtenir des accélérations infinies.

Après être passé à des tensions de 12 Volt aux bornes des moteurs de propulsion, nous avons remarqué l'apparition d'oscillations résiduelles à la fin de toute trajectoire de notre plate-forme (cf figure 18.23). Ces oscillations induisent forcément des erreurs sur le positionnement. Il est donc évident qu'il faut tenter de les supprimer.

Lors du fonctionnement des moteurs en 24 Volt, cet effet était imperceptible étant donné les puissances faibles appliquées aux moteurs.

La solution la plus simple pour laquelle nous avons opté, au risque de garder de minimes erreurs résiduelles de positions, est d'augmenter la décélération en fin de profil de vitesse. Après de nombreux tests, nous sommes donc arrivés au profil représenté à la figure 18.24. Dans ce cas, nous avons alors choisi une décélération (en millimètre par seconde au carré) huit fois plus élevée que celle utilisée lors de nos précédents tests :

```
#define CNormalDec  4000
```



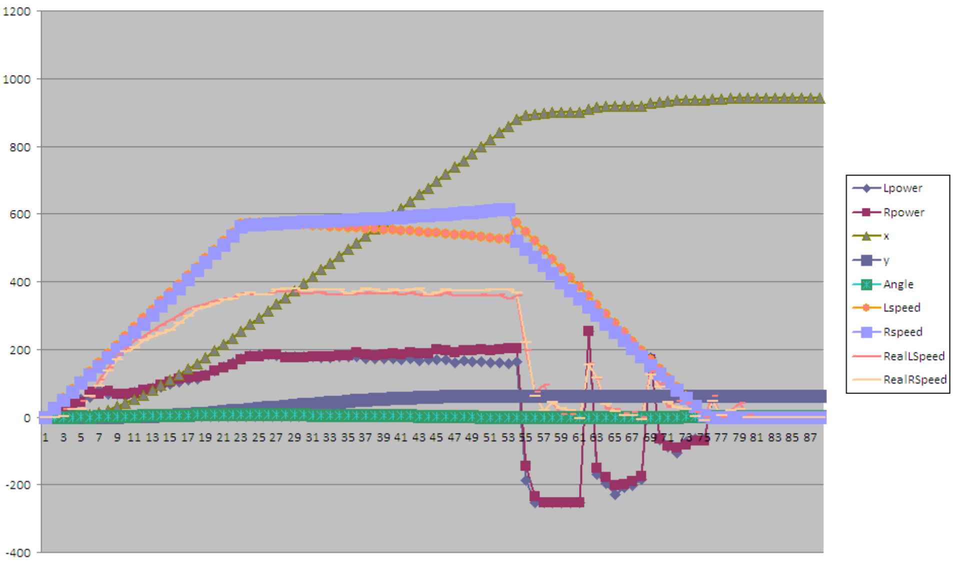


Fig. 18.23 – Profil des déplacements avec une décélération de 500

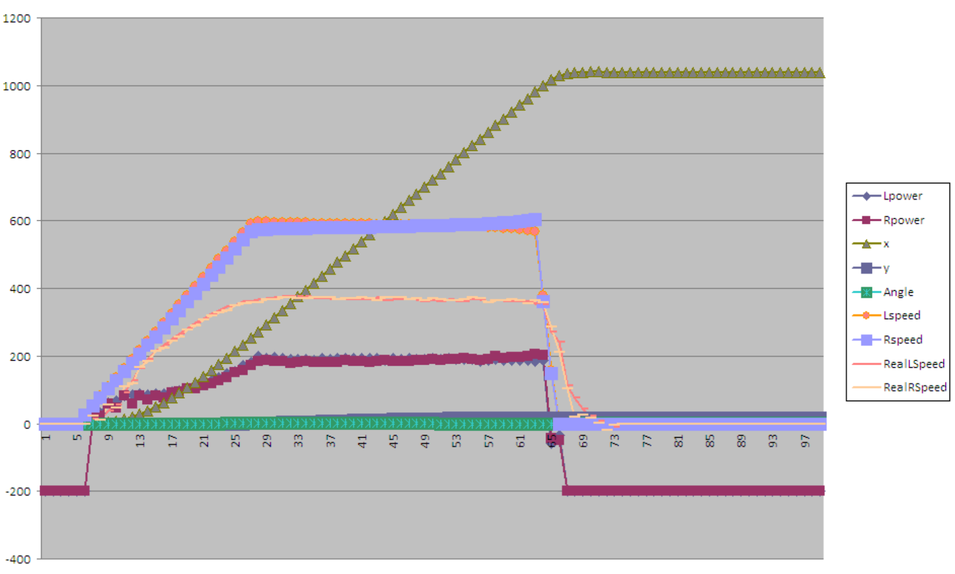
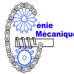


Fig. 18.24 – Profil des déplacements avec une décélération de 4000



Paramétrage de la régulation

Les différents paramètres que nous avons utilisés pour la régulations de notre plate-forme définitive sont listés pour information ci-dessous :

```
#define CCptToDistR      0.011657651
#define CCptToDistL      0.011474825
#define CAxleLenght      130.95
#define CPowerMax        254
#define CPowerMin        -CPowerMax
#define CSpeedMax        600          //mm/s
#define CSpeedMin        -CSpeedMax   //mm/s
#define CSpeedEnd        0            //mm/s
#define CSpeedNullAngle  Pi/10        //deg
#define CNormalAcc       500          //mm/s^2
#define CNormalDec       4000         //mm/s^2
#define CBiasAcc         100          //mm/s^2
#define CBezierDistMin   100          //mm
#define CBezierDistMax   250          //mm
#define CBiasKpPCoef     100
#define CBiasKiPCoef     20
#define CBiasKiPLimit    1
#define CBiasKpSCoef     0.63
#define CBiasKiSCoef     2.6
#define CBiasKdSCoef     0.0025
#define CBiasKiSLimit    20
#define CDistanceMargin  100          //mm
#define CAngleMargin     Pi / 60      //deg
#define CRegulDistance   20           //mm
#define CSpeedDiff       120          //mm/s^2
#define CSpeedFailDur    10000        //ms
#define CBrake           200
#define CBiasRightLeft   1
```

Certains paramètres n'ont pas été décrits dans les chapitres qui précèdes. La raison est simple, c'est parce qu'il n'ont pas été modifiés par rapport aux plate-formes des années précédentes. Ils sont cependant fournis ici à titre indicatif.

Chapitre 19

Capteurs & Actionneurs

Toutes les fonctions de bas niveau existaient déjà au sein du code récupéré en ce qui concerne les actionneurs et les capteurs. Il ne nous restait plus qu'à adapter les appels de fonctions en haut niveau en réglant les actions désirées pour notre utilisation propre.

19.1 Capteurs

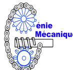
La plupart des entrées que nous avons utilisées provenaient de micro-switchs. Cependant, certaines autres données venaient de capteurs comme les balises Ultra-Son et Infra-Rouge ainsi que le Laser. Bien que nous n'ayons pas connecté les deux derniers capteurs, nous les avons toutefois prévus dans le code que nous avons développé cette année.

Afin de pouvoir identifier chaque capteur, il est utile que chacun possède une adresse informatique sur laquelle l'information désirée revient. Etant donné le nombre conséquent de capteurs que nous avons, plusieurs cartes de gestion des entrées ont été nécessaires. Bieen que nous n'ayons pas utilisé l'ensemble, nous avions les entrées suivantes à notre disposition :

- 8 entrées numériques et 4 entrées analogiques sur la carte principale ;
- 14 entrées numériques réparties sur deux cartes i2c.

Cette année, nous avons utilisé 15 entrées numériques (binaires) au total, réparties sur seulement 2 cartes (la carte principale et une des cartes i2c, la seconde servant de carte de rechange). C'est une structure d'énumération, implémentée dans le fichier *SensorsManager.cpp*, qui permet la définition des capteurs utilisés :

```
enum TSensor {NotUsed1,                  //Debut carte principale
              StrategySwitch,
              StartSwitch,
              BumperSwitch,
              ArmRightSwitch,
              BottomRightSwitch,
              MiddleRightSwitch,
              TopRightSwitch,           //Fin carte principale
              TopLeftSwitch,            //Debut carte secondaire
```



```
MiddleLeftSwitch,
 BottomLeftSwitch,
 ArmLeftSwitch,
 BaliseUS,
 CountSwitch,
 Laser,
 BaliseIR
};                          //Fin carte secondaire
```

Cette structure ayant été définie, il est nécessaire de pouvoir attribuer une adresse à chaque capteur en fonction du port sur lequel il a été connecté. La reconnaissance des capteurs se fait alors, dans un premier temps, grâce à un adressage des cartes de gestion de ces entrées :

```
#define CI2CSwitches1 0x00    // nappe principale
#define CI2CSwitches2 0x03    // carte d'extension
```

Ensuite, grâce à une fonction liée à la fonction d'énumération des capteurs, il est possible d'attribuer à chacun de ceux-ci quatre paramètres :

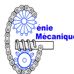
```
const TSensorsConfig CSensorsConfig = {{ 7, 0,   0,   0},
                                       { 6, 0, 100, 100},
                                       { 5, 0, 100, 100},
                                       { 4, 0,   0,   0},

                                       ...

                                      };
```

Le premier paramètre fourni un bit d'adressage à chaque capteur, ce qui permet dès lors de l'identifier de manière unique. Le deuxième donne sa valeur au repos (0 ou 1 puisque nous travaillons en binaire) et les deux derniers paramètres représentent des seuils sur le front montant et le front descendant. Ces deux dernières valeurs sont le temps minimal (en milliseconde) qu'un front doit être maintenu dans son nouvel état avant déclaration du front. Ils sont utilisés pour éviter les déclenchements intempestifs. C'est une manière informatique de filtrer les parasites.

L'adressage ayant été réalisé par la fonction de configuration, il est dès lors possible de pouvoir utiliser les différents capteurs via le nom de variable qui leur a été attribué.

Que l'on ait à faire à n'importe quel type de capteur, notre code n'est capable de récupérer qu'un seul bit sur chacune des adresses. Quatre type de détection sont cependant possibles, chacune pouvant retourner une valeur vraie ou fausse au code selon son implémentation :

- Le front montant ;
- Le front descendant ;
- L'état haut ;
- L'état bas.



Les fonctions permettant ces quatre types de détections sont les suivantes :

```
TBOOL DetectFrontEdge (TSensor Sensor,
                         TBOOL Enabled);
TBOOL DetectBackEdge (TSensor Sensor,
                        TBOOL Enabled);
TBOOL DetectHighState (TSensor Sensor,
                         TBOOL Enabled);
TBOOL DetectLowState (TSensor Sensor,
                        TBOOL Enabled);
```

Il est alors possible de coder les détections des données des capteurs de deux manières différentes. La première solution est de les définir directement comme des constantes de pré-processeur :

```
#define AdverseRobotPresence      DetectLowState(BaliseUS, true)
```

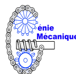
Ou alors, lorsque la valeur de l'entrée est directement liée à la valeur d'une fonction, le codage de la détection peut se faire directement dans la fonction même, comme c'est le cas pour le switch de démarrage dans le fichier *Interface.cpp* :

```
TBOOL IsGameStarted() {
  return DetectBackEdge(StartSwitch,true);
}
```

19.2 Actionneurs

19.2.1 Moteurs DC

Dans le robot, le système d'avalement, l'ascenseur et la catapulte sont actionnés par des moteurs brushless *Maxon EC*. La commande de ces moteurs se fait par la carte *Nucléo*. La partie électronique de puissance de celle-ci est munie de deux relais pouvant être (dés)activés pour alimenter des moteurs auxiliaires à la tension fournie par les batteries (12V). Deux problèmes apparaissent d'emblée : il n'y a que deux relais pour alimenter trois moteurs et la tension fournie par les batteries n'est pas correcte : les moteurs *Maxon EC* doivent être alimentés en 24V. De plus, les relais étaient à l'origine prévus pour alimenter des moteurs à balais et collecteur, dont la commande est plus simple que pour des moteurs brushless : si le courant passe le moteur tourne et pour inverser le sens de rotation il suffit d'inverser la polarité. Par comparaison, les moteurs brushless nécessitent une électronique de commande. Outre la tension de 24V nécessaire pour faire tourner le moteur, il faut aussi envoyer deux signaux distincts (signaux dont la tension est de 0 ou 5V) : le premier pour indiquer le sens de rotation du moteur et le second pour indiquer si le moteur doit tourner ou pas. Cette conception, bien que de premier abord plus compliquée, présente quelques avantages : il est possible de commander et d'inverser la rotation d'un moteur sans aucune modification du circuit électrique l'alimentant : il n'y a pas



besoin d'interrupteur dans le circuit.

Le montage finalement utilisé pour alimenter les moteurs brushless a été conçu et réalisé par M. Ludovic Dufranne (avec l'aide de Stéphane Leclercq pour le câblage). Les deux relais existant pouvant chacun fournir une tension de 12V, on peut obtenir une tension de 24V en les mettant en série (moyennant quelques précautions pour ne pas tout court-circuiter !). Les différents signaux de commande sont obtenus au moyen d'une carte électronique connectée au bus d'extension I^2C. Ainsi, une fois les deux relais activés, les moteurs sont alimentés en 24V ; il ne reste plus qu'à leur envoyer les instructions via le bus I^2C.

La première étape consiste à commander les deux relais. Cette commande se fait au moyen des fonctions suivantes de l'*API* : *StartDCMotors1()* et *StopDCMotors1()* pour le premier relais et *StartDCMotors2()* et *StopDCMotors2()* pour le second relais.

La seconde étape consiste à commander les moteurs proprement dit, et ce au moyen des fonctions suivantes :

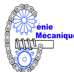
- void InitMotorEC(unsigned char SENS)
- void StartStop_MotorEC(unsigned char MOTOR)
- int GetStatusMotorEC()

Les deux premières fonctions servent à envoyer des instructions de commande aux moteurs tandis que la troisième sert à vérifier l'état des moteurs.

La fonction *InitMotorEC()* est utilisée pour imposer le sens de rotation aux moteurs. elle prend comme paramètre un caractère défini sur 8 bits (de la forme *0x..*, par exemple *0x05*). La valeur de caractère définira quels moteurs doivent tourner en sens inverse. Pour ce faire, chaque moteur est associé à la valeur d'un des bits du caractère : le premier moteur sera associé au bit d'ordre 0, le second au bit d'ordre 1 et le troisième au bit d'ordre 2. Supposons que le sens de rotation par défaut des moteurs soit le sens horlogique. Si le bit associé à un moteur vaut 0, les moteurs tourneront dans le sens horlogique. Si le bit vaut 1, le moteur tournera dans le sens antihorlogique. En forçant les bons bits à 1, on peut faire tourner les moteurs dans le sens qu'on veut. Lors de l'appel de la fonction *InitMotorEC()*, on lui passera plutôt une valeur décimale : la valeur *0x01* forcera le bit d'ordre 0 à 1, la valeur *0x02* forcera le bit d'ordre 1 à 1 et la valeur *0x04* forcera le bit d'ordre 2 à 1. En sommant ces valeurs, on peut changer le sens de plusieurs moteurs simultanément : *InitMotorEC(0x05)* fera tourner les moteurs 1 et 3 dans le sens antihorlogique et le moteur 2 dans le sens horlogique. Pour notre part, comme nous n'avons pas besoin de changer de sens de rotation au milieu du match, nous n'appelons la fonction *InitMotorEC()* qu'une seule fois lors du démarrage du robot.

La fonction *StartStop_MotorEC()* fonctionne sur le même principe que la fonction *InitMotorEC()*. Si le bit associé à un moteur vaut 1, celui-ci tournera. S'il vaut 0, le moteur sera à l'arrêt. On en déduit aisément que l'arrêt de tous les moteurs brushless sera activé par :

```
StartStop_MotorEC(0x00);
```



19.2.2 Servomoteurs

La carte *Nucléo* dispose de quatre emplacements pouvant accueillir des servomoteurs (numérotés de 0 à 3). Leur commande s'effectue assez simplement. Mais avant de pouvoir les utiliser, il faut les déclarer et les initialiser dans le code. Une structure *TServo* est définie dans le fichier *Servo.h*. Pour déclarer des servomoteurs, il n'y a qu'à faire :

```
TServo ArmServo;  //Déclaration de deux servomoteurs
TServo FrontServo;
```

Au démarrage du robot, il faut initialiser les servomoteurs. Cela se fait par l'appel de la fonction *InitServo()*. Celle-ci prend pour paramètres le servomoteur à initialiser, l'emplacement du servomoteur et deux positions angulaires (positions 0 et 1) :

```
InitServo(ArmServo,
          CArmServoId,
          CArmOpenCmd,   //Pos 0
          CArmCloseCmd); //Pos 1
SetServoInPos(ArmServo, 1);
InitServo(FrontServo,
          CFrontServoId,
          CFrontOpenCmd,  //Pos 0
          CFrontCloseCmd);//Pos 1
SetServoInPos(FrontServo, 1);
```

Dans ce code, *CArmServoId*, *CArmOpenCmd*, *CArmCloseCmd*, *CFrontSercoId*, *CFrontOpenCmd* et *CFrontCloseCmd* correspondent à des constantes de préprocesseur, définies plus haut dans le code (cf. section 21.2.1 pour plus de détails). Il est possible de définir des positions angulaires supplémentaires pour un servomoteur au moyen de la fonction *AddServoPos()*.

La fonction *SetServoInPos()* sert à donner l'ordre de déplacer un servomoteur vers une position angulaire (définie lors de l'initialisation). La mise en position d'un servomoteur sera rendu effective par l'appel de la fonction *ActivateServo()* (cette fonction sera appelée lors de la phase d'activation, cf. section 17.3 pour plus de détails sur le fonctionnement du code *Nucléo* et de la phase d'activation).

Chapitre 20

Stratégie

20.1 Introduction

Comme dit précédemment à la section 17.3, la structure générale du code va s'organiser autour de machines d'état et de sous-machines d'état. Nous allons dans ce chapitre voir comment cette structure nous a permis de réaliser la stratégie et comment la trajectoire de déplacement est intégrée à cette structure.

20.2 Grafcets

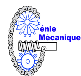
Avant de regarder comment ont étés codés la stratégie et le déplacement, jetons un oeil sur les grafcets. Il y en a en fait trois :

- le grafcet relatif au temps (figure 20.1),
- le grafcet relatif aux déplacements (figure 20.2),
- le grafcet relatif aux actions à effectuer (figure 20.3).

Comme nous pouvons le voir, ces grafcets reprennent la stratégie de manière générale. Ils ne nous donnent aucun renseignement sur le découpage en plusieurs machines d'état. Nous avons réalisé ce découpage en nous demandant si le robot effectuait des actions en se déplaçant ou pas :

- le robot est dans l'état *RobotGoToColorDistributor* : il effectue un déplacement entre son point de départ et le distributeur de balles de couleur (étapes 1 à 6 dans les grafcets) ;
- le robot est dans l'état *RobotPickUpBalls* : il est arrêté devant le distributeur et ramasse les balles de couleur (étape 7 dans les grafcets) ;
- le robot est dans l'état *RobotGoToFrozenContainer*[1] : il effectue un déplacement entre le distributeur de balles de couleurs et le container standard (étapes 8, 9 et 10) ;
- le robot est dans l'état *RobotCatapult*[1] : il est arrêté devant le container standard et largue sa cargaison de balles dans celui-ci (étape 11).

[1] A l'origine, le robot était sensé aller catapulter les balles de couleur dans le container réfrigéré correspondant. Malheureusement, le moteur actionnant la catapulte est tombé en panne deux jours avant la coupe de Belgique. Le robot et la stratégie ont été adaptés pour venir déposer les balles dans le container standard mais les états *RobotGoToFrozenContainer* et *RobotCatapult* ont quand même gardé leur ancien nom.



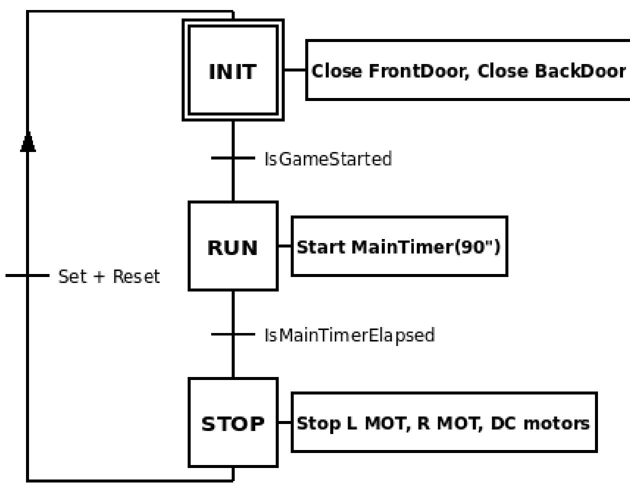


Fig. 20.1 – Grafcet relatif au temps

20.3 Codage de la stratégie et trajectoire de match

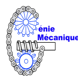
Dans cette section, nous allons voir le codage de la stratégie état par état. Comme nous l'avons déjà vu à la section 17.3.2, chaque machine d'état sera en pratique subdivisée à nouveau en sous-machines d'état. Deux fonctions seront appelées pour chaque machine d'état : *ManageRobot*NomDeLEtat*State()* et *Manage*NomDeLEtat*Command()*. Voyons maintenant plus en détail le fonctionnement des sous-machines d'état.

La structure *Nucléo* va donc se retrouver à l'intérieur de chaque état : après une phase d'initialisation, on va rentrer dans les sous-états. Il n'y aura en général que deux sous-états possibles : *StartTimerState*NomDeLEtat et *ProcessCmd*NomDeLEtat. Notons que la phase d'initialisation est ici utilisée pour démarrer un *timer* propre à chaque état. L'existence de ce *timer* nous permet de limiter dans le temps et d'interrompre le passage au sein d'une machine d'état, même si l'exécution des tâches associées à celle-ci n'est pas terminée. Par exemple, l'état *RobotPickUpBalls* s'interrompra automatiquement au bout de 25 secondes, même si le robot n'a pas ramassé cinq balles ; et dans le cas inverse, si cinq balles ont été comptées à l'entrée du robot, il est possible d'interrompre l'état *RobotPickUpBalls* avant la fin des 25 secondes.

20.3.1 *RobotGoToColorDistributor*

Le déplacement jusqu'au distributeur de balles de couleur et géré au moyen des deux fonctions suivantes :

```
void ManageRobotGoToColorDistributorState(){
  RecordGoToColDState(RobotGoToColDState);
  switch(RobotGoToColDState){
   case StartTimerStateGTCD :
        RobotGoToColDState = ProcessCmdGTCD;
        break;
   case ProcessCmdGTCD :
        if (IsTimerElapsed(GoToColDTimer)){
```



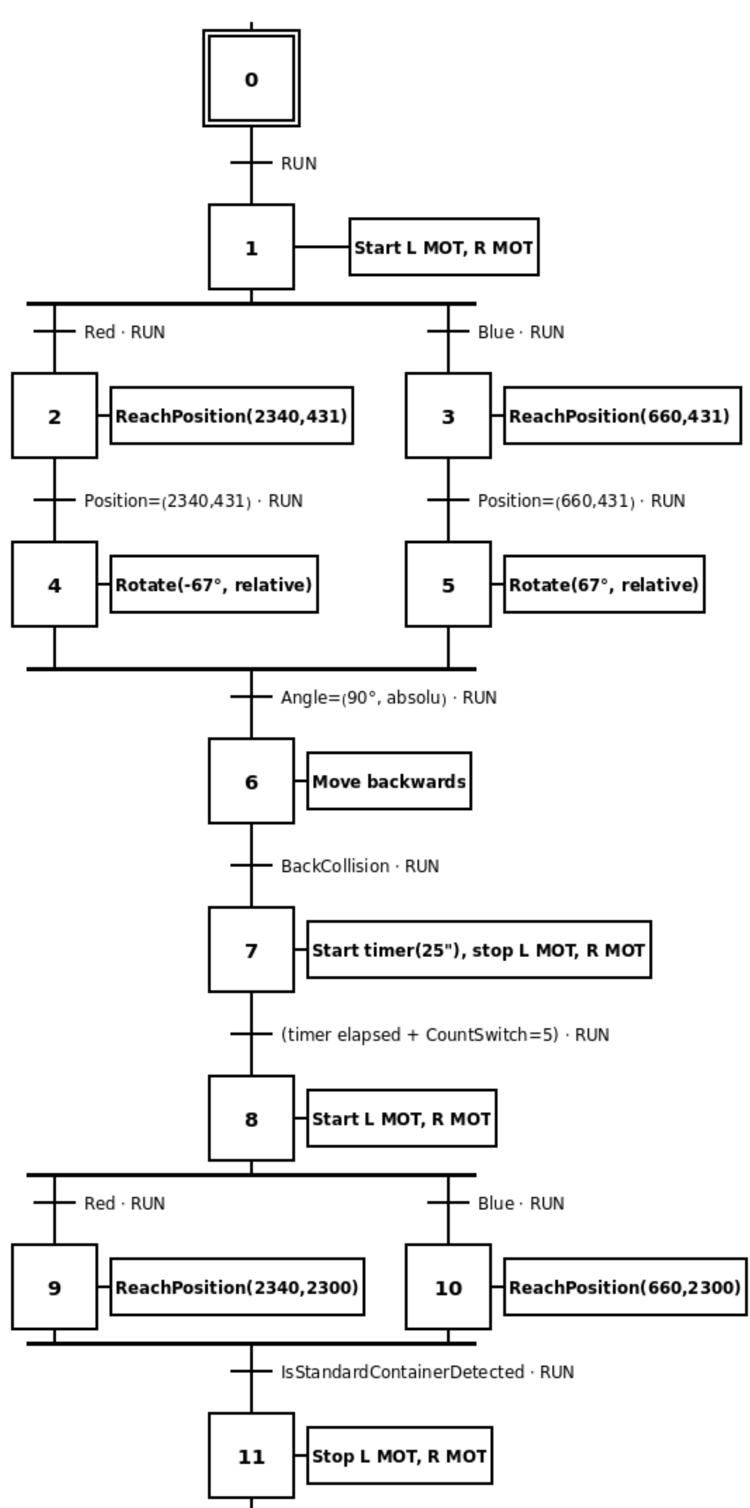
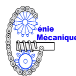


Fig. 20.2 – Grafcet relatif aux déplacements



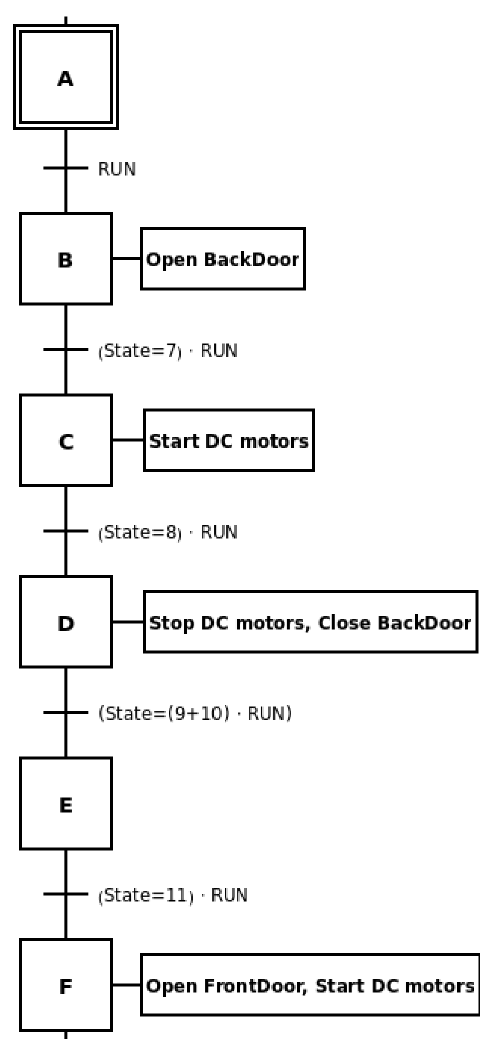
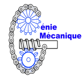
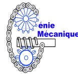


FIG. 20.3 – Grafcet relatif aux actions à effectuer



```
        GoToColorDistributorEnded = true;
        }
        break;
    }
  }

  void ManageRobotGoToColorDistributorCommand(){
    TStopReason ReasonGoToColD;
    TINT8 FlushedCommandsGoToColD;

    switch(RobotGoToColDState){
     case StartTimerStateGTCD :
        break;

     case ProcessCmdGTCD :
       if(!IsGoToColorDistributorInit()){
        GoToColorDistributorInit();
       }
       if (IsStopped(ReasonGoToColD, FlushedCommandsGoToColD))
       {
          RecordStopEvent(ReasonGoToColD, FlushedCommandsGoToColD);
          switch (ReasonGoToColD) {
            case TimerError:
            case CalibDone:
              SetMaxSpeed();
              Move(0);
              SetCmdIdxGoToColD(FlushedCommandsGoToColD);
              SendOdoCmdGoToColD();
          break;

            case BackCollision:
              SetMinSpeed();
              Move(0);
              Print("\nArrivee au distributeur de couleurs");
              GoToColorDistributorEnded = true;
              break;

            case Collision:
              SetMaxSpeed();
              Move(-250);
              SetCmdIdxGoToColD(FlushedCommandsGoToColD-1);
              SendOdoCmdGoToColD();
          break;

            case CommandsEnded:
              //Print("Deplacement termine");
            GoToColorDistributorEnded = true; //FIN DU DEPLACEMENT
              break;

            case AskedByUser:
```



```
            case NotStarted:
              SetCmdIdxGoToColD(FlushedCommandsGoToColD);
              SendOdoCmdGoToColD();
              break;
          }
        } else
        if (IsOdoReady())
        {
         if (GetCCmdNbrGoToColD()!=GetIdxGoToColD()){
              GoToColDCollisionBuffer = 0;
       SendOdoCmdGoToColD();
         }
        }
        break;
    }
  }
```

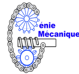
La première des deux fonctions, *ManageRobotGoToColorDistributorState()*, est assez simple : le sous-état associé à *RobotGoToColorDistributor* va passer une première fois dans l'état *StartTimerStateGTCD* puis passera à l'état *ProcessCmdGTCD*. Si le timer associé à *RobotGoToColorDistributor* est écoulé, on quitte le processus et on retourne à la machine d'état parent (*GoToColorDistributorEnded = true*).

La seconde fonction, *ManageRobotGoToColorDistributorCommand()*, est de premier abord un peu plus compliquée que la première. Mais pour analyser son fonctionnement, on peut subdiviser les commandes associées à *ProcessCmdGTCD* en deux parties, chacunes commençant par un test de condition (*if*) :

- La première partie, *if (IsStopped(ReasonGoToColD, FlushedCommandsGoToColD))*, nous sert à gérer les interruptions dans les déplacements : si le robot est contraint de s'arrêter au milieu d'un déplacement, la condition *IsStopped* devient vraie et en fonction de la raison qui a provoqué cet arrêt, on pourra effectuer une tâche différente, adaptée à la raison de l'arrêt. Notons au passage que la raison est renvoyée par l'*OdoManager* au moyen de la fonction *IsStopped()* ;
- La seconde partie est régie par la condition *if (IsOdoReady())* : si l'*OdoManager* est en état d'attente d'instructions, on peut lui en envoyer de nouvelles au moyen de la fonction *SendOdoCmdGoToColdD()*. On effectue avant cela le test *if (GetCCmdNbrGoToColdD()!= GetIdxGoToColdD())*, c'est-à-dire qu'on s'assure qu'il y bien encore des instructions de déplacement relatives au mouvement entre la zone de départ et le distributeur de balles de couleur à envoyer à l'*OdoManager*.

Voyons enfin la fonction *SendOdoCmdGoToColdD()*. Elle se présente comme suit :

```
void SendOdoCmdGoToColD()
{
  switch(IDXGoToColD)
  {
                    //GetStrategyConfig => On est bleu
```



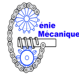
```
                    //!GetStrategyConfig => On est rouge

        case 0 : SetMaxSpeed();
                 if (GetStrategyConfig())  ReachPosition(660, 431);
                 else ReachPosition(2340, 431);
                 IDXGoToColD=1;
                break;

    case 1 : SetMaxSpeed();
     if(GetStrategyConfig()) Rotate(62,true);
     else Rotate(-67,true);
     ForceOpenArm();
                 IDXGoToColD=2;
          break;

    case 2 : SetMinSpeed();
                 Move(-480);
            IDXGoToColD=3;
          break;
        }
    }
```

La variable *IDXGoToColD* utilisée pour faire le *switch* est en réalité une variable globale c'est-à-dire qu'elle est définie à l'extérieur de la fonction. De ce fait, elle peut être utilisée telle quelle par d'autres fonctions et surtout sa valeur ne sera pas réinitialisée à chaque appel de la fonction *SendOdoCmdGoToColD()*.

20.3.2 *RobotPickUpBalls*

Sur le principe de fond, le manager propre à *RobotPickUpBalls* se déroulera de la même manière que celui lié à l'état *RobotGoToColorDistributor* vu précédemment. Néanmoins, vu qu'il n'y a aucun déplacement à effectuer au cours de cet état, la gestion des machines de sous-états va s'en trouver simplifiée :

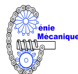
```
    void  ManageRobotPickUpBallsState(){
      RecordPickUpState(RobotPickUpState);
      switch(RobotPickUpState){
        case StartTimerStatePU :
          RobotPickUpState = ProcessCmdPU;
            break;

       case ProcessCmdPU :
            if (IsBallTankFull()) {
               StartStop_MotorEC(0x0); //arret des moteurs auxiliaires (tous)
               StopDCMotors1();
               StopDCMotors2();
               //BallTankDoorState = CloseBallDoor;
               PickUpBallsEnded = true;
```



```
            }
            else if (IsTimerElapsed(PickUpBallsTimer)){
               StartStop_MotorEC(0x0); //arret des moteurs auxiliaires (tous)
               StopDCMotors1();
               StopDCMotors2();
               //BallTankDoorState = CloseBallDoor;
               PickUpBallsEnded = true;
            }
            break;
      }
    }

    void ManageRobotPickUpBallsCommand(){
      switch(RobotPickUpState){
       case StartTimerStatePU :
          break;

       case ProcessCmdPU :
          if(!IsPickUpInit()){
           PickUpInit();
             }
             Print("\nBallCounter=%d\n",GetBallCounter());
             if( GetBallCounter() < 5 ) {
                if(IsBallDetected())
                    IncrementBallCounter();
             }
             else {
                PutBallTankFull(true);
             }
           break;
       }
    }
```

L'étape d'initialisation du ramassage des balles sera par contre un peu plus complexe que pour les autres états puisque outre l'initialisation du *timer* propre au ramassage, c'est aussi à ce moment là que seront mis en route les moteurs du système d'avalement et de l'ascenseur :

```
    void PickUpInit(){
    InitTimer(PickUpBallsTimer);
        StartTimer(PickUpBallsTimer,timeToPickUpBalls);
        StartDCMotors1();   //mise en route des relais
        StartDCMotors2();
        StartStop_MotorEC(0x03); // Mise en route du moteur d'avalement et d'ascenseur
        PickUpInitialized=true;
    }
```



20.3.3 *RobotGoToFrozenContainer*

La programmation du manager de *RobotGoToFrozenContainer* suit sensiblement le même schéma que celui de *RobotGoToColorDistributor*. Nous ne nous attarderons donc pas sur son implémentation. Notons cependant que la trajectoire implémentée consiste simplement à faire traverser la table dans le sens de la largeur par le robot.

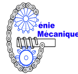
20.3.4 *RobotCatapult*

Pour terminer cette présentation du codage de la stratégie, voyons comment est implémenté le manager de *RobotCatapult* :

```
void  ManageRobotCatapultState(){
  RecordCatapultState(RobotCatapultState);
  switch(RobotCatapultState){
   case StartTimerStateCat :
      RobotCatapultState = ProcessCmdCat;
        break;

   case ProcessCmdCat :
     if (IsTimerElapsed(CatapultTimer)){
        StartStop_MotorEC(0x0);
        StopDCMotors1();
        StopDCMotors2();                // arrêt de tous les moteurs.
      CatapultEnded = true;
     }
     break;
  }
}

void ManageRobotCatapultCommand(){
  switch(RobotCatapultState){
   case StartTimerStateCat :
      break;

   case ProcessCmdCat :
     if(!IsCatapultInit()){
      CatapultInit();
     }
     break;
}
```

Et la fonction d'initialisation associée :

```
void CatapultInit(){
InitTimer(CatapultTimer);
    StartTimer(CatapultTimer,timeToCatapult);
    OpenFrontServo();
```



```
        StartDCMotors1();
        StartDCMotors2();   //mise ON des relays.
        StartStop_MotorEC(0x3); // mise en route des 3 moteurs auxiliaires
        CatapultInitialized=true;
    }
```

Comme nous pouvons le voir, ces fonctions sont assez simples : lors de la phase d'initialisation, on ouvre le servomoteur situé à l'avant du robot pour libérer les balles et on remet en route le système d'avalement et l'ascenseur afin de s'assurer qu'aucune balle ne reste coincée dans ceux-ci. Il ne reste plus qu'à arrêter tous les moteurs une fois que le temps est écoulé.

20.4 Evitement

Un évitement avait déjà été implémenté dans le code *Nucléo* mais celui-ci ne nous a pas convaincu : lorsque le manager d'odométrie renvoie la raison d'arrêt *Collision* à la fonction *ManageRobotGoToColorDistributorCommand()* ou *ManageRobotGoToFrozenContainerCommand()*, le robot s'arrête et amorce une marche arrière. Le problème est que au moment où le robot amorce sa marche arrière, il se trouve toujours à portée de la balise à ultra-sons. Le manager d'odométrie génère donc à nouveau la raison d'arrêt *Collision*, etc. En conséquence, le robot reculera de manière saccadée pendant un temps, finira par se perdre et larguera toute sa cargaison de balles au milieu de la table.

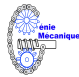
Pour palier à ce problème, nous avons dû modifier la fonction *ManageOdoStates()* au sein du manager d'odométrie. De plus nous avons élaboré une nouvelle manoeuvre d'évitement, afin de s'assurer d'aller jusqu'au container standard malgré tout dans le cas où le robot adverse resterait bloqué devant nous.

20.4.1 La fonction *ManageOdoStates()* modifiée

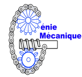
La grosse modification que nous avons apporté à la fonction *ManageOdoStates()* consiste en l'ajout d'une variable booléenne *CollisionFlagged* qui vaut *false* en temps normal et passe à *true* lorsqu'une collision est reportée[1]. La principale difficulté a été de savoir à quel moment on peut refaire passer la variable *CollisionFlagged* à *false*. Si on le fait trop rapidement, on va se retrouver dans le même cas que précédemment (vant modification de la fonction) et si on le fait trop lentement, on risque d'ignorer des signaux de collision capitaux (dans le cas où le robot adverse aurait bougé et se serait placé sur notre nouvelle trajectoire). Après plusieurs essais, nous en sommes donc arrivés à la fonction suivante :

```
    void ManageOdoStates()
    {
```

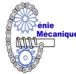
[1] Il y a collision lorsqu'on reçoit un signal de la balise à ultra-sons, c'est-à-dire lorsque le robot adverse se trouve devant nous à moins de 50 centimètres, ou lorsque le pare-chocs avant est enfoncé. A priori, la balise détectera toutjours l'obstacle avant le pare-chocs, celui-ci est là pour prendre le relais en cas de défaillance de la balise.



```
switch (OdoState) {
  case OdoStopped:
    switch (StoppedState) {
      case StoppedRequest:
      {
        StoppedState = StoppingOnGoing;
      }
      break;
      case StoppingOnGoing:
      {
        if (IsRegNoMotion())
        {
          StoppedState = StoppedConfirm;
        }
      }
      break;
      case StoppedConfirm:
      {
        if (! IsCommandFifoEmpty(Commands))
        {
          OdoState = OdoStarted;
          StartedState = StartedCmd;
        }
      }
      break;
    }
    break;
  case OdoStarted:
    switch (StartedState) {
      case StartedCmd:
      {
        StartedState = StartedFwd;
      }
      break;
      case StartedFwd:
      {
        if (! CommandsOnGoing)
        {
          OdoState     = OdoStopped;
          StoppedState = StoppedRequest;
          StopReason   = AskedByUser;
        } else

        if (IsCommandCompleted())
        {
          if (IsCommandFifoEmpty(Commands))
          {
            OdoState     = OdoStopped;
            StoppedState = StoppedRequest;
            StopReason   = CommandsEnded;
```



```
            } else
            {
              if (IsEmergencyBrakeNeeded()) {
               StartedState = StartedEB;
              } else {
                if(CollisionFlagged) CollisionFlagged=false;
/* C'est ici qu'on réinitialise CollisionFlagged à false après une collision */
                StartedState = StartedCmd;
              }
            }
          } else

          if (IsTimerElapsed(LastCommand.Timer))
          {
            OdoState     = OdoStopped;
            StoppedState = StoppedRequest;
            StopReason   = TimerError;
          } else

          if (CalibrationCollision)
          {
            OdoState     = OdoStopped;
            StoppedState = StoppedRequest;
            StopReason   = CalibDone;
          } else

          if (BackCalibrationCollision)
          {
            OdoState     = OdoStopped;
            StoppedState = StoppedRequest;
            StopReason   = BackCollision;
          } else

          if (IsCollisionReported()  && (!CollisionFlagged))
          {
/* Pour qu'une collision puisse être notifiée, il est nécessaire
 que CollisionFlagged=false */
            OdoState     = OdoStopped;
            StoppedState = StoppedRequest;
            StopReason   = Collision;
            CollisionFlagged = true;
          }
        }
        break;
        case StartedEB:
        {
          StartedState = StartedWait;
        }
        break;
        case StartedWait:
```



```
            {
              if (IsRegNoMotion())
              {
                StartedState = StartedCmd;
              }
            }
            break;
          }
          break;
      }
    }
```

20.4.2 La manoeuvre d'évitement

Nous pouvons distiguer l'évitement dans deux cas différents :

- l'évitement lors du déplacement entre le point de départ et le distributeur de balles de couleur ;
- l'évitement lors du déplacement entre le distributeur de balles de couleur et le container standard.

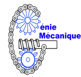
Dans le premier cas, nous n'avons pratiquement pas modifié la manoeuvre d'évitement par rapport à ce qui était préexistant. Nous nous contentons donc simplement de reculer légèrement pour ne pas toucher l'obstacle et d'attendre que celui-ci soit parti. Notons toutefois que grâce à la modification de la fonction *ManageOdoStates()* (cf. section 20.4.1), le robot ne recule plus de manière saccadée et ne se perd plus comme avant.

Dans le second cas, par contre, il a été nécessaire de développer un évitement plus complexe. En effet, la probabilité de croiser le robot adverse entre le distributeur de balles de couleur et le container standard est très grande : celui-ci sera susceptible de venir en travers de notre chemin soit pour se rendre à son container réfrigéré, soit pour venir déposer des balles dans la moitié du container standard qui lui est attribuée[2].

La manoeuvre d'évitement pour laquelle nous avons opté consiste à reculer légèrement puis à contourner le robot adverse en parcourant deux côtés d'un rectangle. Cette manoeuvre nous permet de venir toucher le container standard un peu plus loin sur le bord de la table.

Pour réaliser cet évitement, outre la modification de *ManageOdoStates()*, nous avons dû modifier conjointement les fonctions *ManageRobotGoToFrozCCommand()* (qui pour rappel se charge des commandes associées aux sous-états de *RobotGoToFrozenContainer*) et *SendOdoCmdGoToFrozC()* (qui contient les instructions de déplacement, étape par étape, pour le manager d'odométrie).

[2]Dans le cas des balles blanches, il est important de venir les déposer dans la moitié du container standard qui nous est attribuée. Notre stratégie à nous se basant sur les balles de couleur, nous avons le droit de venir les déposer n'importe où dans le container standard.

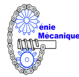


Voyons d'abord la fonction *ManageRobotGoToFrozCCommand()*. Nous n'allons pas reprendre ici toute la fonction (son principe est le même que pour la fonction *ManageRobotGoToColorDistributorCommand()* vue à la section 20.3.1), mais seulement la partie qui a été adaptée, à savoir la gestion d'une collision :

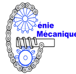
```
if (IsStopped(ReasonGoToFrozC, FlushedCommandsGoToFrozC))
     {
        RecordStopEvent(ReasonGoToFrozC, FlushedCommandsGoToFrozC);
        switch (ReasonGoToFrozC) {

          ...

          case Collision:
            if(GetIdxGoToFrozC() < 2)
            {
                SetMinSpeed();
                Move(-250);
                ForceCmdIdxGoToFrozC(2);
                SendOdoCmdGoToFrozC();
            }
            else
            {
                SetMinSpeed();
                Move(-150);
                SetCmdIdxGoToFrozC(FlushedCommandsGoToFrozC-1);
                SendOdoCmdGoToFrozC();
            }
        break;

          ...
        }
}
```

Et la fonction *SendOdoCmdGoToFrozC()* qui est est associée :

```
void SendOdoCmdGoToFrozC()
{
  switch(IDXGoToFrozC)
  {
                  //GetStrategyConfig => On est bleu
                  //!GetStrategyConfig => On est rouge


    case 0 : SetMaxSpeed();
             if(GetStrategyConfig()) ReachPosition(660,2300);
             else ReachPosition(2300,2300);
             IDXGoToFrozC=1;
             break;
```



```
    case 1 : SetFrontSwitchState();
             if(FrontSwitchState == Standard)
             {
                 IDXGoToFrozC=10;    //trajectoire terminée...
             }
             break;

    case 2 : SetMaxSpeed();
             if(GetStrategyConfig())   Rotate(0, false);
             else Rotate(180, false);
             IDXGoToFrozC=3;
             break;

    case 3 : SetMaxSpeed();
Move(800);
             IDXGoToFrozC=4;
             break;

    case 4 : SetMaxSpeed();
Rotate(90, false);
             IDXGoToFrozC=5;
             break;

    case 5 : SetMaxSpeed();
             Move(700);
             IDXGoToFrozC=6;
             break;

    case 6 : SetMaxSpeed();
             Move(700);
             IDXGoToFrozC=7;
             break;

    case 7 : SetMaxSpeed();
             Move(700);
             IDXGoToFrozC=8;
             break;

    case 8 : SetMaxSpeed();
             Move(700);
             IDXGoToFrozC=9;
             break;

    case 9 : SetFrontSwitchState();
             if(FrontSwitchState == Standard)
                IDXGoToFrozC=10;
             break;

    }
}
```



Si il n'y a pas d'évitement à effectuer, le programme passera uniquement dans les *cases* 0 et 1. Une fois que le container standard sera détecté (au moyen de la fonction *SetFrontSwitchState()*, le déplacement sera terminé (*IDXGoToFrozC=10*). Par contre, en cas de détection du robot adverse, la fonction *ManageRobotGoToFrozCCommand()* va forcer la valeur de *IDXGoToFrozC* à 2, forçant ainsi à effectuer la manoeuvre d'évitement (*cases* 2 à 9 dans *SendOdoCmdGoToFrozC()*).

Chapitre 21

Compléments

21.1 La connexion par port série

La liaison série entre l'ordinateur et la carte *Nucléo* s'effectue au moyen du câble de connexion que l'on peut voir à la figure 21.1. Le câble dispose de trois connecteurs (un mâle et deux femelles) et d'un interrupteur. Le connecteur mâle est relié à la carte *Nucléo*, les deux autres aux ports série de l'ordinateur.

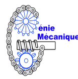
En utilisation standard, le premier port série relié à l'ordinateur sert à avoir le retour d'informations de la carte *Nucléo*. Le second port sera utile en mode *flash*, pour mettre à jour le software embarqué sur la carte *Nucléo*. On passe du mode normal au mode flash en actionnant l'interrupteur situé sur le câble de connexion.

Lorsque l'interface de débogage sous forme de menus (cf. section 21.3), le premier port série sert à afficher et naviguer dans les menus tandis que le second port effectue le retour d'informations (il prend donc le rôle du premier port en utilisation standard).

La mise à jour du software embarqué sur la carte *Nucléo* s'effectue assez simplement. Dans un premier temps, la compilation du code source permet de générer un fichier binaire : *nucleo.bin*. Celui-ci se trouve à la racine du projet. Dans le dossier *tools* du projet, on va trouver un programme appelé *NucleoFL32.exe*. C'est lui qui va nous permettre d'uploader le fichier binaire vers la carte *Nucléo*. Il n'y a qu'à l'exécuter en commandes MS-Dos avec comme paramètres l'emplacement du fichier binaire et le numéro du port série par lequel on uploade les données (le port série sur lequel on a branché le deuxième connecteur du câble de liaison).

```
NucleoFL32.exe ..\nucleo.bin 2
```

Il ne faut bien sûr pas oublier de faire passer la carte *Nucléo* en mode flash au moyen de l'interrupteur situé sur le câble de laison.



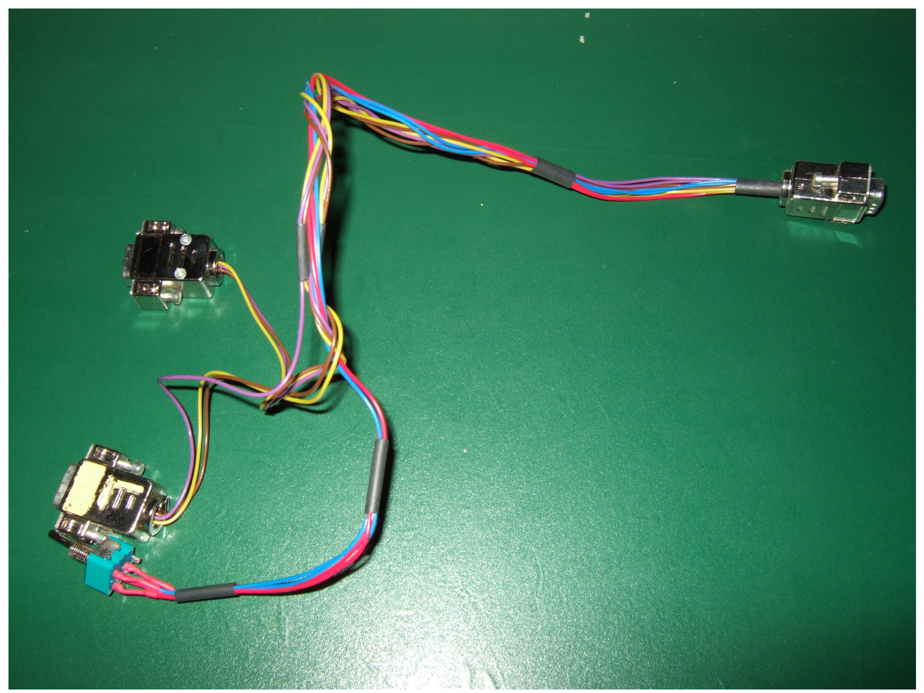


FIG. 21.1 – Câble de connexion série

21.2 Quelques notions de *C*

Dans cette section, nous ne donnerons pas un cours complet de *C*. Cela serait tout bonnement impossible et inutile, vu le nombre d'ouvrages et de sites internet existant sur le sujet. Nous nous contenterons simplement de (re)voir certaines notions qui reviennent souvent, que ce soit dans les codes *Nucléo* ou *Microb Technology*.

21.2.1 Les instructions de préprocesseur

Le préprocesseur est un programme qui s'exécute juste avant la compilation d'un programme en *C/C++*. Les instructions destinées au préprocesseur sont facile à reconnaître, ce sont toutes les lignes commençant par le symbole #. Avant la compilation, le préprocesseur va parcourir tous nos fichiers à la recherche des lignes commençant par # et les exécutera.

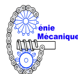
Les instructions les plus courantes sont bien sûr les *#include* :

```
#include "monfichier.h"
```

Lorsque le préprocesseur rencontrera cette ligne, il remplacera littéralement la ligne par le contenu de *monfichier.h*.

Grâce à l'instruction *#define*, il est possible de définir une constante de préprocesseur :

```
#define CCptToDistR     0.011657651
```



Partout où le préprocesseur vera le mot *CCptToDistR* dans le code, il viendra le remplacer par sa valeur (*0.011657651*).

il est aussi possible de définir une constante sans lui attribuer de valeur :

```
#define __TEST__
```

La constante *__TEST__* n'a pas de valeur, mais elle existe. Ce genre de définition simple présente tout son intérêt pour les compilations conditionnelles. Voyons par exemple le code suivant :

```
#ifdef __TEST__

/* Portion de code */

#endif

/* Suite du code*/
```

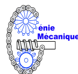
La portion de code située entre les lignes *#ifdef __TEST__* et *#endif* sera conservée pour la compilation seulement si la constante *__TEST__* a été définie préalablement, contrairement à la suite du code qui sera compilée dans tous les cas. Si la constante *__TEST__* n'est pas définie, la portion de code entre *#ifdef __TEST__* et *#endif* sera tout simplement supprimée avant compilation.

21.2.2 Les énumérations

Les énumérations sont souvent utilisées au sein du code *Nucléo*, spécialement pour définir les différents états possible pour une variable. Prenons par exemple la variable d'état *RobotState* : elle est de type *TRobotState*, défini comme suit :

```
typedef TUINT8 TRobotState;
enum {RobotStopped,
  RobotGoToColorDistributor,
      RobotPickUpBalls,
      RobotGoToFrozenContainer,
      RobotCatapult};
```

La variable *RobotState* peut prendre un nombre limité de valeurs (les "constantes nommées"), l'ensemble de ces valeurs possible formant la "liste d'énumérateurs". L'intérêt de l'énumération est que lorsque nous programmons, nous assignons une valeur sous forme de lettres à la variable, valeur qui ne prête pas à confusion (par exemple, *RobotState=RobotPickUpBalls*). Au moment de la compilation, ces différentes constantes nommées seront automatiquement remplacées partout dans le code par leur valeur numérique équivalente (*RobotStopped*=0, *RobotGoToColorDistributor*=1, etc.). Cette façon de procéder nous permet de travailler en assignant des noms en toutes lettres à la valeur d'une variable, tout en économisant de la place en



mémoire (un entier nécessitant beaucoup moins de place pour être stocké en mémoire qu'une chaîne de caractères).

21.3 L'interface *Debug*

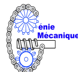
Le code *Nucléo* est muni d'une interface de débogage. Après que nous l'ayons adapté à nos besoins (création de nouvelles fonctions et de nouveaux menus), celle-ci s'est avérée très utile, notamment pour le tunning de nombreux paramètres. L'interface de débogage peut être utilisée de deux manières différentes : soit par retour d'informations additionnelles par le port série, soit par le biais d'une interface sous forme de menus.

Dans la première façon de procéder, le choix des informations à afficher se fait avant compilation, au début du fichier *Debug.cpp*. On retrouve au début de celui-ci toute une série d'instructions de préprocesseur :

```
#define CDebugOdoAction      true
#define CDebugOdoBezier      false
#define CDebugOdoPower       false
#define CDebugOdoPosition    true
#define CDebugOdoCoder       false
#define CDebugOdoSpeed       false
#define CDebugOdoPRegulParam false
#define CDebugOdoSRegulParam false
#define CDebugOdoStopEvent   true
#define CDebugOdoCycleTime   true
#define CDebugSwitches       false
#define CDebugCalibration    false
#define CDebugServo          false
#define CDebugSequenceurS    true
#define CDebugPickUpS      true
#define CDebugGoToColDS      true
#define CDebugGoToFrozCS     true
#define CDebugCatapult       true
#define CDebugPutS          false
#define CDebugHooverS      false
#define CDebugParams         false
```

Les paramètres suivis de *true* seront affichés, ceux suivis de *false* ne le seront pas.

Nous avons notamment utilisé ce mode de débogage pour le réglage des PID et PI : on sélectionne les informations utiles (puissance des moteurs, vitesse...) et celles-ci seront affichées dans la console série. L'affichage aura lieu une fois pas cycle, ce qui nous permet d'avoir une évolution au cours du temps des paramètres. Il est ensuite possible d'exporter les informations affichées dans la console vers Excel, ce qui nous permet de pouvoir tracer des graphiques pour l'analyse des résultats (cf. chapitre 18).



La seconde manière de procéder, par l'utilisation de menus, nécessite d'activer l'affichage desdits menus. Pour ce faire, il suffit de décommenter une ligne dans le fichier *Debug.h* et ensuite de compiler :

```
#define __TEST__
```

L'utilisation des menus est assez intuitive. Pour notre part, nous nous sommes principalement servi de ceux-ci pour contrôler le bon fonctionnement des microswitches et des moteurs auxiliaires et pour déterminer les positions angulaires optimales des servomoteurs (définition des positions ouvertes et fermées).

Notons au passage que lorsqu'on utilise l'interface sous forme de menus, il est possible d'ouvrir une deuxième console (sur le second port série) dans laquelle s'afficheront les informations cycle par cycle, comme dans le premier mode d'utilisation de l'interface de débogage.

21.4 Plate-forme SolidEdge des déplacements

Lors de la génération de nos trajectoires, nous avons cherché à faire le moins de déplacements possible dans le but de gagner le plus de temps mais également parce que notre asservissement en position n'étant pas précis, il fallaitt essayer de minimiser les trajectoires. Dès lors, pour éviter les calculs inutiles lors de la détermination des points de passage des trajectoires, nous avons modélisé la table de jeu en 2D sous SolidEdge (cf figures 21.2 et 21.3).

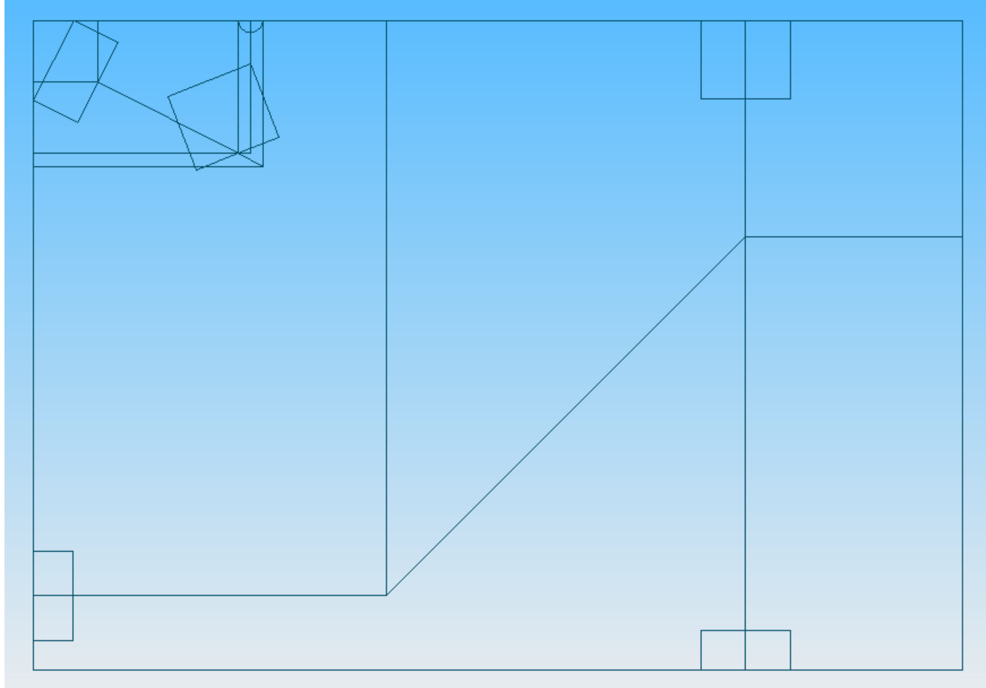
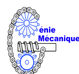


Fig. 21.2 – Plate-forme 2D SolidEdge pour la détermination des trajectoires



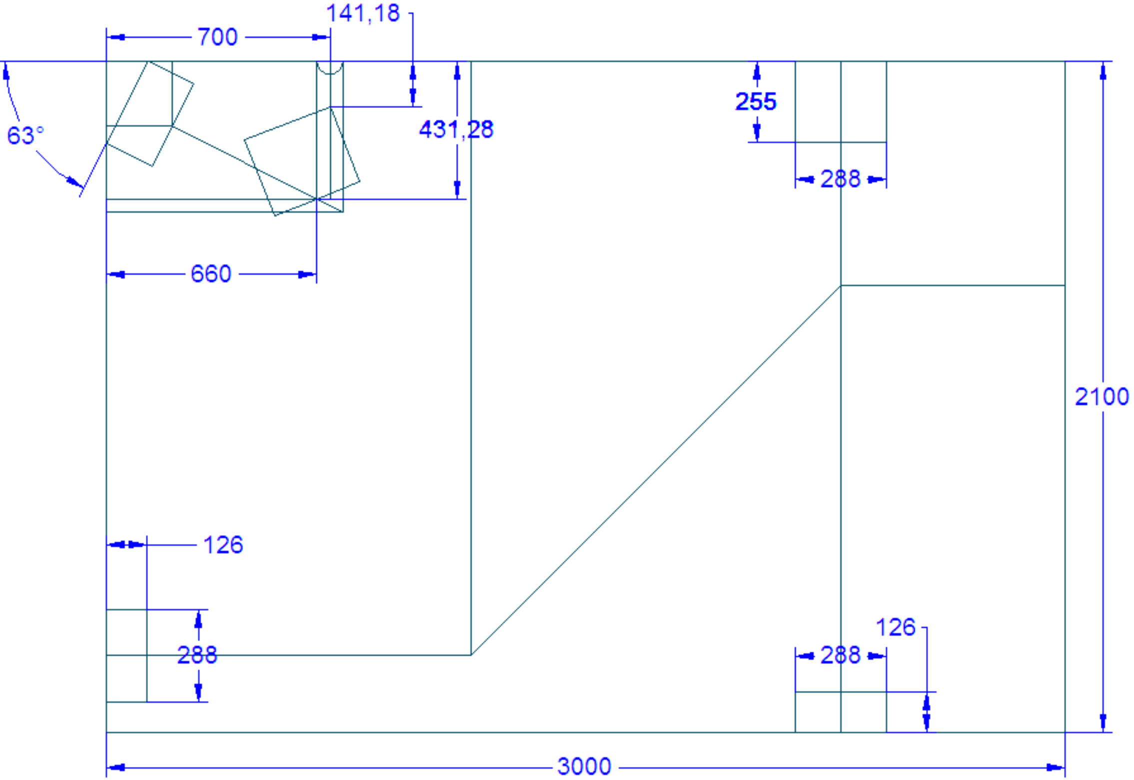


Fig. 21.3 – Plate-forme 2D SolidEdge avec dimensions pour la détermination des trajectoires

Grâce à cette plate-forme graphique, nous avons facilement pu déterminer une trajectoire qui nous permettait d'effectuer le premier virage au plus juste du distributeur de balles de couleur et dès lors repousser toutes les balles placées sur la table aux abords du distributeur, évitant ainsi à notre robot d'avaler une balle qui ne faisait pas partie de notre objectif. Cette modélisation nous a également permis des adaptations rapides de nos trajectoires en cours de coupes de Belgique et d'Europe.

Chapitre 22

Conclusion de la partie informatique

Pour conclure, nous dirons que ce projet a été une expérience très enrichissante, malgré les ennuis techniques rencontrés. C'était d'autant plus enrichissant que la programmation d'un robot autonome ne fait pas partie de notre formation de base d'ingénieur civil en mécanique.

Malgré nos faibles moyens, nous sommes quand même arrivés à un résultat satisfaisant. Nous avons réussi à adapter le code de l'année dernière à nos besoins et nous y avons même apporté quelques améliorations. Notre travail constitue ainsi une bonne base pour les années futures. Certes il n'est pas parfait, mais son efficacité et sa robustesse ne sont plus à prouver.

Grâce à ce projet, nous avons pu apprendre de nombreuses notions de programmation dont plusieurs propres à la robotique, comment réaliser un asservissement efficace, comment implémenter une stratégie de manière optimale... Nous avons même développé une manoeuvre d'évitement qui jusque là était inexistante dans le code *Nucléo*. Bien qu'efficace, cette manoeuvre ne constitue qu'une ébauche d'un évitement plus général et il serait intéressant que celui-ci soit développé dans les années à venir.

Pour terminer, nous retiendrons que ce projet a été avant tout une aventure humaine. Il nous a permis de rencontrer de nombreuses personnes avec qui nos relations furent aussi variées qu'enrichissantes, notamment les autres équipes rencontrées lors des coupes de Belgique et d'Europe. D'ailleurs, c'est ce qui nous donne envie de continuer dans la robotique dans les années à venir, très probablement au sein d'une équipe d'anciens de la Faculté.

Chapitre 23

Conclusions générales

Voici qu'arrive l'heure de faire une dernière fois le point et de tirer les conclusions.

Pour la première fois, nous avons pu réaliser le fruit de notre travail de conception, de design et d'écriture. Nous avons mis au point un robot répondant à un cahier des charges très strict et qui a fait ses preuves lors de la compétition belge et européenne.

Mais avant toute chose, ce projet fut un exemple parfait de ce que sera notre vie après le passage à la Faculté. Gestion de groupe, de planning, prises de décision, expérimentations, validations, commandes, coups de téléphone, envois d'e-mails, réunions de crise, état d'avancement,... ce projet nous a plongé totalement dans le monde de l'ingénieur moderne. Ce fut pour nous tous une expérience inédite et formidable dont peu d'étudiants en Sciences de l'Ingénieur peuvent se prévaloir.

De plus, nous avons également appris à nous servir de nos mains en complément de nos méninges. Les blessures, usures, décharges et autres coups ne sont que fadaises par rapport à la satisfaction ressentie une fois l'objectif atteint.

Au niveau des résultats, la Mons Polytech Team s'est hissée sur le podium belge en décrochant une magnifique troisième place. Après des débuts quelque peu tâtonnants dus aux aléas propres au concours et aux caprices de la robotique, nous avons pu à plusieurs reprises observer les performances de notre machine. La "petite finale" fut un exemple de la parfaite cohésion des trois pôles. En effet, le robot a exécuté à merveille l'entièreté des actions pour lesquelles il avait été conçu : sortir du carré de départ, avaler les 5 balles de sa couleur, traverser la table, éviter l'adversaire et déposer les balles dans le conteneur standard. Ce match de 90 secondes a pu démontrer la fiabilité des systèmes mécaniques conçus, la capacité de détecter l'adversaire et de l'éviter et l'exactitude du code informatique devant gérer les machines d'états. Pour les trois pôles, mais avant tout pour l'équipe au complet, ce fut le plus bel exemple de la réussite de ce projet de longue haleine.

Epilogue

Notre robot ayant décroché la troisième place, nous avons eu la chance de participer aux finales mondiales d'Eurobot 2008. Ce voyage de 5 jours restera pour nous tous une aventure formidable et nous tenions à remercier chaleureusement la Faculté et nos sponsors.

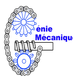
Le robot se classe finalement 27^{e}mondial sur 53 équipes invitées. Nous terminons ex-aequo avec nos collègues de l'U.Lg et les champions de Belgique, l'Helb Inraci terminent second du concours.

Annexe A

Aide-mémoire en électronique

Comment fabriquer une carte électronique ?

1. Dessiner le routage sous Eagle (Conseil : insérer le nom de la carte)
2. Imprimer ce routage sur papier calque (Ne pas oublier de retirer les traits relatifs aux composants)
3. Découper la plaque d'époxy à bonne dimension
4. Retirer le papier collant de l'époxy et fixer le papier calque. (étape rapide !)
5. Insérer la plaque dans l'insoleuse et la fermer à double tour (étape rapide !) (papier calque face aux néons)
6. Insoler pendant 1'15" (1'20" max) (étape précise !)
7. Opération de révélation : retirer la pellicule déposée sur la carte une fois insolée dans un bain révélateur (1 dose de révélateur pour 4 doses d'eau).
8. Rincer à l'eau claire et frotter avec de l'essuie tout pour retirer la pellicule.
9. Mettre la plaque dans la graveuse. Temps = f(vieillesse de l'époxy, et de l'acide). But : faire partir le Cu. Attention, lors de cette étape, on croit que le cuivre ne part pas, mais en fait, il part, une fois que le procédé est lancé, il faut être juste à côté et vérifier souvent...
10. Rinçage : Essuyer à l'essuie-tout et rincer à l'eau
11. Passer la plaque à l'acétone (Mettre de l'acétone sur le papier essuie-tout)
12. Gratter les routes mal gravées avec un stylet et vérifier s'il n'y a bien aucun contact entre les différentes pistes grâce au Ohmètre.
13. Percer la plaque là où il faut.
14. Disposer les composants SMD en premier, ensuite les autres.
15. Admirer le résultat et se sentir fier.



Comment utiliser le bloc HF/ultrasons ?

Principe

Le bloc inséré dans le robot émet des ondes hautes fréquences dans toutes les directions. La balise reçoit ces ondes et transmet, une fois le signal capté, des ondes ultrasons dans toutes les directions également. Le récepteur ultrason placé sur le mât du robot ne peut détecter que les ondes ultrasoniques face à lui. S'il capte une onde, il envoie un signal au bloc inséré dans le robot. Si la LED rouge s'allume, cela traduit que la balise (donc, le robot adverse) est trop près. Un signal est donc envoyé vers la carte principale via la carte de gestion des switches et peut être récupéré dans le code.

Mode d'emploi

1. placer le microcontroleur PIC correspondant à la bonne distance dans le bloc (attention, bien veiller au détrompeur)
2. fixer le bloc dans le robot
3. connecter (via JST) le bloc à la carte de gestion des switches
4. placer le récepteur ultrason sur le mât
5. connecter le récepteur ultrason au bloc
6. allumer la balise (interrupteur central) et la placer sur le mât du robot adverse
7. adapter le code en fonction de l'adresse du logement pour switch choisi

Remarque

Il est possible de récupérer l'information *distance* via le bloc inséré dans le robot. En effet, il suffit de connecter un simple fil au trou laissé à côté du microcontroleur PIC. Le train d'onde reçu traduit la distance et permet le débuggage en particulier lors de la calibration de la loi du microcontroleur relative à la distance séparant l'émetteur et le récepteur ultrason.

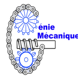
Annexe B

Détection du signal infrarouge

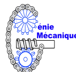
Le code suivant est celui qui a été implémenté de manière à effectuer la détection de la balise infrarouge. Grâce à celui-ci, le microcontrôleur[1] renvoie un "1" ou un "0" logique à la carte de gestion des switches selon que l'on détecte le signal ou pas.

```
#include <avr/interrupt.h>
#include <avr/io.h>

#include <stdint.h>

// Déclaration d'une variable globale de type spécial (8 bits)
volatile uint8_t irdata;

// Fonction qui affecte la variable globale definie ci dessus
 voir p44 de la datasheet)

ISR(USART_RXC_vect)
{
        irdata = UDR;
}

void syst_init()
{
// DDRx traduit : Direction Data Register numéro x
        DDRA = 0xff;
        DDRB = 0xff;
        DDRC = 0xff;
        DDRD = 0x00;
// 0xff : ca met tout en sortie
// Dans notre cas, c'est le DDRC
// (auquel est associé PORTC) qui va
//  servir de sortie

// Remarquons que si on a 0x01, cela traduit
// qu'on associe une résistance de pull-up si on est entrée !
// (c-a-d si le DDRx correspondant en entrée)


        PORTA = 0x00;
        PORTB = 0x00;
        PORTC = 0x00;
        PORTD = 0x00;
}

// Initialisation du timer

void timer_init()
{
        TCCR0 = 0;
        // Attention, il ne faut pas lancer le timer ici , on l'initialise à zéro.
```

[1] il s'agit ici d'un ATMéga16



```
        // Cette ligne sert à rien, c'est juste didactique (Page 80 de la datasheet)
}

/* Initialisation les paramètres  */
void usart_init (void)
{

        UCSRB = (1 << RXCIE) | (0 << TXCIE) | (0 << UDRIE) | (1 << RXEN) | (0 << UCSZ2) | (0 << RXB8);
        // On met 1 décalé du nombre associé à RXEN
        // (ici 4) vers la gauche idem pour les 2 autres
        UCSRC = (1 << URSEL) | (0 << UMSEL) | (0 << UPM1) | (0 << UPM0) | (0 << USBS) | (1 << UCSZ1) | (1 << UCSZ0);
        // UMSEL Pas vraiment utile, car on est asynchrone naturellement
        // attention, UCSZ1 et UCSZ0 doivent être pris à 1 mais il faut aussi que
        // UCSZ2 (qui est dans UCSRB !) soit initialisé à 0.

        /* Initialisation du Baud */
        // On doit traduire 832, via l'octet de poids fort et l'octet de poids faible.

        UBRRH = 3;
        // 0011 en binaire, pas de soucis du fait que URSEL soit a 1 dans UCSRC car en fait,
        // si on dit qu'on impose 3 , on impose en fait
        // 00000011 en binaire
        UBRRL = (1 << 6);
        // 1 décalé de 6 vers la gauche


        /* Autorisation aux fonction interrupt d'intervenir */
        sei();
}


void usart_recept()
{

        char current_value = 0 ;
        // Valeur courante (c-a-d de l'itération qui est en train de se faire)
        char last_value = 0;
        // Valeur de la dernière itération complète réalisée
        char stable_value = 0;
        // Signal ayant l'air de se confirmer.

        while(1)
        {
                current_value = irdata;
        /*Test de comparaison pour savoir si on n'est pas en train de changer de signal */
                if (current_value != last_value)
                {
                        TCCR0 = (1 <<  CS02) | (1 << CS00);
                        // Apparement, le signal est en train de changer, donc, on lance
                        // un timer pour savoir si, après
                        // la valeur du timer, on aura encore le même signal ou pas.
                        TCNT0 = 0;
                        // En effet, si le signal change tout le temps, le timer se réinitialise tout le temps.
                        TIFR = (1 << TOV0);
                        // Le timer sera écoulé (et rentrera dans la boucle suivante) si et seulement si le signal
                        // est resté stable pendant une période assez longue.
                        // En fait, quand on initialise le registre TCCR0, on "lance" le timer (divisé par 1024)
                        // TCNT0 = 0 traduit le fait que l'on commencer à compter à partir de 0.
                        // En fait, TCNT0 est le registre qui s'incrémente de 1 à chaque coup d'horloge.
                }

                if ((TIFR & (1 << TOV0)) != 0)
                        // en fait, c'est : if (timer écoulé). 100 est pris au hasard.
                        // PRATIQUEMENT, on fait jamais ca, car alors, plus le code
                        // sera long, plus l'on mettra du temps a atteindre les 100...
                        // Donc, il faut créer un timer indépendant de la longueur du code.
                        // C'est donc avec le registre TCNT0 (p82) que l'on va travailler.
                        // La difficulté est de savoir combien de temps attendre pour être
                        // sur que le signal soit stable.
                        // Il faut donc faire un calcul (mais on va dire que le temps est
                        // écoulé dès que le bit d'overflow passe à 1, donc, on va pas travailler avec TCNT0)
                {
```



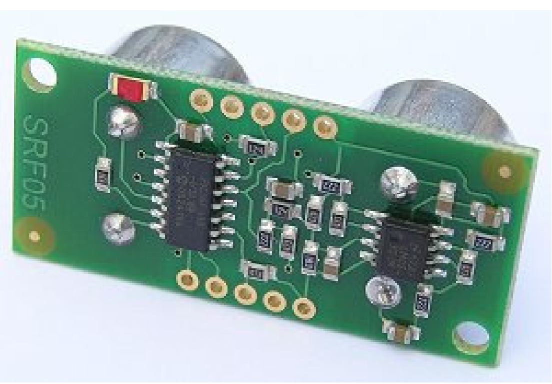
```
                        stable_value = current_value;
                        // On introduit un variable différente de irdata car cette variable
120                     // a une caractéristique supplémentaire. En effet, stable_value est
                        // un irdata STABLE, c-a-d que c'est un irdata
                        // qui dure., qui a la même valeur depuis un petit moment.
                }

125             last_value = current_value;
                        // Une fois les tests concernant la "stabilité" du signal effectué, on peut
                        // "sauvergarder" la valeur de la variable pour la prochaine itération...

                if (stable_value == 'A')
130                     // C'est bien sur stable_value que l'on place la condition et non plus sur irdata
                        // Remarquons que l'on ne rentre dans l'une de ces trois boucles que si le signal
                        // est stable!
                {
                        PORTC &= (0 << PC0);
135                     PORTB &= (0 << PB0);
                        PORTA |= (1 << PA0);
                }
                if (stable_value == 'B')
                {
140                     PORTA &= (0 << PA0);
                        PORTC &= (0 << PC0);
                        PORTB |= (1 << PB0);
                }
                if (stable_value == 'C')
145             {
                        PORTA &= (0 << PA0);
                        PORTB &= (0 << PB0);
                        PORTC |= (1 << PC0);
                }
150             else
                        // Si on capte du bruit ! On ne capte ni A, ni B, ni C... donc, on éteind tout !
                {
                        PORTA = 0;
                        PORTB = 0;
155                     PORTC = 0;
                }

        }
}
160

int main (void)
{
        syst_init();
165     timer_init();
        usart_init();
        usart_recept();

}
```

Annexe C

Datasheet du module ultrasonique SRF05

SRF05 - Ultra-Sonic Ranger

Technical Specification

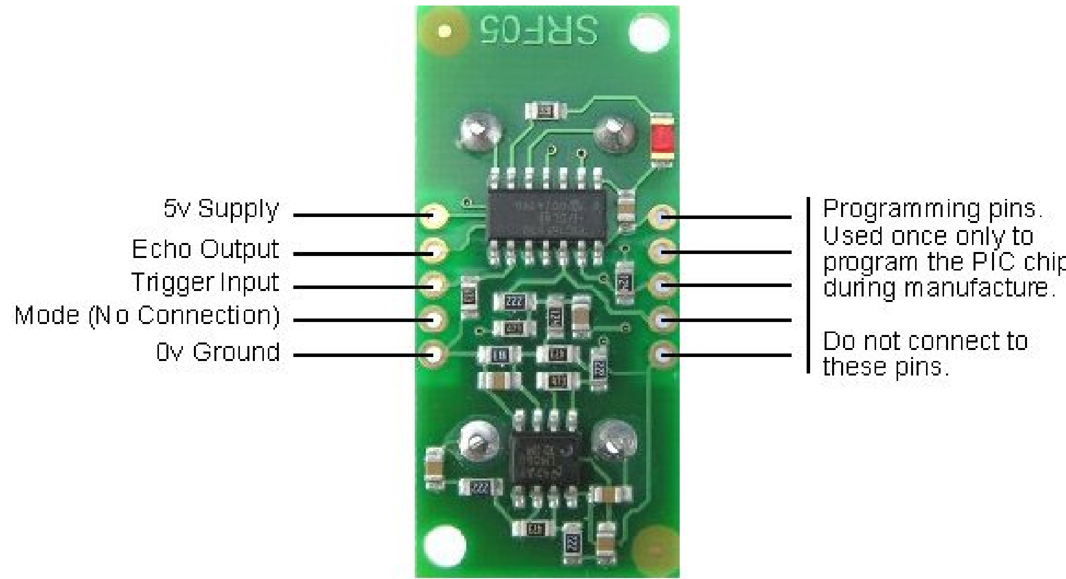


Introduction

The SRF05 is an evolutionary step from the SRF04, and has been designed to increase flexibility, increase range, and to reduce costs still further. As such, the SRF05 is fully compatible with the SRF04. Range is increased from 3 meters to 4 meters. A new operating mode (tying the mode pin to ground) allows the SRF05 to use a single pin for both trigger and echo, thereby saving valuable pins on your controller. When the mode pin is left unconnected, the SRF05 operates with separate trigger and echo pins, like the SRF04. The SRF05 includes a small delay before the echo pulse to give slower controllers such as the Basic Stamp and Picaxe time to execute their pulse in commands.

Mode 1 - SRF04 compatible - Separate Trigger and Echo

This mode uses separate trigger and echo pins, and is the simplest mode to use. All code examples for the SRF04 will work for the SRF05 in this mode. To use this mode, just leave the mode pin unconnected - the SRF05 has an internal pull up resistor on this pin.

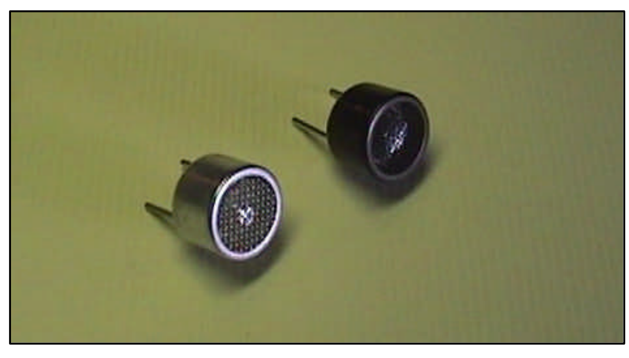


Connections for 2-pin Trigger/Echo Mode (SRF04 compatible)

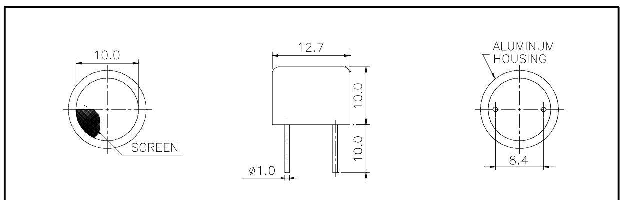
Annexe D

Datasheet de l'émetteur ultrasons utilisé

Air Ultrasonic Ceramic Transducers 400ST/R120



Dimensions: dimensions are in mm



Specification

400ST120	Transmitter
400SR120	Receiver
Center Frequency	40.0±1.0Khz
Bandwidth (-6dB) 400ST120	2.0Khz
400SR120	2.0Khz
Transmitting Sound Pressure Level at 40.0Khz; 0dB re 0.0002µbar per 10Vrms at 30cm	115dB min.
Receiving Sensitivity at 40.0Khz 0dB = 1 volt/µbar	-67dB min.
Capacitance at 1Khz ±20%	2400 pF
Max. Driving Voltage (cont.)	20Vrms
Total Beam Angle -6dB	85° typical
Operation Temperature	-30 to 80°C
Storage Temperature	-40 to 85°C

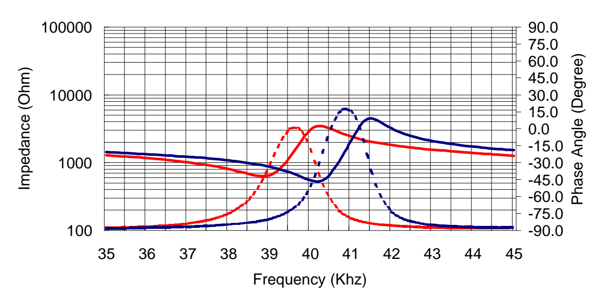
All specification taken typical at 25°C
Closer frequency tolerance can be supplied upon request.

Model available:

1	400ST/R120	Aluminum Housing
2	400ST/R12B	Black Al. Housing

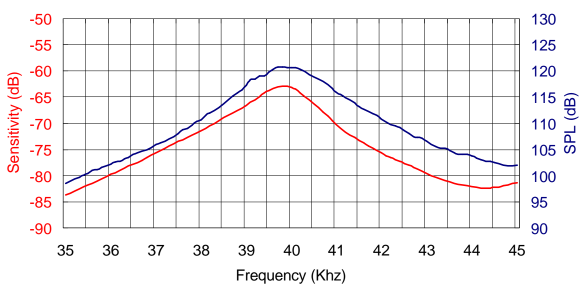
Impedance/Phase Angle vs. Frequency

Tested under 1Vrms Oscillation Level
400SR120 Impedance
400SR120 Phase
400ST120 Impedance
400ST120 Phase

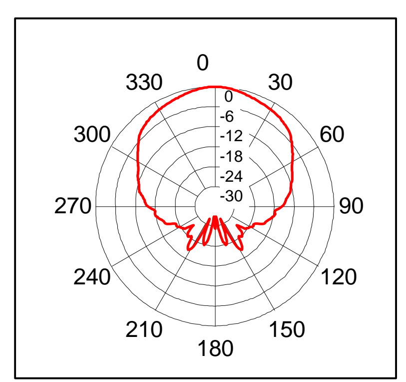


Sensitivity/Sound Pressure Level

Tested under 10Vrms @30cm



Beam Angle: Tested at 40.0Khz frequency



Annexe E

Couverture des émetteurs ultrasons

Le calcul de la distance au centre de la balise à partir de laquelle la couverture est totale relève de simples considérations trigonométriques. On peut successivement calculer les longueurs de segments représentés sur la figure E.1 ci-après :

$$c = \frac{r}{2} \cdot \cos(30) \tag{E.1}$$

$$h = \frac{r}{2} \cdot \sin(30) \tag{E.2}$$

$$H = c \cdot \tan\left(120 - \frac{A}{2}\right) \tag{E.3}$$

$$I = R + H - h \tag{E.4}$$

Après calcul, on trouve que, en utilisant des émetteurs US possédant un cône d'émission de

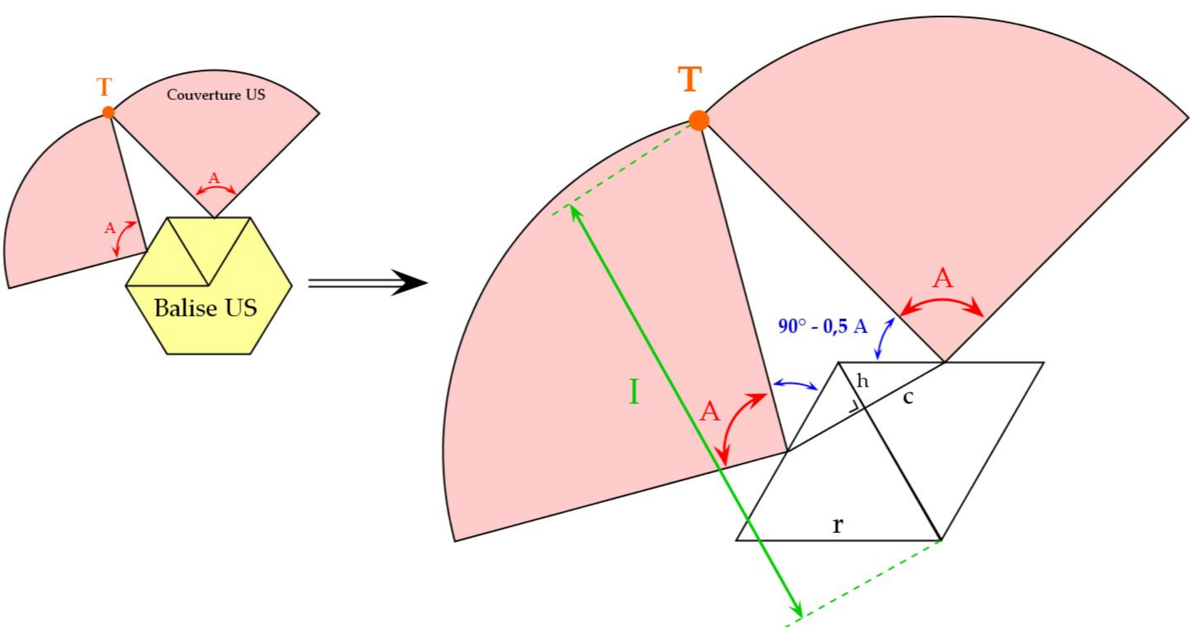
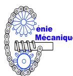


Fig. E.1 – Calcul de la distance minimale à partir de laquelle les émetteurs US assurent une couverture totale, compte tenu de leur cône d'émission

A=85°, on assure une couverture totale à 86,5 mm du centre de la balise. Or, la projection de notre robot sur la table est un carré de 150 mm sur 150 mm. Ce type d'émetteur convient donc



tout à fait. On peut trouver à la figure E.2 une représentation de la zone couverte totalement par le signal US (au delà de la ligne verte).On constate que cette frontière se situe clairement à l'intérieur du robot(lignes bleues) et ne pose donc aucun problème.

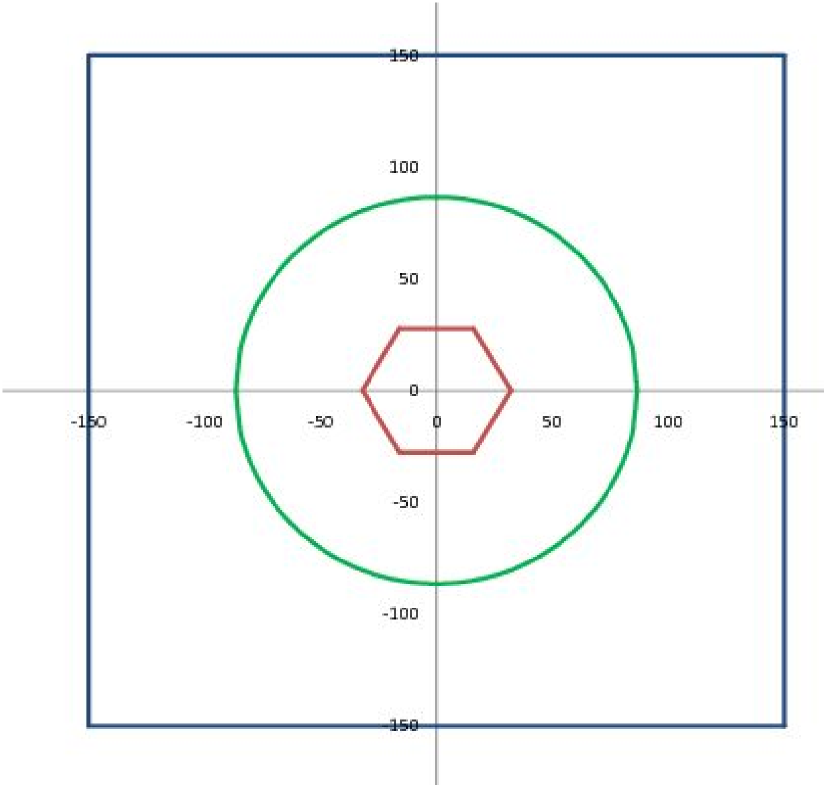


Fig. E.2 – Détermination de la couverture des signaux ultrason

Annexe F

Caractéristiques des moteurs et réducteurs MAXON utilisés pour les différents systèmes mécaniques

EC 22 ∅22 mm, à commutation électronique, 20 Watt, avec l'électronique intégrée, CE certifié

Type de connecteur:
MKF 13264-6-0-404
Stocko Elektronik GmbH

M 1:1

Programme Stock
Programme Standard
Programme Spécial (sur demande)

Numéros de commande

		Unité	201162	200863	201163	200864
Caractéristiques moteur						
	Valeurs à la tension nominale					
1	Tension nominale	V	24.0	24.0	24.0	24.0
2	Vitesse à vide	tr / min	35400	20500	16700	9590
3	Courant à vide	mA	123	57.4	44.0	22.4
4	Vitesse nominale	tr / min	32200	16500	12600	5900
5	Couple nominal (couple permanent max.)	mNm	8.22	14.9	15.3	16.0
6	Courant nominal (courant permanent max.)	A	1.36	1.34	1.11	0.673
7	Couple de démarrage	mNm	16.0	27.8	34.0	46.6
8	Courant de démarrage	A	2.50	2.50	2.50	1.98
9	Rendement max.	%	85	85	83	80
	Caractéristiques					
10	Résistance aux bornes (phase-phase)	Ω	p.i.	p.i.	p.i.	p.i.
11	Inductivité (phase-phase)	mH	p.i.	p.i.	p.i.	p.i.
12	Constante de couple	mNm / A	6.41	11.1	13.6	23.6
13	Constante de vitesse	tr / min / V	1490	860	701	405
14	Pente vitesse / couple	tr / min / mNm	277	207	224	208
15	Constante de temps mécanique	ms	8.69	6.52	7.03	6.55
16	Inertie du rotor	gcm^2	3.00	3.00	3.00	3.00

Spécifications

Données thermiques

17	Résistance therm. carcasse/air ambiant	10 K / W
18	Résistance therm. bobinage/carcasse	2.0 K / W
19	Constante de temps therm. bobinage	4.93 s
20	Constante de temps therm. du moteur	300 s
21	Température ambiante	-20 ... +100°C
22	Température max. de bobinage	+125°C

Données mécaniques (roulements préchargés)

23	Nombre de tours limite	50000 tr / min
24	Jeu axial sous charge axiale < 5 N	0 mm
	> 5 N	max. 0.14 mm
25	Jeu radial	préchargé
26	Charge axiale max. (dynamique)	4 N
27	Force de chassage axiale max. (statique)	60 N
	(statique, axe soutenu)	250 N
28	Charge radiale max. à 5 mm de la face	16 N
	Température maximale de l'électronique	+125°C
	(la charge maximale du moteur est limité par l'électronique.)	

Autres spécifications

29	Nombre de paires de pôles	1
30	Nombre de phases	3
31	Poids du moteur	85 g

Les caractéristiques moteur du tableau sont des valeurs nominales.

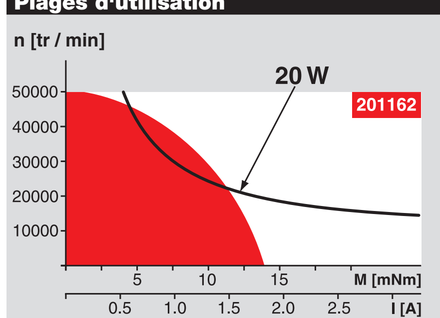
⚠ **Attention**
Le non respect de la polarité entraînera la destruction de l'électronique de commande

⚠ **Attention:** Le sens de rotation (DIR) ne doit être inversé qu'à l'arrêt du moteur sinon l'électronique risque d'être détruite.

Connexions

vert	$+V_{CC}$ 10 ... 50 VDC	Pin 1
bleu	GND	Pin 2
violet	Disable	Pin 3
gris	Direction	Pin 4

Plages d'utilisation



Légende

Plage de fonctionnement permanent
Compte tenu des resistances thermiques (lignes 17 et 18) la température maximum du rotor peut être atteinte au valeur nominal de couple et vitesse et à la température ambiante de 25°C.
= Limite thermique.

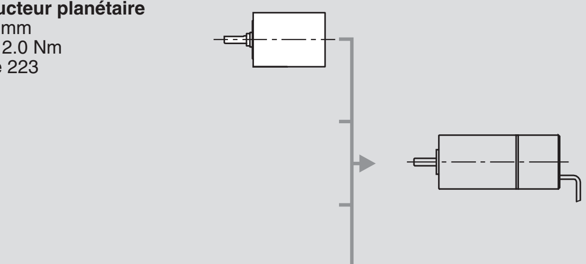
Fonctionnement intermittent
La surcharge doit être de courte durée.

Puissance conseillée

Construction modulaire maxon

Aperçu à la page 16 - 21

Réducteur planétaire
∅22 mm
0.5 - 2.0 Nm
Page 223

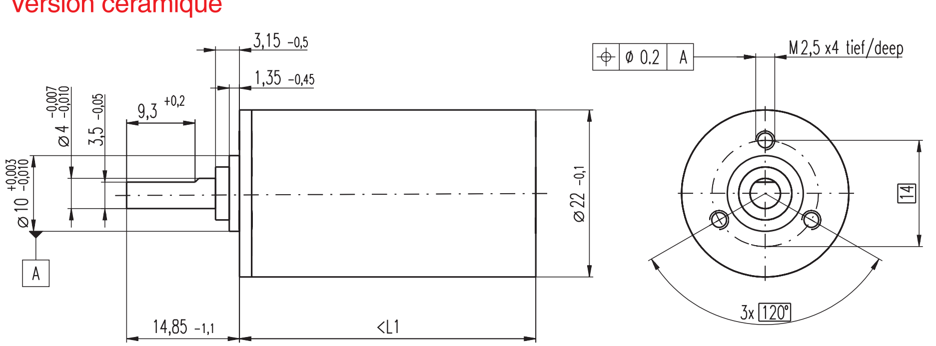


Edition Juillet 2007 / Modifications réservées

Réducteur planétaire GP 22 C ⌀22 mm, 0.5 - 2.0 Nm

Version céramique

maxon gear



M 1:1

Données techniques

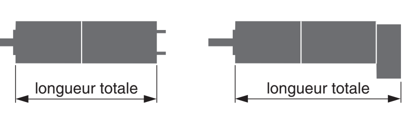
Réducteur planétaire	taille droite
Arbre de sortie	acier inoxydable, trempé
Palier de sortie	roulements à billes
Jeu radial à 10 mm de la face	max. 0.2 mm
Jeu axial	max. 0.2 mm
Charge radiale max. à 10 mm de la face	70 N
Charge axiale max.	100 N
Force de chassage max.	100 N
Sens de rotation entrée / sortie	=
Vitesse d'entrée conseillée	< 8000 tr / min
Plage de température conseillée	-20 ... +100°C
Domaine étendu en option	-35 ... +100°C

Programme Stock
Programme Standard
Programme Spécial (sur demande)

Numéros de commande

		143971	143974	143980	143986	143990	143996	144002	144004	144011	144017	144023
Données sur les réducteurs												
1 Rapport de réduction		3.8 : 1	14 : 1	53 : 1	104 : 1	198 : 1	370 : 1	590 : 1	742 : 1	1386 : 1	1996 : 1	3189 : 1
2 Rapport de réduction exact		15/4	225/16	3375/64	87723/845	50625/256	10556001/28561	59049/100	759375/1024	158340015/114244	285012027/142805	1594323/500
3 Diamètre max. de l'arbre du moteur	mm	4	4	4	3.2	4	3.2	4	4	3.2	3.2	4
Numéros de commande		143972	143975	143981	143987	143991	143997	144003	144006	144012	144018	144024
1 Rapport de réduction		4.4 :1	16 : 1	62 : 1	109 : 1	231 : 1	389 : 1	690 : 1	867 : 1	1460 : 1	2102 : 1	3728 : 1
2 Rapport de réduction exact		57/13	855/52	12825/208	2187/20	192375/832	263169/676	1121931/1625	2885625/3328	3947535/2704	7105563/3380	30292137/8125
3 Diamètre max. de l'arbre du moteur	mm	3.2	3.2	3.2	4	3.2	3.2	3.2	3.2	3.2	3.2	3.2
Numéros de commande		143973	143976	143982	143988	143992	143998	144005	144007	144013	144019	144025
1 Rapport de réduction		5.4 : 1	19 : 1	72 : 1	128 : 1	270 : 1	410 : 1	850 : 1	1014 : 1	1538 : 1	2214 : 1	4592 : 1
2 Rapport de réduction exact		27/5	3249/169	48735/676	41553/325	731025/2704	6561/16	531441/625	10965375/10816	98415/64	177147/80	14348907/3125
3 Diamètre max. de l'arbre du moteur	mm	2.5	3.2	3.2	3.2	3.2	4	2.5	3.2	4	4	2.5
Numéros de commande			143977	143983	143989	143993	143999		144008	144014	144020	
1 Rapport de réduction			20 : 1	76 : 1	157 : 1	285 : 1	455 : 1		1068 : 1	1621 : 1	2458 : 1	
2 Rapport de réduction exact			81/4	1215/16	19683/125	18225/64	5000211/10985		273375/256	601692057/371293	135005697/54925	
3 Diamètre max. de l'arbre du moteur	mm		4	4	2.5	4	3.2		4	3.2	3.2	
Numéros de commande			143978	143984		143994	144000		144009	144015	144021	
1 Rapport de réduction			24 : 1	84 : 1		316 : 1	479 : 1		1185 : 1	1707 : 1	2589 : 1	
2 Rapport de réduction exact			1539/65	185193/2197		2777895/8788	124659/260		41668425/35152	15000633/8788	3365793/1300	
3 Diamètre max. de l'arbre du moteur	mm		3.2	3.2		3.2	3.2		3.2	3.2	3.2	
Numéros de commande			143979	143985		143995	144001		144010	144016	144022	
1 Rapport de réduction			29 : 1	89 : 1		333 : 1	561 : 1		1249 : 1	1798 : 1	3027 : 1	
2 Rapport de réduction exact			729/25	4617/52		69255/208	2368521/4225		1038825/832	373977/208	63950067/21125	
3 Diamètre max. de l'arbre du moteur	mm		2.5	3.2		3.2	3.2		3.2	3.2	3.2	
4 Nombre d'étages		1	2	3	3	4	4	4	5	5	5	5
5 Couple permanent max.	Nm	0.5	0.6	1.2	1.2	1.8	1.8	1.8	2.0	2.0	2.0	2.0
6 Couple intermittent max. admissible	Nm	0.8	0.9	1.9	1.9	2.7	2.7	2.7	3.0	3.0	3.0	3.0
7 Rendement max.	%	84	70	59	59	49	49	49	42	42	42	42
8 Poids	g	42	55	68	68	81	81	81	94	94	94	94
9 Jeu moyen à vide	°	1.0	1.2	1.6	1.6	2.0	2.0	2.0	2.0	2.0	2.0	2.0
10 Moment d'inertie	gcm^2	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
11 Longueur du réducteur L1*	mm	25.4	32.2	39.0	39.0	45.8	45.8	45.8	52.6	52.6	52.6	52.6

*est L1 - 2.8 mm pour le calcul de la longueur totale



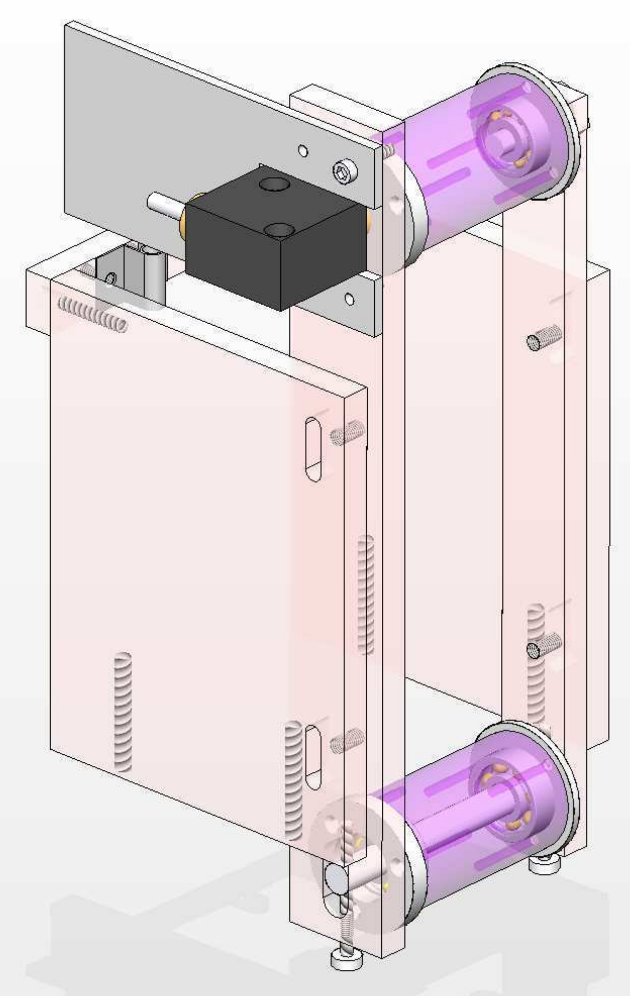
Combinaison

+ Moteur	Page	+ Capteur / frein	Page	Longueur totale [mm] = longueur du moteur + longueur du réducteur + (capteur / frein) + pièces de montage										
A-max 19	105/106			51.6	58.4	65.2	65.2	72.0	72.0	72.0	78.8	78.8	78.8	78.8
A-max 19, 1.5 W	106	MR	247/248	56.7	63.5	70.3	70.3	77.1	77.1	77.1	83.9	83.9	83.9	83.9
A-max 19, 1.5 W	106	Enc 22	253	66.0	72.8	79.6	79.6	86.4	86.4	86.4	93.2	93.2	93.2	93.2
A-max 19, 1.5 W	106	MEnc 13	262	59.1	65.9	72.7	72.7	79.5	79.5	79.5	86.3	86.3	86.3	86.3
A-max 19, 2.5 W	107/108			54.2	61.0	67.8	67.8	74.6	74.6	74.6	81.4	81.4	81.4	81.4
A-max 19, 2.5 W	108	MR	247/248	58.5	65.3	72.1	72.1	78.9	78.9	78.9	85.7	85.7	85.7	85.7
A-max 19, 2.5 W	108	Enc 22	253	68.6	75.4	82.2	82.2	89.0	89.0	89.0	95.8	95.8	95.8	95.8
A-max 19, 2.5 W	108	MEnc 13	262	61.7	68.5	75.3	75.3	82.1	82.1	82.1	88.9	88.9	88.9	88.9
A-max 22	109-112			54.6	61.4	68.2	68.2	75.0	75.0	75.0	81.8	81.8	81.8	81.8
A-max 22	110/112	MR	247/248	59.6	66.4	73.2	73.2	80.0	80.0	80.0	86.8	86.8	86.8	86.8
A-max 22	110/112	Enc 22	253	69.0	75.8	82.6	82.6	89.4	89.4	89.4	96.2	96.2	96.2	96.2
A-max 22	110/112	MEnc 13	262	61.7	68.5	75.3	75.3	82.1	82.1	82.1	88.9	88.9	88.9	88.9
RE-max 21	135/136			51.6	58.4	65.2	65.2	72.0	72.0	72.0	78.8	78.8	78.8	78.8
RE-max 21, 3.5 W	136	MR	247/248	56.7	63.5	70.3	70.3	77.1	77.1	77.1	83.9	83.9	83.9	83.9
RE-max 21	137/138			54.2	61.0	67.8	67.8	74.6	74.6	74.6	81.4	81.4	81.4	81.4
RE-max 21, 6 W	138	MR	247/248	58.5	65.3	72.1	72.1	78.9	78.9	78.9	85.7	85.7	85.7	85.7
RE-max 24	139-142			54.6	61.4	68.2	68.2	75.0	75.0	75.0	81.8	81.8	81.8	81.8
RE-max 24	140/142	MR	247/248	59.6	66.4	73.2	73.2	80.0	80.0	80.0	86.8	86.8	86.8	86.8

Edition Juillet 2007 / Modifications réservées

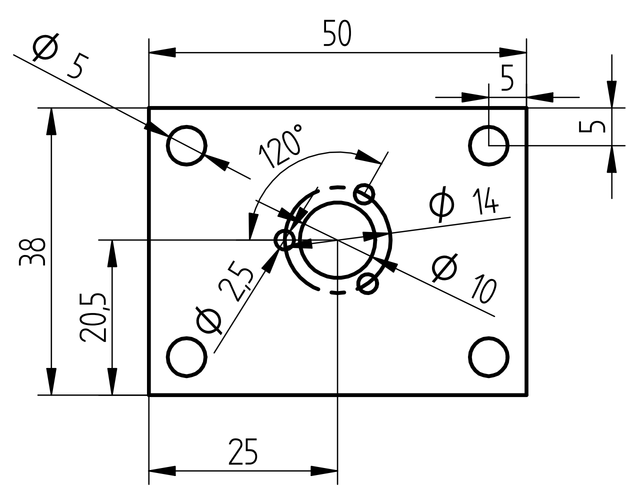
Annexe G

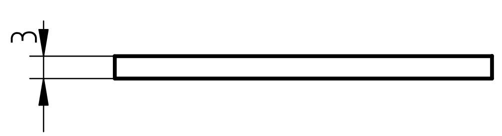
Plans de l'ascenseur



Annexe H

Plans du système d'avalement





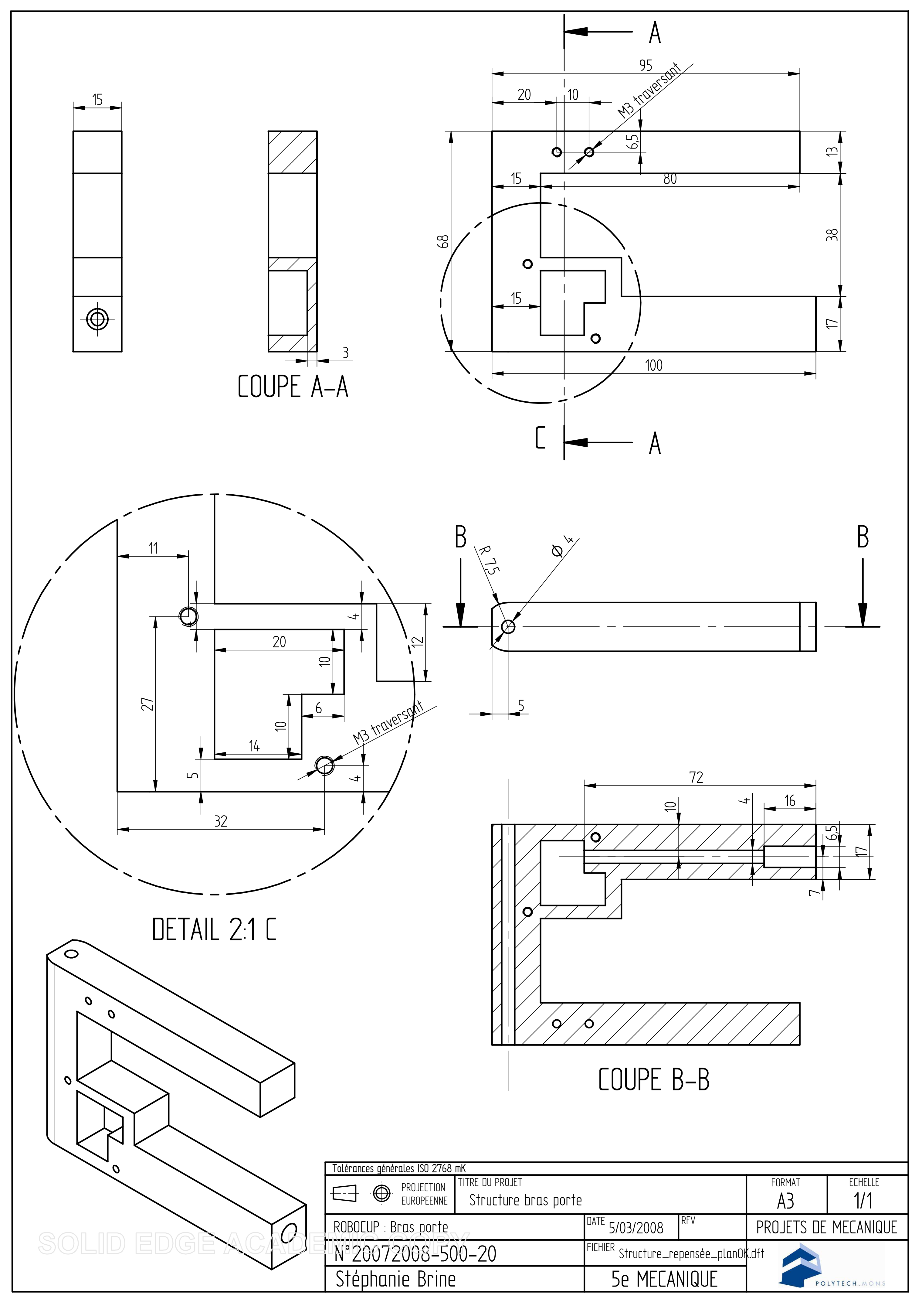
Tolérances générales ISO 2768 mK

PROJECTION EUROPEENNE	TITRE DU PROJET Support moteur avalement		FORMAT A4	ECHELLE 1/1
ROBOCUP : Avalement	DATE 3/06/2008	REV	PROJETS DE MECANIQUE	
N°20072008-500-27	FICHIER Support_moteur.dft		POLYTECH.MONS	
Stéphanie Brine	5e MECANIQUE			

SOLID EDGE ACADEMIC COPY

Annexe I

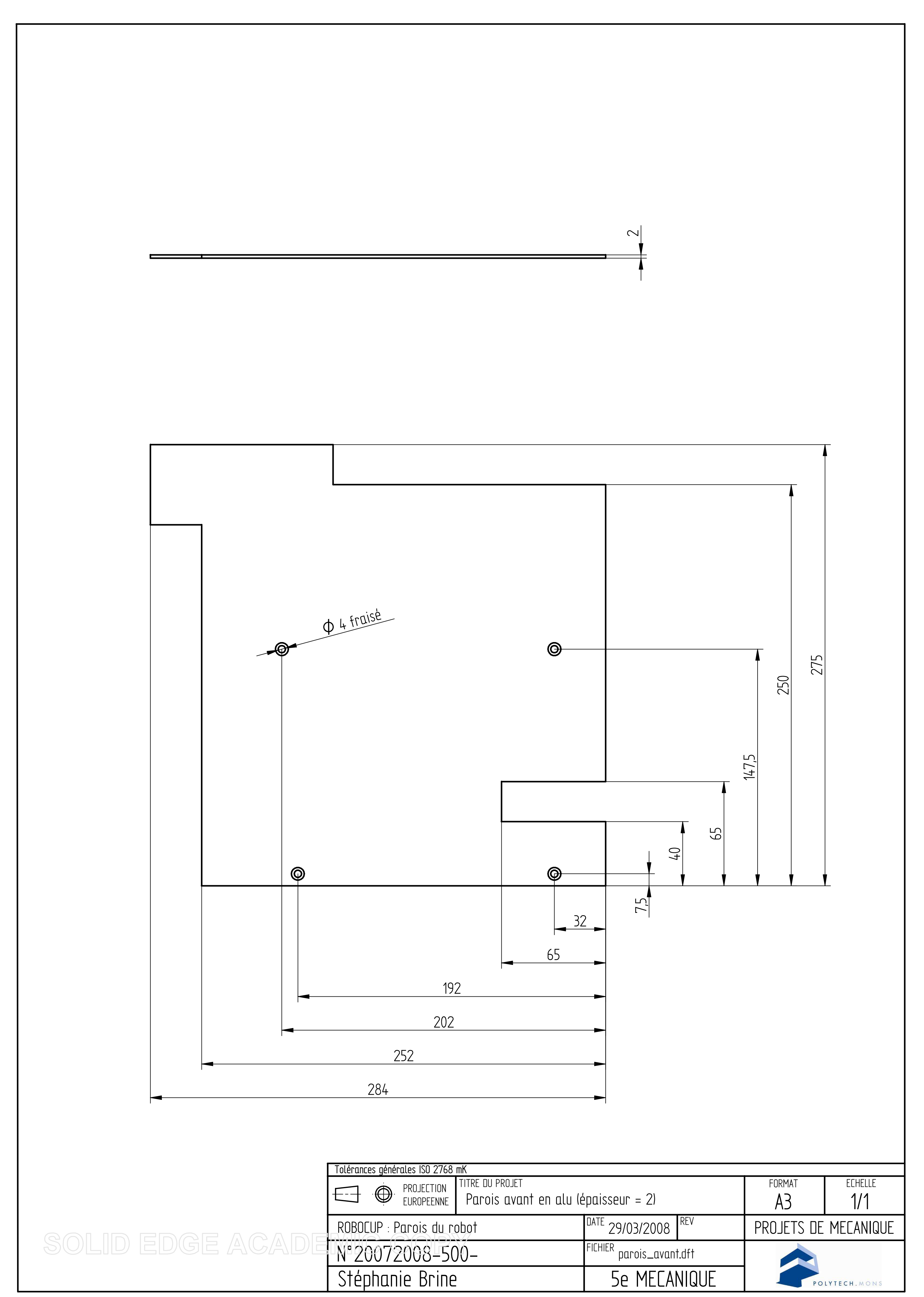
Plans des bras



SOLID EDGE ACADEMIC COPY

Annexe J

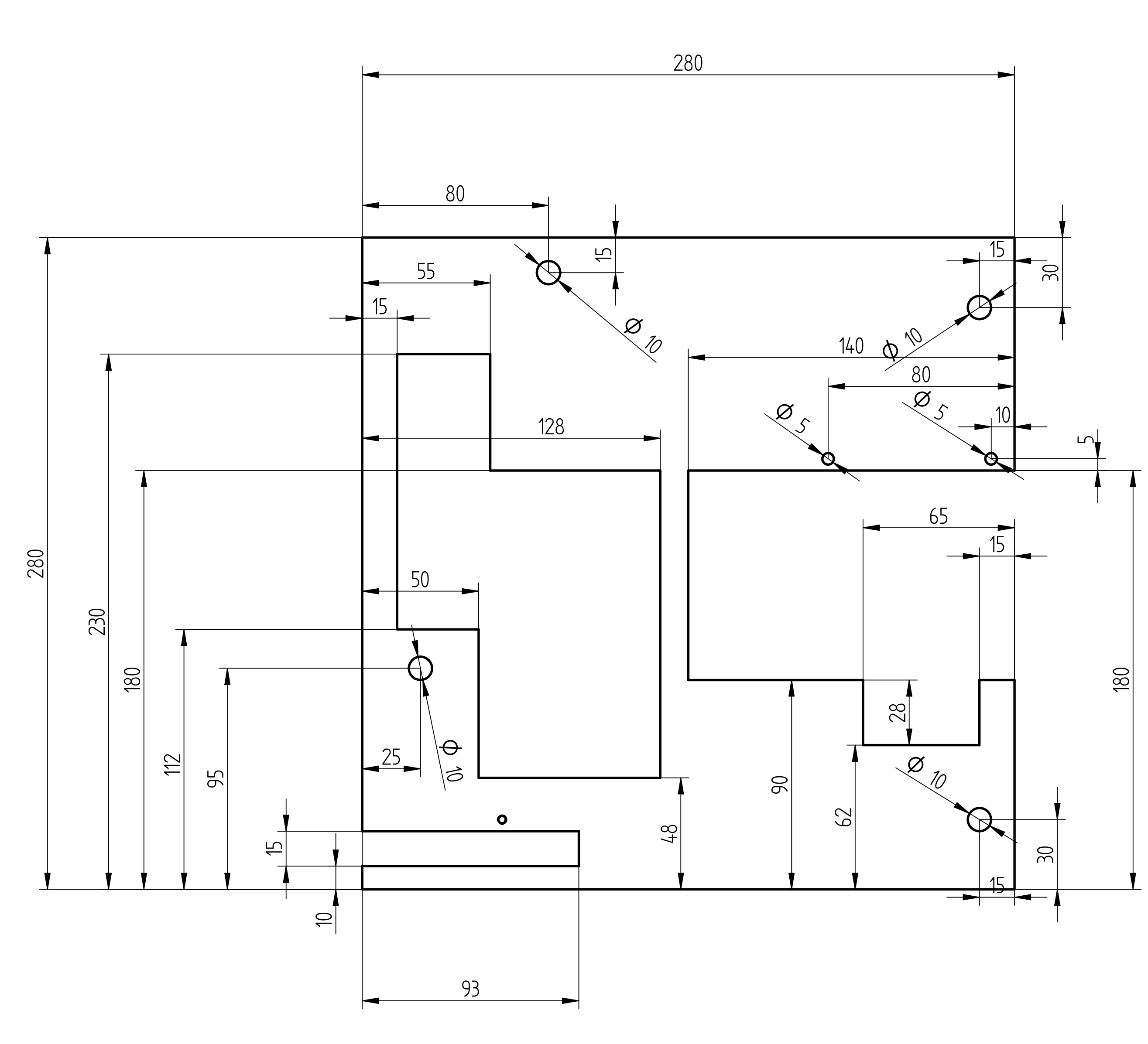
Plans des parois de finition du robot



SOLID EDGE ACADEMIC COPY

Annexe K

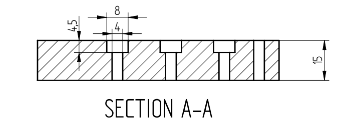
Plans des plate-formes de la struture



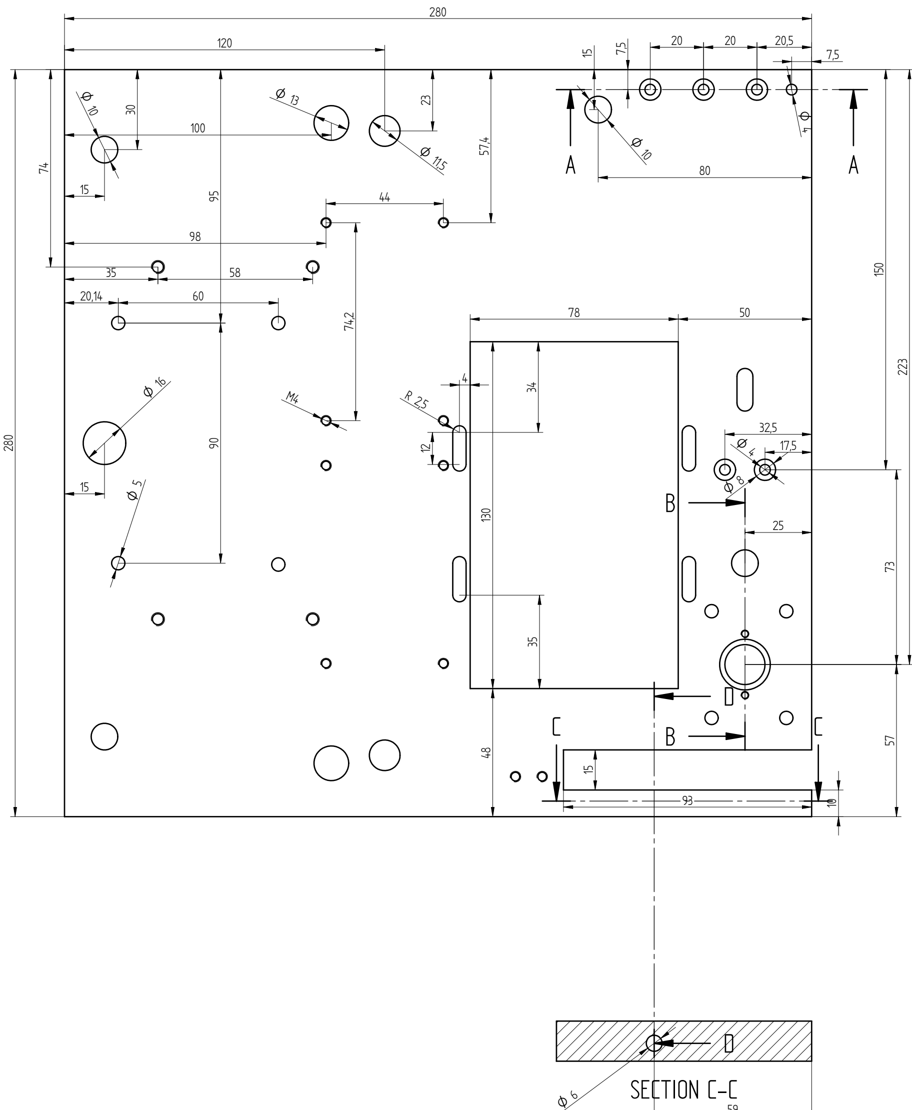
SOLID EDGE ACADEMIC COPY

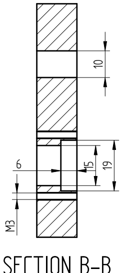
Tolérances générales ISO 2768 mK

PROJECTION EUROPEENNE	TITRE DU PROJET Plaque du haut		FORMAT A3	ECHELLE 1/2
ROBOCUP : structure	DATE 3/06/2008	REV	PROJETS DE MECANIQUE	
N°20072008-500-31	FICHIER plaque.dft		POLYTECH.MONS	
Stéphanie Brine	5e MECANIQUE			



SECTION A-A





SECTION B-B

SECTION D-D

6

SECTION C-C

ϕ 6

59

Tolérances générales ISO 2768 mK			
PROJECTION EUROPEENNE	TITRE DU PROJET Plate forme	FORMAT A1	ECHELLE 1/1
ROBOCUP : Structure	DATE 3/06/2008	REV	PROJETS DE MECANIQUE
N°20072008-500-40	FICHIER Plate-forme.dft		POLYTECH.MONS
Stéphanie Brine	5e MECANIQUE		

SOLID EDGE ACADEMIC COPY

Bibliographie

[1] Wikipédia. Protocoles de communication : i^2c, Mai 2008. http ://fr.wikipedia.org. 66

[2] ATMega Microcontroller with 32K Bytes In-System Programmable Flash, Août 2007. Datasheet. 85

[3] Philippe PAQUIER. A la découverte de la technologie pic, Mai 2008. http ://www.supinfo-projects.com. 88

[4] Planetary gearhead gp 22 c, 2007.

[5] Maxon ec 22 motors, 2007.

[6] Wikipédia. Intergiciel, 2008. http ://fr.wikipedia.org/wiki/Intergiciel.

[7] Pilote (driver), 2008. http ://www.commentcamarche.net/drivers/drivers.php3.

[8] Wikipédia. Interface de programmation, 2008. http ://fr.wikipedia.org/wiki/Interface_de_programmation.

[9] Wikipédia. Le site du zéro, tutoriels en c et c++, 2008. http ://www.siteduzero.com/.

[10] WikiDroids. Aversive/asservissement microb 2008, 2008. http ://wiki.droids-corp.org/mediawiki/index.php/Aversive/Asservissement_Microb_2008.

[11] *Catalogue Maxon Motors - Programme.* 2007 - 2008.

[12] *Catalogue Optibelt - Power Transmission Courroies.* 2002.

[13] *Catalogue Optibelt - Power Transmission Poulies.* 2003.

[14] *Catalogue Gates - Manuel de détermination and Courroies synchrones.* 1999.

[15] *Catalogue général SKF.* E5000 edition, june 2003.

[16] *Catalogue général SKF.* 4000 / ii f edition, 2002-03.

[17] *Catalogue Baudoin Group.*

[18] *Catalogue Sedis - Principales fabrications.*