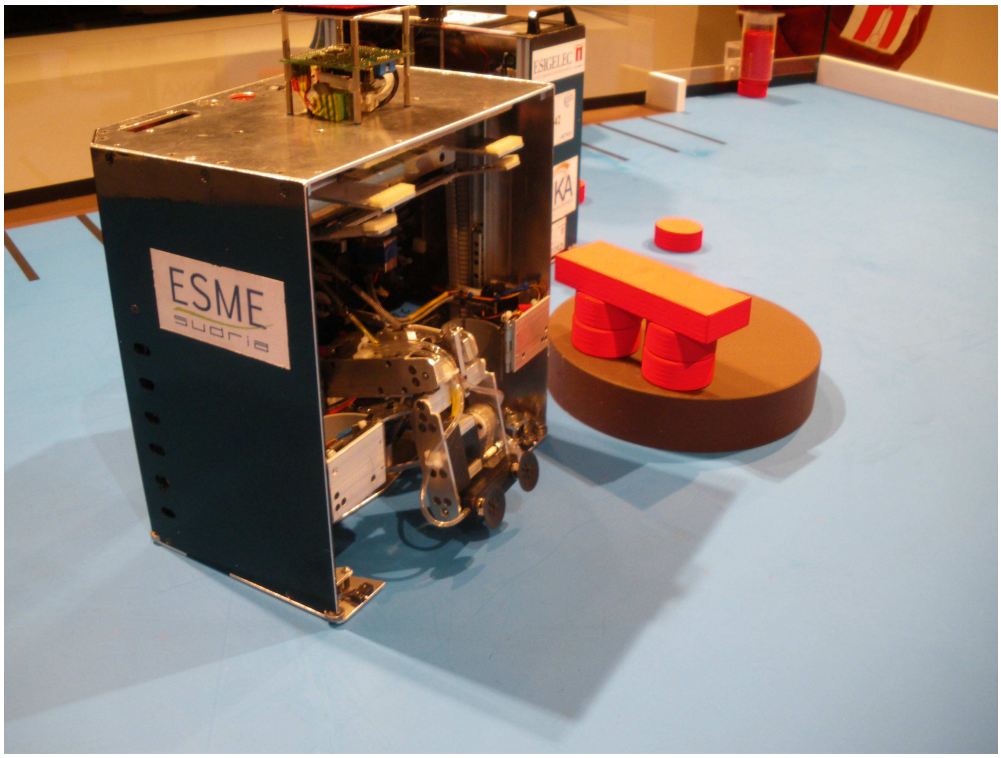
Dossier technique 2009

Sudriabotik

Les Temples of Atlantis



Auteur : BALANDREAU Pierre-Emmanuel (Président 2008-2009)

Sommaire

Introduction

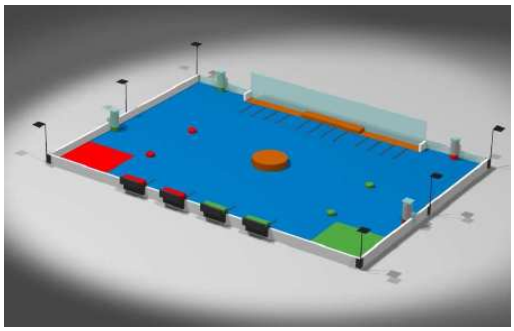
L’année 2009 annonça la seizième édition de la Coupe de France de Robotique, qui s’est déroulée du 20 au 24 mai 2009 dans la Sarthe à la Ferté – Bernard. Ce concours a 16 ans d’existence, et rassemble un nombre important d’équipes formées d’étudiants d’université et de grandes écoles.

Cette année les robots se font bâtisseurs, et doivent aider les Atlantes à ériger les somptueux temples dont nous pourrions aujourd'hui admirer les ruines... si nous avions découvert où se cache l'Atlantide. Comme tout temple antique, les constructions se composent de colonnes surmontées de linteaux qui terminent l'édifice. Par ailleurs, et afin que leurs édifices soient le plus proches des dieux, les Atlantes ont coutume de les bâtir sur les collines les plus élevées de leur continent.

Les matchs font s'opposer deux équipes, chacune avec un seul robot. Ils durent 90 secondes. Chaque équipe se voit attribuer une couleur, rouge ou vert. Elle dispose d'une zone de départ colorée à l'identique, située dans l'un des angles arrière de la table. Les zones d'exploration dans lesquelles les robots peuvent collecter des échantillons et de la glace sont représentées par des distributeurs de balles verticaux et horizontaux.

Les éléments de jeu aux couleurs des équipes sont disponibles en plusieurs endroits de la table, soit directement au sol à des positions prédéfinies, soit dans des distributeurs spéciaux. Les constructions doivent être érigées dans des zones de marquage de points spécifiques, colorées de manière distincte du reste de la table. Ces zones de construction ont des hauteurs différentes. Les points sont attribués en fonction de la hauteur des constructions, de leur composition et de la hauteur de la zone où la construction se trouve.

Il est important de noter que le thème de cette année est orienté vers les actions de construction, et que par conséquent, toute action de destruction (intentionnelle ou non) sera sanctionnée.



Résumé :

Le thème de la coupe de France de robotique change tous les ans. Il se tourne en 2009 vers la construction de temple antique. Intitulé «Temples of Atlantis» l'objectif de la coupe 2009 consiste à construire des temples le plus haut possible. Ceux-ci sont symbolisés par des palets et des linteaux à empiler dans les zones de dépose marron. Le robot qui aura construit un maximum de temple le plus haut possible gagnera le match.

Après avoir discuté des différents systèmes mécaniques pouvant permettre à notre robot de remplir sa mission nous en avons choisi un et décidé de le modéliser en 3 dimensions sous CATIA-V5. Dès la réception de la découpe LASER des plaques d'aluminium nous avons monté la base roulante du robot avec, pour particularité d'avoir les roues motrices et les roues codeuses alignées. Le robot ramasse les palets par devant à l'aide de pinces, ramasse les linteaux à l'aide d'un bras équipé de pompes à vide et construit des temples sur la colline afin d'obtenir le maximum de points.

L'électronique du robot réalisée à base de microcontrôleurs de chez Microchip, permet de traiter l'information bas niveau comme l'acquisition d'état des capteurs ou la commande d'actionneurs à une fréquence de rafraîchissement de 2,5ms. Un bus I²C permet aux cartes de communiquer entre elles.

La stratégie est programmée en langage C et se trouve dans la DsPIC Stratégie. La boucle du programme s'effectue toutes les 50ms. Le programme détermine les trajectoires durant le match et envoie les consignes de vitesse pour les moteurs propulsions et les commandes de certains actionneurs.

Depuis 2006, les robots représentant l'ESME Sudria ont énormément améliorés leur compétitivité et s'approchent de plus en plus de ceux des équipes finalistes.

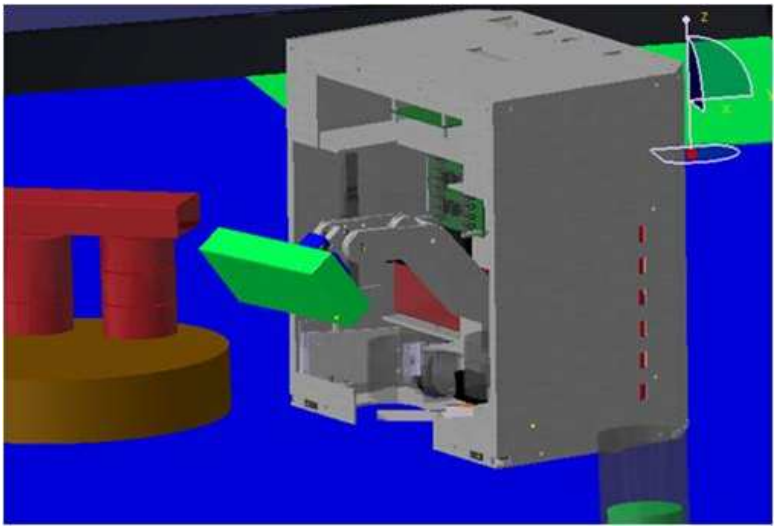
Suite aux problèmes rencontrés durant cette année, des améliorations devront être apportées au niveau informatique, où une correction du nouveau code de la DsPIC Stratégie doit être effectuée afin de pouvoir bénéficier de déplacements plus précis, permettant aussi l'implantation d'un véritable évitement adversaire.

1. Présentation de la mécanique du robot

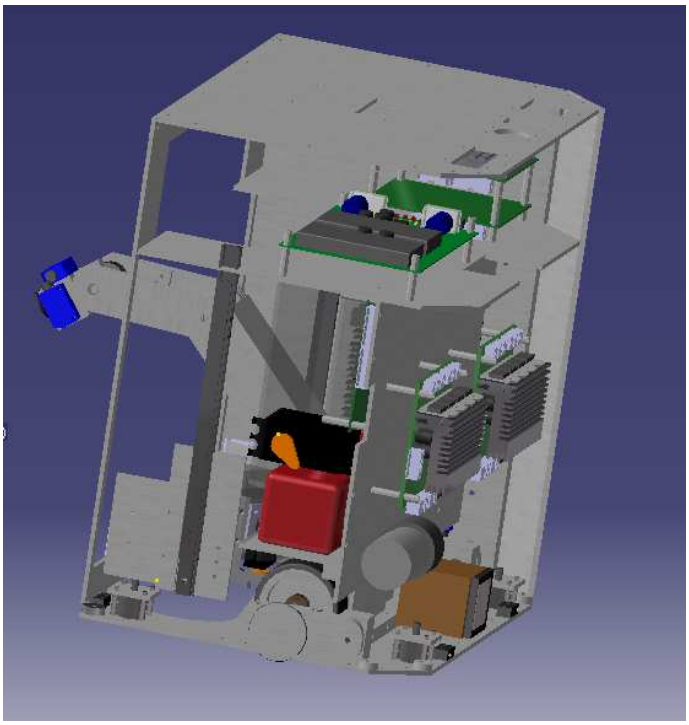
1.1. Modélisation 3D

La modélisation en 3D du robot est devenue indispensable, elle permet principalement de tirer des vues 2D pour la découpe des plaques par des machines à commandes numériques.

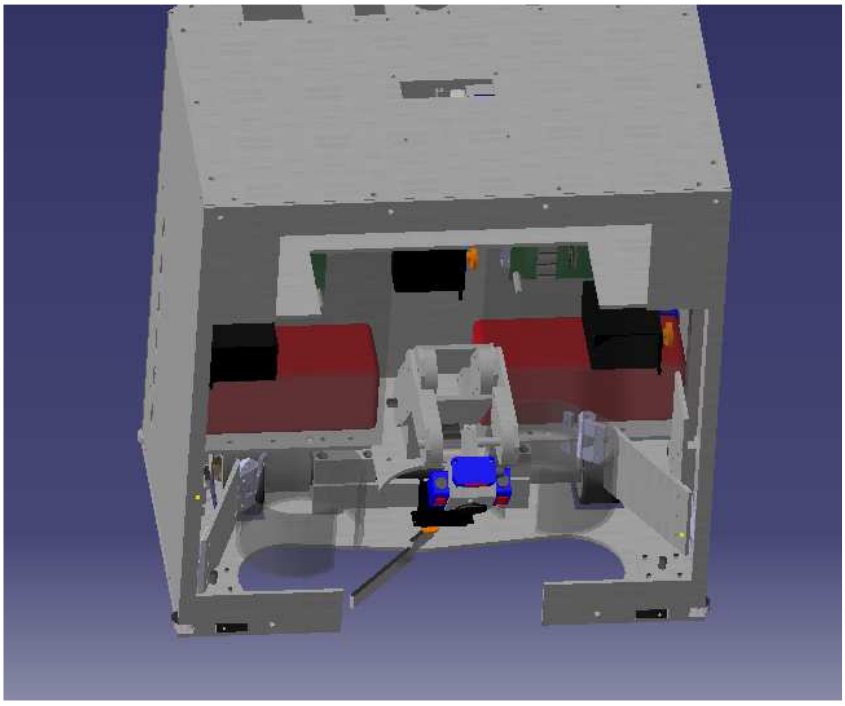
A l'aide de la bibliothèque développée par l'association on peut facilement mettre en place tous les éléments afin de vérifier que les systèmes mécaniques occupent bien la place prévue et qu'ils s'adaptent bien au règlement.



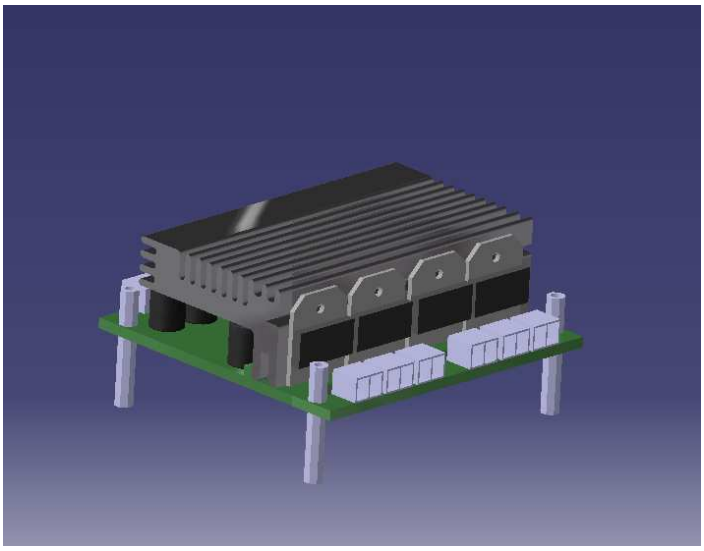
Vue globale du robot



Vue de coté du robot



Vue de face du robot



Carte électronique

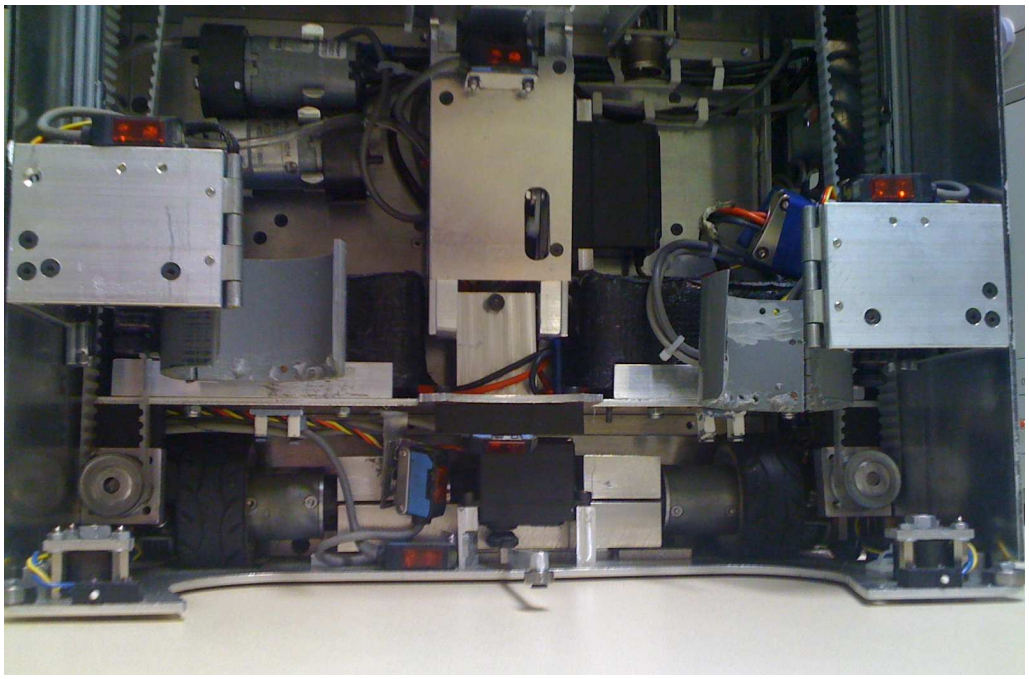
1.2. Base roulante

1.2.1. Châssis

Le châssis sert de support pour :

- Le bloc-moteur
- Les cages à bille
- Les roues codeuses
- Les roulements à billes de bordure
- Le squelette du robot

La majorité des pièces sont en aluminium AU4G, léger et très rigide il est utilisé notamment en aéronautique.

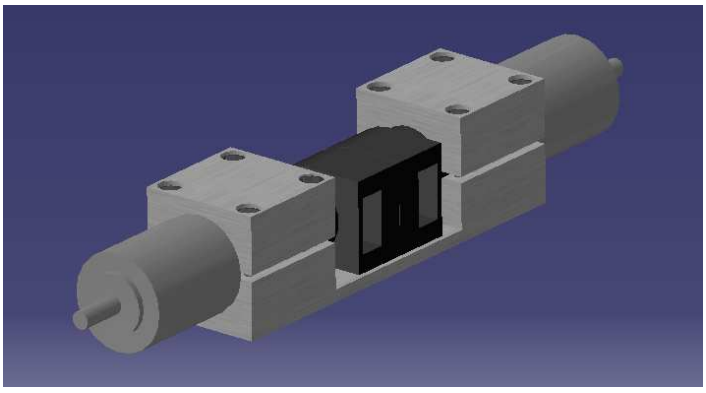


1.2.2. Bloc-moteur

Les motoréducteurs sont constitués d'un moteur Maxon de 25watts et d'un réducteur planétaire PLG32 de 2 étages avec un rapport de réduction de 20.
Les roues sont munies de jantes aluminium AU4G et usinées à l'aide d'un tour numérique. Les pneus de 35 shores sont en caoutchouc très adhérant et leur faible épaisseur minimise au maximum l'écrasement de celui-ci sous les 14 kilos du robot.
Cette combinaison est le meilleur compromis entre couple et accélération.

Détail roue motrice

Le bloc moteur permet l'alignement des motoréducteurs de propulsion du robot. Ainsi les dérives dues à un mauvais alignement des moteurs sont négligeables, ce qui améliore nettement la précision du robot.


Bloc moteur

Les motoréducteurs sont pincés entre les blocs d'aluminium au niveau des bobines des moteurs. Le réducteur est ainsi à découvert, ce qui permet aux jantes des roues motrices d'épouser les réducteurs.

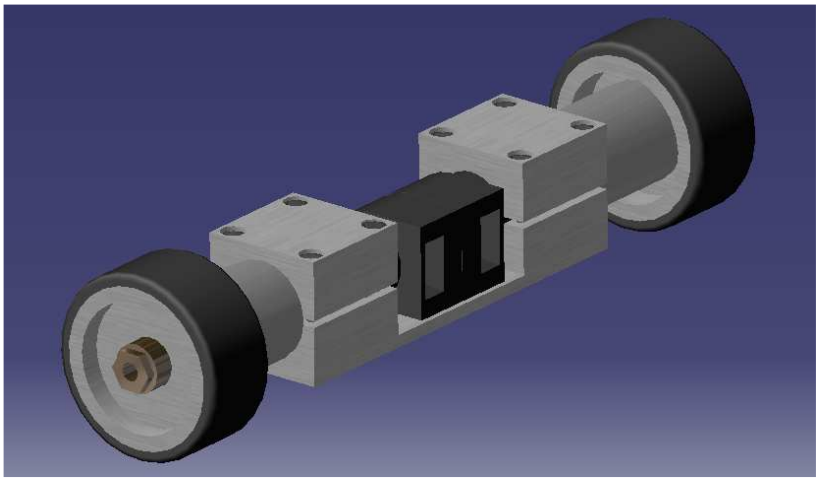


Photo bloc moteur + roues

Ce montage permet un gain de place significatif sur la largeur du robot et permet l'alignement des roues codeuses avec les roues motrices (avant 2008, les codeurs n'étaient pas alignés). Ainsi les erreurs de précision dues à la mécanique sont minimisées, et les erreurs de calcul pour le calcul du centre du robot sont éliminées.

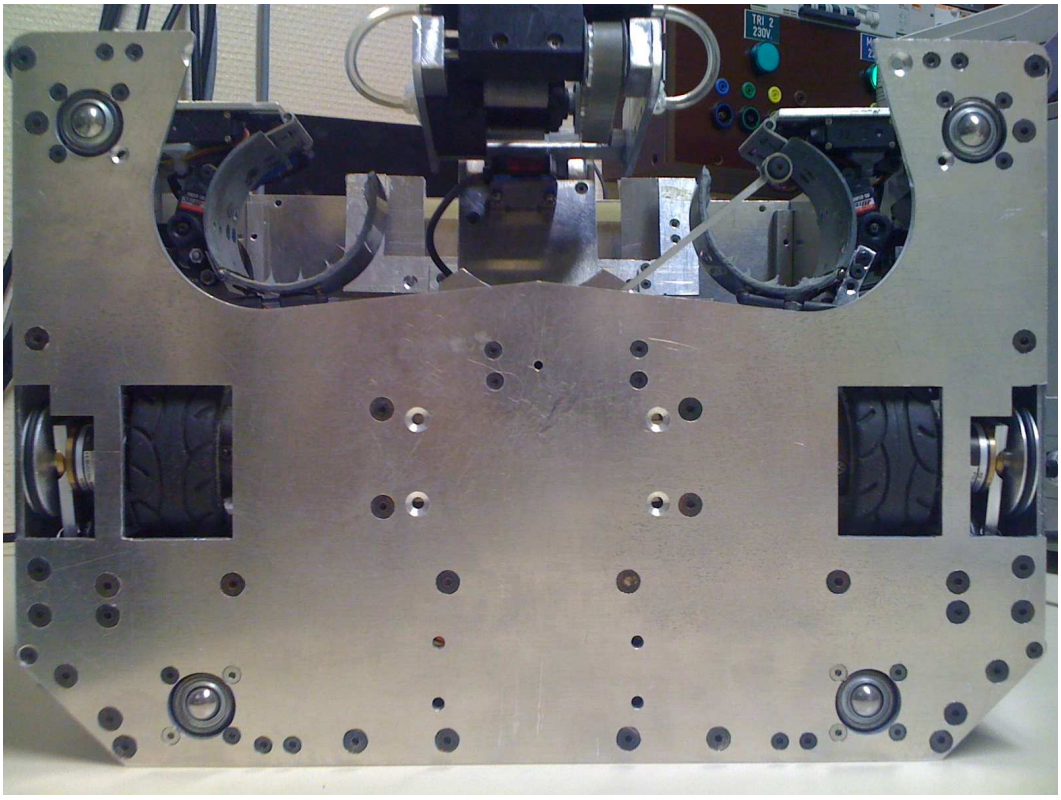


Photo du châssis

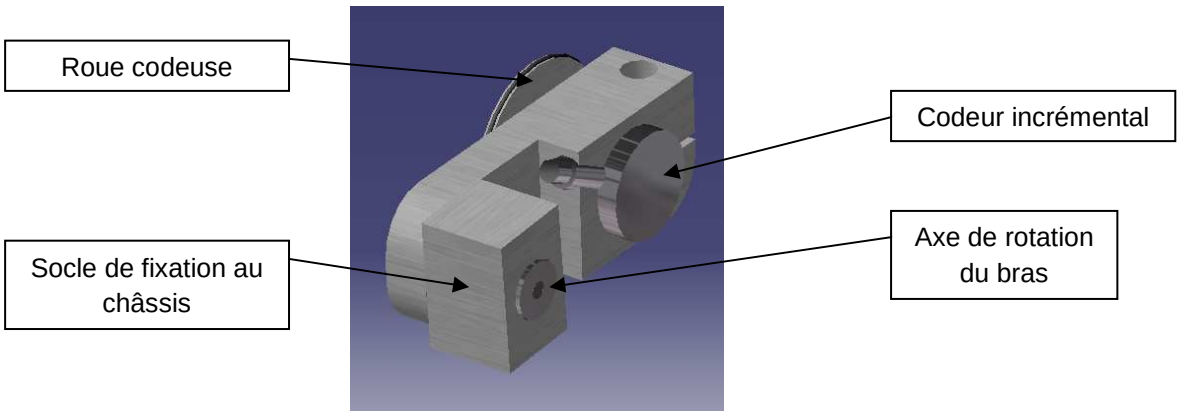
1.2.3. Cage à bille

Les cages à billes donnent au robot, en complément des roues motrices, des points d'appui libres de tout mouvement. En fonction du placement du bloc moteur sur le châssis on place deux ou quatre cages à billes.

Cage à bille

1.2.4. Système de roues codeuses

Nous utilisons des codeurs incrémentaux montés sur des roues très fines et très légères. Elles sont libres de tout mouvement ainsi le robot peu connaître et mettre à jour ses coordonnées à chaque déplacement. Ces roues sont appelées « roues codeuses ».
Le codeur incrémental est fixé sur un pivot muni d'un ressort pour que la roue adhère un maximum à la table.



Système de roue codeuse

Comme vu précédemment, pour limiter les erreurs de glissement lors des rotations, les roues codeuses sont montées de façon à ce que leurs points de contact sur le sol soient alignés avec ceux des roues motrices.

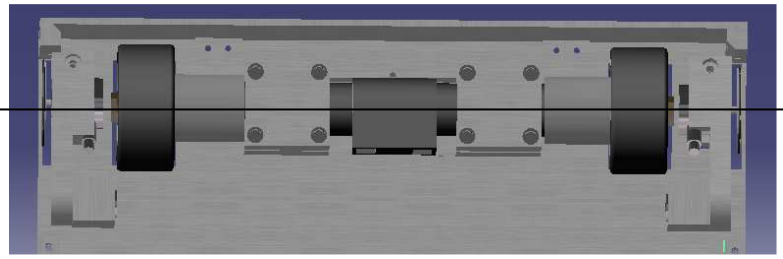
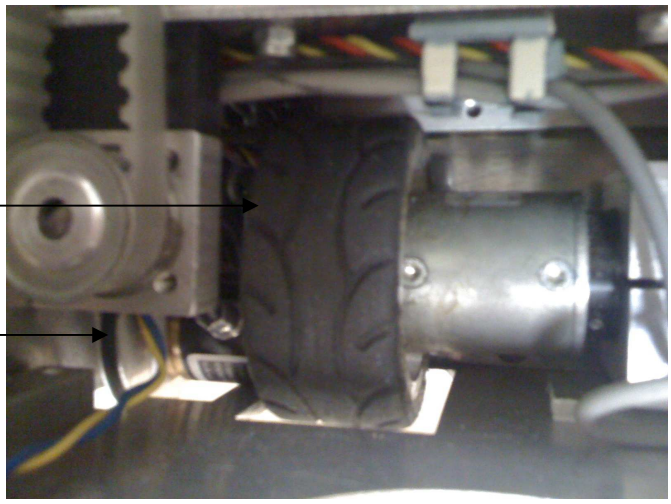


Photo roue codeuses + bloc moteur



Alignement des roues codeuses et des roues motrices

1.2.5. Capteur de contacts

Sur le châssis on fixe des petits boutons poussoir afin de réinitialiser les coordonnées du robot lors d'une procédure de recalage quand les deux contacts avant ou arrière touchent une bordure de terrain.

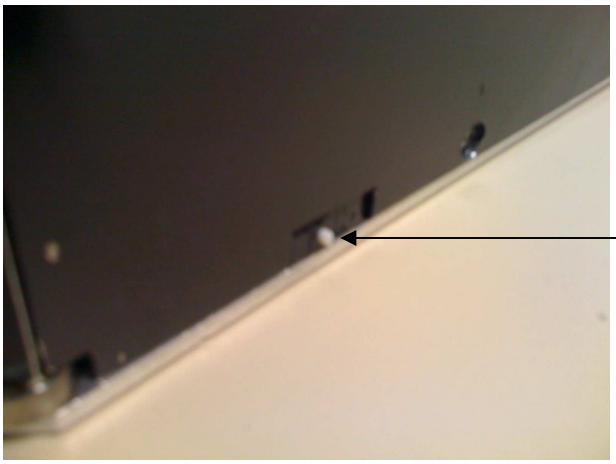


Photo capteur contact arrière

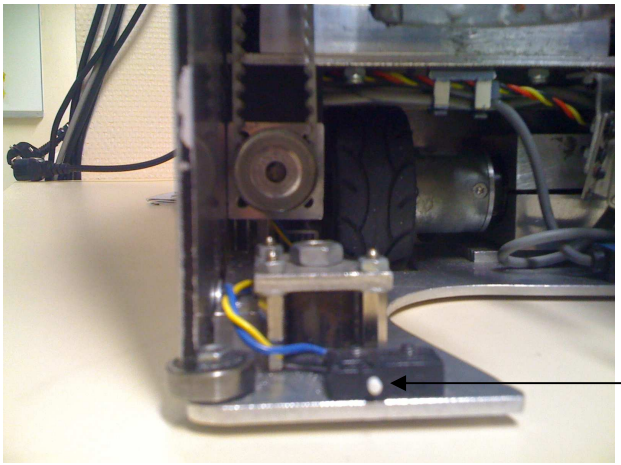
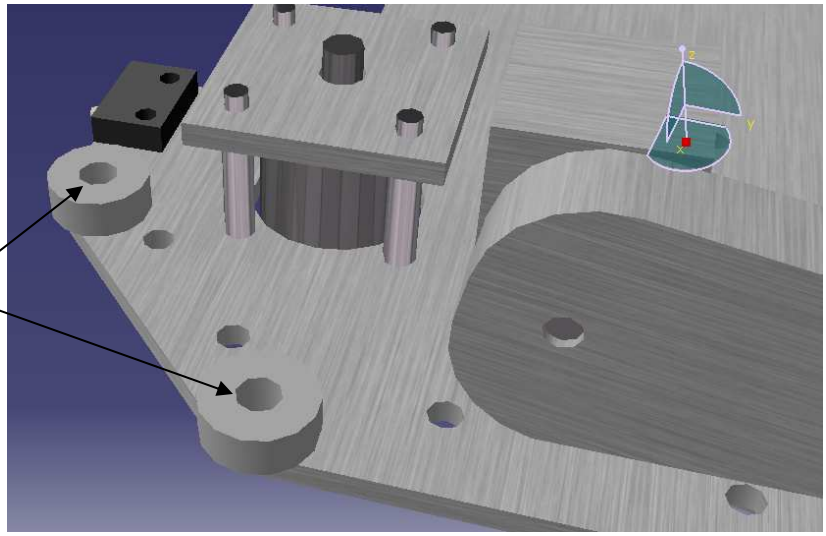


Photo capteur contact avant

1.2.6. Roulement à bille

Les angles du châssis sont droits et par conséquent saillants. Il arrive souvent que les robots touchent la bordure du terrain et abîment la peinture. Pour éviter toute pénalité on place sur les angles des roulements à billes qui permettent au robot de glisser le long de la bordure sans endommager la peinture.



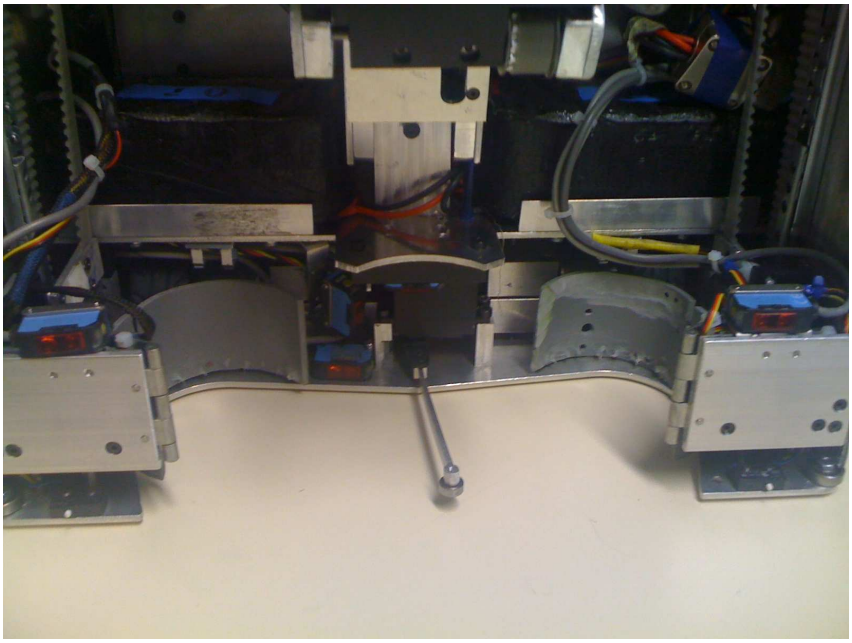
Roulement à billes sur le châssis

Photo roulement à billes sur le châssis

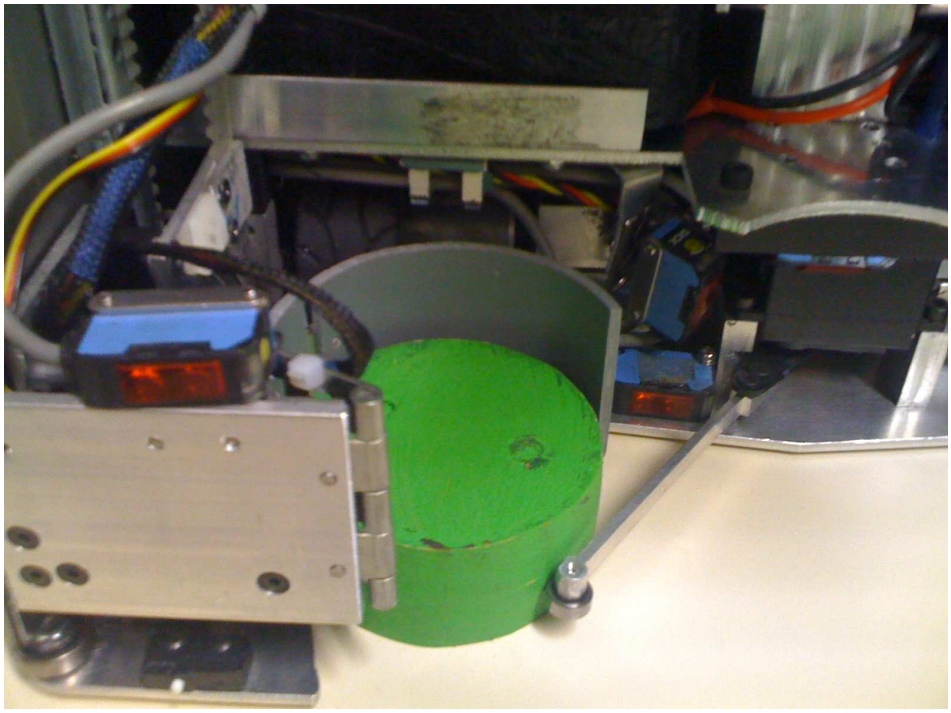
1.3. Les systèmes mécaniques

1.3.1. Système de gestion des palets

Afin de récupérer les palets, le robot possède une petite barre munie d'un roulement à bille permettant d'aiguiller le palet lorsque le robot rencontre les objets sur le plateau de jeu ou encore quand il vide un réservoir vertical.

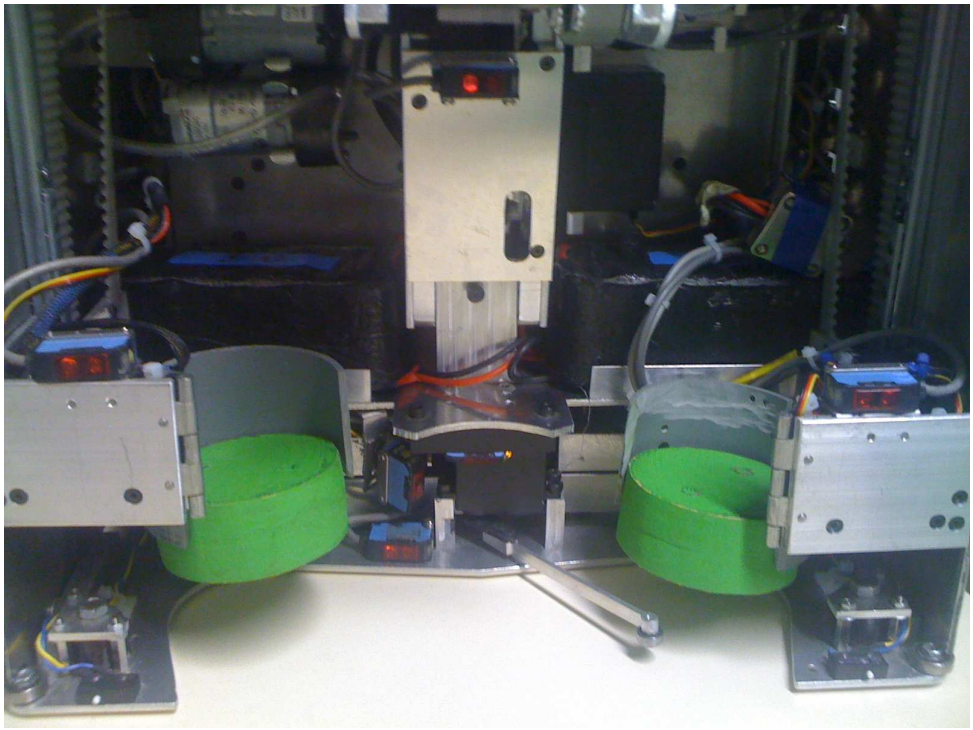


Position de la barre au repos

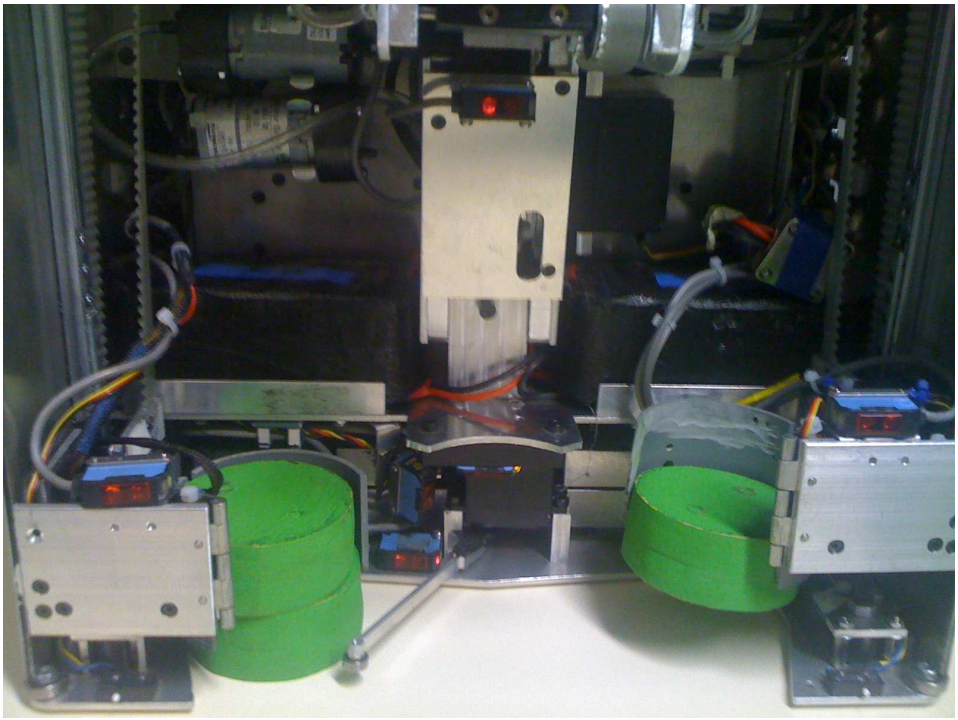


Calage d'un palet dans la pince droite

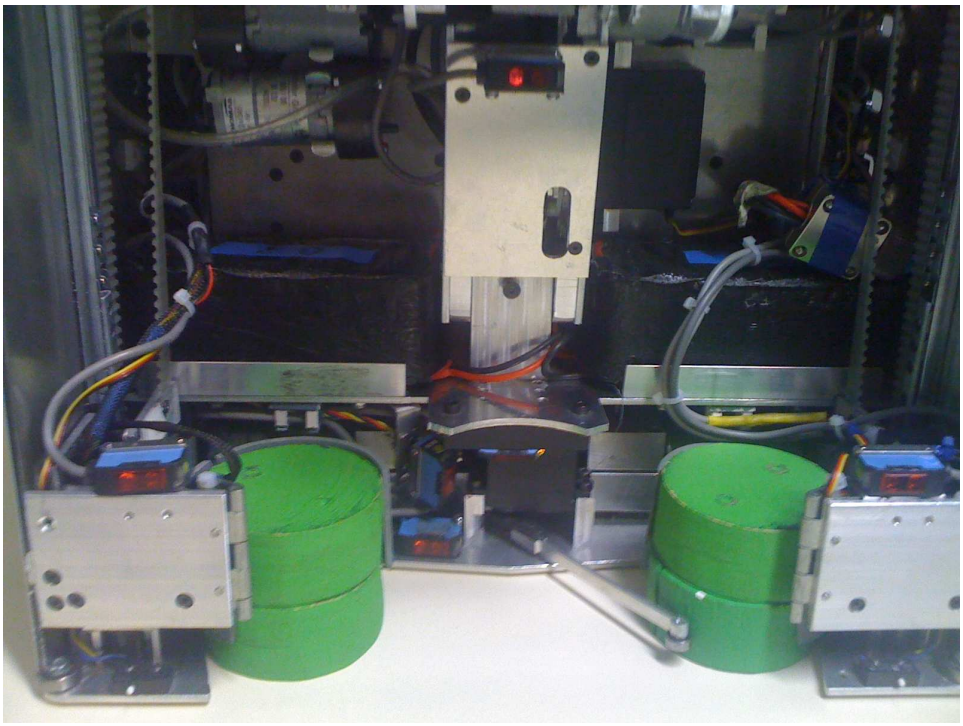
Fin de l'étape 1 du ramassage des palets



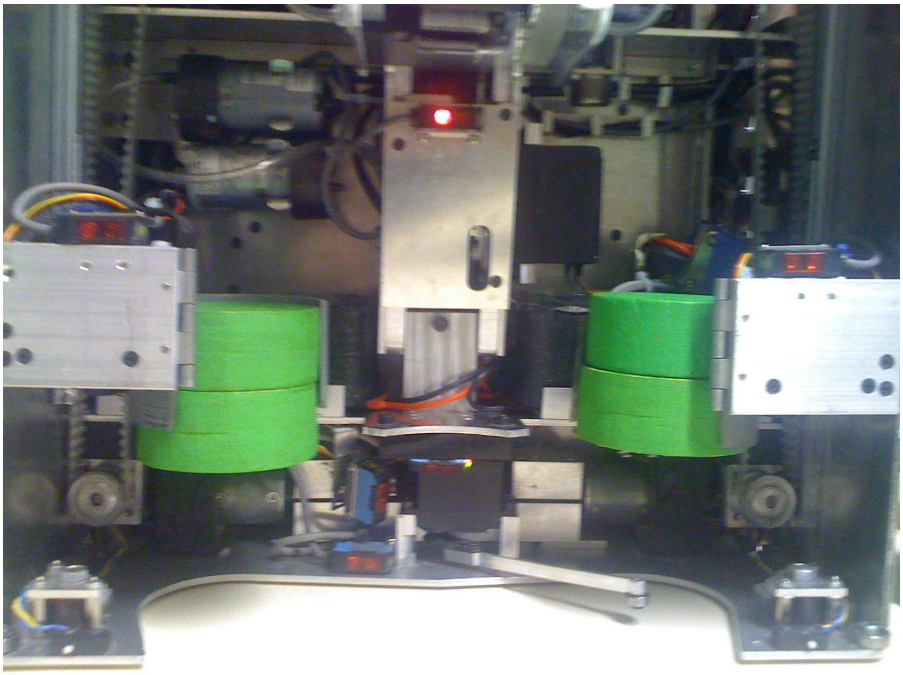
Fin de l'étape 2 du ramassage des palets



Fin de l'étape 3 du ramassage des palets



Etape 4 du ramassage des palets

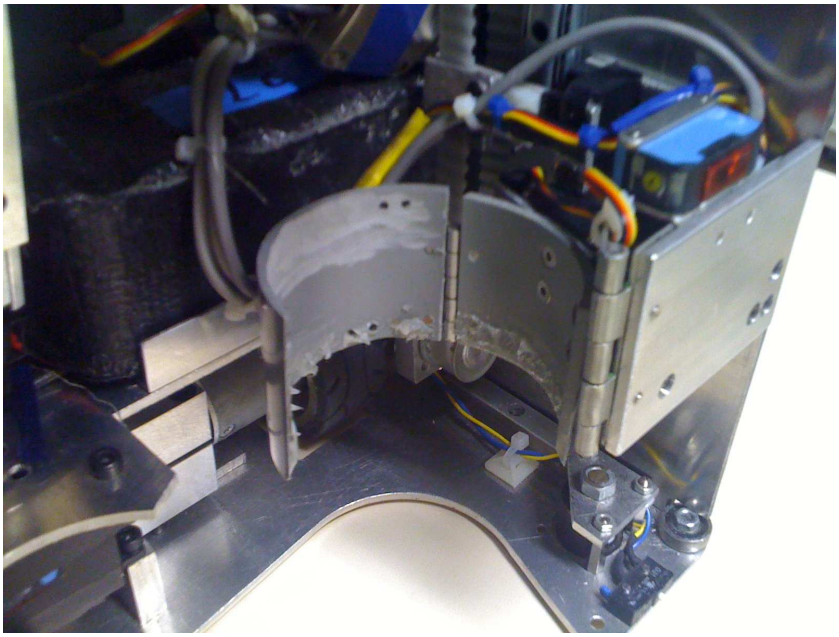


Fin de l'étape 4 du ramassage des palets

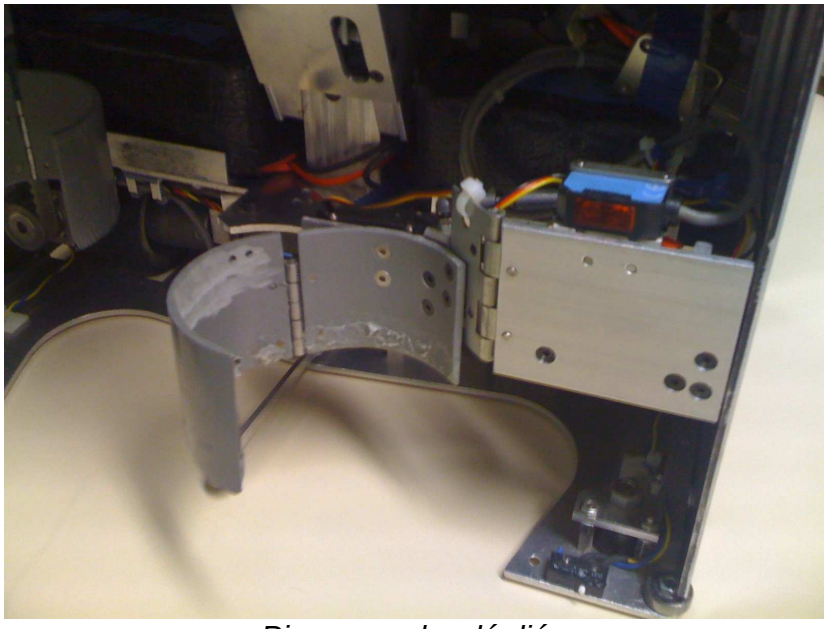
Le palet est ensuite saisi par les pinces puis monté d'un étage. De la même façon les autre palets sont répartis de chaque coté, créant ainsi les deux colonnes du temple.

Chacune des pinces a trois mouvements possibles:

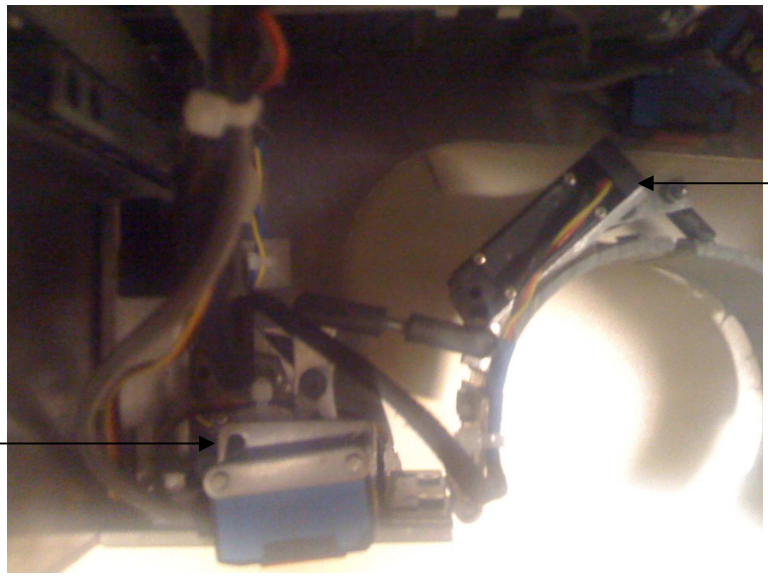
- Ouverture ou fermeture de la pince de préhension des palets
- Montée ou descente de la pince sur 8 niveaux différents
- Rotation pour déposer les palets à l'extérieur du robot



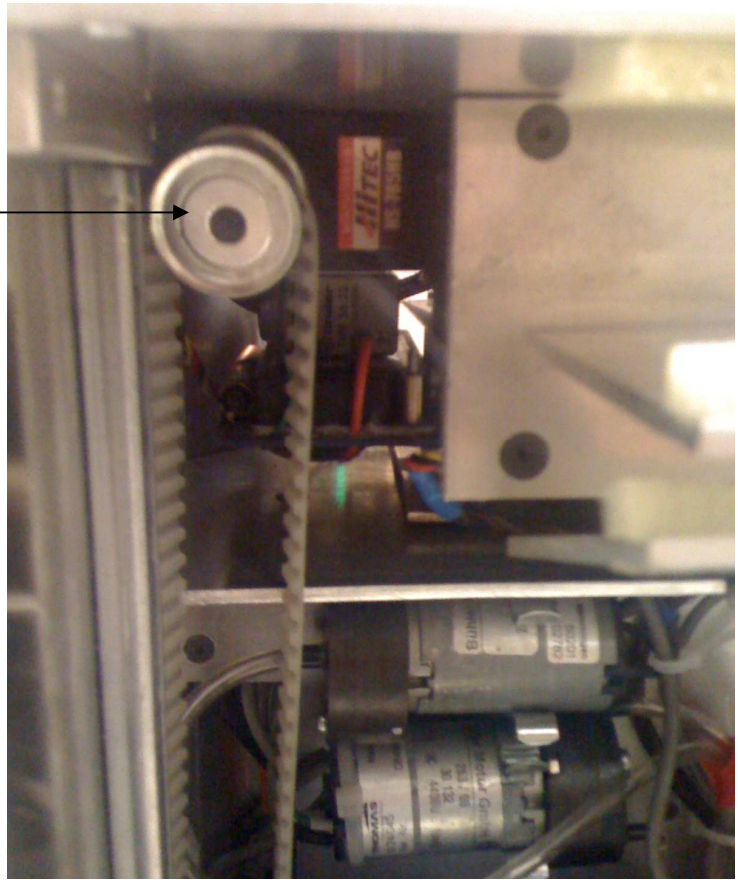
Pince gauche repliée



Pince gauche dépliée



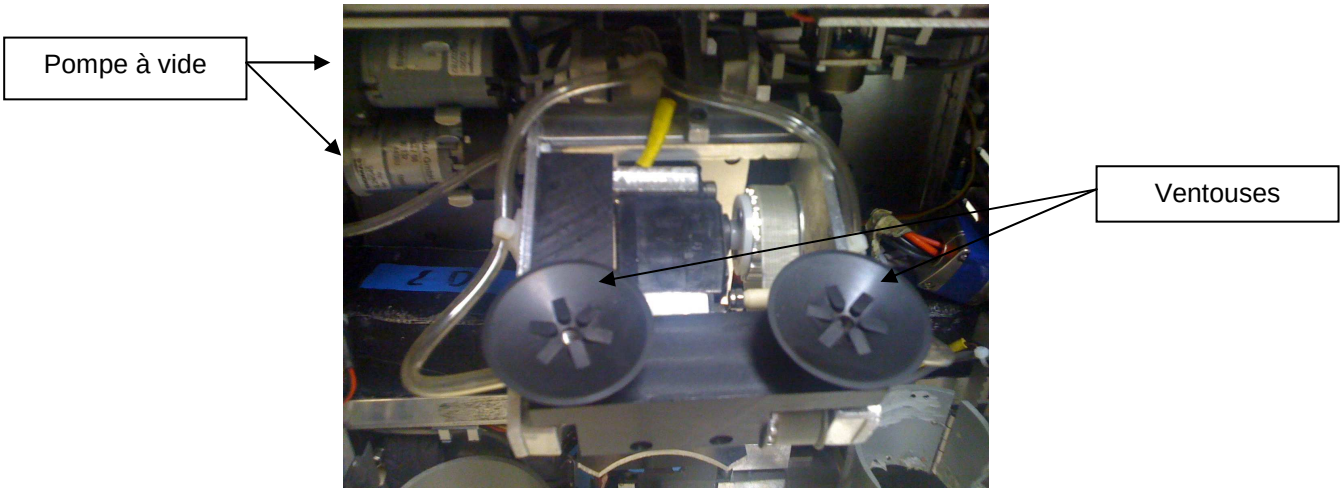
Vue de dessus de la pince droite



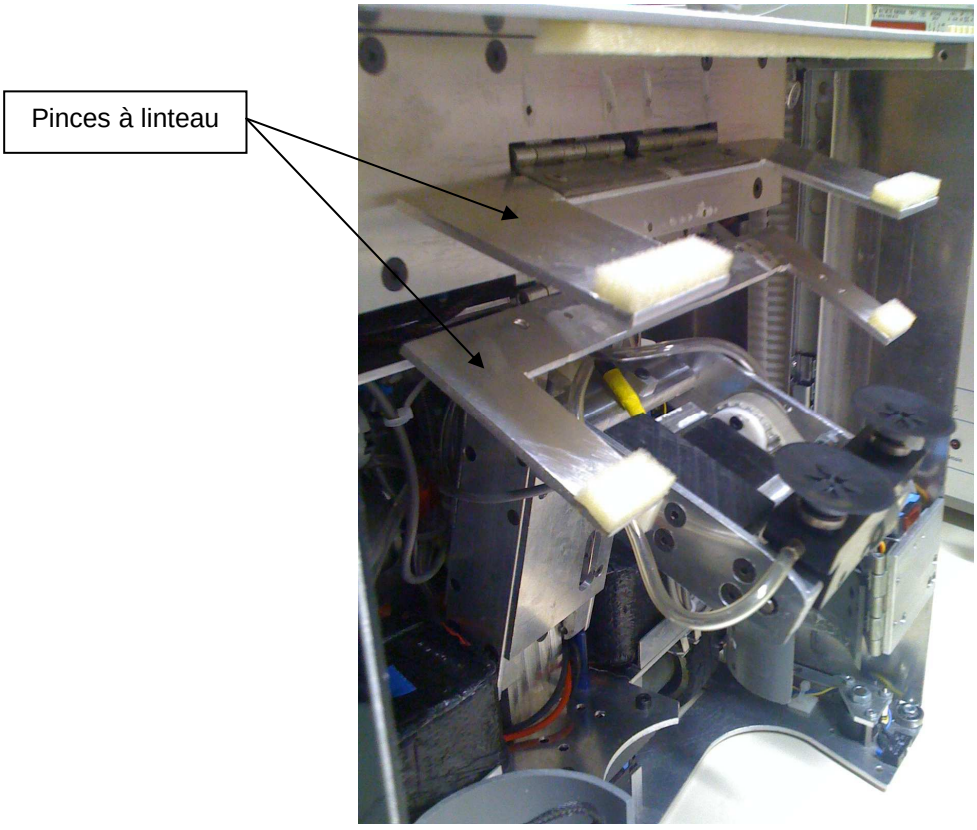
Vue de face du robot

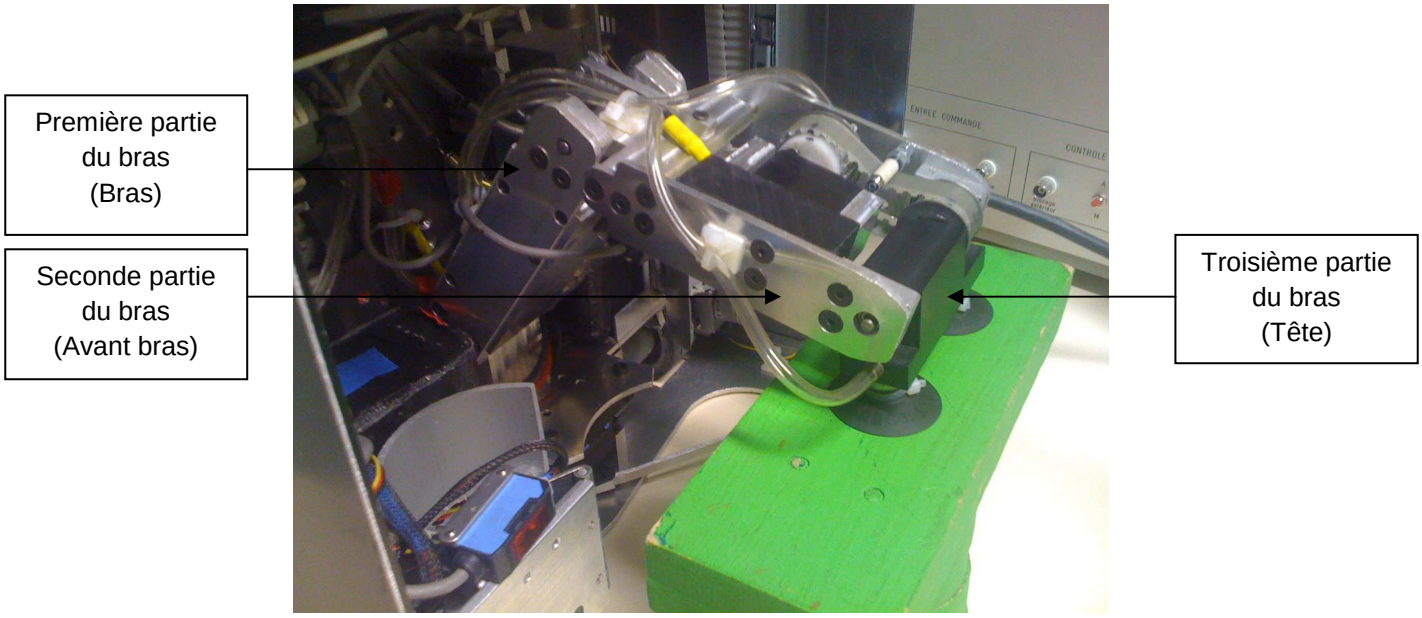
1.3.2. Système de gestion des linteaux

Pour permettre le ramassage et la dépose des linteaux, nous avons utilisé un bras muni de deux ventouses associées à deux pompes à vide Gardner Denvers. Ces dernières sont alimentées en 24V, ont un débit nominal de 6.1l/min et peuvent maintenir une pression maximale en continu de 150mbar.

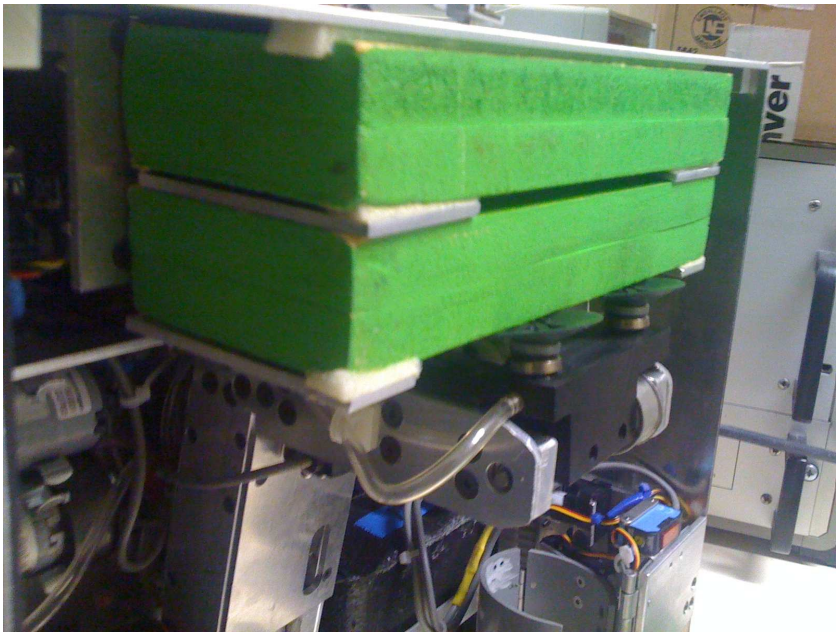


Afin de stocker nos linteaux, nous disposions de deux pinces dans la partie haute du robot. Pour ramasser un linteau, le robot devait alors descendre le bras, faire le vide au niveau de ses ventouses et remonter son bras tout en ouvrant une ou deux pinces de stockage pour y déposer l'élément de jeu saisi. Pour la dépose le mouvement reste le même, mais dans l'ordre inverse.





Détails du bras



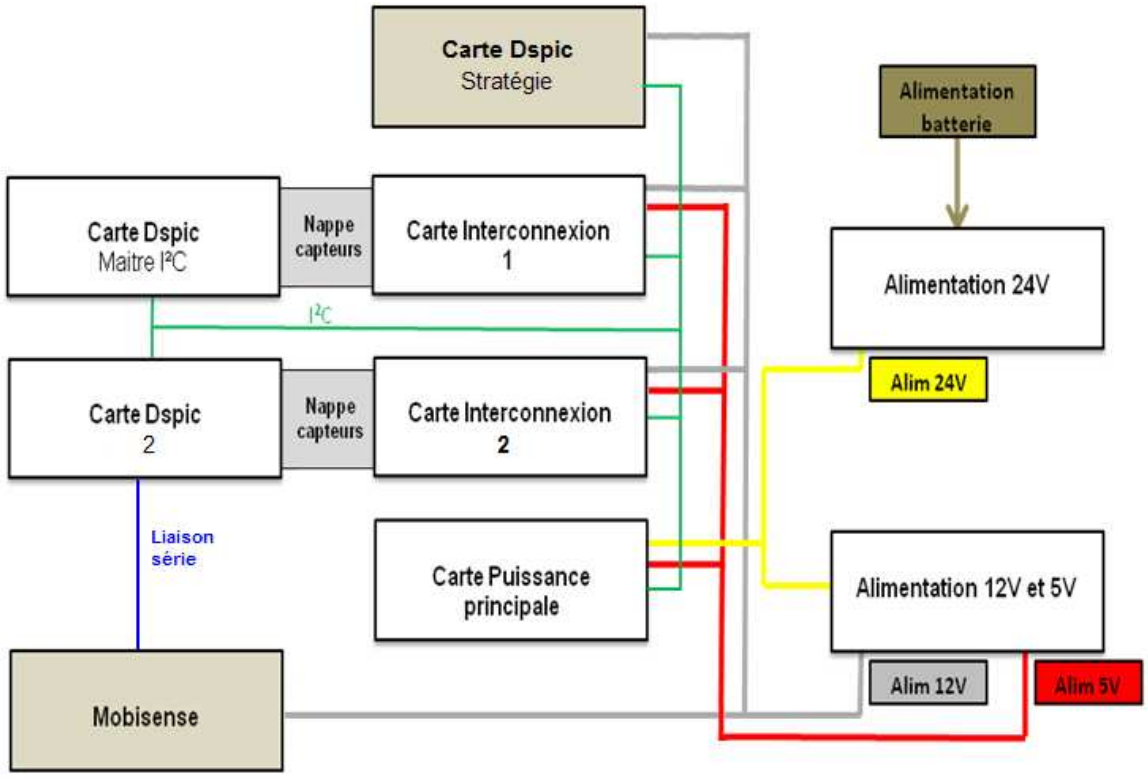
Stockage des linteaux

1.4. Implantation de l'électronique

L'emplacement des systèmes mécaniques laisse l'arrière du robot pour l'implantation des cartes et des câbles.

2. Présentation de l'électronique du robot

On retrouve 2 types de cartes dans le robot. Les cartes qui fournissent l'alimentation 24V, 12V et 5V du robot et les cartes électroniques.



L'architecture du robot permet une très grande modularité. On peut ainsi suivant les besoins de chaque année ajouter une carte puissance/Actionneur pour avoir des moteurs supplémentaires ou encore une carte DsPIC connectée à une carte d'interconnexion pour l'ajout de capteurs ou de servomoteurs. La grosse nouveauté de cette année est le remplacement de notre ancien PC embarqué ('ancien cerveau' du robot) par la DsPIC Stratégie. Cela nous a aussi obligé à mettre en place l'I2C esclave sur DsPIC, nous permettant aussi l'ajout d'une seconde carte DsPIC avec sa carte interconnexion afin de pouvoir bénéficier de plus de capteurs.

2.1. Cartes alimentations

2.1.1. Carte 24V

La carte 24V adapte la tension fournie par les batteries ou l'alimentation externe par l'intermédiaire d'un régulateur 24V Lambda (24V 4,2A).
Nous avons la possibilité d'ajouter un 2eme régulateur 24V.

La carte contient 4sorties :

- Une sortie 24V pour la carte alimentation 12V / 5V.
- Une sortie 24V pour les moteurs propulsions.
- Deux sorties 24V pour des capteurs.

Toutes les entrées et sorties sont protégées avec des fusibles.

2.1.2. Carte alimentation 12V 5A / 6V 10A

La carte fournit une alimentation 12V pour les capteurs et 6V pour les servomoteurs à partir de 2 régulateurs lambda, permettant l'alimentation des cartes électroniques.
Les deux sorties sont protégées pas des fusibles.

2.1.3. Carte alimentation 12V 1A / 5V 5A

Cette carte permet l'alimentation d'une partie des cartes électroniques (carte interconnexion et actionneur).

2.2. Carte DsPIC Maître

Caractéristiques hardware:

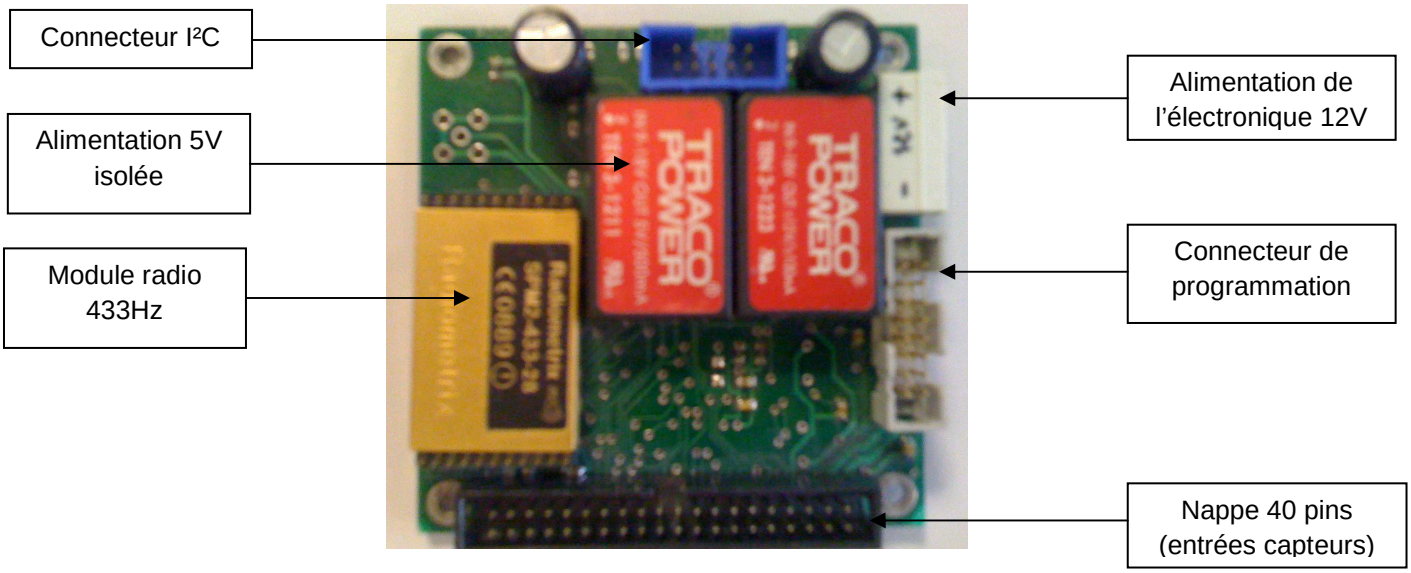
- *DsPIC 30F6012A*
- *Alimentation en 12V, régulée en 5V par un TRACO POWER*
- *Module radio 433Hz*
- *GPIO sur la nappe 40 pins*
- *Connecteur I2C*
- *MAX232 pour la liaison série*

Caractéristiques software :

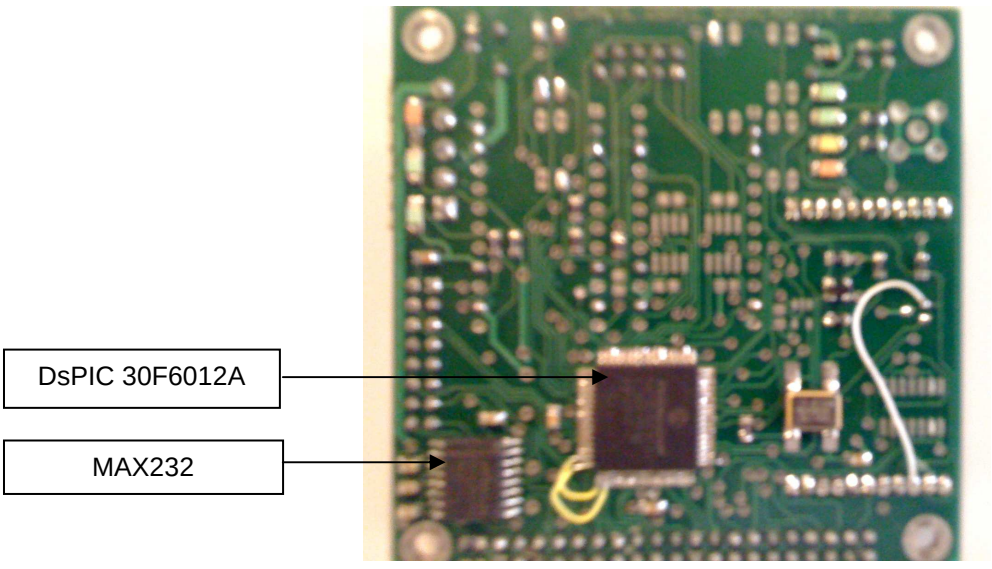
- *Gestion de l'automatisme du robot.*
- *Maitre i2c : cette carte gère l'ensemble des communications I2C.*
- *commande de servomoteurs, commande de moteurs, gestion de 12 capteurs, 16 contacts et 4 codeurs incrémentaux via une carte d'interconnexion.*
- *Le programme principal est décomposé en 20 phases de 2.5ms (20x2.5=50ms=HTR).*
- *Calcul de positionnement toutes les 2,5ms.*
- *Debug par liaison série.*

Communication :

- *Cette carte communique par I2C avec les autres cartes.*
- *Elle communique aussi directement avec la carte d'interconnexion via une nappe.*



Vue de dessus



Vue de dessous

2.3. Carte DsPIC 2

Caractéristiques hardware:

- *Les mêmes que la carte Dspic Maitre*

Caractéristiques software :

- *Gestion supplémentaire de 12 capteurs, 16 contacts et 4 codeurs incrémentaux via une carte d'interconnexion.*
- *Esclave sur le bus I2C.*
- *Le programme principal est décomposé en 20 phases de 2.5ms (20x2.5=50ms=HTR).*
- *Debug par liaison série.*

Communication :

- *Cette carte communique par I2C avec les autres cartes.*
- *Elle communique aussi directement avec la carte d'interconnexion via une nappe.*

2.4. Carte DsPIC Stratégie

Caractéristiques hardware:

- *Les mêmes que la carte Dspic Maitre*

Caractéristiques software :

- *Gestion de l'asservissement du robot, mais aussi de la stratégie du robot, de l'évitement adversaire et de l'évitement d'obstacle.*
- *Esclave sur le bus I2C.*
- *Le programme principal s'exécute séquentiellement, avec une interruption toute les 10ms pour le calcul des nouvelles consignes d'asservissement.*
- *Gère également le timer de 90sec de match.*
- *Debug par liaison série.*

Communication :

- *Cette carte communique par I2C avec les autres cartes*

2.5. Carte Puissance principale

Caractéristiques hardware:

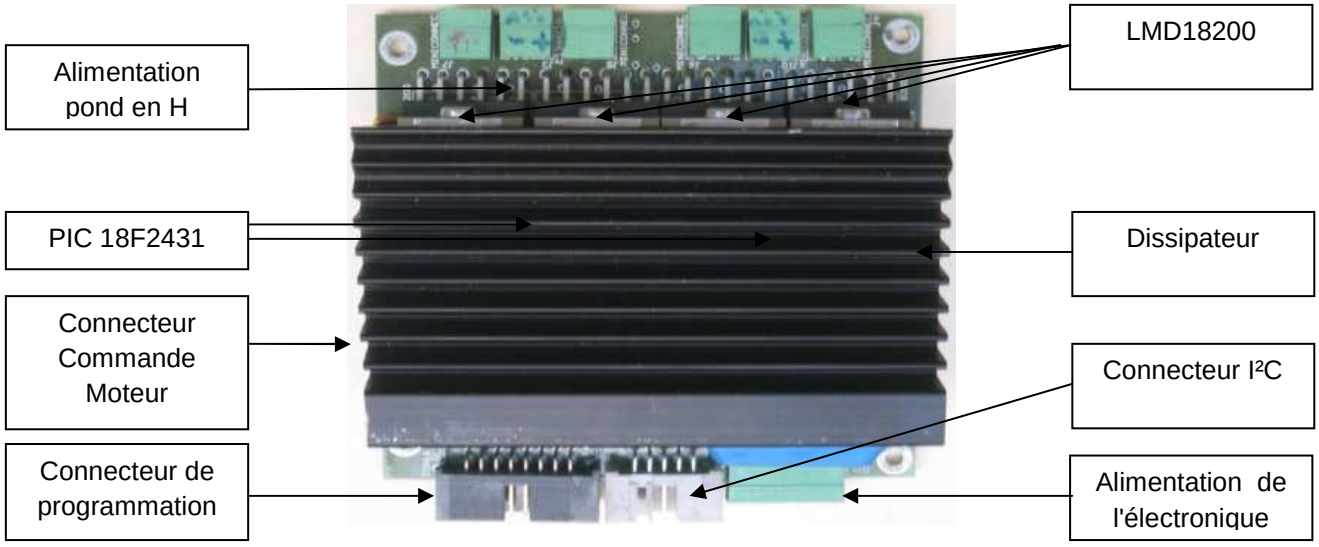
- *4 ponts en H de 3A (LMD18200)*
- *2 PIC 18F2431*
- *MAX232 pour la liaison série*
- *2 entrées codeurs incrémentaux*
- *Alimentation 5V pour les PIC*
- *Alimentation 24V pour les moteurs*

Caractéristiques software :

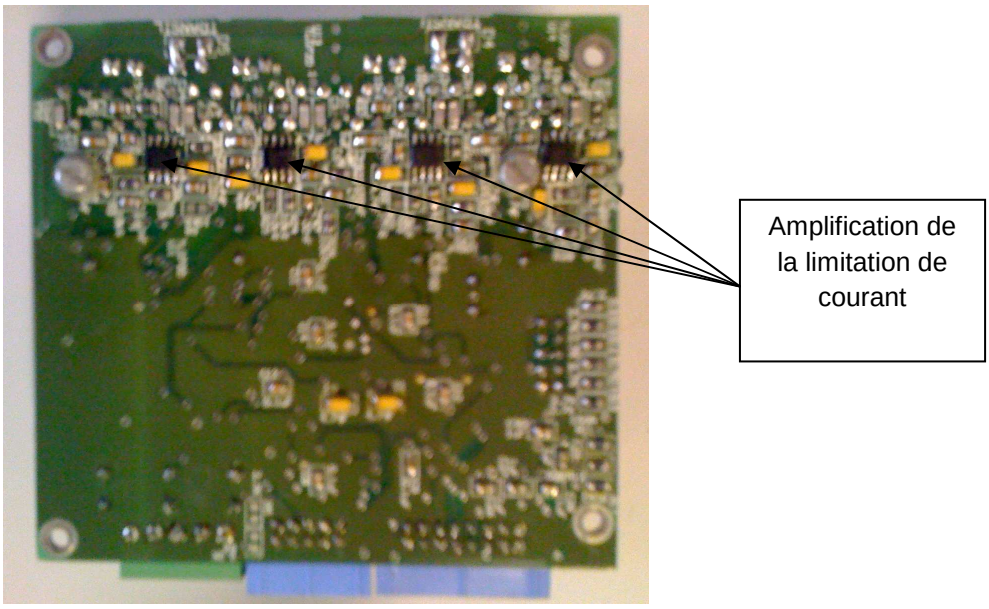
- *Esclave sur le bus I2C.*
- *Limitation en courant sur chaque moteur.*
- *La carte Puissance principale reçoit de la carte DsPIC Stratégie la commande vitesse de chaque.*
- *Les deux PIC 18F transmettent le signal à la puissance, celle-ci est fournie par les LMD18200.*
- *Deux autres moteurs peuvent être connectés en cas de nécessité.*

Communication :

- *Cette carte communique par I2C avec les autres cartes.*



Vue de dessus



Vue de dessous

2.6. Carte d'interconnexion

Caractéristiques hardware :

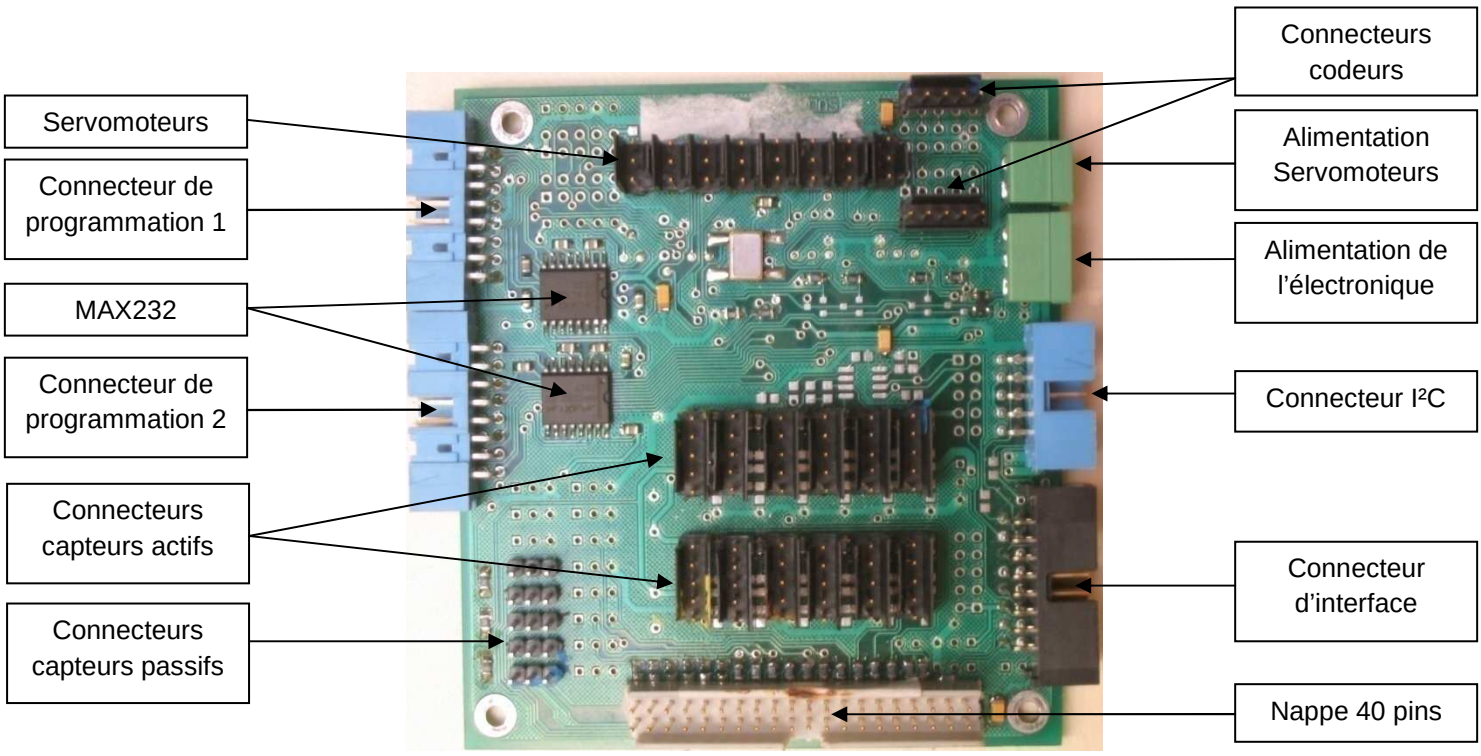
- *4 PIC 18F2431*
- *2 MAX232 pour la liaison série*
- *8 servomoteurs*
- *16 entrées contacts*
- *12 entrées capteurs*
- *4 entrées codeurs incrémentaux*
- *Alimentation 5V pour les PIC*
- *Alimentation 12V pour les capteurs*
- *Alimentation 6V pour les servomoteurs*

Caractéristiques software :

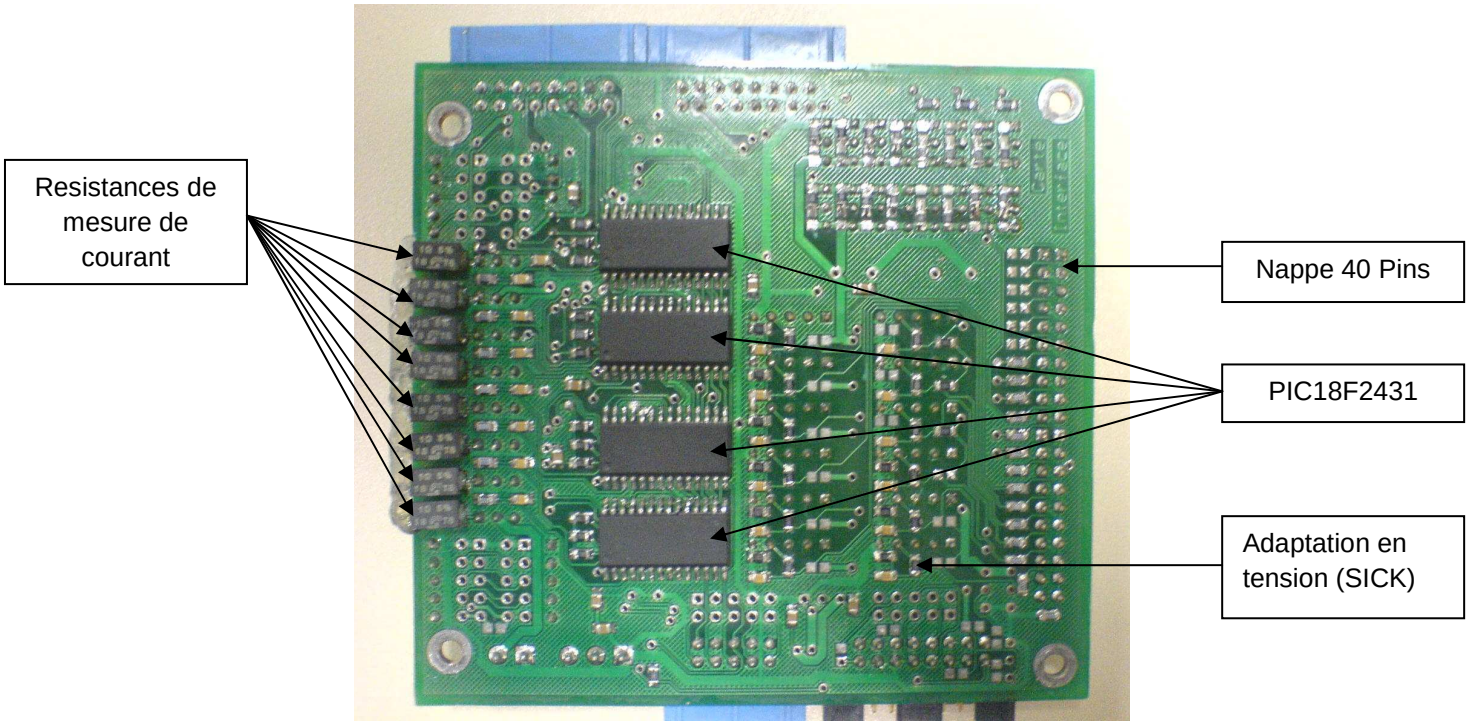
- *Esclave sur le bus I2C.*
- *Limitation en courant sur chaque servomoteur.*
- *Cette carte adapte en tension les signaux des différents capteurs pour la carte DsPIC.*
- *Les PIC contrôlent respectivement deux servomoteurs, une acquisition en quadrature de codeur incrémental, soit au total 8 servomoteurs, 4 capture en quadrature et 4 entrées ou sorties.*
- *Les commandes des servomoteurs sont envoyées par la DsPIC maître via l'I2C.*

Communication :

- *Cette carte communique par I2C avec les autres cartes*



Vue de dessus



Vue de dessous

2.7. Caméra Mobisense (MBS270 v2)

Caractéristiques hardware :

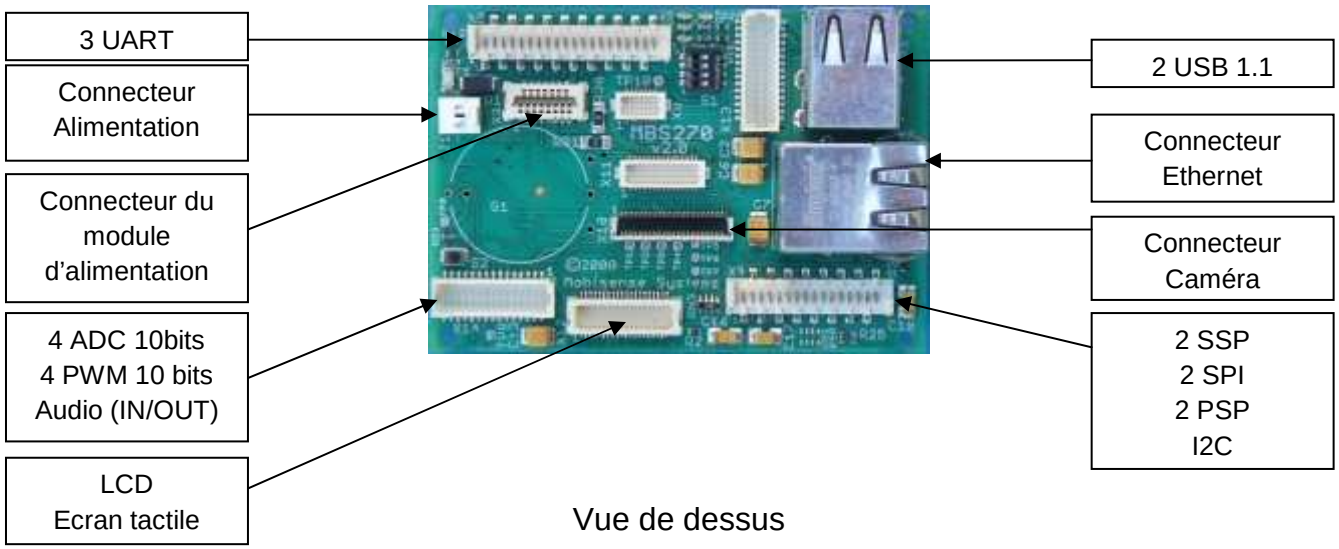
- *Processeur Marvell PXA270 @ 520MHz*
- *64MB SDRAM, 32MB FLASH*
- *Interface caméra 10 bits par FFC 30 voies*
- *Module caméra WVGA 752x480*
- *2 USB 1.1*
- *3 UART*
- *Alimentation 12V, régulé en 5V par un module XP-POWER 2A*
- *Alimentation 3.3V obtenu grâce au module MBSPOW-5V*

Caractéristiques software :

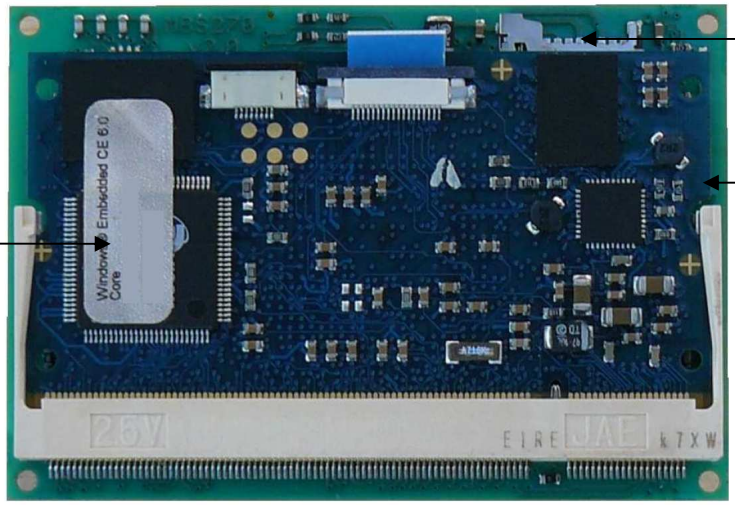
- *Système utilisé : Linux 2.6.20.*
- *Effectue un traitement d'images afin de déterminer la position des palets sur le terrain, la position du réservoir aléatoire et d'un pré-scannage de la colline.*
- *Elle effectue ces différentes opérations sur demande de la DsPIC Maitre. Le flag de demande est d'abord communiqué à la DsPIC 2 qui relaie l'info via une communication série.*

Communication :

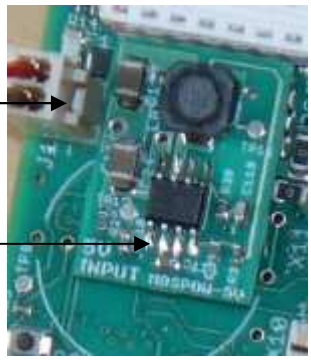
- *Cette carte communique par liaison série avec la carte DsPIC 2.*
- *Nous utilisons l'Ethernet pour la programmer.*
- *Liaison série pour une connexion SSL/Debug.*

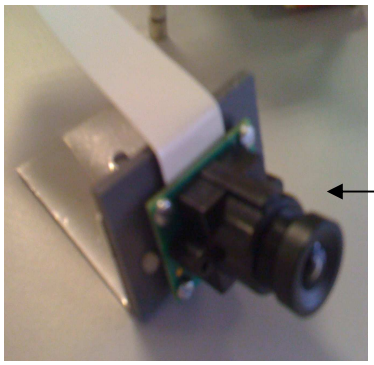


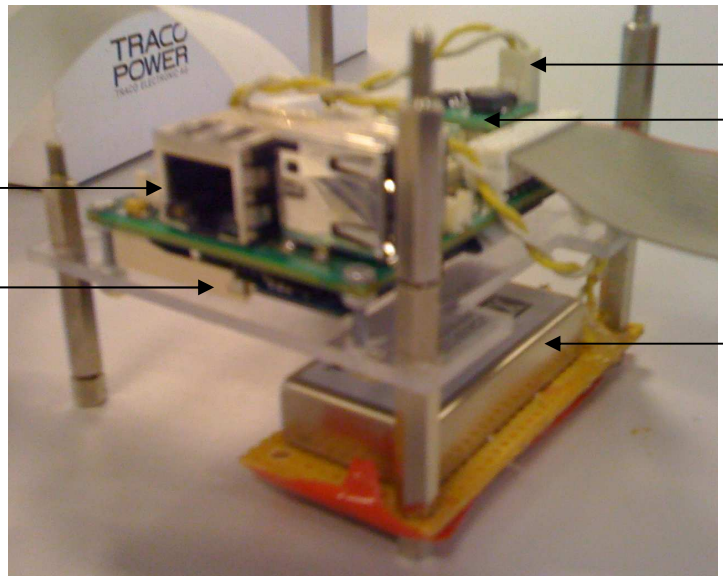
Vue de dessus



Vue de dessous







2.8. Capteurs employés

- **SICK Éliminateur d'arrière plan (EAP)** :

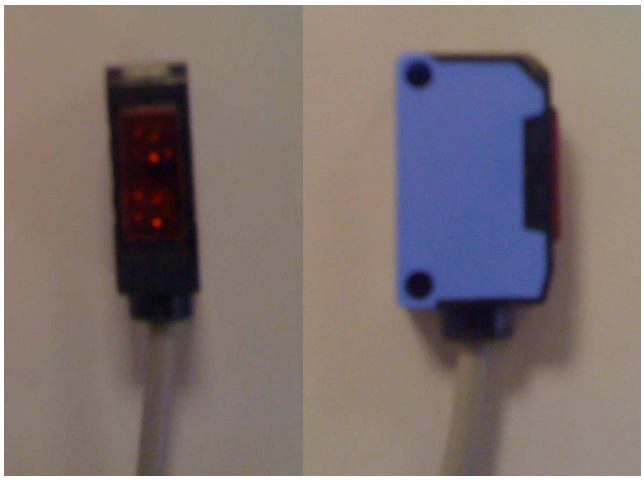
  Réf: WT150-P162

  Fabricant: SICK OPTIC ELECTRONIC

  Type : Détecteur Photoélectrique

  Réglage : seuil réglable à l'aide d'un potentiomètre.

  Utilisation : Pour la détection de palets et pour le scannage de la colline.



- **Ultra Son :**

  Réf: UNDK 20P 6914 S35A

  Fabricant : BAUMER

  Type : Ultra Son

  Réglage : seuil de détection par apprentissage.

  Utilisation : pour la détection adversaire.

3. Informatique

3.1. Communication entre les cartes

L'ensemble des cartes électronique du robot communique grâce au bus I2C implanté au sein du robot. Physiquement une nappe relie entre elle les cartes utilisant ce protocole de communication, de sorte à mettre en commun les deux lignes de communication SDA (Signal Data) et SCL (Signal Clock).

Le protocole I2C requière une carte maître et au minimum une carte esclave. Dans l'architecture électronique de notre robot, la DsPIC Maitre est la carte maître sur l'I2C, et toutes les autres cartes sont esclaves, nous avions ainsi 14 microcontrôleurs sur le même bus.

Les adresses I2C des cartes sont les suivantes :

Adresse Système I2C		
Carte	**Ecriture**	**Lecture**
Mobisense	0x60	0x61
DsPIC Maître	0x40	0x41
DsPIC 2	0x48	0x49
DsPIC Stratégie	0x10	0x11
Carte Puissance 1 PIC_U2	0x20	0x21
Carte Puissance 1 PIC_U3	0x24	0x25
Carte Interconnexion 1 PIC_U1	0x30	0x31
Carte Interconnexion 1 PIC_U2	0x34	0x35
Carte Interconnexion 1 PIC_U3	0x38	0x39
Carte Interconnexion 1 PIC_U4	0x3C	0x3D
Carte Interconnexion 2 PIC_U1	0x50	0x51
Carte Interconnexion 2 PIC_U2	0x54	0x55
Carte Interconnexion 2 PIC_U3	0x58	0x59
Carte Interconnexion 2 PIC_U4	0x5C	0x5D

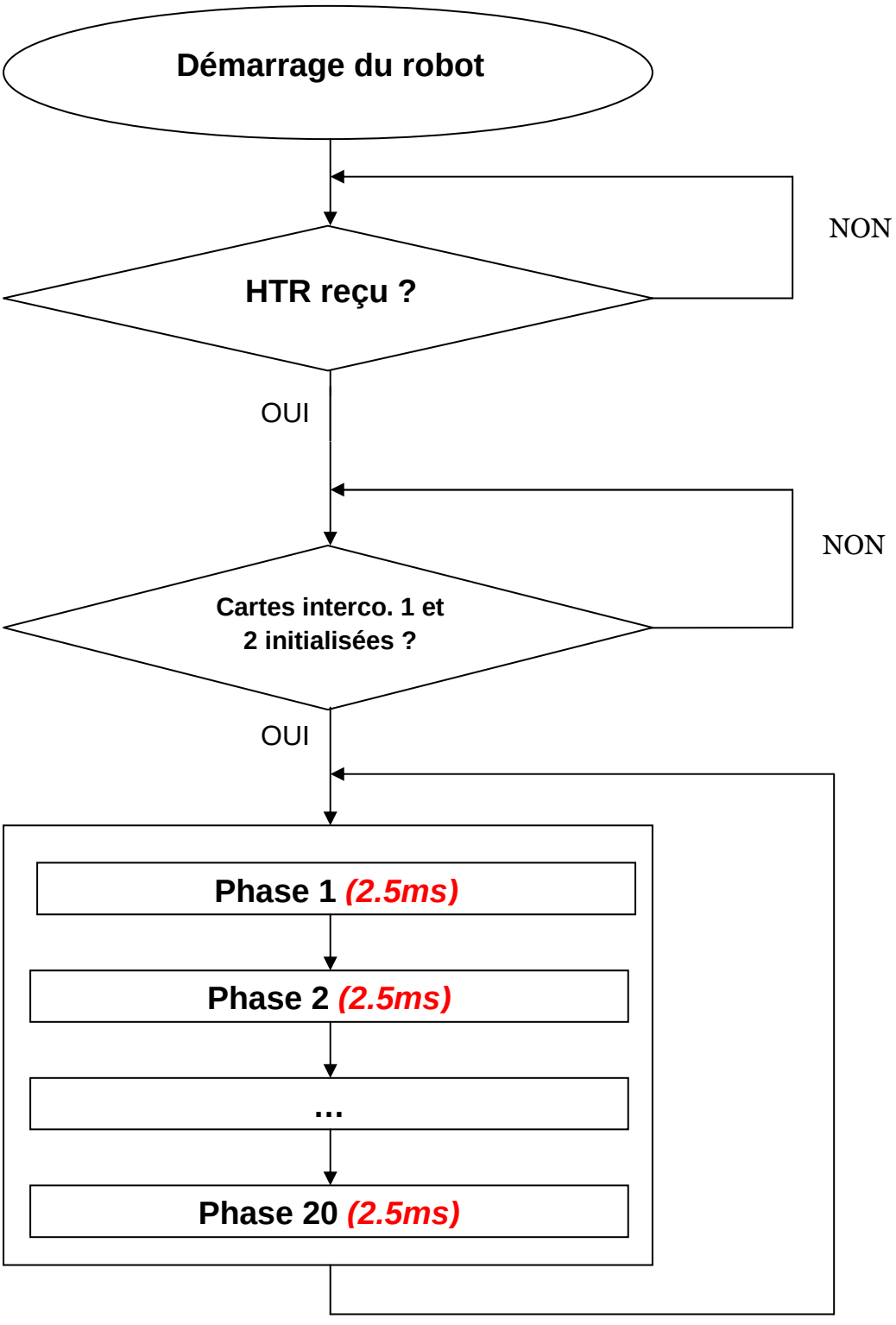
3.2. Carte DsPIC Maître

3.2.1. Architecture du programme

Le programme de la DsPIC Maître est décomposé en plusieurs parties. Avant toute chose, la DsPIC attend de recevoir la HTR générée par la DsPIC Stratégie. Tant qu'elle n'a pas reçu ce signal, la DsPIC maître reste bloquée en attente de celui-ci. Une fois cette étape passée, il y a la phase d'initialisation des cartes interconnexion, qui permet de faire les réglages nécessaires pour que les servomoteurs puissent fonctionner correctement. En effet, afin de pouvoir contrôler convenablement les

différents modèles de servomoteurs utilisés dans le robot, nous devons spécifier la commande qui correspond aux angles 0 et 180 pour chacun d'entre eux.
Après cela, le programme principal peut être lancé. Ce dernier est constitué d'une boucle de 50ms, divisé en 20 phases de 2.5ms. Cette méthode nous permet ainsi d'avoir une parfaite synchronisation de l'ensemble des cartes du robot, mais cela permet aussi une bonne organisation des tâches à effectuer au sein de la Dspic Maître.

Nous pouvons schématiser l'exécution du code de la DsPIC Maitre ainsi :



Afin de garantir un fonctionnement correct du programme, il est important d'équilibrer judicieusement le contenu des différentes phases d'autant plus qu'à chaque phase, le robot effectue son calcul de position (réceptions du nombre d'impulsions codeurs par I2C depuis 2.5ms puis calculs) qui lui prend 1.8ms sur les 2.5ms.

C'est pourquoi nous avons choisi de répartir les différentes tâches de la manière suivante :

Répartition des tâches dans les 20 phases de 2.5ms	
Phase	**Tâches**
1	Envoi de données Position/Contacts/Capteurs/Flag vers DsPIC Stratégie.
2	Automatisme.
3	Envoi de commandes servomoteurs vers carte interconnexion 1.
4	Envoi de commandes servomoteurs vers carte interconnexion 2.
5	Envoi de données Position/Contacts/Capteurs/Flag vers DsPIC Stratégie.
6	Réception de données depuis carte DsPIC 2.
7	Envoi de données vers un PC pour du debug avec notre programme VB.
8	Transaction avec la caméra Mobisense.
9	Envoi de données Position/Contacts/Capteurs/Flag vers DsPIC Stratégie.
10	Envoi de commandes moteurs vers carte Puissance.
11	
12	
13	Envoi de données Position/Contacts/Capteurs/Flag vers DsPIC Stratégie.
14	Surveillance de demande d'initialisation positions depuis DsPIC Stratégie.
15	Communication par liaison série vers un terminal.
16	Vérification initialisation des cartes interconnexion 1 et 2.
17	Envoi données Position/Contacts/Capteurs/Flag vers DsPIC Stratégie.
18	
19	
20	

Afin de gérer nos communications I2C, UART mais aussi la HTR et la découpe de nos phases en 2.5ms, nous utilisons des interruptions.

Voici un tableau récapitulatif des interruptions utilisées :

Interruptions principales utilisées	
Nom	**Fonction**
_INT0Interrupt	Interruption pour HTR.
_T1Interrupt	Interruption pour le Timer1, permet de générer nos phases de 2.5ms.
_MI2CInterrupt	Interruption pour la gestion des communications I2C.
_U1RXInterrupt	Interruption pour la gestion de données entrantes par UART.
_U1TXInterrupt	Interruption pour la gestion de données sortantes par UART.

3.2.2. Gestion de l'I2C

La DsPIC Maître est donc la carte de notre robot qui gère les communications I2C. Cela signifie que si une carte A veut communiquer avec une carte B, et bien elle devra d'abord envoyer les données à la DsPIC Maître qui se chargera ensuite de transférer les informations à la carte B. Sur un système comme le notre où 14 microcontrôleurs sont en mesure de communiquer sur ce bus I2C, il devient donc très important de minimiser les données envoyées et de procéder à l'envoi de façon organisé.

Les données envoyées par la carte DsPIC Maître aux autres cartes et stockées dans le Buffer TX sont les suivantes :

Buffer I2C TX DsPIC Maître			
Phase	**Carte Cible**	**Offset**	**Données**
1	DsPIC Stratégie	0x00	Syst_Position.Cod_Folle.X.Octet.H
		0x01	Syst_Position.Cod_Folle.X.Octet.L
		0x02	Syst_Position.Cod_Folle.Y.Octet.H
		0x03	Syst_Position.Cod_Folle.Y.Octet.L
		0x04	Syst_Position.Cod_Folle.THETA.Octet.H
		0x05	Syst_Position.Cod_Folle.THETA.Octet.L
		0x06	Syst_Position.Cod_Folle.Distance_parcourue.Octet.H
		0x07	Syst_Position.Cod_Folle.Distance_parcourue.Octet.L
		0x08	Syst_StatusPC.ct_part1.Octet
		0x09	Syst_StatusPC.capt_part1.Octet
		0x10	Syst_StatusPC.Flag1.Octet
3	Interco.1PIC_U1	0x66	etat_mesure_courant
		0x67	Servo[1].octet_H
		0x68	Servo[1].octet_L
	Interco.1PIC_U1	0x6C	etat_mesure_courant
		0x6D	Servo[2].octet_H
		0x6E	Servo[2].octet_L
	Interco.1PIC_U2	0x66	etat_mesure_courant
		0x67	Servo[3].octet_H
		0x68	Servo[3].octet_L
	Interco.1PIC_U2	0x6C	etat_mesure_courant
		0x6D	Servo[4].octet_H
		0x6E	Servo[4].octet_L
	Interco.1PIC_U3	0x66	etat_mesure_courant
		0x67	Servo[5].octet_H
		0x68	Servo[5].octet_L
	Interco.1PIC_U3	0x6C	etat_mesure_courant
		0x6D	Servo[6].octet_H
		0x6E	Servo[6].octet_L
	Interco.1PIC_U4	0x66	etat_mesure_courant
		0x67	Servo[7].octet_H
		0x68	Servo[7].octet_L

	Interco.1PIC_U4	0x6C	etat_mesure_courant
		0x6D	Servo[8].octet_H
		0x6E	Servo[8].octet_L
4	Interco.2PIC_U1	0x66	etat_mesure_courant
		0x67	Servo[9].octet_H
		0x68	Servo[9].octet_L
	Interco.2PIC_U1	0x6C	etat_mesure_courant
		0x6D	Servo[10].octet_H
		0x6E	Servo[10].octet_L
	Interco.2PIC_U2	0x66	etat_mesure_courant
		0x67	Servo[11].octet_H
		0x68	Servo[11].octet_L
	Interco.2PIC_U2	0x6C	etat_mesure_courant
		0x6D	Servo[12].octet_H
		0x6E	Servo[12].octet_L
	Interco.2PIC_U3	0x66	etat_mesure_courant
		0x67	Servo[13].octet_H
		0x68	Servo[13].octet_L
	Interco.2PIC_U3	0x6C	etat_mesure_courant
		0x6D	Servo[14].octet_H
		0x6E	Servo[14].octet_L
	Interco.2PIC_U4	0x66	etat_mesure_courant
		0x67	Servo[15].octet_H
		0x68	Servo[15].octet_L
	Interco.2PIC_U4	0x6C	etat_mesure_courant
		0x6D	Servo[16].octet_H
		0x6E	Servo[16].octet_L
5	DsPIC Stratégie	0x00	Syst_Position.Cod_Folle.X.Octet.H
		0x01	Syst_Position.Cod_Folle.X.Octet.L
		0x02	Syst_Position.Cod_Folle.Y.Octet.H
		0x03	Syst_Position.Cod_Folle.Y.Octet.L
		0x04	Syst_Position.Cod_Folle.THETA.Octet.H
		0x05	Syst_Position.Cod_Folle.THETA.Octet.L
		0x06	Syst_Position.Cod_Folle.Distance_parcourue.Octet.H
		0x07	Syst_Position.Cod_Folle.Distance_parcourue.Octet.L
		0x08	Syst_StatusPC.ct_part1.Octet
		0x09	Syst_StatusPC.capt_part1.Octet
		0x10	Syst_StatusPC.Flag1.Octet
8	Mobisense	0x00	Flag_action
9	DsPIC Stratégie	0x00	Syst_Position.Cod_Folle.X.Octet.H
		0x01	Syst_Position.Cod_Folle.X.Octet.L
		0x02	Syst_Position.Cod_Folle.Y.Octet.H
		0x03	Syst_Position.Cod_Folle.Y.Octet.L
		0x04	Syst_Position.Cod_Folle.THETA.Octet.H
		0x05	Syst_Position.Cod_Folle.THETA.Octet.L
		0x06	Syst_Position.Cod_Folle.Distance_parcourue.Octet.H
		0x07	Syst_Position.Cod_Folle.Distance_parcourue.Octet.L

		0x08	Syst_StatusPC.ct_part1.Octet
		0x09	Syst_StatusPC.capt_part1.Octet
		0x10	Syst_StatusPC.Flag1.Octet
10	Puissance 1 PIC_U2	0x00	Syst_Puissance.Verrou_Moteurs
		0x01	cmd_moteur[MOT_POMPE].arret_marche
		0x02	cmd_moteur[MOT_POMPE].vitesse
		0x03	cmd_moteur[MOT_POMPE].sens
	Puissance 1 PIC_U3	0x00	Syst_Puissance.Verrou_Moteurs
		0x01	cmd_moteur[MOT_POMPE].arret_marche
		0x02	cmd_moteur[MOT_POMPE].vitesse
		0x03	cmd_moteur[MOT_POMPE].sens
13	DsPIC Stratégie	0x00	Syst_Position.Cod_Folle.X.Octet.H
		0x01	Syst_Position.Cod_Folle.X.Octet.L
		0x02	Syst_Position.Cod_Folle.Y.Octet.H
		0x03	Syst_Position.Cod_Folle.Y.Octet.L
		0x04	Syst_Position.Cod_Folle.THETA.Octet.H
		0x05	Syst_Position.Cod_Folle.THETA.Octet.L
		0x06	Syst_Position.Cod_Folle.Distance_parcourue.Octet.H
		0x07	Syst_Position.Cod_Folle.Distance_parcourue.Octet.L
		0x08	Syst_StatusPC.ct_part1.Octet
		0x09	Syst_StatusPC.capt_part1.Octet
		0x10	Syst_StatusPC.Flag1.Octet
17	DsPIC Stratégie	0x00	Syst_Position.Cod_Folle.X.Octet.H
		0x01	Syst_Position.Cod_Folle.X.Octet.L
		0x02	Syst_Position.Cod_Folle.Y.Octet.H
		0x03	Syst_Position.Cod_Folle.Y.Octet.L
		0x04	Syst_Position.Cod_Folle.THETA.Octet.H
		0x05	Syst_Position.Cod_Folle.THETA.Octet.L
		0x06	Syst_Position.Cod_Folle.Distance_parcourue.Octet.H
		0x07	Syst_Position.Cod_Folle.Distance_parcourue.Octet.L
		0x08	Syst_StatusPC.ct_part1.Octet
		0x09	Syst_StatusPC.capt_part1.Octet
		0x10	Syst_StatusPC.Flag1.Octet

Les données reçues par la carte DsPIC Maître et stockées dans le Buffer RX sont les suivantes :

Buffer I2C RX DsPIC Maître			
Phase	**Carte émettrice**	**Offset**	**Données**
TOUTES	Interco 1 PIC_U3	0x14	Syst_Position.Delta_Cod.Roue_Fol_G.Octet.H
		0x15	Syst_Position.Delta_Cod.Roue_Fol_G.Octet.L
	Interco 1 PIC_U4	0x16	Syst_Position.Delta_Cod.Roue_Fol_D.Octet.H
		0x17	Syst_Position.Delta_Cod.Roue_Fol_D.Octet.L
6	DsPIC 2	0x00	1
		0x01	i2c_rx_dspic2.part1.Octet
		0x02	i2c_rx_dspic2.part2.Octet

		0x03	i2c_rx_dspic2.part3.Octet
		0x04	i2c_rx_dspic2.part4.Octet
14	DsPIC Stratégie	0x00	Syst_Position.Init_Terrain.Commande_Init
		0x01	Syst_Position.Init_Terrain.X.Octet.H
		0x02	Syst_Position.Init_Terrain.Y.Octet.H
		0x03	Syst_Position.Init_Terrain.THETA.Octet.H
		0x04	Syst_Position.Init_Terrain.X.Octet.L
		0x05	Syst_Position.Init_Terrain.Y.Octet.L
		0x06	Syst_Position.Init_Terrain.THETA.Octet.L
		0x07	Syst_StatusPC.Flag1.Octet

Le descriptif des données précédentes sont les suivantes :

Données	Description
Syst_Position.Cod_Folle.X.Octet.H	Coordonnée en X du robot
Syst_Position.Cod_Folle.X.Octet.L	
Syst_Position.Cod_Folle.Y.Octet.H	Coordonnée en Y du robot
Syst_Position.Cod_Folle.Y.Octet.L	
Syst_Position.Cod_Folle.THETA.Octet.H	Angles THETA du robot
Syst_Position.Cod_Folle.THETA.Octet.L	
Syst_Position.Cod_Folle.Distance_parcourue.Octet.H	Distance parcourue par le robot
Syst_Position.Cod_Folle.Distance_parcourue.Octet.L	
Syst_StatusPC.ct_part1.Octet	Statut des contacts
Syst_StatusPC.capt_part1.Octet	Statut des capteurs
Syst_StatusPC.Flag1.Octet	Statut des flags de stratégie
etat_mesure_courant	Etat de la mesure de courant
Servo[N].octet_H	Position du servomoteur N
Servo[N].octet_L	
Flag_action	Flag pour une demande à la caméra
Syst_Puissance.Verrou_Moteurs	Vérouillage moteur
cmd_moteur[MOT_POMPE].arret_marche	Etat du moteur (Marche/Arrêt)
cmd_moteur[MOT_POMPE].vitesse	Vitesse du moteur
cmd_moteur[MOT_POMPE].sens	Sens du moteur
Syst_Position.Delta_Cod.Roue_Fol_G.Octet.H	Nombre d'impulsion roue folle gauche depuis 2.5ms
Syst_Position.Delta_Cod.Roue_Fol_G.Octet.L	
Syst_Position.Delta_Cod.Roue_Fol_D.Octet.H	Nombre d'impulsion roue folle droite depuis 2.5ms
Syst_Position.Delta_Cod.Roue_Fol_D.Octet.L	
i2c_rx_dspic2.part1.Octet	Etat des contacts, capteurs
i2c_rx_dspic2.part2.Octet	
i2c_rx_dspic2.part3.Octet	
i2c_rx_dspic2.part4.Octet	
Syst_Position.Init_Terrain.Commande_Init	Init de la position du robot si égal à 1
Syst_Position.Init_Terrain.X.Octet.L	Position X d'init
Syst_Position.Init_Terrain.X.Octet.H	
Syst_Position.Init_Terrain.Y.Octet.L	Position Y d'init
Syst_Position.Init_Terrain.Y.Octet.H	

Syst_Position.Init_Terrain.THETA.Octet.L	Angle THETA d'init
Syst_Position.Init_Terrain.THETA.Octet.H	

3.2.3. Automatisme

Cette année l'automatisme du robot devait gérer nos deux pinces à palets, le bras pour les linteaux et les deux pinces à linteaux, ce qui fait 13 servomoteurs.

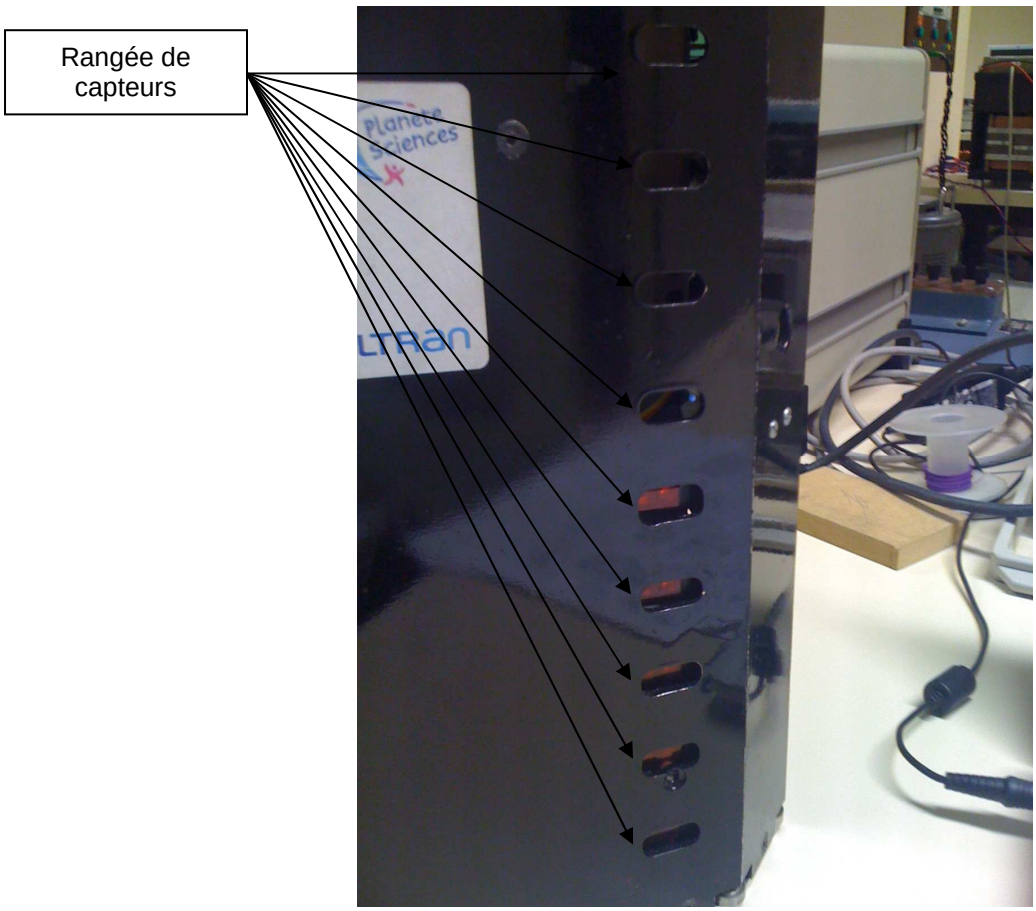
Voici les différentes actions prévues par l'automatisme :

- **Gestion du ramassage de palets :**
  Dans le cas d'un ramassage de palets dont la position est aléatoire ou dans le cas d'un ramassage de palets dans un réservoir, le système comptait le nombre de palets saisis à l'aide de 3 SICK EAP situé sur le devant du robot. C'est ainsi que le robot arrivait à synchroniser les mouvements opposés des pinces et les déplacements du petit bras permettant le calage des palets dans le robot du coté gauche ou droit. Dès que 4 palets étaient maintenus par les pinces, l'information était remontée à la DsPIC Stratégie qui pouvait alors procéder à une dépose d'un temple. A la fin du ramassage, l'automatisme des pinces se mettait à la hauteur de la prochaine dépose la plus probable (si aucun temple n'avait était déposé avant ce ramassage, les pinces montaient à la hauteur de la colline, sinon les pinces montaient à la hauteur d'un temple déposé sur la colline). Par la suite, en attendant un ordre de la DsPIC Stratégie, les pinces restaient à cette dernière position.

- **Gestion du ramassage de linteau :**
  Le ramassage linteau se faisait par la gestion du bras, mais aussi des deux pinces à linteaux. Nous pouvions en prendre jusqu'à trois en même temps, en les stockant à l'aide des deux pinces à linteaux, et en gardant un dernier aspiré par les pompes à vide. Afin de gérer le stockage et le déstockage, le système devait compter, la encore, le nombre d'éléments présents dans le robot.

- **Gestion de la dépose d'un temple :**
  Le système était capable de mémoriser plusieurs informations afin d'avoir un comportement logique selon ce que le robot va déposer et en fonction de ce que le robot avait peut être déjà déposé. Il mémorisait ainsi les ordres de dépose envoyé par la DsPIC Stratégie, les cordonnées (X, Y, THETA) du calage avant la dépose et la hauteur finale de la dernière dépose. Ainsi l'automatisme pouvait savoir à quelle hauteur il devait déposer et dans quel ordre il devait faire ses constructions.

- **Gestion du scan colline :**
  Ne sachant pas les problèmes que nous allions rencontrer par la suite, cette fonction avait été implantée au sein de l'automatisme, mais n'a pû être utilisée lors de la coupe. Cette action aurait était utilisée lors des 16ième de finale, afin de s'assurer que l'adversaire n'avait pas déposé d'éléments sur nos constructions. Nous avons pour cela une rangée de neuf capteurs SICK EAP sur le coté gauche du robot, qui permettait de voir jusqu'à une hauteur de 29cm, soit huit éléments de jeu empilés les uns sur les autres en zone 3 (colline). Pour procéder à un scan, le robot devait tourner autour de la moitié de la colline, pendant que l'automatisme s'occupait de mémoriser et de traiter les données renvoyées par les capteurs. Le système était alors capable de détecter s'il était possible de déposer sur la demi-colline scannée, si un temple était en place ou si de simples colonnes avaient été déposées. Si la dépose était possible, en fonction des éléments de jeu présents dans le robot, le système calculait alors une position (X, Y, THETA) de calage, envoyait cela à la DsPIC Stratégie de sorte à positionner le robot et d'effectuer la dépose en fonction des constructions présentes sur la colline.



- **Gestion de la caméra :**
  Nous souhaitions utiliser la caméra dans différentes tâches tout au long du match. En début de celui-ci, nous avions prévu de faire une acquisition du terrain de sorte à obtenir la position des palets devant le robot. Ceci présente un réel avantage car étant devant le robot, cela nous aurait permis de prendre très rapidement nos quatre palets nécessaires à la construction du premier temple. Le fonctionnement d'un tel système n'est pas très compliqué : la caméra fait son traitement, elle renvoie les informations obtenues à la DsPIC Maître qui les communique à la DsPIC Stratégie. Cette dernière peut alors utiliser ces informations pour faire déplacer le robot.
  Nous souhaitions aussi utiliser la caméra pour effectuer un pré-scannage de la colline. Après calage face à cette dernière, la caméra pouvait indiquer à la DsPIC Stratégie si la zone dans laquelle nous voulions déposer était déjà occupée par l'adversaire ou pas. Ceci nous permettait de gagner du temps en évitant de scanner de la colline si la zone de dépose souhaitée n'était pas occupée par l'adversaire.
  La dernière fonction prévue était la recherche de la position du réservoir aléatoire. Lors d'une dépose ou d'une autre action présentant le robot face à la zone de départ adverse et lors d'une immobilité, nous étions capables de déterminer la position du réservoir aléatoire par un traitement d'images.

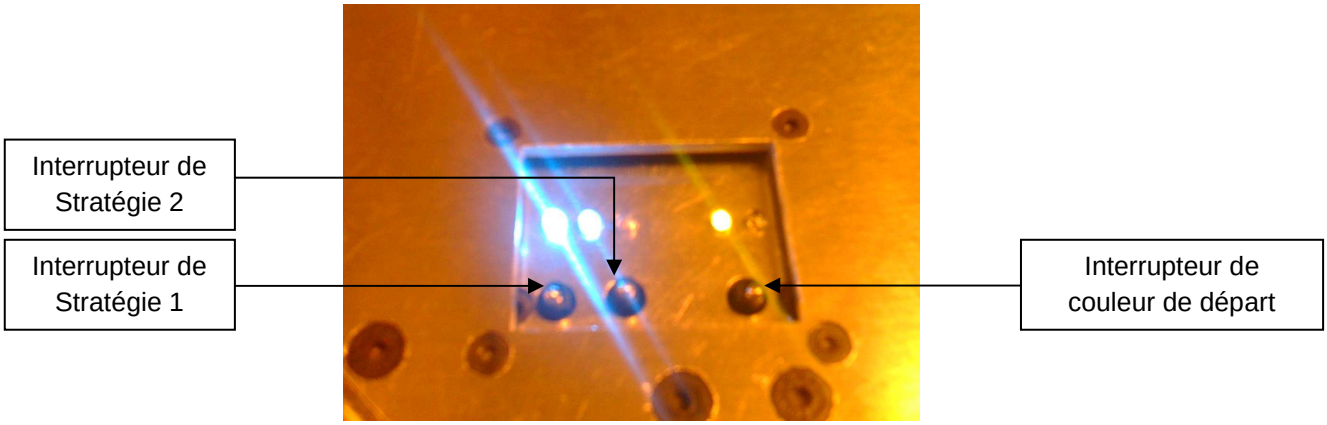
  Cependant, tous ces développements n'ont pu être mis en œuvre lors de la coupe, à cause de nos gros problèmes de déplacements qui ne permettaient pas une interaction entre nos mouvements sur le terrain et la gestion des retours de la caméra.

3.3. Carte DsPIC Stratégie

3.3.1. Architecture du programme

Le changement de l'architecture électronique causé par la perte de notre ancien PC embarqué nous a obligés à recoder l'intégralité du nouveau « cerveau » du robot. Pour cela nous sommes tout d'abord partis sur l'utilisation d'un OS embarqué nommé µCOS2 compatible DsPIC, afin de pouvoir bénéficier d'une architecture temps réel. Cet OS n'étant pas basé sur notre microcontrôleur DsPIC 30F6012A, nous avons dû commencer par un portage pour l'adapter sur notre plate-forme. Cependant, les trois mois ne nous ont pas permis de corriger toutes les erreurs causées par le portage, ce qui nous a forcés à désactiver la partie temps réelle de l'OS, et d'utiliser un système plus classique de TIMER. Ce dernier, simple à mettre en place, ne nous a posés aucun problème durant la coupe et nous a permis de passer à l'étape suivante, la mise en place de l'asservissement.

L'architecture de ne notre programme a été conçu pour que nous puissions intégrer plusieurs stratégies au sein du robot, mais aussi de sorte à ce que nous puissions choisir l'une d'elle sur le terrain à la coupe, par l'intermédiaire de deux interrupteurs.



L'utilisation des interrupteurs se fait la manière suivante :

	Etat du contact stratégie 1	Etat du contact stratégie 2
Stratégie 1	0	0
Stratégie 2	0	1
Stratégie 3	1	0
Stratégie 4	1	1

Remarque :

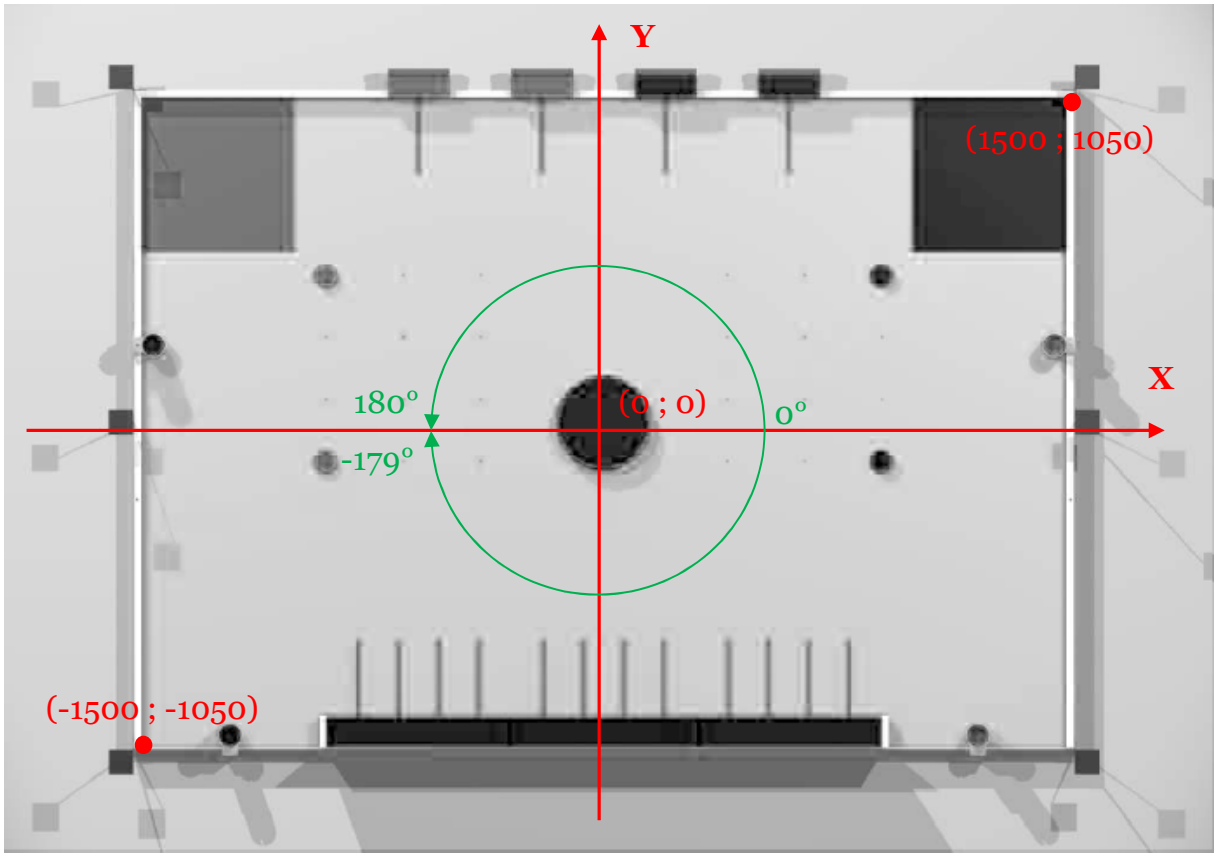
- *0 signifie un état bas*
- *1 signifie un état haut*

La structure du programme a été codée de la manière suivante :

```
#define STRATEGIE_HOMOLOGATION                  //La ligne non commentée définit la stratégie employée
//#define STRATEGIE_AVANT_16IEME                //
//#define STRATEGIE_16IEME                      //

//*********************************************************
//Stratégie employée pour l'homologation
//*********************************************************
#ifdef      STRATEGIE_HOMOLOGATION

            if ( (!CT_STRAT_1) && (!CT_STRAT_2) )
                        //Stratégie 1

            else if ( (!CT_STRAT_1) && (CT_STRAT_2) )
                        //Stratégie 2

            else if ( (CT_STRAT_1) && (!CT_STRAT_2) )
                        //Stratégie 3

            else
                        //Stratégie 4

            end if

//*********************************************************
//Stratégie employée pour les matchs avant les 16ième de finale
//*********************************************************
#elif       STRATEGIE_AVANT_16IEME

            If ( (!CT_STRAT_1) && (!CT_STRAT_2) )
                        //Stratégie 1

            else if ( (!CT_STRAT_1) && (CT_STRAT_2) )
                        //Stratégie 2

            else if ( (CT_STRAT_1) && (!CT_STRAT_2) )
                        //Stratégie 3

            else
                        //Stratégie 4

            end if

//*********************************************************
//Stratégie employée pour les matchs A PARTIR DES 16ième de finale
//*********************************************************
#elif       STRATEGIE_16IEME

            If ( (!CT_STRAT_1) && (!CT_STRAT_2) )
            //Stratégie 1

            else if ( (!CT_STRAT_1) && (CT_STRAT_2) )
            //Stratégie 2

            else if ( (CT_STRAT_1) && (!CT_STRAT_2) )
            //Stratégie 3

            else
                        //Stratégie 4

            end if
#else                                               //Stratégie de test

#endif
```

Remarque : *Cette architecture nous permet de coder jusqu'à 4 stratégies selon la phase de match dans laquelle nous nous trouvons (homologation, avant 16ième de final ou après 16ième de final), mais nous pouvons augmenter très facilement ce nombre à l'aide de 'defines'.*

Notre robot est capable de se repérer précisément (à plus ou moins 2mm) sur le terrain grâce aux roues codeuses. Pour exploiter ces dernières, nous avons défini un repère orthonormé de la manière suivante :



3.3.2. Asservissement

Toutes les 2.5ms, le nombre d'impulsions codeur est remonté à la DsPIC maître depuis la carte interconnexion 1. La DsPIC maître renvoie alors directement ses informations à la DsPIC stratégie qui s'occupe de faire les calculs d'asservissement toutes les 10ms et d'envoyer ensuite les consignes à la carte puissance. Cette dernière effectue une boucle de courant sur nos moteurs. Nous avons donc la possibilité de faire nos asservissements en 2.5ms, cependant nous avons préféré

garantir une stabilité au sein de notre robot, car le fait de diminuer l'asservissement à 2.5ms demande de nombreux tests afin que nous puissions valider ce passage.
L'asservissement du robot se fait par l'appel d'une fonction toutes les 10ms à l'aide d'une interruption.
Pour les déplacements du robot, nous utilisons un asservissement en position et en orientation.

Les consignes distance et orientation suivent le schéma fonctionnel suivant :

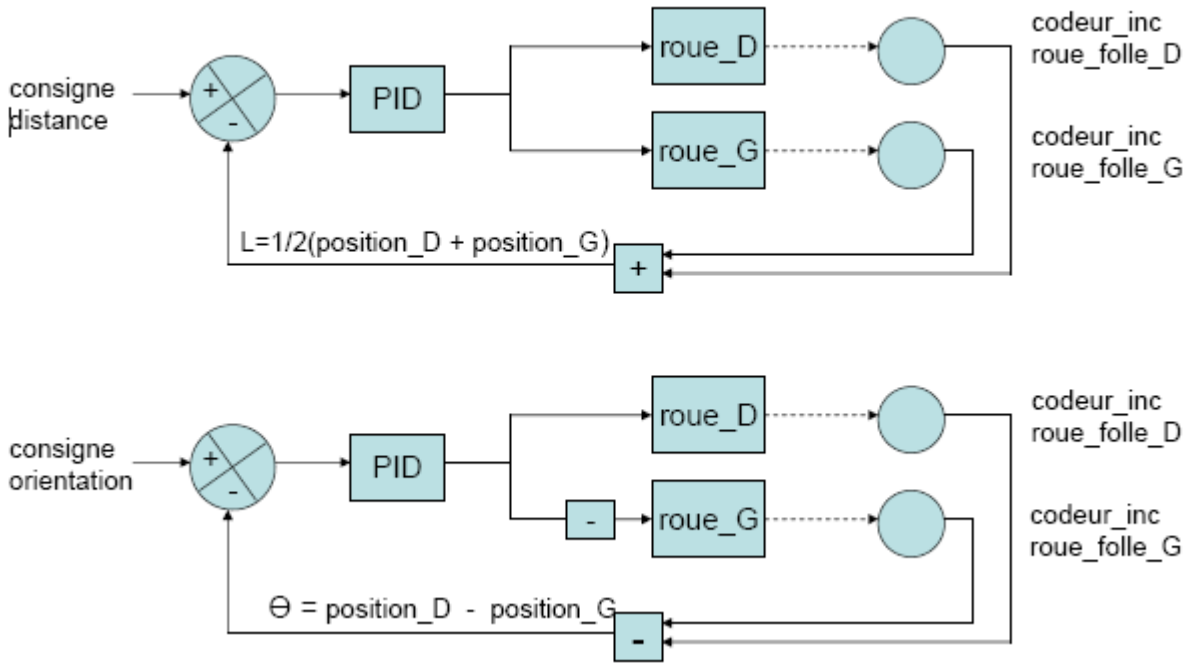
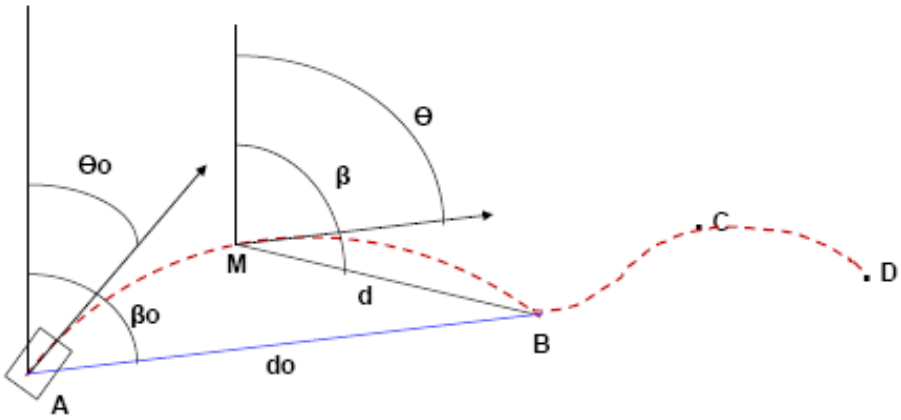


Schéma bloc d'asservissement polaire

Remarque : *on ne fait pas apparaître les saturations et les rampes de commande moteur.*

L'intérêt de l'asservissement polaire apparaît plus évident dans l'exemple suivant :

On propose de se déplacer du point A vers le point B puis vers C, puis D, …avec une orientation initiale **Θo** quelconque.



La trajectoire curviligne de A vers B (pointillé) est obtenue par la loi de commande suivante:

```
distance = (roue_d + roue_g) /2
vitesse = (vitesse_roue_d + vitesse_roue_g)/2
ecart = consigne_distance - distance
commande = ecart * GAIN_PROPORTIONNEL_DISTANCE
commande_distance = commande – GAIN_DERIVE_DISTANCE*vitesse
orientation  = roue_D – roue_G
vitesse_orientation=vitesse_roue_D – vitesse_roue_G
ecart = consigne_orientation - orientation
```

L'algorithme complet en asservissement polaire :

```
commande = ecart * GAIN_PROPORTIONNEL_ROTATION
commande_rotation = commande - GAIN_DERIVE_ROTATION*vitesse_orientation
commande_roue_D= commande_distance + commande_rotation
commande_roue_G= commande_distance - commande_rotation
vitesse_orientation=vitesse_roue_D – vitesse_roue_G
ecart = consigne_orientation - orientation
```

L'idée est d'asservir à chaque instant d'échantillonnage (point M sur la figure) l'orientation **Θ** du robot à l'orientation **β** du point B. Ainsi le robot décrit une trajectoire curviligne comme s'il était attiré par aimantation vers le point B. A noter que la forme de la trajectoire n'est pas imposée. La seule chose qu'on impose au robot est de passer par le point B et ensuite en généralisant de passer par C puis D.

3.3.3. Fonctions de déplacement

Afin de pouvoir faire se déplacer notre robot, nous avons codé des fonctions de déplacement qui permettent une programmation simple de la partie stratégique. En effet l'exécution des commandes se fait alors séquentiellement, ce qui signifie qu'une stratégie de base peut alors se coder sur une 20ène de ligne. De plus chaque fonction de déplacement comporte un timeout, ce qui permet au robot de ne pas rester bloqué contre un élément de jeu ou autre.

Le principe des fonctions de déplacement est simple. Elles modifient les variables d'une structure réservée à l'asservissement, ainsi, lorsque la fonction d'asservissement est appelée toute les 10ms, elle peut alors mettre à jour les consignes à l'aide des modifications apportées par les fonctions de déplacements.

Quant à la structure des fonctions de déplacement, on retrouve à chaque fois une boucle 'while' qui vérifie le timeout et vérifie si la position actuelle du robot est comprise entre deux seuils de la position finale. Dans cette même boucle nous retrouvons aussi une condition d'évitement adversaire qui permet au robot de stopper temporairement l'exécution de la fonction afin de procéder à une action définie.

Les premières fonctions de déplacement codées on été les fonctions suivantes :

```
void avance_ou_recule ( CPU_INT32S consigne_distance_en_mm,
                        CPU_INT32S param_avance_recule,
                        CPU_INT32S vitesse,
                        CPU_INT32S timeout_sortie )
```

Cette fonction permet de faire avancer ou reculer le robot d'une distance donnée en millimètre à une vitesse comprise entre 13 et 30. Le timeout est quant à lui définit en milliseconde, comme dans les fonctions qui suivront.

```
void tourne_angle_repere (      CPU_INT32S consigne_angle,
                                CPU_INT32S vitesse,
                                CPU_INT32S timeout_sortie )
```

Cette fonction fait prendre au robot l'angle **consigne_angle** en degré.
Remarque : *l'angle prit par le robot est l'angle **consigne_angle** de son repère.*

```
void stop ( CPU_INT32S timeout_sortie )
```

La fonction stop, très importante, permet d'arrêter le robot sur place, même s'il n'a pas atteint sa position finale.

Par la suite nous avons pu alors élargir nos déplacements possibles par :

```
void va_vers (      CPU_INT32S x_consigne,
                    CPU_INT32S y_consigne,
                    CPU_INT32S vitesse,
                    CPU_INT32S param_avance_recule,
                    CPU_INT32S timeout_sortie )
```

La fonction **va_vers** permet au robot d'aller à un point (x_consigne, y_consigne) de son repère à une vitesse donnée toujours entre 13 et 30 en avançant ou en reculant. Cette fonction codée quelques semaines avant la coupe, contenait trop d'erreurs pour que nous puissions l'employer, et cela nous à gravement handicapé lors de nos déplacements à la coupe. En effet, l'avantage d'une telle fonction est que le robot peut aller à un point précis du terrain sans que nous nous souciions des distances ou angles que nous devons lui faire faire pour y arriver.

```
void passe_part (   CPU_INT32S x_consigne,
                    CPU_INT32S y_consigne,
                    CPU_INT32S vitesse,
                    CPU_INT32S param_avance_recule,
                    CPU_INT32S timeout_sortie )
```

Cette fonction reprend le principe de la **va_vers** mais permet de garder la vitesse indiquée une fois arrivée au point (x_consigne, y_consigne). Avec une telle fonction nous pouvons donc faire des enchaînements sans arrêt de nos déplacements, et donc gagner du temps.

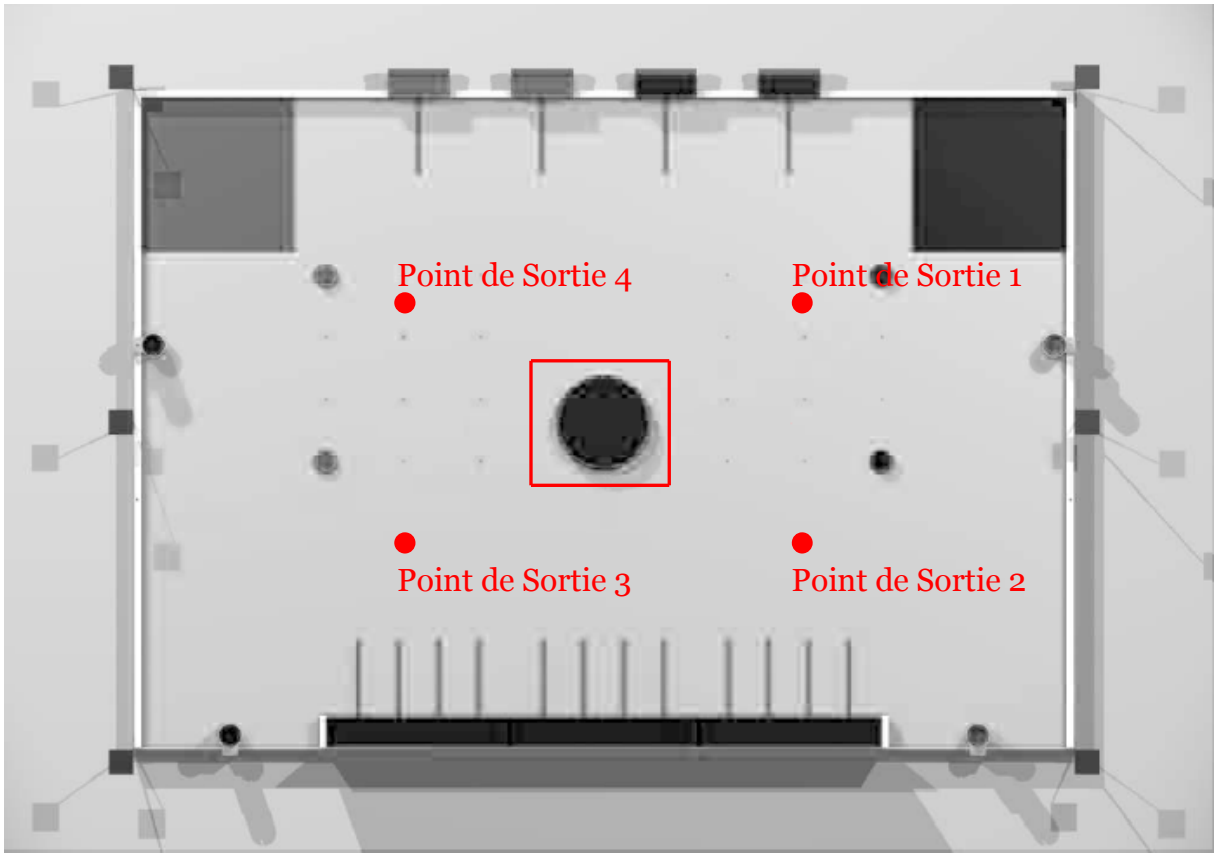
```
void Init_position_robot (CPU_INT32S X_init,
                          CPU_INT32S Y_init,
                          CPU_INT32S THETA_init )
```

Cette dernière fonction, utile lors des recalages, permet d'initialiser la position du robot à un X, Y, THETA souhaité.

3.3.4. Gestion de l'adversaire et des obstacles

Etant une des dernières choses mise en place avant la coupe, notre évitement adversaire et obstacle n'était pas très évolué. Pour ce qui est de l'évitement d'obstacle, celui-ci a été prévu lors de l'emploi de déplacement de type va_vers, donc nous ne l'avons finalement pas utilisé même si cela avait été prévu. Son fonctionnement était le suivant : Nous avons représenté un carré virtuel autour de la colline, si le robot entrait à l'intérieur et qu'il n'était pas en phase de dépose, alors il reculait, faisait une **va_vers** vers un point dit 'de Sortie' selon sa position, et faisait ensuite une **va_vers** en direction de son objectif initial.

Schématiquement cela donne ceci :



Pour l'évitement adversaire, nous vérifions toutes les 10ms l'état des capteurs US afin de déterminer si l'on devait procéder à une action d'évitement adversaire ou pas. Si c'était le cas, une variable globale faisant office de flag était passée à 1. Ensuite, au retour dans la fonction, après la fin de l'interruption, le flag permettait l'exécution de la fonction d'évitement suivante :

```
void stop_evitement ( void )
```

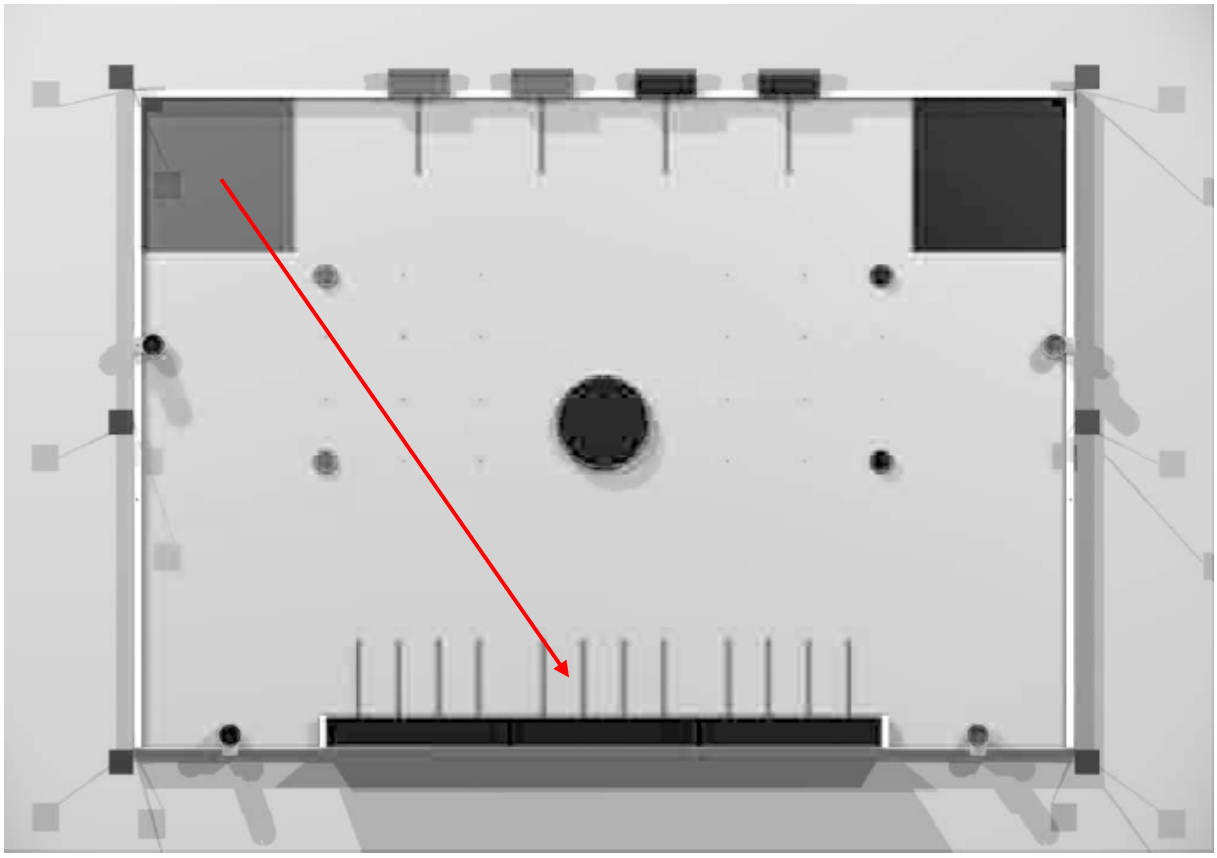
Cette fonction permet tout simplement de stopper le robot, d'attendre que l'adversaire parte, d'attendre encore deux secondes et de reprendre son déplacement.

3.3.5. Stratégie

- **Stratégie d'homologation**

Pour l'homologation du robot nous devions marquer au minimum un point, et nous devions valider notre évitement adversaire. C'est pourquoi nous utilisons une stratégie particulière, afin de limiter les cas « non prévus d'une stratégie complète» et de ne pas nous exposer à des risques inutiles qui se concluraient par une non-homologation. Notre stratégie été très simple, nous avions remarqué qu'en effectuant un départ en diagonal nous pouvions arriver à pousser des palets et ainsi marquer le point requis. Pour l'évitement adversaire nous utilisions les trois US réservés à cet effet afin d'activer notre fonction d'évitement.

Schématiquement notre stratégie d'homologation était la suivante :



- **Stratégie d'avant 16ième de finale**

Notre stratégie principale d'avant 16ième de finale était découpée en 4 phases :

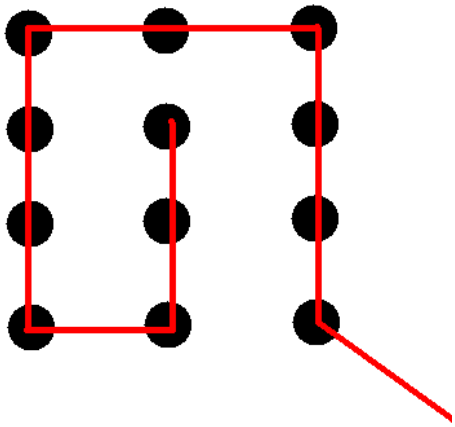
1. Ramassage de 4 palets.
2. Dépose du 1er temple sur la colline (zone 3).
3. Ramassage de deux linteaux.
4. Ramassage de 4 palets au réservoir fixe.
5. Dépose d'un 2ième temple sur notre 1ier temple sur la colline.

Détails des phases :

1. Le ramassage des premiers palets était basé sur l'utilisation de notre caméra embarquée. Celle-ci permettait la détection des palets sur le terrain, ce qui autorisait le robot à choisir les déplacements appropriés afin de pouvoir les ramasser. Cependant, par manque de temps nous n'avons pas eu le temps

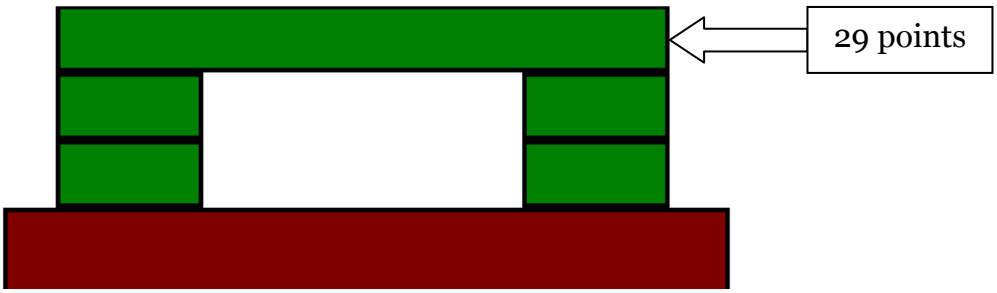
de finir l'intégration de la caméra au sein de notre stratégie, mais nous avons quand même gardé le principe pour le ramassage des palets.

En effet, en étudiant les cartes de positionnement des palets en début de match (information donnée par le règlement), nous avons remarqué que les palets pouvaient être ramassés par un déplacement dit en « escargot ». Une représentation simple peut se faire ainsi :



L'avantage de ramasser les palets de cette manière, est que nous pouvons découper les déplacements en 5 phases, qui peuvent être arrêtés si le nombre suffisant de palets (4 maximum) est atteint.

2. Après le ramassage des palets nous pouvions aller déposer notre premier temple. Nous avons choisi d'aller le construire le plus près du dernier palet ramassé, dans la zone de dépose la plus haute du terrain afin de gagner un maximum de points même si ce n'était pas le choix le plus simple à réaliser. En effet, la colline étant circulaire, le calage demandé, devait être assez précis. Il est à noter aussi que pour être considéré comme valable, la construction ne devait pas dépasser de la colline.

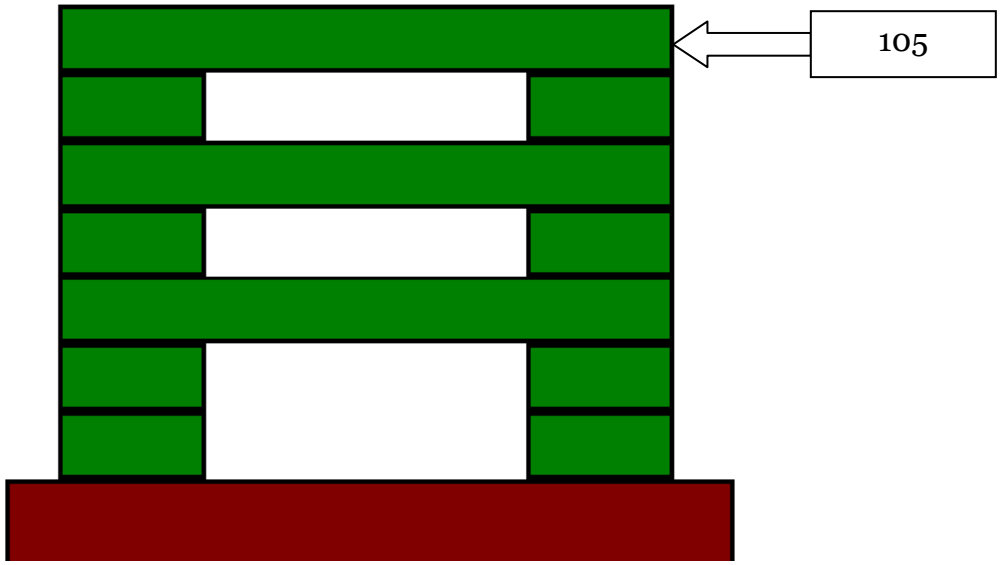


3. Une fois construit, nous allions ramasser deux linteaux pour préparer la construction de notre second temple.

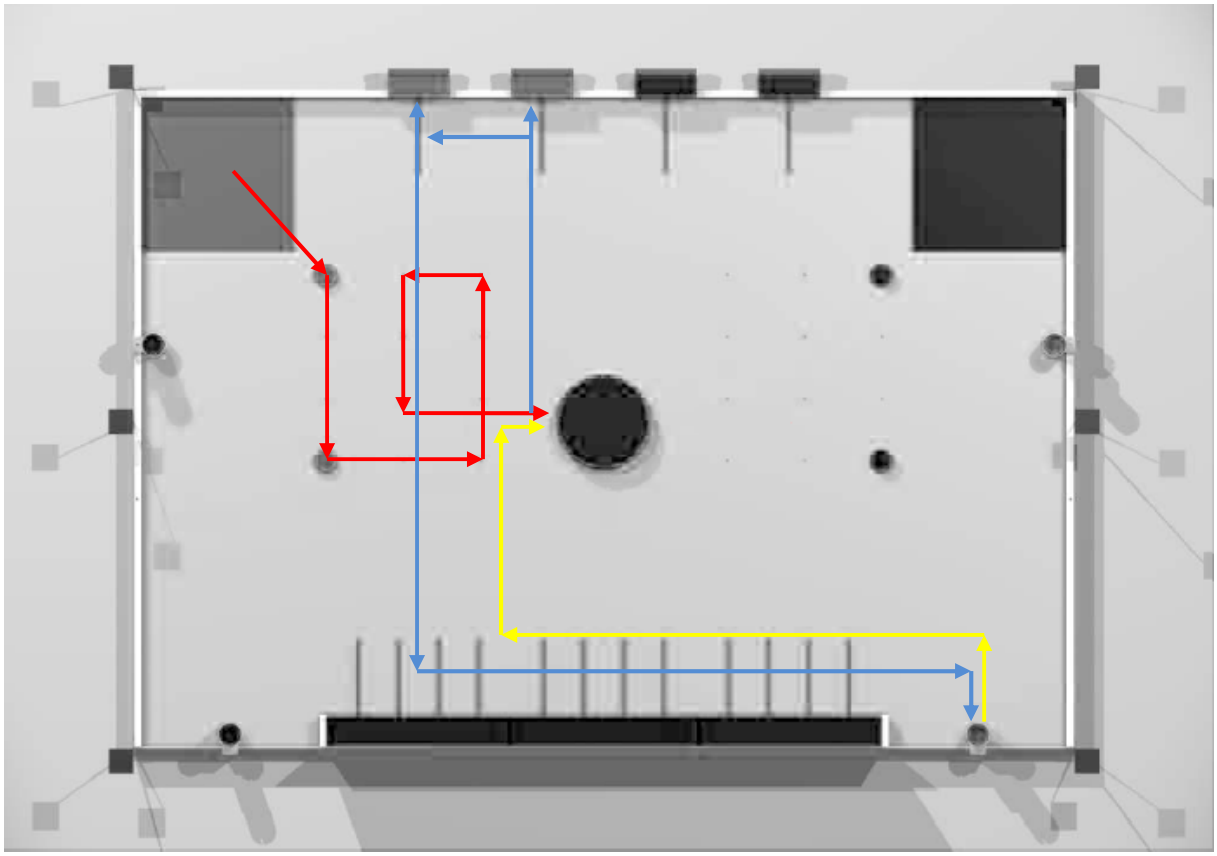
4. Après avoir récupéré les deux linteaux, nous devions aller chercher des palets dans le réservoir fixe. Pour faire ces déplacements nous avons étudié le comportement des robots lors de la coupe de Belgique, ceci nous a permis de voir que la probabilité qu'un robot adversaire soit présent sur notre trajectoire été faible. De plus la précision requise pour le faire n'était pas nécessaire, ce qui nous à conduit à choisir une vitesse assez élevée afin de limiter la rencontre avec l'adversaire.

5. Dernière phase de notre stratégie, la dépose du second temple. Nous devions alors revenir vers la colline, au X, Y et THETA enregistré lors du calage du premier temple. La dépose de ce second temple demandait d'être très précis afin de ne pas démolir notre premier temple ou de tout déposer à côté. Nous faisions donc des déplacements à vitesse normale, avec un recalage après le calage réservoir de sorte à nous garantir une position en X, Y et un angle THETA correct.

Nous obtenions alors la construction finale suivante :



Schématiquement notre stratégie est la suivante :

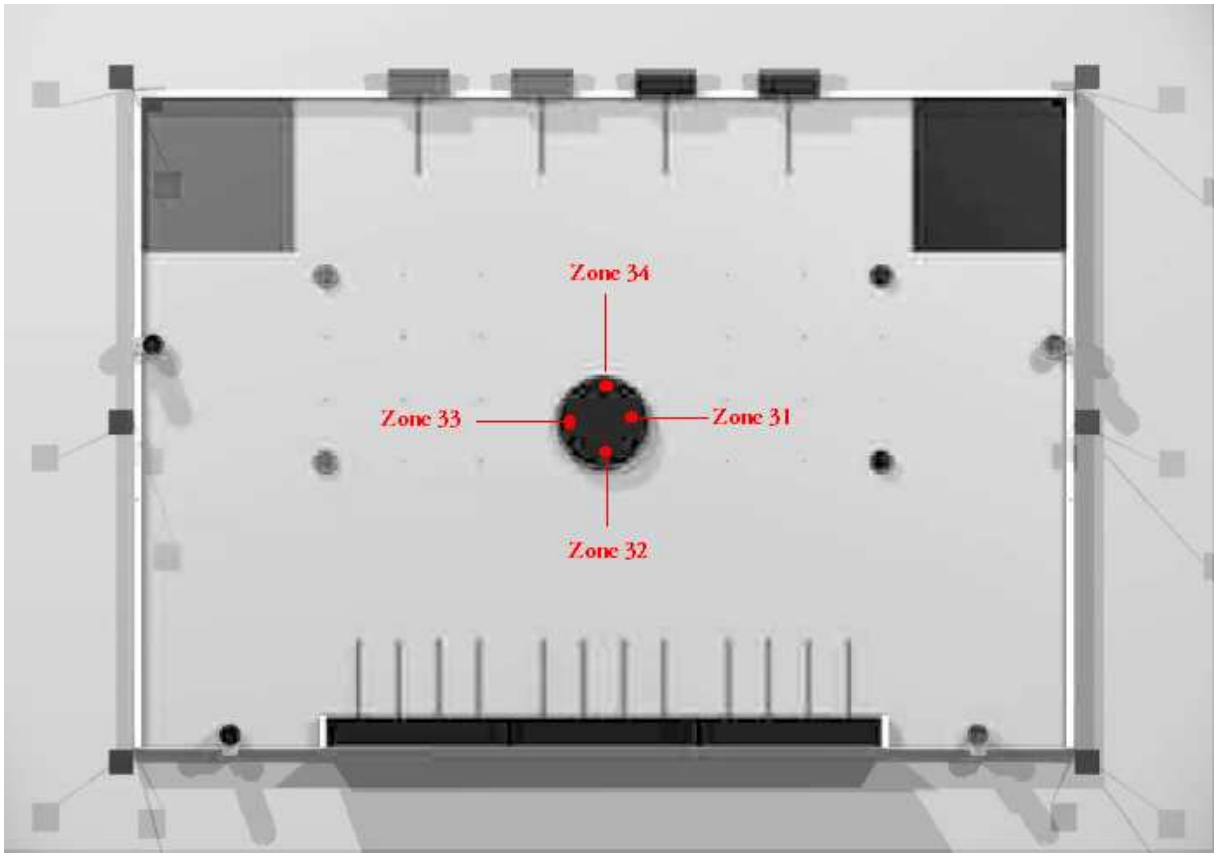


Remarque :

- *En rouge la phase 1, 2.*
- *En bleu la phase 3, 4.*
- *En jaune la phase 5.*

- **Stratégie à partir des 16ième de finale**

Notre stratégie principale à partir des 16ième de finale reprenait notre stratégie d'avant 16ième, avec un changement pour ne pas avoir un comportement prévisible par l'adversaire.
Nous avons choisi de changer notre position de dépose de nos temples, selon le schéma suivant :



Le choix de la zone devait être fait avant le match selon le comportement de l'adversaire durant ses précédents matchs.

3.4. Caméra

3.4.1. Description du module

La caméra utilisée est une Mobisense. Il s'agit d'un module très polyvalent destiné à des applications robotiques. La puissance de calcul de la carte est contenue sur le module Colibri plogué sur la carte qui contient un processeur Intel PAX270. L'ensemble des tâches sont gérées grâce à un OS Linux.

L'algorithme implémenté permet de détecter les palets sur le terrain et de les différencier suivant leur couleur. La structure du programme a été pensée pour que les fonctions principales puissent être réutilisées d'une année sur l'autre.

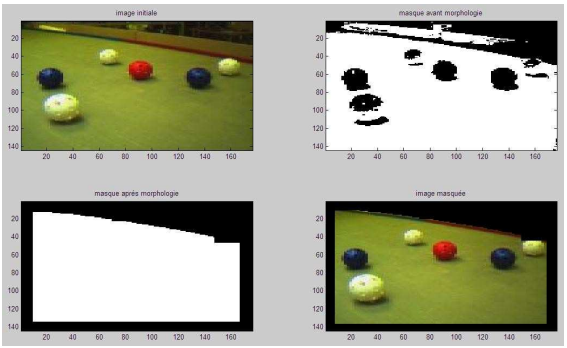
Système embarqué Mobisense

3.4.2. Description de l'algorithme

L'ensemble de l'algorithme a été validé grâce à Matlab et Simulink et provient en grande partie d'un projet de deuxième année réalisé en 2008 s'intitulant « tracking de balles à l'aide d'une caméra numérique », encadré par Mme Elodie Roullot. Le code exécutable à été implémenté en langage C.

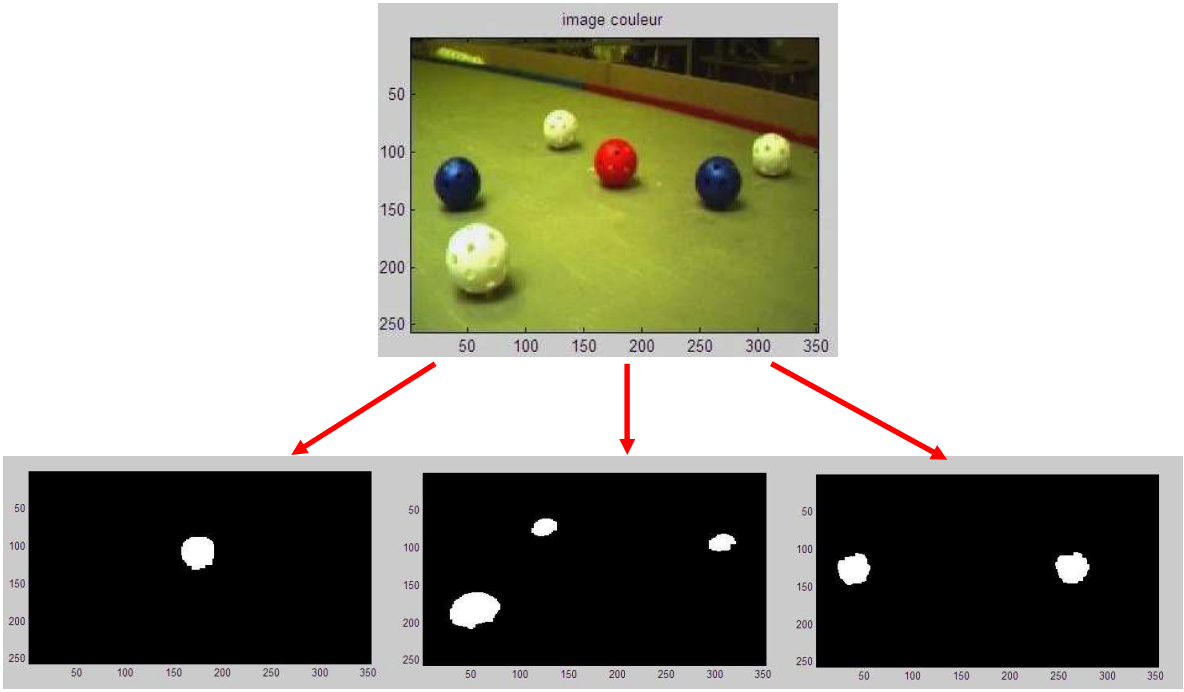
L'algorithme est décomposé en plusieurs parties :

- ***Détection de l'espace de jeu.*** Cette opération est faite en détectant la couleur du terrain par seuillage. Ceci permet de se concentrer seulement sur la zone qui nous intéresse et d'éliminer tout le reste. Des opérations de morphologie mathématique permettent d'améliorer le résultat obtenu.



Etapes successives du masquage du terrain de 2008

- Ensuite, on ***<u>détecte les palets</u>*** suivant leur couleur, en appliquant le même procédé que décrit précédemment (seuillage et morphologie mathématique).



Détection des balles de 2008 de différentes couleurs dans l'ordre: rouges, blanches et bleues.

- Puis, on applique un algorithme d'étiquetage, qui affecte un numéro à chacun des palets détectés, ce qui permet de les différencier. On peut ainsi calculer la position des palets dans le plan image.

- Enfin, une fois les palets détectés on peut calculer la distance de ceux-ci par rapport au robot. La précision des résultats dépend en grande partie de l'algorithme appliqué précédemment et de la qualité du capteur (Micron/Aptina MT9V032).

3.4.3. Perspectives futures

L'ensemble de l'algorithme étant codé sous Matlab, il est facile de pouvoir le réactualiser chaque année en suivant le règlement, ainsi que de l'améliorer. Le but étant de l'améliorer chaque année afin d'adapter dynamiquement la stratégie en fonction de l'environnement. Afin d'aller plus vite dans la phase de développement, une documentation a été réalisée afin de pouvoir programmer rapidement la carte.

De plus, peu d'équipes possèdent un robot avec un système de vision, ce module constitue donc un atout non négligeable pour se démarquer des autres équipes.

En 2008, nous avions commencé le développement de l'algorithme sur une CMUCAM 3 (module peut couteux mais très lent). Nous avons donc opté cette année pour une plateforme plus puissante ne nous limitant pas dans notre utilisation. Ainsi notre puissance de calcul passe de 60 MIPS à 800 MIPS.

3.5. Programme de debug

Afin de nous aider dans la correction de nos fonctions de déplacement, nous avons développé un programme en Visual Basic qui nous affiche différentes informations utiles et qui stocke les données envoyées dans un classeur Excel.

Voici une capture de celui-ci :

Conclusion

Malgré les gros problèmes rencontrés au cours de cette année, nous avons énormément amélioré notre compétitivité et notre niveau s'approche de plus en plus des performances des meilleures équipes. La perte de notre duo PC embarqué et carte µP, qui permettaient de réaliser respectivement les déplacements du robot et l'asservissement vitesse de ce dernier, nous a obligés à mettre en place une nouvelle architecture électronique et à recoder en moins de trois mois une base stratégique ainsi qu'un asservissement en position, ce qui est assez conséquent.

Au niveau mécanique nous continuons de nous améliorer, ceci étant dû entre autre à l'achat d'une fraiseuse qui nous a permis cette année d'accroître notre précision dans les pièces usinées tout en continuant l'utilisation de machines à commandes numériques pour fabriquer le squelette du robot.

Notre robot est désormais équipé, à l'exception de la caméra Mobisense, uniquement de cartes conçues par les membres de l'association, ce qui nous permettra dans le futur de nous adapter avec facilité au règlement de l'année suivante et d'être ainsi très réactif au niveau des modifications électroniques.

Nos problèmes se localisent actuellement au niveau informatique, où une correction du nouveau code de la DsPIC Stratégie doit être effectuée afin de pouvoir bénéficier de déplacements plus précis, permettant aussi l'implantation d'un véritable évitement adversaire.