V 0.5
TX coupeM6 A06

Pinkasfeld Joseph
Morvan Sébastien
Paul Morelle

Rapport de TX
Projet de Robotique

Objectifs :

- Remplacement d'un composant d'asservissement par une solution bon marché et personnalisable.
- Mise en place d'un asservissement polaire.
- Développement d'un simulateur physique en 3 dimensions.

Motivations

Ces 3 objectifs découlent du bilan des résultats obtenus lors de la coupe de France de robotique 2005 et 2006 sur les robots présentés par l'équipe de l'UTC. En 2005 une PR avait été lancée suite au résultat décevant obtenu en 2004 et aboutissait à une base mobile réutilisable et fiable. L'avancée majeure résidait dans l'utilisation de codeurs sur les moteurs de propulsion permettant de connaître a priori la position des roues et donc du robot. Pour gérer ces codeurs et donc permettre l'asservissement des moteurs, l'équipe de l'époque choisit d'utiliser un composant tout intégré du commerce : le LM629. Celui-ci est reconnu dans le domaine des bricoleurs de robot comme étant fiable mais aussi comme pouvant être complexe à mettre en œuvre. Celui-ci permet de réaliser un asservissement PID sur un moteur équipé d'un codeur.

En 2005, (le bowling) la stratégie visait à gagner en restant à distance, le robot se déplaçait assez peu. Le LM629 réagissait comme prévu et permettait de réaliser des déplacements précis (rotation, avancée en ligne droite).

En 2006 (le golf) la stratégie impliquait beaucoup de déplacements, il fut difficile de reproduire des trajectoires préprogrammées avec précision. Notre robot avait tendance à rentrer dans les obstacles et les robots adverses, et l'asservissement se coupait en cas de choc, sans que l'on sache pourquoi.

Le problème ne fut pas identifié précisément mais on suppose qu'il s'agissait d'un problème au niveau des LM629 (faux contact ou mauvaise configuration des registres). De plus une semaine avant de partir sur le lieu de la compétition un mauvais branchement du composant les détruit, nous coûtât 150 € et nous causa beaucoup de stress lié à la peur de ne pas recevoir les composants de remplacement à temps. La fonction réalisée par ce composant si cher n'est pas si complexe et on trouve dans le commerce des microcontrôleurs dédiés à être programmés pour ce type d'application pour seulement quelques dollars.

De plus une impression globale de lenteur et de mollesse se dégageait du robot. Cela venait du fait que notre robot se déplaçait exclusivement en ligne droite, réalisant ses virages sur place. Chaque mouvement nécessitant une phase d'accélération et de décélération, le simple fait de changer de direction peut prendre 10 secondes, quand on sait qu'un match dure 90 secondes et qu'un certain nombre de virages est nécessaire, on comprend que beaucoup de temps est perdu. La solution évidente est de ne pas ralentir lors d'un virage.

Ces 2 points nous poussèrent à repenser la façon dont les déplacements étaient gérés sur nos robots.

1. Rappel sur les principes d'un asservissement

a. Définition

Le but d'un asservissement est de placer un système dans un état donné appelé consigne d'état et de l'y faire rester quelque soit « l'influence du monde extérieur » qui pourrait perturber l'état de notre système.

b. Terminologie

4 termes sont importants dans notre étude :

i. L'état consigne : état dans lequel on souhaite que le système arrive et demeure.
ii. L'état courant : état où est le système à l'instant « présent ».
iii. L'erreur : différence entre l'état courant et l'état consigne.
iv. La commande : énergie que l'on décide de donner au système à un instant donné pour qu'il rejoigne l'état consigne.
v. Le correcteur : système que nous réalisons. Il va délivrer la commande au système en fonction de l'erreur et de son évolution dans le temps.

c. Exemple d'asservissement : le PID (Proportionnel Intégral Dérivé)

La commande issue d'un asservissement PID est calculée à partir de l'erreur, bien sûr, mais sous 3 formes différentes. La commande finale est une somme de ces 3 formes.
Prenons le cas d'un asservissement discret comme c'est le cas pour la TX.

1er terme : P*Erreur, avec P une constante appelé coefficient proportionnel. Plus l'erreur sera grande plus la commande sera grande. L'inconvénient de cette commande est qu'elle est généralement instable si on cherche un asservissement rapide, et peu précise.

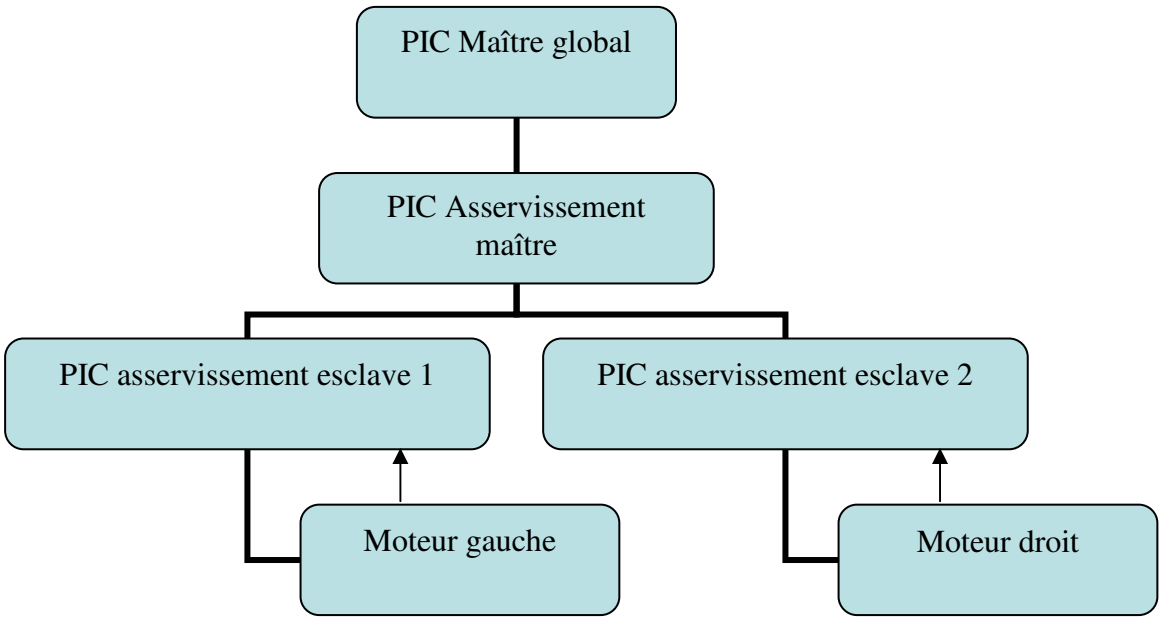
2ème terme : I*(somme des erreurs), avec I une constante appelée coefficient intégral. Si le système ne corrige pas l'erreur assez vite alors la somme des erreurs va augmenter au cours du temps forçant le système à aller plus vite vers sa consigne. De plus il est à noter que si le système est bloqué, même si l'erreur reste constante, alors la commande augmente fortement et peut devenir dangereuse pour l'utilisateur ou les systèmes. Cependant cette commande possède l'avantage d'annuler l'erreur statique.

3ème terme : D*(erreur – erreur précédente), avec D une constante appelée coefficient dérivé. Si l'erreur diminue au cours du temps, on réduit la commande. Cela permet de rendre le système plus stable quand on s'approche de la consigne (réduire le nombre d'oscillations).

Les asservissements de type PID sont très utilisés. Nous en avons d'ailleurs implémenté deux dans notre système : l'un au niveau du contrôle de la vitesse/positon des moteurs, l'autre vis-à-vis de la position/orientation du robot. Nous avons abouti à ce que certains appellent un asservissement Polaire.

2. L'architecture électronique choisie :

Après réflexion nous avons décidé d'adopter une architecture proche des années précédentes. Nous avons juste ajouté une «couche » entre le maître global et les composants dédiés à un moteur. Le PIC asservissement maître s'occupe de coordonner les deux PIC esclaves pour réaliser un asservissement polaire qui sera détaillé plus loin.



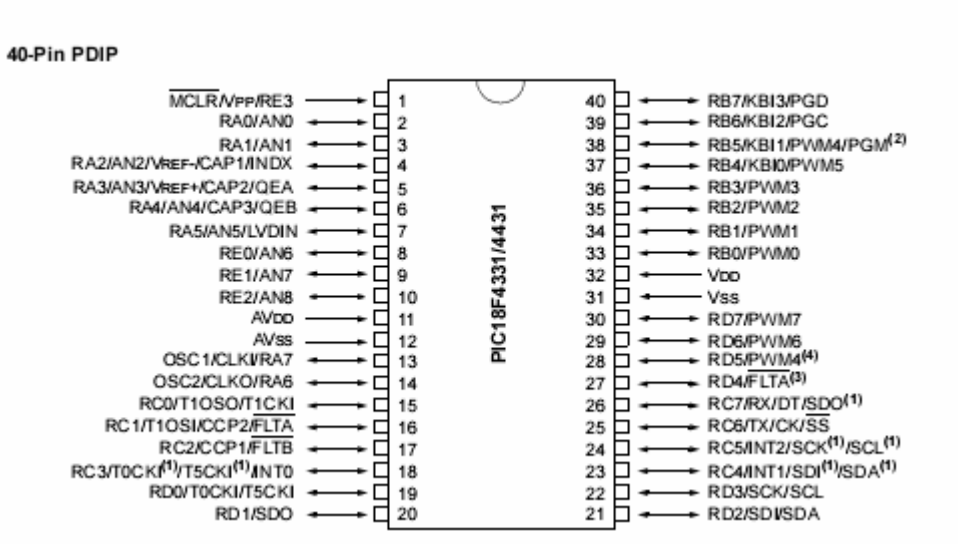
- Le PIC Maître global est relié au PIC asservissement maître par une liaison i2c.
- Le PIC asservissement maître est relié à ses esclaves par un bus parallèle sur 8 bits comme pour les LM629.
- Les PIC asservissement esclaves sont reliés au moteur par une sortie PWM et les moteurs remontent leurs déplacements par les sorties en quadrature de phase des codeurs.

3. L'asservissement moteur

a. Les composants

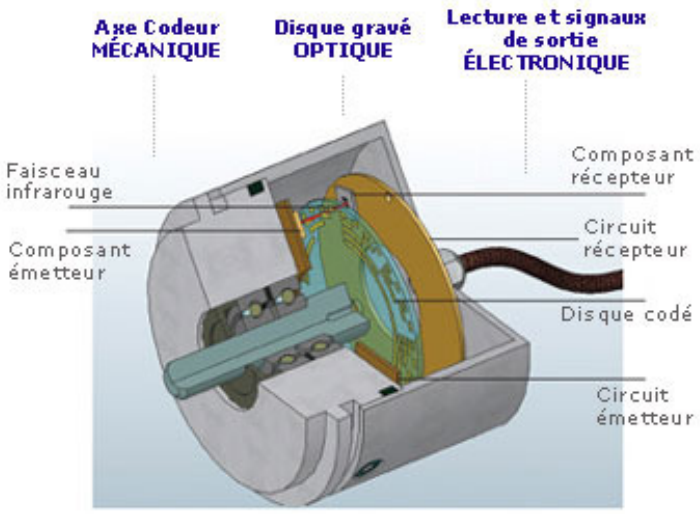
Cette partie de la TX consiste principalement à remplacer les LM629 utilisés précédemment. Pour ce faire nous avons utilisés des microcontrôleurs Microchip choisis pour leurs coûts et leur facilité de mise en oeuvre.

Un microcontrôleur PIC Microchip

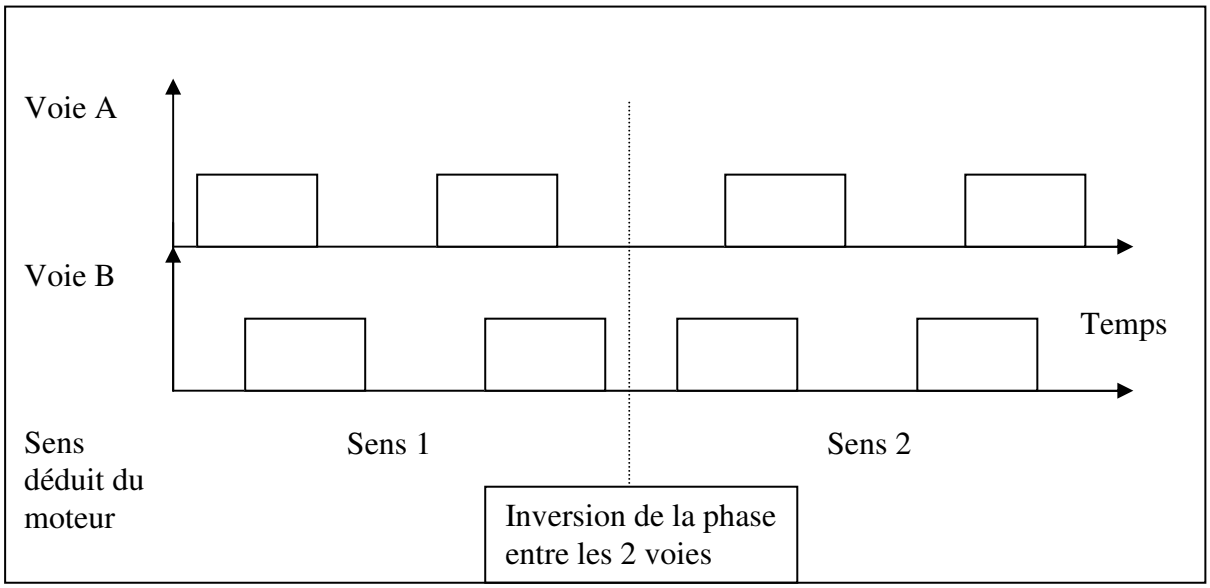


Détail du brochage du PIC esclave.

Le composant choisi porte comme référence PIC18F4431 offrant 16Ko de mémoire flash pour contenir le programme et 768 octets de mémoire RAM pour travailler sur les variables. Il comporte plusieurs sorties PWM permettant, une fois reliées à une interface de puissance, de contrôler la vitesse d'un moteur en boucle ouverte. Nous utilisons la sortie PWM associée à la patte 17, ainsi qu'une interface pour des encodeurs en quadrature, aussi appelés codeurs, raison pour laquelle nous avons choisi ce PIC. En effet, pour connaître la position d'un moteur on peut utiliser des roues codeuses.



Ces roues possèdent des fentes disposées sur tout le tour. Deux capteurs infrarouges, A et B légèrement décalés l'un par rapport à l'autre détectent le passage de ces fentes. L'ordre d'activation des capteurs infrarouges permet de connaître le sens de rotation des roues et la fréquence d'activation permet de calculer la vitesse (le nombre de passage de fentes devant un des deux capteurs par unité de temps).



Notre microcontrôleur possède les entrées nécessaires pour le connecter directement aux codeurs (voir brochage plus haut: patte 4 index, patte 5 A, patte 6 B). Un fois certains registres de contrôle configurés il n'y a plus qu'à lire à un emplacement précis de la mémoire pour récupérer la distance parcourue par le moteur. (Registre CAP2BUFH et CAP2BUFL)

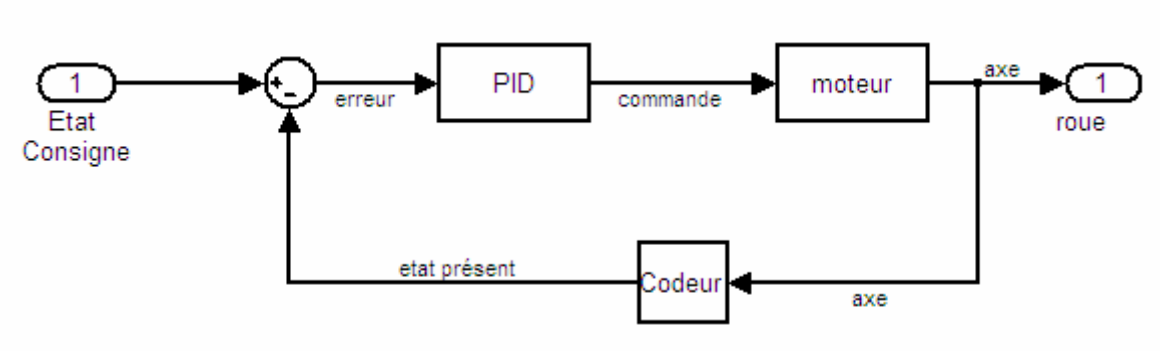


Schéma de principe de l'asservissement moteur.

b. L'implémentation

Nous avons utilisé le compilateur C de CCS, PCWH qui permet grâce à de nombreuses fonctions prédéfinies d'accélérer la configuration des PIC pour mettre en place les composants interne comme la liaison série, les sorties PWM ou les interruptions.

Exemple :
La simple ligne
#use rs232(baud=115200,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)
Configure touts les registres nécessaires à la liaison série pour fonctionner à 115200 bauds et toute les instructions du type printf ou scanf utiliserons la liaison série.

Du fait de devoir implanter des algorithmes temps réel nous avons décidé d'utiliser des variables à portée globale dans le projet. Il n'y a pas besoin de les passer en argument et on économise le temps passé à les copier à chaque appel de sous fonction.

i. Variables

Nous avons notamment utilisé les variables globales suivantes :

- *Accélération* : l'accélération de la vitesse des roues en pas de codeur par itération par itération (p/i/i).
- *Coef_p* , *Coef_i* , *Coef_d* : les coefficient du PID.
- *Position_courante* : contenant la position du moteur en pas de codeur.
- *Position_précédente* : la position du moteur à l'itération précédente.
- *Position_theorique* : la position où devrait se trouver le moteur si l'asservissement fonctionne.
- *Position_consigne* : la position que doit atteindre au final le moteur.
- *Vitesse_consigne* : la vitesse que doit atteindre au final le moteur en pas de codeur par itération (p/i).
- *Vitesse_theorique* : la vitesse que devrait avoir le moteur à l'instant présent si l'asservissement fonctionne parfaitement.
- *Vitesse_courante* : la vitesse actuelle du moteur en pas de codeur par itération.
- *Vitesse_max* : vitesse maximum qu'on autorise pour le moteur.
- *Erreur_position* : la différence entre la position théorique et la position courante.
- *Erreur_somme* : la somme des erreurs de position (pour le PID).
- *Erreur_precedente* : l'erreur de position de l'itération précédente (pour le PID)
- *Erreur_max* : l'erreur maximale autorisée, au delà on considère que le système a un problème.

ii. Fonctions

Fonction de base du système d'asservissement esclave (tiré du polycopié A06 de l'UV MC06 enseigné par MR Christophe FORGEZ) :

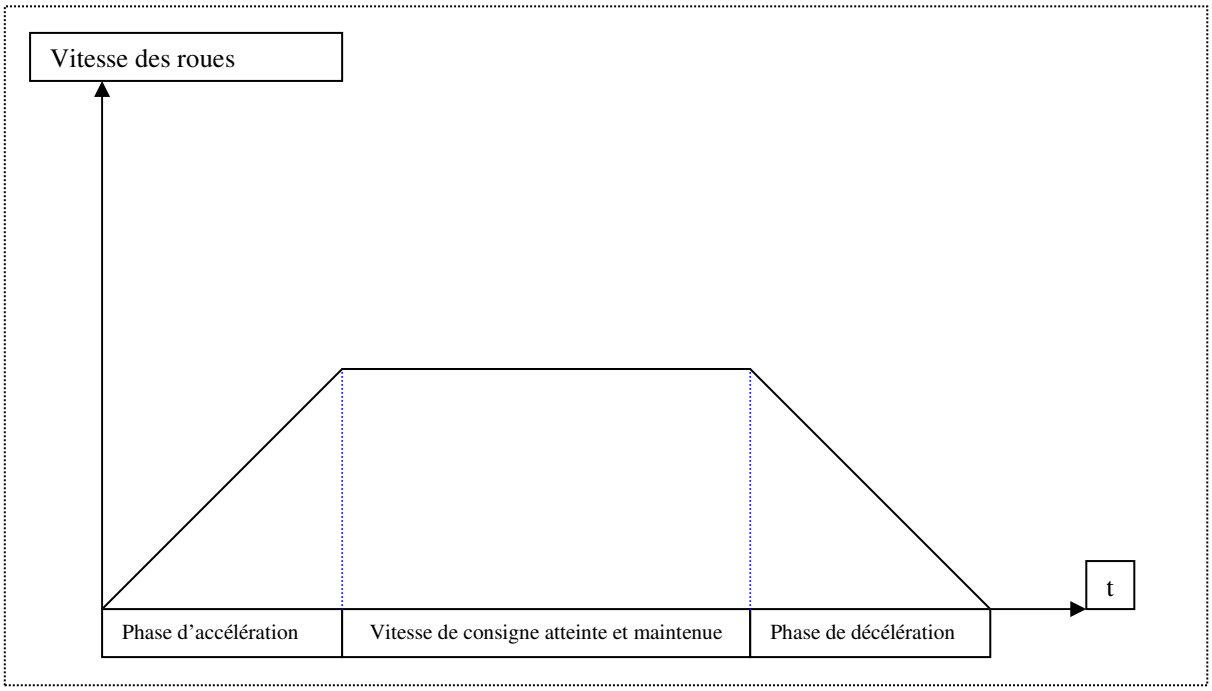
Principe : calculer le rapport cyclique (duty) du PWM (Pulse Width Modulation) appliqué au moteur en fonction du PID de l'erreur de position du moteur.

```
void calcul_pid_consigne_moteur(void)
{
	//application du PID
	duty = erreur_position * coef_p/100
		+ erreur_somme*coef_i/10000
		+ (erreur_position-erreur_precedente)* coef_d/1000 ;

	duty=abs(duty);
	// on borne la valeur pour rester dans les limites mecaniques
	duty = ecretage(duty,1000,0);
	// on calcule les termes nécessaires au PID pour la prochaine itération
	erreur_somme+=erreur_position;
	erreur_precedente=erreur_position;
	// on applique le nouveau rapport cyclique à la sortie PWM 1 du PIC
	set_pwm1_duty(duty);
	// on fait tourner le moteur dans un sens ou dans l'autre en fonction
	// du signe de l'erreur
	if (erreur_position < 0) output_low(patte_sens);
	else if (erreur_position > 0) output_high(patte_sens);
}
```

Commentaire : Cette fonction est appelée à intervalles réguliers au moyen d'une interruption toutes les 10ms environ.

Pour éviter les accélérations trop brutales qui abîmeraient la mécanique et pourraient causer des dérapages, nous devions contrôler la variation de la vitesse des roues. C'est pour cela que nous avons des paramètres de consigne et des paramètres théoriques. Nous avons choisi un asservissement de la vitesse des roues en trapèze. De cette façon l'accélération est constante et le robot ne subit pas d'à-coups.



La fonction suivante permet de réaliser ce trapèze si elle est exécutée à intervalles constants. Celle-ci est suffisamment explicite pour se passer de commentaire.

```
void fonction_asservissement_vitesse(void)
{
	if ( vitesse_theorique < vitesse_consigne ) vitesse_theorique+=acceleration;
	if ( vitesse_theorique > vitesse_consigne ) vitesse_theorique-=acceleration;
}
```

A l'aide des deux fonctions précédentes on peut faire tourner un moteur à une vitesse donnée de façon précise, robuste, rapide et sans à-coups. Mais la vitesse n'est pas tout si on souhaite atteindre une position particulière.
La fonction suivante permet de faire varier la vitesse de consigne pour que le moteur se place à une position donnée : la position de consigne. Le problème est le suivant : si on souhaite que la vitesse soit nulle au moment précis où le moteur arrive à la position de consigne en suivant les trapèzes de vitesse vu plus haut, il est alors nécessaire de savoir quand entrer dans la phase de décélération. Pour cela on va calculer à chaque itération *le temps de freinage* avec la formule suivante

$$t_f = \frac{vitesse\_théorique}{accéleration}$$

(attention t n'est pas en seconde mais en nombre d'itérations, notre unité de temps).
On peut alors le comparer au temps restant

$$t_r = \frac{d}{vitesse\_théorique}$$

avec d la distance restant à parcourir avant d'atteindre la position de consigne.
Donc si $t_f \geq t_r$ il est temps de rentrer en phase de décélération, cela se fait en plaçant la vitesse de consigne à 0, la fonction d'asservissement de la vitesse s'occupera alors de réduire la vitesse en suivant le trapèze. Sinon on donne comme vitesse de consigne la vitesse maximale.

```
void fonction_asservissement_position(void)
{
	distance_restante = position_consigne - position_courante;
	if (abs(distance_restante) < 1000) flag_asservissement_position_fini=1;
	if (!flag_asservissement_position_fini)
	{
		if (distance_restante > 0) vitesse_consigne=vitesse_max;
		if (distance_restante < 0) vitesse_consigne=-vitesse_max;
		if (	abs(vitesse_theorique/(acceleration))	>
			abs(distance_restante/vitesse_theorique)	)
			vitesse_consigne=0;
	}
	else  vitesse_consigne=0;
}
```

On peut à présent appeler toutes ces fonctions dans la boucle principale du programme d'asservissement.

```
WHILE(1) // début de la boucle principale
{
	// Mise à jour des variables de consigne ou du mode de fonctionnement
	//	si de nouvelles valeurs sont reçues du maître
	 maj_buf_to_var();

	// si le temps d'échantillonage (10ms) est écoulé
	//	on procède à une itération de l'asservissement
	if (Te_atteint)
	{
		// On fait clignoter une LED
		//	pour indiquer que le système est en cours d'exécution.
		heart_beat();

		// Mise à jour des variables de position et de vitesse

		// On lit dans la mémoire la position du moteur
		lecture_position_courante();
		vitesse_courante=position_courante-position_precedente;
		position_precedente=position_courante;
		// Si le système est en mode asservissement en position
		//	la fonction précédente est appelée
		if (asservissement_position)   fonction_asservissement_position();
		// On peut choisir de désactiver l'asservissement en vitesse
		if (asservissement_vitesse)   fonction_asservissement_vitesse();
		// Calcul de la position théorique et de l'erreur de position
		position_theorique+=vitesse_theorique;
		erreur_position=position_theorique-position_courante;
		// Si l'asservissement est en marche
		//	on calcule la nouvelle consigne et on l'applique
		 if (asservissement_on)  calcul_pi_consigne_moteur();

		// L'asservissement pour cette période d'échantillonnage a été réalisée
		//	on repasse Te_atteint à 0
		// L'interruption sur le Timer repassera Te_atteint à 1 dans 10 ms
		//	– le temps qui a été nécessaire à l'exécution de cette boucle
		Te_atteint=0;
	}
}
```

4. Le protocole de communication

Le microcontrôleur maître doit être relié à ses deux esclaves par un canal de communication. Ce canal doit être rapide pour que le transfert de données d'un coté à l'autre se fasse sans gêner l'asservissement à adressage multiple (nous avons deux esclaves). Nous avons envisagé d'utiliser le bus i2c, celui-ci étant très simple et déjà présent sur le robot, intégré aux microcontrôleurs. Cependant nous avons préféré réaliser notre propre bus de communication pour deux raisons :
- La vitesse et l'encombrement du bus. En effet le bus i2c est un bus série, un bus parallèle permet d'aller beaucoup plus vite.
- Beaucoup de communications étaient prévues entre le maître et les esclaves, or en i2c on ne trouve qu'un seul maître, cette fonction est déjà réservée aux maître global. Pour transmettre une donnée entre le maître et les esclaves de l'asservissement il aurait fallut à chaque fois passer par le maître global, l'interrompre quelque chose de peu important pour lui.

a. Le bus

Notre bus est le suivant :

Les entrées/sorties de données :
Sur le maître et les esclaves.

- 8 fils d'entrées/ sorties câblés sur le port B du maître et des 2 esclaves

Les entrées/sorties de contrôle :
Sur les esclaves :

- 1 entré CS : Chip Select : pour indiquer à l'esclave de se mettre en écoute car la communication lui est destinée, ou de lui signaler d'envoyer la donnée que le maître lui a demandé. Câblé sur l'entrée d'interruption externe.
- 1 entrée R/W Read Write : indique si la transaction en cours est une écriture ou une lecture de données.
- 1 entré C/D Command Data : indique si la transaction en cours est une commande ou une donnée.

Sur le maître :

- 2 sorties de sélection câblées aux entrées CS des esclaves. Permet de choisir à quel esclave on s'adresse.
- 1 sortie R/W câblée aux 2 esclaves
- 1 sortie C/D câblée aux 2 esclaves

b. Le protocole

Le maître contrôle la transaction. Le protocole est le suivant :

i. Chez le maître :

Ecriture :

- Passer R/W à un l'état haut
- Passer C/D à l'état haut si la transaction est une commande, à l'état bas si c'est une donnée.
- Passer le bus en sortie
- Placer l'octet de donnée sur le bus
- Passer à l'état haut la sortie CS correspondant au PIC auquel s'adresse la transaction.
- Attendre un certain temps, pour permettre à l'esclave qu'il comprenne la transaction et récupère la donnée.
- Passer à l'état bas la sortie CS correspondant au pic auquel s'adresse la transaction.
- Replacer le bus en entrée pour le remettre en haute impédance

Lecture :

- Passer R/W à un l'état bas
- Passer C/D à l'état bas, on ne lit que des données.
- Passer le bus en entrée
- Passer à l'état haut la sortie CS correspondant au PIC auquel s'adresse la transaction.
- Attendre un certain temps, pour permettre à l'esclave qu'il comprenne la transaction qu'il place la donnée demandée sur le bus
- Lire cette donnée.
- Replacer à l'état bas la sortie CS correspondant au pic auquel s'adresse la transaction.
- Replacer le bus en entrée pour le remettre en haute impédance.

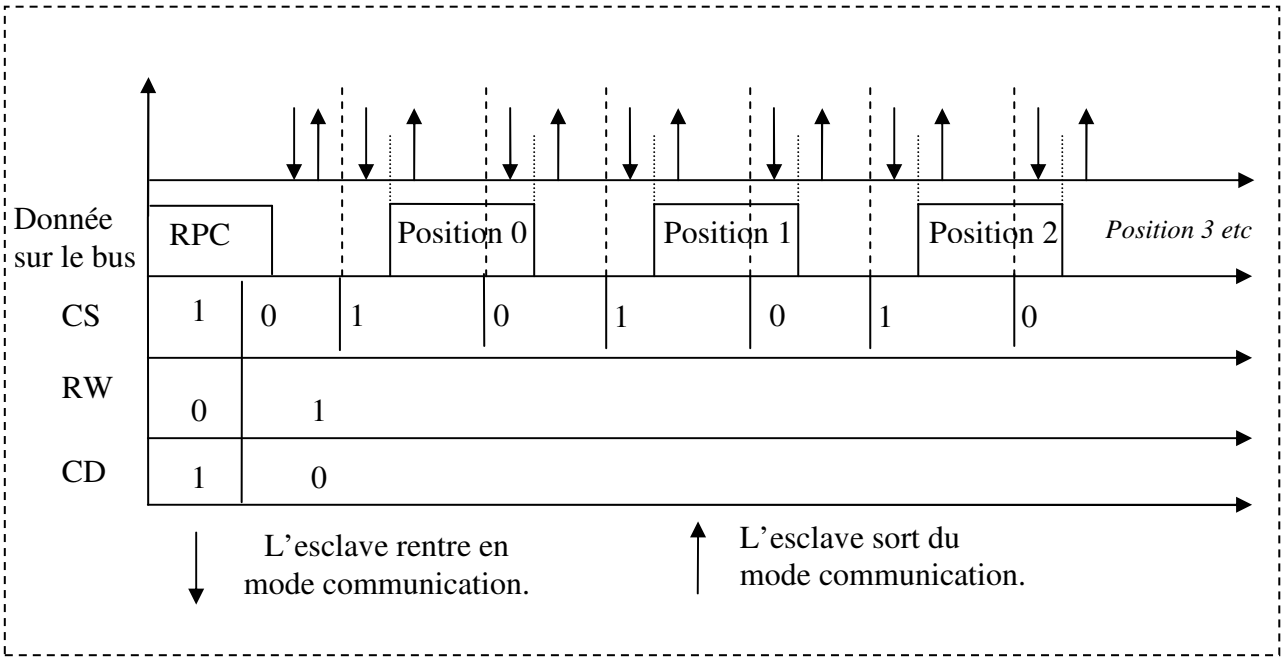
ii. Chez l'esclave

Le fonctionnement est plus complexe car la communication passe au second plan pour l'esclave, ce dernier devant avoir pour priorité l'asservissement. On va donc fractionner les opérations de communication pour que les itérations d'asservissement puissent avoir lieu à la bonne date et que les deux processus ne se bloquent pas l'un l'autre, car tout deux utilisent une interruption. Il y a par ailleurs une mémoire des opérations précédentes à travers diverses variables pour savoir la position dans les cycles de communication impliquant plusieurs lectures/écritures, notamment pour les variables possédant 4 octets à transmettre.

Prenons l'exemple de la lecture de la position de la roue droite :

1. Le maître écrit la commande RPC (Read Position Courante) soit 0X21
2. L'esclave lit cette commande quand le maître l'écrit et l'enregistre comme commande en cours. Par ailleurs il place sont compteur de lecture à 0. Puis se replace en attente de communication ou d'itération.
3. Le maître fait une lecture, l'esclave place l'octet 0 de la position de la roue droite sur le bus, et incrémente son compteur de lecture. Puis se replace en attente de communication ou d'itération.

4. Le maître arrête de lire (CS repasse à 0, voir au dessus), l'esclave libère le bus et se replace en attente de communication ou d'itération.
5. Le maître fait une lecture, l'esclave place l'octet 1 de la position de la roue droite sur le bus, et incrémente son compteur de lecture, puis se replace en attente de communication ou d'itération.
6. Le maître arrête de lire (CS repasse à 0, voir au dessus), l'esclave libère le bus et se replace en attente de communication ou d'itération.
7. Et ainsi de suite jusqu'à l'octet 3 de la position.



Chronogramme de la transaction décrite.

On remarque le décalage entre le moment où CS passe à 1 et celui où les positions deviennent disponibles sur le bus. Cela correspond au temps de traitement pour l'esclave d'une itération dans le cas où la communication intervient juste après le début d'une itération. Quand l'itération a fini d'être traitée, la communication débute. Cet intervalle de temps est de l'ordre de 50 microsecondes et n'est pas négligeable.
On remarque également le temps entre le moment où CS passe à 0 et le moment où la position cesse d'être sur le bus. Cet intervalle de temps est du même ordre que le précédent, et vient de la même cause.

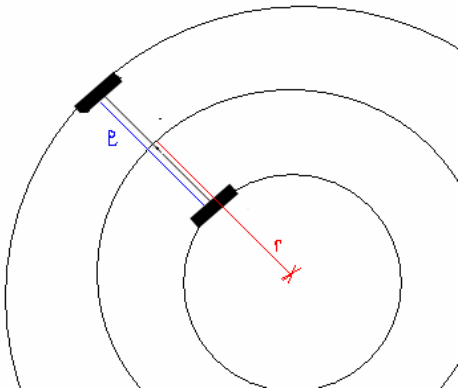
5. L'asservissement polaire.

La principale évolution que nous cherchions à mettre en place au cours de cette TX était de ne plus devoir s'arrêter pour prendre les virages. L'asservissement polaire répondait à ce critère.

a. Modélisation du robot

x,y position du mobile
θ orientation du mobile
v1 vitesse de la roue droite
v2 vitesse de la roue gauche
u1 commande de la roue droite (dérivé de la v1)
u2 commande de la roue gauche (dérivé de v2)
entraxe distance entre les 2 roues
dm distance au bord
X =(y,θ,v1,v2) Le vecteur d'état du robot

Tentons de déterminer les équations régissant le déplacement du robot en fonction du déplacement de ses roues. On simplifie le modèle en réduisant le robot à 2 roues indépendantes possédant chacune un moteur. Prenons le cas du robot avançant avec v1 différent de v2. Le robot va décrire des cercles concentriques:



Logiquement on trouve :

$$v_1 \times t = 2\pi \times (r + \frac{entraxe}{2}) \quad (1)$$

$$v_2 \times t = 2\pi \times (r - \frac{entraxe}{2}) \quad (2)$$

$$v \times t = 2\pi \times (r) \quad (3)$$

Si on somme (1) et (2) il vient

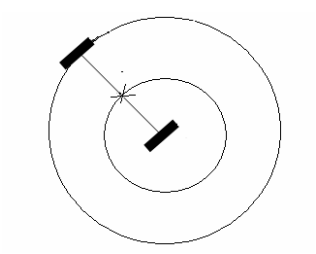
$$(v_1 + v_2) \times t = 2 \times 2\pi \times r \quad (4)$$

Finalement qu'on injecte (3) dans (4) alors :

$$v = \frac{v1 + v2}{2}.$$

L'orientation peut se calculer de la façon suivante :
Si la roue gauche avance et que la droite reste fixe alors **θ** va diminuer. Du coup

$$\theta \times \left(\frac{entraxe}{2} \right) = -d2 \quad (5)$$



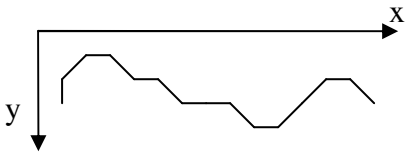
Si c'est la roue droite qui avance alors **θ** va augmenter et

$$\theta \times \left(\frac{entraxe}{2} \right) = d1 \qquad (6)$$

Si on somme 5 et 6 alors

$$\theta = \frac{d1 - d2}{entraxe} \text{ modulo } 2\pi .$$

On peut alors calculer une approximation de la position du robot à tout instant si on discrétise et qu'on approxime la trajectoire curviligne par une succession de segments de droite suffisamment petits.



Trajectoire curviligne discrétisée en segment.

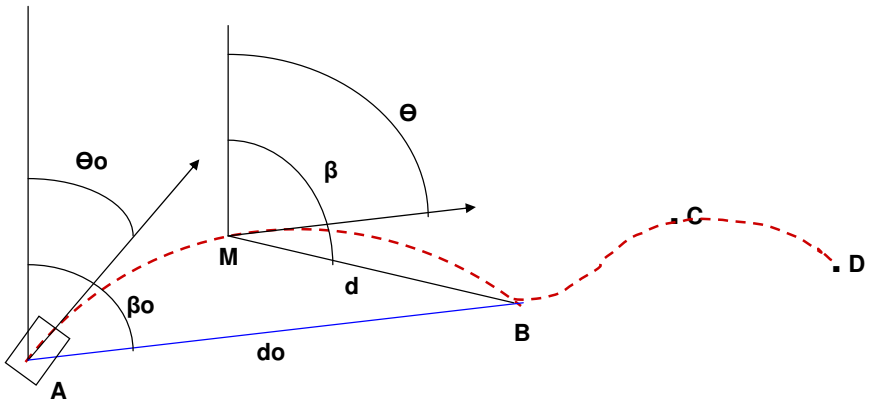
Puis on calcule la position en répétant à intervalles réguliers l'opération suivante :

$$\partial x = V \times \cos(\theta)$$
$$\partial y = V \times \sin(\theta)$$
$$X = \partial x + X$$
$$Y = \partial y + Y$$

On peut donc connaître en temps réel la position de notre robot, ainsi que son orientation.

b. Principe

Soit M la position du robot. On souhaite que le robot actuellement en A passe par B puis C puis D en décrivant des courbes.



On calcule donc de façon régulière l'erreur de distance d, c'est-à-dire la distance entre M et le point ciblé, ici B ; de même que l'erreur d'orientation, soit l'écart entre β, l'orientation idéale pour rejoindre la destination et θ.

c. L'asservissement

A présent que nous avons modélisé notre robot et les erreurs de positionnement nous pouvons appliquer PID hybride. L'idée vient de l'IUT de Ville d'Avray.
On peut appliquer un PID sur l'erreur d'orientation et l'erreur de distance pour contrôler la vitesse de consigne des roues.

Exemple avec juste le coefficient P pour les 2 asservissements.

$$V_d = K_{DP} \times d$$

Puis on écrête ces vitesses par $V_{d\max}$: la vitesse maximal en ligne droite.

$$V_\theta = K_{\theta P} \times (\theta - \beta)$$

Puis on écrête ces vitesses par $V_{\theta\max}$: la vitesse maximale en rotation.
Enfin on somme les 2 consignes calculées précédemment pour chaque roue :

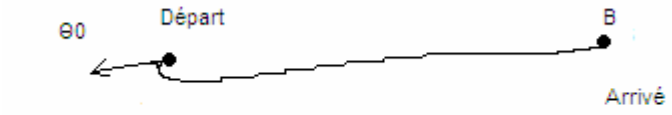
$$V_1 = V_d + V_\theta$$
$$V_2 = V_d - V_\theta$$

Cependant, cette méthode possède un inconvénient. Si le mobile doit suivre un fort changement de trajectoire, alors le rayon de courbure dépend de $V_{d\max}$. Plus $V_{d\max}$ sera grand plus le rayon sera grand.

Prenons l'exemple où le robot possède une orientation initiale complètement opposée à celle nécessaire pour arriver en B. Il fait donc un large détour autour du point de rotation C avec un rayon de courbure r.
Nous nous sommes penchés sur le problème, et la solution retenue pour accélérer la rotation sans toucher à $V_{d\max}$ fut de placer un coefficient sur V_d pour que ce dernier soit annulé quand l'erreur d'orientation est maximale (180 degrés).

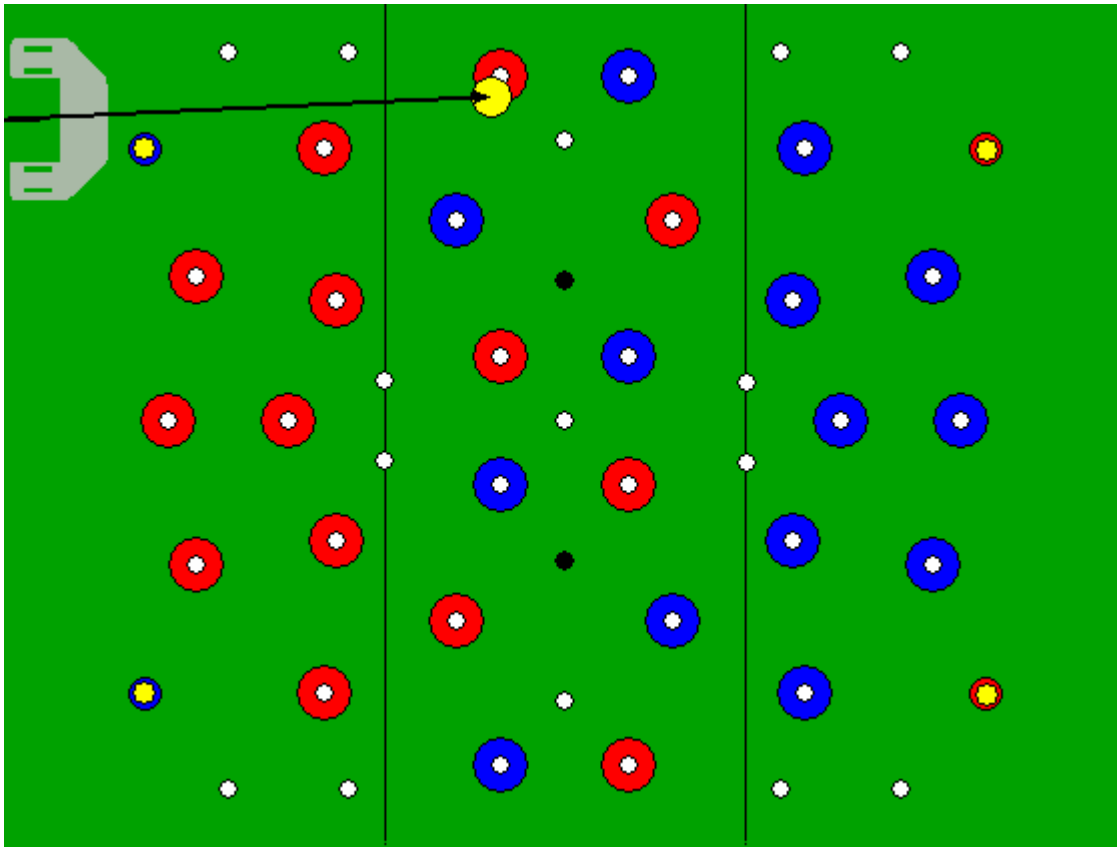
$$V_d = K_{DP} \times d \times \left(1 - \frac{(\theta - \beta)}{180}\right)$$

On obtient le résultat suivant



Le robot commence par tourner sur place et quand l'erreur devient suffisamment faible l'asservissement en distance s'applique normalement.

Une 2ème astuce est de changer la valeur d'erreur d'angle max par une plus petite valeur comme 160. Si l'erreur est supérieure à 160 alors V_d recevra une valeur négative. Le robot ne va plus se contenter de tourner sur place, il va également reculer. Cela est bien pratique si notre robot est contre un mur et doit aller à un point derrière lui.



Exemple avec le simulateur de l'an passé on voit le robot sur le terrain vu du dessus: Le robot est orienté vers un mur et doit suivre la flèche jusqu'au point jaune. Cependant s'il se retourne sans reculer, il risque de se bloquer contre le mur de gauche.

d. Trajectoire

Pour que le robot suive une trajectoire, nous avons mis en place une file dans lesquelles les points de passage sont insérés au fur et à mesure. Quand d devient inférieur à un seuil (2 cm pour nos essais) l'algorithme charge depuis la file le point suivant comme nouvelle destination. Si l'algorithme arrive au bout de la file, le robot s'arrête sur le dernier point de la trajectoire et attend le chargement d'un nouveau point dans la file.

6. Limitations

Pour le moment on ne peut pas donner une consigne de d'orientation pure au robot. On ne peut pas lui dire d'aller se plaquer le dos au mur par exemple. Nous comptons trouver des solutions à ce problème durant l'intersemestre.

Un problème qui a surgit fut le réglage de l'entraxe dans le programme d'asservissement polaire. En effet, le calcul de l'orientation repose sur la connaissance de la taille de l'entraxe. Cependant notre unité de distance à l'intérieur du robot est le pas de codeur, et un millimètre de développé de roue correspond à environ 450 pas. Nous avons donc calculé une distance de 17000 pas environ. Mais l'erreur sur cette valeur devient vite importante à mesure que les moteurs parcourent de la distance.

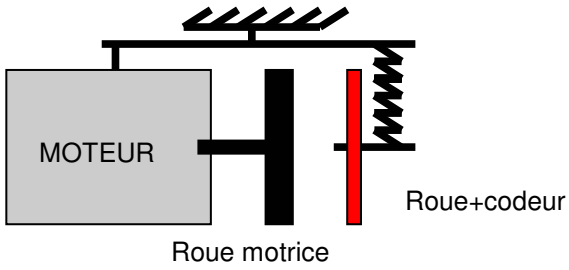
Rappel : $\theta = \frac{d1 - d2}{entraxe} \, modulo \; 2\pi$.

A mesure que la différence $d1 - d2$ grandit, l'orientation dévie de plus en plus. Pour trouver une valeur fiable de l'entraxe. Nous nous sommes servi du simulateur de l'an passé. Le robot envoie sa position et son orientation calculée avec l'entraxe, et grâce à ces mesures on affiche en temps réel un robot sur le terrain virtuel. On peut alors comparer la réalité que nous percevons, en voyant le vrai robot sur le terrain, à celle que le robot calcul. Le protocole était le suivant :

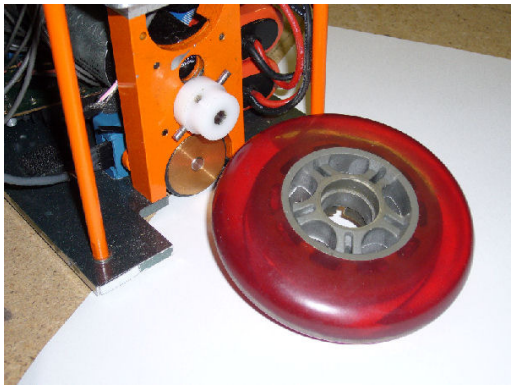
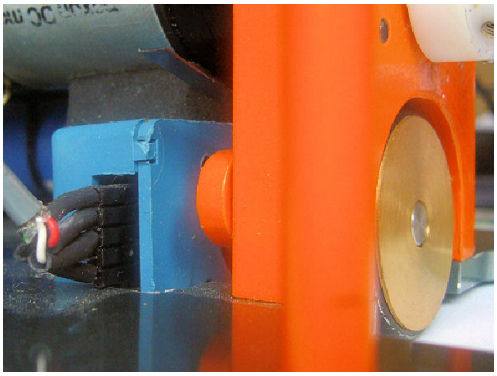
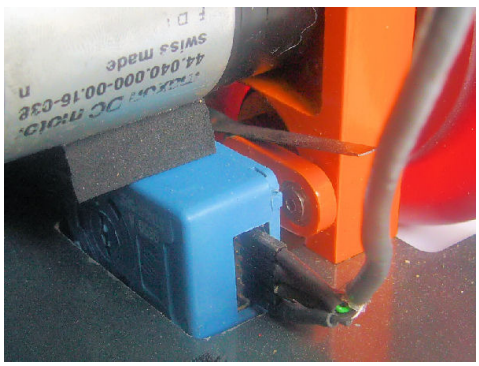
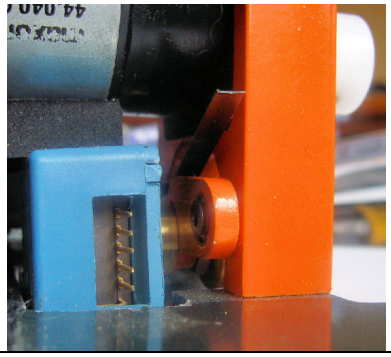
- Faire faire au robot beaucoup de tours dans le même sens pour que $d1 - d2$ grandisse.
- Caler le robot contre un bord.
- Envoyer différentes valeurs d'entraxe au robot jusqu'à ce que l'image du robot virtuel coïncide avec celui sur le terrain.
- Une fois la bonne valeur identifiée, la placer dans le code pour que le robot ait la bonne valeur d'entraxe dès l'allumage.

7. Ce qui reste à faire

Une autre évolution que nous souhaitions ajouter à notre robot est la possibilité de se cogner sans que celui-ci ne se perde si les roues dérapent. Jusqu'à présent nous utilisions des codeurs directement sur les axes moteurs. L'idée est de placer les codeurs sur des axes différents montés sur ressort pour coller au sol même si une des roues du robot décolle un peu. La société Vicatronic nous a justement sponsorisé pour cette partie en nous fournissant deux petits codeurs 500 points à tige sortante.



Exemple tiré du forum de la coupe de France de robotique : mise en place d'un codeur avec un petite roue sous le moteur de propulsion on distingue également la lamelle destinée à plaquer l'ensemble (roue+codeur) au sol.

<table>
<tr>
<td> 
La roue motrice démontée permet de voir que les 2 axes sont bien alignés.</td>
<td> 
Une petite roue permet de gagner en résolution (Pas de codeur/m).</td>
</tr>
<tr>
<td>La lamelle permet de plaquer au sol.
  </td>
<td>On voit le moteur de propulsion juste au dessus.
  </td>
</tr>
</table>

Note : La partie sur le simulateur se trouve dans le 2ème rapport dédié au sujet.

Conclusion

Cette TX s'est révélée fort enrichissante de par l'ensemble des problèmes soulevés et résolus. Nous avons réalisé l'ensemble des objectifs fixés début septembre mais ce n'est pas terminé, le robot n'est pas encore prêt ! Tous nos tests ont été réalisés sur le robot de l'an passé. L'intersemestre permettra d'avancer grandement et de mettre placer le nouveau système de positionnement sur le prototype.