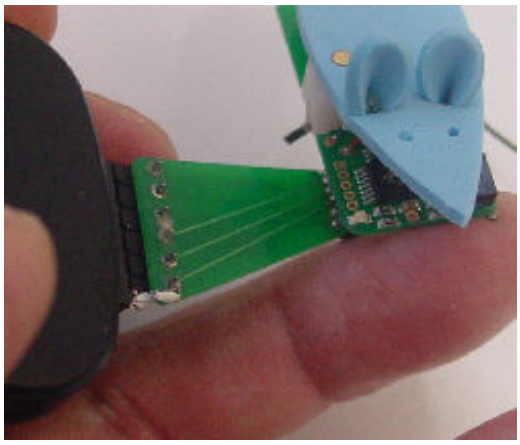
Didel SA, Mouette 5, CH-1092 Belmont
Fax ++41 21 728-6157
info@didel.com www.didel.com

www.didel.com/RhodoDemo.pdf

Rhodo –modifions la demo

Rhodo part dans le mode Demo s'il ne détecte pas les signaux d'un transmetteur infrarouge compatible (Emir2/Bimo, Emir4). La démo est prévue pour que le Rhodo reste dans un espace restreint et aie des pauses pour économiser l'accu. Avez-vous remarqué que le clignotement donne une indication du temps d'attente avant redémarrage ?

Le programme Demo est facile à modifier, mais il faut un programmateur de PIC, un Pickit2 et l'adaptateur Pk2, disponibles pour 70.- Ceci permet de programmer tous les microcontrôleurs PIC de Microchip, donc les Cœurs, Orion, Baton lumineux, Ub4 et autres produits de Didel qui sont toujours des portes ouvertes vers la créativité.



Le langage qui décrit la trajectoire est basé sur des macros d'assembleur et le langage est simple : on donne la vitesse des deux moteurs de la souris et on dit pour combien de temps il faut l'appliquer.
Il n'y a que 3 vitesses par moteur, trois en avant, nommées **a1 a2 a3** et trois en arrière, **r1 r2 r3**.
L'ordre **Bouge** a 3 paramètres : la durée avec seulement 2 possibilités codées **1 2** qui correspondent à 0.3s et 0.7s environ, la vitesse du moteur gauche, puis celle du droite.

Bouge 2, a2, a2 fait avancer en ligne droite, à mi-vitesse, pendant 0.7 secondes
Bouge 1, a2, r2 fait tourner la souris sur elle-même à droite.

De quel angle ? Difficile à prévoir, la souris est capricieuse : quel est le sol, la charge de l'accu, l'encrassement de l'axe, la position des pneus ?
Pour un arrêt bref, qui permet à la souris de finir par inertie son mouvement précédent, on a l'appel **Stop**, avec un seul paramètre, la durée **1,2** de 0.3, 0.7 s.

Stop 2 moteurs arrêté 0.7 seconde pendant que Rhodo finit par exemple de tourner sur lui-même par inertie.

Pour que la souris se repose (et économise son accu), on écrit **Pause** avec comme paramètre la durée en secondes (max 63 secondes).

Pause 25 plus de mouvement pendant 25 secondes**.**

Dans ces ordres, la durée zéro ne doit pas être utilisée. Le zéro est interprété pour terminer et recommencer la séquence.

Recommence (pas besoin d'expliquer !)

Il y a encore l'ordre **Fin** si vous ne voulez pas que cela recommence

Encore une astuce : Le moteurs ne sont jamais à fond, ils tournent trop vite et la ligne droite réagit mal à fond. Mais en toupie, le Rhodo est impressionnant. Pour tourner dans un sens ou dans l'autre, il faut écrire
Bouge 1,0,Spin ou Bouge 1,Spin,0 (le 1 est la durée, on peut mettre 2)

La séquence des mouvements est dans un fichier **MtDemo.asi** qui est appelé dans le programme **Mtl09.asm** ; on verra les détails plus loin.
Il faut ignorer le début et la fin en jaune, et on voit la séquence qui existe dans tous le Rhodos. La touche Tab est utilisée pour encolonner.

```
\prog;MtDemo.asi

Trajet:
        Move    PtTraj,W
        Inc     PtTraj
        Add     W,PCL

        Bouge  2, a1,a1
        Bouge  1, a1,a2
        Bouge  1, 0, 0
        Bouge  2,a1,a2
        Stop   2

        Bouge  2,r1,a1
        Bouge  1,0,0
        Bouge  1, a1, a1
        Bouge  2,r1,r1
        Bouge  1,a2,a1
        Bouge  2,a1,a2
        Bouge  2,0,0
        Bouge  1,a3,a3
        Bouge  1,0,0
        Bouge  2,Spin,0
        Bouge  1,0,0
        Bouge  2,0,Spin
        Bouge  2, a1,a1
        Bouge  1, a1,a2
        Bouge  1, 0, 0
        Bouge  2,a1,a2
        Stop   2

        Bouge  2,a1,a1
        Bouge  1,0,0
        Bouge  1, a1, a1
        Bouge  2,a2,a2
        Bouge  1,a2,a1
        Bouge  2,r2,r2

        Bouge  2,a1,a1
        Bouge  1,0,0
        Bouge  1, a1, a1
        Bouge  2,a2,a2
        Bouge  1,a2,a1
        Bouge  2,r2,r2
        Bouge  1,Spin,0
        Bouge  1, 0, 0
        Bouge  1, 0,Spin
        Bouge  2, r2,0
        Bouge  2, a2,0
        Bouge  1, r2,r2
        Bouge  1, a2,a2
        Bouge  2, 0,0

        Bouge  2, a1,r1
        Bouge  2, a1,a1
        Bouge  2, a1,r1
        Bouge  2, a1,a1
        Bouge  2, a1,r1
        Bouge  2, a1,a1
        Bouge  2, a1,r1
        Bouge  2, r1,r1
        Bouge  2, 0,0
        Bouge  1,Spin,0
        Bouge  1, 0, 0
        Bouge  2, r1,0
        Bouge  2, a1,0
        Bouge  1, 0,r2
        Bouge  1, 0,a2
        Bouge  2, 0,0
        Bouge  2, a1,r1
        Bouge  2, r1,a1
        Bouge  2, 0,0
        Bouge  1,Spin,0
        Bouge  1, 0, 0
        Bouge  1, 0,Spin
        Bouge  2, r2,0
        Bouge  2, a2,0
        Bouge  1, r2,r2
        Bouge  1, a2,a2
        Bouge  2, 0,0
        Bouge  1, r3,r3
        Bouge  1, a3,a3
        Bouge  2, 0,0
        Recommence
.End
```

Ce fichier pourrait être environ 2 fois plus long, mais la taille mémoire ne permet pas plus.

Comment modifier cette table et reprogrammer ?
Il faut un éditeur-assembleur, c'est Smile-NG développé à l'EPFL par Patrick Faeh et Sebastian Gerlach ; c'est gratuit. Il faut un programmateur pour transférer le code binaire dans le processeur. Microchip propose le PicKit2, qui sait programmer tous les PICs, et en particulier le 16F630 du Rhodo. Il faut encore l'adaptateur Pk2 utilisable

avec tous les circuits miniatures de Didel. Le coût est de 70.- pour le Pickit2, l'adaptateur et un CD avec les logiciels Smile, Pickit2 et Rhodo.

Si tout est en main, il faut installer. Comme pour tous les outils informatiques, c'est en principe facile – en principe !

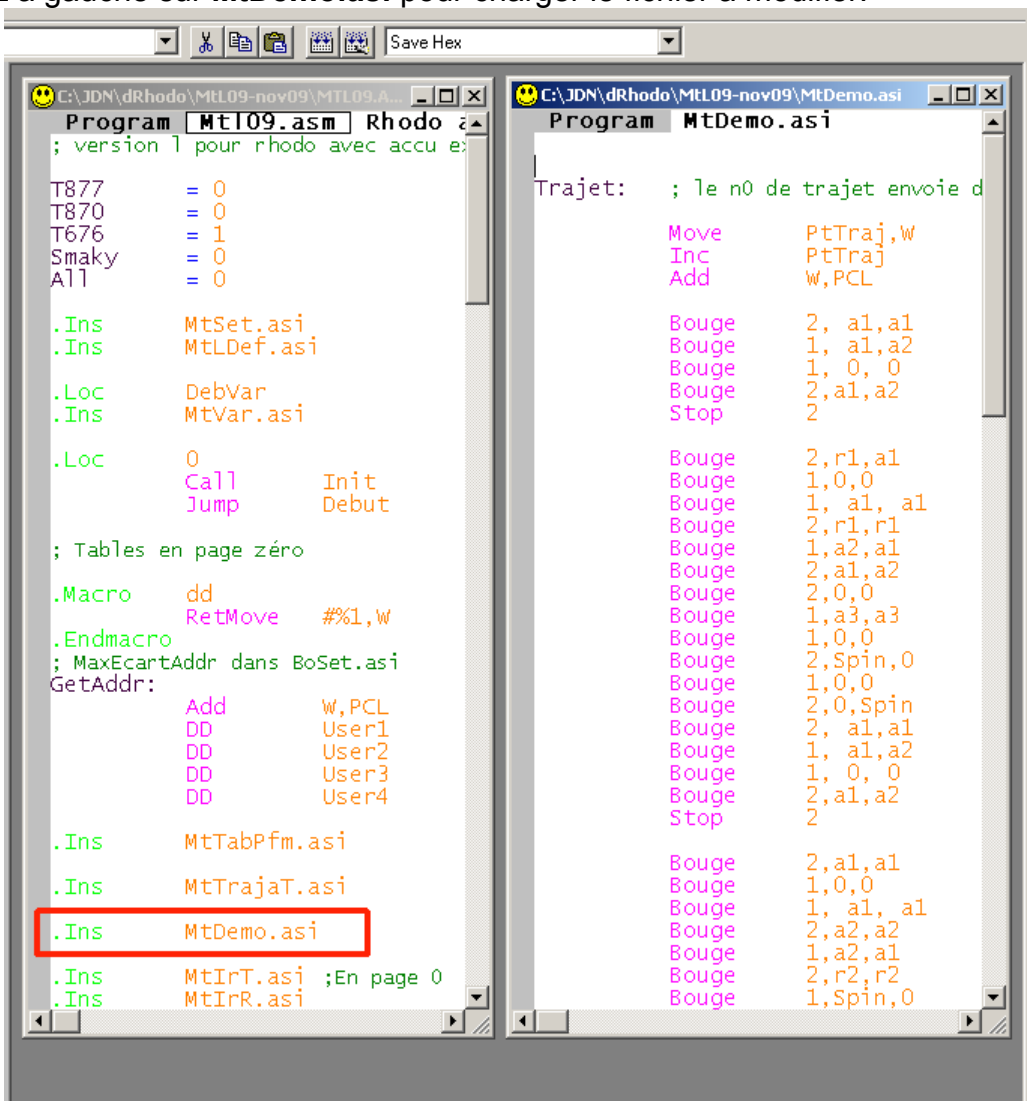
SmileNG Doc et environnement sur le CD
Pickit2 Doc et environnement sur le CD

SmileNG peut se charger depuis http://www.bricobot.ch/docs/SmileNG.zip
Le Pickit2 de Microchip depuis http://www.bricobot.ch/docs/Microchip.zip

Des explications éventuellement plus complètes se trouvent sous
http://www.bricobot.ch/docs/Abimo07.pdf ou www.didel.com/Pickit2.pdf

Les programmes en assembleur sont sur le CD. On les trouves aussi sous
www.didel.com/RhodoMTL09Soft.zip (minuscules et majuscules à respecter)

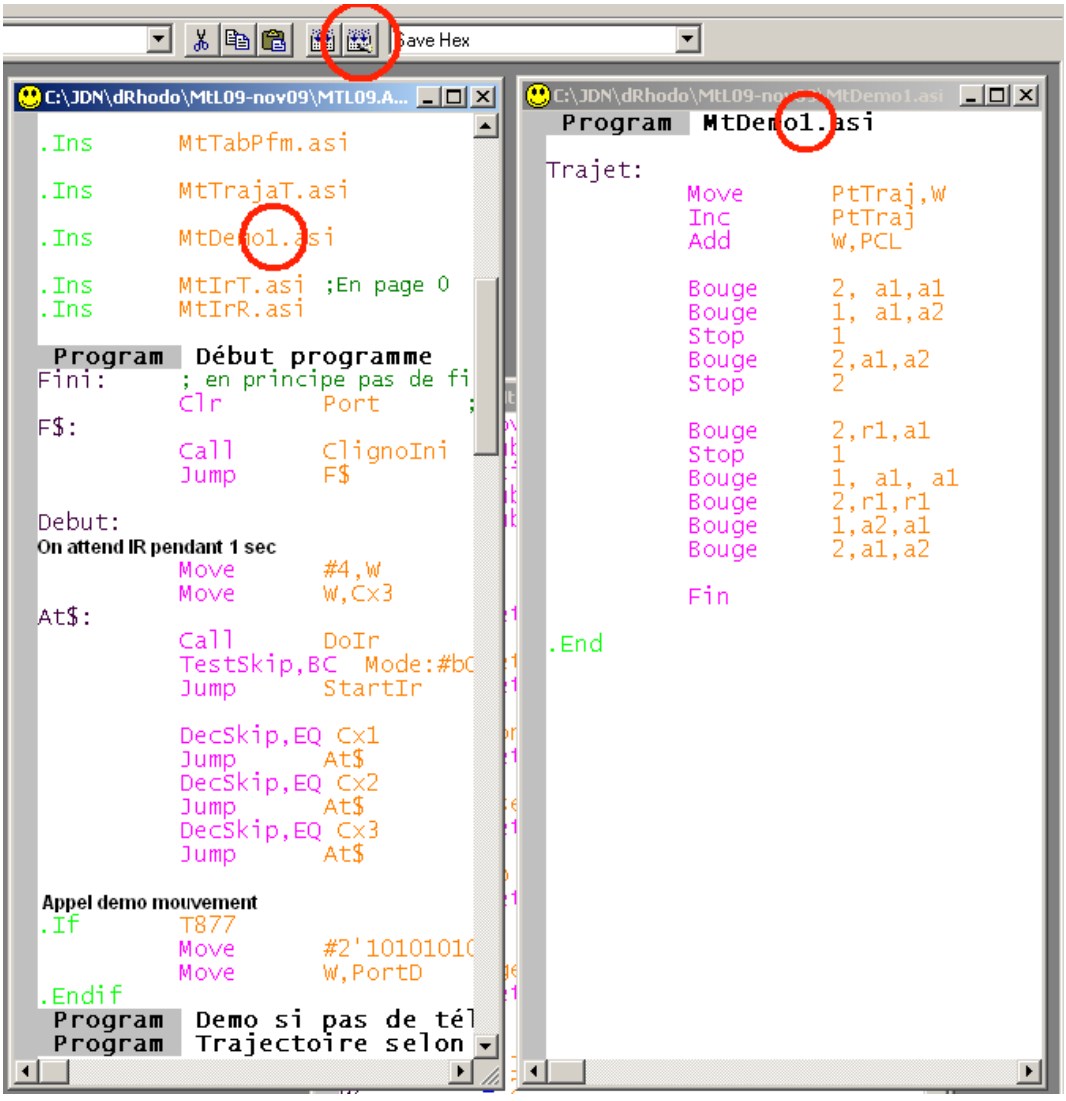
Appelez SmileNG et chargez **Mtl09.asm** (minuscules et majuscules sans importance)
Cliquez à gauche sur **MtDemo.asi** pour charger le fichier à modifier.



Il faut modifier le logiciel du Rhodo, qui est compliqué, mais il est formé de plusieurs modules, et il nous suffit de comprendre et modifier le module **MtDemo.asi** qui code les trajectoires. Comme on vient de le voir, les ordres sont simples.

Attention, il n'y a de la place que pour 105 ordres. L'assembleur mettra un message ''Aie, débordement en page 0'' si c'est trop long.
Ignorez les autres fichiers, le logiciel du Rhodo est évidemment assez compliqué.

Pour se familiariser, commencez par faire quelques mouvements simples que vous pouvez tester immédiatement. Sauvez le fichier MtDemo avec le nom MtDemo1 avant de le modifier. Il faut évidemment mettre à jour ce nom dans le programme principal.



Pour assembler, cliquer sur l'icône cerclée, ou presser la touche F5. Si l'assemblage n'est pas correct, une faute d'orthographe ou un point utilisé à la place d'une virgule sera signalé. Mais pas un paramètre incorrect, les macros sont trop primitives.

Pour exécuter, il faut accrocher d'abord le Rhodo au Pickit2, appeler le programme Pickit2, importer le fichier Mtl08.hex dans le Pickit2 et cliquer sur Write. Voir la documentation, ce n'est pas évident la première fois, et tant mieux si on peut vous montrer cette procédure la plus simple.
Pour les chargements suivants, c'est très efficace : on modifie le source, on assemble, on passe sur le Pickit2 et on clique sur Write. Le message ''reload file'' montre que c'est le fichier modifié qui est repris.

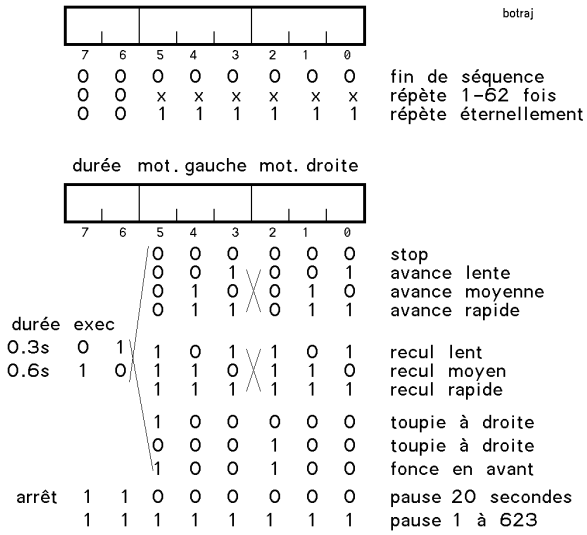
Amusez-vous bien !

Complément pour les curieux et (futurs) spécialistes

Comment coder des trajectoires ?
On pourrait imaginer de les dessiner avec la souris sur l'écran, et avoir un programme qui décompose en petits déplacements et génère un code compatible. Joli projet pour celui qui maîtrise le VisualC, VisualBasic, Java ou autre.
Un élément de trajectoire doit définir la distance, la vitesse, le rayon de courbure. C'est trois nombres qui se succédent dans la mémoire et sont analysés (on dit interprétés) par un programme qui lit successivement ces valeurs, calcule et commande les moteurs.
Avec le Rhodo, pas moyen de mesurer la distance, mais on peut définir la durée. Pas moyen de définir précisément la vitesse (il faudrait un moteur avec un encodeur), donc aussi pas moyen de contrôler le rayon de courbure. Tout ce que l'on peut faire c'est programmer la puissance donnée sur les moteurs. La vitesse dépendra des frottements, du sol. Ce n'est pas une simulation, il faut apprendre à faire de son mieux dans un monde réel imparfait !
De plus, le Rhodo a peu de mémoire et son processeur ne sait pas bien calculer.
Alors, faisons au plus simple. Le PIC permet de préparer des mots de 8 bits dans sa mémoire, grâce à une instruction astucieuse. Prenons un seul octet par déplacement élémentaire pour coder ce qui nous intéresse :

- 2 bits pour la durée (donc la distance parcourue)
- 3 bits pour la vitesse du moteur gauche (en fait la longueur des impulsions envoyées au moteur)
- 3 bits pour le moteur droite

Dans ce codage, il faut encore prévoir comment on va coder la fin de la trajectoire, et ce serait bien de pouvoir la répéter.
Magnifique, tout peut se résumer dans une figure ! Mais il faut savoir la décortiquer.



Si les deux premiers bits qui codent la durée sont à zéro, on a 3 possibilités. Tout à zéro, c'est la fin. Si les 6 bits de droite sont à « un », la séquence sera répétée éternellement. Autrement ces 6 bits disent combien de fois cela sera répété. On pourrait se contenter de répéter 7 fois au maximum, et coder autre chose avec les bits 3,4,5.
Si les deux premiers bits ne sont pas à zéro, ils codent 3 durées, assez courtes. On peut répéter le même ordre pour doubler la durée.
Naturellement, cette durée est définie quelque-part par un paramètre (IniDurPDiv) que l'on peut modifier pour accélérer ou ralentir la séquence.
Les 6 bits de poids faibles sont deux fois trois bits, pour chaque moteur. Il faut pouvoir les arrêter, aller en avant en arrière. Huit combinaisons sont possibles, et permettent de coder seulement 3 vitesses dans chaque sens. Ces codes de vitesse passent par une table pour définir la vitesse. Là aussi, un spécialiste peut agir.
Comme il y a encore une position de codage, elle est triée pour déclencher une rotation du robot sur soi-même, à pleine vitesse.

Ecrivons une première trajectoire en mettant en mémoire quelques octets :

- 01000010 seul le moteur droite avance à mi-vitesse pendant 0.3s
- 10111000 seul me loteur gauche recule à pleine vitesse pendant 0.6 sec

Ce n'est pas très patique d'écrire comme cela. Donnons des noms aux 8 valeurs possibles pour les moteurs
On donnera aussi des noms aux 4 possibilités de codage des 2 premiers bits. Pour l'instant, utilisons pour les deux premiers bits les chiffres 0, 1, 2, 3.
0 termine ou répète, 1, 2 pour les 2 durées et 3 vu plus loin pour le repos (prévu pour des démos de longues durées).

```
a1    = 1    ; avance lente
a2    = 2
a3    = 3
r1    = 5
r2    = 6
r3    = 7    ; recul rapide
spin  = 4    ; tourne sur place
```

La trajectoire ci-dessus s'écrit maintenant

```
1, 0,a2
2, r3,0
```

C'est nettement plus lisible, et une macro de l'assembleur va convertir ces trois paramètres dans des bits placés au bon endroit :

```
.Macro  Bouge
        RetMove     #%1*64+%2*8+%3,W
.EndMacro
```

Si on écrit Bouge 1,0,a2 l'assembleur va prendre l'instruction Retmove et calculer la valeur associée en prenant dans la ligne les paramètres %1 (le premier) %2 %3. Les trois paramètres doivent figurer, séparés par des virgules.
L'assembleur calcule 1 * 64 + 0 * 8 + 2 = 66, en binaire 01000010. L'assembleur calcule tout en binaire, c'est plus pratique pour nous d'écrire 64 que 2**6 ou 2'0100000. Pour un débutant, ces détails ne sont pas faciles à comprendre, mais il n'y a pas besoin de bien comprendre pour coder une trajectoire.

JD 080617/091117