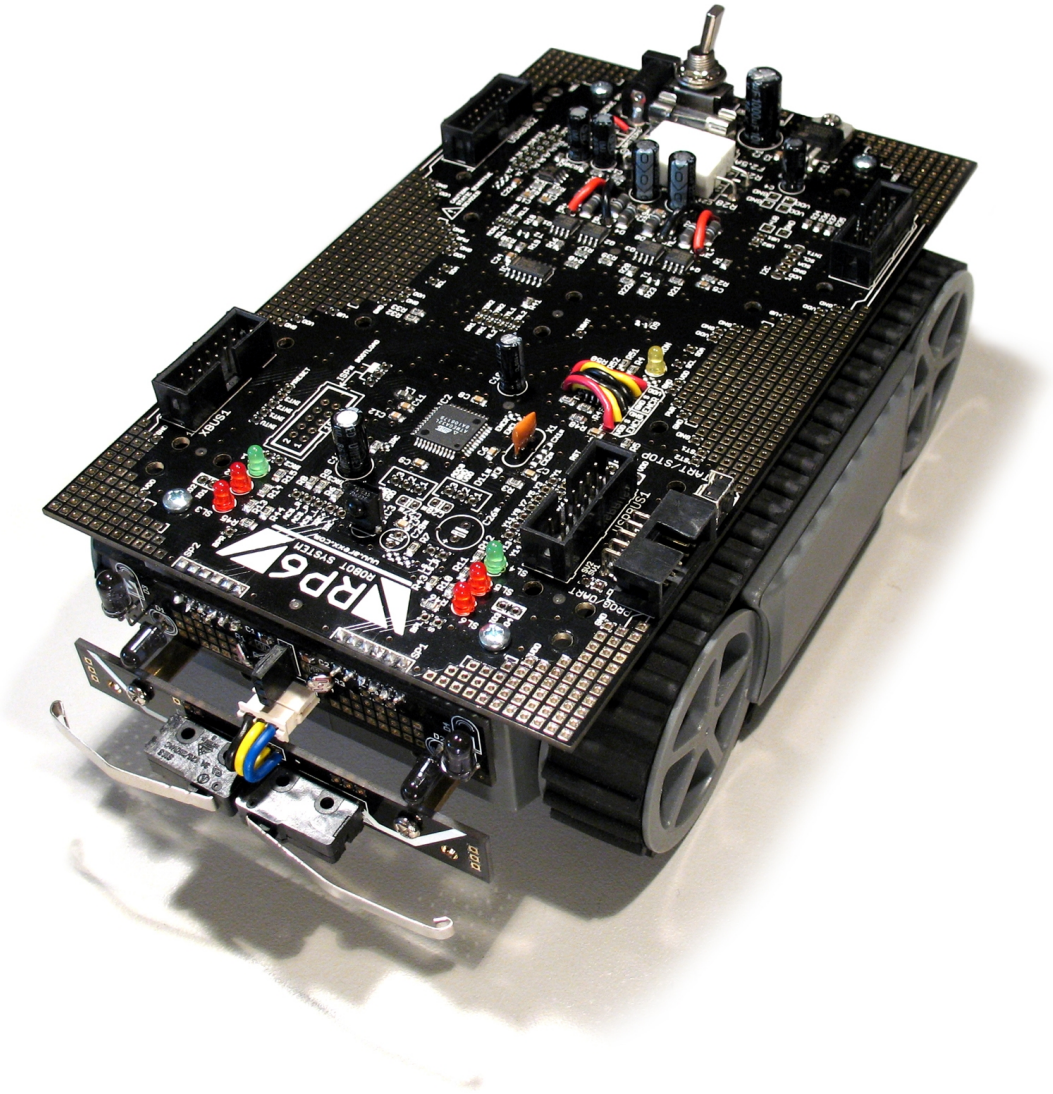
RP6 ROBOT SYSTEM

RP6 ROBOT BASE



AREXX Engineering

RP6-BASE

www.arexx.com

RP6
Système Robotique

Manuel d'Utilisation

- Français (French) -

Version RP6-BASE-FR-20071029

INFORMATIONS IMPORTANTES!
A lire absolument!

Avant la mise en service du RP6 ou de ses accessoires, lisez attentivement ce manuel ainsi que, le cas échéant, le manuel de ou des accessoires! Il vous explique la bonne utilisation et attire votre attention sur des dangers éventuels! Par ailleurs, il contient des informations importantes qui ne sont pas forcément connues de tous les utilisateurs.

Le non-respect des consignes contenues dans ce manuel invalide la garantie! Par ailleurs, AREXX Engineering décline toute responsabilité pour des dommages quels qu'ils soient, qui résultent du non-respect de ce manuel.

Lisez surtout le chapitre « Consignes de Sécurité »!

Connectez l'interface USB sur votre PC seulement après avoir lu le chapitre 3 -"Mise en Service" et installé correctement le logiciel!

Impressum

©2007 AREXX Engineering

Nervistraat 16
8013 RS Zwolle
The Netherlands

Tel.: +31 (0) 38 454 2028
Fax.: +31 (0) 38 452 4482

Restrictions de garantie et de responsabilité

La garantie d'AREXX Engineering se limite au remplacement ou à la réparation du robot pendant la période légale de garantie si la défaillance provient d'un défaut de fabrication tel qu'un défaut mécanique ou une implantation erronée ou manquante de composants électroniques à l'exception de tous les composants implantés par des connecteurs. AREXX Engineering décline toute responsabilité pour des dommages résultant directement ou indirectement de l'utilisation du robot à l'exception des droits basés sur les obligations légales de responsabilité du fabricant.
Si vous modifiez le robot d'une manière irréversible (p.ex. soudage d'autres composants, perçage de trous, etc) ou le robot est endommagé suite au non-respect de ce manuel, tout droit à garantie s'éteint!

Nous ne pouvons pas garantir que le logiciel fourni répondra aux attentes individuelles ou pourra travailler sans aucune interruption, ni défaillance.
Par ailleurs, le logiciel est librement modifiable et chargé dans l'appareil par l'utilisateur. Par conséquent, l'utilisateur assume la totalité du risque concernant la qualité et la performance de l'appareil y compris du logiciel.
AREXX Engineering garantit la fonctionnalité des exemples d'application fournis à condition de respecter les conditions spécifiées dans les caractéristiques techniques. Si, au-delà, le robot ou le logiciel PC s'avèrent défaillants ou insuffisants, tous les frais de service, de réparation ou de correction sont à la charge du client.
Respectez également les accords de licence indiqués sur le CD ROM!

Symboles

Les symboles suivants sont utilisés dans ce manuel:

Le point d'exclamation dans le triangle attire l'attention sur des consignes très importantes qu'il faut respecter scrupuleusement. Une erreur pourrait entraîner la destruction du robot ou de ses accessoires et même mettre en danger votre santé ou celle d'autrui!

Le « i » dans un cercle attire l'attention sur des chapitres qui contiennent des astuces et conseils utiles ou des informations de fond. Ce n'est pas toujours essentiel de tout comprendre mais généralement très utile.

Table des Matières

1. Introduction

Le RP6 est un robot mobile et autonome à un prix abordable destiné à donner un aperçu du monde fascinant de la robotique aux débutants mais aussi aux développeurs confirmés en électronique et en informatique.

Le robot est livré complètement monté et convient parfaitement à tous ceux qui ont peu d'expérience en soudage et en bricolage et qui souhaitent se concentrer sur le logiciel. Cela ne signifie pourtant pas qu'il n'est pas possible d'ajouter ses propres circuits et extensions! Au contraire: Le RP6 est conçu pour des extensions et peut servir de point de départ à un grand nombre d'expériences intéressantes!

Il est le successeur du très populaire « C-Control Robby RP5 » (CCPR5, RP5 signifie ici « Robot Project 5 ») commercialisé en 2003 par Conrad Electronic SE. Cependant, son électronique a profondément changé. Le microprocesseur sur le RP6 n'est plus le C-Control 1 de Conrad Electronic et le robot n'est par conséquent plus programmable directement en Basic. Il a été remplacé par l'ATMEGA32 de chez Atmel qui est beaucoup plus performant et programmable en C. Un module d'extension est en cours de finalisation qui permettra d'utiliser des versions C-Control plus récentes sur le robot. Ainsi, le robot sera également programmable dans le langage Basic plus simple et acceptera un grand nombre d'interfaces avec une capacité de mémoire supplémentaire.

Une autre nouveauté est l'interface USB livrée avec le robot qui constitue un système d'extension beaucoup plus flexible, offrant de meilleures possibilités de montage, des encodeurs largement améliorés (résolution 150x supérieure à celle du prédécesseur), une alimentation électrique plus précise (le prédécesseur nécessitait un autre module d'extension pour cela), un pare-chocs avec deux détecteurs tactiles et encore bien d'autres. Le module d'expérimentation pour vos propres circuits livré avec le robot, est également une nouveauté remarquable. Globalement le rapport qualité/prix s'est largement amélioré comparé au prédécesseur.

La mécanique du RP5 a été reprise mais optimisée pour un fonctionnement beaucoup plus silencieux. Quelques perforations supplémentaires ont été ajoutées afin de permettre le montage d'extensions mécaniques.

Le processeur du RP6 est compatible avec les robots ASURO et YETI qui utilisent tous deux le plus petit ATMEGA8 et les mêmes outils de développement (WinAVR, avr-gcc). Toutefois, ASURO et YETI sont livrés en kit et doivent être montés par l'utilisateur. Le RP6 est destiné aux utilisateurs plus exigeants qui demandent d'excellentes possibilités d'extension, un meilleur microprocesseur et davantage de détecteurs.

Plusieurs modules d'extension sont prévus voire déjà disponibles qui vous permettent d'élargir les capacités du robot. Il s'agit notamment de l'extension C-Control mentionnée plus haut, un module d'extension comportant un autre MEGA32 et bien évidemment la plaquette d'expérimentation pour vos propres circuits qui est également disponible séparément (vous pouvez en monter plusieurs sur le robot). D'autres modules intéressants sont en cours de conception et vous pouvez bien sûr développer également vos propres modules ! **Nous vous souhaitons beaucoup de plaisir et de succès avec le RP6 Robot System!**

1.1. Support technique

Si vous avez des questions ou rencontrez des problèmes, vous pouvez joindre notre assistance technique par Internet (Avant de nous contacter, **lisez entièrement le mode d'emploi!** Par expérience nous savons que la plupart des questions y trouveront une réponse! Reportez-vous également à l'annexe A – Diagnostic de défaillance):

- par notre forum: `http://www.arexx.com/forum/`

- par courrier électronique: `info@arexx.nl`

Notre adresse postale figure dans l'impressum de ce manuel. Des informations de contact plus actualisées, des mises à jour de logiciel et autres informations figurent sur notre page d'accueil:

http://www.arexx.com/

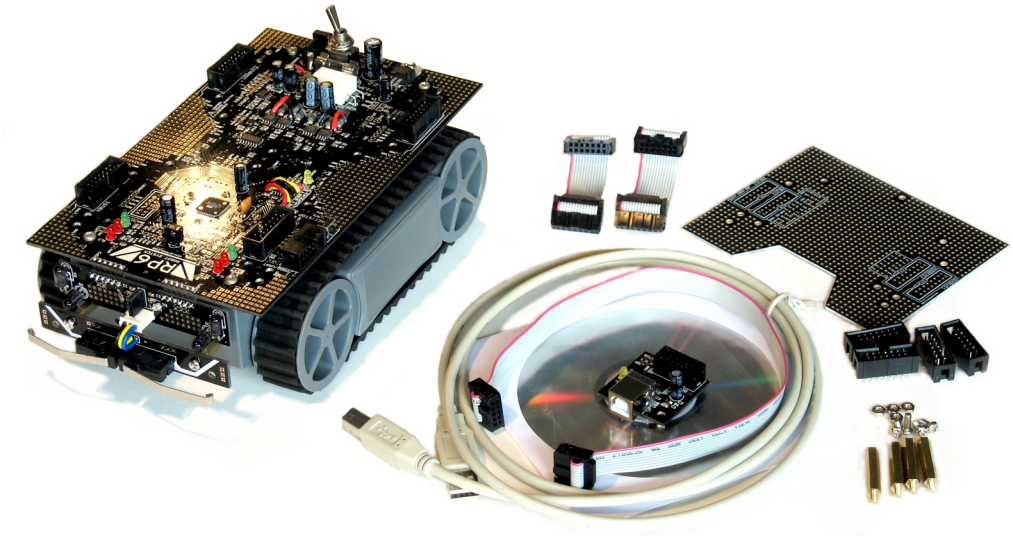
et sur la page d'accueil du robot:

http://www.arexx.com/rp6

1.2. Contenu du carton

Le carton du RP6 doit contenir les pièces suivantes:

- **Robot RP6 entièrement monté**
- RP6 Interface USB
- Cordon USB A->B
- Câble en nappe à 10 points
- CD-ROM RP6
- Instructions rapides
- **Plaquette d'expérimentation RP6**
- 4 entretoises 25mm M3
- 4 vis M3
- 4 écrous M3
- 4 connecteurs nappe 14 broches
- 2 câbles nappe 14 points



1.3. Propriétés et caractéristiques techniques

Ce chapitre donne un aperçu des capacités du robot et sert en même temps à l'introduction de certaines notions et désignations de composants du robot que certains utilisateurs ne connaissent peut-être pas encore. Une grande partie est expliqué plus en détail dans le manuel.

Propriétés, Composants et Caractéristiques techniques du RP6:

- **Microcontrôleur performant Atmel ATMEGA32 8-Bit**
  - ◇ Vitesse 8 MIPS (=8 millions d'instructions par seconde) à un cycle horloge de 8MHz
  - ◇ Mémoire: 32KB Flash ROM, 2KB SRAM, 1KB EEPROM
  - ◇ Librement programmable en C (avec WinAVR / avr-gcc)!
  - ◇ ... et beaucoup plus! Vous trouverez d'autres détails dans le chapitre 2.
- **Système d'extension flexible basé sur le bus I²C**
  - ◇ Ne nécessite que deux conducteurs de signaux (TWI -> "Two Wire Interface")
  - ◇ Vitesse de transmission jusqu'à 400kBit/s
  - ◇ Basé sur Maître-> Esclave
  - ◇ Jusqu'à 127 esclaves peuvent être connectés simultanément sur le bus
  - ◇ Système de bus très répandu: Il existe un grand nombre de circuits imprimés, de détecteurs et autres, disponibles chez différents fournisseurs qui, pour la plupart, se branchent directement.
- **Possibilités de montage symétrique des modules à l'avant et l'arrière**
  - ◇ En théorie, un nombre illimité de modules d'extension peut être empilé mais en raison de la consommation électrique/poids, 6 à 8 modules semblent raisonnables (-->3 – 4 à l'avant et à l'arrière).
  - ◇ 22 trous de montage libres de 3,2mm sont disponibles sur la carte-mère et 16 autres sur le châssis du robot ce qui fait au total 38 perforations. En plus, le châssis dispose de beaucoup d'espace pour vos propres perforations!
- **Plaquette d'expérimentation déjà incluse!** (voir photo du contenu du carton)
- **Interface PC USB** pour le téléchargement du programme du PC sur le microcontrôleur

◊ Filaire pour une vitesse maximale. Le téléchargement du programme se fait normalement à 500kBaud – la mémoire complètement libre du microcontrôleur pour le programme est écrite en quelques secondes (30Ko, 2ko sont réservés pour le chargeur d'amorçage (bootloader)).

◊ L'interface peut être utilisée pour la programmation de tous les modules d'extension disponibles pour le RP6 qui possèdent un microcontrôleur AVR.

◊ S'utilise pour la communication avec le robot ou avec les modules, ce qui facilite largement la recherche d'erreurs dans les programmes puisque vous pouvez envoyer des mesures, messages en texte et autres données dans le PC.

◊ Le driver de l'interface crée une interface série virtuelle sous tous les systèmes d'exploitation courants comme Windows 2000/XP et Linux qui est utilisable avec presque tous les programmes de terminal et un logiciel personnalisé.

◊ Afin de faciliter le téléchargement du programme, le logiciel **RP6Loader** est inclus. Il contient également un petit terminal pour communiquer avec le robot par des messages de texte et tourne sous Windows et Linux.

- **Nouveau système d'entraînement performant par chenilles** afin de minimiser le bruit *(par rapport au prédécesseur CCRP5...)*

◊ Deux moteurs DC 7,2V puissants

◊ Vitesse maximale 25cm/sec (30cm/sec sans limitation par le logiciel). En fonction, entre autre, de l'état de charge/qualité des accumulateurs et du poids total.

◊ Coussinets frittés auto-lubrifiants sur les quatre essieux de 4mm.

◊ Deux chenilles en caoutchouc

◊ Surmonte sans problème de petits obstacles (jusqu'à env. 2cm de hauteur) tels que bords de tapis, irrégularités du sol, magazines posés par terre, et autres. Monte des pentes jusqu'à 30% (avec la platine Bumper; sans Bumper et avec 2 modules d'extension maximum, il arrive à monter des pentes jusqu'à 40% en fonction de la nature du sol).

- **Deux moteurs MOSFET performants** (ponts en H)

◊ Nombre de tours et sens de rotation sont commandés par le microcontrôleur.

◊ Deux **détecteurs de courant** ayant une plage de mesure allant jusqu'à env. 1,8A pour les deux moteurs (idéal pour réagir rapidement à un blocage et une surcharge du moteur).

- **Deux encodeurs de haute résolution** pour mesurer la vitesse et la distance parcourue

◊ Résolution de 625 CPR (« Counts per Revolution » = Incréments par tour), cela signifie que les encodeurs comptent 625 segments du disque d'encodage de la

roue par tour de roue! (Sur l'ancien RP5, ce n'étaient que env. 4 CPR)

◊ Possibilité de mesure de vitesse et de réglage précis et rapide!

◊ Résolution de parcours élevée d'env. 0,25mm par incrément!

- **Système anti-collision (Anti Collision System, ACS)** qui sait reconnaître des obstacles à l'aide d'un récepteur infrarouge et de deux diodes infrarouges montées sur la platine et dirigées vers la droite et la gauche.

  ◊ Reconnaît si des objets se trouvent au milieu, à gauche ou à droite du robot.

  ◊ La portée/puissance de transmission est réglable en plusieurs niveaux afin de reconnaître également des objets faiblement réfléchissants.

- **Système de communication Infrarouge (IRCOMM)**

  ◊ Capte les signaux de télécommandes à infrarouge normales de téléviseurs ou de magnétoscopes. Ainsi le robot peut être dirigé avec une télécommande normale (RC5-)! Le protocole est adaptable au moyen d'un logiciel. Seul le protocole courant RC5 est programmé d'usine.

  ◊ Peut être utilisé pour la communication avec plusieurs robots (réflexion sur le plafond ou contact visuel) ou pour la transmission de données télémétriques.

- **Deux détecteurs de lumière** – p.ex. pour les mesures de luminosité et la poursuite d'une source lumineuse.

- **Deux détecteurs de pare-chocs (Bumper)** pour identifier des collisions

- **6 LEDs d'état** – afin de représenter des détecteurs et états de programme

  ◊ Quatre des ports à LED peuvent également servir pour d'autres fonctions!

- **Deux canaux convertisseurs analogique/numérique (CAN)** pour vos propres détecteurs. (Ils sont également utilisables comme des broches I/O normales).

- **Régulateur de tension précis de 5V**

  ◊ Charge maximale: 1.5A

  ◊ Grande surface de refroidissement en cuivre sur la platine.

  ◊ Le courant permanent ne devrait pas dépasser 1A sans refroidissement supplémentaire! (Il est recommandé de ne pas dépasser une charge permanente maximale de 800mA)

- **Fusible 2,5A** facile à remplacer.

- **Faible consommation de courant de repos** de moins de 5mA (et env. 17 à 40mA en fonctionnement. Varie en fonction de ce qui est en service (LEDs, détec-

teurs, etc.). Cette indication ne s'applique qu'à l'électronique, sans moteurs et sans modules d'extension!)

- **Alimentation avec 6 accumulateurs NiMH LR6** (non inclus)

  ◊ P.ex. Panasonic ou Sanyo (NiMH, 1.2V, 2500mAh, HR-3U , Size AA HR6) ou Energizer (NiMH, 1.2V, 2500mAh, NH15-AA)

  ◊ Durée de fonctionnement env. 3 à 6 heures selon la charge et la qualité/capacité des accus (dépend bien sûr aussi de la fréquence et de la durée d'utilisation des moteurs et de leur charge! Si les moteurs tournent peu, le robot peut fonctionner beaucoup plus longtemps).

- **Branchement pour chargeurs externes** – le commutateur principal du robot commute entre « Charge/Arrêt » et « Fonctionnement/Marche ».

  ◊ Il suffit de faire sortir des contacts afin de connecter des alimentations externes ou des accumulateurs supplémentaires sur le robot.

  ◊ Des blocs d'alimentation externes compatibles sont le chargeur rapide Delta-Peak Voltcraft 1A / 2A, Ansmann ACS110, ACS410 ou AC48). Les chargeurs varient en équipement et en temps de charge de 3 à 14h.

- **6 petites surfaces d'extension** sur la carte-mère (et 2 tout petits sur la platine du détecteur), afin de pouvoir implanter ses propres circuits de détection sur la carte-mère. Ainsi, vous pourriez ajouter d'autres détecteurs IR tout autour du robot afin qu'il réagisse encore mieux aux obstacles.

- **De nombreuses possibilités de modification!**

Par ailleurs, quelques exemples de programmation en C ainsi qu'une bibliothèque de fonctions très riche sont inclus afin de faciliter au maximum la programmation.

Le site Internet du robot contiendra bientôt d'autres programmes et mises à jour pour le robot et ses modules d'extension. Vous pouvez également échanger vos propres programmes avec d'autres utilisateurs RP6 par Internet. La bibliothèque RP6Library et les exemples de programmation sont sous la licence Open Source GPL!

1.4. Quelles sont les Capacités du RP6?

A peu près nulles lorsqu'il sort de son emballage!

Il n'acquiert ses capacités que par la programmation – et c'est à vous et à votre créativité de déterminer ce qu'il saura faire exactement! C'est ce qui présente l'intérêt des kits de robotique et autres: concrétiser ses propres idées et améliorer ou modifier ce qui existe déjà. Évidemment, vous pouvez commencer par essayer et modifier les programmes déjà écrits afin d'avoir une idée des possibilités installées d'usine.

Ici, nous ne citons que quelques exemples. C'est à vous de décider ce que vous voulez faire du RP6. Il y a encore plein d'autres possibilités (voir page suivante)!

D'usine, le RP6 peut p.ex.. ...:

- ... se déplacer en toute autonomie (c'est-à-dire *sans* télécommande)
- ... éviter des obstacles
- ... chercher/suivre des sources lumineuses
- ... reconnaître des collisions, des moteurs bloqués et un faible niveau de charge de la batterie et réagir
- ... mesurer et régler automatiquement la vitesse de déplacement – pratiquement indépendamment de l'état de charge des accus, du poids du robot, etc. (c'est la principale application de l'encodeur à haute résolution)
- ... parcourir des distances données, tourner à un angle défini et déterminer le chemin parcouru (toutefois avec quelques déviations, voir chapitre 2)
- ... rouler selon un schéma ou une figure déterminés tels que des cercles, polygones, etc.
- ... Echanger des données avec d'autres robots ou appareils par le biais du système de communication à infrarouge et d'être commandé par ce système. Cela fonctionne avec des télécommandes de téléviseurs, vidéo ou HiFi. Ainsi, vous pouvez commander le robot comme une voiture télécommandée.
- ... transmettre des valeurs du détecteur et autres données par l'interface USB au PC
- ... évoluer facilement et simplement par le système de bus.
- ... être modifié et adapté à ses propres idées. Il suffit de consulter les schémas techniques sur le CD et de regarder de plus près la platine. Cependant, ne faites des modifications que si vous savez ce que vous faites! Il vaut mieux commencer avec une des plaquettes d'extension – surtout si vous n'avez encore jamais soudé un circuit...

1.5. Propositions d'Application et Idées

Le RP6 est conçu pour évoluer – avec des modules d'extension et des détecteurs supplémentaires, le RP6 pourrait « apprendre » les choses suivantes (certaines des tâches nommées ici sont assez compliquées et pas très simples à réaliser. Les sujets sont *grossièrement* triés en fonction de leur degré de difficulté):

- Equiper le robot avec d'autres contrôleurs et donc lui donner une capacité, des mémoires, ports I/O et convertisseurs A/D, etc. supplémentaires. Ou, comme abordé brièvement dans les exemples de programme, l'élargir par de simples extensions de port I²C et convertisseurs A/D.
- Afficher les données des détecteurs et des textes sur un écran à cristaux liquides sur le robot.

- Réagir aux bruits et émettre des sons
- L'équiper de détecteurs d'objets, à ultrasons, à infrarouge, etc. supplémentaires qui déterminent la distance jusqu'aux objets afin de mieux les éviter.
- Suivre des lignes noires au sol
- Chercher/suivre d'autres robots ou objets.
- Commander le robot par infrarouge à partir du PC (nécessite un matériel spécial – est malheureusement impossible avec des interfaces IRDA normales!) ou utiliser directement un module HF.
- Commander le RP6 avec un PDA ou un Smartphone (ici cela signifie que ces appareils sont montés sur le robot et ne sont pas utilisés comme une télécommande bien que ce serait également faisable).
- Ramasser des objets (p.ex. des bougies chauffe-plat ou des petites billes, des pièces métalliques, etc.
- Monter un petit bras/griffe pour attraper des objets.
- Naviguer avec une boussole électronique et reconnaître des balises infrarouge (donc de petites tours équipées de nombreuses LEDs infrarouge dont la position est connue avec précision) et déterminer ainsi sa propre position dans l'espace et se diriger vers des objectifs pré-déterminés
- Si le robot est équipé d'un dispositif de tir et d'autres détecteurs, il pourrait même jouer au football!
- ... et tout ce qui vous vient à l'esprit!

Tout d'abord vous devez lire ce manuel et vous familiariser avec le robot et sa programmation. Ceci était juste une petite mise en bouche.

Et si la programmation ne fonctionne pas du premier coup, ne jetez pas immédiatement l'éponge. Le début est toujours difficile!

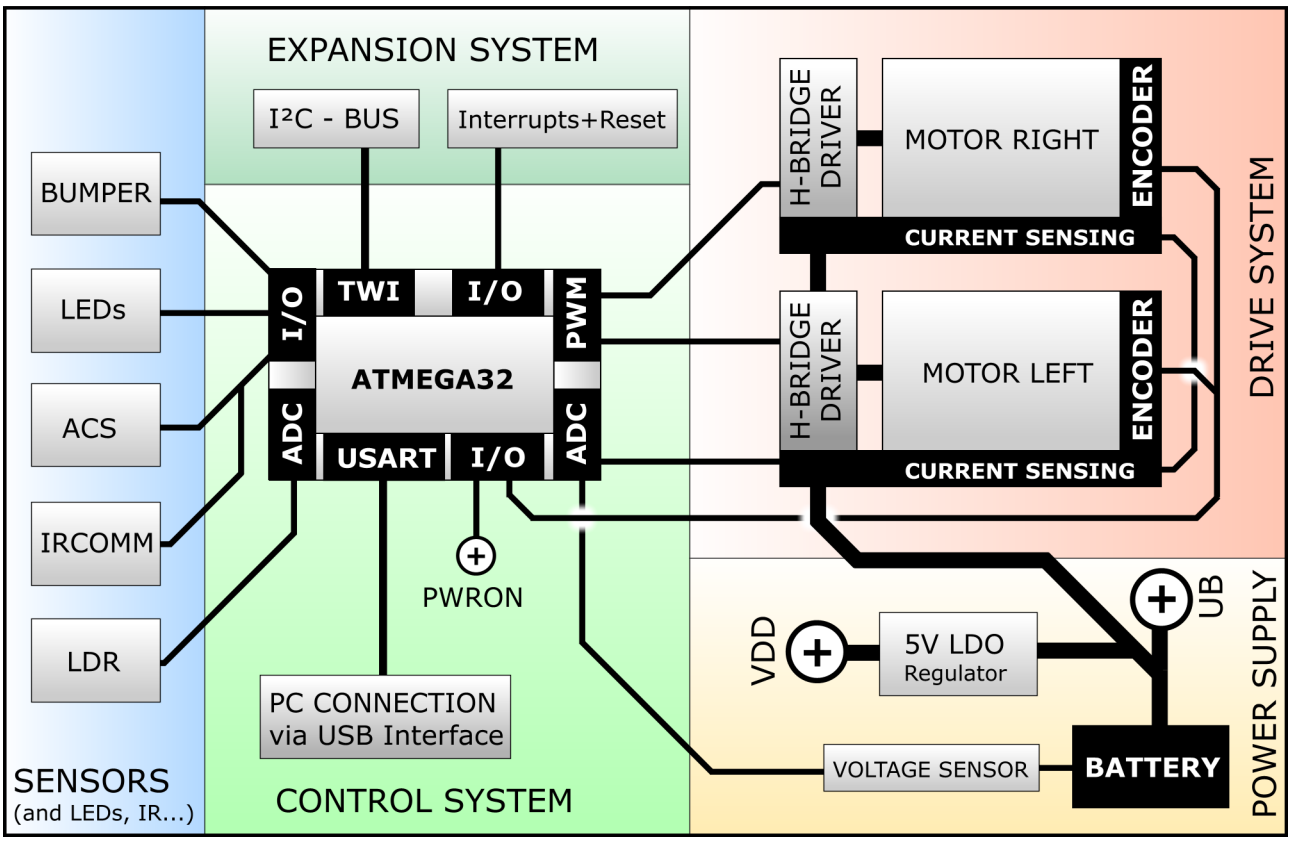
2. Le RP6 en détail

Ce chapitre traite plus en détail des composants matériels du RP6. Nous nous consacrerons au fonctionnement de l'électronique et à l'interaction avec le logiciel sur le microcontrôleur. Si vous avez déjà de l'expérience avec des microcontrôleurs et l'électronique, il vous suffira probablement de survoler bon nombre de paragraphes dans ce chapitre. Nous recommandons cependant à tous les débutants de la robotique de lire attentivement le chapitre en entier afin d'avoir une meilleure idée du fonctionnement du RP6.

Si vous ne pouvez pas attendre car vous voulez immédiatement essayer le robot, vous pouvez aller directement au chapitre 3 mais vous devriez revenir ultérieurement sur ce chapitre. Il est très utile pour comprendre la programmation car vous voulez certainement savoir ce que vous commandez avec le logiciel et comment cela fonctionne à peu près.

Nous n'allons pas approfondir le sujet mais il peut y avoir plusieurs points dans ce chapitre qui ne sont pas si faciles à comprendre à première vue. Pourtant l'auteur a essayé de tout expliquer le plus clairement possible.

Cela vaut la peine de rechercher d'autres informations sur Internet ou dans des livres. `http://www.wikipedia.fr/` est souvent un bon point de départ.

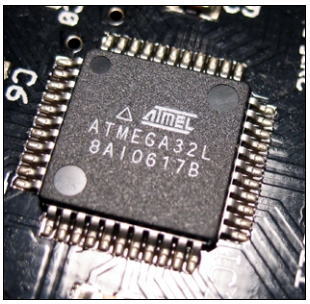


Les images sont souvent plus éloquents que des mots. Donc, nous allons commencer avec un schéma fonctionnel du RP6 qui est une représentation fortement simplifiée des composants électroniques du robot.

Le RP6 peut être divisé grossièrement en 5 groupes de fonctions:

- Système de commande (Control System)
- Alimentation (Power Supply)
- Détections, communication IR et indications (sensors) – tout ce qui communique avec l'extérieur ou mesure des valeurs.
- Système d'entraînement (Drive System)
- Système d'extension (Expansion System)

2.1. Système de Commande



Vous voyez sur le schéma fonctionnel que l'élément central du robot est le microcontrôleur 8 bits ATMEGA32 de chez ATMEL (voir fig.).

Un microcontrôleur est un petit ordinateur complet qui tient sur une puce. La différence avec un gros ordinateur devant lequel vous vous trouvez probablement à l'instant, est qu'il possède beaucoup moins de choses ou qu'elles lui manquent complètement. Evidemment, il ne dispose pas d'un immense disque dur ou de plusieurs Giga octets de RAM! Un microcontrôleur n'a pas non plus besoin de tout cela. Le MEGA32 ne possède p.ex. « que » 32Ko (32768 octets) de mémoire flash – ce qui constitue quasiment son « disque dur ». Il y sauvegarde toutes les données du programme. Sa mémoire de travail ne fait que 2Ko (2048 octets) ce qui est plus que suffisant pour nos projets. (A titre de comparaison: Le microcontrôleur sur l'ancien RP5 avait tout juste 240 octets de RAM dont presque la totalité était utilisée par l'interprète Basic).

Comment est-ce qu'un microcontrôleur peut travailler avec si peu de mémoire? C'est très simple: Il ne traite pas de grandes quantités de données et n'a besoin ni d'un système d'exploitation comme Linux ou Windows, ni d'une surface graphique ou autre. Il n'y a qu'un seul programme qui tourne sur le contrôleur, à savoir le nôtre!

Ce n'est absolument pas un inconvénient mais un de ses plus grands avantages par rapport à un gros ordinateur (en plus des besoins énergétiques, de l'espace et des coûts). Il peut traiter des choses très sensibles au temps qui doivent être exécutées à la microseconde près. Il est généralement possible de déterminer avec précision combien de temps il lui faut pour exécuter une certaine partie du programme car la performance de calcul n'est pas partagée avec d'autres programmes comme sur un PC normal.

Le contrôleur sur le RP6 est réglé sur un cycle horloge de 8MHz et exécute par conséquent 8 millions d'instructions par seconde. Il pourrait travailler à 16MHz mais la consommation d'énergie serait trop importante. Cette vitesse est tout à fait suffisante pour les tâches qu'il accomplit habituellement. (A titre de comparaison: le RP5 n'exécutait que 1000 instructions en Basic par seconde avec un cycle horloge de 4MHz. C'est pour cela que, entre autres, la commande ACS a dû être prise en charge par un

autre petit contrôleur ce qui n'est plus nécessaire maintenant). Si vous souhaitez plus de performance, vous pouvez connecter un ou plusieurs contrôleurs supplémentaires sur les modules d'extension. Le module RP6Control disponible séparément offre, entre autres, un deuxième MEGA32 qui est réglé sur la vitesse maximale de 16MHz.

Le contrôleur communique avec le monde extérieur par ses 32 broches I/O (Input/Output donc broches d'entrée et de sortie). Les broches I/O sont organisées en « ports » contenant chacun 8 broches I/O. Le MEGA32 en possède 4: PORTA à PORTD.

Le contrôleur lit l'état logique des ports et le traite par le logiciel. De même, il peut aussi envoyer des états logiques par les ports et commuter de faibles charges telles que des LEDs (jusqu'à env. 20mA).

En outre, le contrôleur dispose de nombreux modules de matériel intégrés qui peuvent assumer des fonctions spécifiques. Dans la plupart des cas, ces fonctions ne seraient pas ou difficilement réalisables par le logiciel et prendraient trop de temps précieux de traitement. Ainsi, il possède p.ex. trois différents « timers » qui peuvent, entre autres, compter des cycles horloge et sont souvent utilisés pour mesurer un temps. Les timers fonctionnent indépendamment de l'exécution du programme du microcontrôleur et celui-ci peut exécuter d'autres tâches pendant qu'il attend p.ex. une certaine position de compteur. Un des timers est utilisé dans le RP6 pour générer deux signaux à modulation de largeur d'impulsion (MLI) (PWM = « Pulse Width Modulation ») qui permettent de régler la tension des moteurs et donc leur nombre de tours. Lorsque l'un des timers a été configuré dans le programme, il travaille en toute autonomie en tâche de fond. Vous trouverez d'autres informations sur le MLI dans le chapitre « Système d'entraînement ».

D'autres modules du MEGA32 sont p.ex.:

- Une interface série (UART) qui est utilisée pour la communication avec le PC par l'interface USB. Tant que l'interface n'est pas connectée, on pourrait y brancher d'autres microcontrôleurs avec un UART.

- Le module "TWI" (="Two Wire Interface", donc interface à deux conducteurs) pour le bus I²C du système d'extension.

- Un convertisseur analogique/numérique ("Analog to Digital Converter", CAN) possédant 8 canaux d'entrée qui arrive à mesurer la tension à une résolution de 10octets. Cela permet de surveiller la tension des accumulateurs sur le RP6, d'évaluer les détecteurs de courant moteur et de mesurer l'intensité lumineuse par le biais de deux résistances photo-dépendantes.

- Trois entrées externes d'interruption. Elles génèrent un événement d'interruption qui coupe le déroulement du programme dans le contrôleur et le fait passer sur une routine d'interruption. Il exécute d'abord cette routine et retourne ensuite à l'emplacement dans le programme où il se trouvait avant l'interruption. Ceci est p.ex. utilisé pour les encodeurs. Mais nous y reviendrons ultérieurement...

Il serait possible d'ajouter ces fonctions spéciales aux broches I/O normales puisqu'elles n'ont pas de broches propres sur le microcontrôleur. Vous pouvez sélectionner

la fonction qui est active. Puisque presque tout est fermement câblé sur le RP6, cela ne fait pas tellement de sens de modifier l'affectation standard.

Le MEGA32 comporte encore beaucoup d'autres fonctions intégrées mais nous ne pouvons pas toutes les décrire ici. Celui qui veut en savoir davantage peut consulter le feuille technique du fabricant (sur le CD).

Il faut évidemment maîtriser la langue anglaise. Presque toutes les documentations de fabricants tiers n'existent qu'en anglais. C'est normal car la langue anglaise est le standard en électronique et en informatique.

2.1.1. Chargeur d'amorçage

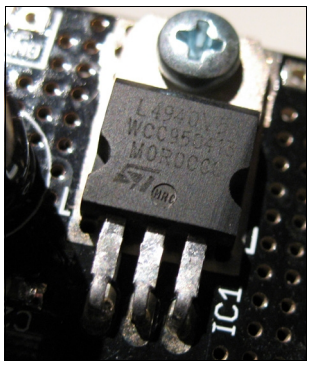
Le chargeur d'amorçage se trouve dans une zone spécifique du microcontrôleur. Il s'agit d'un petit programme qui permet de charger de nouveaux programmes dans la mémoire par le biais d'une interface série du microcontrôleur. Le chargeur d'amorçage communique avec un PC hôte par le logiciel fourni RP6Loader. Cela évite un dispositif de programmation qui serait normalement nécessaire. Un seul petit inconvénient: Sur les 32Ko de mémoire du MEGA32, vous ne pouvez utiliser que 30Ko! Mais ce n'est pas grave, cela suffit même pour des programmes très complexes (A titre de comparaison: le petit robot ASURO d'AREXX ne dispose que de 7Ko de mémoire libre et cela suffit amplement).

2.2. Alimentation

Un robot a besoin d'énergie que le RP6 transporte avec lui dans 6 accus. La durée est limitée par la capacité des accus car même si l'électronique ne consomme que peu d'énergie, les moteurs consomment beaucoup en fonction de la charge.

Afin que les accus durent le plus longtemps possibles et que le robot ne s'arrête pas constamment, il faudrait lui offrir des accumulateurs un peu plus gros d'une capacité d'env. 2500mAh. De plus petits accus de 2000mAh ou plus fonctionnent aussi mais avec de bons accus, la durée de fonctionnement peut atteindre 3 à 6 heures selon l'activité des moteurs, leur charge et la qualité des accus. Les 6 accus qui sont nécessaires possèdent une tension nominale totale de 6x 1,2V = 7,2V. Dans le schéma fonctionnel c'est représenté par « UB » (= »U-Battery », U étant la lettre représentant la tension). On parle de tension « nominale » car elle change fortement avec le temps. A vide et entièrement chargés, les accus peuvent délivrer jusqu'à 8,5V! Cette tension diminue pendant la décharge et varie avec la charge (moteur sous tension ou arrêté, rapide, lent, etc. - la variation de tension dépend aussi de la qualité des accumulateurs utilisés. La résistance interne est ici la valeur critique).

Cela pose bien sûr un problème dans la mesure des détecteurs et autres, mais ce n'est pas le plus grave. Un grand nombre des composants utilisés tel que le microcontrôleur, sont conçus pour 5V ou moins de tension d'alimentation et seraient détruits par des tensions aussi fortes. Il faut donc baisser la tension des accus à une valeur définie et la stabiliser.



Cela se fait par un régulateur de tension 5V qui délivre un courant maximum de 1,5A (voir photo). Cependant, il dégagerait beaucoup de chaleur à 1,5A. C'est pourquoi la platine comporte une grande surface de refroidissement sur laquelle le régulateur est vissé. Au-delà de 1A, il vaut mieux limiter l'utilisation du régulateur à quelques secondes malgré le refroidissement si vous n'ajoutez pas un dissipateur thermique plus grand.

Il est recommandé de ne pas dépasser une charge continue de 800mA. A une telle charge et en conjonction avec les moteurs, les accus se videraient très vite. Pendant un fonctionnement normal sans extension, le robot ne consomme pas plus que 40mA (sauf si l'IRCOMM est en transmission). Cela ne pose donc aucun problème pour le régulateur et il reste encore suffisamment de réserves pour les modules d'extension qui ne consomment généralement pas plus que 50mA à condition qu'aucun moteur, grande LED, etc. ne soient branchés.

2.3. Détecteurs

Nous avons déjà abordé la plupart des détecteurs du robot dans d'autres chapitres mais nous allons les examiner d'un peu plus près maintenant.

Dans le schéma fonctionnel, certains détecteurs ne figurent pas dans les zones bleues « Sensors » parce qu'ils appartiennent plutôt à d'autres domaines. Malgré tout, les encodeurs, capteurs de courant moteur et capteurs de tension des batteries font partie des détecteurs et sont donc décrits dans ce chapitre

2.3.1. Capteur de courant batterie (Voltage Sensor)

Ce « détecteur » n'est en fait qu'un simple diviseur de tension composé de deux résistances. Nous partons du principe que les accumulateurs peuvent délivrer au maximum 10V. 6 accus NiMH resteront toujours en dessous de cette tension. La tension de référence du CAN, donc la tension à laquelle il compare la tension mesurée, s'élève à 5V. Etant donné que la tension de fonctionnement s'élève également à 5V, il ne faut pas la dépasser. Nous devons donc baisser la tension mesurée de moitié. Ceci se fait par un diviseur de tension composé de 2 résistances, qui adapte la tension à la plage de mesure du CAN.

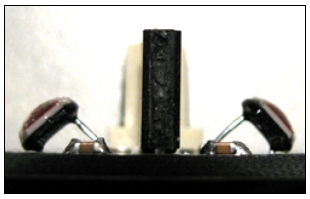
Le CAN présente une résolution de 10 octets (plage de valeur 0 à 1023) ce qui donne une résolution de $\frac{10\text{V}}{1024} = 9.765625\text{mV}$. Une valeur mesurée de 512 correspond donc à 5V et 1023 seraient environ 10V. Normalement, il ne faudra jamais atteindre ses valeurs limites!

La précision n'est pas particulièrement élevée car les résistances ne sont nullement des résistances de précision. Des écarts de quelques pour-cent vers le haut ou le bas sont possibles. Même la tension de référence de 5V n'est pas particulièrement précise et peut varier légèrement sous une charge normale. Cela ne pose pas de problème dans le cas présent car il nous suffit d'avoir une valeur approximative pour savoir

quand les accus s'approchent de l'épuisement. Pour mesurer la tension avec précision, il faut un multimètre pour détecter l'erreur de mesure et la compenser dans le logiciel.

Si l'on ne recherche pas la précision, il est possible d'évaluer approximativement la tension directement à partir des valeurs du CAN grâce au rapport de conversion favorable: 720 correspondent grossièrement à 7,2V, 700 à env. 7V et 650 seraient à peu près 6,5V. A une valeur constante inférieure à 560, on peut déduire que les accus sont presque vides.

2.3.2. Détecteurs de lumière (LDR=Résistances photo-dépendantes)



A l'avant de la petite platine de détection du robot se trouvent deux LDR (="Light Dependant Resistors" donc résistances photo-dépendantes) qui sont orientées vers la droite et la gauche. Entre les deux détecteurs se trouve une petite « paroi de séparation » afin que la lumière provenant d'une direction ne puisse atteindre qu'un seul détecteur. Ensemble avec une résistance normale chacun, ils forment un autre diviseur de tension, comme le capteur de courant, à la différence qu'il s'agit ici de mesurer la lumière. La tension de fonctionnement de 5V est également divisée mais ici, une des résistances est variable. Le rapport de division change donc en fonction de l'intensité de la lumière et une tension dépendante de la luminosité est envoyée dans un des canaux du convertisseur analogique/numérique.

La différence de tension entre les deux détecteurs permet de savoir où se trouve la source lumineuse la plus forte en face du robot: à gauche, à droite ou au milieu.

Avec un programme approprié le robot pourrait suivre le rayon lumineux d'une lampe torche puissante dans une pièce *sombre* ou chercher l'emplacement le plus lumineux dans une pièce. Cela fonctionne très bien avec un projecteur halogène à main assez puissant: Si vous dirigez le rayon lumineux vers le sol, le robot peut suivre le point lumineux.

Evidemment cela fonctionne aussi dans l'autre sens: le robot pourrait essayer de se cacher de la lumière...

Si vous montez encore une ou deux LDR à l'arrière du robot, vous pourriez affiner la recherche de la source lumineuse, car le robot a du mal à distinguer si la source lumineuse se trouve devant ou derrière lui. Deux des canaux du CAN sont encore libres...

2.3.3. Système Anti Collision (ACS)

Le détecteur le plus complexe d'un point de vue logiciel est l'ACS - le système anti-collision (en anglais: Anti-Collision System). Il est composé d'un récepteur IR (voir photo) et de deux LED IR montées à gauche et à droite à l'avant de la platine de détection. Les LED IR sont directement commandées par le microcontrôleur. Vous pouvez librement modifier et améliorer les routines de pilotage. Sur le modèle précédent, il fallait pour cela un autre contrôleur dont le programme ne pouvait pas être modifié par l'utilisateur...

Les LED IR envoient des impulsions infrarouge courtes modulées à 36kHz auxquelles le récepteur IR réagit. Si les impulsions IR sont réfléchies par un objet devant le robot et détectées par le récepteur IR, le microcontrôleur peut réagir et p.ex. engager une manœuvre de contournement. Afin d'éviter une réaction trop rapide de l'ACS due à des interférences, le logiciel attend un certain nombre d'impulsions sur une durée déterminée. Il exécute également une synchronisation avec la réception RC5 afin de ne pas réagir aux signaux RC5 de télécommandes TV et Hifi. Cette garantie n'existe cependant pas pour d'autres codes et il est possible que l'ACS détecte des obstacles là où il n'y en a pas!

Puisqu'il y a une LED gauche et une LED droite, l'ACS arrive à différencier grossièrement si l'objet se trouve à droite, à gauche ou au milieu devant le robot.

Par ailleurs, vous pouvez régler l'intensité du courant des impulsions envoyées aux deux LED IR en 3 niveaux. Même au niveau le plus élevé, l'ACS n'est pas toujours fiable car la nature de la surface de l'objet joue un rôle important. Un objet noir réfléchit la lumière IR beaucoup plus mal qu'un objet blanc et un objet angulaire et réfléchissant renverrait la lumière IR surtout dans un certaine direction. La portée dépend donc toujours de l'objet en question. C'est une faiblesse commune à quasiment tous les détecteurs IR (en tout cas dans notre catégorie de prix).

En dépit de cela, la plupart des obstacles sont correctement détectés et contournés. Si jamais cela ne fonctionne pas, il y a encore le pare-chocs avec les détecteurs tactiles et si ceux-là ne sont pas non plus activés , le robot peut encore reconnaître si les moteurs bloquent ou non à l'aide des capteurs de courant moteur ou des encodeurs (voir ci-dessous).

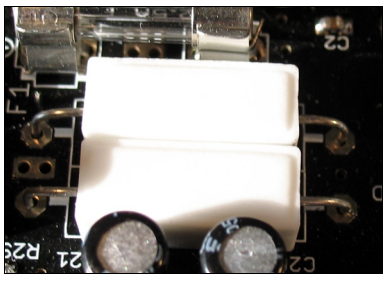
Si cela ne vous suffit pas, vous pouvez encore installer des détecteurs à ultrasons...

2.3.4. Détecteurs de Pare-chocs (Bumper)

Deux microcommutateurs avec un long levier chacun sont montés sur le devant du robot sur une platine à part. Cela protège les LED IR sur la platine qui ne se déforment plus si facilement lorsque le robot heurte un obstacle. Les deux commutateurs permettent au microcontrôleur d'enregistrer le choc et de reculer, tourner un peu et continuer sa route. Les commutateurs sont connectés sur deux des ports qui sont déjà reliés aux LED et n'occupent ainsi pas de port supplémentaire. C'est pourquoi les LED s'allument toujours lorsqu'un commutateur est appuyé. Comme cela arrive relativement rarement, cela ne gêne en rien.

Il est également possible de démonter le pare-chocs et le remplacer par un dispositif de tir ou de ramassage de balles ou similaire.

2.3.5. Capteurs de courant moteur (Current sensing)



Les deux circuits du courant moteur contiennent deux résistances. La loi d'Ohm U=R x I démontre que la tension qui diminue à une certaine résistance, se comporte proportionnellement au courant qui traverse celle-ci!

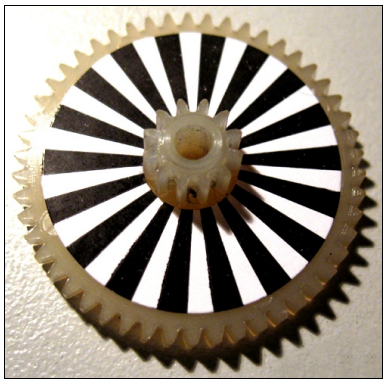
Afin que les chutes de tension ne soient pas trop impor-

tantes aux résistances, il faut choisir de petites résistances. Dans notre cas, elles ont une valeur de 0,1 Ohms.

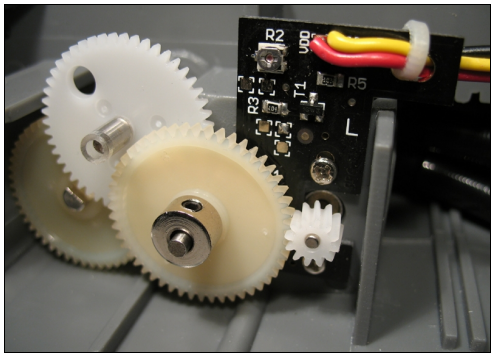
La perte de tension est donc très faible (0,1V pour un courant de 1A) et il faut l'augmenter avant de pouvoir la mesurer avec le CAN. Pour cela, on utilise un amplificateur opérationnel (AOP) dans chaque canal moteur du circuit du RP6. La plage de mesure va jusqu'à 1,8A. A 1,8A, la résistance provoque une chute de tension d'env. 0,18V et la tension à la sortie de l'AOP s'élève à env. 4V. L'AOP utilisé ne peut pas délivrer davantage avec une tension de fonctionnement de 5V.

Les résistances de puissance présentent une tolérance de 10%, les résistances de l'AOP 5%. Tout cela n'est donc pas très précis (des imprécisions allant jusqu'à 270mA sont possibles si les détecteurs n'ont pas été calibrés!). Toutefois, une valeur approximative nous suffit pour savoir si les moteurs sont fortement ou faiblement sollicités. Ainsi le robot arrive à reconnaître facilement des moteurs ou encodeurs bloqués ou défaillants. Les moteurs DC nécessitent plus de courant lorsqu'ils sont fortement sollicités (couple) et le courant monte fortement lorsque les moteurs sont bloqués. C'est ce que le logiciel du robot utilise pour l'arrêt d'urgence: si les moteurs étaient alimentés en permanence par un courant élevé, ils deviendraient très chauds et pourraient être endommagés. Cela permet également de détecter une défaillance des encodeurs quel qu'en soit la raison, puisque le nombre de tours mesuré serait 0. Si on fait tourner les moteurs à pleine puissance et le courant reste quand-même faible (donc les chenilles ne sont pas bloquées), on peut en déduire que soit l'encodeur, soit les moteurs ont été défaillants (ou que l'encodeur et les capteurs de courant ne fonctionnent pas... Cela arrive p.ex. quand on a oublié de les mettre sous tension avec le logiciel).

2.3.6. Encodeur (Encoder)



Le fonctionnement des encodeurs est très différent de celui des détecteurs mentionnés en dernier. Ils sont montés sur les engrenages des moteurs pour mesurer le nombre de tours. Il s'agit ici de barrières rétro-réfléchissantes qui sont dirigées vers des disques encodeur comportant chacun 18 segments blancs et 18 segments noirs, donc un total de 36 segments (voir photo). Ces encodeurs ont été collés sur une roue dentée de chaque engrenage. Lorsqu'il tourne, les segments passent devant la barrière rétro-réfléchissante. Les segments blancs reflètent la lumière infrarouge, les segments noirs ne la reflètent que très peu. Tout comme les autres détecteurs, les encodeurs génèrent un signal analogique mais il est interprété d'une manière numérique. Tout d'abord le signal est amplifié et ensuite il est transformé par un trigger Schmitt en un signal rectangulaire. Les flancs de ce signal, donc les changements de 5V à 0V et inversement, déclenchent à chaque fois une interruption qui est comptée par le logiciel. Cela permet de mesurer la distance parcourue et de déterminer, à l'aide d'un timer pour la mesure du temps, le nombre de tours et donc la vitesse. La détermination du nombre de tours est la principale application des encodeurs car seuls les encodeurs permettent de régler le nombre de tours sur la valeur cible. Sans ce réglage, le nombre de tours serait tributaire de la tension des accus, de la charge des moteurs, etc. La résolution élevée permet de régler même des vitesses faibles avec une précision relativement haute.



Chacune des deux roues dentées centrales de l'engrenage possède 50 dents sur la grande et 12 sur la petite roue dentée (voir photo). Les encodeurs se trouvent sur la roue dentée à côté du moteur. Ainsi il faut calculer:

$$\frac{50}{12}\cdot\frac{50}{12}=17\frac{13}{36} \quad ; \quad 17\frac{13}{36}\cdot\mathbf{36}=625$$

C'est pourquoi les encodeurs ont leurs 36 segments car cela donne un chiffre rond sans décimales. Les encodeurs génèrent donc 625 flancs par rotation de roue, chaque flanc correspondant à un segment.

Pour un diamètre de roue d'env. 50mm chenille comprise, cela donne mathématiquement une circonférence d'env. 157mm ce qui correspond à 0,2512mm par incrément des encodeurs. Puisque les chenilles s'enfoncent presque toujours un peu dans le sol, on peut partir sur la base de 0,25mm par incrément, souvent même un peu moins p.ex. seulement 0,24 ou 0,23mm. Il faut déterminer ceci en mesurant des parcours de test tel que c'est décrit en annexe. Cependant en raison du glissement des roues (ou plutôt des chenilles) et autres facteurs similaires, la précision n'est toujours pas très élevée, notamment dans le cas de rotations sur place. Lorsque le robot avance tout droit, la déviation est faible mais lorsqu'il tourne sur place, elle atteint vite des valeurs importantes. Il faut effectuer d'autres tests pour déterminer les écarts et les inclure dans les calculs. Malheureusement c'est le cas de tous les entraînements par chenilles, même sur des robots beaucoup plus chers et sophistiqués. D'un autre côté, un robot entraîné par chenilles est beaucoup plus mobile par rapport aux robots équipés d'un différentiel normal avec deux roues motrices et une roue d'appui. Il surmonte généralement sans problème de petits obstacles et des rampes. C'est là que les encodeurs trouvent tout leur intérêt car la vitesse se règle facilement quelque soit la surface et la charge du moteur.

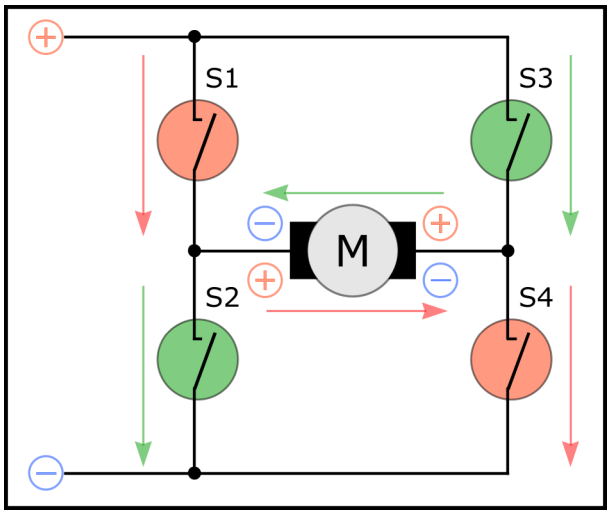
A 50 segments par seconde, la vitesse se situe à 1,25cm/sec., si l'on se base sur un incrément de 0,25mm. Env. 50 segments par seconde est aussi la vitesse la plus basse réglable (cela varie un peu d'un robot à l'autre). A 1200 segments par seconde, ce serait le maximum possible de 30 cm/sec. (à une résolution de 0,25mm. A 0,23 ce seraient 27,6 cm/sec.). La bibliothèque des fonctions limite cela à 1000 flancs par seconde. La vitesse maximale dépend de l'état de charge des accus ce qui signifie que 30cm/sec ne pourraient pas être tenus très longtemps. En plus, la durée de vie des engrenages et moteurs augmente à vitesse lente.

Lorsque le robot a compté 4000 segments, il n'a avancé que d'env. 1m à condition que la résolution soit à 0,25mm. Sans calibrage, les écarts seront plus ou moins importants. Si la précision n'est pas tellement importante, ce n'est pas la peine de calibrer et vous vous basez simplement sur 0,25mm ou mieux encore sur 0,24mm.

L'idéal est de ne pas être obligé de s'appuyer sur les données des encodeurs pour les mesures de parcours et d'angle mais de disposer de systèmes externes tels que des balises infrarouge ou une boussole électronique précise.

2.4. Système d'entraînement

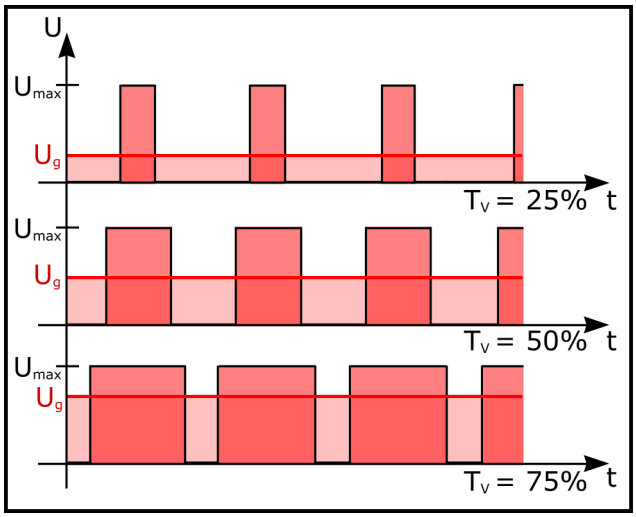
L'entraînement du RP6 se compose de deux moteurs DC avec engrenage qui entraîne les deux chenilles (voir schéma ci-dessus).



Les moteurs peuvent présenter une consommation relativement élevée selon la charge et ne peuvent pas être commandés directement par le microcontrôleur. Pour ce faire, il faut installer un pont en H par moteur. Le principe de base est représenté ci-contre. Un pont en H est composé de 4 « commutateurs » qui sont disposés autour d'un moteur sous forme de H. Supposons que tous les commutateurs sont éteints. Si l'on met sous tension S1 et S4 (rouge), une tension est appliquée au moteur et il tourne p.ex. vers la droite. Si nous coupons maintenant S1 et S4 et mettons sous tension S2 et S3 (vert), la tension moteur change de polarité et il tourne dans l'autre sens, donc vers la gauche. Il faut veiller à ne pas mettre sous tension simultanément S1 et S2 ou S3 et S4 car on provoquerait un court-circuit qui risquerait de détruire les « commutateurs »!

Bien évidemment, nous n'utilisons pas de commutateurs mécaniques sur le RP6 mais des MOSFET. Ils établissent le contact lorsqu'une tension suffisamment élevée se présente à la sortie. Les commutations peuvent se produire à très grande vitesse, c'est-à-dire plusieurs Kilohertz pour notre application.

Cela permet de déterminer déjà le sens de la rotation. Et comment faut-il faire pour accélérer ou ralentir le moteur? Plus la tension appliquée est élevée, plus un moteur électrique tourne vite. Donc, le nombre de tours est réglable par le biais de la tension – et c'est exactement à cela que nous pouvons utiliser le pont en H!



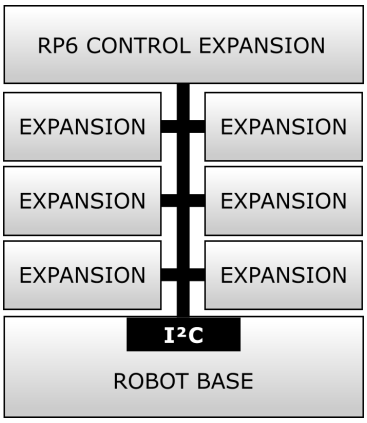
L'illustration montre le principe selon lequel nous allons procéder. Nous générons un signal rectangulaire d'une fréquence *fixe* sur le quel nous modifions le rapport cyclique. Le rapport cyclique est le ratio entre la durée du phénomène sur une période et la durée de cette même période.

Le moteur est donc alimenté par une tension continue moyenne plus basse qui correspond au rapport cyclique. Dans l'illustration c'est représenté par une ligne rouge (Ug) et les zones rouges. Ainsi, si une tension des accus de 7V est appliquée aux moteurs et ceux-ci sont pilotés avec un rapport cyclique de 50%, la tension moyenne se situerait à env. la moitié, donc à 3,5V. Ce n'est pas tout à fait vrai dans

la réalité mais cela permet de se faire une meilleure idée.

Le système d'entraînement est relativement fort en raison du rapport élevé (env. 1:72) ce qui permet au RP6 de porter des charges bien plus lourdes que le petit robot ASURO. Toutefois, plus le poids est élevé, plus le besoin en énergie est important et plus vite se déchargeront les accus....

Comparé à une voiture de course télécommandée, le RP6 semble lent – ce qui est vrai - mais c'est fait exprès. Le robot est commandé par un microcontrôleur, et si le programmeur commet une erreur de programmation, il ne serait pas judicieux de foncer dans un mur à 10km/h! A la vitesse plus lente du RP6, cela n'arrive pas si facilement et cela laisse en plus le temps au détecteurs de chercher des obstacles dans leur environnement. Par ailleurs, il y a aussi l'avantage d'une charge plus élevée et le réglage de vitesse beaucoup plus précis. Le RP6 peut rouler très lentement à une vitesse constante!



2.5. Système d'Extension

Une des qualités les plus importantes du RP6 est son système d'extension. Vous pouvez le faire évoluer à votre guise. Pour des raisons de coûts, le système de base ne comprend que relativement peu de détecteurs. Il comporte plus que d'autres robots de sa catégorie de prix mais le vrai plaisir du robot commence lorsqu'il y a plus de détecteurs. L'ACS ne sait que vaguement si un obstacle se trouve devant le robot. Avec des détecteurs à ultrasons ou des détecteurs IR supplémentaires de meilleure qualité, vous pourriez déterminer la distance et effectuer de meilleures manœuvres de contournement!

Outre des détecteurs, il serait également judicieux de monter des contrôleurs supplémentaires afin de mieux répartir les tâches tel que le RP6 CONTROL M32 avec un autre microcontrôleur MEGA32.

Le système d'extension doit évidemment être en mesure de relier ensemble un grand nombre de modules d'extension (en anglais Expansion Modules, voir illustration), se contenter de peu de lignes de signaux et offrir une vitesse suffisamment élevée.

2.5.1. Le Bus I²C

Le bus I²C (--> IIC = „Inter IC Bus“), remplit ces conditions. Il n'a besoin que de 2 lignes de signaux, arrive à relier ensemble jusqu'à 127 participants et offre une vitesse de transmission maximale de 400kb/sec.

Le bus I²C a été développé par Philips Semiconductors dans les années 1980 et '90 et est devenu depuis un système de bus très répandu. Le bus I²C est largement utilisé dans des appareils électroniques audio-visuels tels que des magnétoscopes et téléviseurs mais également dans des appareils et systèmes industriels. Dans les PC et PC portables, il est utilisé sous forme de SMBus qui sert, entre autre, à la gestion du refroidissement et à la surveillance de la température. Il est également employé dans de nombreux robots et il n'est donc pas étonnant qu'il existe divers modules de détec-

tion tels que des détecteurs à ultrasons, des boussoles électroniques, des capteurs de température et autres qui possèdent cette interface.

Il s'agit d'un bus orienté maître-esclave. Un ou plusieurs maîtres règlent le trafic de données entre et vers 127 esclaves au maximum. Nous allons cependant nous contenter de décrire l'utilisation du bus avec un seul maître, même si le bus était multi-maîtres pour ne pas compliquer inutilement les choses.

Les deux lignes de signaux sont désignées par SDA et SCL. SDA est l'abréviation de « Serial Data » et SCL de « Serial Clock », donc un signal de données et un signal d'horloge. Le SDA est bidirectionnel ce qui signifie qu'il accepte aussi bien des données maître qu'esclave. L'horloge du SCL est toujours générée par le maître.

Le bus transmet les bits de données toujours synchrones au signal d'horloge qui est imposé par le maître. Le niveau du SDA ne doit changer que pendant que le SCL est low (sauf à la condition Start et Stop, voir ci-dessous). La vitesse de transmission peut varier entre 0 et 400kb/sec. au cours d'une transmission.

START	ADR	W	ACK	DATA	ACK	···	DATA	ACK	STOP
START	ADR	R	ACK	DATA	ACK	···	DATA	ACK	STOP

Deux processus de transmission sont représentés ci-dessus. La première ligne montre une transmission de données du maître vers un esclave. Les cases blanches indiquent le trafic de données du maître vers l'esclave, les cases foncées sont les réponses de l'esclave.

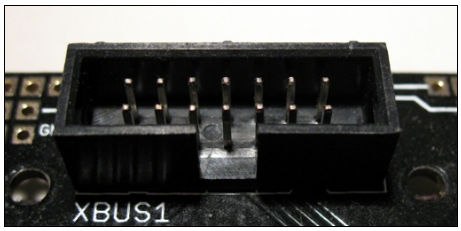
Chaque transmission doit être introduite par un StartBit et terminée par un StopBit. La condition de départ est remplie lorsque le SDA passe à 0 alors que SCL reste à 1. L'inverse s'applique à la fin de la communication.

Après la condition de départ, l'adresse de 7bit de l'esclave avec qui le maître veut communiquer, est envoyée, suivie d'un byte (octet) qui détermine si les données doivent être écrites ou lues. L'esclave le confirme avec un *acknowledge* « ACK » (confirmation). Ensuite, un nombre d'octets est envoyé qui sont chacun confirmés par un acknowledge ACK de la part de l'esclave. La communication se termine par la condition d'arrêt.

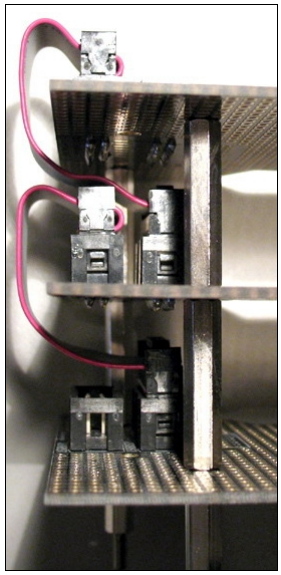
Ce n'était qu'une description très succincte du bus I²C. Vous trouverez plus de détails dans les caractéristiques techniques du bus I²C de Philips. Dans la fiche technique du MEGA32, vous trouverez également quelques informations!

Les exemples de programme vous montrent l'application. Les fonctions de pilotage du bus I²C existent déjà dans la bibliothèque de fonctions du RP6. Ce n'est donc pas la peine de réfléchir au protocole lui-même mais c'est toujours utile d'avoir quelques notions.

2.5.2. Connecteurs d'extension



Quatre connecteurs d'extension sont disposés sur la carte-mère du robot dont deux sont repérés par « XBUS1 » et « XBUS2 ». XBUS est une abréviation pour « eXpansion BUS », donc bus d'extension. XBUS1 et XBUS2 sont entièrement connectés ensemble et disposés de manière symétrique sur la carte-mère. C'est pourquoi il est possible de monter les modules d'extension aussi bien à l'avant qu'à l'arrière du robot. Chaque module d'extension comporte sur son côté deux connecteurs XBUS. Un câble en nappe de 14 points permet de relier les modules d'extension avec le carte-mère et entre eux. C'est pourquoi il y a deux connecteurs identiques et reliés ensemble sur chaque module d'extension. La fiche extérieure doit être utilisée pour la connexion vers le bas tandis que la fiche intérieure est conçue pour la connexion vers le haut. C'est ainsi que vous pouvez (en théorie) empiler un nombre illimité de modules (voir illustration: les trois modules d'extension à grille perforée RP6 peuvent servir pour vos propres circuits).



Les connecteurs des XBUS possèdent une alimentation, un bus I²C décrit ci-dessus, des signaux Master Reset et d'interruption.

L'alimentation est assurée par deux tensions qui existent aux connecteurs: évidemment les 5V stabilisés du régulateur de tension mais aussi la tension des accus. Cette dernière change avec le temps et varie en fonction de la sollicitation. Elle se situe normalement dans la zone de 5,5V (accus vides) jusqu'à env. 8,5V (accus fraîchement chargés, variable en fonction du fabricant) mais peut également être en dehors de cette plage en fonction de la sollicitation, de la nature et de l'état de charge des accus.

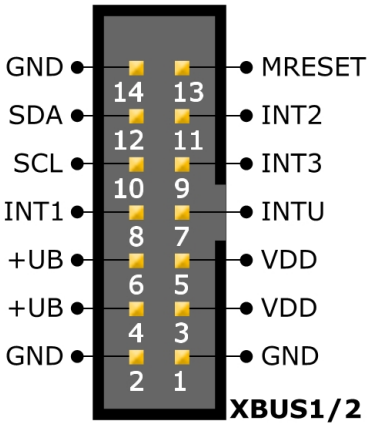
Le signal Master Reset est important pour remettre à zéro tous les microcontrôleurs en appuyant sur la touche Start/Stop ou lors de la programmation. Les chargeurs d'amorçage dans les contrôleurs démarrent d'ailleurs leurs programmes par un low pulse (high-low-high) sur la ligne SDA. Ainsi les programmes sur tous les contrôleurs (AVR) démarrent en même temps après avoir appuyé sur la touche Start/Stop ou lancé le programme par le logiciel d'amorçage (pour le démarrage, le chargeur d'amorçage ne génère non seulement un low pulse mais un General Call I²C complet avec 0 comme byte de donnée).

Les lignes d'interruption sont utilisées par certains modules pour faire savoir au microcontrôleur maître par un signal d'interruption externe, que de nouvelles données sont disponibles ou qu'une tâche précise a été effectuée et qu'il attend de nouvelles instructions. Si ces lignes n'existaient pas, le microcontrôleur maître (sur certains modules d'extension) devrait interroger en permanence les esclaves s'il y a de nouvelles données. Ce serait également possible mais avec des lignes d'interruption supplémentaires, on économise de la charge de bus et du temps de calcul. Puisqu'il n'existe que 3 lignes d'interruption et une ligne librement attribuable par l'utilisateur, des modules identiques comme des détecteurs à ultrasons devront, le cas échéant, se partager une

des lignes ou les interroger toutes.

Les deux autres connecteurs d'extension repérés par « USRBUS1 » et « USRBUS2 » sur la carte-mère ne sont PAS reliés entre eux. Les différentes lignes sont exécutées sur des cosses à souder sur tous les modules d'extension afin de pouvoir appliquer ses propres signaux à ces connecteurs. D'où leur nom « USRBUS » ce qui est une abréviation de « User-Bus » donc « Bus d'utilisateur ». Vous pouvez utiliser ces fiches d'extension à 14 contacts pour tout ce que vous voulez: votre propre système de bus, des conducteurs d'alimentation (Attention: Limitez la charge. Les pistes ne sont pas particulièrement larges), etc.. Vous pouvez également relier deux modules d'extension ensemble sans les connecter aux autres extensions. C'est utile pour des circuits ou détecteurs plus sophistiqués qui ne s'adapteraient pas sur un module d'extension. En plus, le câblage est plus propre.

Evidemment, vous ne pouvez pas brancher un nombre illimité de modules d'extension. Tout au plus à 6 modules empilés à l'avant ou à l'arrière, la tenue de route du RP6 serait compromise. Par ailleurs, il faut également considérer la consommation d'énergie! Si elle est trop élevée, les accus se déchargeront vite. En règle générale, on peut dire que le RP6 peut accepter jusqu'à 8 modules d'extension, 4 à l'avant et 4 à l'arrière.



Le schéma montre l'affectation des contacts des deux fiches d'extension. La broche 1 est toujours placée sur le côté de la carte-mère où se trouve le repérage blanc XBUS1 ou XBUS2, ou bien elle est repérée par un « 1 » à côté du contact.

+UB est la tension des accus, VDD les +5V de tension de fonctionnement, GND désigne le négatif (GND = Ground, donc la masse), MRESET est le signal de reset maître, INTx sont les lignes d'interruption, SCL la ligne d'horloge et SDA la ligne de données du bus I²C.

Vous devrez souder vous-même sur les contacts de l'USRBUS tout ce qu'il vous faut d'autre.

Conseil important: **Ne chargez pas les lignes d'alimentation VDD et +UB des contacts d'extension au-delà de 1A maximum (s'applique aux 2 broches REUNIES! Donc, respectivement les broches 4+6 (+UB) et 3+5 (VDD))!**

3. Mise en Service

Avant de mettre en service le RP6 ou ses accessoires, respectez ces quelques consignes de sécurité, *surtout* si ce sont des enfants qui utilisent le RP6!

Lire très attentivement le chapitre suivant !

3.1. Consignes de Sécurité

En raison de sa forme ouverte, le RP6 présente quelques angles pointus et arêtes coupantes. Il **ne** doit donc **pas** être utilisé comme jouet pour un enfant de moins de 8 ans. Surveillez les enfants en bas âge qui se trouvent dans la pièce lorsque le robot est en service et mettez-les en garde contre les dangers potentiels!
Ne pas utiliser le robot lorsque des animaux domestiques tel qu'un hamster, se trouvent en liberté dans la pièce afin qu'ils ne puissent pas être blessés. (*Inversement*, le robot pourrait être endommagé par un animal plus grand tel qu'un chien ou un chat).

Veillez à ne pas mettre vos doigts entre la chenille et les roues. Les moteurs sont assez puissants et vous risquez de vous blesser. Ne mettez pas vos doigts entre la platine et la chenille!

Attention: Même avec le logiciel standard, les moteurs peuvent augmenter automatiquement leur puissance! Selon la programmation du robot, les moteurs peuvent démarrer d'une manière inattendue et le robot peut effectuer des manœuvres de conduite imprévisibles! **Ne jamais utiliser le robot sans surveillance!**

3.1.1. Décharges statiques et Courts-circuits

Un grand nombre de **composants et de pistes non isolés** se trouvent sur la surface de la carte-mère, l'interface USB et tous les modules d'extension. Ne provoquez pas de courts-circuits par des objets métalliques ou des outils posés par inadvertance.

La tension de fonctionnement est très basse et donc sans danger pour l'homme. Par contre, un grand nombre de composants présente une grande sensibilité électrostatique et **vous ne devez donc pas toucher ces composants sur les platines** si vous pouvez l'éviter. Lorsque l'air est très sec (et notamment si vous portez des vêtements en synthétique), le corps humain peut se charger en électricité statique. Ceci s'applique également au robot – où la nature du revêtement au sol est déterminante. Au contact avec des objets conducteurs, cette charge se décharge par une petite étincelle. De telles décharges sur des composants électroniques peuvent les détruire. Avant de manipuler le robot ou ses accessoires, vous devez toucher un grand objet relié à la terre (p.ex. le boîtier en métal d'un PC, un conduit d'eau ou de chauffage), afin de décharger l'électricité éventuelle. Une décharge du robot même contre des objets mis à la terre est sans danger mais peut cependant entraîner des bugs dans le programme ou un fonctionnement incontrôlé du robot.

Toutes les connexions électriques à partir ou vers le matériel doivent être effectuées

avant le branchement de la tension d'alimentation.
Sinon le branchement ou le retrait de câbles de connexion/modules ou l'établissement ou le retrait de liaisons pourrait entraîner la destruction partielle du robot ou de ses accessoires.

3.1.2. Environnement du robot

Ne pas utiliser le robot sur une table ou un endroit d'où il risque de tomber. Ne sous-estimez pas la capacité d'ascension des chenilles! C'est facile pour le robot de surmonter de petits obstacles et de pousser des objets légers. Avant la mise en service du robot, vous devez sécuriser ou enlever tous les contenants de liquides tels que tasses de café, bouteilles ou vases dans le rayon d'action du robot.

La cuvette fermée constitue une petite protection pour la mécanique contre les facteurs environnants les plus agressifs mais elle n'est hermétique ni à l'eau, ni à la poussière. L'électronique est également assez exposée. N'utilisez le robot qu'à l'intérieur dans un environnement propre et sec. Des saletés, des corps étrangers et l'humidité risquent d'endommager ou de détruire la mécanique et l'électronique. La température ambiante doit rester dans une fourchette de 0° à 40°C.

Pendant le fonctionnement, de petites étincelles peuvent se former à l'intérieur des moteurs électriques. Il est donc **interdit** d'utiliser le robot dans un environnement contenant des liquides, gaz ou poussières inflammables ou explosifs!

En cas de non-utilisation prolongée, ne pas entreposer le robot dans des pièces à forte humidité relative. Par ailleurs, vous devez retirer les accus afin d'éviter qu'ils ne fuient!

3.1.3. Tension d'alimentation

L'alimentation du robot se fait par une tension continue de 7,2V qui est délivrée par 6 accus NiMH. La tension d'alimentation maximale s'élève à 10V et ne devra jamais être dépassée. Pour recharger les accumulateurs, n'utilisez que des chargeurs homologués conformément aux règles de sécurité en vigueur!
A titre exceptionnel, le robot peut aussi être alimenté par 6 piles alcalines-manganèse de haute qualité. Etant donné que les piles normales se déchargeront très rapidement et provoqueront de ce fait des frais élevés et une pollution importante de l'environnement, il est recommandé d'utiliser des accumulateurs. Par ailleurs, les accus supportent une intensité de courant plus élevée et se rechargent aisément dans le robot.
Respectez les consignes de sécurité et de recyclage pour piles et accumulateurs en annexe!

N'effectuez des modifications sur le robot que si vous savez parfaitement ce que vous faites. Vous risquez d'endommager le robot définitivement et de vous mettre en danger vous-même et les autres.
(Ne citons en exemple qu'un incendie dans la pièce provoqué par la surcharge de certains composants ...)!

3.2. Installation du logiciel

Passons maintenant à l'installation du logiciel. Un logiciel correctement installé est la condition sine-qua-non pour tous les chapitres suivants.

Vous devez avoir des droits d'administrateur. Commencez par vous identifier comme administrateur le cas échéant!

Nous vous conseillons de lire attentivement la totalité du chapitre et de suivre ensuite pas à pas les instructions d'installation.

De bonnes connaissances de base dans l'utilisation d'ordinateurs sous Windows ou Linux ainsi que la maîtrise de programmes courants tels que des gestionnaires de fichiers, le navigateur Internet, l'éditeur de texte, des logiciels de compression (WinZip, WinRAR, unzip et autres) et éventuellement Linux-Shell, etc. sont une condition préalable! Si vos connaissances sont très limitées, vous devez d'abord vous familiariser avec l'informatique *avant* de mettre en service le RP6! Ce manuel n'explique pas le fonctionnement d'un ordinateur car cela dépasserait largement son cadre. Il ne s'agit que du RP6, de sa programmation et du logiciel nécessaire.

3.2.1. Le CD-ROM RP6

Insérez maintenant le CD-ROM dans votre lecteur. Le menu du CD devrait s'afficher rapidement sous Windows. Si ce n'est pas le cas, vous pouvez ouvrir le fichier « start.htm » dans le répertoire principal du CD à l'aide d'un gestionnaire de fichiers dans un navigateur Internet tel que Firefox. Les fichiers d'installation pour Firefox se trouvent d'ailleurs sur le CD dans le classeur

```
<Lecteur CD-ROM>:\Software\Firefox
```

au cas où vous n'auriez pas encore installé un navigateur Internet à jour (il devrait s'agir au minimum de Firefox 1.x ou Internet Explorer 6...)

Après la sélection de la langue, vous trouvez dans le menu du CD, outre ce manuel (qui existe aussi en version téléchargeable sur notre page d'accueil), un grand nombre d'informations, de fiches techniques et de photos, également le menu « Software ». Dans ce menu, vous trouverez tous les outils du logiciel, le driver USB et les exemples de programme avec leur code source pour le RP6.

En fonction des réglages de sécurité de votre navigateur Internet, vous pouvez démarrer les programmes d'installation directement à partir du CD. Si votre navigateur ne le permet pas pour des raisons de sécurité, vous devez d'abord enregistrer les fichiers dans un répertoire sur votre disque dur et les démarrer à partir de là. Vous trouverez des instructions plus détaillées sur la page Logiciel du Menu CD. Sinon vous pouvez basculer sur le lecteur CD à l'aide d'un gestionnaire de fichiers et installer le logiciel à partir du CD. Les noms des répertoires sont choisis de façon à pouvoir les attribuer sans équivoque aux paquets de logiciels et systèmes d'exploitation correspondants.

3.2.2. WinAVR - pour Windows

Tout d'abord nous allons installer WinAVR. Mais comme son nom l'indique WinAVR n'est disponible que pour Windows!

Les utilisateurs de Linux doivent aller directement au chapitre suivant.

WinAVR (qui se prononce comme le mot anglais "whenever") est une collection de nombreux programmes utiles et nécessaires pour le développement de logiciels destinés aux microcontrôleurs AVR en langage C. WinAVR contient outre le GCC pour AVR (cela s'appelle alors « AVR-GCC », d'autres informations suivront) également l'éditeur de texte source convivial « Programmers Notepad 2 » qui nous allons utiliser pour le développement du programme pour le RP6. WinAVR est un projet privé qui n'est pas adossé à une société ou autre. Il est disponible gratuitement sur Internet. Vous trouverez des versions plus récentes et d'autres informations à l'adresse suivante:

```
http://winavr.sourceforge.net/
```

Entretemps le projet a trouvé le soutien officiel d'ATMEL et l'AVR-GCC se laisse intégrer dans l'AVRStudio, l'environnement de développement pour AVR d'ATMEL. Nous n'allons cependant pas nous attarder sur ce sujet car le Programmers Notepad est mieux approprié pour notre projet.

Vous trouverez le fichier d'installation de WinAVR sur le CD dans le dossier:

```
<Lecteur CD-ROM>:\Software\AVR-GCC\Windows\WinAVR\
```

L'installation de WinAVR est très simple et rapide. Normalement il n'y a rien à modifier et il suffit de cliquer toujours sur « Suivant »!

Si vous travaillez sous Windows Vista, vous devez absolument utiliser la version la plus récente 20070525 de WinAVR. Tout devrait fonctionner sans problèmes sous Windows 2000 et XP. Si ce n'est pas le cas, vous pouvez essayer l'une des deux versions plus anciennes qui se trouvent également sur le CD (avant une nouvelle installation, désinstallez systématiquement la version de WinAVR déjà installée!). Officiellement des systèmes Win x64 ne sont pas encore supportés mais le CD contient un patch pour ces systèmes au cas où il y aurait un problème. Vous trouverez d'autres informations sur la page du logiciel dans le menu du CD.

3.2.3. AVR-GCC, avr-libc et avr-binutils - pour Linux

Les utilisateurs de Windows peuvent sauter ce paragraphe!

Sous Linux les choses se compliquent un peu. Les paquets requis existent certes déjà dans certaines distributions mais il s'agit souvent de versions obsolètes.

C'est pourquoi vous devez compiler et installer des versions plus récentes.

Ne pouvant pas rentrer dans le détail des nombreuses distributions Linux tels que SuSE, Ubuntu, RedHat/Fedora, Debian, Gentoo, Slackware, Mandriva, etc. en x versions différentes ayant chacune ses particularités, nous n'en faisons qu'un descriptif

très général.

Cela s'applique également à tous les autres paragraphes traitant de Linux dans ce chapitre!

Le procédé décrit ici ne sera donc pas forcément couronné de succès dans votre cas. Souvent il peut être utile de rechercher sur Internet avec des mots-clés comme « <LinuxDistribution> avr gcc » (essayez plusieurs orthographes). Cela s'applique aussi aux autres paragraphes sur Linux – évidemment avec des mots-clé adaptés. Si vous rencontrez des problèmes lors de l'installation de l'AVR-GCC, vous pouvez également consulter notre forum ou celui du réseau de robotique ou bien un des nombreux forums de Linux.

Tout d'abord vous devez désinstaller des versions existantes de l'avr-gcc, des avr-binutils et de l'avr-libc car elles sont généralement obsolètes. Vous pouvez le faire au moyen du gestionnaire de paquets de votre distribution en faisant une recherche sur « avr » et en désinstallant les trois paquets mentionnés ci-dessus, dans la mesure où elles existent.

Vous pouvez savoir si et où l'avr-gcc est déjà installé à l'aide d'une ligne de commande

```
> which avr-gcc
```

Si un chemin s'affiche, il existe déjà une version.

Dans ce cas, entrez tout simplement:

```
> avr-gcc --version
```

et regardez le résultat. Si un numéro de version inférieur à 3.4.6 s'affiche, vous devez absolument désinstaller cette ancienne version. Si le numéro de la version se situe entre 3.4.6 et 4.1.0, vous pouvez d'abord essayer de compiler les programmes (voir chapitre suivant) et installer les nouveaux outils seulement si la compilation échoue. Nous allons installer ici la version actuellement la plus récente 4.1.1 (Version de mars 2007) avec quelques patchs importants.

Si les paquets ci-dessus n'apparaissent pas dans le gestionnaire de paquets bien qu'il soit sûr qu'un avr-gcc existe déjà, vous devez effacer les fichiers binaires correspondants manuellement, donc rechercher et effacer tous les fichiers commençant par « avr » dans les répertoires /bin, /usr/bin, etc. (évidemment UNIQUEMENT ces fichiers et rien d'autre!). Des répertoires éventuellement existants comme /usr/avr ou /usr/local/avr doivent également être effacés.

Attention: Vous devez absolument vous assurer que les outils de développement normaux de Linux tels que GCC, make, binutils, libc, etc. sont installés avant de pouvoir commencer la traduction et l'installation. Utilisez pour cela le gestionnaire de paquets de votre distribution. Chaque distribution Linux devrait déjà livrer les paquets requis sur le CD d'installation ou bien mettre des paquets actualisés à disposition sur Internet.

Assurez-vous que le programme « textinfo » est installé. Installez le paquet correspondant, le cas échéant, avant de continuer, sinon cela ne fonctionnera pas!

Lorsque cela sera fait, vous pouvez commencer avec l'installation proprement dite.

Il y a deux possibilités: soit vous faites tout manuellement, soit vous utilisez un scripte d'installation très simple à mettre en œuvre.

Nous vous conseillons d'essayer d'abord le scripte et si cela échoue, vous pouvez installer le compilateur manuellement.

Attention: Vous devez disposer de suffisamment d'espace mémoire sur le disque dur! L'installation requiert temporairement plus de 400Mb dont plus de 300Mb pourront être effacés après l'installation mais pendant la traduction, vous aurez besoin de tout cet espace.

De nombreuses étapes de l'installation nécessitent des **DROITS ROOT**. Vous devez donc vous enregistrer éventuellement par la commande « su » comme root ou vous exécutez les commandes critiques par « sudo » ou similaire, comme c'est le cas p.ex. avec Ubuntu (le scripte d'installation, mkdir dans les répertoires /usr/local et make install nécessitent des droits root).

Respectez l'orthographe EXACTE de toutes les commandes! Chaque signe est important et même si certaines commandes semblent étranges, il ne s'agit pas d'une erreur typographique! (Mais il faut quand même remplacer `<Lecteur CD-ROM>` par le

chemin du lecteur CD-ROM!)

Toutes les données d'installation importantes pour l'avr-gcc, avr-libc et binutils se trouvent sur le CD dans le dossier:

```
<Lecteur CD-ROM>:\Software\avr-gcc\Linux
```

Copiez tous les fichiers d'installation dans un répertoire sur votre disque dur – **cela s'applique aux deux variantes d'installation!** Nous utilisons ici le répertoire Home (l'abréviation courante pour le répertoire Home est le tilde: „~"):

```
> mkdir ~/RP6
> cd <Lecteur CD-ROM>/Software/avr-gcc/Linux
> cp * ~/RP6
```

Vous pouvez effacer les fichiers après l'installation réussie afin d'économiser de l'espace!

3.2.3.1. Script d'installation automatique

Lorsque le scripte est devenu exécutable avec chmod, vous pouvez commencer:

```
> cd ~/RP6
> chmod -x avrgcc_build_and_install.sh
> ./avrgcc_build_and_install.sh
```

A la question si vous voulez faire l'installation avec cette configuration, répondez par « y ».

ATTENTION: La traduction et l'installation prendront en fonction de la capacité de votre système, un certain temps (env. 15 mn. sur un Notebook CoreDuo 2GHz – ou plus si votre système est plus lent).

Le scripte intègre aussi quelques patchs. Ce sont tous les fichiers .diff qui se trouvent dans le répertoire.

Si tout se passe bien, le message suivant devrait s'afficher tout à la fin:

```
(./avrgcc_build_and_install.sh)
(./avrgcc_build_and_install.sh) installation of avr GNU tools complete
(./avrgcc_build_and_install.sh) add /usr/local/avr/bin to your path to use the avr GNU tools
(./avrgcc_build_and_install.sh) you might want to run the following to save disk space:
(./avrgcc_build_and_install.sh)
(./avrgcc_build_and_install.sh) rm -rf /usr/local/avr/source /usr/local/avr/build
```

Alors vous pouvez exécuter la proposition

```
rm -rf /usr/local/avr/source /usr/local/avr/build
```

Cela efface tous les fichiers temporaires dont vous n'aurez plus besoin.

Vous pouvez sauter le paragraphe suivant et mettre le chemin sur les avr tools.

Si l'exécution du scripte échoue, vous devez examiner soigneusement les messages d'erreur (montez aussi plus haut dans la liste). Souvent, il manque quelques programmes qu'il faut installer auparavant (comme p.ex. le « textinfo » mentionné plus haut). Avant de continuer après une erreur, vous devriez effacer les fichiers déjà générés dans le répertoire d'installation standard « /usr/local/avr ». Effacez de préférence le répertoire entier.

Si vous ne savez pas exactement où se trouve l'erreur, enregistrez toutes les lignes de commande dans un fichier et adressez-vous au support. Envoyez toujours un maximum d'informations car cela facilitera la recherche d'erreur.

3.2.3.2. Installation manuelle

Si vous préférez installer le compilateur manuellement ou l'installation ne fonctionne pas avec le scripte, vous pouvez suivre les instructions dans le paragraphe suivant:

Cette description est basée sur cet article:

```
http://www.nongnu.org/avr-libc/user-manual/install_tools.html
```

qui se trouve aussi dans la documentation AVR Libc au format PDF sur le CD:

```
<Lecteur CD-ROM->:\Software\Documentation\avr-libc-user-manual-1.4.5.pdf
```

à partir de la page PDF 240 (soit page 232 d'après les numéros de page du document).

Nous allons largement simplifier la procédure mais intégrer quand-même quelques patchs sans lesquels certaines fonctions ne se dérouleront pas correctement.

Nous devons d'abord créer un répertoire dans lequel nous allons installer tous les outils. Cela doit être le `/usr/local/avr` .

`Donc, entrez dans une ligne de commande en tant que` **ROOT** :

```
> mkdir /usr/local/avr
> mkdir /usr/local/avr/bin
```

Ce n'est pas obligatoirement ce répertoire. Nous allons simplement créer la variable `$PREFIX pour ce répertoire`:

```
> PREFIX=/usr/local/avr
> export PREFIX
```

Il est IMPERATIF d'ajouter maintenant la variable du chemin:

```
> PATH=$PATH:$PREFIX/bin
> export PATH
```

Binutils pour AVR

Vous devez dépaqueter le code source des binutils et intégrer quelques patchs. Nous supposons ici que vous avez tout copié dans le répertoire home ~/RP6:

```
> cd ~/RP6
> bunzip2 -c binutils-2.17.tar.bz2 | tar xf -
> cd binutils-2.17
> patch -p0 < ../binutils-patch-aa.diff
> patch -p0 < ../binutils-patch-atmega256x.diff
> patch -p0 < ../binutils-patch-coff-avr.diff
> patch -p0 < ../binutils-patch-newdevices.diff
> patch -p0 < ../binutils-patch-avr-size.diff
> mkdir obj-avr
> cd obj-avr
```

Exécutez maintenant le scripte « configure »:

```
> ../configure --prefix=$PREFIX --target=avr --disable-nls
```

Ce scripte détecte ce qui est disponible dans votre système et génère les makefiles correspondants. Maintenant vous pouvez traduire et installer les binutils:

```
> make
> make install
```

Selon la capacité de calcul de votre système, ceci peut prendre quelques minutes.

Cela s'applique également aux deux paragraphes suivants, surtout pour le GCC!

GCC pour l'AVR

Le GCC est patché, traduit et installé comme les binutils:

```
> cd ~/RP6
> bunzip2 -c gcc-4.1.1.tar.bz2 | tar xf -
> cd gcc-4.1.1
> patch -p0 < ../gcc-patch-0b-constants.diff
> patch -p0 < ../gcc-patch-attribute_alias.diff
> patch -p0 < ../gcc-patch-bug25672.diff
> patch -p0 < ../gcc-patch-dwarf.diff
> patch -p0 < ../gcc-patch-libiberty-Makefile.in.diff
> patch -p0 < ../gcc-patch-newdevices.diff
> patch -p0 < ../gcc-patch-zz-atmega256x.diff
> mkdir obj-avr
> cd obj-avr
> ../configure --prefix=$PREFIX --target=avr --enable-languages=c,c++ \
      --disable-nls --disable-libssp –with-dwarf2
> make
> make install
```

Après le \ vous pouvez simplement appuyer sur ENTREE et continuer à écrire. La commande peut ainsi être répartie sur plusieurs lignes mais on peut aussi la laisser complètement de côté.

AVR Libc

Maintenant il ne reste plus que l'AVR libc:

```
> cd ~/RP6
> bunzip2 -c avr-libc-1.4.5.tar.bz2 | tar xf -
> cd avr-libc-1.4.5
> ./configure --prefix=$PREFIX --build=`./config.guess` --host=avr
> make
> make install
```

Attention: veillez à bien saisir un accent grave dans la commande –build=`./config.guess` et non pas un simple apostrophe ou guillemet (appuyez sur la touche Alt Gr et la touche 7 de votre clavier) sinon cela ne fonctionnera pas.

3.2.3.3. Enregistrer le chemin

Vous devez faire en sorte maintenant que le répertoire `/usr/local/avr/bin` soit également enregistré dans la variable du chemin, sinon vous ne pouvez pas appeler l'avr-

gcc de la ligne de commande ou des makefiles. Pour cela, vous devez enregistrer le chemin dans le fichier `/etc/profile` ou bien `/etc/environment` ou similaire (varie

d'une distribution à l'autre) – séparé des autres enregistrements déjà existants par deux points « : ». Dans le fichier cela ressemblerait à ceci:

```
PATH="/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/usr/local/avr/bin"
```

Entrez maintenant dans une ligne de commande quelconque « avr-gcc --version » comme décrit auparavant. Si cela fonctionne, l'installation a réussi!

3.2.4. Java 6

Le RP6Loader (informations voir ci-dessous) a été développé pour la plateforme Java et est utilisable sous Windows et Linux (théoriquement aussi sous d'autres systèmes d'exploitation comme OS X mais là, AREXX n'est malheureusement pas encore en mesure de donner un support officiel). A cet effet, il est nécessaire d'installer un Java Runtime Environment (JRE) actualisé. Il est souvent déjà installé dans votre ordinateur mais il doit s'agir au minimum de la version 1.6 (= Java 6). Si vous n'avez pas encore de JRK ou JDK, vous devez d'abord installer le JRE 1.6 de la société SUN Microsystems qui se trouve sur le CD ou télécharger une version plus récente de `http://www.java.com` ou `http://java.sun.com`.

3.2.4.1. Windows

Sous Windows, le JRE 1.6 se trouve dans le dossier suivant:

```
<Lecteur CD-ROM>:\Software\Java\JRE6\Windows\
```

Sous Windows l'installation de Java est très simple. Il vous suffit de démarrer le Setup et de suivre les instructions à l'écran – c'est tout. Vous pouvez sauter le paragraphe suivant.

3.2.4.2. Linux

Sous Linux, l'installation se passe généralement sans problème mais certaines distributions exigent quelques interventions manuelles.

Dans ce dossier:

```
<Lecteur CD-ROM>:\Software\Java\JRE6\
```

vous trouverez le JRE 1.6 comme RPM (SuSE, RedHat, etc.) et comme archive auto-extractible « .bin ». Sous Linux, il est recommandé de chercher d'abord dans le gestionnaire de paquets de votre distribution les paquets Java (mots-clé: „java", „sun", „jre", „java6" ...) et d'utiliser ces paquets propres à la distribution et non pas ceux qui se trouvent sur le CD-ROM. Veillez en tout cas à installer Java 6 (=1.6) ou une version plus récente mais en aucun cas une version plus ancienne!

Sous Ubuntu ou Debian, l'archive RPM ne fonctionne pas directement. Vous devez faire appel au gestionnaire de paquets de votre distribution pour trouver un paquet d'installation approprié. Cependant le RPM devrait fonctionner sans problème avec

beaucoup d'autres distributions tels que RedHat/Fedora et SuSE. Sinon il reste la possibilité de dépaqueter le JRE de l'archive auto-extractible (.bin) (p.ex. dans /usr/lib/Java6) et de créer ensuite manuellement les chemins vers le JRE (PATH et JAVA_HOME etc.).

Respectez les consignes d'installation de Sun qui se trouvent également dans le répertoire mentionné ci-dessus et sur le site Web Java (voir ci-dessus)!

Vérifiez dans une ligne de commande si Java a été correctement installé en exécutant la commande „`java -version`". Le résultat suivant devrait s'afficher:

```
java version "1.6.0"
Java(TM) SE Runtime Environment (build 1.6.0-b105)
Java HotSpot(TM) Client VM (build 1.6.0-b105, mixed mode, sharing)
```

Si un autre message s'affiche, vous avez soit installé la mauvaise version, soit un autre Java VM est encore installé sur votre système.

3.2.5. RP6Loader

Le RP6Loader a été développé pour pouvoir charger facilement de nouveaux programmes et tous les modules d'extension dans le RP6 (dans la mesure où ils disposent d'un microcontrôleur avec chargeur d'amorçage). Par ailleurs, quelques fonctions supplémentaires utiles ont été intégrés tel qu'un programme de terminal simple.

Ce n'est pas nécessaire d'installer le RP6Loader. Le programme peut tout simplement être copié dans un nouveau classeur sur le disque dur. Le RP6Loader se trouve dans une archive zip sur le CD-ROM:

```
<Lecteur CD-ROM>:\Software\RP6Loader\RP6Loader.zip
```

Il vous suffit de le dépaqueter quelque part sur le disque dur, p.ex. dans un nouveau classeur `C:\Programme\RP6Loader` (ou similaire). Dans ce dossier vous trouverez le

fichier RP6Loader.exe et vous pouvez le démarrer avec un double-clic.

Le véritable programme RP6Loader se trouve dans l'archive Java (JAR) RP6Loader_lib.jar. Vous pouvez aussi le démarrer à partir de la ligne de commande.

Sous Windows:

```
java -Djava.library.path=".\lib" -jar RP6Loader_lib.jar
```

Linux:

```
java -Djava.library.path="./lib" -jar RP6Loader_lib.jar
```

Cette longue option -D est nécessaire pour que le JVM puisse trouver toutes les librairies utilisées. Sous Windows, on n'en a pas besoin. Il suffit d'utiliser le fichier .exe pour le démarrage. Pour Linux, il existe un scripte shell « RP6Loader.sh » qu'il faut

éventuellement rendre exécutable (`chmod -x ./RP6Loader.sh`). Ensuite vous pouvez le

démarrer dans une ligne de commande avec „./RP6Loader.sh".

Il est recommandé de créer un lien sur le bureau ou dans le menu Démarrer afin de pouvoir démarrer le RP6Loader facilement. Sous Windows, il suffit de faire un clic droit sur le fichier RP6Loader.exe et de sélectionner dans le menu « Envoyer à » l'option « Créer un raccourci sur le bureau ».

3.2.6. RP6 Library, RP6 CONTROL Library et Exemples de Programmes

La RP6Library et les exemples de programmes associés se trouvent dans une archive Zip sur le CD:

```
<Lecteur CD-ROM>:\Software\RP6Examples\RP6Examples.zip
```

Vous pouvez décompresser ces fichiers directement dans un répertoire de votre choix sur le disque dur. Il est recommandé de dézipper les exemples de programmes dans un dossier sur une partition de données ou dans le dossier « Mes Fichiers » dans un sous-dossier „RP6\Examples\" ou sous Linux dans le répertoire Home. C'est à vous de décider.

Les différents exemples de programmes seront abordés dans le chapitre sur le logiciel!

Cette archive contient déjà les exemples pour le module d'extension RP6 CONTROL M32 et les fichiers Library associés!

3.3. Branchement de l'interface USB – Windows

Les utilisateurs de Linux peuvent sauter ce chapitre!

Il y a plusieurs façons d'installer l'interface USB, la plus simple étant d'**installer le driver AVANT la première connexion** du matériel. Le programme d'installation pour le driver se trouve sur le CD.

Pour les systèmes **32 et 64 Bit Windows XP, Vista, Server 2003 et 2000** :

```
<Lecteur CD-ROM>:\Software\USB_DRIVER\Win2k_XP\CDM_Setup.exe
```

*Pour les anciens systèmes **Win98SE/ME** il n'existe malheureusement pas un programme aussi convivial. Il faut installer manuellement un driver plus ancien après avoir connecté le matériel (voir ci-dessous).*

Il suffit d'exécuter le programme d'installation CDM. Un message s'affiche brièvement après l'installation du driver et c'est tout.

Ensuite vous pouvez connecter l'interface USB au PC. NE LE CONNECTEZ PAS ENCORE AU ROBOT! Branchez juste le cordon USB sur le robot! Veillez à tenir la platine de l'interface USB uniquement par les bords ou la fiche USB ou le boîtier en plastique de la fiche de programmation (voir consignes de sécurité relatives aux décharges statiques)! En règle générale, évitez de toucher les composants sur la platine, les soudures ou les contacts de la fiche si ce n'est pas absolument nécessaire afin d'éviter des décharges électrostatiques.

Le driver installé auparavant est maintenant utilisé automatiquement pour le matériel sans autre intervention de votre part. Sous Windows XP/2000, de petites bulles apparaissent au-dessus de la barre de tâches. Le dernier message devrait ressembler à ceci: « Le matériel a été installé et est prêt à l'emploi ».

Si vous avez branché l'interface USB avant l'installation (ou si vous utilisez Win98/ME) – ce n'est pas grave non plus. Windows vous demandera un driver qui se trouve également sous forme décompressée sur le CD.

Si c'est le cas, un dialogue s'affiche généralement sous Windows pour installer le driver du nouveau matériel détecté. Vous devez indiquer au système le chemin où il peut trouver le driver. Sous Windows XP/2000, il faut d'abord choisir l'installation manuelle du driver et ne pas chercher un service Internet ou autre. Dans notre cas d'espèce, le driver se trouve sur le CD dans les répertoires indiqués ci-dessus.

Donc, il suffit d'indiquer le répertoire correspondant à votre version de Windows ainsi que quelques fichiers, le cas échéant, que le système ne trouve pas automatiquement (sont tous dans les répertoires indiqués ci-après!)...

Sous Windows XP et les versions ultérieures, un message s'affiche généralement qui prévient que le driver n'a pas été signé/vérifié par Microsoft (dans notre cas c'est peu probable puisque les drivers FTDI sont signés). Ce message n'a pas d'importance et peut être confirmé sans aucun problème.

Pour systèmes **32 et 64 Bit Windows XP, Vista, Server 2003 et 2000** :

```
<Lecteur CD-ROM>:\Software\USB_DRIVER\Win2k_XP\FTDI_CDM2\
```

Pour les systèmes plus anciens **Windows 98SE/Me** :

```
<CD-ROM-Laufwerk>:\Software\USB_DRIVER\Win98SE_ME\FTDI_D2XX\
```

Les versions plus anciennes de Windows telles que Win98SE nécessitent éventuellement un redémarrage du système après l'installation. **ATTENTION**: Sous Win98/ME un seul des deux drivers fonctionne: Virtual Comport ou le driver D2XX de FTDI! Il n'existe malheureusement pas de driver qui intègre les deux fonctions et il n'existe normalement pas de Comport virtuel puisque le RP6Loader utilise d'office le driver D2XX sous Windows (c'est modifiable – contactez notre équipe d'assistance le cas échéant!).

3.3.1. Vérifiez si le matériel a été correctement branché

Sous Windows XP, 2003 et 2000, vous pouvez utiliser, en plus du RP6Loader, le gestionnaire de périphériques pour vérifier la bonne installation du matériel: Clic droit sur Poste de Travail --> Propriétés --> Matériel --> Gestionnaire de périphériques
OU : Démarrer --> Configuration --> Administration système --> Performance et Maintenance --> Système --> Matériel --> Gestionnaire de périphériques
et vérifier dans l'arborescence sous « Connexions (COM et LPT) » si vous voyez un « USB-Serial Port (COMX) », X étant le numéro du port ou bien regarder dans « USB-Controller » si vous trouvez un « USB Serial Converter »!

3.3.2. Désinstallation ultérieure du Driver

Si vous voulez désinstaller le driver un jour (*Non, ne le faites surtout pas maintenant – il ne s'agit que d'une information!)*: Si vous avez utilisé le programme d'installation CDM, vous pouvez effectuer la désinstallation directement par Démarrer --> Configurations --> Administration Système --> Logiciel. Dans la liste qui s'affiche, vous trouverez le poste « FTDI USB Serial Converter Drivers ». Sélectionnez-le et cliquez sur Désinstaller!

Si vous avez installé le driver manuellement, vous pouvez exécuter le programme "FTUNIN.exe" dans le répertoire du driver USB correspondant à votre système.

Attention: Des adaptateurs USB-->RS232 avec puces FTDI utilisent aussi très souvent ce driver!

3.4. Connexion de l'interface USB – Linux

Les utilisateurs de Windows peuvent sauter ce chapitre!

Dans les systèmes Linux avec noyau (kernel) 2.4.20 ou plus, le driver requis existe déjà (tout du moins pour le prédécesseur compatible FT232BM de la puce sur notre interface USB, le FT232R). Le matériel est reconnu automatiquement et vous n'avez plus rien à faire. Si vous rencontrez un problème, vous trouverez des drivers Linux (et de l'assistance et éventuellement des drivers plus récents) directement chez FTDI:

```
http://www.ftdichip.com/
```

Après avoir branché le matériel, vous pouvez faire afficher avec:

```
cat /proc/tty/driver/usbserial
```

si le port série USB a été correctement installé. Normalement vous n'avez rien d'autre à faire.
Il faut cependant mentionner le fait que sous Windows, le RP6Loader utilise les drivers D2XX et les désignations complètes USB apparaissent dans la liste des ports (p.ex. „USB0 | RP6 USB Interface | serialNumber"). Sous Linux, ce seront les désignations virtuelles Comport /dev/ttyUSB0, /dev/ttyUSB1 etc.. Les Comports classiques s'affichent également comme „dev/ttyS0" etc.. Vous devez tester lequel des ports est le bon!
Malheureusement, il n'existe pas de drivers faciles à installer pour Linux qui remplissent les deux fonctions et il était donc plus logique d'utiliser les drivers Virtual Comport qui existent de toute façon déjà dans le noyau. Les drivers D2XX exigeraient une large intervention manuelle lors de l'installation...

3.5. Fin de l'installation du Logiciel

Voilà tout ce qu'il y avait à faire pour l'installation du logiciel et des interfaces USB!

Maintenant vous pouvez copier les fichiers les plus importants du CD sur le disque dur (surtout un dossier complet « Documentation » et, si cela n'a pas été déjà fait, les exemples de programmes). Cela vous évite de chercher perpétuellement sur le CD si vous avez besoin d'un de ces fichiers. Les dossiers sur le CD sont tous nommés d'une façon à ce qu'ils soient facilement assignables aux logiciels ou aux documentations associés. Si jamais vous égarez votre CD, vous pouvez télécharger les fichiers les plus importants tel que ce manuel, le RP6Loader et les exemples de programmes de la page d'accueil du site Internet d'AREXX. Vous y trouverez également les liens vers d'autres paquets de logiciel dont vous aurez besoin.

3.6. Installation des Accus

Maintenant nous arrivons enfin au robot lui-même qui nécessite encore 6 accus!

Nous recommandons l'utilisation d'accus NiMH LR6 de la meilleure qualité (fabriqués p.ex. Par Sanyo, Panasonic, et autres) d'une capacité réelle d'au moins 2000mAh (2500mAh seraient l'idéal). N'utilisez pas de piles ordinaires. Non seulement, le coût serait très élevé à long terme mais elles nuisent également à l'environnement.

Chargez les accus avant de les installer! Veillez à ce que **tous les accus présentent le même état de charge** (donc tous fraîchement chargés ou déchargés) et qu'ils sont neufs (si possible)! En fonction de l'âge, du nombre de cycles de charge, du type de charge et de la température ambiante, les accus perdent de plus en plus de leur capacité. Donc, évitez d'utiliser les vieux accus qui traînent dans le placard depuis 3 ans, même s'ils n'ont pas été utilisés...

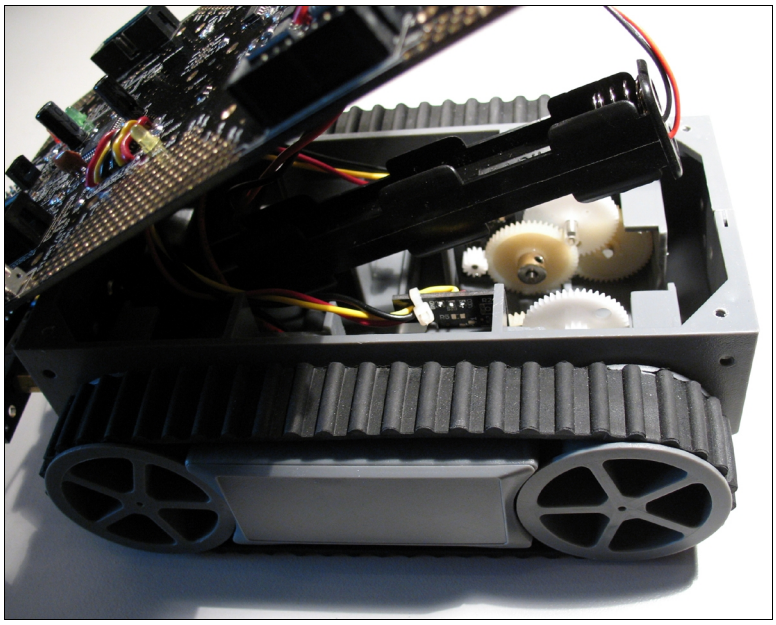
Si vous utilisez un chargeur externe (recommandé mais non inclus dans la livraison), vous n'avez à installer les accus qu'une seule fois. Nous recommandons vivement l'utilisation d'un chargeur contrôlé par microprocesseur afin d'assurer une charge adaptée des accus. N'utilisez que des chargeurs homologués et contrôlés!

Si vous ne possédez pas encore de chargeur externe disposant d'un branchement approprié, vous devez charger les accus dans tous les cas avant l'installation et les démonter, recharger et remonter à chaque fois lorsqu'ils sont vides!

Installation des accus:

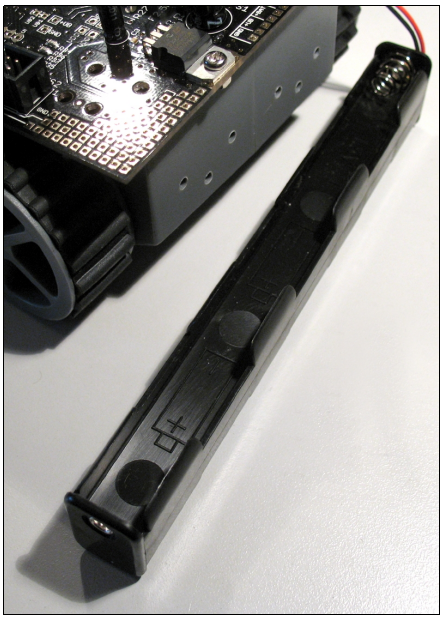
Commencez par déserrer les quatre vis de la carte-mètre (voir fig.)!

Relevez doucement l'arrière de la carte-mère (voir fig. ci-dessous).

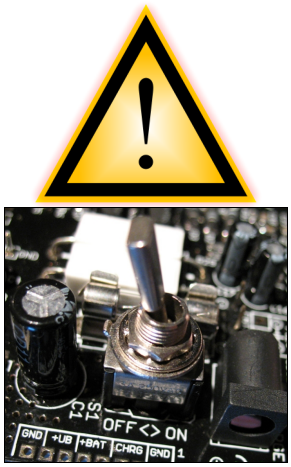


Vous n'êtes pas obligé de débrancher la petite fiche à 3 contacts de la platine du pare-choc! Ne touchez la carte-mère que par les bords ou par des parties en plastique afin d'éviter des décharges électrostatiques !

La carte-mère est reliée aux moteurs, aux encodeurs et au porte-accu par quelques câbles soudés. Ecartez légèrement ces câbles.

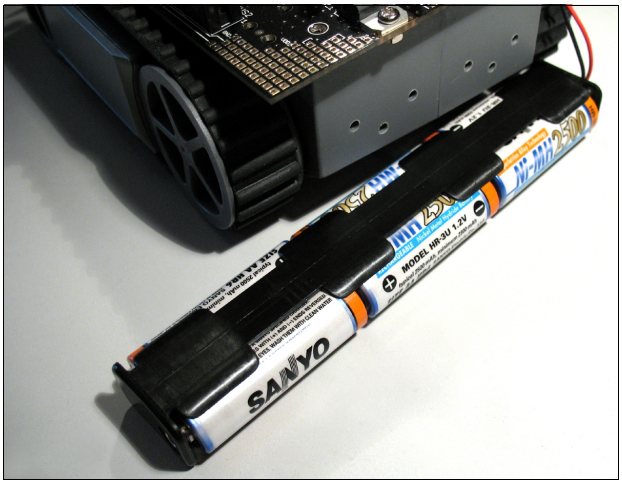


Ensuite vous sortez le porte-accu noir du centre vers l'arrière (voir fig.).



Vérifiez maintenant que le commutateur sur la carte-mère du robot est en position OFF donc placé en direction du texte « OFF » ou du grand condensateur cylindrique sur la carte-mère (voir fig.)!

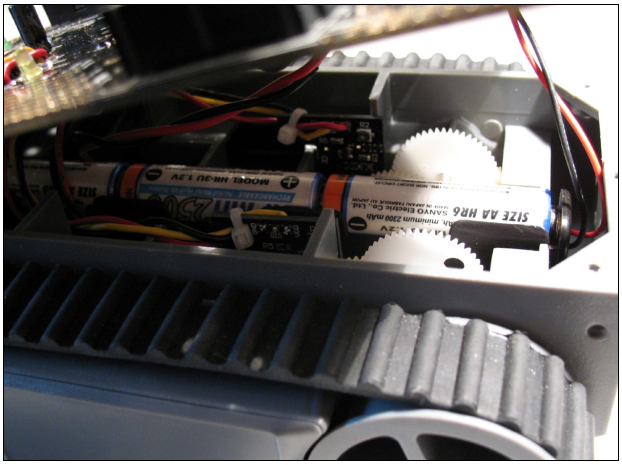
Avant de remettre le robot sous tension, vous devez vérifier si les accus ont été placés en respectant la bonne polarité.



Maintenant vous pouvez placer les 6 accus LR6 NiMH **AVEC LA BONNE POLARITE** dans le porte-accu! **Attention: Si vous insérez les accus à l'envers, le fusible devrait sauter!**

Il se peut également que l'électronique soit partiellement endommagée!

Il vaut mieux mettre les accus tout de suite dans le bon sens pour éviter ce genre de problèmes! **Le porte-accu et les accus portent des repères** ((+) et (-), le côté négatif (le côté méplat) des accus doit être dirigé vers les ressorts dans le porte-accus...)!



Il vaut mieux vérifier trois fois que les accus sont correctement installés!

Posez le porte-accus sur le châssis et **vérifiez que tous les câbles sont correctement rangés dans le châssis et ne peuvent pas toucher les roues dentées des deux engrenages!**

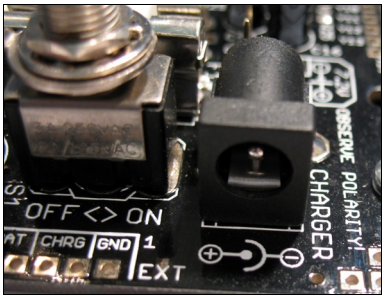
Puisque le robot est ouvert, vous pouvez rapidement vérifier les deux engrenages et encodeurs pour savoir si rien n'a été endommagé ou s'est déserré pendant le transport. Tournez **une fois très doucement et lentement** les roues arrières! Un demi tour vers l'avant et l'arrière devrait suffire. Vous devez sentir une nette résistance mais la roue devrait quand-même bien tourner. Les roues dentées doivent pouvoir tourner librement! Voir annexe A!

Remettez la carte-mère en place. Ecartez avec les doigts ou un tournevis les câbles qui se seraient glissés entre la carte-mère et les entretoises en plastique du châssis **afin que la carte-mère pose parfaitement à plat sur le châssis!** Avant de serrer les vis, **vérifiez à nouveau** qu'aucun câble ne s'est coincé entre la carte-mère et le châssis, ni s'est glissé dans les engrenages. Ensuite vous pouvez fixer la carte-mère avec les 4 vis!

3.7. Chargement des Accus

Si ce n'est déjà fait, c'est le moment de charger les accus à l'aide d'un chargeur externe. L'interrupteur principal doit rester en position OFF! Vous ne devez charger les accus qu'à l'état éteint. L'interrupteur connecte les accus soit sur l'électronique du RP6, soit sur la fiche de charge.

Veillez à brancher le chargeur avec la bonne polarité sur la prise de charge (marquée « Charger ») à côté de l'interrupteur général du robot! La polarité est imprimée sur la carte-mère devant la fiche (voir fig.). **Le négatif est à l'EXTERIEUR, le positif à l'INTERIEUR!**



La durée de charge dépend du chargeur et des accus utilisés (3 à 4 heures avec un chargeur rapide commandé par microcontrôleur tel que le Voltcraft 1A / 2A Delta Peak ou Ansmann ACS110 / 410, ou env. 14 heures avec un chargeur normal tel que l'AC48) – vous trouverez plus de précisions dans le manuel de votre chargeur.

NE PAS mettre sous tension le robot pendant le processus de charge: Débranchez le chargeur avant de remettre le robot sous tension!

3.8. Le premier Test

ATTENTION! Lisez d'abord entièrement ce chapitre et le suivant avant de mettre le robot sous tension! Si tout ne se déroule pas comme décrit ici, éteignez *immédiatement* le robot et notez précisément la défaillance. Si vous ne trouvez pas la solution dans le chapitre « Diagnostic de défaillance », contactez l'assistance!

Voici le moment venu de mettre le robot sous tension pour la première fois! Mettez le robot sous tension à l'interrupteur général. Les deux LEDs d'état rouges à côté de l'interrupteur général devraient s'allumer et peu de temps après, une des autres LEDs rouges (SL6) devraient commencer à clignoter pendant qu'une des LEDs vertes (SL1) reste allumée. Cela signifie par ailleurs que la mémoire de la commande ne contient pas de programme d'application. Si c'était le cas, seule la LED d'état verte SL1 clignoterait.

La LED jaune PWRON devrait s'allumer pendant 1 seconde après la mise sous tension

et s'éteindre immédiatement. Vous pouvez économiser de l'énergie en éteignant la plupart des détecteurs non-utilisés tels que les encodeurs.

Après env. une demie minute, la LED rouge SL6 devrait s'arrêter de clignoter et toutes les LEDs doivent s'éteindre. Puisque le microcontrôleur sur le robot n'a pas encore chargé de programme d'utilisateur qu'il pourrait exécuter, il commute maintenant automatiquement en veille dont il pourra sortir par l'interface USB ou une pression sur la touche start/stop ou l'arrêt/mise sous tension. Même en veille le robot consomme un peu d'énergie (jusqu'à 5mA), donc n'oubliez pas de l'éteindre après utilisation. D'ailleurs, lorsqu'un programme est chargé, le robot ne commute pas automatiquement en mode veille. Il attend toujours de nouvelles commandes utilisateur ou bien la commande de départ provenant de l'interface sérielle (envoi d'un « s ») ou du bus I²C.

3.8.1. Connexion de l'Interface USB et Démarrage du RP6Loader

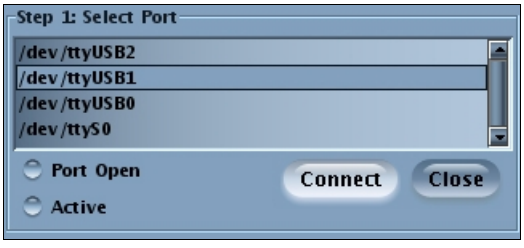
Ensuite nous testons le téléchargement du programme par l'interface USB. Connectez l'interface USB au PC **(Effectuez toujours la connexion au PC EN PREMIER!)** et ensuite à la fiche « PROG/UART » à côté du commutateur Start/Stop au moyen du câble en nappe à 10 contacts. Le câble en nappe à 10 contacts est protégé mécaniquement contre l'inversion de polarité, c'est-à-dire à moins d'user de la force, il est impossible de le brancher à l'envers.

Démarrez ensuite le RP6Loader. En fonction de la langue sélectionnée, les menus peuvent être nommés un peu différemment. Sur les impressions d'écran, nous avons représenté la version anglaise. Dans le menu « Options -> Préférences » et ensuite dans « Language / Sprache », vous pouvez sélectionner la langue (anglais ou allemand) et cliquer sur OK. Après la modification de la langue, vous devez redémarrer le RP6Loader afin d'appliquer les changements.

Ouvrir le port- <u>Windows</u>

Vous pouvez maintenant sélectionner le port USB. Si aucun autre adaptateur USB-série avec contrôleur FTDI n'est connecté sur le PC, vous ne verrez qu'un seul port dans la liste que vous devez sélectionner. S'il existe plusieurs ports, vous pouvez identifier le port à l'aide du nom « RP6 USB Interface » (ou « FT232R » USB UART ») plus le numéro série programmé.

Si aucun port n'apparaît, vous pouvez actualiser la liste dans le menu « RP6Loader -->Refresh Portlist » (RP6Loader --> actualiser la liste des ports).



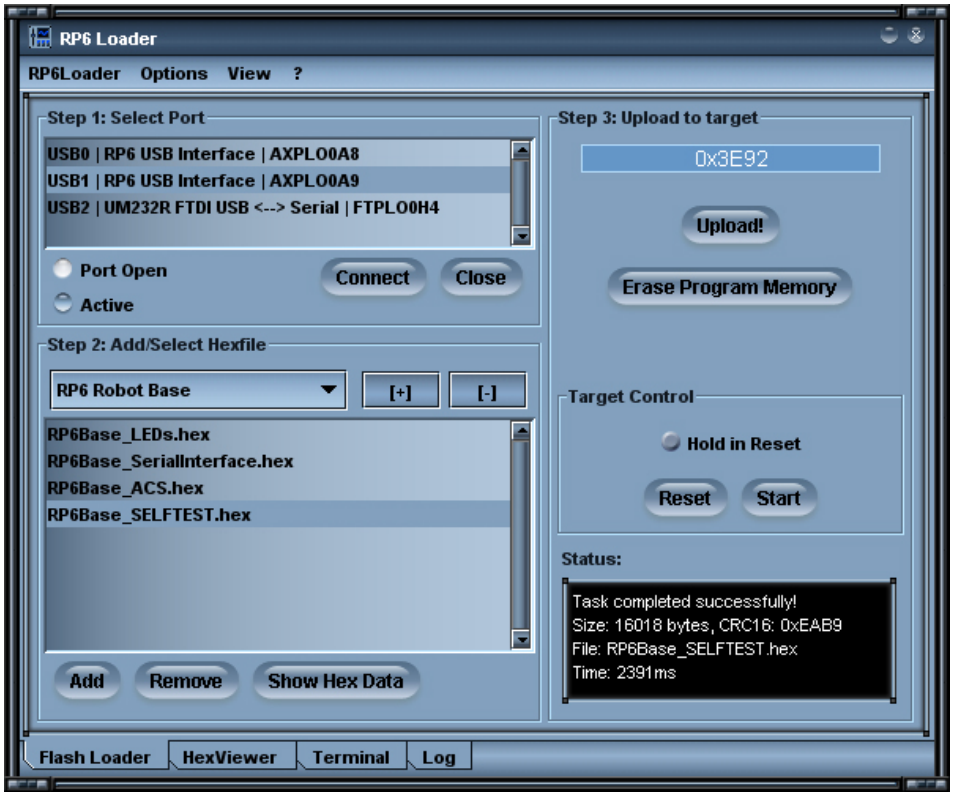
Ouvrir le port – <u>Linux</u>

Sous Linux, l'adaptateur série – USB est traité comme un port COM ordinaire. L'installation du driver D2XX de FTDI sous Linux ne serait pas si

simple et les drivers normaux Virtual Comport (VCP) sont de toute manière déjà inclus dans les noyaux Linux actuels. Tout fonctionne presque comme sous Windows sauf qu'il faut trouver le nom de l'interface USB RP6 et veiller à ne pas séparer le port USB du PC tant que la connexion est ouverte (sinon il faut éventuellement redémarrer le RP6Loader afin de rétablir la connexion). Les Comports virtuels s'appellent sous Linux „/dev/ttyUSBx", x représentant un chiffre p.ex. „/dev/ttyUSB0" ou „/dev/ttyUSB1". Les Comports normaux s'appellent sous Linux „/dev/ttyS0", „/dev/ttyS1" etc.. Ils apparaissent également dans la liste des ports.

S'il existe plusieurs ports, le RP6Loader retient quel port vous avez utilisé en dernier et le sélectionne automatiquement à chaque nouveau démarrage du programme (en principe, la plupart des réglages et sélections sont conservés).

Maintenant vous pouvez cliquer sur le bouton « Connect » (= « Connecter »). Le RP6-Loader ouvre le port et effectue un test pour savoir si la communication avec le chargeur d'amorçage sur le robot fonctionne. Normalement un message s'affiche dans la zone noire « Status » qui dit « Connected to: RP6 Robot Base... » ou similaire ainsi qu'une information sur la tension d'accu actuellement mesurée. Si ce n'est pas le cas, faites une nouvelle tentative! Si cela ne fonctionne toujours pas, il y a une erreur. Mettez immédiatement le robot hors tension et lisez attentivement le chapitre sur le diagnostic de défaillance et les solutions. Un avertissement s'affiche si la tension des accus est trop faible. **Au plus tard maintenant vous devez recharger les accus** (l'idéal est de les charger lorsque la tension est tombée en dessous de 5,9V)!



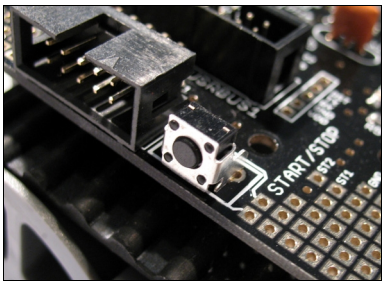
Lorsque cela a bien fonctionné, vous pouvez exécuter un petit programme d'auto-test

pour vérifier l'état de bon fonctionnement des tous les systèmes du robot. Pour ce faire, cliquez sur le bouton « Add » (Ajouter ») dans la fenêtre du RP6Loader et sélectionnez le fichier « RP6Base_SELFTEST\RP6Base_SELFTEST.hex » dans le répertoire d'exemple. Ce fichier contient le programme d'auto-test au format hexadécimal.

Le fichier que vous venez de sélectionner, apparaît ensuite dans la liste. Ainsi, vous pouvez ajouter ultérieurement d'autres fichiers hexadécimaux provenant de vos propres programmes et des exemples de programmes (voir l'impression d'écran. Plusieurs fichiers hexadécimaux y ont déjà été ajoutés). Le RP6Loader arrive à gérer différentes catégories de fichiers hexadécimaux qui permettent de trier les fichiers d'une manière plus claire. C'est particulièrement utile lorsque vous avez plusieurs modules d'extension programmables sur le RP6 ou si vous utilisez différentes variantes de programmes. A la fin du programme, la liste est toujours enregistrée automatiquement. Evidemment, seuls les chemins vers les fichiers hexadécimaux sont enregistrés, pas les fichiers hexadécimaux eux-mêmes. Si vous travaillez sur un programme, vous n'avez qu'à ajouter une seule fois le fichier hexadécimal et le sélectionner. Après chaque nouvelle traduction du programme, vous pouvez le charger immédiatement dans le microcontrôleur (également à l'aide des touches de claviers [STRG+D] ou [STRG+Y] afin de démarrer le programme immédiatement après le transfert). Les noms des chemins varient évidemment d'un système d'exploitation à l'autre. Vous pouvez cependant utiliser le RP6Loader sans autres modifications sous Windows et Linux car il existe des listes séparées pour Windows et pour Linux.

Sélectionnez maintenant le fichier « RP6Base_SELFTEST.hex » dans la liste et cliquez sur le bouton « Upload ! » en haut à droite en dessous de la barre de progression. Le programme est maintenant chargé dans le MEGA32 sur le RP6. Cela ne devrait pas prendre plus de quelques secondes (maximum 5 secondes pour le programme d'auto-test).

Passez maintenant sur l'onglet « Terminal » (tout en bas dans la fenêtre du programme). Vous pouvez également y accéder par le menu « View » (= Affichage).



Démarrez maintenant le programme en appuyant sur le commutateur Start/Stop sur le RP6, à côté de la fiche de programmation (voir fig.). Ultérieurement, vous pouvez le faire aussi par le menu RP6Loader --> Start Target ou la combinaison de touches [STRG]+[S]. Toutefois, de cette façon, vous pouvez tester en direct si le commutateur fonctionne correctement. Tout d'abord un message d'avertissement devrait s'afficher dans le terminal qui indique que le RP6 allume les moteurs pendant le test n° 8.

ATTENTION! Tenez le RP6 à la main pendant le test n° 8 (test des moteurs et des encodeurs) ou posez-le sur un objet approprié qui empêche le contact des deux chenilles d'entraînement avec le sol! ***Il ne faut <u>en aucun cas</u> toucher ou bloquer les chenilles pendant le test 8!*** Sinon le test échouera probablement! Si le RP6 était posé sur le sol, le comportement de l'entraînement changerait et le test ne pourrait plus fonctionner correctement. En plus, le RP6 avancerait sur une bonne distance et il faudrait le suivre avec le cordon USB - à condition qu'il soit assez long...

Donc: Prendre le RP6 dans la main ou placer un objet en dessous au milieu du RP6 (p.ex. une petite boîte ou une télécommande). Si vous placez un objet en dessous du RP6, vous devriez quand-même le tenir avec une main pendant les test pour qu'il ne glisse pas et tombe de la table par inadvertance!

L'avertissement s'affiche juste avant le test n° 8 et doit être accepté avant de pouvoir continuer, tout comme maintenant au début du programme de test. Saisissez un 'x' minuscule sur le clavierl et appuyez sur la touche d'entrée (vous devez faire cela encore assez souvent, à chaque fois qu'un tel message s'affiche ou pour arrêter un test...).

```
##########################################################################
#########            RP6 Robot Base Selftest                 #########
######### HOME VERSION                 v. 1.0 - 12.02.2007 #########
##########################################################################
#####       Main Menu          #########      Advanced Menu      #####
#                              #                                       #
# 0 - Run ALL Selftests (0-8)  # s - Move at speed Test               #
# 1 - PowerOn Test             # d - Move distance Test               #
# 2 - LED Test                 # c - Encoder Duty-Cycle Test          #
# 3 - Voltage Sensor Test      #                                      #
# 4 - Bumper Test              #                                      #
# 5 - Light Sensor Test        #                                      #
# 6 - ACS (and RC5 receive) Test #                                    #
# 7 - IRCOMM/RC5 Test          #                                      #
# 8 - Motors and Encoders Test #                                      #
#                              #                                      #
##########################################################################
# Please enter your choice (1-8, s, d, c)!                            #
##########################################################################
```

Le menu de texte ci-contre devrait s'afficher maintenant mais cela pourra encore changer d'ici la version définitive du logiciel.

Vous pouvez modifier et démarrer les programmes de test en saisissant le chiffre ou la lettre correspondants.

Nous voulons effectuer tous les tests standard – donc saisissez un '0' et appuyez sur Entrée!

Les indications suivantes doivent s'afficher sur le terminal:

```
# 0

##########################################################################
##########################################################################
## Test #1 ##

### POWER ON TEST ###
Please watch the yellow PowerOn LED and verify that it lights up!
(it will flash a few times!)
```

Observez maintenant la LED jaune PowerON sur le RP6. Elle devrait s'allumer et s'éteindre plusieurs fois de suite. Si ce n'est pas le cas, le test est soit déjà fini avant

que vous ayez eu le temps de regarder, soit il y a un défaut. Le programme de test continue cependant car il ne peut pas constater de lui-même si la LED fonctionne ou non. C'est à vous de le vérifier.

La LED indique d'ailleurs que les encodeurs, le récepteur IR et les détecteurs de courant sont sous tension. Ensemble avec la LED, ils consomment quand-même presque 10mA. C'est pourquoi tout ne s'allume qu'en cas de besoin pour économiser du courant.

Le programme allume toutes les LEDs d'état, plusieurs fois toutes ensembles et plusieurs fois individuellement. Cela vous permet de voir immédiatement si toutes les LEDs fonctionnent correctement ou sont endommagées. Sortie:

```
## Test #2 ##

### LED Test ###
Please watch the LEDs and verify that they all work!
Done!
```

Maintenant commencez le test du détecteur de courant des accus. Le détecteur avait en fait déjà été testé parce que le RP6Loader lit et affiche la tension des accus mais le voici encore une fois par acquis de conscience.

```
######################################################################
######################################################################
## Test #3 ##

### Voltage Sensor Test ###
Be sure that you are using good accumulators!

Enter "x" and hit return when you are ready!
```

Confirmez par 'x'!

```
# x
Performing 10 measurements:
Measurement #1: 07.20V --> OK!
Measurement #2: 07.20V --> OK!
Measurement #3: 07.20V --> OK!
Measurement #4: 07.20V --> OK!
Measurement #5: 07.20V --> OK!
Measurement #6: 07.20V --> OK!
Measurement #7: 07.20V --> OK!
Measurement #8: 07.20V --> OK!
Measurement #9: 07.20V --> OK!
Measurement #10: 07.20V --> OK!
Done!
```

La sortie devrait se présenter à peu près ainsi. Les valeurs de tension doivent se situer dans la plage de 5,5 à 9,5V pour que la valeur mesurée soit considérée comme normale. Si la valeur mesurée se situe en dehors de cette fourchette, un message d'erreur s'affiche. Dans ce cas, vérifiez les accus. Ils n'ont peut-être pas été chargés ou sont défectueux! Si les accus sont bons, c'est peut-être le « détecteur » (deux résistances...) qui est endommagé.

Maintenant nous allons tester les détecteurs de chocs. Pour cela, il vous suffit d'appuyer un peu sur les pare-chocs et d'observer les LEDs ou les indications sur le terminal. A chaque pression sur un des pare-chocs et à chaque relâchement, une indication devrait s'afficher sur le terminal et l'état des LEDs devrait changer. Cela pourrait ressembler à ceci:

```
## Test #4 ##

Bumper Test
Please hit both bumpers and verify
that both Bumpers are working properly!
The Test is running now. Enter "x" and hit return to stop this test!
OBSTACLE: LEFT!
FREE: LEFT!
OBSTACLE: RIGHT!
FREE: RIGHT!
OBSTACLE: LEFT!
FREE: LEFT!
OBSTACLE: RIGHT!
FREE: RIGHT!
OBSTACLE: LEFT!
OBSTACLE: RIGHT!
FREE: LEFT!
FREE: RIGHT!
```

Vous pouvez arrêter le test avec 'x' + Entrée si tout se passe bien!

Maintenant le test du détecteur photoélectrique! Vous devez couvrir les deux cellules photoélectriques sur le devant du robot avec les mains et vérifier si les valeurs mesurées changent en conséquence. Plus la valeur est basse, moins la cellule capte de lumière. A la lumière normale du jour, les valeurs devraient se situer entre 200 et 900.

Si vous vous approchez avec une lampe torche puissante que vous dirigez droit sur les détecteurs ou si vous exposez le robot à la lumière du soleil, les valeurs peuvent monter jusqu'à 1000. Si la pièce est très sombre, les valeurs devraient être inférieures à 100.

Au début du test, vous devez confirmer avec 'x':

```
## Test #5 ##

### Light Sensor Test ###
Please get yourself a small flashlight!
While the test runs, move it in front of the Robot
and watch if the values change accordingly!

Enter "x" and hit return when you are ready!
# x
The Test is running now. Enter "x" and hit return to stop this test!
Performing measurements...:
Left: 0510,  Right: 0680
Left: 0511,  Right: 0679
Left: 0512,  Right: 0680
Left: 0560,  Right: 0710
Left: 0630,  Right: 0750
Left: 0640,  Right: 0760
Left: 0644,  Right: 0765

[...]
```

Arrêtez le test avec 'x' lorsque vous avez vérifié les détecteurs.

Maintenant nous allons passer directement au test ACS. Il n'y a rien à confirmer, le test démarre immédiatement. Bougez une main ou un objet devant le robot tout en laissant suffisamment d'espace libre devant le robot pour que les détecteurs ne voient pas en permanence un obstacle.

L'affichage devrait ressembler à ceci:

```
## Test #6 ##

ACS Test
Please move your hand or other obstacles in front of the Robot
and verify that both ACS channels are working properly!

ACS is set to Medium power/range!

You can also send RC5 Codes with a TV Remote Control
to the RP6 - it will display the Toggle Bit, Device Adress
and Keycode of the RC5 Transmission!
Make sure your remote control transmits in RC5 and not
SIRCS or RECS80 etc.! There are several other formats that will NOT work!

The Test is running now. Enter "x" and hit return to stop this test!
OBSTACLE: LEFT!
FREE: LEFT!
OBSTACLE: LEFT!
FREE: LEFT!
OBSTACLE: LEFT!
OBSTACLE: RIGHT!
FREE: RIGHT!
FREE: LEFT!
OBSTACLE: LEFT!
OBSTACLE: RIGHT!
FREE: RIGHT!
FREE: LEFT!
```

Pendant le test, vous pouvez recevoir des codes d'une télécommande IR compatible RC5. Toggle bit, adresse et keycode s'afficheront alors.

Pour arrêter le test, saisissez 'X' et appuyez sur Enter.

Voici maintenant le test IRCOMM où il faut confirmer avec 'x' pour démarrer l'envoi de paquets de données IR. Toutes les données reçues sur le terminal sont automatiquement vérifiées (l'IRCOMM reçoit normalement aussi les signaux envoyés par lui-même puisque les diodes IR sont assez fortes. En l'absence totale d'objets réfléchissants ou bien si le robot est couvert, cela peut ne pas fonctionner mais c'est peu probable).

Les résultats du tests devraient se présenter à peu près comme ceci:

```
#### TEST #7 ####

IRCOMM Test
[...]

TX RC5 Packet: 0
RX RC5 Packet --> Toggle Bit:0 | Device Address:0 | Key Code:0 --> OK!
TX RC5 Packet: 3
RX RC5 Packet --> Toggle Bit:0 | Device Address:3 | Key Code:3 --> OK!
TX RC5 Packet: 6
RX RC5 Packet --> Toggle Bit:0 | Device Address:6 | Key Code:6 --> OK!
TX RC5 Packet: 9
RX RC5 Packet --> Toggle Bit:0 | Device Address:9 | Key Code:9 --> OK!
TX RC5 Packet: 12
RX RC5 Packet --> Toggle Bit:0 | Device Address:12 | Key Code:12 --> OK!
[...]
TX RC5 Packet: 57
RX RC5 Packet --> Toggle Bit:1 | Device Address:25 | Key Code:57 --> OK!
TX RC5 Packet: 60
RX RC5 Packet --> Toggle Bit:1 | Device Address:28 | Key Code:60 --> OK!
TX RC5 Packet: 63
RX RC5 Packet --> Toggle Bit:1 | Device Address:31 | Key Code:63 --> OK!

Test finished!
Done!
```

Le test ne devrait durer que 5 secondes.

Voici maintenant le test du moteur et des encodeurs! Vous devez à tout prix prendre le RP6 dans la main car les chenilles ne doivent toucher ni le sol, ni d'autres objets.

Sinon le test échouera probablement! Veillez à ce que le RP6 ne tombe pas de la table si vous voulez placer un objet en dessous comme indiqué auparavant.

Le test ne dure pas très longtemps – env. 30 secondes. Observez attentivement des messages d'erreurs éventuels pendant le test! Il peut arriver que des valeurs individuelles soient défaillantes et le test échoue pour cette raison. Donc, si les moteurs démarrent normalement et le test s'arrête en plein milieu, ce n'est pas obligatoirement une catastrophe. Effectuez le test encore une fois en lisant auparavant le chapitre sur les défaillances en annexe!

Pendant le test, l'entraînement passe par différents niveaux de vitesse jusqu'à env. 50% de la vitesse maximale et inverse également parfois le sens de rotation des moteurs. Pendant ce temps, les valeurs mesurées sont constamment surveillées par les encodeurs et les détecteurs de courant. Si un élément avait été endommagé pendant le transport (p.ex. un court-circuit dans un des moteurs ou un engrenage coincé ce que l'on aurait dû remarquer déjà dans le test avant l'insertion des accus) le courant mesuré sera très élevé et le test sera immédiatement interrompu.

Cela se présenterait à peu près ainsi (légèrement abrégé):

```
######################################################################
######################################################################
#### TEST #8 ####

Automatic speed speed regulation test

######################################################################
### ATTENTION!!! DANGER!!! WARNING!!!
Make sure that the RP6 can __NOT__ move!
The caterpillar tracks should __NOT__ touch the ground!
(hold it in your hands for example...)
THE RP6 WILL START MOVING FAST! YOU CAN DAMAGE IT IF YOU DO NOT
MAKE SURE THAT IT CAN __NOT__ MOVE!
Make sure both crawler tracks are FREE RUNNING! DO NOT BLOCK THEM!
--> OTHERWISE THE TEST WILL FAIL!
######################################################################

Enter "x" and hit return when TO START THIS TEST!
Make sure the RP6 can not move!

# x
T: 000 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 000 |IR: 003 |UB: 07.28V
T: 000 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 002 |IR: 003 |UB: 07.28V
[...]
Speed Left: OK
Speed Right: OK
T: 020 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 000 |IR: 003 |UB: 07.28V
T: 020 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 000 |IR: 003 |UB: 07.28V
```

```
T: 020 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 000 |IR: 003 |UB: 07.28V
T: 020 |VL: 000 |VR: 000 |PL: 020 |PR: 020 |IL: 006 |IR: 009 |UB: 07.26V
T: 020 |VL: 001 |VR: 014 |PL: 039 |PR: 030 |IL: 020 |IR: 020 |UB: 07.27V
[...]
Speed Left: OK
Speed Right: OK
T: 040 |VL: 021 |VR: 019 |PL: 037 |PR: 028 |IL: 025 |IR: 021 |UB: 07.25V
T: 040 |VL: 020 |VR: 020 |PL: 037 |PR: 029 |IL: 026 |IR: 022 |UB: 07.25V
T: 040 |VL: 018 |VR: 020 |PL: 044 |PR: 036 |IL: 028 |IR: 023 |UB: 07.23V
T: 040 |VL: 038 |VR: 038 |PL: 055 |PR: 044 |IL: 035 |IR: 029 |UB: 07.23V
T: 040 |VL: 037 |VR: 042 |PL: 055 |PR: 043 |IL: 033 |IR: 028 |UB: 07.24V
T: 040 |VL: 043 |VR: 041 |PL: 052 |PR: 042 |IL: 032 |IR: 026 |UB: 07.23V
T: 040 |VL: 043 |VR: 041 |PL: 052 |PR: 040 |IL: 030 |IR: 024 |UB: 07.24V
T: 040 |VL: 037 |VR: 041 |PL: 052 |PR: 040 |IL: 030 |IR: 023 |UB: 07.24V
T: 040 |VL: 043 |VR: 040 |PL: 050 |PR: 039 |IL: 029 |IR: 022 |UB: 07.24V
Speed Left: OK
Speed Right: OK
T: 060 |VL: 040 |VR: 039 |PL: 053 |PR: 040 |IL: 033 |IR: 024 |UB: 07.24V
T: 060 |VL: 036 |VR: 040 |PL: 053 |PR: 040 |IL: 034 |IR: 026 |UB: 07.24V
T: 060 |VL: 042 |VR: 039 |PL: 052 |PR: 041 |IL: 034 |IR: 027 |UB: 07.23V
T: 060 |VL: 042 |VR: 040 |PL: 063 |PR: 052 |IL: 038 |IR: 032 |UB: 07.22V
T: 060 |VL: 058 |VR: 060 |PL: 068 |PR: 056 |IL: 038 |IR: 032 |UB: 07.25V
T: 060 |VL: 062 |VR: 062 |PL: 067 |PR: 054 |IL: 037 |IR: 029 |UB: 07.22V
T: 060 |VL: 060 |VR: 062 |PL: 067 |PR: 053 |IL: 038 |IR: 028 |UB: 07.23V

[...]

Speed Left: OK
Speed Right: OK
T: 100 |VL: 082 |VR: 078 |PL: 080 |PR: 068 |IL: 043 |IR: 036 |UB: 07.23V
T: 100 |VL: 079 |VR: 079 |PL: 081 |PR: 069 |IL: 047 |IR: 038 |UB: 07.22V
T: 100 |VL: 078 |VR: 082 |PL: 092 |PR: 078 |IL: 049 |IR: 039 |UB: 07.23V
T: 100 |VL: 095 |VR: 099 |PL: 101 |PR: 082 |IL: 055 |IR: 039 |UB: 07.20V
T: 100 |VL: 098 |VR: 100 |PL: 109 |PR: 081 |IL: 056 |IR: 040 |UB: 07.19V
T: 100 |VL: 095 |VR: 099 |PL: 111 |PR: 082 |IL: 062 |IR: 042 |UB: 07.19V
T: 100 |VL: 102 |VR: 101 |PL: 111 |PR: 082 |IL: 058 |IR: 041 |UB: 07.21V
T: 100 |VL: 102 |VR: 101 |PL: 109 |PR: 081 |IL: 056 |IR: 039 |UB: 07.20V
T: 100 |VL: 093 |VR: 100 |PL: 113 |PR: 081 |IL: 063 |IR: 038 |UB: 07.20V
T: 100 |VL: 104 |VR: 099 |PL: 112 |PR: 082 |IL: 056 |IR: 042 |UB: 07.22V
Speed Left: OK
Speed Right: OK
T: 080 |VL: 086 |VR: 071 |PL: 022 |PR: 000 |IL: 020 |IR: 012 |UB: 07.28V
T: 080 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 001 |IR: 003 |UB: 07.28V
T: 080 |VL: 004 |VR: 011 |PL: 088 |PR: 084 |IL: 051 |IR: 045 |UB: 07.21V
T: 080 |VL: 079 |VR: 101 |PL: 103 |PR: 077 |IL: 064 |IR: 039 |UB: 07.21V
T: 080 |VL: 082 |VR: 076 |PL: 098 |PR: 072 |IL: 061 |IR: 041 |UB: 07.19V
T: 080 |VL: 081 |VR: 081 |PL: 096 |PR: 071 |IL: 055 |IR: 040 |UB: 07.20V
T: 080 |VL: 080 |VR: 082 |PL: 095 |PR: 070 |IL: 057 |IR: 038 |UB: 07.21V
T: 080 |VL: 082 |VR: 080 |PL: 094 |PR: 069 |IL: 058 |IR: 036 |UB: 07.22V
T: 080 |VL: 077 |VR: 080 |PL: 095 |PR: 069 |IL: 056 |IR: 036 |UB: 07.23V
Speed Left: OK
Speed Right: OK
T: 060 |VL: 082 |VR: 079 |PL: 095 |PR: 069 |IL: 054 |IR: 038 |UB: 07.22V
T: 060 |VL: 079 |VR: 079 |PL: 095 |PR: 071 |IL: 058 |IR: 040 |UB: 07.21V
T: 060 |VL: 082 |VR: 081 |PL: 093 |PR: 070 |IL: 056 |IR: 039 |UB: 07.19V
T: 060 |VL: 069 |VR: 070 |PL: 080 |PR: 054 |IL: 048 |IR: 029 |UB: 07.23V
T: 060 |VL: 064 |VR: 059 |PL: 075 |PR: 054 |IL: 046 |IR: 029 |UB: 07.22V
T: 060 |VL: 058 |VR: 057 |PL: 075 |PR: 055 |IL: 043 |IR: 032 |UB: 07.24V
T: 060 |VL: 059 |VR: 059 |PL: 075 |PR: 056 |IL: 046 |IR: 034 |UB: 07.23V
T: 060 |VL: 060 |VR: 059 |PL: 075 |PR: 056 |IL: 046 |IR: 035 |UB: 07.23V
T: 060 |VL: 057 |VR: 060 |PL: 076 |PR: 056 |IL: 047 |IR: 033 |UB: 07.22V
T: 060 |VL: 058 |VR: 061 |PL: 077 |PR: 055 |IL: 045 |IR: 030 |UB: 07.23V
Speed Left: OK
Speed Right: OK
T: 040 |VL: 045 |VR: 035 |PL: 043 |PR: 023 |IL: 027 |IR: 018 |UB: 07.24V
T: 040 |VL: 000 |VR: 000 |PL: 011 |PR: 000 |IL: 013 |IR: 007 |UB: 07.28V
T: 040 |VL: 002 |VR: 000 |PL: 038 |PR: 038 |IL: 015 |IR: 014 |UB: 07.24V
T: 040 |VL: 038 |VR: 061 |PL: 059 |PR: 052 |IL: 035 |IR: 035 |UB: 07.24V
T: 040 |VL: 044 |VR: 043 |PL: 057 |PR: 044 |IL: 035 |IR: 028 |UB: 07.23V
T: 040 |VL: 038 |VR: 039 |PL: 057 |PR: 044 |IL: 035 |IR: 027 |UB: 07.24V
T: 040 |VL: 039 |VR: 042 |PL: 055 |PR: 043 |IL: 033 |IR: 025 |UB: 07.23V
T: 040 |VL: 043 |VR: 041 |PL: 053 |PR: 041 |IL: 032 |IR: 023 |UB: 07.24V
T: 040 |VL: 040 |VR: 041 |PL: 054 |PR: 041 |IL: 032 |IR: 023 |UB: 07.25V
Speed Left: OK
Speed Right: OK
```

```
T: 020 |VL: 037 |VR: 040 |PL: 054 |PR: 041 |IL: 031 |IR: 024 |UB: 07.24V
T: 020 |VL: 022 |VR: 019 |PL: 022 |PR: 012 |IL: 017 |IR: 016 |UB: 07.28V
T: 020 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 004 |IR: 007 |UB: 07.28V
T: 020 |VL: 000 |VR: 006 |PL: 030 |PR: 027 |IL: 020 |IR: 020 |UB: 07.24V
T: 020 |VL: 013 |VR: 019 |PL: 043 |PR: 030 |IL: 029 |IR: 022 |UB: 07.24V
T: 020 |VL: 026 |VR: 020 |PL: 038 |PR: 029 |IL: 027 |IR: 022 |UB: 07.24V
T: 020 |VL: 020 |VR: 021 |PL: 038 |PR: 029 |IL: 028 |IR: 023 |UB: 07.25V
T: 020 |VL: 021 |VR: 020 |PL: 038 |PR: 029 |IL: 028 |IR: 023 |UB: 07.24V
T: 020 |VL: 018 |VR: 019 |PL: 038 |PR: 030 |IL: 027 |IR: 024 |UB: 07.24V
T: 020 |VL: 022 |VR: 020 |PL: 037 |PR: 029 |IL: 027 |IR: 023 |UB: 07.23V
Speed Left: OK
Speed Right: OK

***** MOTOR AND ENCODER TEST OK! *****
```

Les valeurs individuelles qui sont affichées pendant les tests sont (de gauche à droite): T – valeur de consigne de la vitesse actuelle, VL/VR – vitesse mesurée gauche/droite, PL/PR – Valeur MLI gauche/droite, IL/IR – Courant moteur gauche/droite, UB – tension des accus.

Si les indications ***ressemblent*** à ce qui précède, tout va bien.

S'il y a un dysfonctionnement ou des messages d'erreur s'affichent, lisez le chapitre concernant la résolution de problèmes en annexe.

Voilà ce qui termine le programme de test. Si tout s'est déroulé comme prévu, vous pouvez passer directement au chapitre suivant.

4. Programmation du RP6

Maintenant nous arrivons enfin à la programmation du robot.

4.1. Installation de l'Editeur de Texte Source

Tout d'abord, nous devons installer un environnement de développement. Il faut bien que nous arrivions à rentrer le « texte source » (appelé également code source ou en anglais 'source code') pour nos programmes en 'C' dans l'ordinateur!

A cet effet, nous n'allons surtout pas utiliser des logiciels comme OpenOffice ou Word. Nous le précisons car ce n'est pas évident pour tout le monde. Ces logiciels sont parfaits pour écrire des manuels tels que celui-ci mais ne conviennent absolument pas à la programmation. Un texte source est un texte pur, sans aucun formatage. La taille de caractère et la police n'intéressent pas du tout le compilateur...

Pour une personne, c'est nettement plus clair si certains mots-clés ou types de textes sont automatiquement mis en évidence par des couleurs. L'éditeur de texte que nous allons utiliser - « Programmers Notepad 2 » - (appelé « PN2 » par la suite), offre cette fonction (appelée coloration syntaxique) et encore bien d'autres. (*ATTENTION: Sous Linux vous devez utiliser un autre éditeur qui offre des fonctions similaires au PN2. Généralement, plusieurs éditeurs sont déjà pré-programmés tels que kate, gedit, ex-macs ou autres).* Outre la mise en exergue des mots-clés et autres, il existe aussi une gestion rudimentaire de projets. Il est même possible d'organiser plusieurs fichiers de texte source en projets et afficher une liste de tous les fichiers appartenant à un même projet. Par ailleurs, il est facile d'appeler des programmes tels que l'AVR-GCC avec le PN2 et faire traduire les programmes par des commandes de menu. L'AVR-GCC est normalement un pur programme en ligne de commande sans environnement graphique...

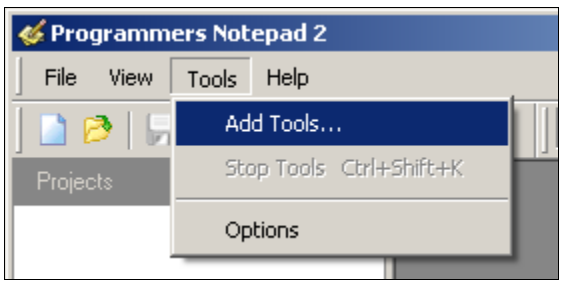
Vous trouverez des versions plus récentes du PN2 sur la page d'accueil:

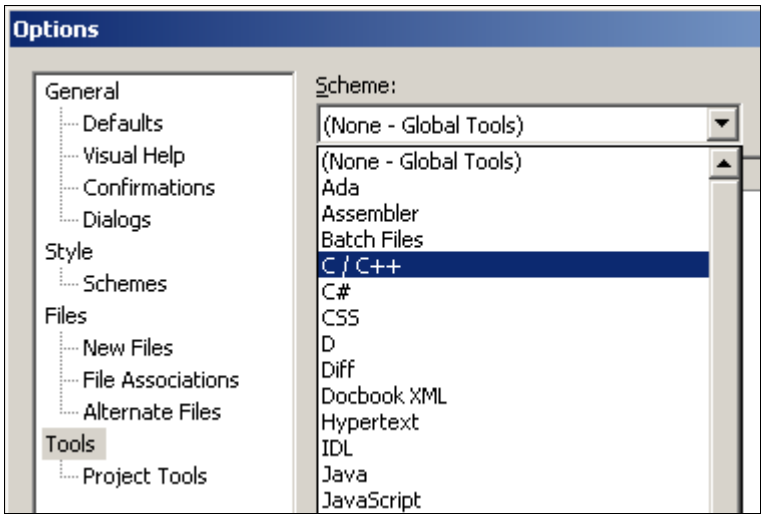
`http://www.pnotepad.org/`

4.1.1. Créer des Commandes dans un Menu

ATTENTION: Vous pouvez laisser ce paragraphe de côté si les commandes existent déjà dans le PN2 (les commandes de menu s'appellent alors „[WinAVR] Make All", etc.. regardez tout simplement dans le menu). Comme ce n'est pas le cas sur toutes les versions et qu'il est toujours utile de savoir comment il faut procéder (vous pouvez aussi insérer d'autres programme que le RP6Loader dans ce menu!), nous donnons ici une rapide explication sur la manière d'ajouter des commandes dans un menu.

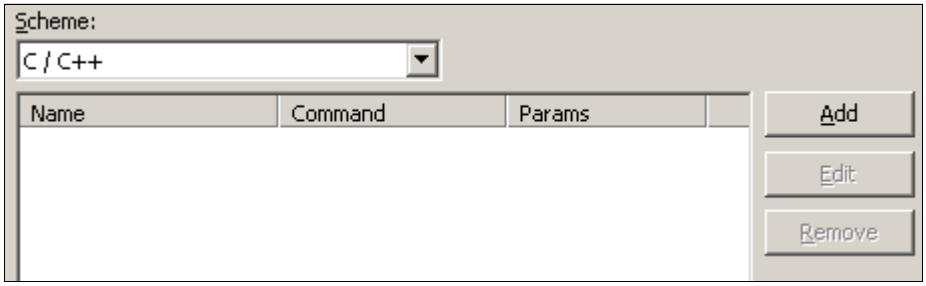
Démarrez PN2 et choisissez dans le menu « Tools » la commande « Add Tools... » (voir impression écran).



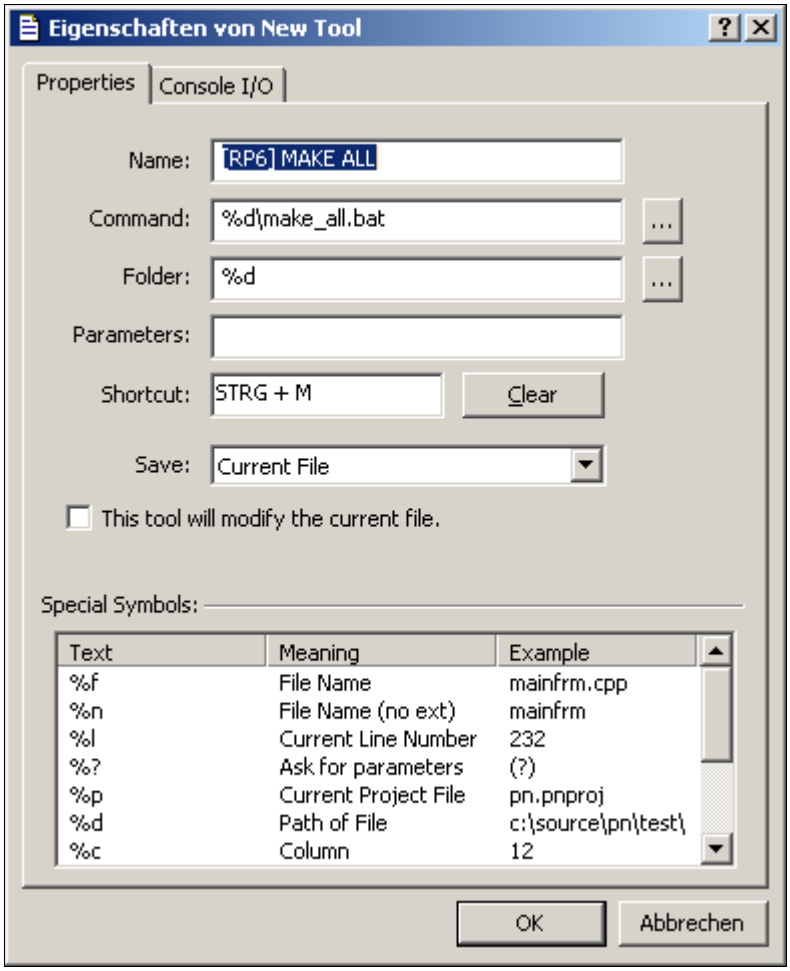


La fenêtre de dialogue « Options » s'affiche où vous pouvez modifier différents paramètres du PN2. Nous ne voulons cependant qu'ajouter une commande dans le menu 'Tools'.

Pour cela, sélectionnez dans le menu déroulant „Schemes:" l'option „C/C++" !



Cliquez sur le bouton „Add"!



La fenêtre ci-contre s'affiche.

Saisissez exactement ce que vous voyez sur l'impression écran ci-contre.

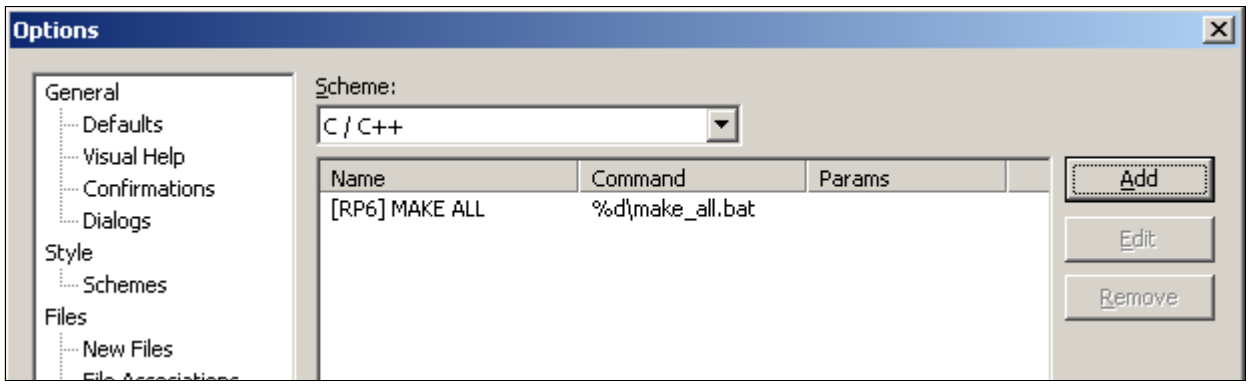
„%d" renvoie au répertoire du fichier actuellement sélectionné et „%d\make_all.bat" renvoie à un petit fichier batch qui se trouve dans chaque répertoire des projets préparés pour le RP6.

Dans le champs « Shortcut », vous pouvez p.ex. taper [STRG] + [M] sur le clavier mais c'est une option.

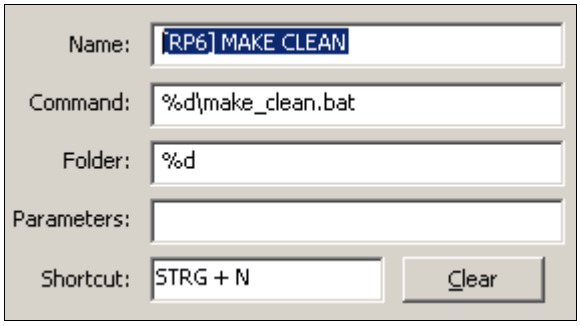
Cette commande appelle l'outil « make » via le fichier batch « make_all.bat » et déclenche ainsi la compilation des fichiers dans le répertoire du fichier actuellement sélectionné mais nous y reviendrons plus tard.

Au lieu de saisir „%d/make_all.bat" il suffit d'écrire « make » dans le champs « Command » et « all » dans les paramètres.

Cliquez sur OK et une nouvelle commande apparaît dans la liste:



...cliquez à nouveau sur « Add »!



Saisissez maintenant les mêmes inscriptions comme dans la fenêtre ci-contre et cliquez sur OK.

Une autre commande apparaît dans la liste:

„[RP6] MAKE CLEAN"

Cette commande permet d'effacer rapidement tous les fichiers temporaires que le compilateur a généré pendant son travail puisque nous n'en aurons plus besoin après la compilation. Le fichier hex généré ne sera d'ailleurs pas effacé et pourra toujours être rechargé dans le robot.

Comme ci-dessus, il existe une alternative en saisissant simplement « make » dans le champs « Command » au lieu de „%d/make_clean.bat" et „clean" dans le champs « Parameters ».

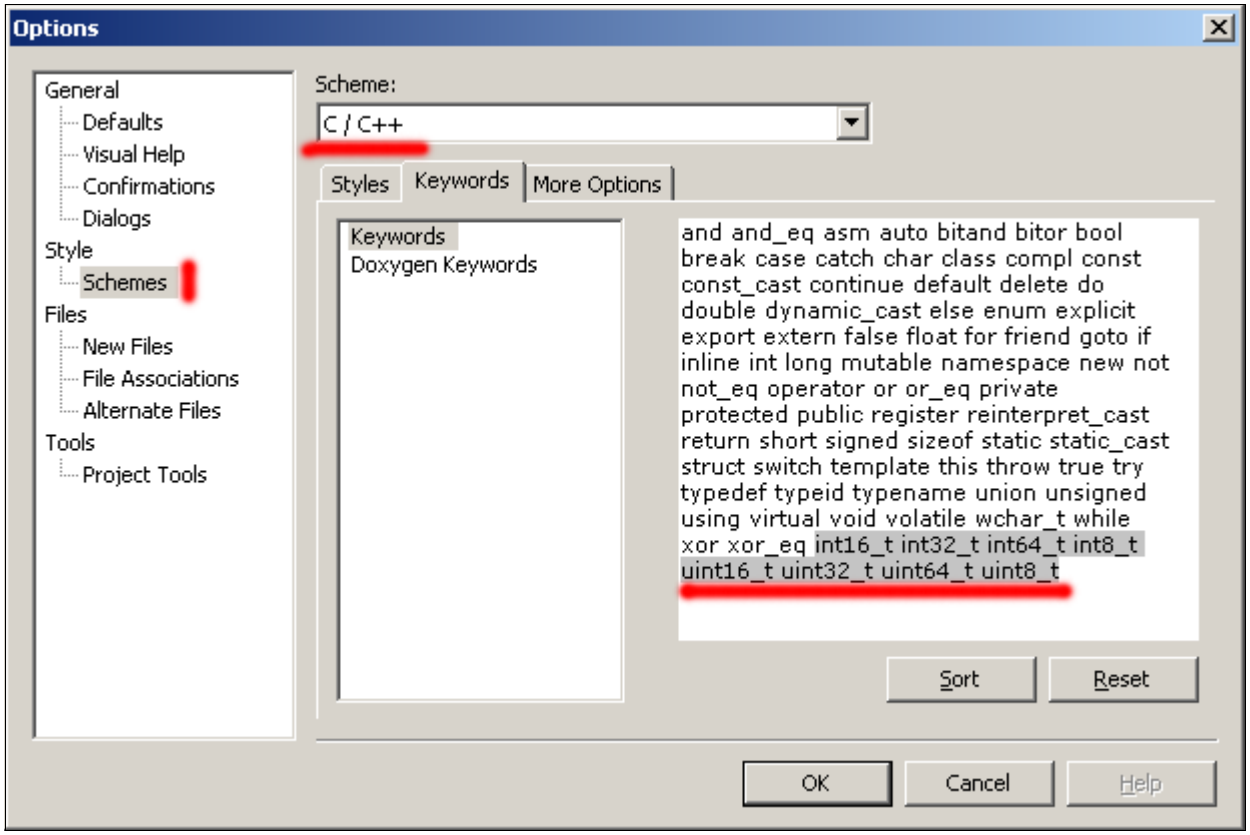
Pour finir, vous devez confirmer les modification dans la fenêtre de dialogue « Options » en cliquant sur OK!

4.1.2. Réglage de la Coloration Syntaxique

Un autre réglage que vous devriez modifier, concerne la coloration syntaxique. Il existe encore quelques « Keywords » (=mots-clé) qu'il faut ajouter au schéma C/C++ normal.

Il s'agit des mots-clé suivants (vous pouvez procéder directement par Copier/Coller ([STRG]+[C] = copier, [STRG]+[V] = coller) dans le champs de dialogue):

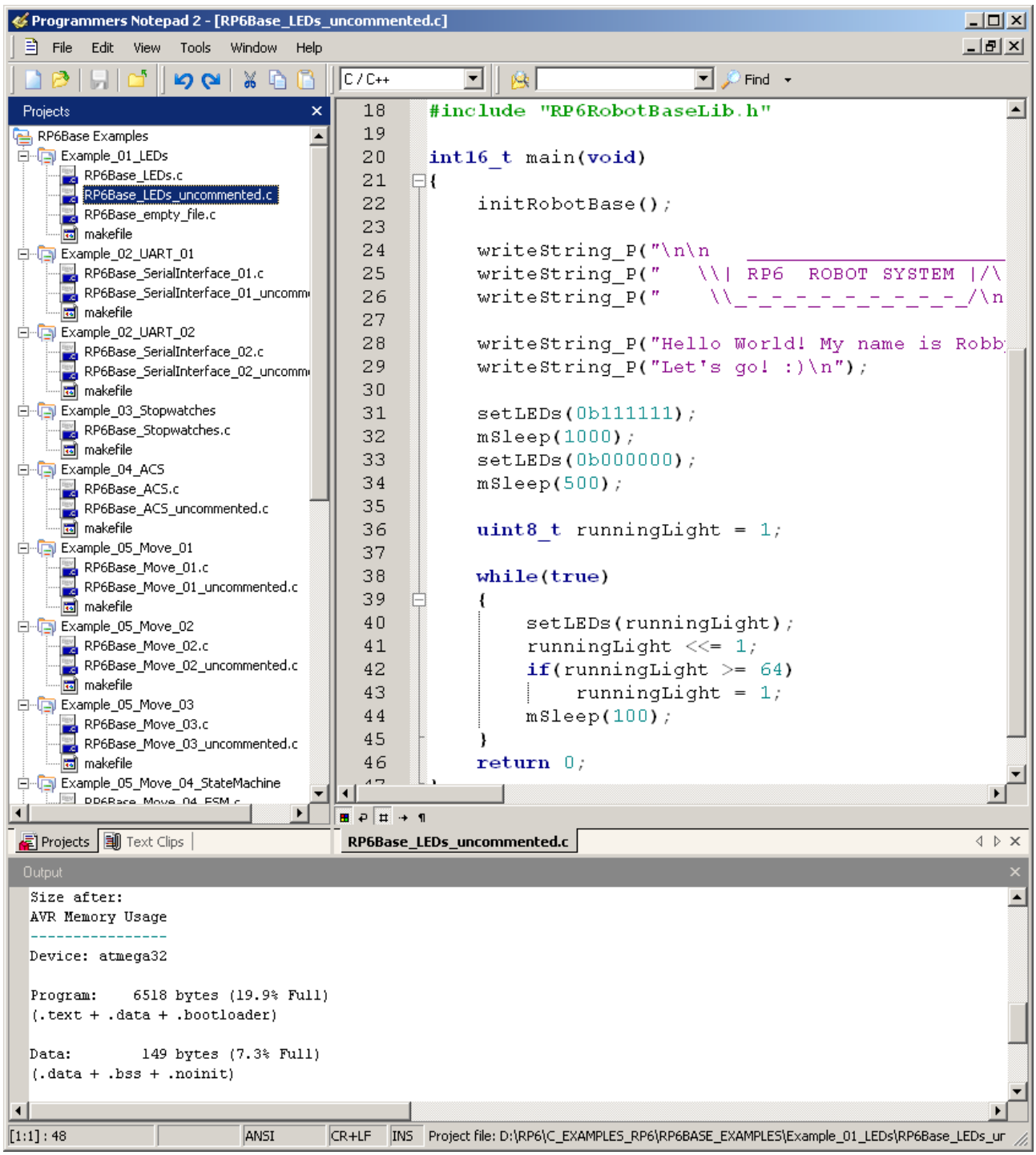
int8_t int16_t int32_t int64_t uint8_t uint16_t uint32_t uint64_t



Cliquez une fois sur « Sort » (trier) et confirmez le dialogue avec OK!

Attention: *Sur les versions plus récentes de WinAVR et Programmers Notepad (WinAVR-20070525), ces mots-clé sont déjà enregistrés et vous n'avez plus rien à modifier. Dans cette version, Programmers Notepad se présente aussi d'une façon un peu différente que sur ces impressions écran.*

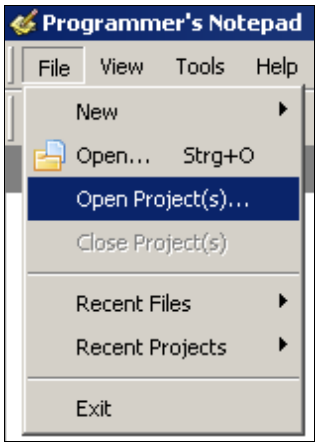
Lorsque l'installation est terminée et après avoir ouvert les projets-exemples décrits *dans le paragraphe suivant*, le PN2 devrait se présenter à peu près comme ceci:



Tous les exemples de projets figurent à gauche, l'éditeur de texte (avec la coloration syntaxique) à droite et en-dessous l'édition des Tools (dans notre cas l'édition du compilateur).

Vous pouvez modifier encore beaucoup d'autres choses dans le PN2 qui offre de nombreuses autres fonctions utiles.

4.1.3. Ouverture et Compilation d'un Exemple de Projet



Nous allons maintenant effectuer un test de fonctionnement en ouvrant les exemples de projets:

Dans le menu « File » sélectionnez la commande « Open Project(s) ».

Un dialogue de sélection de fichier normal s'affiche. Recherchez le dossier « RP6Base_Examples\ » dans le classeur dans lequel vous avez enregistré les exemples de programmes.

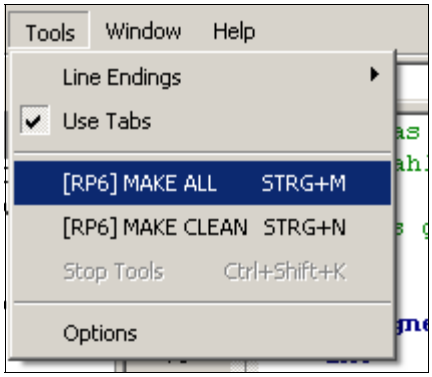
Ouvrez maintenant le fichier « RP6BaseExamples.ppg ». C'est un groupe de projets pour le PN2 qui charge tous les exemples de programmes ainsi que la RP6Library dans la liste des projets (« Projects »). Ainsi tous les exemples de programme sont toujours à portée de la main pour pouvoir s'y référer au début ou consulter la RP6Library etc..

Ouvrez maintenant le premier exemple de programme tout en haut de la liste (Sélectionnez « Example_01_LEDs » et le fichier « RP6Base_LEDs.c ») qui se trouve sur le côté gauche de la fenêtre en cliquant tout simplement deux fois sur « RP6Base_LEDs.c ». Un éditeur de texte s'affiche dans une fenêtre à l'intérieur du programme.

En bas de la fenêtre du PN2, une zone d'édition devrait apparaître. Si ce n'est pas le cas, vous devez activer cette zone avec le menu « View » -> « Output » OU si la zone est trop petite, l'agrandir avec la souris en l'étirant (le curseur de la souris se transforme en une double-flèche si vous la placez en bas de la fenêtre du programme sur le bord supérieur de la zone grise dans lequel est marqué « Output »).

Vous pouvez jeter un coup d'œil rapide sur l'éditeur de texte que vous venez d'ouvrir mais vous n'êtes pas obligé de comprendre ce qui s'y passe exactement. Nous verrons cela ultérieurement. Sachez seulement que le texte en vert contient des commentaires qui ne font pas partie du programme proprement dit et qui ne servent qu'à la description/documentation. Nous y reviendrons plus tard en détail (il existe aussi une version de ce programme SANS commentaires qui montre combien le programme est court. Ce sont les commentaires qui le rallongent mais ils sont nécessaires à la compréhension. La version sans commentaires est pratique pour copier le code dans vos propres programmes!).

Pour l'instant nous voulons juste savoir si la traduction des programmes fonctionne correctement.



Le menu « Tools » devrait contenir les deux commandes créées auparavant (voir. Fig.) (ou bien les commandes [WinAVR] standard du PN, cela n'a pas d'importance, les deux fonctionnent).

Cliquez maintenant sur „MAKE ALL"!

Le PN2 appelle maintenant le fichier batch „make_all.bat" mentionné auparavant qui appelle à son tour le

programme « make » sur lequel nous reviendrons ultérieurement .

Le programme est maintenant traduit (ce qui s'appelle « compiler » du terme anglais « to compile » ou « Compiler » = « Traducteur/compilateur »). Un fichier hex est généré qui contient le programme dans sa forme traduite pour le microcontrôleur. Il pourra y être chargé et exécuté plus tard. Pendant la compilation, un grand nombre de fichiers temporaires sont générés (avec des terminaisons telles que « .o, .lss, .map, .sym, .elf, .dep »). Vous pouvez les ignorer *tous*. L'outil fraîchement créé « make clean » permet de les effacer facilement. Seul le fichier hex est intéressant pour vous et il ne sera d'ailleurs pas effacé par la commande « make clean ».

Après la commande MAKE ALL, l'édition suivante devrait s'afficher (largement raccourci cependant! Certaines lignes peuvent être bien sûr un peu différentes):

```
> "make" all
-------- begin --------

[...]

Compiling: RP6Base_LEDs.c

avr-gcc -c -mmcu=atmega32 -I. -gdwarf-2   -Os -funsigned-char -funsigned-bitfields -fpack-struct
-fshort-enums -Wall -Wstrict-prototypes -Wa,-adhlns=RP6Base_LEDs.lst -I../../RP6lib
-I../../RP6lib/RP6base -I../../RP6lib/RP6common -std=gnu99 -MD -MP -MF .dep/RP6Base_LEDs.o.d RP6-
Base_LEDs.c -o RP6Base_LEDs.o

Compiling: ../../RP6lib/RP6base/RP6RobotBaseLib.c

[...]

Creating load file for Flash: RP6Base_LEDs.hex
avr-objcopy -O ihex -R .eeprom RP6Base_LEDs.elf RP6Base_LEDs.hex

[...]
Size after:
AVR Memory Usage

----------------

Device: atmega32
Program:    6858 bytes (20.9% Full)
(.text + .data + .bootloader)

Data:        148 bytes (7.2% Full)
(.data + .bss + .noinit)
-------- end --------
> Process Exit Code: 0
> Time Taken: 00:01
```

Il est important que „`Process Exit Code: 0`“ apparaisse en bas. Cela signifie qu'il n'y a eu

aucune erreur pendant la compilation. Si un autre code apparaît, il y a une erreur dans le code source qu'il faut corriger sinon cela ne fonctionnera pas. Le compilateur affiche dans ce cas différents messages d'erreur en haut du texte qui donnent plus d'informations.

Veuillez noter aussi que « Process Exit Code: 0 » n'est pas synonyme d'un programme 100% sans erreurs! Evidemment le compilateur ne trouve pas d'erreurs de réflexion dans votre programme et il ne peut pas non plus empêcher que le robot rentre dans un mur ☺.

IMPORTANT: Des avertissements et autres informations très utiles peuvent figurer au début du texte qui signalent presque toujours des problèmes importants qu'il faut absolument résoudre! Le PN2 met en évidence des avertissements et erreurs par des couleurs afin de pouvoir les identifier facilement. Le numéro de ligne contenant le défaut détecté par le compilateur est également indiqué. Si vous cliquez sur ce message en couleur, le PN2 saute directement sur la ligne dans l'éditeur concerné.

L'indication finale „AVR Memory Usage" est également très utile.

```
Size after:
AVR Memory Usage

----------------


Device: atmega32


Program:    6858 bytes (20.9% Full)
(.text + .data + .bootloader)


Data:        148 bytes (7.2% Full)
(.data + .bss + .noinit)
```

Cela signifie que notre programme fait 6858 octets et que 148 octets de RAM sont réservés pour des variables statiques (à cela s'ajoutent les zones dynamiques pour Heap (tas) et Stack (pile), mais cela nous mènerait trop loin... réservez tout simplement toujours quelques centaines d'octets de mémoire). Nous avons en tout 32 Ko (32768 octets) de Flash ROM et 2Ko (2048 octets) de RAM. Sur les 32Ko, 2 Ko sont occupés par le chargeur d'amorçage – donc il ne nous reste que 30 Ko. Veillez toujours à ce que le programme rentre dans l'espace mémoire disponible! (Le RP6Loader ne transfère pas le programme s'il est trop volumineux).

Dans le cas du programme ci-dessus, il reste encore 23682 octets. Le programme RP6Base_LED.c assez court en soi atteint cette taille parce que la RP6Library est incluse. Donc, ne vous inquiétez pas, il y a assez d'espace pour vos programmes et de si petits programmes ne nécessitent normalement pas beaucoup de mémoire. La bibliothèque des fonctions occupe à elle seule déjà plus de 6,5 Kb de la mémoire flash, mais elle vous évite aussi beaucoup de travail et c'est pour cela que vos propres programmes seront relativement petits comparés à la RP6Library.

4.2. Chargement de Programmes dans le RP6

Chargez maintenant le programme que vous venez de compiler, dans le robot à l'aide du RP6Loader.

A cet effet, vous insérez le fichier hex que vous venez de générer, dans la liste du RP6Loader avec « Add » ou « Hinzufügen » (=ajouter), vous le sélectionnez et vous cliquez sur le bouton « Upload! » exactement comme vous l'avez déjà fait pour le programme d'auto-test. Ensuite vous pouvez revenir sur le terminal et regarder la sortie du programme. Il faut évidemment démarrer l'exécution du programme en appuyant sur les touches [STRG] + [S] du clavier du terminal ou en utilisant le menu (ou en envoyant simplement un « s ». Après un Reset, vous devez toujours attendre le message « [READY] » sur le terminal! [STRG] + [Y] est également une combinaison de touches très utile puisqu'elle charge le programme sélectionné dans le RP6 et le démarre immédiatement. Ce n'est donc pas la peine de revenir du terminal sur l'onglet « Flash Loader » ou d'utiliser le menu.

Le programme est très simple et ne comprend qu'un petit séquenceur à LED et un peu de texte.

Avant de pouvoir écrire vos propres programmes, voici un petit cours intensif du langage C...

4.3. Pourquoi C? Et que signifie „GCC"?

Le langage de programmation C est très largement répandu. C'est LE langage standard que tout ceux qui s'intéressent au développement de logiciels, auront certainement déjà utilisé un jour (ou des langages à la syntaxe similaire). Il existe au moins un compilateur C pour chaque microcontrôleur actuellement disponible. C'est la raison pour laquelle tous les robots récents d'AREXX Engineering (actuellement ASURO, YETI et le RP6) sont programmables en C.

Puisque C est très largement répandu, la documentation sur Internet et la littérature sont abondantes. Cela facilite bien sûr la tâche aux débutants même si C est déjà un langage relativement complexe qui ne s'apprend pas en deux, trois jours sans avoir de solides connaissances de base... Donc, ne jetez pas tout de suite le robot par la fenêtre si cela ne fonctionne pas du premier coup :-)).

Heureusement, les bases sont faciles à comprendre et on peut élargir et améliorer ses connaissances continuellement. Cela demande cependant une certaine initiative personnelle! C ne s'apprend pas tout seul – comme toutes les langues étrangères normales.

Lorsque l'on commence à maîtriser un peu le langage C, l'initiation aux autres langages de programmation se trouve facilitée puisqu'ils utilisent souvent des concepts très similaires.

Tout comme pour nos autres robots, le RP6 fait appel à une version spéciale du compilateur C de la GNU Compiler Collection ou GCC. Le GCC est un système de compilation universel qui supporte des langages très différents. Ainsi, il permet aussi la traduction de textes sources rédigés en C++, Java, Ada et FORTRAN.

Le GCC supporte non seulement l'AVR mais a été développé en fait pour des systèmes bien plus grands et connait plusieurs douzaines de systèmes-cibles différents.

Le projet le plus connu pour lequel a été utilisé GCC est évidemment Linux. Presque tous les programmes d'application qui tournent sous Linux, ont également été compilés avec le GCC. Il s'agit donc ici d'un outil très au point et professionnel qui est utilisé dans de nombreuses grandes sociétés.

D'ailleurs, lorsque nous parlons de GCC, nous ne pensons pas à la collection de compilation complète mais presque toujours au compilateur C. Initialement GCC ne signifiait que « GNU C Compiler ». La signification la plus récente est devenue nécessaire lorsque d'autres langages se sont rajoutés.

Si vous voulez en savoir davantage sur le GCC, vous pouvez visiter le site officiel: `http://gcc.gnu.org/`

Le GCC ne supporte pas directement l'AVR mais doit d'abord être adapté. Cette version du GCC s'appelle alors AVR-GCC. Pour les utilisateurs de Windows, ce compilateur est compris prêt à l'emploi dans WinAVR. Les utilisateurs de Linux doivent souvent faire la traduction eux-même, ce que vous avez déjà dû faire normalement.

4.4. Cours intensif de C pour Débutants

Ce chapitre constitue une introduction très succincte dans la programmation en C. Nous n'abordons ici que les points absolument indispensables pour le RP6. Ce chapitre ainsi qu'un grand nombre de exemples de programmes sont davantage un aperçu de tout ce qui existe et ce que l'on peut faire. Nous donnons des exemples et expliquons les bases mais il appartient au lecteur de s'y plonger davantage.

Ce n'est donc pas plus qu'une petite leçon d'initiation! Une initiation complète dépasserait le cadre de ce *mode d'emploi* et remplit normalement de gros ouvrages spécialisés. Heureusement il en existe beaucoup dont certains[1] sont même disponibles sur Internet. Voici donc juste un petit aperçu...

4.4.1. Littérature

Les livres et tutoriels suivants traitent de la programmation en C, surtout pour le PC et autres « grands » ordinateurs. *Beaucoup de sujets abordés dans ces livres n'existent pas pour le microcontrôleur AVR. Le langage est le même mais la plupart des bibliothèques utilisables sur un PC sont tout simplement trop grandes pour un microcontrôleur. Le meilleur exemple sont des fonctions telles que « printf » - une banalité sur un PC! Cette fonction existe certes aussi pour des microcontrôleurs mais elle nécessite beaucoup d'espace et de temps de calcul et vaut mieux être évitée. Nous parlerons ultérieurement d'alternatives bien plus efficaces pour nos objectifs.*

Certains tutoriels C / Livres en lignes (juste une sélection minuscule):

`http://www.galileocomputing.de/openbook/c_von_a_bis_z/`

`http://de.wikibooks.org/wiki/C-Programmierung`

`http://suparum.rz.uni-mannheim.de/manuals/c/cde.htm`

`http://www.roboternetz.de/wissen/index.php/C-Tutorial`

`http://www.its.strath.ac.uk/courses/c/`

Il en existe beaucoup d'autres livres sur le marché. Il suffit d'aller dans une librairie bien achalandée et vous y trouverez certainement ce que vous cherchez.

C'est inutile de s'acheter un livre pour réaliser quelques expériences avec le robot. Vous apprendrez de toute façon une grande partie au fur et à mesure de l'utilisation. Toutes les informations nécessaires se trouvent sur les pages indiquées sur Internet, et les exemples de programmes fournis sur le CD sont également assez complets.

1 *Une recherche dans un moteur de recherche renommée sur « c tutorial » donne plusieurs millions de réponses! Evidemment il n'y en a pas autant mais il y en a quand-même quelques-uns.*

Pour l'AVR-GCC et les microcontrôleurs AVR, il existent de très bons sites.

Par ailleurs, vous pouvez également aller sur la page d'accueil et documentation WinAVR:

http://winavr.sourceforge.net/

http://winavr.sourceforge.net/install_config_WinAVR.pdf

ainsi que la documentation AVR-LibC:

http://www.nongnu.org/avr-libc/user-manual/index.html

Ce n'est pas nécessaire de lire tous les tutoriels et livres! Tous ne sont pas aussi complets et beaucoup d'entre eux traitent de sujets très différents. Cependant, cela vaut la peine d'en lire quelques-uns.

4.4.2. Premier Exemple de Programme

En effet, apprendre en pratiquant est le meilleur moyen d'aborder le langage C. Si vous avez lu et compris certaines leçons, vous devez les mettre en pratique.

Il faut évidemment éclaircir certains points mais pour vous donner tout de suite une idée ce dont nous parlons, nous allons examiner un petit programme typique en C pour le RP6.

```
1  /*
2   * Un petit programme en C „Salut le Monde“ pour le RP6!
3   */
4
5  #include "RP6RobotBaseLib.h"
6
7  int main(void)
8  {
9        initRobotBase();
10       writeString("Salut le Monde!\n");
11       return 0;
12 }
```

Si vous n'avez encore jamais programmé en C, cela paraît comme une langue étrangère à première vue (et c'est le cas!) mais les bases sont vraiment très faciles à comprendre. Le langage C a été développé dans des pays anglophones[2] et c'est pourquoi toutes les commandes sont en anglais. Ce n'est pas seulement le cas pour le langage

2 ... Plus exactement au début des années 1970 aux USA où il a servi de base de développement au système d'exploitation UNIX. Ensuite de nombreux améliorations et compléments ont été apportés...

C mais s'applique à presque tous les langages de programmation.

Concernant les exemples de programmes plus importants, nous nous y tenons rigoureusement et donnons à toutes les fonctions et variables des désignations en anglais. Dans ce manuel et ce cours intensif du langage C, nous allons faire de temps en temps une exception.

Dans les programmes de ce cours d'initiation, les commentaires sont en français mais les véritables exemples de programmes et la RP6Library sont entièrement en anglais.

La raison est simple: il faudrait gérer deux versions du code source et à chaque modification ou ajout de nouvelles fonctions, il faudrait mettre à jour deux versions et rédiger des commentaires en deux langues. Non seulement cela représente beaucoup de travail mais retarde également la publication d'améliorations et de nouvelles versions.

L'anglais est inévitable si l'on veut apprendre le langage C ou bien s'intéresser de plus près aux microcontrôleurs, à l'électronique et à la robotique.

Toutefois, le fonctionnement des programmes sera expliqué en français dans le chapitre correspondant!

Revenons à notre exemple. Ce petit programme est opérationnel mais ne fait rien d'autre qu'initialiser le microcontrôleur et éditer le texte:

`"Salut le Monde!"` + indentation / retour à la ligne

via l'interface série. Un programme type qui se trouve dans tous les livres.

Vous pouvez évidemment tester le petit programme. Cela peut être très utile de le taper afin de s'habituer à ce langage. Il faut surtout s'habituer dans un premier temps au grand nombre de points-virgules et de caractères spéciaux...

Celui qui trouve le programme ci-dessus trop monotone, peut aller sur le CD où il trouvera dans les exemples un programme « Salut le Monde » un peu plus intéressant avec un petit séquenceur à LED et quelques textes en plus.

Nous allons maintenant passer en revue ligne par ligne le programme dans le listing 1 et l'expliquer!

Lignes 1 - 3: `/* Un petit programme en C „Salut le Monde“ pour le RP6! */`

Il s'agit d'un commentaire. Le compilateur ignore cette partie du code source. Les commentaires servent à la documentation du texte source et facilitent la compréhension de textes sources étrangers (et personnels!). Ils permettent de mieux comprendre le cheminement intellectuel du programmeur tiers ou de se remémorer ses propres pensées après quelques années. Les commentaires commencent par /* et finissent par */.

Entre les deux, vous pouvez rédiger des commentaires de n'importe quelle longueur et également commenter des parties de votre code source pour tester p.ex. une autre variante sans effacer l'ancien code source. Outre ces commentaires à lignes multiples, le GCC supporte aussi des commentaires à 1 ligne qui sont introduits par un « // ». Tout ce qui se trouve dans une ligne après un « // » est interprété comme un commentaire par le compilateur.

Ligne 5: `#include "RP6RobotBaseLib.h"`

Cette ligne inclut la bibliothèque des fonctions RP6 qui offre de nombreuses fonctions utiles et prédéfinies qui facilitent énormément le pilotage du matériel. Cela se fait en incluant ce qui s'appelle une en-tête (terminaison « *.h ») car sinon le compilateur ne sait pas où trouver toutes les fonctions. Les en-têtes sont nécessaires pour tout ce qui se trouve dans des fichiers C externes. Si vous examinez le contenu du RP6RobotBaseLib.c et du RP6RobotBaseLib.h, vous allez mieux comprendre le principe. Nous aborderons plus en détail le « #include » dans le chapitre sur le préprocesseur.

Ligne 7: `int main(void)`

La partie la plus importante de l'exemple de programme: la fonction Main. Nous expliquerons plus loin en quoi consiste exactement une fonction. Pour l'instant, il suffit de savoir que le programme commence ici (anglais « Main Function » signifie en français « Fonction principale »).

Ligne 8 et Ligne 12: { }

En langage C, on définit des « blocs » entre accolades, donc '{' et '}'. (Sur le clavier, ce sont les touches [AltGr] + [4] pour '{' et [AltGr] + [+] pour '}').

Les blocs contiennent plusieurs commandes.

Ligne 9: `initRobotBase();`

Cette ligne appelle une fonction de la RP6Library qui initialise le microcontrôleur AVR. Elle permet de configurer les fonctions de matériel de l'AVR. Sans cet appel, la plupart des fonctions du microcontrôleur ne travailleraient pas correctement. Donc, n'oubliez jamais de l'appeler en premier!

Ligne 10: `writeString("Hello World!\n");`

Cette ligne appelle la fonction „writeString" de la bibliothèque RP6 avec le texte "Hello World!\n" comme paramètre. Le texte est publié par cette fonction via l'interface série.

Ligne 11: `return 0;`

Le programme se termine ici. Vous quittez la fonction Main et vous renvoyez la valeur 0. Les grands ordinateurs l'utilisent souvent pour des codes erreur ou similaires. En fait, nous n'en avons pas besoin sur le microcontrôleur et il est là seulement parce que le standard C l'a prévu ainsi.

Voilà un petit aperçu de la façon dont fonctionne un programme C. Maintenant nous

devons étudier quelques bases avant de pouvoir continuer.

4.4.3. Les Bases du Langage C

Comme nous l'avons dit au début, un programme C est composé de pur texte ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange). Il faut distinguer rigoureusement entre majuscules et minuscules. Cela signifie que vous devez nommer une fonction qui s'appelle « jeSuisUneFonction » toujours exactement de la même façon. « JesuiSUnefonCtion » ne fonctionnerait pas!

Vous pouvez insérer le nombre d'espaces, de tabulations et de retours à la ligne que vous voulez sans que la signification du programme ne change.

Vous avez pu constater dans l'exemple l'indentation des commandes par des tabulations afin d'améliorer la lisibilité mais ce n'est pas une obligation. La partie à partir de la ligne 7 du listing ci-dessus pourrait également s'écrire comme ceci:

```
1  int main(void){initRobotBase();writeString("Salut le Monde!\n");return 0;}
```

La signification est la même mais la clarté est moindre. Nous avons simplement retiré les retours à la ligne et les indentations. Le compilateur ne s'en soucie guère! (Certains espaces p.ex. entre « int » et « main » sont évidemment nécessaires pour séparer certains mots-clés et indicateurs et vous ne devez pas non plus mettre des retours à la ligne entre les guillemets!)

Plusieurs commandes sont réunies dans un seul bloc par des accolades { }. Nous en avons besoin pour les fonctions, conditions et boucles.

Chaque instruction se termine par un point-virgule ';' pour que le compilateur puisse les différencier.

Une astuce importante dès le départ si vous voulez taper les programmes dans ce tutoriel: On oublie très facilement de séparer les commandes par des points-virgules ou on le place au mauvais endroit et on s'étonne ensuite pourquoi le programme ne fait pas ce qu'il devrait faire. Si vous oubliez un point-virgule dans certaines parties du programme, le compilateur risque de générer une multitude de messages d'erreur même s'il ne s'agit que d'une seule erreur au départ. Souvent le premier message d'erreur est celui qui indique la véritable erreur.

Une autre erreur fréquente est d'oublier de fermer les accolades ou bien une erreur dans l'orthographe des commandes. Le compilateur ne pardonne pas d'erreurs syntaxiques (= » fautes d'orthographe »). Au début, il faut s'habituer à cette discipline mais on apprend vite après avoir écrit quelques programmes soi-même.

Chaque programme C commence par la fonction Main. Les commandes suivantes sont traitées dans l'ordre de la première à la dernière. Le microcontrôleur AVR ne peut pas exécuter plusieurs commandes en même temps.

Mais cela ne fait rien car il existe de nombreuses possibilités d'influencer le déroule-

ment du programme et de passer à d'autres endroits dans le programme (à l'aide d'un contrôle de flux, mais nous en reparlerons plus tard).

4.4.4. Variables

Apprenons d'abord comment entrer et lire des données dans la mémoire de travail. Cela se fait à l'aide de variables. Dans le langage C, il existe différentes sortes de variables appelées types. Ce sont toujours des types à nombres entiers de 8, 16 ou 32-bits qui peuvent être employés avec ou sans signe. Le nombre de bits d'une variable dépend de la valeur des nombres qu'elle doit pouvoir contenir.

Pour le RP6, nous utilisons les types suivants:

Type	Alternative	Valeur	Observation
signed char	int8_t	8 Bit: -128 ... +127	1 octet
char	uint8_t	8 Bit: 0 ... 255	'' sans signe
int	int16_t	16 Bit: -32768 ... +32767	2 octets
unsigned int	uint16_t	16 Bit: 0 ... 65535	'' sans signe
ong	int32_t	32 Bit: -2147483648 ... +2147483647	4 octets
unsigned long	uint32_t	32 Bit: 0 ... 4294967295	'' sans signe

Le type de données « int » n'étant pas normalisé sur les différentes plate-formes et fait p.ex. 16 bits sur notre microcontrôleur mais 32 bits sur un PC, nous allons utiliser l'identification normalisée plus récente de: int16_t

Ces types de données sont toujours construits de la façon suivante: [u] int N _t

```
u : unsigned (pas de signe)
int : Integer (nombre entier)
N : Nombre de Bits, p.ex. 8, 16, 32 ou 64
_t : t comme „type" afin d'éviter des confusions avec d'autres identificateurs.
```

Sur un microcontrôleur, chaque octet de mémoire compte. C'est pourquoi cet identifiant permet de mieux garder le contrôle. L'identificateur montre immédiatement qu'il doit s'agir d'un type de données de 16bit (en raison du 16 dans le nom). S'il est précédé d'un « u », il s'agit d'un type sans signe, donc « unsigned ». Sinon, le type de données est « signed », donc porte un signe.

Dans le tableau ci-dessus, seul « signed char » porte cet identifiant car int et ing portent systématiquement un signe et seulement char est systématiquement sans signe, même si ce n'est pas écrit expressément. Cela résulte d'une option du compilateur que nous utilisons et qui est presque toujours activée sur l'AVR-GCC.

Dans le cas de chaînes de caractères (anglais =« Strings »), on continue à utiliser « char » puisque sinon, certaines choses de la bibliothèque standard de C ne seraient pas compatibles en raison de la définition de uint8_t et qu'il est de toute manière plus logique d'utiliser à cet effet le type de donnée char (=anglais pour «caractère»). Ce sujet sera traité plus en détail dans le chapitre qui traite de la RP6Library lors de la sortie de texte par l'interface série.

Donc, nous retiendrons tout simplement: Pour les caractères et chaînes de caractères, utiliser toujours « char », pour les valeurs numériques utiliser toujours uintN_t ou intN_t!

Pour pouvoir utiliser une variable, elle doit d'abord être déclarée. La déclaration détermine le type de données, le nom et éventuellement la valeur de départ de la variable. Le nom d'une variable doit commencer par une lettre. (Le trait du bas "_" compte également comme une lettre) et peut contenir des chiffres mais *pas* de caractères spéciaux tels que „äöüß#'[]²³|*+-.,<>%&/(){}$§=´°?!^".

Il faut bien différencier entre les majuscules et les minuscules. Donc abC et aBc sont des variables différentes. Par convenance, les variables sont en minuscules, tout du moins la première lettre.

Les désignations suivantes sont déjà utilisées pour d'autres choses et ne peuvent PAS servir de nom de variable, de nom de fonction ou autre identificateur:

auto	**default**	**float**	**long**	**sizeof**	**union**
break	**do**	**for**	**register**	**static**	**unsigned**
case	**double**	**goto**	**return**	**struct**	**void**
char	**else**	**if**	**short**	**switch**	**volatile**
const	**enum**	**int**	**signed**	**typedef**	**while**
continue	**extern**				

En outre, il existe des nombres à virgule flottante des types float et double. Toutefois sur un petit microcontrôleur AVR, il vaut mieux les éviter car ils prennent généralement trop de temps de calcul et d'espace mémoire et on s'en sort mieux avec des nombres entiers. Donc, pour le RP6, nous n'aurons pas besoin de ce type de donnée.

C'est extrêmement simple de déclarer une variable, en voici un char avec le nom x:

```
char x;
```

Après cette déclaration, la variable x est valide dans le code de programmation suivant et peut être utilisé.

Ainsi, nous pouvons p.ex. lui attribuer la valeur 10

```
x = 10;
```

On peut également attribuer une valeur à la variable au moment de sa déclaration:

```
char y = 53;
```

Les calculs de base s'appliquent normalement:

```
signed char z; // Char avec signe!
z = x + y;   // z porte maintenant la valeur z = x + y = 10 + 53 = 63
z = x - y;   // z porte maintenant la valeur z = 10 - 53 = -43
z = 10 + 1 + 2 - 5;   // z = 8
z = 2 * x;   // z = 2 * 10 = 20
z = x / 2;   // z = 10 / 2 = 5
```

Voici quelques abréviations utiles:

```
z += 10;  // correspond à: z = z + 10; donc z = 15
z *= 2;   // z = z * 2 = 30
z -= 6;   // z = z - 6 = 24
z /= 4;   // z = z / 4 = 8

z++;  // Abréviations pour z = z + 1; donc z est ici 9
z++;  // z = 10  // z++ s'appelle aussi „incrémenter z"
z++;  // z = 11 ...
z--;  // z = 10  // z-- s'appelle aussi „décrémenter z"
z--;  // z = 9
z--;  // z = 8 ...
```

En haut, nous avons encore utilisé le type de donnée « char » mais dans tous les autres programmes du RP6, nous n'utiliserons (presque) que des types de données normalisés

Donc: `int8_t x;`

est identique à: `signed char x;`

Et: `uint8_t x;`

identique à: `unsigned char x;  // dans notre cas tout simplement « char »` puisque ce type de donnée est toujours „unsigned".

4.4.5. Conditions

Les conditions avec une construction « if-else » sont très, très importantes pour contrôler le déroulement du programme. Elles permettent de vérifier si une condition donnée est vraie ou fausse et exécuter ou non un code de programme donné.

En voici un exemple:

```
1  uint8_t x = 10;
2  if(x == 10)
3  {
4        writeString("x ist gleich 10!\n");
5  }
```

Dans la ligne 1, la variable à 8 bit x est déclarée et la valeur 10 lui est attribuée. Maintenant nous vérifions dans la condition if suivante dans la ligne 2, si la variable x a la valeur 10. C'est évidemment toujours le cas et donc le bloc est exécuté à condition que « x est égal à 10 ». Si nous avions initialisé x avec 231, il ne se serait rien passé ici.

En général, une condition if présente toujours la syntaxe suivante:

```
if ( <Condition X> )
      <Bloc d'instruction Y>
else
      <Bloc d'instrucion Z>
```

Traduit en français, cela signifie: « si X alors Y sinon Z »".

Un autre exemple à ce propos:

```
1  uint16_t superVariable = 16447;
2
3  if(superVariable < 16000) // Si superVariable < 16000
4  {                         // Alors:
5        writeString("superVariable est plus petit que 16000!\n");
6  }
7  else                      // Sinon:
8  {
9        writeString("superVariable est plus grand que ou égal à 16000!\n");
10 }
```

La sortie serait ici „superVariable est plus grand que ou égal à 16000!", car superVariable est ici 16447 et donc plus grand que 16000. La condition n'est donc pas remplie et la partie else est exécutée. Comme vous pouvez le constater sur le nom « superVariable », il n'y a pas d'autres restrictions pour nommer des variables (et autres) que celles mentionnées ci-dessus.

Il est également possible d'enchaîner plusieurs conditions If-then-else pour demander plusieurs cas alternatifs:

```
1  if(x == 1) { writeString("x est 1!\n"); }
2  else if(x == 5) { writeString("x est 5!\n"); }
3  else if(x ==  244) { writeString("x est 244!\n"); }
4  else { writeString("x a une autre valeur!\n");}
```

A l'intérieur des conditions, vous pouvez utiliser les opérateurs comparatifs suivants:

x == y	Comparaison logique basée sur l'égalité
x != y	Comparaison logique basée sur l'inégalité
x < y	Comparaison logique basée sur « plus petit que »
x <= y	Comparaison logique basée sur « plus petit que ou égal à »
x > y	Comparaison logique basée sur »plus grand que »
x >= y	Comparaison logique basée sur « plus grand que ou égal à »

Ensuite, il existe des opérateurs logiques de liaison:

x && y	Vrai, si x et y sont vrais
x || y	Vrai, si x est vrai et/ou Y est vrai
!x	Vrai, si x n'est pas vrai

Vous pouvez lier et imbriquer tout cela à votre guise et placer autant de parenthèses que vous voulez:

```
1  if( ((x != 0) && (!(x > 10))) || (y >= 200)) {
2        writeString("OK!\n");
3  }
```

La condition ci-dessus devient vraie si x est inégal à 0 `(x != 0)` ET x n'est pas supérieur à 10 `(!(x > 10))` OU si y est supérieur ou égal à 200 `(y >= 200)`. On pourrait y ajouter encore beaucoup d'autres conditions si nécessaire.

4.4.6. Sélection à choix multiples

Il faut souvent vérifier une variable d'après un grand nombre de valeurs numériques et exécuter des codes de programmes en conséquence. Cela peut évidemment se faire en utilisant une multitude de conditions if-then-else mais c'est plus élégant d'utiliser une sélection à choix multiples 'switch'.

Un exemple:

```
1  uint8_t x = 3;
2
3  switch(x)
4  {
5        case 1: writeString("x=1\n"); break;
6        case 2: writeString("x=2\n"); break;
7        case 3: writeString("x=3\n");   // „Break" manque ici, donc le
8        case 4: writeString("Hallo\n"); // programme enchaîne immédiatement
9        case 5: writeString("du\n");    // avec le cas suivant et suivant, et
10       case 6: writeString("da!\n"); break; // ne s'arrête qu'ici!
11       case 44: writeString("x=44\n"); break;
12       // Le programme arrive ici si aucun des cas ci-dessus
13       // ne s'est avéré:
14       default : writeString("x est autre chose!\n"); break;
15 }
```

Le résultat est *similaire* à l'exemple sur la page précédente avec « if-else-if-else-if-else... » mais avec une autre écriture.

La sortie serait dans ce cas (avec x = 3):

```
x=3
Hallo
du
da!
```

Si x = 1 la sortie serait „`x=1\n`" et avec x=5 la sortie serait:

```
du
da!
```

Nous constatons ici que le « break » met fin à la sélection switch. S'il n'y était pas, le programme parcourrait tous les autres cas jusqu'à ce que la fin de la sélection switch était atteinte ou un autre « break » serait exécuté peu importe si les conditions suivantes sont avérées ou non.

Si x = 7, aucun de ces cas ne couvrirait cette valeur. Le programme finirait dans la partie Défaut et la sortie serait: „`x est autre chose!\n`".

Toutes ces sorties de texte ne sont que des exemples. Elles permettraient dans un programme réel de déclencher p.ex. des séquences de mouvement différents du robot. Dans certains exemples d'application, des constructions switch sont utilisées p.ex. pour des automates finis pour mettre en œuvre différents comportements du robot.

4.4.7. Boucles

Les boucles sont utilisées lorsque certaines opérations doivent être répétées.

En voici un petit exemple:

```
uint8_t i = 0;
while(i < 10)                  // tant que i est plus petit que 10...
{                              // ... répète le code programme suivant:
      writeString("i=");       // "i=" sortir,
      writeInteger(i, DEC);    // sortir la valeur décimale (angl. „DECimal“)
                               // de i et ...
      writeChar('\n');         // ... un retour à la ligne.
      i++;                     // incrémenter i.
}
```

Il s'agit ici d'une boucle « while » qui sort dans l'ordre „i=0\n“, „i=1\n“, „i=2\n“, ... „i=9\n“. Le bloc après la tête de boucle, donc le „while(i < 10)“ est répété autant de fois que la condition est vraie. Dans ce cas, la condition est « i est plus petit que 10 ». En français, cela signifierait « répète le bloc suivant tant que i est plus petit que 10 ». Comme i est 0 au départ et augmente de 1 à chaque passage en boucle, la boucle passera en tout 10 fois et sort les chiffres 0 à 9. La condition dans la tête de boucle pourra être construite de la même façon que pour les conditions if.

Outre la boucle while, il existe la boucle « for ». Son fonctionnement est similaire mais vous pouvez mettre davantage dans la tête de boucle.

Voici un exemple:

```
uint8_t i;  // n'est pas initialisé ICI mais dans la tête de boucle!
for(i = 0; i < 10; i++)
{
      writeString("i=");
      writeInteger(i, DEC);
      writeChar('\n');
}
```

Cette boucle génère exactement la même sortie que la boucle while ci-dessus. Cependant la tête de boucle contient davantage.

La structure de base est la suivante:

```
for ( <Initialisation du paramètre> ; <Condition d'arrêt> ; <Modification du paramètre> )
{
      <Bloc d'instruction>
}
```

Pour les microcontrôleurs, on fait souvent appel aux boucles sans fin, donc des boucles qui se répètent à l'infini. Presque chaque programme d'un microcontrôleur possède une boucle sans fin, soit pour mettre le programme dans un état final défini après l'avoir traité, soit pour exécuter des opérations pendant toute la durée de fonctionnement du microcontrôleur.

Cela peut se faire tout simplement avec une boucle while ou for:

```
while(true) { }
```

ou bien

```
for(;;) { }
```

Le bloc d'instructions sera exécuté à l'infini (ou bien jusqu'à ce que le microcontrôleur reçoive la commande „break").

Par acquis de conscience, nous allons également nommer la boucle do-while qui est une variante de la boucle normale while. La différence est que le bloc de commandes sera exécuté au moins une fois même si la condition n'est pas remplie.

La structure est la suivante:

```
do
{
      <Bloc d'instructions>
}
while(<Condition>);
```

Ne pas oublier le point-virgule! (Dans une boucle normale, il faut bien sûr ne pas le mettre)

4.4.8. Fonctions

Les fonctions constituent un élément essentiel du langage. Nous avons vu et utilisé précédemment plusieurs fonctions telles que la fonction « writeString » et « writeInteger » et bien sûr la fonction Main.

Les fonctions sont utiles lorsque certaines séquences du programme reviennent dans d'autres parties du programme – un bon exemple sont évidemment toutes les fonctions de sortie de textes que nous avons déjà fréquemment utilisées. Ce serait évidemment très lourd s'il fallait copier ces parties de programme identiques à tous les endroits où nous en avons besoin et cela occuperait inutilement de l'espace mémoire. Les fonctions permettent également d'effectuer des changements à un emplacement central ce qui évite de le faire en plusieurs endroits. Par ailleurs, le programme gagne en clarté lorsqu'il est divisé en plusieurs fonctions.

C'est pourquoi C permet de réunir des séquences de programme en fonctions qui sont toujours construites de la manière suivante:

```
<Type de retour> <Nom de la fonction> (<Paramètre 1>, <Paramètre 2>, ... <Paramètre n>)
{
      <Séquence de programme>
}
```

Afin d'avoir une meilleure idée, voici un petit exemple avec deux fonctions simples et la fonction Main déjà connue:

```
8   void someLittleFunction(void)
9   {
10        writeString("[Fonction 1]\n");
11  }
12
13  void someOtherFunction(void)
14  {
15        writeString("[Fonction 2 - pour changer]\n");
16  }
17
18  int main(void)
19  {
20        initRobotBase(); // Toujours appeler ceci en premier pour le RP6!
21
22        // Quelques appels de fonction:
23        someLittleFunction();
24        someOtherFunction();
25        someLittleFunction();
26        someOtherFunction();
27        someOtherFunction();
28        return 0;
29  }
```

La sortie du programme serait la suivante:

```
[Fonction 1]
[Fonction 2 – pour changer]
[Fonction 1]
[Fonction 2 – pour changer]
[Fonction 2 – pour changer]
```

La fonction Main sert de point d'entrée dans le programme. Dans chaque programme C, elle est appelée la première et DOIT toujours exister.

Dans notre fonction Main ci-dessus, la fonction initRobotBase de la RP6Library est toujours appelée en premier car elle initialise le microcontrôleur (sur le RP6, il faut toujours l'appeler la première dans la fonction Main, sinon beaucoup de choses ne fonctionnent pas). Elle est construite d'après le même principe que les deux autres fonctions dans le listing. Ensuite les fonctions que nous venons de définir, sont appelées à la suite et le code programme dans les fonctions est exécuté.

La définition des fonctions ne se limite pas uniquement à celle de l'exemple: nous pouvons utiliser également des paramètres et valeurs de retour. Dans le exemple de programme ci-dessus, le paramètre et type de retour est « void » ce qui signifie à peu près « vide ». Cela veut dire tout simplement que ces fonctions ne possèdent pas de valeur de retour, ni de paramètres. Nous pouvons définir un grand nombre de paramètres pour une fonction. Les paramètres sont séparés par des virgules.

Un exemple:

```
1  void outputSomething(uint8_t something)
2  {
3        writeString("[La valeur suivante a été attribuée à la fonction: ");
4        writeInteger(something, DEC);
5        writeString("]\n");
6  }
7
8  uint8_t calculate(uint8_t param1, uint8_t param2)
9  {
10       writeString("[CALC]\n");
11       return (param1 + param2);
12 }
13
14 int main(void)
15 {
16       initRobotBase();
17
18       // Quelques appels de fonction:
19       outputSomething(199);
20       outputSomething(10);
21       outputSomething(255);
22
23       uint8_t result = calculate(10, 30);
24       outputSomething(result);
25       return 0;
26 }
```

Sortie:

```
[La fonction a reçu la valeur suivante: 199]
[La fonction a reçu la valeur suivante: 10]
[La fonction a reçu la valeur suivante: 255]
[CALC]
[La fonction a reçu la valeur suivante: 40]
```

La RP6Library contient également beaucoup de fonctions différentes. Examinez-en quelques-unes ainsi que des exemple de programmes et le principe apparaît rapidement.

4.4.9. Tableaux, Chaînes de Caractères, Pointeurs ...

Il y aurait encore beaucoup de choses à dire sur les éléments de langage et des détails mais nous devons vous renvoyer à la littérature existante.

La majeure partie de tout cela n'est pas nécessaire pour comprendre les exemples de programmes. Nous ne présentons ici que quelques exemples et notions pour donner un aperçu mais il ne s'agit pas d'une description particulièrement détaillée.

D'abord les tableaux: Dans un tableau (anglais: array=champs), vous pouvez mémoriser un nombre prédéterminé d'éléments d'un seul type de donnée. Ainsi, vous pourriez déclarer un tableau avec 10 octets:

```
uint8_t notreSuperTableau[10];
```

Cela fait pratiquement 10 variables déclarées avec le même type de donnée auxquelles on accède par un index:

```
notreSuperTableau[0] = 8;
notreSuperTableau[2] = 234;
notreSuperTableau[9] = 45;
```

Chacun de ces éléments peut être traité comme une variable normale.

Attention: L'index commence toujours par 0 et si l'on a déclaré un tableau avec n éléments, l'index va toujours de 0 à n-1! Donc pour 10 éléments de 0 à 9.

Les tableaux sont très utiles lorsqu'il faut mémoriser beaucoup de valeurs du même type. On peut aussi les parcourir en boucle dans l'ordre:

```
uint8_t i;
for(i = 0; i < 10; i++)
      writeInteger(notreSuperTableau[i],DEC);
```

Ainsi tous les éléments contenus dans le tableau sortent dans l'ordre (ici bien sûr sans séparateur, ni retour à la ligne). Parallèlement à cela, il est également possible de déclarer un tableau dans une boucle avec des valeurs.

Les chaînes de caractères dans C sont construites d'une manière similaire. Des chaînes de caractères normales sont toujours codées en ASCII et il faut 1 octet par caractère. En langage C, des chaînes de caractères ne sont rien d'autre que des tableaux que l'on peut interpréter comme des chaînes de caractères. Ainsi on peut écrire ceci:

```
uint8_t chainedecaracter[16] = "abcdefghijklmno";
```

Et voilà que la chaîne de caractères entre parenthèses est enregistrée dans la mémoire.

Nous avons déjà utilisé quelques fonctions UART qui sortent des chaînes de caractères par l'interface série. Ces chaînes de caractères sont tout simplement des tableaux. Cependant la fonction ne transmettra pas un tableau complet mais seulement l'adresse du premier élément du tableau. La variable qui contient cette adresse s'appelle « pointeur » (anglais: pointer). Pour générer un pointeur pour un élément précis du tableau, il faut écrire &notreSuperTableau[x], x étant l'élément qui pointe vers le pointeur. Cette écriture est parfois utilisée dans les exemples de programme. P.ex. ainsi:

```
uint8_t * pointeurSurUnElement =  &chainedecaracter[4];
```

Vous n'en aurez pas besoin pour comprendre les exemple et écrire vos propres programmes. Nous ne le mentionnons ici que par acquis de conscience.

4.4.10. Déroulement du Programme et Interruptions

Nous avions déjà mentionné auparavant qu'un programme est toujours exécuté instruction après instruction du haut vers le bas. Par ailleurs, il existe le contrôle de flux avec des conditions et boucles ainsi que des fonctions.

Outre le déroulement normal du programme, il existe des « interruptions ». Les différents modules de matériel (horloge, TWI, UART, Interruptions externes, etc.) du microcontrôleur peuvent déclencher des événements auxquels le microcontrôleur doit réagir le plus rapidement possible. Pour ce faire, où qu'il se trouve (ou presque) dans le programme normal, le microcontrôleur saute dans une routine d'interruption (ISR) où il peut réagir le plus rapidement possible à l'événement. Ne vous inquiétez pas, vous n'avez pas à vous en occuper. La RP6Library s'est déjà chargée de tous les ISR nécessaires. Nous abordons ce sujet uniquement pour que vous sachiez de quoi il s'agit et à quoi servent ces « fonctions » étranges de la bibliothèque.

Les ISR se présentent de la façon suivante:

```
ISR ( <InterruptVector> )
{
      <Bloc d'instructions>
}
```

p.ex. pour l'encodeur gauche sur l'entrée d'interruption externe 0:

```
ISR (INT0_vect)
{
      // A chaque flanc de signal, le compteur augmente de 2:
      mleft_dist++;     // distance parcourue
      mleft_counter++;  // Mesure de vitesse
}
```

Il est impossible d'appeler ces routines d'interruption directement. Cela se fait toujours automatiquement et souvent, il est impossible de prédire à quel moment cela se produira. Cela peut arriver à tout moment dans n'importe quelle partie du programme (sauf pendant une routine d'interruption déjà en cours ou si les interruptions sont désactivées). La routine d'interruption est alors exécutée et ensuite le microcontrôleur revient à l'endroit où le programme a été interrompue. Lorsque des interruptions sont utilisées, il faut également effectuer toutes les tâches dont la durée est primordiale, pendant les routines d'interruption. Des pauses calculées sur la base de cycles d'horloge pourraient être trop longues si elles sont interrompues par une ou plusieurs routines d'interruption.

La RP6Library utilise des interruptions pour générer la modulation de 36kHz pour la détection infrarouge et la communication ainsi que pour le décodage RC5, le timing et les fonctions de temporisation, pour évaluer les encodeurs, pour le module TWI (bus I²C) et encore quelques autres petites choses.

4.4.11. Préprocesseur C

Nous allons aborder brièvement le préprocesseur que nous avons déjà utilisé auparavant, à savoir : `#include "RP6RobotBaseLib.h"`!

Le préprocesseur traite cette instruction avant la traduction proprement dite par le GCC. `#include "Fichier"` insère le contenu du fichier indiqué à cet endroit. Dans le

cas de notre exemple, il s'agit du fichier RP6BaseLibrary.h qui contient les définitions des fonctions contenues dans la RP6Library. Il faut procéder de cette manière pour que le compilateur puisse trouver et attribuer cette fonction correctement.

Cependant, le préprocesseur ne se limite pas à cela. Vous pouvez également définir des constantes (donc des valeurs fixes non variables):

```
#define CECI_EST_UNE_CONSTANTE 123
```

Ceci définirait la constante: « CECI_EST_UNE_CONSTANTE » par la valeur 123. Le préprocesseur remplace tout simplement chaque occurrence de cette constante par la valeur (en fait, il s'agit d'un remplacement de texte). Ainsi dans:

```
writeInteger(CECI_EST_UNE_CONSTANTE,DEC);
```

le „CECI_EST_UNE_CONSTANTE“ serait remplacé par « 123 » et serait donc identique

à:

```
writeInteger(123,DEC);
```

(d'ailleurs, le « DEC » de writeInteger n'est qu'une constante qui a été définie ici par 10 pour le système décimal.)

Le préprocesseur connaît également de simples conditions if:

```
1  #define DEBUG
2
3  void someFunction(void)
4  {
5        // Fais quelque chose
6        #ifdef DEBUG
7           writeString_P("someFunction a été exécuté!");
8        #endif
9  }
```

Le texte ne sortirait ici que si DEBUG a été défini (il n'est pas nécessaire d'attribuer une valeur, il suffit de le définir). C'est utile pour activer uniquement en cas de besoin certaines sorties qui ne servent qu'à la recherche d'erreurs. Si DEBUG n'était pas défini dans l'exemple ci-dessus, le préprocesseur ne transmettrait même pas le contenu de la ligne 7 au compilateur.

Dans la RP6Library, nous avons souvent besoin de macros qui sont également définis

par #define. On peut leur attribuer des paramètres comme dans les fonctions:

```
#define setStopwatch1(VALUE) stopwatches.watch1 = (VALUE)
```

Cela s'appelle comme une fonction normale (`setStopwatch1(100);`).

Une autre chose très importante: Derrière les définitions du préprocesseur, il ne faut pas mettre de point-virgule!!

4.5. Makefiles

L'outil « Make » nous évite beaucoup de travail car normalement il faudrait écrire des lignes et des lignes de commandes afin de compiler un programme en C.

Make traite un fichier appelé « Makefile » qui contient toutes les séquences de commande et informations nécessaires à la compilation d'un projet. Ces Makefiles sont déjà tout prêts dans chacun des projets-exemples du RP6. Vous êtes évidemment libre de créer des Makefiles vous-même par la suite. La description détaillée d'un Makefile serait trop long et ne vous servirait à rien dans notre projet. Pour tous les projets du RP6, seules les quatre entrées suivantes sont importantes pour vous, le reste ne présente pas d'intérêt pour un débutant:

```
TARGET = programmName

RP6_LIB_PATH=../../RP6lib

RP6_LIB_PATH_OTHERS=$(RP6_LIB_PATH)/RP6base $(RP6_LIB_PATH)/RP6common

SRC += $(RP6_LIB_PATH)/RP6base/RP6RobotBaseLib.c
SRC += $(RP6_LIB_PATH)/RP6common/RP6uart.c
SRC += $(RP6_LIB_PATH)/RP6common/RP6I2CslaveTWI.c
```

Nos Makefiles contiennent également quelques lignes de commentaire avec des explications et indications. Les commentaires commencent toujours par un # dans un Makefile et seront ignorés par Make.

Les projets-exemples pour le R6 contiennent des Makefiles tout prêts avec les entrées appropriées. Donc, vous ne devez les changer que si vous voulez p.ex. ajouter de nouveaux fichiers en C au projet ou renommer des fichiers.

Derrière « TARGET » il faut mettre le nom du fichier C qui contient la fonction Main. Ici, vous ne saisissez que le nom SANS le « .c » à la fin! Sur beaucoup d'autres lignes, il faut impérativement indiquer la terminaison. Donc, examinez toujours les exemples et les informations données dans les commentaires!

Dans l'entrée « RP6_LIB_PATH », vous devez indiquer le répertoire avec les données de la RP6Library. Il s'agit de « ../RP6lib » ou « ../../RP6lib », donc un chemin relatif. (« ../ » signifie « un niveau de répertoire au dessus »).

Dans RP6_LIB_PATH_OTHERS il faut indiquer tous les autres répertoires que l'on a utilisé. La RP6Library est subdivisée en plusieurs répertoires dont il faut indiquer tous ceux que l'on a effectivement utilisés.

Et enfin, sous « SRC » vous devez indiquer tous les fichiers C (pas de fichiers à entête! Donc pas les fichiers qui se terminent par « .h »! Ceux-ci sont recherchés automatiquement dans tous les répertoires indiqués) que vous utilisez en plus du fichier contenant la fonction Main. Vous devez également indiquer les fichiers de la RP6Library dans la mesure où ils doivent servir.

Que signifie en fait $(RP6_LIB_PATH) ? C'est très simple: C'est la façon d'utiliser des variables dans des Makefiles. Nous avons déjà déclaré une « variable » RP6_LIB_PATH. Maintenant vous pouvez insérer son contenu partout par $ (<variable>). C'est très pratique et fait gagner beaucoup de temps de saisie...

Normalement vous n'avez rien à modifier de plus dans les Makefiles du RP6. Si vous voulez en savoir davantage, voici un manuel très détaillé:
http://www.gnu.org/software/make/manual/

4.6. Bibliothèque de Fonctions du RP6 (RP6Library)

La bibliothèque de fonctions du RP6 appelée aussi RP6Library ou en abrégé RP6Lib, offre de nombreuses fonctions utiles pour piloter le matériel du RP6, évitant ainsi à l'utilisateur de s'occuper des problèmes liés au matériel (ou presque!). De toute manière, il n'est pas nécessaire d'avoir lu la fiche technique de 300 pages de l'ATMEGA32 pour pouvoir programmer le robot! Il est cependant toujours utile d'avoir une idée approximative des tâches de la bibliothèque de fonctions du RP6. Par ailleurs, bon nombre de fonctions dans la RP6Library ne sont pas parfaites, justement pour vous laisser un peu de travail à faire!

Vous pourriez ajouter quelques fonctions complémentaires et optimiser beaucoup d'autres. Les fonctions de la RP6Library constituent une excellente base pour le développement personnalisé.

Dans ce chapitre, nous décrivons les fonctions les plus importantes et donnons quelques exemples. Si vous souhaitez approfondir le sujet, vous devez lire les commentaires dans les fichiers de la bibliothèque, analyser et refaire les fonctions et exemples.

4.6.1. Initialisation du Microcontrôleur

```
void initRobotBase(void)
```

Vous devez IMPERATIVEMENT appeler cette fonction en premier dans la fonction Main!

Elle initialise les modules de matériel du microcontrôleur. Seulement si vous appelez cette fonction en premier, le microcontrôleur travaillera comme nous le voulons pour le RP6! Certes une partie sera initialisée par le chargeur d'amorçage mais pas la totalité.

Exemple:

```
1  #include "RP6RobotBaseLib.h"
2
3  int main(void)
4  {
5        initRobotBase(); // Initialisation - APPELER TOUJOURS EN PREMIER!
6
7  // [...] Code Programme...
```

```
7
8           while(true);      // Boucle sans fin
9           return 0;
10    }
11
12
```

Chaque programme RP6 doit au minimum ressembler à ceci! La boucle sans fin en ligne 9 est nécessaire pour garantir une fin définie du programme! Sans cette boucle, le programme pourrait se comporter autrement que prévu!

En voici la raison: Normalement votre propre code programme est exécuté dans la boucle sans fin. Donc, on effacerait le point-virgule dans la ligne 9 et insérerait à la place un bloc entre accolades dans lequel on écrit son propre programme. Avant la fonction Main (donc la ligne 2), vous pouvez définir vos propres fonctions que vous pouvez appeler autant de fois que vous voulez à partir de la boucle principale.

4.6.2. Fonctions UART (Interface série)

Dans le cours intensif de C, nous avons déjà utilisé quelques fonctions de la RP6Library et surtout les fonctions UART. Ces fonctions permettent d'envoyer et de recevoir des messages de texte vers le PC (ou un autre microcontrôleur) en passant par l'interface série du robot.

4.6.2.1. Envoi de Données par l'Interface Série

```
void writeChar(char ch)
```

Cette fonction envoie un seul caractère ASCII à 8 bit via l'interface série.

L'application est très simple:

```
writeChar('A');
writeChar('B');
writeChar('C');
```

La sortie est « ABC ». On pourrait également envoyer directement des codes ASCII:

```
writeChar(65);
writeChar(66);
writeChar(67);
```

La sortie sur le terminal serait également « ABC » puisque chaque caractère ASCII est attribué à un numéro précis, p.ex. la lettre 'A' au 65. Un logiciel de communication adapté permettrait même d'interpréter les pures valeurs binaires.

Nous utiliserons souvent:

```
writeChar('\n');
```

pour commencer une nouvelle ligne sur le terminal.

```
void writeString(char *string)
und  writeString_P(STRING)
```

Ces fonctions sont très importantes pour la recherche d'erreurs dans les programmes car elles permettent d'envoyer des messages texte de toute nature au PC. Nous pouvons évidemment les utiliser aussi pour la transmission de données.

La différence entre writeString et writeString**_P** réside dans le fait qu'avec writeString_P, les textes sont enregistrés seulement dans le Flash-Rom (**P**rogram Memory) d'où ils sont aussi lus, tandis qu'avec writeString, ils sont en plus enregistrés dans la mémoire vive ce qui occupe donc le double en espace mémoire – et nous ne disposons que de 2Ko. Donc, lorsqu'il ne s'agit que de sortir du texte fixe non modifiable, il est conseillé d'utiliser toujours writeString_P. S'il faut sortir des données dynamiques qui existent de toute façon dans la RAM, il **faut** bien sûr utiliser le writeString normal.

Là encore, l'application est extrêmement simple:

```
writeString("ABCDEFG");
```

La sortie est « ABCDEFG » mais la chaîne de caractères occupe également de la mémoire vive.

```
writeString_P("ABCDEFG");
```

La sortie est également « ABCDEFG » mais sans occuper inutilement de RAM!

```
void writeStringLength(char *data,  uint8_t length,  uint8_t offset);
```

Si vous voulez sortir des textes à longueur (length) et position de départ (offset) réglables, vous pouvez utiliser cette fonction.

Un exemple:

```
writeStringLength("ABCDEFG", 3, 1);
```

Sortie: „BCD"

```
writeStringLength("ABCDEFG", 2, 4);
```

Sortie: „EF"

Toutefois, cette fonction occupe également de la mémoire vive et n'est destinée en fait qu'à une sortie de texte dynamique (est utilisée p.ex. par writeIntegerLength...).

```
void writeInteger(int16_t number, uint8_t base);
```

Cette fonction très utile sort des valeurs numériques sous forme de texte ASCII! Nous avons déjà pu constater que writeChar(65) p.ex. sort un 'A' et non pas un 65 ...

D'où la nécessité d'une telle fonction de conversion.

Exemple:

```
writeInteger(139, DEC);
```

Sortie: „139"

```
writeInteger(25532, DEC);
```

Sortie: „25532"

Vous pouvez sortir toute la plage de valeur 16bit avec signe de -32768 jusqu'à 32767. Si vous avez besoin de nombres plus grands, vous devez adapter la fonction ou, mieux encore, écrire votre propre fonction!

A quoi sert le « DEC » comme second paramètre? C'est très simple: Cela signifie que la sortie se fera par un nombre décimal (anglais DECimal). Il existe d'autres systèmes numériques que le système décimal basé sur le 10. Ainsi, les valeurs peuvent également sortir en binaire (BIN, base 2), octal (OCT, base 8) ou hexadécimal (HEX, base 16).

Exemples:

```
writeInteger(255, DEC);
```

Sortie: „255"

```
writeInteger(255, HEX);
```

Sortie: „FF"

```
writeInteger(255, OCT);
```

Sortie: „377"

```
writeInteger(255, BIN);
```

Sortie: „11111111"

Notamment HEX et BIN peuvent être très utiles puisque vous voyez sans calcul mental comment les différents bits sont disposés dans une valeur numérique.

```
void writeIntegerLength(uint16_t number, uint8_t base, uint8_t length);
```

Une variante de writeInteger qui permet en plus d'indiquer le nombre de digits (length) qu'il faut sortir. Si un nombre est plus court que le nombre de digits indiqué, il sera précédé par des zéros. S'il est plus long, seuls les derniers digits sont représen-

tés.

Exemples:

```
writeIntegerLength(2340, DEC, 5);
```

Sortie: „02340"

```
writeIntegerLength(2340, DEC, 8);
```

Sortie: „00002340"

```
writeIntegerLength(2340, DEC, 2);
```

Sortie: „40"

```
writeIntegerLength(254, BIN, 12);
```

Sortie: „000011111110"

4.6.2.2. Réception de Données par l'Interface Série

Ces fonctions ont été complètement réécrites dans la version 1.3 de la RP6Library et c'est pourquoi ce chapitre y a été adapté.

La réception de données par l'interface série se déroule entièrement sur la base d'interruption. Les données reçues sont automatiquement mémorisées en tâche de fond dans un buffer tournant.

La fonction

```
char readChar(void)
```

permet de lire les différents octets/caractères reçus dans le buffer tournant. Si vous appelez cette fonction, le prochain caractère disponible sera renvoyé et effacé du buffer tournant.

Si le buffer tournant est vide, il renvoie la fonction 0. Toutefois avant chaque appel avec la fonction

```
uint8_t getBufferLength(void)
```

il faut vérifier combien de nouveaux signes se trouvent encore dans le buffer tournant.

La fonction

```
uint8_t readChars(char *buf, uint8_t numberOfChars)
```

permet d'appeler plusieurs signes à la suite du buffer. Les paramètres de la fonction sont un pointeur sur un tableau et le nombre de signes qu'il faut copier dans ce tableau. La fonction renvoie le nombre réel de signes copiés. C'est utile si le buffer

contient moins de signes que demandés.

Si le buffer est plein, les anciennes données ne sont pas écrasées à la réception de nouveaux signes mais une variable status est écrite (uart_status) qui signale un « débordement » du buffer (anglais « Buffer overflow », UART_BUFFER_OVERFLOW). Vous devriez concevoir vos programmes de telle façon que cela ne puisse pas se produire. Cela arrive le plus souvent parce que le taux de données était trop élevé ou le programme a été bloqué par mSleep ou autre pendant une durée prolongée. Vous pouvez également adapter la dimension du buffer tournant. En principe, le buffer tournant a une capacité de 32 signes. Vous pouvez adapter la capacité par la définition UART_RECEIVE_BUFFER_SIZE dans le fichier RP6uart.h.

Vous trouverez un programme plus important à ce propos dans les exemples! L'exemple « Example_02_UART_02 » a été adapté aux nouvelles fonctions.

4.6.3. Fonctions Delay (Temporisations et Pilotage temporel)

Il faut souvent intégrer des temporisations (Delays) dans son programme ou attendre un certain laps de temps avant d'exécuter une certaine action.

La RP6Library a prévu des fonctions pour cela qui utilisent une des horloges du MEGA32 pour les temporisations afin d'être le plus indépendant possible de l'exécution du programme (interruptions) et d'assurer une certaine précision des temporisations.

Cependant il faut bien réfléchir à quel moment vous pouvez utiliser ces fonctions. Si vous utilisez le réglage automatique de vitesse et l'ACS (sera expliqué ultérieurement), vous pourriez rencontrer des problèmes! Il est recommandé de n'intégrer que de toutes petites pauses de quelques millièmes de secondes (<10ms). Vous pouvez cependant facilement contourner ce problème en utilisant des chronomètres qui seront décrits dans le chapitre suivant.

```
void sleep(uint8_t time)
```

Cette fonction permet d'arrêter le déroulement normal du programme pendant une durée déterminée. Le laps de temps est indiqué en incréments de 100µs (100µs = 0,1ms = 0,0001s qui sont très courts pour la perception humaine...). Etant donné que nous avons utilisé une variable 8bit, le laps de temps maximum est de 25500µs = 25,5ms. Des événements d'interruption continuent à être traités, seul le déroulement normal du programme est interrompu.

Exemples:

```
sleep(1); // Pause de 100µs
sleep(10); // Pause de 1ms
sleep(255); // Pause de 25.5ms
```

```
void mSleep(uint16_t time)
```

Pour des pauses plus longues, on peut utiliser la fonction mSleep car ici le laps de temps est donné en millièmes de secondes. La durée maximale est de 65535ms ou 65,5 secondes.

Exemple:

```
mSleep(1);     // Pause de 1ms
mSleep(100);   // Pause de 100ms
mSleep(1000);  // Pause de 1000ms = 1s
mSleep(65535); // Pause de 65.5 secondes
```

Stopwatches (Chronomètres)

Le problème avec les fonctions de temporisation normales est qu'elles interrompent complètement le déroulement normal du programme. Ce n'est pas toujours souhaité car souvent ce n'est qu'une partie du programme qui doit attendre brièvement pendant que le reste continue..

L'avantage majeur d'un chronomètre est qu'il fonctionne indépendamment du déroulement du programme. La RP6Library met ainsi en œuvre des « chronomètres » à utilisation universelle. Cette désignation n'est pas usuelle mais le nom convenait à l'auteur car ils fonctionnent presque comme des chronomètres réels. Les stopwatches facilitent un grand nombre de tâches. Normalement on écrit des fonctions d'horloge sur mesure pour le problème spécifique mais la RP6Library propose ceci d'une façon plus universelle et simple ce qui permet de les utiliser pour beaucoup d'autres choses.

Les chronomètres permettent d'effectuer un grand nombre de tâches quasiment en simultané, tout du moins c'est l'impression que cela donne à l'observateur.

La bibliothèque contient huit chronomètres à 16bit (Stopwatch1 à Stopwatch8) que vous pouvez démarrer, arrêter, poser et lire. La résolution s'élève à 1 millième de seconde comme pour la fonction mSleep. Cela signifie que toutes les millièmes de seconde chaque chronomètre augmente la position de son compteur de 1. Cela ne convient cependant pas aux tâches d'une haute précision temporelle puisque la requête pour savoir si un chronomètre a atteint une certaine valeur, ne se présente généralement pas exactement à ce moment.

L'utilisation de cette fonction est bien illustrée par l'exemple suivant:

```
1  #include "RP6RobotBaseLib.h"
2
3  int main(void)
4  {
5        initRobotBase(); // Initialiser le microcontrôleur
6        writeString_P("\nRP6 Programme démo Stopwatches\n");
7        writeString_P("_____________________________\n\n");
8
9        startStopwatch1(); // Démarrer Stopwatch1!
10       startStopwatch2(); // Démarrer Stopwatch2 !
11
12       uint8_t counter = 0;
13       uint8_t runningLight = 1;
14
15       // Boucle principale:
16       while(true)
17       {
18             // Un petit séquenceur à LED:
19             if(getStopwatch1() > 100) // 100ms (= 0.1s) sont passées?
```

```
17              {
18                  setLEDs(runningLight); // Régler LEDs
19                  runningLight <<= 1;    // LED suivante (Opération shift)
20                  if(runningLight > 32)  // Dernière LED?
21                      runningLight = 1;  // Oui,recommencer depuis le début!
22                  setStopwatch1(0);      // Remettre Stopwatch1 à 0
23              }
24
25              // Sortir la position du compteur sur le terminal:
26              if(getStopwatch2() > 1000) // 1000ms (= 1s) sont passées?
27              {
28                  writeString_P("CNT:");
29                  writeInteger(counter, DEC); // Sortir position du compteur
30                  writeChar('\n');
31                  counter++;          // Augmenter le compteur de 1
32                  setStopwatch2(0);   // Remettre à zéro Stopwatch2
33              }
34      }
35      return 0;
36  }
37
38
39
```

Le programme est très simple. Chaque seconde la position du compteur est sortie par l'interface série et incrémentée (lignes 29 à 36). En outre, un séquenceur de lumière est exécuté (lignes 19 à 26) qui commute en avant toutes les 100ms. A cet effet, nous utilisons Stopwatch1 et Stopwatch2. Les deux Stopwatches sont démarrés en lignes 9 et 10 et commencent à compter. Dans la boucle sans fin (lignes 16 à 37), une requête est envoyée en permanence pour savoir si les chronomètres ont dépassé une certaine position du compteur. Ainsi pour le séquenceur de lumière en ligne 19, la requête demande si *au moins* 100ms sont passées depuis que le chronomètre a commencé à compter à partir de 0. Si c'est le cas, la LED suivante est allumée et le chronomètre est remis à 0 (ligne 25) et le microcontrôleur attend de nouveau 100ms.

Vous trouverez le programme plus détaillé sur le CD. Il ne s'agit que d'un exemple simple puisque vous pouvez réaliser des choses plus complexes avec les stopwatches et même les démarrer et arrêter sous certaines conditions, etc...

Dans l'exemple sur le CD, le séquenceur et le compteur (il y en a même 3...) ont chacun leur propre fonction qui peut être appelée à partir de la boucle sans fin. Cela rend des programmes complexes plus clairs et des parties sont récupérables par simple copier/coller dans un autre programme, p.ex. la partie du séquenceur.

Il existe plusieurs macros pour le contrôle des chronomètres.

```
startStopwatchX()
```

Démarre un certain chronomètre. Le chronomètre n'est pas remis à zéro mais conti-

nue tout simplement à partir de la dernière position du compteur.

Exemples:

```
startStopwatch1();
startStopwatch2();
```

stopStopwatchX()

Arrête un certain chronomètre.

Exemple:

```
stopStopwatch2();
stopStopwatch1();
```

uint8_t isStopwatchXRunning()

Indique si le chronomètre fonctionne.

Exemple:

```
if(!isStopwatch2Running) {
      // Stopwatch ne tourne pas, donc fais quelque chose...
}
```

setStopwatchX(uint16_t preset)

Ce macro règle le stopwatch X sur la valeur transmise.

Exemple:

```
setStopwatch1(2324);
setStopwatch2(0);
setStopwatch3(2);
setStopwatch4(43456);
```

getStopwatchX()

Ce macro renvoie la valeur de stopwatch X.

Exemple:

```
if(getStopwatch2() > 1000) { ... }
if(getStopwatch6() > 12324) { ... }
```

4.6.4. LED d'état et Bumper (pare-chocs)

```
void setLEDs(uint8_t leds)
```

Vous pouvez contrôler les 6 LED d'état (Status LEDs) via cette fonction. Pour plus de clarté, donnez une valeur binaire à la fonction (se présente toujours ainsi: 0bxxxxxx).

Exemple:

```
setLEDs(0b000000); // Cette commande éteint toutes les LED.
setLEDs(0b000001); // Celle-ci allume la LED d'état1 et éteint toutes les autres
setLEDs(0b000010); // StatusLED2
setLEDs(0b000100); // StatusLED3
setLEDs(0b001010); // StatusLED4 et StatusLED2
setLEDs(0b010111); // StatusLED5, StatusLED3, StatusLED2 et StatusLED1
setLEDs(0b100000); // StatusLED6
```

Il existe l'alternative suivante:

```
statusLEDs.LED5 = true;    // Activer LED5 dans le registre des LED
statusLEDs.LED2 = false;   // Désactiver LED2 dans le registre des LED
updateStatusLEDs();        // Appliquer les changements!
```

Dans ce cas, la LED d'état 5 est allumée et la LED d'état 2 est éteinte. Les autres LED restent dans le même état qu'auparavant. Cela peut s'avérer utile/plus clair lorsque différentes LED devaient être modifiées à partir de différentes fonctions.

Attention: `statusLEDs.LED5 = true;` n'allume PAS directement la LED5! Cela place un bit dans une variable. Seulement `updateStatusLEDs();` allume vraiment la LED5!

Deux des broches du port sur lequel sont branchés les LED sont utilisées en plus pour les pare-chocs. Afin d'évaluer les pare-chocs, la broche du port est commutée brièvement sur le sens d'entrée pour déterminer si le commutateur est fermé. A cet effet, il existe deux fonctions préfabriquées.

Cette fonction

```
uint8_t getBumperLeft(void)
```

évalue le bumper de gauche et celle-ci

```
uint8_t getBumperRight(void)
```

le bumper de droite.

Comme ces fonctions sont exécutées très rapidement, les LED ne s'assombrissent pas même lorsque la fonction commute la broche sur le sens d'entrée. On devrait cependant insérer une pause de quelques millièmes de seconde entre les différents appels.

Les ports ne devraient être contrôlés que par les fonctions préparées! Les ports aux-

quels sont connectés les pare-chocs sont protégés par des résistances mais s'ils commutent sur la masse et un bumper se ferme, un courant assez élevé passe par le port ce qu'il faut donc éviter.

Exemple:

```
if(getBumperLeft() && getBumperRight())  // Les deux pare-chocs...
      escape(); // Définir ici sa propre fonction, p.ex. Reculer + tourner
else if(getBumperLeft()) // A gauche...
      escapeLeft(); // reculer et tourner à droite.
else if(getBumperRight()) // A droite...
      escapeRight(); // reculer et tourner à gauche.
mSleep(50); // N'évaluer les pare-chocs que 20 fois par seconde (20Hz)...
```

Les deux LED 6 et 3 s'allument dans tous les cas, lorsque les pare-chocs sont touchés. C'est fait exprès (n'est pas possible autrement) et non pas une erreur. Normalement les pare-chocs ne commutent que rarement et cela ne gêne donc pas.

Vous pouvez connecter aux ports des quatre autres LED d'autres pare-chocs/détecteurs à sortie numérique ou bien des transistors pour commuter des charges ainsi que des LED ou des petits moteurs. Cependant les fonctions adaptées n'ont pas encore été créées...

Attention: *Dans tous les cas, insérez des résistances d'au moins 470 Ohms entre les détecteurs/acteurs et les ports afin de ne pas endommager les ports du microcontrôleur par une surtension!*

D'ailleurs vous pouvez désactiver les LED pendant l'amorçage à l'aide du RP6Loader. C'est utile si l'on veut affecter autre chose à ces ports et on ne veut pas que ces ports soient mis sous et hors tension pendant l'amorçage. Le premier octet dans l'EEPROM interne (donc l'adresse 0) est réservé aux réglages comme celui-ci. Donc, vous ne devriez pas utiliser cet octet dans vos propres programmes (cela ne gêne rien d'important mais on pourrait se demander pour les LED ne s'allument plus lorsque l'on met le RP6 sous tension).

Le RP6 possède beaucoup de choses qu'il faut constamment évaluer et surveiller pour que tout se déroule correctement, tel que p.ex. l'ACS qui envoie et reçoit fréquemment des impulsions IR. Ce n'est pas possible de tout automatiser dans des routines d'interruption parce qu'elles demandent toujours une exécution très rapide. C'est pourquoi il faut appeler constamment quelques fonctions du programme principal qui exécutent des tâches comme celle-ci. Si l'on construit son propre programme correctement, cela donne l'impression comme si tout se déroulait en tâche de fond.

Ces fonctions seront présentées au cours de ce chapitre. Pour comprendre les fonctions pour les pare-chocs, nous devions les aborder déjà brièvement ici!

Si déjà un certain nombre de choses est exécuté en tâche de fond, autant exécuter en même temps des tâches plus petites telles que la lecture des pare-chocs.

Afin de surveiller les pare-chocs automatiquement, il faut appeler cette fonction:

```
void task_Bumpers(void)
```

constamment à partir de la boucle principale (voir aussi Fonctions d'entraînement où le procédé est décrit encore plus en détail). Elle vérifie les pare-chocs automatiquement toutes les 50ms et inscrit les valeurs (appuyées ou non) dans les variables

```
bumper_left und bumper_right
```

Celles-ci pourront être utilisées et interrogées ensuite tout à fait normalement dans des conditions if, etc. ou être affectées à d'autres variables.

Exemple:

```
#include "RP6RobotBaseLib.h"

int main(void)
{
	initRobotBase();  // Initialiser le microcontrôleur

	setLEDs(0b001001); // Allumer les LED 1 et 4 (vertes)

	while(true)
	{
		// Nous réglons les LED en fonction du pare-choc qui est
		// appuyé:
		statusLEDs.LED6 = bumper_left;  // Pare-choc gauche appuyé
		statusLEDs.LED4 = !bumper_left; // Gauche non appuyé
		statusLEDs.LED3 = bumper_right; // Pare-choc droit appuyé
		statusLEDs.LED1 = !bumper_right; // droit non appuyé
		// Les deux pare-chocs appuyés:
		statusLEDs.LED2 = (bumper_left && bumper_right);
		statusLEDs.LED5 = statusLEDs.LED2;
		updateStatusLEDs(); // Actualiser les LED...

		// Vérifier l'état du pare-choc:
		task_Bumpers(); // Appeler constamment de la boucle principale!
	}
	return 0;
}
```

Dans le exemple de programme, nous utilisons les LED d'état pour indiquer l'état des deux pare-chocs. Si le pare-choc gauche est appuyé, la LED 6 est allumée et la LED4 est éteinte. Inversement, s'il n'est pas appuyé, la LED6 est éteinte et la LED4 allumée. La LED6 s'allume de toute manière lorsque le pare-choc gauche est appuyé mais on pourrait aussi visualiser toute autre chose par les LED. Ce n'est qu'un exemple du déroulement en général. Pour le pare-choc droit, il se passe la même chose avec les LED3 et LED1. Si les deux pare-chocs sont touchés, les LED2 et LED5 s'allument.

Puisque l'évaluation automatique des pare-chocs toutes les 50ms existe déjà dans la bibliothèque, autant intégrer quelque chose qui appelle automatiquement une fonction

définie par l'utilisateur lorsque l'état change. Normalement, les pare-chocs sont sollicités très rarement et il est donc plus sensé d'appeler quelque chose dans le programme principal si cela répond à un réel besoin.

C permet aussi de définir des pointeurs vers des fonctions et de les appeler sans définir la fonction elle-même dans la bibliothèque. Normalement une fonction doit être connue au moment de la traduction du programme et être enregistrée dans la bibliothèque du RP6 pour pouvoir être appelée.

Ainsi vous pouvez utiliser des fonctions définies sur mesure en tant que « Event Handler » , donc pour le traitement d'événements. Lorsqu'un pare-choc est touché, une fonction est appelée dans les 50ms qui a été créée exprès pour cette tâche et qui a été enregistrée auparavant en tant que programmation événementielle. Pour cela, la fonction doit avoir une certaine signature. Dans ce cas, il doit s'agir d'une fonction qui ne possède ni de valeur de retour, ni de paramètre (les deux sont void).
La signature de la fonction doit donc se présenter ainsi: `void bumpersStateChanged(void)`. Vous pouvez enregistrer le „Event Handler" au début de la fonction Main. Pour l'enregistrement de cette programmation événementielle, vous devez utiliser la fonction suivante:

```
void BUMPERS_setStateChangedHandler(void (*bumperHandler)(void))
```

Ce n'est pas nécessaire de comprendre exactement la notation. Il suffit de savoir qu'un pointeur est attribué à une fonction...
Voici un exemple simple:

```
#include "RP6RobotBaseLib.h"

// Notre fonction „Event Handler" pour les pare-chocs
// La fonction est appelée automatiquement de la RP6Library:
void bumpersStateChanged(void)
{
	writeString_P("\nBumper L'état a changé:\n");

	if(bumper_left)
		writeString_P(" - Pare-choc gauche touché!\n");
	else
		writeString_P(" - Pare-choc gauche pas touché.\n");
	if(bumper_right)
		writeString_P(" - Pare-choc droit touché!\n");
	else
		writeString_P(" - Pare-choc droit pas touché.\n");
}

int main(void)
{
	initRobotBase();

	// Enregistrer l'Event Handler:
	BUMPERS_setStateChangedHandler(bumpersStateChanged);

	while(true)
	{
		task_Bumpers(); // Evaluer le pare-choc automatiquement toutes
les 50ms
	}
	return 0;
}
```

A chaque changement de l'état du pare-choc, le programme sort une fois l'état actuel des deux pare-chocs

Si vous appuyez sur le pare-choc droit, la sortie serait p.ex.:

```
Etat du pare-choc a changé:
 - Pare-choc gauche non touché.
 - Pare-choc droit touché!
```

Si vous appuyez sur les deux:

```
Etat du pare-choc a changé:
 - Pare-choc gauche touché!
 - Pare-choc droit touché!
```

Puisque dans la réalité il est impossible de toucher les deux vraiment en même temps, un message supplémentaire pourrait sortir où seulement l'un des deux a été touché.

Vous constaterez que dans le code programme ci-dessus, la fonction `bumpersStateChanged` n'est jamais appelée directement. Cela se fait automatiquement à

partir de la fonction `task_Bumpers` dans la RP6Library à chaque modification de l'état

des pare-chocs. Puisque `task_Bumpers` ne connaît même pas vraiment notre fonction,

cela ne peut se faire qu'au moyen d'un pointeur vers la fonction correspondante qui ne reçoit sa valeur que lors de l'exécution de l'appel en ligne 24.

La programmation événementielle peut évidemment déclencher encore d'autres actions que de sortir du texte. Elle pourrait p.ex. arrêter et faire reculer le robot. Toutefois il est déconseillé de le faire dans l' »Event Handler » lui-même mais plutôt dans d'autres parties du programme. Ainsi vous pourriez définir une variable de commande quelconque dans l' »Event handler » qui est interrogée dans le programme principal et qui commande les moteurs en conséquence. Toutes les programmations événementielles doivent toujours être les plus courtes possibles.

Vous pouvez certes utiliser toutes les fonctions de la RP6Library dans les « Event handlers » mais nous vous recommandons la plus grande prudence avec les fonctions « rotate » et « move » que nous allons aborder plus tard. N'UTILISEZ PAS le mode de blocage (car si vous appuyez plusieurs fois sur les pare-chocs, cela ne fonctionne plus comme vous le voulez ;-))!

Le principe avec les « Event handlers » est utilisé aussi pour d'autres fonctions p.ex. pour l'ACS où il fonctionne d'une manière très similaire que pour les pare-chocs car chaque changement d'état des détecteurs d'objet appelle un Event handler.

Les « Event handlers » sont également utilisés dans la réception de codes RC5 via une télécommande. A chaque réception d'un nouveau code RC5, un « Event handler » correspondant peut être appelé.

Ce n'est pas une *obligation* d'utiliser un Event handler pour toutes ces choses. Cela peut se faire aussi par des conditions if simples qui déclenchent une réaction correspondante mais les Event handlers facilitent certaines choses. Tout dépend en fait de vos préférences personnelles.

Le CD contient d'ailleurs quelques exemples de programmes plus détaillés sur ce sujet!

4.6.5. Lecture du CAN (Détecteurs du courant batterie, moteur et éclairage)

Comme nous l'avons déjà décrit au chapitre 2, beaucoup de détecteurs du RP6 sont branchés sur le CAN (Analogue to Digital Converter = Convertisseur analogique-numérique). Bien sûr, la RP6Library offre également une fonction qui permet de les lire:

uint16_t readADC(uint8_t channel)

Cette fonction renvoie une valeur à 10bit (0...1023), donc il nous faut une variable à 16bits pour les valeurs des détecteurs.

Les canaux suivants sont disponibles:

```
ADC_BAT           --> Détecteur de la tension de batterie
ADC_MCURRENT_R    --> Détecteur du courant moteur du moteur droit
ADC_MCURRENT_L    --> Détecteur du courant moteur du moteur gauche
ADC_LS_L          --> Détecteur photoélectrique gauche
ADC_LS_R          --> Détecteur photoélectrique droit
ADC_ADC0          --> Canal ADC libre pour autres détecteurs
ADC_ADC1          --> Canal ADC libre pour autres détecteurs
```

Attention: Les connecteurs pour les deux canaux CAN libres ne sont pas implantés. Vous pouvez y braser vos propres connecteurs dont les broches sont espacées de 2,54mm et ajouter éventuellement deux petits condensateurs de 100nF et un gros condensateur électrolytique de 470µF si vous voulez connecter des détecteurs qui nécessitent des courants crête élevés comme p.ex. des détecteurs de distance de chez Sharp... Pour cela, il vous faut cependant un peu d'expérience de brasage sinon il est préférable d'utiliser un module d'extension!

Exemples:

```
uint16_t ubat = readADC(ADC_BAT);
uint16_t iMotorR = readADC(ADC_MCURRENT_R);
uint16_t iMotorL = readADC(ADC_MCURRENT_L);
uint16_t lsL = readADC(ADC_LS_L);
uint16_t lsR = readADC(ADC_LS_R);
uint16_t adc0 = readADC(ADC_ADC0);
uint16_t adc1 = readADC(ADC_ADC1);

if(ubat < 580) writeString_P("Avertissement! Batterie presque vide!");
```

Par principe, la tension d'alimentation de 5V est utilisée comme référence standard. Il est également possible de décrire la fonction de telle façon que la référence interne de 2,56V de l'ATMEGA32 est utilisée (consultez à cet propos la fiche technique du MEGA32). Pour les détecteurs normaux du RP6, nous n'en avons pas besoin.

Il est judicieux de mesurer plusieurs valeurs CAN (à mémoriser p.ex. dans un tableau) et de calculer d'abord la moyenne ou de déterminer un minimum/maximum avant d'évaluer les valeurs mesurées des CAN car il arrive que des perturbations entraînent des erreurs de mesure. Dans le cas de la tension des accus, il est p.ex. pour un mode automatique d'économie absolument nécessaire de calculer la moyenne de quelques valeurs car la tension peut varier fortement, surtout si les moteurs tournent ou sont chargés en alternance.

Tout comme pour les pare-chocs, on peut automatiser la mesure du CAN. Il existe une

petite fonction qui nous facilitera un peu la vie.

```
void task_ADC(void)
```

Dans ce cas, elle raccourcit le temps qu'il faut pour la lecture de tous les canaux CAN dans le programme. Si on appelle constamment cette fonction, tous les canaux CAN sont lus l'un après l'autre « en tâche de fond » (lorsque le temps le permet) et les valeurs mesurées sont enregistrées dans des variables.

Le CAN requiert un peu de temps pour chaque mesure et la fonction readADC interromprait le déroulement normal du programme pendant ce temps. Puisque la mesure se fait aussi sans notre intervention (après tout le CAN est un module de matériel dans le microcontrôleur), nous pouvons nous occuper d'autres choses pendant ce temps.

Pour les différents canaux il existe les variables 16bit suivants que vous pouvez utiliser partout et toujours dans le programme:

```
ADC_BAT:          adcBat
ADC_MCURRENT_L:   adcMotorCurrentLeft
ADC_MCURRENT_R:   adcMotorCurrentRight
ADC_LS_L:         adcLSL
ADC_LS_R:         adcLSR
ADC_ADC0:         adc0
ADC_ADC1:         adc1
```

Dès que vous utilisez la fonction task_ADC, vous ne devez utiliser plus que ces variables et NON PAS la fonction read_ADC

Exemple:

```
#include "RP6RobotBaseLib.h"

int main(void)
{
	initRobotBase();
	startStopwatch1();
	writeString_P("\n\nPetit programme de mesure CAN...\n\n");
	while(true)
	{
		if(getStopwatch1() > 300) // Toutes les 300ms...
		{
			writeString_P("\nADC Détect. Photoélectr. gauche: ");
			writeInteger(adcLSL, DEC);
			writeString_P("\nADC Détect. Photoélectr. droit: ");
			writeInteger(adcLSL, DEC);
			writeString_P("\nADC Accu: ");
			writeInteger(adcBat, DEC);
			writeChar('\n');
			if(adcBat < 600)
				writeString_P("Attention! Accu bientôt vide!\n");
			setStopwatch1(0); // Remettre stopwatch1 à 0
		}
		task_ADC(); // Evaluation CAN - appeler constamment de la
boucle
	}			// principale - ne plus utiliser readADC !
	return 0;
}
```

Toutes les 300ms, les valeurs mesurées des deux détecteurs photoélectriques et de l'accu sont sorties. Si la tension des accus atteint *env.* 6V, un avertissement est émis.

4.6.6. ACS – Système Anti-Collision

A la différence du RP5, le système anticollision n'est plus piloté par un petit co-processeur mais directement par le MEGA32. Cela nécessite certes plus de programmation sur le contrôleur principal, mais présente l'avantage de pouvoir être modifié et amélioré à tout moment. Sur le RP5 le programme dans le co-processeur n'était pas modifiable...

La portée, à savoir la puissance de transmission des deux LED IR du système anticollision est réglable par les fonctions suivantes:

```
void setACSPwrOff(void)  --> ACS LED IR éteints

void setACSPwrLow(void)  --> Faible portée

void setACSPwrMed(void)  --> Portée moyenne
```

```
void setACSPwrHigh(void) --> Portée maximale
```

Si vous voulez évaluer l'ACS il faut appeler constamment la fonction:

```
void task_ACS(void)
```

qui pilote tout l'ACS. Le reste ressemble aux fonctions du pare-chocs.

Il y a deux variables:

```
obstacle_left und obstacle_right
```

qui ont chacune la valeur true dès qu'un obstacle a été détecté. Si les deux sont true (vrais), l'obstacle se trouve au milieu devant le robot.

Si vous voulez, vous pouvez également créer un Event Handler comme pour les pare-chocs mais ce n'est pas une obligation.

```
void ACS_setStateChangedHandler(void (*acsHandler)(void))
```

Cette fonction permet d'enregistrer l'Event Handler qui doit porter la signature suivante: `void acsStateChanged(void)`

Le nom exact de la fonction n'a d'ailleurs presque pas d'importance.

L'exemple suivant montre l'application. D'abord, il faut enregistrer l'Event Handler (ligne 44), ensuite mettre en service l'alimentation du récepteur IR (ligne 46 – sinon cela ne fonctionne pas!) et régler la force de transmission des LED IR de l'ACS (ligne 47). En bas de la boucle principale, la fonction task_ACS() est appelée constamment.

Le reste se fait automatiquement – la fonction acsStateChanged est toujours appelée lorsque l'état de l'ACS change – donc p.ex. lorsqu'un objet est signalé devant les détecteurs et disparaît. Le programme indique l'état actuel de l'ACS sous forme de texte sur le terminal et à l'aide des LED.

```
#include "RP6RobotBaseLib.h"

void acsStateChanged(void)
{
	writeString_P("L'état de l'ACS a changé!   L: ");

	if(obstacle_left)  // Obstacle à gauche
		writeChar('o');
	else
		writeChar(' ');

	writeString_P(" | R: ");

	if(obstacle_right) // Obstacle à droite
		writeChar('o');
	else
		writeChar(' ');

	if(obstacle_left && obstacle_right) // Milieu?
		writeString_P("   MILIEU!");
	writeChar('\n');

	statusLEDs.LED6 = obstacle_left && obstacle_right; // Milieu?
	statusLEDs.LED3 = statusLEDs.LED6;
	statusLEDs.LED5 = obstacle_left;     // Obstacle à gauche
	statusLEDs.LED4 = (!obstacle_left);  // Inversion de LED5!
	statusLEDs.LED2 = obstacle_right;    // Obstacle à droite
	statusLEDs.LED1 = (!obstacle_right); // Inversion de LED2!
	updateStatusLEDs();
}

int main(void)
{
	initRobotBase();

	writeString_P("\nRP6 Programme de test ACS\n");
	writeString_P("_________________________\n\n");

	setLEDs(0b111111);
	mSleep(1000);
	setLEDs(0b001001);

	// Enregistrer ACS Event Handler:
	ACS_setStateChangedHandler(acsStateChanged);

	powerON(); // Mettre sous tension le récepteur ACS (et l'encodeur
etc.)
	setACSPwrMed(); // Régler l'ACS sur une force de transmission
moyenne.

	while(true)
	{
		task_ACS(); // Appeler constamment la fonction task_ACS!
	}
	return 0;
}
```

```
51
52
53
54
```

Cet exemple illustre parfaitement comment on peut régler les LED une par une.

Lorsque l'on exécute le programme, il faudrait bien sûr laisser le RP6 branché sur l'ordinateur et visionner les sorties sur l'écran. Passez simplement la main ou un objet juste devant le robot!

Des sources de parasites peuvent diminuer la fonction de l'ACS. Certains tubes de néon peuvent rendre le robot quasiment « aveugle » ou tout du moins réduire la sensibilité. Si vous rencontrez des problèmes, commencez par éteindre toutes les sources lumineuses potentiellement gênantes. (Conseil: Si vous avez placé le robot juste devant un écran plat ou un téléviseur à écran plat, le problème peut venir de là. Après tout, c'est souvent un tube fluorescent qui est utilisé pour l'éclairage de fond...) La portée dépend fortement des objets eux-mêmes car une surface noire reflète la lumière IR moins bien qu'une surface blanche. Le RP6 ne reconnaît pas toujours très bien des objets très sombres!

De ce point de vue, il serait judicieux de renforcer l'ACS par des détecteurs à ultrasons ou mieux encore des détecteurs IR.

Ce que vous devez absolument faire avant de faire tourner le robot: Testez l'ACS avec différents objets pour savoir lesquels il ne reconnaît PAS très bien et les mettre éventuellement ailleurs pendant l'utilisation du robot. Cependant cela a moins d'importance maintenant que pour le prédécesseur puisqu'il y a maintenant les pare-chocs qui empêchent dans une large mesure que les LED IR soient déformées.

4.6.7. Fonctions IRCOMM et RC5

Le RP6 peut recevoir également des signaux de télécommandes TV/HiFi normales au moyen du récepteur IR. Cela ne peut fonctionner que si la télécommande émet en code RC5. La plupart des télécommandes universelles (voir fig.) peuvent être réglées sur ce format (voir pour cela le mode d'emploi de la télécommande). Si le code RC5 n'est pas explicitement mentionné dans le tableau des codes, essayez simplement quelques fabricants.

Si la télécommande émet au format RC5, ce signal est ignoré par la reconnaissance d'obstacle de l'ACS et l'ACS n'est que très peu ou pas du tout gêné. Il peut encore reconnaître des obstacles mais avec un peu de retard puisqu'il ne peut utiliser que les pauses de transmission RC5. Si la télécommande ne transmet pas au format RC5, le

code émis n'est pas reconnu et risque de perturber l'ACS.

Un logiciel adapté dans le microcontrôleur permettrait de contrôler complètement le RP6 à l'aide d'une télécommande et de lui envoyer des commandes!

Le système IRCOMM permet également d'envoyer des signaux IR. Les deux diodes émettrices sur le devant du robot sont dirigées en haut vers le plafond. Par le biais de la réflexion au plafond et d'autres objets ou s'il existe un contact visuel direct vers l'avant, on peut communiquer avec d'autres robots ou une station de base. Cela ne se fait que relativement lentement (un paquet de données requiert env. 20ms suivi d'une courte pause) mais des commandes simples et des valeurs de mesure individuelles arrivent à passer. La portée est évidemment un peu limitée et cela ne fonctionne que dans la même pièce avec une distance maximale entre les robots de 2 à 4m (en fonction de la luminosité, des obstacles, la nature du plafond et les extensions sur les robots). La portée peut être augmentée si vous ajoutez encore quelques LED IR (p.ex. au moyen d'un autre MOSFET, d'un gros condensateur et d'une petite pré-résistance).

En raison de la synchronisation nécessaire avec l'ACS, la fonction task_ACS() est également compétente pour ce type de tâches. Par ailleurs, elle est constamment appelée dans la boucle principale pour pouvoir réagir aux données IR reçues et pour effectuer les transmissions avec l'IRCOMM.

Les paquets de données RC5 contiennent toujours une adresse de matériel, le « keycode » et un « toggle bit ». L'adresse à 5 bits indique quel matériel doit être piloté. Lors d'une utilisation normale, cela permet de différencier entre la télécommande pour le téléviseur, le magnétoscope, la chaîne HIFI, etc. Pour notre application, cela permet p.ex. de piloter plusieurs robots. Le keycode à 6 bits est le code de touches sur une télécommande normale mais il permet également de transmettre des données. Certes seulement 6 bits mais on pourrait répartir des données à 8 bits sur deux transmissions ou bien utiliser encore 2 sur les 5 bits de l'adresse du matériel ou le toggle bit.

Le toggle bit sert normalement à différencier si une touche sur la télécommande est maintenue appuyée ou appuyée plusieurs fois de suite. Cependant, dans la transmission de robot à robot, son utilisation est libre.

Cette fonction permet d'envoyer des données RC5:

```
void IRCOMM_sendRC5(uint8_t adr, uint8_t data)
```

`adr` est l'adresse du matériel et `data` est le keycode ou les données. Dans `adr` vous

pouvez ajouter le toggle bit en remplaçant le bit le plus élevé de cet octet. A cet effet, vous pouvez utiliser la constante `TOGGLEBIT` de la manière suivante:

```
IRCOMM_sendRC5(12 | TOGGLEBIT, 40);
```

Cette commande envoie un paquet de données RC5 à l'adresse 12 avec un toggle bit défini et avec « 40 » comme données.

```
IRCOMM_sendRC5(12, 40);
```

Voilà à quoi cela ressemble si on ne veut PAS définir un toggle bit.

La réception de données RC5 fonctionne par le biais d'un Event handler, un peu comme pour les pare-chocs et l'ACS. L'Event handler est appelé automatiquement à partir de la fonction task_ACS() dès que de nouvelles données RC5 se présentent. Ainsi, lorsque le code de touche 4 est reçu, le robot pourrait tourner à gauche et avec le code 6 à droite...

Un des exemples de programme fait exactement cela: il dirige les mouvements du robot entièrement par la télécommande.

L'Event handler doit porter la signature suivante:

```
void receiveRC5Data(RC5data_t rc5data)
```

Le nom de la fonction n'a presque pas d'importance non plus!

```
void IRCOMM_setRC5DataReadyHandler(void (*rc5Handler)(RC5data_t))
```

Cette fonction permet d'enregistrer l'Event handler défini auparavant.

P.ex. de cette manière:

```
IRCOMM_setRC5DataReadyHandler(receiveRC5Data);
```

Ensuite cette fonction est appelée à chaque fois qu'un code RC5 valide a été reçu.

`RC5data_t` est un type de données spécialement défini qui contient les bits d'adresse

RC5 (Device Address), le toggle bit et le keycode (ou bien les données). Ils sont ensuite utilisables comme des variables ordinaires et portent les désignations suivantes:

```
rc5data.device, rc5data.toggle_bit, rc5data.key_code
```

Le CD contient un grand exemple de programme à ce sujet.

Attention: Ne mettez jamais la broche de l'IRCOMM sous tension en continu! Les LED IR et la commande sont conçues pour des impulsions et ne doivent rester sous tension qu'une millième de seconde à la fois, sinon le courant consommé est trop élevé lorsque les accus sont entièrement chargés. Ne modifiez pas les fonctions de l'IRCOMM vous-même si vous ne savez pas exactement ce que vous faites! Surtout la routine d'interruption qui est utilisée pour le pilotage, ne doit en aucun cas être modifiée!

4.6.8. Fonctions d'Economie d'Energie

Auparavant nous avons utilisé powerON() sans autres explications. Le RP6 permet d'éteindre l'ACS, les encodeurs, les détecteurs de courant moteur et la LED Power ON afin d'économiser de l'énergie (cela fait quand-même env. 10mA de moins lorsque l'on a pas besoin de ces détecteurs).

Il y a des macros:

```
powerON()
```

pour mettre les détecteurs mentionnés sous tension

```
powerOFF()
```

et pour tout mettre hors tension. Ces deux macros ne font rien d'autre que de commuter une broche I/O sur high ou low.

Avant de pouvoir utiliser l'ACS, l'IRCOMM et la régulation des moteurs, il faut impérativement appeler la macro powerON(), sinon les détecteurs ne sont pas alimentés en énergie. Ainsi la régulation des moteurs ne peut fonctionner correctement que si un signal vient des encodeurs et des détecteurs de courant.
Si vous oubliez d'appeler powerON(), les moteurs se coupent immédiatement après une brève tentative de démarrage et les quatre LED d'état rouges commencent à clignoter.

4.6.9. Fonctions d'Entraînement

Les fonctions qui existent déjà dans la RP6Library permettent de commander très aisément l'entraînement du robot, à savoir réguler automatiquement la vitesse des moteurs à l'aide des encodeurs, surveiller le courant moteur, parcourir automatiquement certaines distances et bien d'autres choses encore. C'est très agréable mais il faut, tout comme pour les autres systèmes, respecter quelques particularités si l'on veut utiliser ces fonctions.

La version actuelle n'est certes pas encore optimale et il reste beaucoup de choses à ajouter et à améliorer!

```
void task_motionControl(void)
```

La fonction **task_motionControl** doit être appelée constamment du programme princi-

pal sinon cette régulation automatique ne fonctionne pas. Appeler constamment du programme principal ne signifie rien d'autre qu'à chaque passage de boucle, il faut appeler cette fonction une fois dans la boucle principale du programme. Cela suffit si la fonction est appelée une fois toutes les 10 à 50 millièmes de seconde mais il est préférable d'appeler cette fonction beaucoup plus souvent. Cela ne pose aucun problème de l'appeler plus rapidement car le timing est contrôlé par un des programmateurs de matériel. C'est pourquoi cela n'a pas d'importance si la fonction est appelée à des intervalles réguliers ou si l'on a parfois besoin de 1ms ou 10ms entre les différents appels. Le temps de calcul supplémentaire nécessaire pour des appels plus fréquents est insignifiant. La fonction n'est exécutée en entier que s'il y a réellement une tâche à effectuer.

Si cette fonction est correctement utilisée, elle gère avec une précision relativement

élevée le nombre de tours souhaité des moteurs.

Cela se fait en calculant l'écart à chaque mesure de vitesse et en additionnant tous les écarts mesurés (donc un régulateur intégrant). Les modules MLI du microcontrôleur adaptent alors la tension aux moteurs à l'aide de cet écart. Si la vitesse est trop faible, les écarts sont positifs et la tension aux moteurs est augmentée en conséquence. Si la vitesse est trop élevée, la tension est diminuée. L'ensemble se stabilise assez rapidement sur une valeur MLI relativement constante (de petites variations sont normales). Cette méthode permet de réguler la vitesse indépendamment de la tension des accus, de la charge (poids, nature du sol, montée/descente, etc.) et des tolérances de fabrication. Si les canaux MLI étaient réglés sur une valeur fixe, la vitesse varierait fortement en fonction de la charge et de la tension des accus et serait différente pour chaque canal en raison des tolérances de fabrication qui existent toujours.

Le sens de rotation des moteurs est également pilotée par cette fonction car ce serait néfaste pour la durée de vie des moteurs et des engrenages si le sens de rotation changeait souvent subitement à 15cm/sec. Donc, si le sens de rotation doit changer, le robot freine à 0cm/sec, le sens de rotation change et ensuite seulement il accélère jusqu'à la vitesse cible.

Outre la régulation de vitesse et le contrôle du sens de rotation, le courant moteur est surveillé et les moteurs sont automatiquement coupés si l'un deux consommait trop de courant. Cela empêche une surcharge et une surchauffe des moteurs qui les endommagerait à terme. Après trois événements de surcharge en l'espace de 20 secondes, le robot est complètement arrêté et les quatre LED d'état rouges commencent à clignoter.

Il est également surveillé si l'un des encodeurs ou moteurs est défaillant (cela peut arriver si l'on les bricole de trop...), car si c'était le cas, la fonction motionControl augmenterait la valeur MLI au maximum et ferait éventuellement perdre le contrôle du robot... et ce ne serait pas souhaitable! Dans ce cas aussi, le robot est complètement arrêté.

Afin de conserver une certaine clarté, les routines pour parcourir des trajets ou tourner à un angle précis ont également été incluses dans cette fonction.

Vous voyez donc que cette fonction est extrêmement importante pour le fonctionnement automatique du système d'entraînement. La fonction motionControl elle-même ne contient d'ailleurs pas de paramètres telle que la vitesse cible. Cela se fait avec d'autres fonctions que nous aborderons maintenant en détail.

```
void moveAtSpeed(uint8_t desired_speed_left, uint8_t desired_speed_right)
```

Cette fonction permet de régler la vitesse cible. Les deux paramètres indiquent la vitesse pour le moteur gauche et droit. La vitesse sera réglée sur cette valeur si la fonction motionControl décrite ci-dessus est appelée constamment. Si la valeur est réglée sur 0, les modules MLI seront complètement désactivés lorsque le robot aura freiné.

getDesSpeedLeft() et **getDesSpeedRight()**

Ces macros permettent de lire la vitesse cible actuellement réglée.

L'application de la fonction moveAtSpeed est relativement simple – voici un exemple:

```
1  #include "RP6RobotBaseLib.h"
2
3  int main(void)
4  {
5        initRobotBase(); // Initialiser le microcontrôleur
6
7        powerON();  // Mettre sous tension les encodeurs et détecteurs
8                    // de courant moteur (IMPORTANT!!!)
9
10       moveAtSpeed(70,70); // Régler la vitesse
11
12       while(true)
13       {
14             // Appeler constamment la fonction motionControl de la boucle
15             // principale - elle règle les vitesses moteur:
16             task_motionControl();
17             task_ADC(); // Est appelé à cause des détecteurs de courant
18                         // moteur!
19       }
20       return 0;
21 }
```

... et le RP6 part! Mais il ne réagit pas du tout aux obstacles et roule simplement tout droit! Seule la vitesse est maintenue à un niveau relativement constant.

Le cas échéant, le robot augmente ou diminue automatiquement la puissance des moteurs, p.ex. lorsqu'il monte une pente.

ATTENTION: Cette particularité peut s'avérer dangereux pour vos doigts – Veillez à ne pas mettre vos doigts entre la chenille et les roues ou entre la platine et la chenille! Vous risquez de vous **blesser**! En cas d'obstacle, la puissance des moteurs est automatiquement augmentée et l'entraînement peut développer une force assez élevée.

Dans la fonction moveAtSpeed la vitesse n'est pas exprimée en cm/sec ou similaire mais en nombre de tours. C'est plus logique car la vitesse réelle dépend de la circonférence des roues/chaînes et donc de la résolution des encodeurs. Comme nous l'avons déjà vu au chapitre 2, il y a une marge de manœuvre relativement grande d'env. 0,23 à 0,25mm par incrément que nous devons d'abord calibrer.

Le nombre de tours est mesuré toutes les 200ms, donc 5x par seconde. C'est pourquoi l'unité est « segments d'encodeur par 200ms« ou autrement dit, « Segments d'encodeurs qui ont été comptés en 0,2sec. ». Pour une valeur de 70 comme dans le exemple de programme ci-dessus, cela ferait donc $70 \cdot 5 = 350$ segments/incréments par seconde (cela fait env. 8 à 8,7 cm/sec, en fonction de la résolution réelle de l'en-

codeur). Le nombre de tours minimum réglable est de $10 \cdot 5 = 50$ et le nombre maximum de $200 \cdot 5 = 1000$. Les fonctions du moteur limitent le nombre de tours à cette valeur maximale comme nous l'avons déjà vu dans le chapitre 2. Nous recommandons de ne pas dépasser les 160 pour une utilisation continue.

getLeftSpeed() et **getRightSpeed()**

Vous pouvez lire le nombre de tours mesuré par les encodeurs à l'aide de ces deux macros. L'unité de mesure utilisée est évidemment la même que celle décrite ci-dessus!

void changeDirection(uint8_t dir)

Cette fonction permet de changer/régler le sens de rotation des moteurs. D'abord le moteur freine, ensuite le sens change et après seulement il accélère jusqu'à la vitesse cible.

Les paramètres possibles sont:

FWD – pour Avancer (FWD est une abréviation pour „Forwards")
BWD – pour Reculer (BWD est une abréviation pour „Backwards")
LEFT – tourner à gauche
RIGHT – tourner à droite

Par le biais du macro:

getDirection()

vous pouvez lire le sens actuellement réglé.

Exemple:

```
1  #include "RP6RobotBaseLib.h"
2
3  int main(void)
4  {
5        initRobotBase(); // Initialiser le microcontrôleur
6        powerON();       // Mettre sous tension encodeurs et détecteurs de
7                         // courant moteur!
8
9        moveAtSpeed(60,60); // Régler la vitesse
10       startStopwatch1();  // Démarrer Stopwatch1
11
12       while(true)
13       {
14             if(getStopwatch1() > 4000) // Sont 4000ms (= 4s) passées?
15             {
16                   // Changer le sens de rotation:
17                   if(getDirection() == FWD) // Si nous avançons...
18                         changeDirection(BWD); // régler sur Reculer!
19                   else if(getDirection() == BWD) //... ou reculons?
                           changeDirection(FWD); // Régler sur Avancer!
                     setStopwatch1(0); // Mettre Stopwatch1 à 0
             }
             task_motionControl(); // Réglage automatique de l'entraînement
```

```
20              task_ADC(); // Appelé à cause des détecteurs de courant
21                          moteur!
22      }
23      return 0;
24  }
25
26
```

Dans cet exemple, le RP6 avance tout droit – c'est toujours le cas après une remise à zéro. Nous utilisons l'un des chronomètres pour attendre 4 secondes et changer ensuite la direction. En lignes 16 et 18, on demande dans quel sens nous nous déplaçons actuellement et la direction est inversée. Cela se répète toutes les 4 secondes et le robot avance et recule pendant tout ce temps.

Et il ne réagit toujours PAS aux obstacles!

Les fonctions abordées jusqu'à présent, ne permettent pas de parcourir des distances déterminées. Pour cela, il existe deux fonctions spécifiques:

```
void move(uint8_t desired_speed, uint8_t dir, uint16_t distance,
                  uint8_t blocking)
```

La fonction move permet de parcourir une distance prédéterminée en ligne droite. Il faut indiquer la vitesse cible, la direction (FWD ou BWD) et la distance en incréments d'encodeur.

Avec la macro:

```
DIST_MM(DISTANCE)
```

on peut convertir une distance en millimètres en incréments d'encodeur. Pour cela, il faut avoir calibré la résolution des encodeurs (voir annexe). L'exemple ci-après illustre cette application.

Le robot va essayer de parcourir le trajet souhaité avec le plus de précision possible. La fonction motionControl fait accélérer le robot jusqu'à la vitesse souhaitée. Un peu avant l'arrivée, le robot freine pour ne pas aller trop loin. Cela ne peut se faire qu'à une précision de plus ou moins 5mm ce qui est suffisant dans la plupart des cas.

Cette fonction ne peut pas gérer de très courtes distances avec une grande précision. La distance devra être d'au moins 5cm mais cela pourra être largement amélioré.

Le dernier paramètre de la fonction - »blocking »- est une particularité qui mérite une explication plus détaillée.

La fonction ne pose normalement que quelques variables et le déroulement normal du programme se poursuivrait. Le parcours serait contrôlé automatiquement « en tâche de fond » par la fonction motionControl. C'est pratique lorsque le robot doit faire d'autres choses en parallèle tel que réagir aux obstacles. S'il ne s'agit que de parcourir un simple trajet, on peut l'influencer par le paramètre « blocking ».

Si le paramètre est « true » (donc 1), la fonction appelle autant de fois la fonction motionControl dans une boucle jusqu'à ce que le trajet souhaité ait été parcouru. La fonction ne quitte pas tout simplement mais bloque le déroulement normal du programme pendant tout ce temps.

Si le paramètre est « false », la fonction se comporte comme décrit ci-dessus et quitte immédiatement après avoir donné « l'instruction de parcourir un trajet ». Si l'on appelle d'autres fonctions pendant ce temps telles que fixer la vitesse, donner de nouvelles commandes de conduite ou autres, le programme ne pourrait pas fonctionner comme nous le voulons.

La fonction

```
uint8_t isMovementComplete(void)
```

permet de vérifier si le mouvement est terminé. Si la fonction retourne « false », le mouvement est en cours. S'il est terminé, de nouvelles commandes peuvent être données.

S'il faut arrêter la fonction prématurément p.ex. à cause d'un obstacle, vous pouvez arrêter tous les mouvements à l'aide de la fonction:

```
void stop(void)
```

et stopper le robot.

La fonction ci-dessus pourrait même servir à la rotation autour de certains angles en indiquant comme paramètre directionnel LEFT ou RIGHT au lieu de FWD ou BWD et en donnant la distance qui correspond à l'angle souhaité. Cependant c'est très compliqué et ne fonctionne pas très bien. C'est pourquoi il existe une fonction spécifique qui est écrite pour la rotation sur place:

```
void rotate(uint8_t desired_speed, uint8_t dir, uint16_t angle,
                 uint8_t blocking)
```

Cela fonctionne exactement comme la fonction move. La seule différence est que l'indication de la distance à parcourir a été remplacée par un angle. Le comportement de cette fonction est également commandé par le paramètre « blocking ».

Cet exemple montre comment les deux fonctions peuvent être utilisées:

```
1   #include "RP6RobotBaseLib.h"
2
3   int main(void)
4   {
5         initRobotBase();
6         setLEDs(0b111111);
7         mSleep(1500);
8
9         powerON(); // Mettre sous tension encodeurs et détecteurs de
10                      courant moteur!
11
12        while(true)
```

```
11      {
12            setLEDs(0b100100);
13            move(60, FWD, DIST_MM(300), true); // Avancer de 30cm
14            setLEDs(0b100000);
15            rotate(50, LEFT, 180, true);  // Tourner à gauche à 180°
16            setLEDs(0b100100);
17            move(60, FWD, DIST_MM(300), true); // Avancer de 30cm
18            setLEDs(0b000100);
19            rotate(50, RIGHT, 180, true); // Tourner à droite à 180°
20      }
21      return 0;
22}
23
```

Le robot avance d'abord de 30cm, tourne à 180° vers la gauche, recule de 30cm, tourne à 180° vers la droite et recommence la manœuvre depuis le début. Si nous mettions le paramètre « blocking » sur false au lieu de true, le programme ne fonctionnerait pas du tout car la fonction task_motionControl ne serait pas appelée dans la fonction Main et tous les appels de fonction se font directement à la suite l'un de l'autre. Si un seul des paramètre « blocking » était modifié en false, le programme ne fonctionnerait plus non plus comme prévu puisqu'une des phases de mouvement serait complètement laissée de côté.

Pour des processus purement séquentiels, il faut toujours mettre le paramètre « blocking » sur true!
Il est même possible de réagir d'une manière rudimentaire aux obstacles et autres choses quand le paramètre « blocking » est fixé sur true – par le biais de l'Event handler. Ils seront appelés quand-même! Mais cela ne fonctionne pas forcément pour des problèmes plus complexes.

En règle générale, lorsque vous voulez éviter des obstacles, interpréter des commandes provenant d'une commande master ou autres, il faut utiliser le mode « non-blocking », donc mettre le paramètre sur false.
Comme nous l'avons dit plus haut, les fonctions move/rotate règlent le trajet du robot indépendamment du déroulement du reste du programme

Sur le CD se trouvent quelques exemples plus complets sur ces fonctions.

4.6.10. task_RP6System()

Dans les chapitres précédents, nous avons vu que quatre fonctions doivent être appelées constamment à partir de la boucle principale pour que l'ACS/IRCOMM, moteurs et pare-chocs puissent fonctionner correctement et que les CAN sont évalués en tâche de fond. Afin d'économiser un peu de travail d'écriture et gagner en clarté, la fonction:

```
void task_RP6System(void)
```

a été introduite. Elle appelle les fonctions:

```
task_ADC();
task_ACS();
task_bumpers();
task_motionControl();
```

dans l'ordre. Dans la plupart des programmes sur le CD, vous ne trouverez que ces fonctions, les quatre autres ne sont utilisées que rarement directement et généralement appelées par cette fonction.

4.6.11. Fonctions du Bus I²C

A la fin de ce chapitre, nous arrivons aux fonctions du bus I²C qui sont nécessaires pour la communication avec d'autres microcontrôleurs et modules d'extension.

Il existe deux versions des fonctions I²C, une pour le mode esclave et une pour le mode maître.

Attention: Vous ne pouvez pas utiliser les deux versions en même temps!

Vous pouvez inclure une des deux versions et la déclarer dans le Makefile! Les déclarations correspondantes y existent déjà – mais commentées. Vous ne devez utiliser qu'une des deux déclarations sinon le compilateur sort une erreur (parce que le vecteur d'interruption TWI serait défini deux fois...).

4.6.11.1. Esclave I²C

Le mode esclave est très important dans la base du robot car on utilisera certainement souvent un autre microcontrôleur pour la commande du robot. Il existe aussi un exemple de programme qui offre pratiquement toutes les fonctions de la base du robot via le bus I²C (RP6Base_I2CSlave).

Tout comme le mode maître, le mode esclave est basé sur une routine d'interruption. Concernant l'esclave, on ne peut pas faire autrement (toutefois sans grande difficulté), quant au maître, cela pourrait s'organiser aussi autrement. Afin de conserver une certaine homogénéité, les deux versions sont basées sur l'interruption. Les transmissions en mode maître pourront se dérouler en partie en tâche de fond afin de gagner un peu de temps.

```
void I2CTWI_initSlave(uint8_t address)
```

Cette fonction initialise le module TWI du microcontrôleur en tant que esclave I²C. Vous pouvez déterminer ici l'adresse du contrôleur. L'adresse indique en même temps si le contrôleur doit réagir à ce que l'on appelle des « General Calls ». Si le bus I²C est adressé avec 0, cela s'appelle General Call et tous les appareils connectés peuvent y réagir. C'est utile p.ex. pour commuter tous les contrôleurs en même temps en mode d'économie d'énergie.

Exemples:

```
I2CTWI_initSlave( adr |  TWI_GENERAL_CALL_ENABLE ); // General call activé
I2CTWI_initSlave(adr); // General call NON activé
```

Registre I²C

Les périphériques courants du bus I²C sont généralement commandés par quelques registres que l'on peut lire et/ou écrire. C'est pour cela que les routines esclaves sont conçues de telle façon que le bus maître peut lire et écrire ces « registres » (dans ce cas un tableau composé de variables à 8 bit). En plus de l'adresse esclave, le maître transmet une adresse de registre et peut lire et écrire ensuite les données de ce registre.

A cet effet, il existe deux tableaux avec le type de données uint8_t. L'un d'eux met à disposition les registres lisibles et les autres les inscriptibles.

Les deux tableaux et la variable pour les General Calls s'appellent:

```
I2CTWI_readRegisters, I2CTWI_writeRegisters et I2CTWI_genCallCMD
```

I2CTWI_readRegisters sont les registres lisibles et les I2CTWI_writeRegisters sont les registres inscriptibles. I2CTWI_genCallCMD mémorise la dernière commande General Call reçue. En mode esclave, tous les échanges de données par le bus I²C se déroulent par ces registres. Afin de rendre les données disponibles sur le bus I²C, il faut les inscrire dans le tableau I2CTWI_readRegisters. Ensuite, un maître peut y accéder par la position du tableau (= numéro du registre). Si un maître doit p.ex. recevoir des valeurs d'un détecteur de l'esclave, il suffit de les écrire dans des positions préalablement fixées dans le tableau I2CTWI_readRegisters. Ensuite le maître peut lire ces données en transmettant d'abord le numéro du registre à l'esclave pour lire ensuite les fichiers. Le numéro du registre est incrémenté automatiquement, donc le maître peut lire plusieurs registres en même temps.

L'écriture fonctionne d'une manière similaire. Le maître transmet d'abord le numéro du registre cible souhaité et commence ensuite à écrire les données. Pour écrire dans plusieurs registres à la suite, inutile de transmettre le numéro pour chaque registre. Vous continuez simplement à écrire puisque le numéro de registre est incrémenté après chaque accès.

Voilà ce qui se passe dans l'esclave en tâche de fond entièrement basé sur la routine d'interruption.

Lors de l'écriture dans l'esclave, des inconsistances peuvent facilement se produire si l'on utilise directement les données des registres. Pendant que l'on lit un registre, le maître peut avoir déjà écrasé un autre. C'est souvent judicieux de mémoriser temporairement les données avant leur utilisation. Des inconsistances peuvent également survenir lors de la lecture quand plusieurs variables dépendant l'une de l'autre (p.ex. Low et high Byte d'une variable 16 bit).

```
I2CTWI_readBusy et I2CTWI_writeBusy
```

Lors des accès pour écriture par le maître, la routine d'interruption met la variable I2CTWI_writeBusy sur true. Cela permet de vérifier si une écriture est en cours. Ensuite les données des registres peuvent être écrites dans des variables temporaires avec lesquelles vous pouvez continuer à travailler.

Le exemple de programme sur la page suivante et le exemple de programme esclave sur le CD le démontrent: Il y a un registre de commandes par lequel le maître peut envoyer des commandes à l'esclave du genre « avance à la vitesse 100 ». A cet effet, l'esclave évalue constamment le registre 0 dans la boucle principale lorsque I2CTWI_busy est false. Si le registre 0 a reçu une commande du maître, le registre 0 ainsi que les registres 1 à 6 sont écrits dans des variables temporaires qui seront évalués ultérieurement. Les paramètres sont traités en fonction du contenu de la variable de commande. Ainsi, le paramètre 1 pourrait contenir la vitesse lors d'une commande de mouvement et le paramètre 2 la direction du mouvement. Les autres paramètres seraient ignorés dans cette commande.

La variable I2CTWI_readBusy fonctionne de la même façon. Elle est posée lorsqu'un registre est lu afin de vérifier si l'on peut écrire dans le registre sans provoquer des inconsistances. Cependant ce n'est pas garanti à 100%. Pour éviter complètement des inconsistances, il faudrait désactiver le TWI Interrupt pendant la durée de l'écriture des registres ce qui risquerait par contre de créer d'autres problèmes...

Voici un exemple de programme:

```
1  #include "RP6RobotBaseLib.h"
2  #include "RP6I2CslaveTWI.h"  // Inclure le fichier I2C Library  (!!!)
3     // ATTENTION: Il faut aussi le déclarer dans le Makefile (!!!)
4
5  #define CMD_SET_LEDS 3 // Commande à LED qui doit être reçue par
6                         // le bus I2C.
7  int main(void)
8  {
9        initRobotBase();
10       I2CTWI_initSlave(10); // Initialiser le TWI et régler l'adresse 10
11       powerON();
12
13       while(true)
14       {
15             // Réception d'une commande et PAS d'écriture active?
16             if(I2CTWI_writeRegisters[0] && !I2CTWI_writeBusy)
17             {
18                   // Mémoriser le registre:
19                   uint8_t cmd = I2CTWI_writeRegisters[0];
20                   I2CTWI_writeRegisters[0] = 0; // et remettre à zéro (!!!)
21                   uint8_t param = I2CTWI_writeRegisters[1]; // Paramètre
22
23                   if(cmd == CMD_SET_LEDS) // Recevoir commande à LED?
24                         setLEDs(param); // Déclarer LED avec le paramètre
25             }
26             if(!I2CTWI_readBusy) // Pas de lecture active?
27                   // Ecrire l'état actuel de la LED dans le registre 0:
28                   I2CTWI_readRegisters[0] = statusLEDs.byte;
29       }
30       return 0;
31 }
```

Le programme ne fait rien de lui-même. Il nous faut impérativement un maître qui pilote le contrôleur comme esclave. Dans cet exemple, l'esclave est accessible à l'adresse 10 du bus I²C (voir ligne 10). Il n'y a que deux registres inscriptibles et un registre lisible. Le premier registre (= numéro de registre 0) est un registre de commandes qui peut recevoir des commandes. Dans cet exemple simple, ce n'est que la

commande « 3 » pour déclarer les LED (peut être un chiffre quelconque). Si une commande quelconque est reçue et aucune écriture n'est active (ligne 16), la valeur du registre de commande est d'abord mémorisée dans la variable cmd (ligne 19) et ensuite le registre de commande est remis à zéro afin que la commande ne soit pas exécutée une deuxième fois! Ensuite le paramètre dans le registre 1 est mémorisé dans une variable, suivi d'une interrogation si la commande 3 a été reçue (ligne 23). Si c'est le cas, les LED sont déclarées avec la valeur du paramètre.
La ligne 26 interroge si aucune écriture n'a lieu et mémorise, le cas échéant, la valeur actuelle des LED dans le registre de lecture 0.
Si le contrôleur sur le robot a chargé ce programme, un contrôleur maître peut déclarer les LED sur le robot via le bus I²C et lire leur état actuel.

A part cela, le programme ne fait rien. Le CD contient un exemple plus complet qui permet de piloter pratiquement tout ce qui se trouve sur le robot. Cet exemple-ci ne montre que le principe de fonctionnement.

16 registres inscriptibles et 48 registres lisibles existent en standard. Si vous avez besoin de plus ou de moins, vous pouvez modifier ces valeurs dans le fichier RP6I2CSlaveTWI.h de la RP6Library.

4.6.11.2. I²C Maître

En mode maître, le module TWI du MEGA32 permet de commander d'autres matériels/microcontrôleur/détecteurs par le bus I²C.

```
void I2CTWI_initMaster(FREQ)
```

Cette fonction initialise le module TWI comme maître. Il n'est pas nécessaire d'indiquer une adresse pour le mode maître mais il faut déterminer une fréquence avec laquelle le module TWI doit envoyer les données. Le paramètre FREQ permet de régler la fréquence en kHz. On utilise généralement 100kHz, donc le mieux est d'écrire 100. Si vous voulez aller un peu plus vite, vous pouvez aller jusqu'à 400 mais pas au-delà car c'est la vitesse maximale spécifiée du module TWI.

Selon les spécifications d'Atmel, le module TWI du MEGA32 sur la platine principale ne peut être utilisée que jusqu'à env. 220kBit/sec. en mode maître (voir fiche technique). Pour des raisons d'économie d'énergie, le contrôleur est réglé sur une vitesse d'horloge de 8MHz. Pour 400kBit/sec. il faudrait une vitesse d'horloge de plus de 14,4 MHz. En dessous de 14,4MHz, le timing peut facilement différer des spécifications ce qui n'est pas très important pour la plupart des appareils I²C. Pour le mode esclave cela n'a aucune importance puisque 400kBit/sec. sont atteints sans aucun problème. Pour toutes les applications normales, les 220kBit/sec. garantis sont plus que suffisamment rapides. Si vous avez réellement besoin de vitesses plus élevées, vous pouvez utiliser le module d'extension RP6 CONTROL M32. Il peut exploiter les 400kBit/sec maximales dans le mode maître. Ou vous testez simplement si les appareils qui sont connectés sur le bus de votre robot gèrent sans problème le timing généré par le MEGA32.

Envoi de données

Il existe plusieurs fonctions pour transmettre des données via le bus I²C. Leur principe de fonctionnement est très similaire. Ils ne se différencient que par le nombre d'octets à transmettre.

```
void I2CTWI_transmitByte(uint8_t adr, uint8_t data)
```

Envoie un seul octet à l'adresse indiquée.

```
void I2CTWI_transmit2Bytes(uint8_t adr, uint8_t data1, uint8_t data2)
```

Envoie deux octets à l'adresse indiquée.

Cette fonction est utilisée très souvent car de nombreux appareils attendent les données au format:

Adresse Esclave – Adresse du registre – Données

```
void I2CTWI_transmit3Bytes(uint8_t adr, uint8_t data1, uint8_t data2,
uint8_t data3)
```

Est également fréquemment utilisé surtout dans le cadre du programme esclave décrit ci-dessus. Ici vous pouvez envoyer les données au format

Adresse Esclave – Registre de commande – Commande - Paramètre1

```
void I2CTWI_transmitBytes(uint8_t targetAdr, uint8_t *msg,
                          uint8_t numberOfBytes)
```

Cette fonction transmet en standard jusqu'à 20 octets à l'adresse indiquée. Si vous voulez envoyer plus d'octets à la file, vous pouvez augmenter l'inscription

I2CTWI_BUFFER_SIZE dans le fichier d'en-tête.

S'il faut indiquer un registre auquel il faut envoyer les données, vous pouvez utiliser tout simplement le premier octet du buffer.

L'application de ces fonctions est très simple. Exemple:

```
I2CTWI_transmit2Bytes(10, 2, 128);
I2CTWI_transmit2Bytes(10, 3, 14);
I2CTWI_transmit3Bytes(64, 12, 98, 120);
```

Deux octets sont envoyés deux fois de suite à l'esclave qui porte l'adresse 10 et puis encore trois à l'esclave portant l'adresse 64.

Les autres fonctions transmitXBytes fonctionnent d'une façon analogique.

Un autre exemple:

```
uint8_t messageBuf[4];
messageBuf[0] = 2; // Indiquer le registre à écrire éventuellement...
messageBuf[1] = 244;  // Données...
messageBuf[2] = 231;
messageBuf[3] = 123;
messageBuf[4] = 40;
I2CTWI_transmitBytes(10,&messageBuf[0],5);
```

De cette façon, vous pouvez envoyer plusieurs octets (5 dans cet exemple) via le bus I²C.

Les fonctions indiquées bloquent le déroulement du programme seulement si l'interface I²C est active. Le programme attend jusqu'à ce que la transmission soit complète. Si on l'interroge avant d'appeler la fonction, on peut transmettre des données et effectuer d'autres tâches pendant la transmission.

Les transferts via le bus I²C durent relativement longtemps comparé à la vitesse du microcontrôleur.

La macro

I2CTWI_isBusy()

indique si le module TWI est en cours d'utilisation, sinon vous pouvez transmettre de nouvelles données.

Réception de Données

Pour la réception des données, la RP6Library offre plusieurs possibilités. D'une part des fonctions de blocage qui sont construites d'une manière similaire aux fonctions d'écriture et, d'autre part, des fonctions qui appellent les données automatiquement « dans le fond ».

Voyons tout d'abord la variante simple qui consiste à lire les données avec blocage.

```
uint8_t I2CTWI_readByte(uint8_t targetAdr);
```

Cette fonction lit un seul octet d'un esclave. Elle ne peut pas être utilisée seule, il faut presque toujours transmettre d'abord le numéro du registre. Cela se fait avec I2CTWI_transmitByte.

P.ex. si l'on veut lire le registre 22 de l'esclave à l'adresse 10:

```
I2CTWI_transmitByte(10, 22);
uint8_t result = I2CTWI_readByte(10);
```

```
void I2CTWI_readBytes(uint8_t targetAdr, uint8_t * messageBuffer,
                      uint8_t numberOfBytes);
```

Si vous voulez appeler plusieurs octets, vous pouvez utiliser cette fonction.

Exemple:

```
I2CTWI_transmitByte(10, 22);
uint8_t results[6];
I2CTWI_readBytes(10,results,5);
```

Ici sont lus 5 octets du registre 22 de l'esclave à l'adresse 10. S'ils sont vraiment lus dans le registre 22 varie d'un esclave à l'autre. Certains incrémentent automatiquement le numéro du registre (comme le code esclave de la RP6Library) et autres fonctions d'une façon tout à fait différente. A ce propos, vous devez toujours lire la documentation qui s'y rapporte!

L'affaire devient plus complexe lorsque l'on veut lire des données en tâche de fond. Il faut lancer d'abord une requête (request) après un certain nombre d'octets sur un esclave donné. L'esclave va chercher ensuite ces octets en tâche de fond. Le contrôleur peut effectuer d'autres tâches pendant ce temps et n'est interrompu que de temps en temps par la routine d'interruption. Il faut toutefois appeler constamment une fonction dans la boucle principale qui vérifie si les données demandées ont été reçues par l'esclave ou si une erreur s'est éventuellement produite. Si c'est le cas, cette fonction appelle automatiquement les fonctions Event handler qui ont été réglées auparavant et on peut traiter les données reçues et le cas échéant demander immédiatement d'autres données des autres registres. Chaque requête reçoit une ID pour pouvoir être affectée.

```
void task_I2CTWI(void)
```

C'est la fonction qui doit être appelée constamment de la boucle principale. Elle vérifie si le transfert a été effectué complètement sans erreurs et appelle les Event handlers en fonction du résultat.

```
void I2CTWI_requestDataFromDevice(uint8_t requestAdr, uint8_t requestID,
                                  uint8_t numberOfBytes)
```

Cette fonction permet de demander des données à un esclave. Après avoir appelé cette fonction, le processus se déroule automatiquement en tâche de fond, comme décrit ci-dessus.

Ensuite vous pouvez appeler les données avec la fonction:

```
void I2CTWI_getReceivedData(uint8_t *msg, uint8_t msgSize)
```

On peut appeler les données dès que l'Event handler correspondant est appelé.

Il faut enregistrer celui-ci avec la fonction:

```
void I2CTWI_setRequestedDataReadyHandler(void (*requestedDataReadyHandler)
(uint8_t))
```

L'Event Handler doit porter la signature

```
void I2C_requestedDataReady(uint8_t dataRequestID)
```

L'Event handler reçoit l'ID que l'on a attribuée à la demande de données correspondante. Elle permet de différencier les accès aux différents esclaves.

Outre le requestedDataReady Handler, il existe un autre Event handler qui est appelé en cas d'erreurs. Si quelque chose se passe mal pendant la transmission, p.ex. quand l'esclave ne répond pas, c'est cet Event handler qui est appelé.

On enregistre l'Event handler avec cette fonction:

```
void I2CTWI_setTransmissionErrorHandler(void (*transmissionErrorHandler)
(uint8_t))
```

Il doit porter la signature:

```
void I2C_transmissionError(uint8_t errorState)
```

Le paramètre est un code d'état d'erreur. La signification du code d'erreur est expliquée dans le fichier d'en-tête du mode maître I²C.

L'utilisation de cet Event handler ne dépend pas du fait si les données sont appelées dans le fond ou non. Même avec les fonctions de blocage, cet Event handler est appelé en cas d'erreur.

Vous trouverez quelques exemples de programmes sur le mode maître sur le CD. Dans le chapitre suivant, les exemples de programmes sont abordés plus en détail.

4.7. Exemples de Programmes

Le CD contient quelques exemples de programmes simples qui illustrent les fonctions de base du robot. La plupart d'entre eux ne sont ni très complexes, ni ne constituent la solution optimale. Ils ne s'entendent que comme point de départ pour vos propres programmes. C'est fait intentionnellement pour qu'il vous reste un peu de travail à faire. Ce serait quand-même ennuyeux de se limiter à essayer des programmes tout faits!

Certains des programmes s'adressent plutôt à des utilisateurs avancés. C'est le cas notamment des programmes comportementaux où le robot doit se comporter p.ex. comme un insecte. Ainsi dans un des programmes, ils se comporte un peu comme un papillon de nuit qui cherche la lumière et évite les obstacles. Une explication plus complète dépasserait largement le cadre de ce manuel et nous devons vous renvoyer à la littérature existante à ce sujet.

Internet vous permet d'échanger vos propres programmes avec d'autres utilisateurs. La RP6Library ainsi que tous les exemples de programmes sont placés sous une licence open source « GPL » (General Public licence) et vous êtes donc autorisé à modifier, publier et mettre à la disposition de tiers vos propres programmes sous les conditions de la GPL dans la mesure où vous placez vos programmes sous la licence GPL.

En fait, de nombreux exemples de programmes pour le MEGA32 existent déjà puisque le contrôleur de la famille AVR est très populaire auprès des utilisateurs amateurs. Toutefois, il faut toujours veiller à adapter d'autres exemples de programmes au matériel du RP6 et à la RP6Library, sinon ils risquent de ne pas fonctionner correctement (les problèmes les plus évidents sont d'autres affectations de broche, l'utilisation de modules de matériel déjà occupés à autre chose tels que des programmateurs, une autre fréquence d'horloge, etc.).

Tous les exemples de programmes à l'exception de ceux qui utilisent le bus I²C, sont conçus pour ne tourner que sur le robot de base – donc sans modules d'extension. Même si ce n'est pas un problème en règle générale, vous ne devriez monter les modules d'extension sur le robot que lorsque vous aurez essayé tous les programmes et que vous vous êtes familiarisé avec les possibilités du robot de base.

Chaque module d'extension programmable doit être livré avec l'exemple de programme correspondant ou bien déjà exister sur notre page d'accueil sur Internet (pour le RP6 CONTROL M32, ils sont même déjà inclus sur le CD!).

D'ailleurs la programmation d'un grand nombre de modules d'extension est plus simple parce qu'ils ne contiennent que peu d'éléments qui doivent être exécutés en simultané et qui exigent par conséquent une précision d'horloge absolue. Cela simplifie la programmation de base p.ex. du RP6-M32 puisqu'il y a moins de particularités à prendre en compte.

En outre, le RP6-M32 dispose de plus de temps de calcul et de mémoire pour d'autres choses.

Exemple 1: Programme „Hello World" avec séquenceur à LED
Répertoire: <RP6Examples>\RP6BaseExamples\Example_01_LEDs
Fichier: RP6Base_LEDs.c

Le programme génère des sorties sur l'interface série. Vous devez donc connecter le robot sur le PC et regarder les sorties sur le terminal du logiciel RP6-Loader!

Dans ce programme, le robot ne se déplace pas! Vous pouvez donc le placer sur une table à côté du PC.

Cet exemple de programme sort un texte court « Hello World » via l'interface série et exécute ensuite une séquence de lumière.

La petite particularité de la séquence de lumière est l'opérateur « shift left » que nous avons rencontré déjà auparavant mais sans l'expliquer plus en détail:

```
1  setLEDs(runningLight); // Définir la lED
2  runningLight <<= 1;    // LED suivante (Opération shift-left)
3  if(runningLight > 32)  // Dernière LED?
4      runningLight = 1;  // Oui, donc recommencer dès le début!
```

C'est ce que que nous allons faire maintenant. Une opération « shift left », donc « << » (ligne 2) permet de déplacer les bits dans une variable par un nombre donné vers la gauche. Cela fonctionne aussi vers la droite avec l'opérateur shift right « >> ». Donc `runningLight <<= 1;` déplace tous les bits dans la variable runningLight d'une

position vers la gauche. La ligne ci-dessus n'est d'ailleurs qu'une abréviation pour `runningLight = runningLight << 1;`

tout comme on peut le faire aussi avec += ou *=.

La variable runningLight a d'abord la valeur de 1 ce qui signifie qu'un seul bit a été défini dans la variable. A chaque opération shift, ce bit se déplace d'une position vers la gauche. Puisque nous utilisons cette variable pour définir les LEDs, cela résulte en un point lumineux « courant » d'où le nom anglais « running light ». Afin de permettre à l'œil humain de percevoir ce point lumineux, nous utilisons un mSleep qui génère une pause de 100ms après chaque passage de boucle.

Si le bit 7 a été défini dans la variable runningLight (elle a donc une valeur de >32), nous devons retourner au début et ne définir qu'un seul bit (lignes 3+4).

Exemple 2: Suite de l'interface série
Répertoire: <RP6Examples>\RP6BaseExamples\Example_02_UART_01
Fichier: RP6Base_SerialInterface_01.c

Le programme génère des sorties sur l'interface série

Le robot ne se déplace pas dans ce programme!

Cet exemple illustre l'utilisation des fonctions writeInteger et writeIntegerLength et

fait sortir quelques chiffres en différents formats via l'interface série.

Par ailleurs, nous présentons ici la variable « timer » qui est nouvelle dans la version 1.3 de la RP6Lib et qui peut être utilisée pour des mesures de temps avec une résolution de 100µs.

Exemple 3: Programme Question-Réponse
Répertoire: <RP6Examples>\RP6BaseExamples\Example_02_UART_02
Fichier: RP6Base_SerialInterface_02.c

Le programme génère des sorties sur l'interface série

Le robot ne se déplace pas dans ce programme!

Cet exemple un peu plus complexe est un petit programme de Question-Réponse. Le robot pose quatre questions simples auxquelles vous pouvez répondre à votre guise. Le robot réagit p.ex. par une réponse en texte ou il exécute une routine de séquence lumineuse. Le programme n'est rien de plus qu'une illustration de la réception et du traitement de données provenant de l'interface série. En outre, il présente la fonction writeStringLength et constitue un autre exemple de la construction switch-case.

Les anciennes fonctions relatives à la réception de données via l'interface série ont été remplacées par des nouvelles nettement meilleures de la RP6Lib. Ce programme démontre leur application. Maintenant c'est devenu relativement facile de traiter des lignes de commandes complètes et de réagir mieux à des commandes erronées.

Exemple 4: Programme de démonstration des Stopwatches (chronomètres)
Répertoire: <RP6Examples>\RP6BaseExamples\Example_03_Stopwatches
Fichier: RP6Base_Stopwatches.c

Le programme génère des sorties sur l'interface série

Le robot ne se déplace pas dans ce programme!

Ce programme utilise quatre stopwatches: une pour générer une séquence de lumière à 100ms d'intervalle et les autres pour trois compteurs différents qui augmentent leur position de comptage à des intervalles différents et les sortent par l'interface série.

Exemple 5: Programme de démonstration ACS & Bumper
Répertoire: <RP6Examples>\RP6BaseExamples\Example_04_ACS
Fichier: RP6Base_ACS.c

Le programme génère des sorties sur l'interface série

Le robot ne se déplace pas dans ce programme!

Même si le nom du fichier ne fait référence qu'à l'ACS, ce programme illustre l'utilisa-

tion de l'ACS *et* des pare-chocs avec des 'event handlers' appropriés. Il représente l'état des deux canaux ACS par les LED et le sort par l'interface série. L'état des pare-chocs est uniquement sorti par l'interface série.

Bougez juste votre main devant le robot et appuyez sur les pare-chocs!

Vous pouvez utiliser le programme pour tester différents objets avec l'ACS pour savoir lesquels sont mieux détectés que d'autres. Vous pouvez également modifier la force de transmission de l'ACS. Elle est réglée d'usine sur le niveau moyen.

Exemple 6: Le robot roule en cercle
Répertoire: <RP6Examples>\RP6BaseExamples\Example_05_Move_01
Fichier: RP6Base_Move_01.c

ATTENTION! Le robot se déplace dans ce programme! Vous devez donc déconnecter le robot du PC après le chargement du programme et poser le robot sur une surface suffisamment grande qui devrait faire au minimum 1m x 1m mais de préférence 2m x 2m.

Enfin le robot peut démarrer ses moteurs: Ce programme fait tourner le robot en rond en réglant les deux moteurs sur des vitesses différentes. Veillez à ce que le robot dispose toujours de suffisamment d'espace pour tous les programmes dans lesquels il se déplace. Pour certains programmes, il faut prévoir au moins 1 ou 2m².

Si le robot heurte un obstacle avec ses pare-chocs pendant qu'il roule en cercles, les moteurs s'arrêtent et deux des LED commencent à clignoter. Le robot ne bougera plus jusqu'au redémarrage du programme.

Exemple 7: Le robot va et vient – avec une rotation à 180°
Répertoire: <RP6Examples>\RP6BaseExamples\Example_05_Move_02
Fichier: RP6Base_Move_02.c

ATTENTION: Le robot se déplace dans ce programme!

Dans le chapitre sur la RP6Library, nous avons étudié un exemple dans lequel le robot s'est déplacé commandé par le temps. Dans cet exemple, le robot avance sur une certaine distance, tourne à 180°, revient sur la même distance, tourne à nouveau à 180° et recommence.

Ce programme fait appel au mode de blocage des fonctions move et rotate. C'est pourquoi il n'est pas nécessaire d'appeler constamment la fonction task_RP6System parce que ces fonctions le font déjà automatiquement.

Exemple 8: Le robot roule au carré
Répertoire: <RP6Examples>\RP6BaseExamples\Example_05_Move_03
Fichier: RP6Base_Move_03.c

ATTENTION: Le robot se déplace dans ce programme!

Le robot essaie maintenant d'effectuer des carrés de 30cm de côté. Il fait un demi-tour complet après chaque carré et roule dans le sens opposé.

Cela ne peut se faire qu'avec un bon calibrage de la résolution des encodeurs car sinon le robot ne peut pas tourner exactement à un angle de 90° mais p.ex. à 98° ou 80°. Cet exemple illustre parfaitement que même avec des encodeurs parfaitement calibrés, il n'est pas simple de commander le trajet d'un robot à entraînement par chenilles à l'aide des mesures des encodeurs – comme nous l'avons déjà dit au chapitre 2. Selon la nature du sol, les chaînes dérapent facilement, notamment pendant des rotations, ou s'emballent ce qui fait que le trajet réellement parcouru est nettement inférieur au trajet mesuré par les encodeurs. Une programmation habile peut certes compenser partiellement cet écart mais ce ne sera probablement jamais à 100% précis. Cette précision n'est réalisable qu'avec des détecteurs supplémentaires qui déterminent plus exactement la position et l'orientation du robot (voir Annexe).

Exemple 9: Excursion – Automates finis
Répertoire: <RP6Examples>\RP6BaseExamples\Example_05_Move_04_FSM
Fichier: RP6Base_Move_04_FSM.c

Le programme génère des sorties sur l'interface série

Dans ce programme, le robot ne se déplace *pas*!

Pour les programmes complexes, le méthodes simples que nous avons utilisées dans les programmes précédents, ne suffisent pas toujours.

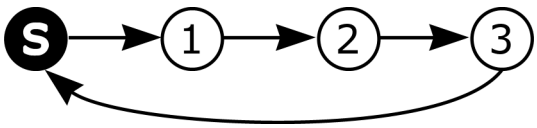
Ainsi, pour des robots commandés par le comportement, nous avons besoin d'automates finis (anglais Finite State Machines ou en abrégé FMS). Ce programme montre un automate fini simple qui change son état lorsque les pare-chocs sont appuyés. Afin de bien comprendre le programme, le mieux est de l'exécuter et appuyer deux fois sur les pare-chocs. Il faut regarder les sorties sur le terminal et les LED d'état tout en appuyant d'abord lentement sur les pare-chocs et relâcher!

Dans la plupart des cas, des automates finis sont réalisés en C avec une construction switch-case (ou une multitude de conditions if-else-if-else... switch est cependant plus clair).

Un petit exemple:

```
#include "RP6RobotBaseLib.h"

#define STATE_START  0
#define STATE_1      1
#define STATE_2      2
#define STATE_3      3

uint8_t state = STATE_START;

void simple_stateMachine(void)
{
   switch(state)
   {
      case STATE_START: writeString("\nSTART\n");     state = STATE_1;
        break;
      case STATE_1:     writeString("Etat 1\n");    state = STATE_2;
        break;
      case STATE_2:     writeString("Etat 2\n");    state = STATE_3;
        break;
      case STATE_3:     writeString("Etat 3\n");    state = STATE_START;
        break;
   }
}

int main(void)
{
   initRobotBase();
   while(true)
   {
       simple_stateMachine();
       mSleep(500);
   }
   return 0;
}
```

Il ne s'agit que d'un exemple très, très simple pour illustrer le principe. Un automate est composé d'états et de transitions entre ces états. Nos états sont ici STATE_START et STATE_1 à 3. Sous forme de graphique, l'automate du listing ci-dessus se présenterait ainsi:



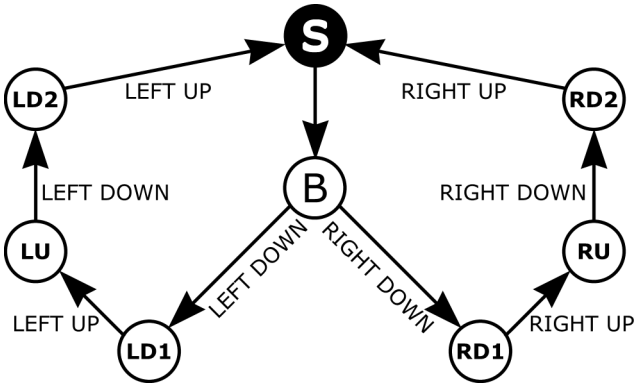
S est l'état de départ. Comme nous n'avons pas intégré de condition pour le changement vers un autre état, l'automate passe à chaque étape à l'état suivant et revient à l'état de départ après l'état 3. Entre les étapes, nous avons intégré dans ce programme très simple des pauses de 500ms afin que les sorties ne se fassent pas trop vite.
La sortie du programme serait la suivante:

```
DEPART
Etat 1
Etat 2
Etat 3
```

```
DEPART
Etat 1
Etat 2
...
... etc.
```

Ce serait la sortie sans fin du programme.

L'exemple de programme dans le fichier **RP6Base_Move_04_FSM.c** possède toutefois un automate plus complexe avec 8 états. La structure de base est représentée dans le schéma suivant (les états ont bien sûr d'autres désignations dans le programme. Ils ont été abrégés ici pour gagner de la place):



L'automate démarre à l'état S et change immédiatement (après avoir sorti un court message de texte) en état B. Il attend jusqu'à ce que l'un des deux pare-chocs ait été appuyé et commute soit en état LD1, s'il s'agissait du pare-choc gauche, soit en état RD1 s'il s'agissait du pare-choc droit. Il attend de nouveau jusqu'à ce que le pare-choc soit relâché et change en état LU ou RU. Là il attend la deuxième pression sur un des pare-chocs. Dans cet état et même déjà dans le précédent, l'autre bumper est complètement ignoré. Il ne réagit à aucune pression sur l'autre pare-chocs. Si le pare-choc est appuyé une deuxième fois, il commute en l'état RD2 ou LD2. Si le pare-choc est relâché, il commute à l'état de départ et tout recommence depuis le début.
Bien sûr, dans cet exemple, l'automate sort aussi du texte sur l'interface série à chaque changement d'état et défini les LED d'état mais il n'y avait plus assez de place dans le schéma ci-dessus qui devait simplement illustrer le principe de fonctionnement.

Exemple 10: Automates finis 2ème partie
Répertoire: <RP6Examples>\RP6BaseExamples\Example_05_Move_04_FSM2
Fichier: RP6Base_Move_04_FSM2.c

ATTENTION: Le robot se déplace dans ce programme!

Nous allons nous aventurer maintenant dans un automate où le robot bouge! La fonction du programme est très simple: Tout d'abord, les LED d'état clignotent pour nous signaler: « Quelqu'un peut-il appuyer sur le pare-choc gauche svp? » Si nous le faisons, le RP6 recule d'env. 15cm et fait clignoter la LED d'état 2 ce qui signifie qu'il faut maintenant appuyer sur le pare-chocs droit pour que quelque chose se passe. Si nous appuyons, le RP6 avance les 15cm et le cycle recommence.

La structure de l'automate n'a rien de particulier et ressemble à celle ci-dessus qui a constamment sorti « START Etat1 Etat2... ». Nous avons juste ajouté quelques conditions en plus. Ainsi, dans deux états nous attendons avec la fonction « isMovement-

Complete() » que le robot termine le processus de déplacement. Et une autre petite particularité: Dans deux des états, nous exécutons un fragment de programme simple. Dans ce cas, nous avons utilisé un stopwatch pour commuter une LED toutes les 500ms. A la place, nous aurions pu exécuter tout autre chose comme p.ex. le séquenceur de lumière de l'exemple 1.

Nous pourrions remplir des pages sur des automates et autres, mais ceci n'est qu'un mode d'emploi et non pas un ouvrage spécialisé. Donc, continuons avec les exemples de programmes...

Exemple 11: Robot basé sur le comportement
Répertoire: <RP6Examples>\RP6BaseExamples\Example_05_Move_04
Fichier: RP6Base_Move_04.c

ATTENTION: Le robot se déplace dans ce programme!

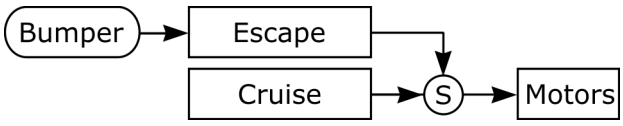
Le but des exemples sur les automates se trouve dans ce programme.

Il met en œuvre un robot simple basé sur le comportement. Au départ, le robot ne possède que deux comportements pour rester simple. Au cours des programmes suivants, nous allons constituer pas à pas un comportement simple imitant un insecte. A la première étape ici, notre « insecte » ne possède que deux petites antennes qui préviennent s'il y a eu collision ou non.

Les deux comportements auxquels nous allons nous limiter pour commencer, s'appellent « Cruise » et « Escape ». Le comportement Cruise donne tout simplement l'ordre d'aller tout droit et rien d'autre. Il est évidemment possible d'y intégrer d'autres comportements et négocier un virage à un moment bien précis, ralentir ou accélérer selon l'état des accus, etc. Toutefois si nous voulons conserver une certaine modularité, il est recommandé de gérer ces comportements individuellement.

Escape, par contre, est déjà relativement complexe et s'active lorsque des objets rentrent en collision avec les pare-chocs. En fonction du pare-choc qui a été touché, le robot recule de quelques centimètres, tourne légèrement et rend le contrôle à l'autre comportement.

Nous pourrions le schématiser de la manière suivante:



Le comportement Escape supprime les sorties du comportement Cruise via le « suppressor » S et donne ses propres instructions aux moteurs. Escape est donc prioritaire sur l'autre comportement.

Ces deux comportements simples suffisent déjà à faire circuler le robot qui « dé-

couvre » son environnement. Il ne peut se fier qu'à deux détecteurs tactiles qui signalent juste la détection ou non d'un objet. Cela ne permet évidemment pas des comportements très complexes.

Imaginez simplement que vous devez vous repérer dans votre appartement avec seulement la pointe de deux doigts – sans yeux, sans oreilles ni d'autres détecteurs tactiles – juste vos deux doigts que vous ne pouvez que tenir droit devant vous...

Plus nous équipons le robot de détecteurs, plus ses comportements seront complexes! Un insecte typique dispose de bien plus de capteurs tactiles que deux et qui donnent en plus des valeurs analogiques sur la force de la pression. Il est clair que si le robot possédait les yeux « à facettes » d'un insecte type, il agirait d'une manière bien plus « intelligente »...

Nous utilisons ici un procédé qui a été développé par Rodney Brooks
(`http://people.csail.mit.edu/brooks/`) vers 1985, appelée

« Architecture Subsumption ».

L'original en langue anglaise se trouve ici:

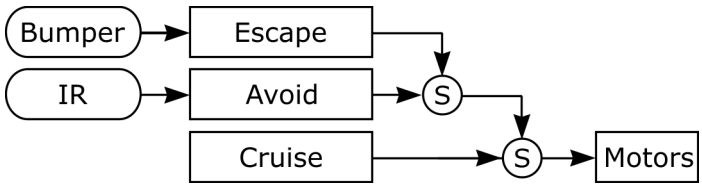
`http://people.csail.mit.edu/brooks/papers/AIM-864.pdf`

En cherchant sur Internet, vous trouverez certainement aussi des documents en français.

Exemple 12: Robot comportemental 2
Répertoire: <RP6Examples>\RP6BaseExamples\Example_05_Move_05
Fichier: RP6Base_Move_05.c

ATTENTION: Le robot se déplace dans ce programme!

Dans cet exemple, notre robot comportemental a été complété par un comportement appelé « Avoid » qui englobe l'ACS afin que le robot ne réagisse non seulement aux collisions mais essaie auparavant de les éviter. Les trois comportements ont donc la structure suivante:



Le comportement Escape possède toujours la plus haute priorité car si le robot heurte un obstacle, cette information est plus importante que l'état de l'ACS. Escape écrase donc les commandes d'Avoid et Avoid fait de même avec les instructions de Cruise.

Si les capteurs IR détectent un obstacle, le comportement Avoid déclenche des manœuvres de contournement. Si le canal ACS gauche détecte un obstacle, le robot tourne à droite et si l'obstacle est détecté par le canal ACS droite, le robot négocie un

virage à gauche. Si les deux canaux détectent un obstacle, le robot tourne sur place à gauche ou à droite en fonction du canal qui a été activé en premier. Le robot continue même à tourner un peu après que les deux canaux aient signalé que la voie est libre. Ceci est commandé par les stopwatches.

Exemple 13: LDRs – Détecteurs de lumière
Répertoire: <RP6Examples>\RP6BaseExamples\Example_06_LightDetection
Fichier: RP6Base_LightDetection.c

Le programme génère des sorties sur l'interface série

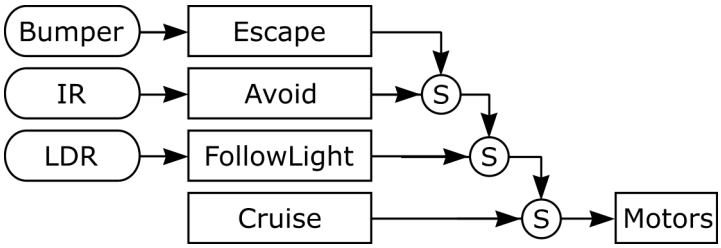
Le robot ne se déplace *pas* dans ce programme!

Cet exemple de programme illustre l'utilisation des détecteurs de lumière. Il montre par les LED d'état quel détecteur de lumière est éclairé plus fortement que l'autre ou si les deux sont éclairés avec la même intensité. Ces informations sont continuellement indiquées avec les valeurs mesurées sur l'interface série.

Exemple 14: Robot comportemental 3
Répertoire: <RP6Examples>\RP6BaseExamples\Example_07_LightFollowing
Fichier: RP6Base_LightFollowing.c

ATTENTION: Le robot se déplace dans ce programme!

Un comportement appelé « FollowLight » permet d'intégrer les détecteurs de lumière:



La priorité de FollowLight se situe en dessous d'Escape et d'Avoid mais au-dessus de Cruise. C'est pourquoi Cruise n'est pratiquement jamais actif dans cet exemple à moins qu'il fasse trop sombre (les deux valeurs LDR inférieures à 100). Dans ce cas, le comportement FollowLight est désactivé.

FollowLight essaie de suivre des sources lumineuses claires à l'aide des LDR ou bien de trouver l'endroit le plus éclairé de la pièce. Cela ne fonctionne pas toujours très bien, notamment lorsqu'il existe plusieurs sources lumineuses. Si vous faites le test dans une pièce sombre avec un projecteur à main puissant, tout devrait bien se passer.

Les LED remplissent une double fonction dans ce programme: Lorsque l'ACS ne détecte pas d'obstacle, elles indiquent quel détecteur de lumière détecte la plus forte intensité lumineuse. Si l'ACS a détecté un obstacle, il est signalé par les LED.

Voilà tous les comportements couverts par les exemples de programmes. Le robot se comporte maintenant comme un insecte simple tel qu'un papillon de nuit qui cherche la lumière tout en évitant les obstacles.

Il est évidemment possible de créer vos propres comportements pour des détecteurs supplémentaires sur des modules d'extension et améliorer largement les comportements existants. C'est à vous de laisser libre cours à votre créativité et vos capacités de programmation!

Exemple 15: Téléguidage au moyen d'une télécommande RC5
Répertoire: <RP6Examples>\RP6BaseExamples\Example_08_TV_REMOTE
Fichier: RP6Base_TV_REMOTE.c

ATTENTION: Le robot se déplace dans ce programme!

Ce programme permet de commander le robot via une télécommande RC5 comme une voiture télécommandée. Seulement le RP6 offre quelques mouvements de plus que la plupart des voitures télécommandées. Il peut bien sûr avancer et reculer mais il peut aussi tourner sur place à gauche ou à droite. Par ailleurs, il peut prendre des virages vers l'avant gauche et droite ainsi que l'arrière gauche et droite, et faire tourner qu'un seul moteur et le faire avancer ou reculer.

Ce programme est constitué de telle façon que les mouvements sont initialisés lors de la réception du code de touche attribué auparavant. Si la touche sur la télécommande est maintenue appuyée, le robot accélère doucement le mouvement. Si vous relâchez la touche, le robot freine (encore plus doucement que lors de l'accélération). Donc, le robot le fait exprès d'accélérer et de freiner si lentement. Vous pouvez également stopper le mouvement subitement à l'aide d'une autre touche. Le robot freine immédiatement et s'arrête à env. un à deux centimètres (s'il n'est pas trop chargé et en fonction de sa vitesse).

La réception RC5 peut être utilisée pour de nombreuses autres choses tels que démarrer des programmes, influencer des comportements, modifier des paramètres de réglage de la régulation de vitesse, etc.

Il serait également possible de commander plusieurs robots avec une seule télécommande universelle par le bais de l'adressage. Les télécommandes universelles possèdent généralement plusieurs touches de sélection pour les différents appareils. Il faut les programmer de telle façon qu'elles émettent toutes en code RC5 et possèdent des adresses différentes dans la mesure où la télécommande en question le permet.

Exemple 16: Interface de Bus I²C – Mode Maître
Répertoire: <RP6Examples>\RP6BaseExamples\Example_I2C_MASTER_01
Fichier: RP6Base_I2C_MASTER_01.c

Ce programme montre l'utilisation possible du mode Maître du bus I²C. Il nous faut bien sûr un Esclave approprié qui est connecté sur le bus I²C. La façon de le piloter varie d'un esclave à l'autre.

Dans cet exemple, nous allons réaliser un petit séquenceur à 8 LED du type « Knight Rider ». Les LED sont branchées sur un PCF8574. Le PCF8574 est une extension de

port courant à 8 bit avec une interface I²C. Vous pouvez braser un PCF8574 sur une des platines d'extension expérimentales (ou le monter sur une platine d'expérimentation pour le tester d'abord) et vous vous retrouvez avec 8 ports I/O libres qui permettent d'évaluer des capteurs numériques et commuter des charges via des transistors externes ou des modules d'entraînement. Les LED et autres appareils à faible consommation peuvent être branchées directement.

Donc, une puce très pratique! Vous pouvez même en utiliser plusieurs. Dans ce cas, vous devez modifier l'adresse de la puce au moyen des trois broches d'adresse ou bien utiliser en plus un PCF8574A s'il y a plus que 8 extensions de port, car il possède un espace d'adressage tout différent. Ainsi vous pouvez utiliser théoriquement 8 ports sur chaque CI, donc 16*8=128 broches de port I/O sur le bus I²C!

```
Exemple 17: Interface de Bus I²C – Mode Maître
Répertoire: <RP6Examples>\RP6BaseExamples\Example_I2C_MASTER_02\
Fichier: RP6Base_I2C_MASTER_02.c

Ce programme montre l'utilisation possible du mode Maître du bus I²C. Il nous
faut bien sûr un Esclave approprié qui est connecté sur le bus I²C.
```

Maintenant s'ajoutent encore des routines pour un PCF8591 à notre programme. Cette puce offre un convertisseur analogique/numérique (CAN) à 8 bit et 4 canaux ainsi qu'un convertisseur numérique/analogique (CNA) qui permet de sortir une tension analogique. Le PCF8574 et le PCF8591 se complètent donc parfaitement. Il est maintenant possible de mesurer quatre tensions. Dans notre exemple, nous supposons que quatre LDR supplémentaires sont connectés sur le PCF8591 (connectés en diviseurs de tension!) mais cela aurait pu être très bien autre chose tels que quatre détecteurs de distance IR GP2D120 de chez Sharp ou bien des capteurs thermiques ou toute autre chose.

Le seul inconvénient de ces deux IC est naturellement que l'évaluation via le bus I²C prend relativement beaucoup de temps. Les deux IC ne sont donc destinés qu'à des tâches relativement simples. Si une évaluation et une commande très rapides sont primordiales, un deuxième microcontrôleur est généralement inévitable. C'est certes plus compliqué à gérer mais beaucoup plus flexible puisque le microcontrôleur est librement programmable.

Les fiches techniques du PCF8574 et du PCF8591 avec les informations exactes se trouvent sur le CD!

```
Exemple 18: Interface du Bus I²C – Mode Maître
Répertoire: <RP6Examples>\RP6BaseExamples\Example_I2C_MASTER_03\
Fichier: RP6Base_I2C_MASTER_03.c

Ce programme montre l'utilisation possible du mode Maître du bus I²C. Il nous
faut bien sûr un Esclave approprié qui est connecté sur le bus I²C.
```

Il s'agit d'un programme de démonstration portant sur la commande d'un détecteur à ultrasons Devantech SRF08 ou SRF10 via le bus I²C. Vous pouvez bien sûr l'adapter à d'autres détecteurs provenant d'autres fabricants.

Exemple 19: Interface du Bus I²C – Mode Maître
Répertoire: <RP6Examples>\RP6BaseExamples\Example_I2C_MASTER_04
Ficher: RP6Base_I2C_MASTER_04.c

Ce programme montre l'utilisation possible du mode Maître du bus I²C. Il nous faut bien sûr un Esclave approprié qui est connecté sur le bus I²C.

Ce programme pilote quatre participants au bus I²C: deux SRF08/SRF10, un PCF8574 et un PCF8591. Nous avons utilisé des parties de programme des trois exemples de programmes précédents.

D'autres exemples de programmes portant sur la périphérie I²C sont publiés avec les modules d'extension.

Exemple 20: Interface du Bus I²C – Mode Esclave
Répertoire: <RP6Examples>\RP6BaseExamples\Example_I2C_SLAVE
Fichier: RP6Base_I2C_SLAVE.c

Ce programme ne fait rien tout seul. Il faut d'abord équiper le robot d'un module d'extension qui prend le contrôle et agit en tant que I²C Maître.

Il est judicieux d'équiper le robot avec d'autres contrôleurs. Généralement, on voudrait utiliser tout de suite un des contrôleurs sur les modules d'extension pour la commande de tout le robot. C'est judicieux parce que le contrôleur sur l'unité de base est déjà très occupé par l'ACS, la régulation des moteurs et tout le reste. Un contrôleur externe dispose de beaucoup plus de temps de calcul qui n'est pas interrompu par d'autres tâches.

Et c'est précisément l'objectif de ce programme: Il peut commander toutes les fonctions importantes du robot par le bus I²C. Le contrôleur est accessible par une adresse réglable en tant que Slave Device. Les caractéristiques bien connues telle que la régulation automatique de vitesse ressort particulièrement bien avec ce programme puisqu'il suffit maintenant d'envoyer juste l'instruction « Roule à <la vitesse> tout droit » via le bus I²C et le reste se fait automatiquement sans s'en occuper. Bien sûr ce procédé peut être interrompu p.ex. lorsqu'un détecteur annonce un obstacle.

Le code du programme contient une liste des commandes valides et des paramètres associés. Vous y trouverez également d'autres informations concernant le pilotage.

Le contrôleur sur la base du robot déclenche, le cas échéant, automatiquement un signal d'interruption sur la première ligne d'interruption (INT1) des connexions d'extension lorsque l'état des détecteurs a changé ou une commande de déplacement a été exécutée.

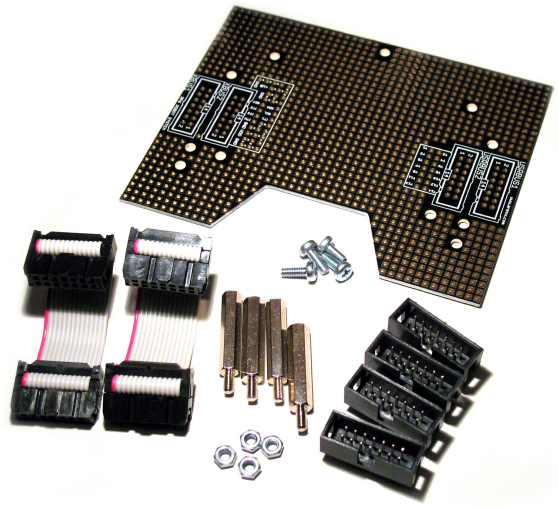
Toutes les valeurs des détecteurs sont lisibles via une série de registres. Donc, lorsqu'une interruption a été déclenchée, vous pouvez d'abord lire quelques registres d'état et vérifier quel détecteur a déclenché l'événement. Ensuite, vous pouvez lire le registre correspondant.

Une autre possibilité consiste à lire tous les registres de détection en une seule fois mais cela prend bien sûr un peu de temps.

Vous trouverez davantage de détails et surtout les désignations exactes des registres dans le code source du programme.

Nous publierons prochainement d'autres exemples de programmes concernant le pilotage de l'esclave en conjonction avec les modules d'extension. Certains se trouvent déjà sur le CD-ROM avec les exemples pour le RP6 CONTROL M32.

5. Platine d'Expérimentation



Le robot est livré avec une platine d'expérimentation qui vous permet de monter vos propres circuits avec des éléments filaires. La platine est livrée en kit. Vous devez d'abord braser les fiches DANS LE BON SENS (voir impression sur la platine) avant de pouvoir l'utiliser.

Le montage de circuit requiert un peu d'expérience de brasage et il faut savoir ce que l'on fait exactement.

Quel genre de circuit pouvons-nous monter sur cette platine?

Nous avons déjà cité deux exemples ci-dessus! D'une part, nous pourrions aménager le robot avec des extensions de port I²C ou des convertisseurs A/N pour l'équiper de ports I/O et de canaux A/N supplémentaires et y brancher des capteurs ou des actionneurs (détecteurs de lumière, détecteurs IR, détecteurs tactiles, LED, etc..). D'autre part, nous pourrions y connecter des modules de capteurs plus complexes munis d'une interface de bus I²C tels que des détecteurs à ultrasons, des modules électroniques de boussole, des capteurs de couple et d'accélération. Des détecteurs de pression atmosphérique, de température et hygrométriques permettraient de construire une station météo mobile.

Vous pouvez visser autant de platines d'extensions sur le robot que vous le souhaitez. Vous n'êtes pas obligé de vous limiter à un seul. Si vous voulez effectuer des tâches plus complexes, vous devez utiliser un microcontrôleur supplémentaire qui existe p.ex. sur le RP6 Control M32 qui dispose en plus de 14 I/O libres dont 6 sont des CAN. Les I/O débouchent sur des connecteurs à 10 contacts. Vous pouvez confectionner vous-même de petits câbles en nappe afin de connecter le module sur la platine d'extension.

Le module d'extension C-Control bientôt disponible s'y prêtera très bien.

La carte-mère du robot offre 6 surfaces d'extension ce qui est très pratique si vous voulez installer des détecteurs le plus bas possible (p.ex. d'autres détecteurs IR). Toutefois, avant de commencer à faire des brasures sur la carte-mère, il vaut mieux commencer sur la plaque d'expérimentation normale qui est plus facile à changer en cas de problème...

6. Le Mot de la Fin

Ici se termine la petite excursion dans le monde de la robotique! Nous espérons que le RP6 vous donne entière satisfaction et que cela continuera!

Les débuts sont toujours difficiles, notamment dans un domaine aussi complexe mais lorsque les premiers obstacles ont été surmontés, la robotique est très intéressante et instructive.

Lorsque vous serez plus à l'aise avec la programmation en C, vous pourrez commencer à réaliser des projets bien plus vastes. Ce manuel contient déjà quelques exemples de tout ce que vous pouvez faire avec un robot. Ces idées ne sont évidemment pas exhaustives!

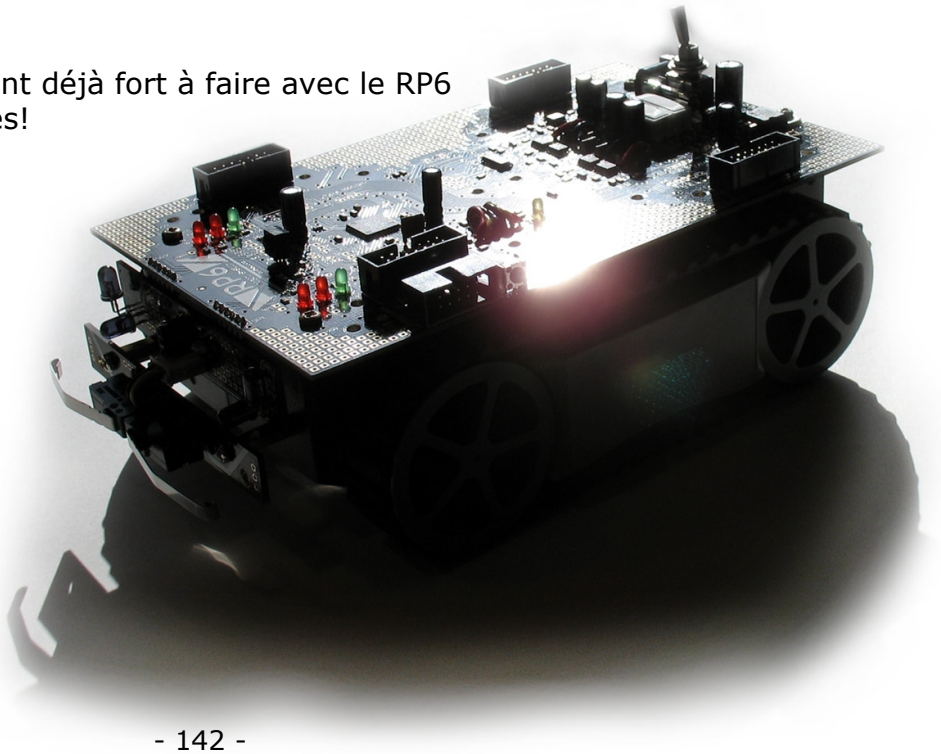
Il existe de nombreux forums de discussion sur Internet entre les utilisateurs de la robotique et de l'électronique qui échangent leurs expériences et s'entraident en cas de problèmes. C'est souvent plus utile que tous les manuels même les plus complets, d'autant plus que nous sommes loin d'avoir abordé tous les sujets et de nombreux sujets n'ont été présentés que très succinctement.

D'autres modules d'extension pour le RP6 sont à l'état de projet. Déjà disponibles sont le module d'extension pour boîtiers DIP ainsi que le RP6 CONTROL M32 offrant un autre contrôleur MEGA32 (+ microphone, transducteurs piézo, mémoire EEPROM externe de 32kb, commutateurs, LED et un port LCD). Un module d'extension pour les modules C-Control plus récents de Conrad Electronic tel que le C-Control PRO MEGA128 est en cours de développement.

Des modules comportant des détecteurs supplémentaires sont également en cours de développement mais nous ne pouvons rien dire de plus pour le moment...

Dès que d'autres modules voient le jour, nous les annoncerons sur la page d'accueil d'AREXX et dans le forum!

D'ici là vous aurez certainement déjà fort à faire avec le RP6 et les modules déjà disponibles!



ANNEXE

A – Diagnostic de Défaillance

Voici une liste d'erreurs qui peuvent se produire avec des propositions de solution et l'explication des causes possibles. Cette liste est constamment complétée et mise à jour.

1. **Le robot ne se met pas sous tension. Aucune des LED ne s'allume et il ne réagit pas au commutateur Start/Stop!**
   - L'interface USB est branchée sur le robot mais pas sur le PC. Dans ce cas, le microcontrôleur maintient le robot en mode Reset qui ne peut commander aucune des LED. Branchez l'interface USB toujours en premier sur le PC avant de la brancher sur le RP6. Le même problème peut survenir lorsque le PC est éteint!

   - Un module d'extension maintient tous les microcontrôleurs en mode Reset. C'est parfois le cas avec une construction personnalisée (même avec des modules finis mais seulement s'ils sont défectueux ou ont été mal programmés). Effectuez un test en retirant tous les modules d'extension!

   - Les accus n'ont pas été montés ou sont vides.

   - Le fusible a sauté ce que vous pouvez vérifier au moyen d'un testeur de continuité (p.ex. un multimètre). Souvent, il suffit de regarder le filament dans le cylindre en verre du fusible qui se coupe généralement au milieu (mais pas toujours. C'est pourquoi il faut vérifier à l'aide d'un multimètre si tout paraît normal).
     ATTENTION: Si le fusible saute, il y a un problème quelque part! Il ne saute qu'en présence d'un court-circuit (avez-vous posé par mégarde des pièces métalliques sur le robot?) ou parce que un ou plusieurs composants ont consommé trop de courant. Regardez de plus près la platine et tous les composants. Avez-vous fait des modifications sur le robot ou voyez-vous des composants défectueux? Est-ce que les accus ont été placés avec la bonne polarité dans le support? N'avez-vous coincé aucun câble entre la carte-mère et le châssis? Avez-vous ajouté des modules d'extension récemment? Si oui, démontez-les tous avant de faire un nouvel essai. Si tout paraît OK et tous les modules d'extension ont été démontés, insérez un fusible rapide approprié de 2,5A et faites un nouvel essai.

2. **Le robot ne démarre pas les programmes!**

   - Avez-vous appuyé sur le commutateur Start/Stop après le téléchargement du programme?

   - Avez-vous correctement chargé un programme dans le RP6? Est-ce le bon programme?

3. **Pendant le trajet, le robot déclenche constamment des Resets et arrête l'exécution du programme!**

   - Les accus ne sont pas suffisamment chargés.
   - Les accus utilisés sont de mauvaise qualité ou déjà très vieux et la tension chute trop violemment pendant l'utilisation ce qui déclenche automatiquement un Reset! Il vaut mieux utiliser des accus neufs pour le robot.
   - Les accus ne sont pas fermement insérés dans le support ou le contact est défaillant pour une autre raison.
   - Un module d'extension ou autre pourrait consommer trop de courant pendant un bref instant!

4. **Le RP6Loader n'arrive pas à établir le contact avec le robot!**

   - Le câble USB ou le câble en nappe à 10 contacts ne sont pas correctement branchés (testez aussi la fiche USB!).
   - Le mauvais port dans le logiciel RP6Loader.
   - Le port est utilisé par une autre application et n'apparaît donc pas dans la liste du RP6Loader. Dans ce cas, quittez toutes les applications qui pourraient utiliser un port USB ou le convertisseur USB/série et mettez la liste des ports à jour dans le RP6Loader ou bien redémarrez le RP6Loader.
   - Dans les réglages du RP6Loader, vous avez sélectionné »Inverser la broche Reset ». Désactivez cette option!
   - Le robot est éteint ou les accus sont vides ou presque.
   - Le câble ou des fiches/prises sont endommagés. C'est peu probable mais peut arriver lorsqu'ils sont trop sollicités.

5. **Le Robot fait des bruits suspects pendant le trajet!**

   - Cela peut avoir des origines diverses. Dans tous les cas, vous devez dévisser le robot et regarder l'engrenage et les moteurs. Est-ce que les câbles des accus se sont coincés dans l'engrenage? **Quelques bruits de cliquetis sont normaux et ne posent aucun problème**. A des vitesses plus élevées, le robot devient aussi plus bruyant (mais ce n'est rien par rapport à son prédécesseur très bruyant, le CCRP5).
   - Est-ce que l'une des bagues de serrage (vis de fixation) sur les roues dentées ou les roues d'entraînement s'est déserrée ou est-ce que autre chose est étrangement mobile? Tout doit pouvoir tourner librement (tournez très doucement les roues d'entraînement!) mais les roues dentées ne doivent pas frotter l'une contre l'autre!

- Une ou deux gouttes d'huile de roulement sur les deux roulements de chaque moteur, donc à l'avant et à l'arrière où on voit l'arbre du moteur, arrivent déjà à réduire sensiblement les bruits (**ATTENTION:** Utilisez uniquement de l'huile spéciale pour roulements!!! Après avoir appliqué quelques gouttes sur les roulements, il faudra tourner le moteurs un peu vers l'avant et l'arrière afin de bien répartir l'huile dans le roulement (tenir le moteur quelques minutes à la verticale de façon à faire renter l'huile dans le roulement). Après que l'huile ait pénétré dans le roulement, essuyez les résidus éventuels à l'extérieur. Les roulements ont été lubrifiés à l'usine mais il faut répéter le processus après un certain temps d'utilisation.

  Surtout, **ne lubrifiez PAS** les roues dentées! Cela endommagera à coup sûr les encodeurs ou les rendra tout du moins inutilisables! Pas forcément l'électronique mais les disques codés/autocollants qui absorbent facilement tout liquide ce qui change complètement les propriétés réfléchissantes. En plus, de l'huile de roulement liquide ainsi que tout autre lubrifiant liquide, risque de gicler sur tout le robot lorsque les roues dentées tournent. **De la graisse lubrifiante (donc non liquide!) ne doit être appliquée que sur les arbres (!) des deux roues à gradins!** De toute manière, la graisse n'apporte généralement pas d'amélioration signifiante...

6. **Dans chaque programme, les moteurs accélèrent très brièvement jusqu'à une très haute vitesse, arrêtent tout de suite et les 4 LED rouges commencent à clignoter!**

- La possibilité la plus simple: Vous avez chargé votre propre programme ou un exemple de programme modifié dans le robot et oublié d'exécuter la commande « `powerON();` » avant de démarrer les moteurs!

- Avez-vous apporté une autre modification au logiciel? Est-ce que d'autres exemples de programmes ou l'auto-test fonctionnent?

- Si ce n'est pas le cas: En plus du clignotement des LED, un message d'erreur devrait sortir sur l'interface série qui décrit le problème plus en détail. C'est souvent un signe d'un problème avec les encodeurs. Les bagues de serrage décrites ci-dessus qui maintiennent normalement en place les deux roues à gradins avec les disques codés, se sont peut-être déserrées? Cela permettrait aux roues dentées de s'éloigner trop des détecteurs qui ne délivrent plus de signal. D'ailleurs, un tout petit potentiomètre (= résistance variable) est placé sur les encodeurs qui permet d'ajuster les détecteurs (utiliser un tournevis à lame plate très fine de 1,8mm ou un tournevis cruciforme approprié! MAIS ATTENTION: Normalement l'ajustement doit se faire au moyen d'un oscilloscope! Sinon cela risque de devenir difficile. Le programme d'auto-test permet de vérifier si cela a bien fonctionné. Ce programme contient un mode spécial (c – Duty Cycle Test) qui permet de mesurer grossièrement la durée de cycle du signal rectangulaire. Le rapport optimal est de 50%, des écarts et variations entre 25 et 75% sont normaux et ne présente pas de problème (quelques messages d'erreurs de temps à autre sont normaux. Cela provient du programme et non pas des encodeurs). Toutefois, la majeure partie des sorties du programme doivent

se situer entre 30 et 70% et indiquer l'état « OK ». Dans ce programme de test, la régulation des moteurs est d'ailleurs désactivée et les moteurs ne sont entraînés qu'avec une valeur MLI fixe. La vitesse mesurée en pas d'encodeurs par 200ms est également indiquée

- Est-ce que les disques codés ont été endommagés? P.ex. par de l'huile ou du lubrifiant comme décrit ci-dessus?

- Est-ce que les câbles vers les encodeurs sont en bon état ou endommagés? Vérifiez avec un multimètre (tenir le testeur de continuité aux extrémités du câble lorsque le robot est éteint!)!

- Est-ce que les petites barrières lumineuses à réflecteur des encodeurs sont sales ?

- Les LED commencent aussi à clignoter si l'un des deux moteurs ne fonctionne pas correctement. Donc, vérifiez les câbles et la platine à proximité de l'électronique du moteur.

7. **Le robot se déplace très lentement ou pas du tout et/ou peu après ou immédiatement après le démarrage du programme, les 4 LED rouges commencent à clignoter!**

- Les LED commencent également à clignoter lorsque la tension des accus est trop faible. Branchez le robot sur le PC et regardez si un avertissement de faible batterie s'affiche. Lorsque vous appuyez sur la touche Reset du robot et le chargeur d'amorçage attend le signal de départ, les quatre LED rouges commencent à clignoter lorsque la tension des accus est trop faible. Le programme arrive à démarrer quand-même mais le robot ne va pas très loin et les LED commencent à clignoter peu après...

- Si vous êtes certain que les accus sont chargés et le RP6Loader indique une tension de plus de 6V dans le champs d'état, un message d'erreur devrait s'afficher sur l'interface série tout comme pour le point précédent sur la liste des défaillances. Il est possible qu'un seul ou les deux engrenages soient bloqués. Ainsi une des bagues de serrage pourrait être trop serré ou un câble s'est glissé entre les roues dentées.

- Dans le pire des cas, les moteurs voire l'électronique de commande sont endommagés. Seul un remplacement des composants concernés pourra y remédier... Faites tourner le programme d'auto-test 8 (le robot NE DOIT PAS toucher le sol pendant ce temps – les deux chenilles doivent pouvoir tourner librement! Ne pas bloquer avec les mains sinon le test échoue immédiatement!). Ce programme teste les moteurs et l'électronique de commande y compris les détecteurs de courant moteur. Y a-t-il des messages d'erreur? (Il est recommandé de sauvegarder le contenu du terminal via le menu « Options --> Enregistrer le Terminal » dans un fichier texte!).

8. **Mon Chargeur ne charge pas les Accus dans le Robot!**

- Avez-vous branché le chargeur en respectant la bonne polarité?

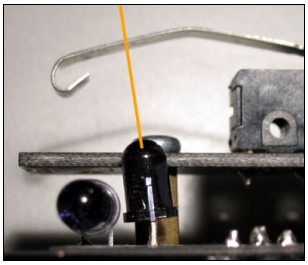
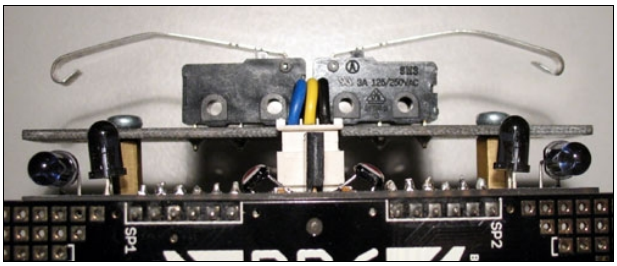
- Est-ce que les accus ont été correctement montés? Pas de faux contact, ni montés à l'envers?

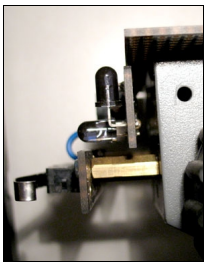
- Est-ce que l'intérieur de la fiche du chargeur est trop grand et n'établit pas le contact avec la broche intérieure de la prise de charge? (Il existe plusieurs versions de ces fiches qui sont normalement librement interchangeables et votre chargeur doit être livré avec différentes versions).

9. **L'ACS a du mal à identifier des obstacles!**

- Avez-vous essayé différents distances/réglages de performance?

- Est-ce que les LED IR ou le récepteur IR sont déformés? Les LED IR devraient monter tout droit de la platine du détecteur et être très légèrement inclinées à 5° ou 10° vers l'extérieur. Le récepteur IR doit pointer tout droit vers le haut (voir photo au chapitre 2 et sur cette page).



- L'ACS a souvent du mal à reconnaître des objets noirs parce que le noir absorbe plus ou moins bien la lumière. C'est donc malheureusement normal! Il faudra vous dépanner avec d'autres détecteurs (Ultrasons...)!

- Des sources lumineuses puissantes se trouvent dans l'environnement du robot p.ex. le soleil, des tubes fluorescents provenant de l'éclairage de la pièce ou l'éclairage de fond d'écrans plats et de téléviseurs. Toutes ces sources lumineuses peuvent perturber le récepteur IR.

10. **L'IRCOMM ne reçoit pas de signaux de ma télécommande! Le robot n'est pas télécommandable!**

- Est-ce que la télécommande émet vraiment au format RC5? Si vous ne le savez pas à coup sûr, il est fort probable que ce n'est pas le cas. Le logiciel ne reconnaît que des signaux RC5. Vous devez régler la télécommande sur un autre code (ce qui n'est généralement possible qu'avec une télécommande universelle) ou bien changer de télécommande.

- L'affectation du code de touches varie d'une télécommande et d'un fabricant à l'autre. Vous devez donc commencer par changer l'affectation des touches dans le programme. Le texte source du exemple de programme RC5 vous explique comment procéder. Regardez les sorties sur le terminal! Les codes

de touches y sont indiqués pour le exemple de programme RC5.

11. **Le robot ne tourne pas exactement à l'angle indiqué.**

- Nous avons déjà expliqué auparavant pourquoi c'était normal. Dans le cas d'un entraînement par chenilles, le glissement des chenilles provoque toujours des écarts. En plus, il faut correctement calibrer les encodeurs avant que cela puisse fonctionner correctement. Reportez-vous également à l'annexe B.

12. **Même des programmes très courts occupent déjà 7 Ko de mémoire! Pourquoi?**

- Comme nous l'avons expliqué auparavant, le RP6Library est toujours incluse. Puisqu'elle occupe déjà à elle seule 6,5 Ko de mémoire, c'est une taille normale pour un programme. Si vous élargissez vos programmes, vous allez constater que la taille du programme n'augmente plus dans les mêmes proportions. Donc, ne vous faites pas de soucis, il y a assez d'espace de mémoire. Et si jamais cela ne devrait pas suffire, vous pouvez simplement ajouter un module d'extension équipé d'un microcontrôleur supplémentaire.

13. **Je ne peux pas compiler mes programmes – un message d'erreurs s'affiche à chaque fois!**

- Quel message d'erreur s'affiche exactement? Si vous voulez vous adresser au Support avec une telle question, vous devez joindre les sorties complètes du compilateur et votre texte source. Voici une petite liste avec les erreurs typiques:

  - Vous avez oublié d'inclure la RP6Library ou les chemins dans le Makefile ne sont pas corrects. Si vous construisez votre propre projet, vous devez éventuellement adapter les chemins dans le Makefile, sinon le compilateur ne trouve pas les fichiers.

  - Avez-vous oublié un point-virgule quelque part dans le programme?

  - Est-ce qu'une parenthèse manque ou est en trop quelque part?

  - Avez-vous fait attention à l'orthographe exacte? A part les commentaires et les signes de formatage habituels tels que des espaces et tabulations, chaque signe en C est important et l'orthographe exacte est primordiale. S'ilya des eRReur dans le mde d'emPLi, ce n'es pas grave, on cprEnd quanMEêm... Mais le compilateur ne comprend plus rien et génère de nombreux messages d'erreur alors qu'il n'y a en fait qu'une seule erreur. Malheureusement, le compilateur ne possède pas de correcteur automatique d'erreurs comme nous, les hommes.

14. **Mes programmes ne fonctionnent pas correctement et le robot ne fait**

pas du tout ce que je veux – où est l'erreur?

- Aucune idée ;-) Vous devez être un peu plus précis avant de contacter le Support! Il y a souvent des questions comme: « Pourquoi mon programme ne fonctionne-t-il pas? » Il faudrait quand-même préciser ce que vous attendez du programme, sinon ce serait un quizz ...

- Des erreurs de débutant fréquentes sont:

  - Un point-virgule mal placé, p.ex. derrière une boucle ou une condition if. C'est vrai que l'on peut y placer souvent un point-virgule – mais ce n'est pas toujours ce qu'on voulait!

  - Les parenthèses dans les constructions if-else sont mal placées. On peut se tromper facilement si les blocs ne sont pas indentés correctement.

  - Vous avez utilisé le mauvais type de donnée pour une variable – un uint8_t ne peut accepter que des valeurs jusqu'à 255 mais ne sait pas compter jusqu'à 1500! Pour cela, il faudra utiliser un uint16_t. De même, un uint8_t ne peut pas comporter de valeurs négatives. Cela n'est possible qu'avec des types de données à signe tel que l'int8_t. Reportez-vous à la liste des types de données au début du cours intensif!

  - Vous avez oublié la boucle sans fin à la fin du programme. Sans cette boucle, le programme risque de vous créer des surprises.

  - Vous utilisez le mode non-bloquant des fonctions move ou rotate mais vous n'appelez pas constamment la fonction task_motionControl ou task_RP6System. Ou bien vous générez de longues pauses avec mSleep dans votre programme. Si vous utilisez le mode non-bloquant des fonctions rotate et move ou si vous utilisez l'ACS ou autre, vous devez effectuer toutes les tâches chronométrées qui exigent des pauses de plus de 10ms avec des stopwatches.! Il ne faut pas effectuer des fonctions mSleep et autres fonctions bloquantes avec les fonctions non-bloquantes. Relisez tranquillement le chapitre concernant la RP6Library et examinez les exemples de programmes!

  - **N'oubliez jamais d'enregistrer les programmes avant de les compiler à nouveau après une modification du texte source! Sinon c'est l'ancienne version, sans les modifications, qui sera traduite. En cas de doute, effectuez MAKE CLEAN et refaites une compilation!**

15. Votre problème ne figure pas dans cette liste?

- Relire des passages importants dans le mode d'emploi apporte souvent la solution mais pas toujours...

- Avez-vous déjà téléchargé les toutes dernières versions du logiciel et la version la plus récente de ce mode d'emploi du site http://www.arexx.com/ ou bien de la page d'accueil RP6 http://www.arexx.com/rp6 ?

- Utilisé la fonction de recherche dans le forum? --> http://www.arexx.com/forum/

- Regardé dans le réseau de la robotique? --> http://www.roboternetz.de/

- Etes-vous déjà allé sur l'un des deux forums? (Ne lancez pas tout de suite un nouveau sujet. Utilisez d'abord la fonction de recherche pour voir s'il existe déjà un sujet concernant votre problème).

B – Calibrage des Encodeurs

La résolution des encodeurs dépend du diamètre réel des roues d'entraînement et des chenilles en caoutchouc. Les chenilles s'enfoncent dans le sol et s'écrasent également un peu elles-mêmes. En plus il y a bien sûr les variations de fabrication. C'est pourquoi il est impossible de calculer le diamètre avec précision. Il faut effectuer des séries de mesure afin de déterminer avec plus d'exactitude la résolution réelle des encodeurs.

La résolution est la distance que le robot parcourt par incrément d'encodeur. Théoriquement ce sont 0,25mm par incrément mais dans la pratique, la résolution réelle se situe plutôt autour de 0,23 à 0,24mm.

Afin de déterminer la résolution, vous devez laisser avancer le robot tout droit sur une distance assez longue (un mètre ou plus) et mesurer avec un mètre la distance qu'il a réellement parcourue. Le robot devra rester relié au PC et indiquer les incréments d'encodeur comptés. Le câble USB et le câble en nappe doivent être tenus lâchement au-dessus du robot. Ne pas tirer sur les câbles ou tenir les câbles trop fermement. Le rebord avant de la platine des pare-chocs doit se trouver au départ exactement sur le début du mètre. Le robot doit être placé avec précision pour qu'il longe le mètre avec précision pendant tout le trajet.

Vous pouvez (comme petit exercice) écrire un programme qui fait avancer le robot exactement sur une distance de 1m - ou 2m ou toute autre distance. Vous pouvez recompiler votre programme après chaque modification de distance et le charger de nouveau dans le robot. Le programme devrait également indiquer à des intervalles réguliers la distance déjà parcourue en incréments d'encodeur.

A une résolution de 0,25mm, un mètre correspond à exactement 4000 incréments. Si le robot ne parcourt que 96,5cm donc 965mm lors du test et compte 4020 incréments, cela correspondrait à une résolution de 0,24mm. Il faut simplement diviser 965mm par 4020. Notez les valeurs mesurées dans un tableau. Répétez le test encore deux fois et notez le valeurs dans le tableau. Ensuite vous inscrivez la moyenne dans le fichier RP6Lib/RP6Base/RP6Config.h sous ENCODER_RESOLUTION (un chemin relatif

du répertoire principal des exemples de programmes – ne pas oublier l'enregistrement!), vous compilez le programme à nouveau et vous le chargez dans le robot. Ensuite vous refaites le test trois fois. La précision devrait être nettement meilleure. Sinon, il faudra enregistrer la nouvelle valeur moyenne dans le fichier. La perfection sera difficile à atteindre – en tout cas pas sans détecteurs supplémentaires.

Les choses se compliquent encore pour la rotation sur place car les chenilles glissent sur place pendant la rotation et le trajet réellement parcouru est plus court que le trajet mesuré. En outre, le sol a une très grande importance puisque les chenilles « glissent différemment » sur de la moquette que p.ex. sur du parquet. C'est pourquoi il faut s'attendre à des imprécisions allant jusqu'à 10°. Sur certains sol, le robot dévie sur le côté. Il faut alors effectuer quelques essais en plus.

Si vous soulevez l'avant du robot au niveau de la platine des pare-chocs pendant une rotation afin qu'il ne tourne plus que sur les roues arrières, vous voyez jusqu'où il devrait aller lors d'une rotation!

De meilleures Méthodes pour la Mesure du Trajet et la Détermination de la Position

Avec les encodeurs seuls, nous n'atteindrons jamais une grande précision. Une plus grande précision de navigation avec un véhicule à chenilles exige d'autres détecteurs.

L'Université du Michigan a p.ex. construit une petite remorque pour l'un de ses gros robots avec entraînement par chenilles que le robot traîne derrière lui (le robot est relié à la remorque par une barre métallique mobile). Cette remorque détermine la position réelle par des encodeurs et des encodeurs à valeur absolue et quelques calculs. Ici vous trouverez plus de précisions:

`http://www-personal.engin.umich.edu/%7Ejohannb/Papers/umbmark.pdf`

à partir de la page 20 (ou bien 24 dans le document PDF)
ou ici `http://www.eecs.umich.edu/~johannb/paper53.pdf`

On pourrait construire quelque chose de similaire pour le RP6 (mais ce ne serait pas si simple) ou bien faire des essais avec le détecteur d'une souris optique que l'on monte à l'avant ou à l'arrière du robot. Ajoutons à cela une boussole électronique qui sera installée un peu plus en hauteur au-dessus du robot (pour qu'il ne soit pas perturbé par les moteurs et le reste de l'électronique) et cela devrait être possible de tourner à des angles relativement précis.

Une autre alternative serait des gyromètres (détecteurs de vitesse de rotation) ...

Cependant tout cela n'a pas encore été testé et ne doit être que le point de départ d'une réflexion sur tout ce qui pourrait encore être amélioré et analysé. Si la précision vous importe peu, vous pouvez également laisser cet aspect de côté. Après tout, le détecteur d'une souris optique restreint la mobilité tout terrain du robot.

Une véritable précision ne pourra être atteinte que si l'on utilise des repères, donc des points dans l'environnement du robot dont la position est connue avec précision et que le robot arrive à identifier. Il pourrait s'agir p.ex. de balises à infrarouge qui envoient un signal à l'aide duquel le robot arrive à déterminer la direction dans laquelle se trouvent ces balises par rapport à sa position actuelle.

Ou bien on installe des détecteur de tracé sur le robot qui lui permettent de reconnaître des repères sur le sol.

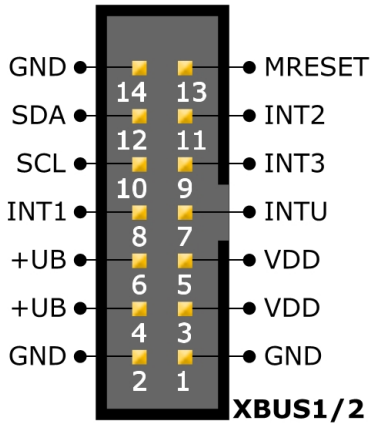
Il existe bien entendu des méthodes bien plus compliquées pour cela. Ici, nous ne présentons que quelques variantes simples qui se passent de calculs compliqués. On pourrait installer p.ex. une caméra au plafond et commander le robot par radio ou infrarouge à partir d'un PC équipé d'un logiciel spécial de traitement d'image. Quelques utilisateurs l'ont fait il y a quelque temps avec le petit ASURO...

Si vous commandez le robot avec une télécommande de téléviseur, vous remarquerez que vous pouvez guider les mouvements avec une précision relativement élevée. Cela s'explique par le fait que l'homme voit immédiatement comment et dans quelle direction le robot se déplace. Malheureusement le robot lui, ne voit rien du tout. On pourrait y remédier avec une caméra au plafond et p.ex. un repère bien visible en couleurs

sur le dessus du robot qui indique aussi l'orientation...

C – Affectation des Broches

Ce chapitre vous informe sur l'affectation des broches des connecteurs et pattes à souder les plus importants sur la carte-mère et vous donne quelques conseils importants sur l'utilisation.



Voici encore une fois pour mémoire l'affectation des connecteurs d'extension avec l'extrait correspondant du chapitre 2:

La broche 1 est toujours posée sur le côté de la carte-mère où se trouve le repérage blanc XBUS1 ou XBUS2 ou est repéré par un « 1 » à côté de la fiche.

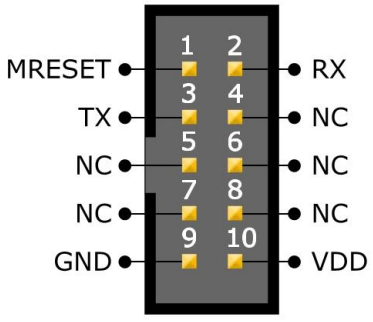
+UB est la tension des accus, VDD les +5V de tension de fonctionnement, GND désigne le négatif (GND = Ground, donc la masse), MRESET est le signal de reset maître, INTx sont les lignes d'interruption, SCL la ligne d'horloge et SDA la ligne de données du bus I²C.

Conseil important: **Ne chargez pas les fils d'alimentations VDD et +UB des connecteurs d'extension au-delà de 1A maximum (s'applique aux deux broches REUNIES! Donc respectivement les broches 4+6 (+UB) et 3+5 (VDD))!**

Si vous avez besoin d'autres choses, vous devez les braser vous-même sur les contacts USRBUS. Les contacts USRBUS sont connectés 1:1 avec les cosses sur la platine, c'est-à-dire pin1 du connecteur est relié à Y1, pin2 à Y2, etc..

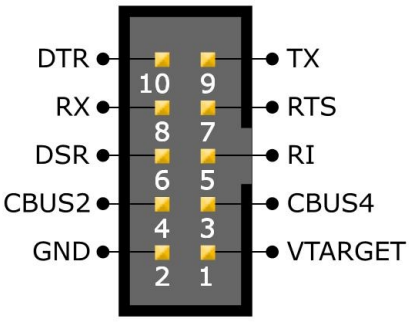
Les deux connecteurs à 10 pattes pour la connexion sur l'interface USB présentent des affectations un peu différentes:

Carte-mère:



RX et TX doivent être inversés pour que la communication puisse fonctionner. La disposition des connecteurs est symétrique pour que le câble en nappe avec soulagement de traction soit orienté correctement.

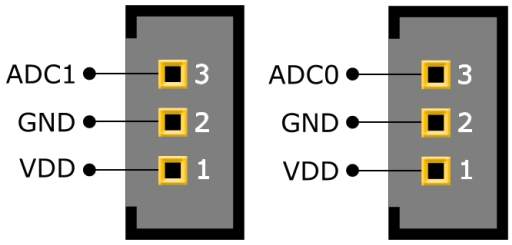
Interface USB:



Si la carte-mère n'est pas connectée sur l'interface USB, vous pouvez utiliser le connecteur sur la carte-mère pour d'autres tâches, p.ex. pour faire commander le robot via UART par un autre microcontrôleur

Connexion CAN:

L'affectation des deux connexions CAN est représentée dans la figure ci-dessous. Ils ne sont pas équipés de fiches. Vous pouvez donc utiliser des fiches à 3 contacts d'un entr'axe de 2,54mm de votre choix et les braser vous-même. Mais soyez prudent et n'endommagez rien sur la carte-mère par la brasure! Il est conseillé de le faire que si vous avez déjà de l'expérience dans la brasure. Donc, dans le doute, il vaut mieux commencer avec une platine d'extension!



Vous pouvez brancher deux détecteurs analogiques ou numériques de votre choix sur les connecteurs (la tension de sortie des détecteur doit se situer entre 0 et 5V) et les alimenter en 5V. Eventuellement il faudrait implanter le gros condensateur électrolytique sur la carte-mère – 220 à 470µF (Pas plus! La stabilité de la tension devrait être d'au moins 16V) conviennent à la plupart des applications.

Cela n'est nécessaire que si vous utilisez des détecteurs présentant un courant crête très élevé tels que les détecteurs de distance IR très populaires de chez Sharp. Des condensateurs de dérivation (100nF) sur la carte-mère ne se prêtent qu'à des longueurs de câbles très courts. Lorsque les chemins sont plus longs, il est préférable de les souder directement sur les détecteurs (ce qui est même vivement conseillé pour des chemins de câble plus courts!).

Toutes les autres connexions sont suffisamment bien identifiées sur la carte-mère. Un coup d'œil sur les schémas techniques que vous trouverez sur le CD, s'avère aussi très utile!

D – Conseils de Recyclage et de Sécurité

Traitement des Déchets

Le RP6 ne doit pas être jeté dans les ordures ménagères! Comme tous les appareils électriques il doit être amené à un point de collecte d'appareils électriques usagés!

Si vous avez des questions à ce sujet, adressez-vous à votre revendeur.

Consignes de Sécurité pour Accumulateurs et Piles

Tenir les accumulateurs et piles hors de la portée des enfants! Ils risquent d'être avalés par un enfant ou un animal domestique. Dans ce cas, consultez immédiatement un médecin.

Des batteries endommagées ou qui présentent des fuites peuvent provoquer des brûlures en cas de contact avec la peau. Dans ce cas, utilisez des gants de protection appropriés. Ne pas court-circuiter ni jeter des batteries au feu. Ne pas charger une pile ordinaire! Risque d'explosion! Uniquement des accumulateurs rechargeables tels que des NiMH peuvent être rechargés à l'aide d'un chargeur approprié.

Consignes de Traitement pour Accumulateurs et Piles

Il est interdit de jeter des accus et des piles dans les ordures ménagères! Vous en tant que consommateur final, êtes légalement tenu de rapporter tous les accus/piles usagés.

Tous les accus/piles hors d'usage doivent être ramené chez votre revendeur ou à un point de collecte spécifique pour les piles que vous trouverez partout où ces produits sont en vente.

Ainsi vous faites votre devoir légal de citoyen et contribuez à la protection de l'environnement.