Programmation sous Orccad pour le Cycab et Installation de Linux sur le Pekee

Romain Lacroix

15/08/2004

Contents

Chapter 1

Présentation de l'environnement à l'INRIA

1.1 L'INRIA Rhône-Alpes

L'INRIA[1], Institut National de Recherche en Informatique et en Automatique, est un établissement public à caractère scientifique et technologique, placé sous la double tutelle du Ministère de la Recherche et du Ministère de l'Economie, des Finances et de l'Industrie. Créée en décembre 1992, l'unité de recherche INRIA Rhône-Alpes regroupe plus de 400 personnes réparties sur trois sites : la Zirst de Meylan-Montbonnot, le campus universitaire de Grenoble et le site technopolitain de Lyon (dont Lyon-Gerland et le domaine scientifique de la Doua). Cf Photo site de Montbonnot. 1.1 J'ai eu la chance d'effectuer mon stage de deuxième année dans le département SED [2], dont la mission est le support aux plates-formes expérimentales.

1.2 SED et la halle robotique

Les principaux secteurs concernés sont

- Les plates-formes Robotique et Vision
- Les plates-formes Réalité Virtuelle
- Les plates-formes Grappes

[1] http://www.inrialpes.fr/presentation.html
[2] Support Expérimentation et Développement

Figure 1.1: Panorama INRIA

Figure 1.2: Cycab

- Le support au développement logiciel

Le support expérimental proposé par l'équipe de SED [3] permet donc de maintenir les installations (matériel et logiciels spécialisés), de développer (mise en place d'expérimentations, logiciels), et d'assurer la recherche (conception de systèmes, participation aux expérimentations). Le but est de favoriser les expérimentations inter-projets, la mise en commun des moyens expérimentaux et d'assurer la réutilisabilité des outils (environnement de développement, matériel de réalité virtuelle..). La participation de SED dans le support aux plates-formes Robotique et Vision se base principalement sur le matériel présent dans la Halle Robotique [4]:

- BiP, le robot marcheur[5]. Le robot Bipède dont la conception mécanique a été réalisée par le Laboratoire de Mécanique des Solides de Poitiers possède 15 degrès de liberté : 6 par jambe (2 à la cheville, 1 au genou, 3 à la hanche) et 3 pour articuler le tronc. Sa conception et son développement ont été réalisés avec l'aide de Orccad.

- Les véhicules électriques Cycab[6]: des véhicules électriques de la taille d'une voiture de golf.

- Le robot Pekee[7]: un petit robot mobile avec 3 roues, une caméra et un pc embarqué de type 486.

Un contrôleur générique pour les robots, nommé Orccad[8], est utilisé pour les plates-formes robotiques. Orccad est un outil d'aide à la conception de contrôleur robotique d2́veloppé par l'INRIA et utilisé pour le robot Bip, le premier prototype du Cycab et le RX90.

Je vais détailler dans la partie suivante les deux robots sur lesquels j'ai été amené à travailler: le Cycab et le Pekee.

[3] http://www.inrialpes.fr/sed/presentation.html
[4] http://www.inrialpes.fr/sed/plateformesRobVis.html
[5] http://www.inrialpes.fr/sed/robvis/bip.html
[6] http://www.inrialpes.fr/sed/robvis/cycab.html
[7] http://www.wanyrobotics.com/robots.html
[8] http://www.inrialpes.fr/sed/Orccad/

1.3 Les robots: Cycab et Pekee

1.3.1 Le Cycab

Il s'agit de 2 véhicules CyCabs (Cf Figure 1.2) pouvant évoluer sur le parking adjacent à la halle robotique à l'arrière du bâtiment de l'UR. Ce parking est équipé d'un système de vidéo-surveillance (2 caméras situées sur le toit de la halle robotique + carte d'acquisition vidéo sur PC Linux connecté au rseau Inria. Tous les véhicules sont aussi sur le réseau via Ethernet HF. Le premier prototype (désormais hors-service) est le CyCab bleu dont la partie contrôle/commande a été concue et réalisée à l'Inria en 97 par un ingénieur expert (L. Lisowski) embauché dans le cadre de l'action Praxitèle et avec l'aide des ingénieurs du service. Il a été construit sur une base Andruet (chassis tubulaire + 4 moteur-roues + un vérin de direction) et son système de contrôle/commande a les caractéristiques principales suivantes :

- carte processeur MVME162 sous VxWorks
- Ethernet HF
- 2 noeuds MC68332
- bus CAN servant de média de communication entre la carte VME et les "noeuds",
- ORCCAD

À partir de ce prototype, deux autres Cycabs (rouge et blanc) ont été achetés chez Robosoft en 2000 suite à un appel d'offre dont le but était de faire un transfert industriel de notre savoir faire. Les caractéristiques principales des véhicules sont :

- un chassis en nid d'abeille
- 4 moteur-roues MBR
- 2 vérins motorisés de direction
- PC embarqué sous Linux RTAI
- 2 noeuds MPC555 avec variateurs de puissance AMC à commande PWM
- bus CAN servant de mdia de communication entre la carte VME et les "noeuds"
- SynDEx

Les véhicules sont équipés de capteurs proprioceptifs (télémètre laser à balayage, cameras linéaire et matricielle, stéréo vision, proximètre à ultrasons). Ces modéles sont bridés pour une vitesse maximale de 17 kilomètres/heure. Avant mon stage, les Cycabs n'étaient pas connectés au logiciel Orccad.

1.3.2 Le Pekee

Le robot Pekee est fabriqué par la société Wany Robotics[9] (Cf Figure 1.3). C'est une plate-forme de développement pour les chercheurs en robotique qui est très modulaire, grâce à l'emploi de cartouches additionnelles qui contiennent un pc embarqué (Cf Figure 1.4).

Il peut ainsi utiliser des outils pour la collecte d'informations comme une caméra pour la détection de collisions, un capteur infra-rouge, un thermomètre, des capteurs lumineux, des compteurs de vitesse, des détecteurs de chocs... La communication entre le PC et les différents composants mécaniques (direction/puissance...) se fait via un bus OPP[10] (Cf Figure 1.5). Les caractéristiques principales du Pekee sont:

[9] http://www.wanyrobotics.com
[10] Open Parallel Platform

Figure 1.3: Pekee 1

Figure 1.4: Pekee Cartridge

Figure 1.5: Pekee Open

- PC et vidéo embarqués
- 802.11 Wireless pour le réseau
- Le micro-contrôleur Mitsubishi M16C.
- Beaucoup de capteurs “built-ins” (temperature, mouvement, chocs, caméras...)
- Solide châssis en plastique
- 2 moteurs électriques avec différentiels pour les roues avec des codeurs incrémentaux et des suspensions intégrées.
- 2 batteries de 12 volts.

Le PC embarqué a les caractéristiques suivantes: built-in Ethernet RJ-45 et USB, connectiques standard clavier/souris/vga, caméra vidéo en couleurs (720x50 pixels)... La connection réseau 802.11 se fait via une seconde carte-fille dediée aux communications pour le streaming des informations et de la video.

Le PC embarqué du robot Pekee est sous windows 98, système d'exploitation peu adapté à la robotique (principalement car il n'est pas orienté vers le développement).

Dans ce contexte, le sujet de mon stage est décrit dans la partie suivante.

1.4 Sujet du stage

Le travail qui m'a été demandé portait sur les contrôleurs logiciels des robots. Il s'est décomposé en deux parties:

- La programmation sous Orccad du Cycab version Robosoft.
- La migration du robot Pekee sous linux.

Chapter 2

Programmation sous Orccad pour le Cycab

Le travail de connection du Cycab sur Orccad m'a demandé une grande période de formation et d'apprentissage. Après la description des outils logiciels, je présenterai le travail que j'ai réalisé.

2.1 L'environnement de dévoloppement Orccad

ORCCAD est un environnement logiciel permettant de concevoir et de mettre en oeuvre le contrôle et la commande d'un système robotique complexe. Il permet également la spécification et la validation des missions à réaliser par ce système. ORCCAD est principalement destiné aux applications temps réel critiques en robotique, dans lesquelles les aspects relevant de l'automatique (les asservissements, les commandes) sont amenés à interagir étroitement avec ceux manipulant des événements discrets. De tels systèmes sont souvent qualifiés d'hybrides. Dans cette classe d'applications, ORCCAD s'adresse particulièrement aux systèmes présentant une forte interaction avec l'environnement, par le biais de nombreux capteurs et actionneurs. Le contrôle/commande de ces systèmes est souvent embarqué, et le caractère critique de l'application apparaît dans le coût extraordinairement élevé attaché à une défaillance: l'impossibilité ou la difficulté d'intervention sur un sous-marin autonome longue portée, sur un engin intervenant après un incident technologique majeur ou sur un vhicule planétaire rendent impératif la minimisation du risque de non réalisation de la mission. A cet effet, ORCCAD offre sûreté de programmation et possibilités de validation par simulation extensive ou vérification formelle.

Moins formellement, ORCCAD permet de concevoir des systèmes robotiques complexes, en définissant des modules de type "Ressource Physique" ou "Module Algorithmique" ou "Module d'Automate". Ces modules sont définis par plusieurs procédures écrites en C qui implémentent l'initialisation et la tâche atomique associée à ce module. Il s'agit ensuite de synchroniser tous ces modules en les reliants avec des bus de communication. L'outil ORCCAD s'occuppe ensuite de générer du code C++ correspondant au comportement global désiré pour le système robotique complet.

Concernant l'interface graphique, le site web officiel de Orccad la définit ainsi : ORCCAD TOOLS V3.1 est opérationnel. Cette version permet de définir des actions élémentaires (Tâche Robot) à travers une interface graphique de schéma-bloc et de les séquencer (Procédure Robot) en utilisant le langage Estérel.

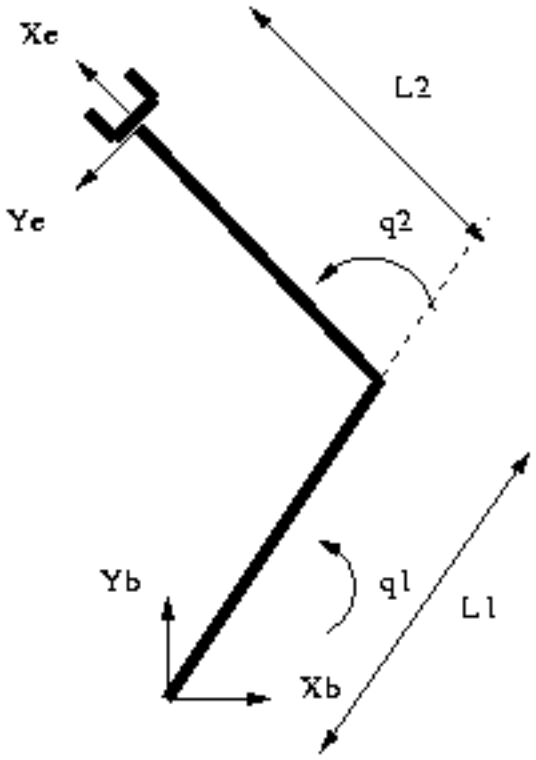


Figure 2.1: ArmX 0

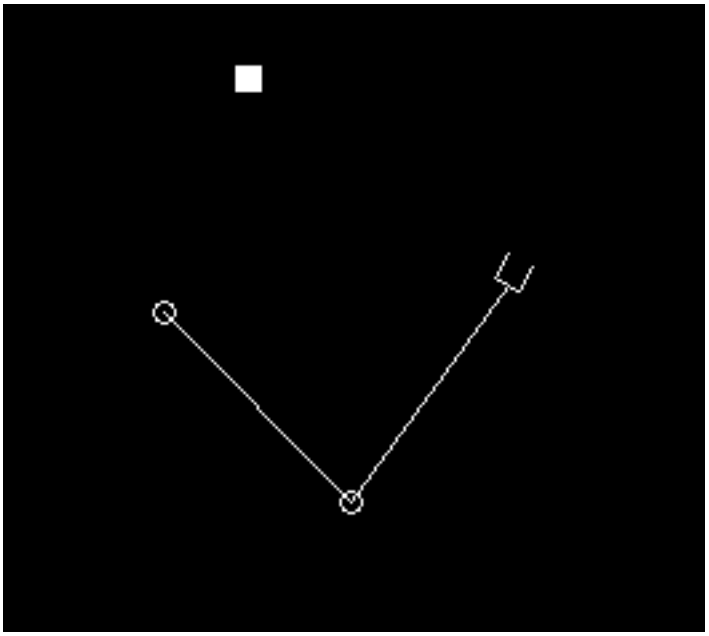


Figure 2.2: ArmX 1

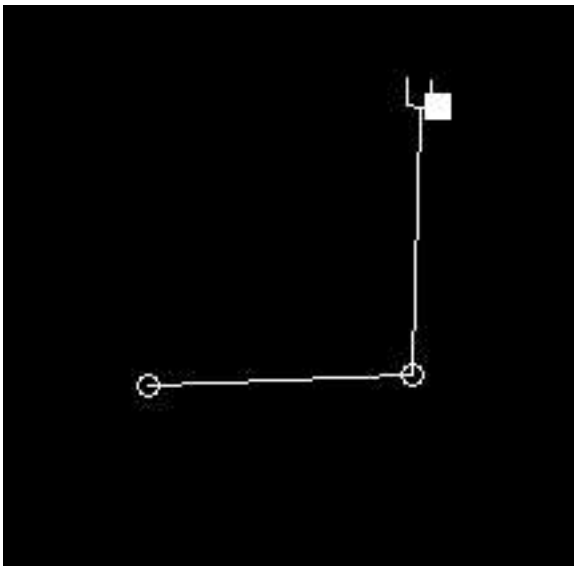


Figure 2.3: ArmX 2

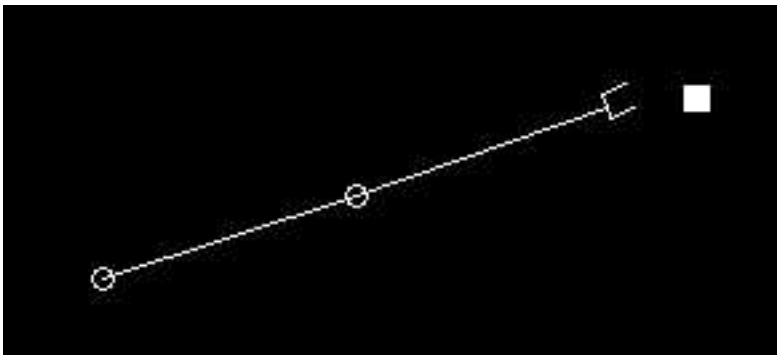


Figure 2.4: ArmX 3

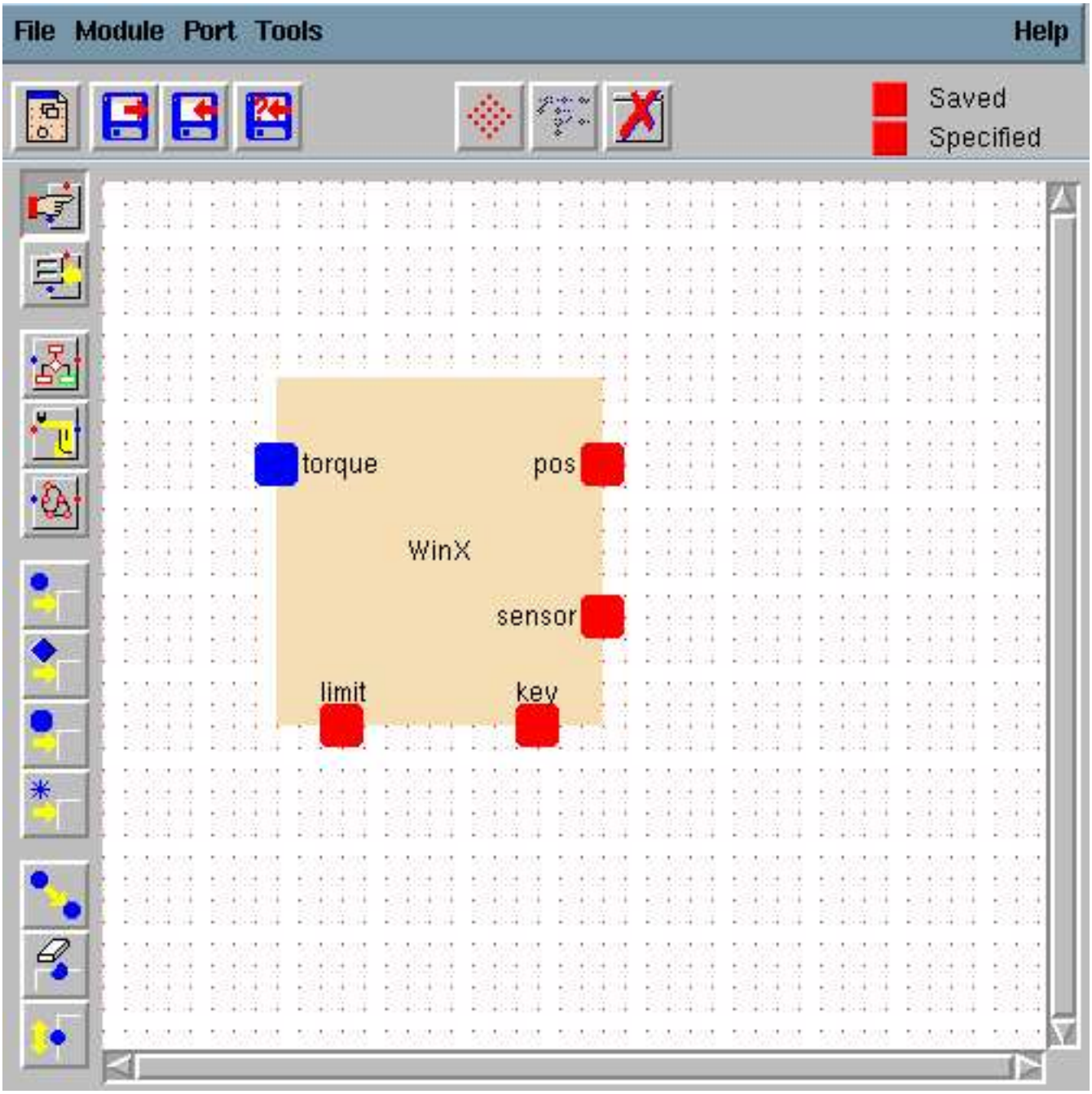


Figure 2.5: Module Ressource Physique de Orccad pour l'exemple du bras

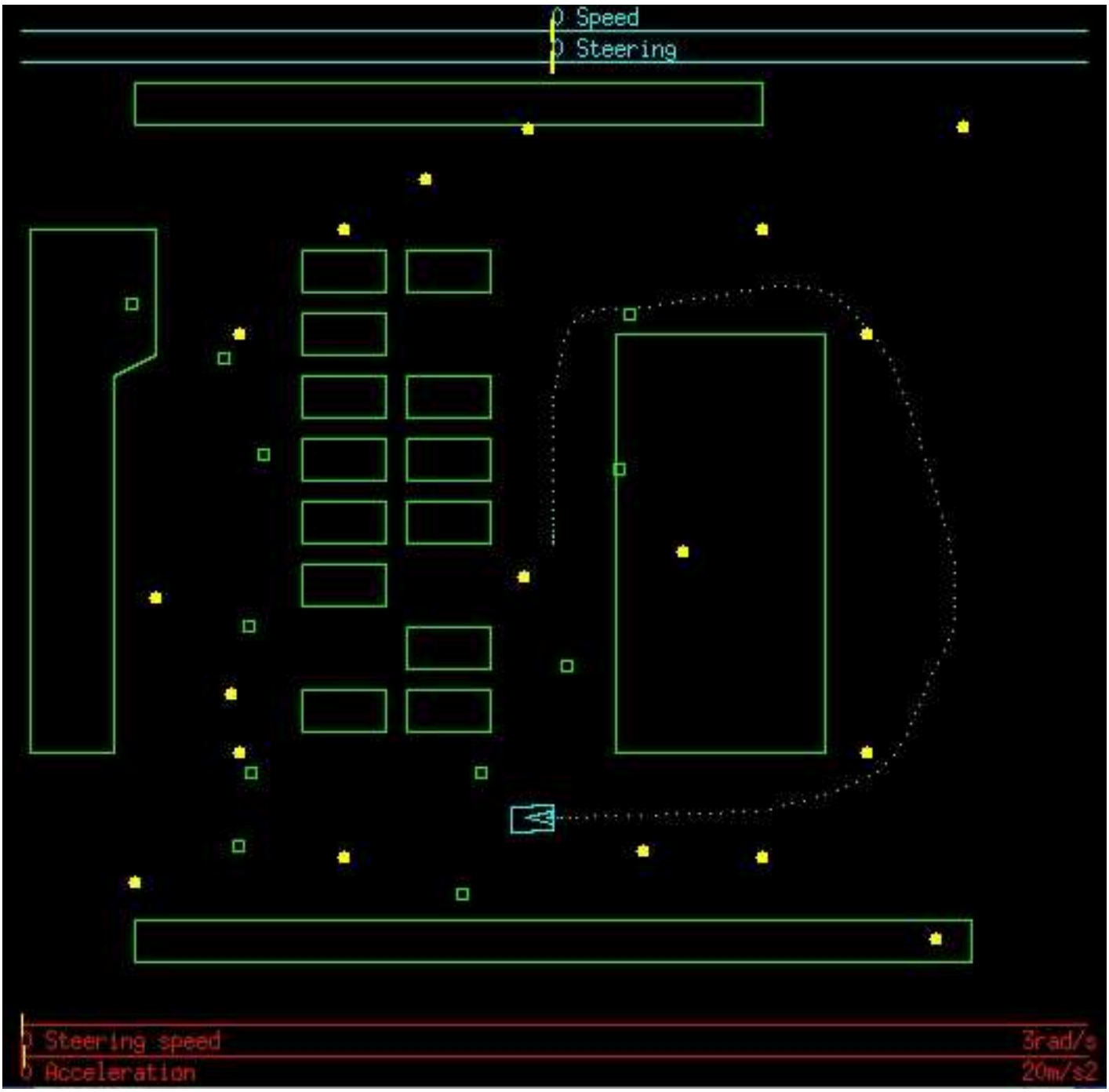


Figure 2.6: Simulateur de Cycab

2.2 L'exemple du bras à 2 degrés de liberté

Le premier tutorial que j'ai suivi pour comprendre ORCCAD propose de simuler un bras articulé à 2 degrés de liberté [1] (Cf Figure 2.1). J'ai passé beaucoup de temps sur ce tutorial, pour prendre en main l'outil de développement complet qu'est ORCCAD. Mon maître de stage Roger Pissard-Gibollet[2] m'a enseigné quelques notions élémentaires de robotique qui me manquaient pour comprendre pleinement le tutorial, et j'ai pu comprendre en expérimentant comment s'utiliser ORCCAD pour modéliser un problème de robotique. Il s'agit principalement de bien cerner l'approche par "Tâche-Robot", ou "Robot Task", construit à partir de plusieurs sortes de modules élémentaires: modules algorithmiques ou modules d'automates ou modules Ressource Physique.

Pour l'exemple du bras à 2 degrés de liberté, le driver correspondant à la ressource physique (l'objet sur lequel les actions ont effectivement lieu) est une fenêtre X11 représentant le bras vu de dessus: Cf Figure 2.2. Ce système robotique tente de résoudre un problème simple: pour une cible dont la position connue (un objet carré dans la fenêtre), le système doit réagir de l'une des deux manières suivantes :

- Soit la position de la cible est atteignable (dans le cercle de portée du bras), alors, le bras va positionner son extrêmité au dessus de la cible (Cf Figure 2.3).
- Soit la position de la cible n'est pas atteignable (hors de portée du bras), alors, le bras va pointer adns la direction de la cible (Cf Figure 2.4).

Cet exemple est implementé grâce à quelques modules dans Orccad: le premier module étant la ressource physique qu'est la fenêtre X11, qui sert pour l'envoi et la réception d'informations avec l'utilisateur. Ce module est caractérisé par des "Driver Ports" comme on le voit sur la figure 2.5 pour échanger des informations avec les autres modules: il a un port d'information en entrée (le port bleu) qui est le torque (le mouvement) que le bras doit effectuer. Les autres ports en rouges sont des ports de sortie d'informations provenant des capteurs tels que la position actuelle ou encore la limite (distance maximale qu'on peut atteindre).

Des procédures plus élaborées sont ensuite formées en associant plusieurs modules élémentaires et en les faisant communiquer avec leurs "Driver Ports": ce sont les "Robot Tasks". On peut ainsi programmer le comportement d'un système robotique complet tout en travaillant dans un environnement de développement global et cohérent.

2.3 Le simulateur de Cycab et la classe de test

J'ai ensuite pu prendre en main le simulateur de cycab, un logiciel développé par Cédric Pradallier [3] dans le cadre de sa thèse MESR. Il est constitué d'une fenêtre donnant une vue aérienne du déplacement de la voiture et d'autres fenêtres regroupant les informations recues par les différents capteurs de la voiture:

- Le temps total
- La position du joystick
- Les vitesses de chaque roue
- Les vitesses différentielles de chaque roue
- Les angles de braquage de chaque roue

Il est possible de simuler la conduite au joystick en l'activant dans une fenêtre de contrôle. Les objets verts représentent des obstacles infranchissable et les points jaunes des piétons. J'ai du passer beaucoup de temps à configurer le simulateur pour qu'il compile et se lance sur ma machine. En particulier, je n'ai

[1] http://www.inrialpes.fr/sed/Orccad/ExempleArmX/frame-eng.html
[2] http://www.inrialpes.fr/sed/people/pissard/welcome.html
[3] cedric.pradalier@inrialpes.fr

pas réussi à lancer ce simulateur dans un debugger afin de mieux l'étudier, bien que les conditions pour le debuggage semblent avoir été respectées. J'ai donc ensuite créé rapidement une petite classe de test afin de récupérer les informations des capteurs et d'envoyer des commandes. Cette simple classe de test m'a permis d'écrire une classe qui pourrait facilement être utilisée avec Orccad pour contrôler l'interface du cycab.

2.4 La classe "interface de la ressource physique"

Mon objectif était ensuite d'interfacer ce simulateur de cycab dans Orccad afin de modéliser ultérieurement le cycab dans Orccad. J'ai donc créé une classe supplémentaire dans la hiérarchie du simulateur afin d'utiliser les primitives de contrôle d'une manière efficace avec Orccad. Le header de cette classe est le suivant :

```
/**
   La classe que va utiliser Orccad pour acceder a la Physical Ressource Cycab.
*/

class CycabOrccadPR {


 protected:

  bool init;

  CycabSimul cycab; // l'objet cycab qu'on utilise

  Record rec; // un record contenant l'etat actuel du cycab

  double loctime; // le temps depuis l'initialisatio

  void update() ; // mise a jour des infos des capteurs

 public:

  CycabOrccadPR() ;

  ~CycabOrccadPR() ;

  // ACCESSEURS

  double getlocTime()   {update();return loctime;}

  Record getrecord()    {update();return rec;}

  int getxn()           {update();return rec.xn;} // axe X du joystick
  int getyn()           {update();return rec.yn;} // axe Y du joystick

  // vitesse differentielle appliquee gauche
  int getleft()         {update();return rec.left;}

  // vitesse differentielle appliquee droite
  int getright()        {update();return rec.right;}
```

```
  // angle de braquage avant ou arriere
  float getphirad(CycabRearFrontEnum RearOrFront)   {
    update();
    return rec.phiRad[RearOrFront];
  }


  // vitesse en metre/seconde pour chaque roue
  float getvmsec(CycabRearFrontEnum RearOrFront, CycabLeftRightEnum
LeftOrRight) {
    update();
    return rec.vmsec[RearOrfront][LeftOrRight];
  }

  // affichage de l'etat courant
  void print();

  // Changement direction et vitesse
  void setvmsecdirection(float * * vmsec, float * phirad);

  // Changement de direction seulement
  void setdirection(float * phirad);


  // Changement de vitesse seulement
  void setvmsec(float * * vmsec);


};
```

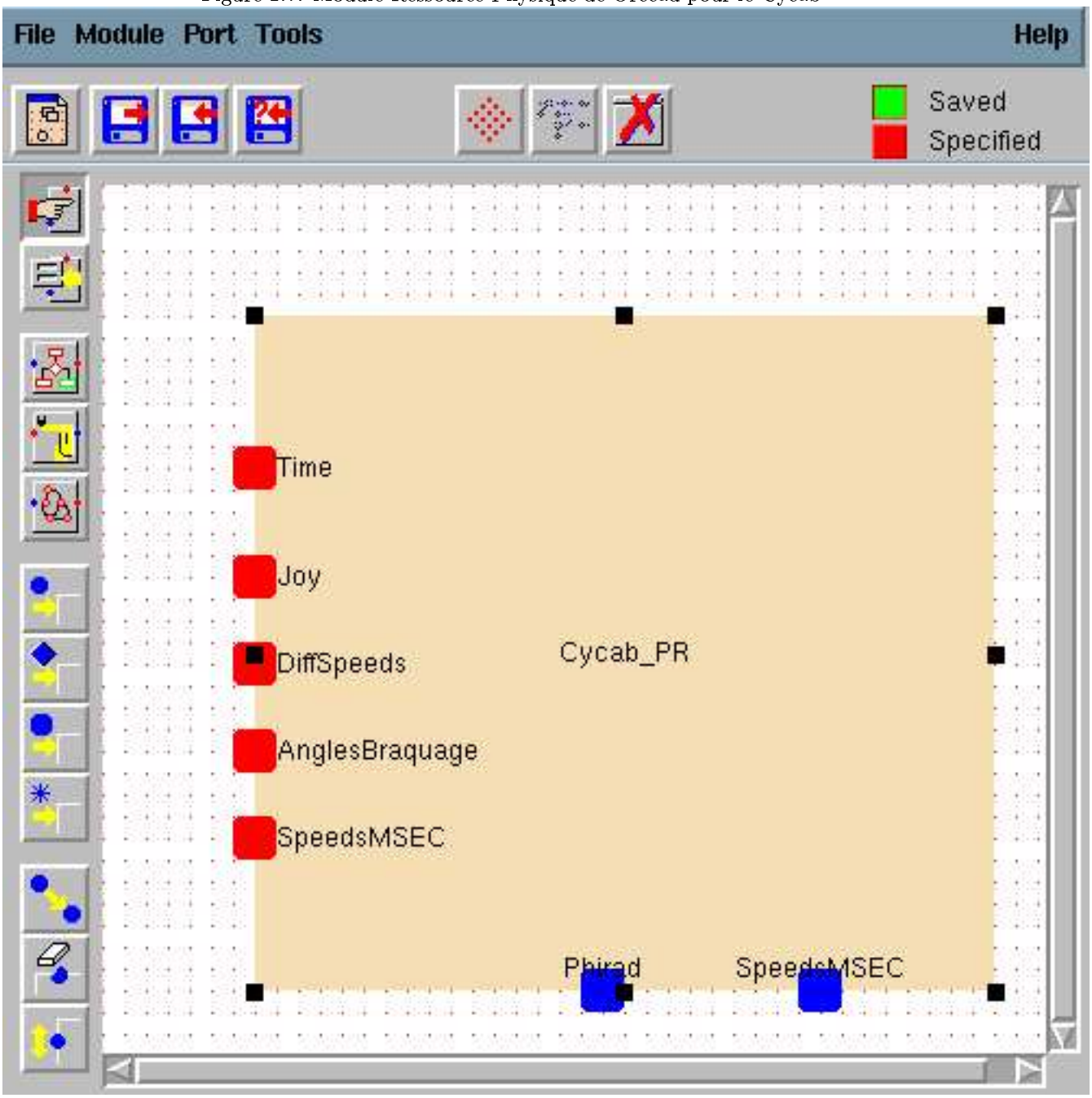
J'ai ensuite pu utiliser cette "interface vers le cycab" dans Orccad. Cette interface peut facilement contrôler soit le simulateur de Cycab, soit le vrai Cycab. Le module Ressource Physique correspond à la figure 2.7.

On remarque qu'il y a 5 "Driver Ports" en sortie, qui regroupent toutes les informations recueillies par les capteurs du cycab, et 2 port en entrée, pour contrôler l'angle des roues (phirad) et la vitesse (SpeedsMSEC) [4].

Après avoir réalisé la connection du Cycab vers Orccad, il m'a été demandé de participer à la migration du robot Pekee sous linux, ce qui me paraissait un objectif très intéressant, notamment grâce à la méthode d'installation que nous allions suivre.

[4] Un Driver Port en entrée ou en sortie peut accepter des informations matricielles, par exemple pour envoyer 4 vitesses indépendantes aux roues en une seule commande

Figure 2.7: Module Ressource Physique de Orccad pour le Cycab



Chapter 3

Installation de linux sur le Pekee

3.1 Le Pekee et la plate-forme de test

J'ai pu passer une partie de mon stage à aider un autre stagiaire sur son projet[1]: il s'agissait d'installer Linux sur un robot de type Pekee, pour ensuite programmer le driver de communication et un programme de contrôle du robot. En effet, le Pekee est à la base concu avec le système windows installé par défaut. Toutefois, comme les ressources matérielles sont assez limitées, il nous a fallu faire l'installation "from scratch", afin d'utiliser le moins d'espace disque possible.

Au début de l'installation, nous ne disposions pas encore de la nouvelle carte PC embarquée, et nous avons expérimenté sur un ordinateur portable dont les caractéristiques techniques étaient proches de la cible finale: il s'agissait d'un Pentium 150 MMX. L'architecture finale, quant à elle est un 486 cadencé à 66 MHZ et disposant de 128 mo de disque dur. Nous avons choisi d'utiliser la suite de logiciels BusyBox[2] pour disposer de beaucoup d'outils de base tout en utilisant peu d'espace disque.

3.2 Compile Toochain, Kernel, Network

Nous avons commencé par compiler la chaîne de compilation nécessaire pour avoir un cross-compilateur pour 486 (notre architecture cible) : gcc, binutil (pour ld et ar...), puis la libc. Grâce à ce compilateur, nous pouvions travailler sur une machine de développement disposant de beaucoup plus de ressources pour la compilation (Pentium 4 1.5GHZ).

Afin d'implémenter le pilote par la suite, nous avions besoin d'un kernel qui gère les timers de facon précise. Il nous a été demandé de ne pas utiliser une des versions 2.6 du kernel linux, c'est pourquoi il nous a fallu utiliser un kernel 2.4.25 avec le patch HRT [3]. Nous avons donc pu compiler le kernel 2.4.25-hrt. Cette étape n'était pas la plus compliquée une fois que la chaîne de compilation était en place.

Les premiers tests consistaient à booter le kernel uniquement sur la machine cible. Nous avons rencontré les problèmes de transferts des fichiers et de configuration du bootloader. Pour ce qui concerne les transferts, nous avons utilisé un outil formidable: Tom's floppy[4]. Il s'agit d'une distribution linux bootable sur disquette, qui avait le support nécessaire pour activer les composants réseaux du portable. Nous pouvions ainsi transférer nos programmes fraichement compilés depuis la machine de développement. Toutefois, cette solution nous obligeait a redémarrer avec la Tom's floppy à chaque nouveau transfert. Plus tard, après avoir rendu notre système bootable, nous avons pu configurer le réseau dans notre installation, en nous aidant des fichiers de configuration trouvés sur la Tom's floppy.

[1] il s'agit d'Amaury Negre, étudiant à l'imag, email : Amaury.Negre@inrialpes.fr, url du stage : http://www.inrialpes.fr/sed/stages/stages2004/stage5.html

[2] http://www.busybox.net/

[3] High Resolution Timer, http://high-res-timers.sourceforge.net/

[4] http://www.toms.net/rb/

Figure 3.1: ls -lh /bin avec BusyBox

```
lrwxrwxrwx  1 negre sed     7 Aug 17 14:39 grep -> busybox*
lrwxrwxrwx  1 negre sed     7 Aug 17 14:39 gunzip -> busybox
lrwxrwxrwx  1 negre sed     7 Aug 17 14:39 gzip -> busybox*
lrwxrwxrwx  1 negre sed     7 Aug 17 14:39 hostname -> busyb
lrwxrwxrwx  1 negre sed     7 Aug 17 14:39 ip -> busybox*
lrwxrwxrwx  1 negre sed     7 Aug 17 14:39 ipaddr -> busybox
lrwxrwxrwx  1 negre sed     7 Aug 17 14:39 kill -> busybox*
lrwxrwxrwx  1 negre sed     7 Aug 17 14:39 ln -> busybox*
lrwxrwxrwx  1 negre sed     7 Aug 17 14:39 login -> busybox*
lrwxrwxrwx  1 negre sed     7 Aug 17 14:39 ls -> busybox*
lrwxrwxrwx  1 negre sed     7 Aug 17 14:39 mkdir -> busybox*
lrwxrwxrwx  1 negre sed     7 Aug 17 14:39 mknod -> busybox*
lrwxrwxrwx  1 negre sed     7 Aug 17 14:39 mktemp -> busybox
lrwxrwxrwx  1 negre sed     7 Aug 17 14:39 more -> busybox*
lrwxrwxrwx  1 negre sed     7 Aug 17 14:39 mount -> busybox*
lrwxrwxrwx  1 negre sed     7 Aug 17 14:39 mv -> busybox*
```

Figure 3.2: Tom's Floppy

Pour la configuration du bootloader, nous avons choisi Grub[5], pour sa puissance et sa flexibilité de configuration. L'installation de grub sur disquette est très simple: il suffit de copier sur une disquette, les 2 fichiers stage1 et stage2 fournis par le logiciel, qui correspondent aux deux étapes éxécutées par grub au démarrage. Cette copie se fait avec l'outil dd, afin que la copie se fasse en "raw-mode". Une fois la copie effectuée, l'intérêt de grub est que cette disquette peut booter n'importe quel linux, sans configuration préalable: une fois que l'ordinateur est booté avec cette disquette, grub propose un shell interactif, dans lequel il faut commencer par préciser, grâce à des commandes simples, la partition root que l'on souhaite booter. Après cela, il suffit de désigner le fichier sur la partition correspondant au kernel que l'on souhaite initialiser. Cette sélection de fichier est grandement aidée par la fonction d' "auto-completion" assurée par le shell grub une fois qu'une partition a été correctement sélectionnée[6]. Il est bien sûr possible d'ajouter des options supplémentaires au kernel. Il nous a ainsi été facile de tester plusieurs configurations de kernel préalablement compilés, pour lesquels nous avions pris soin de changer le paramètre EXTRAVERSION dans le Makefile. Après quelques tests, nous avons pu installer grub correctement configuré directement dans le MBR et nous passer de l'étape de selection du kernel à la main.

[5] GRand Unified Bootloader http://www.gnu.org/software/grub/

[6] grub assure aussi l'auto-completion au moment de la sélection de la partition. Toutefois, son système de nommage des partitions est très différent du classique /dev/hd?? ou sd??.

3.3 BusyBox

Figure 3.3: Busy Box Logo

`The Swiss Army Knife of Embedded Linux`

Il s'agit de la plupart des programmes des packages fileutils et shellutils pour les distributions classiques, regroupés dans un seul exécutable. Cette suite de logiciels est très adaptée aux solutions embarquées dans la mesure où elle est construite en pensant aux problèmes de ressources limitées que nous avons rencontrés, et qu'elle reste très modulaire: nous avons pu choisir un kernel avec des patchs spécifiques pour cohabiter avec BusyBox, ainsi que d'autres logiciels annexes.

Afin d'économiser de l'espace disque, BusyBox n'est constitué que d'un seul binaire: un "Multicall Binary"[7]. Ainsi, tous les programmes peuvent partager du code et donc économiser de l'espace disque. Les différentes commandes utilisables sont des liens dans le répertoire /bin vers le binaire busybox: Cf Figure 3.1.

La configuration de la suite BusyBox se fait via une interface ncurses similaire à l'interface de configuration du kernel menuconfig. Nous avons donc pu sélectionner uniquement les outils nécessaires. Nous avons choisi de ne pas configurer/compiler le kernel "a la busybox" (en utilisant les fonctions intégrées à busybox pour installer le kernel), mais de compiler le kernel par nous même, car la procédure integrée à BusyBox semblait hasardeuse.

3.4 User Space Software

Une fois que le kernel et busybox étaient configurés et fonctionnels, il nous restait encore quelques logiciels à installer afin de faciliter le développement: un serveur ftp pour mettre à jour nos fichiers et un serveur telnet pour tester efficacement nos changements. Malheureusement, ces installations n'ont pas été faciles. Finalement, nous avons pu faire fonctionner ProFTPD[8] et Gnu Telnetd[9].

Plusieurs des problèmes que nous avons rencontrés étaient dûs au fait que nous devions nous occuper de toute l'initialisation du système par nous même: il a d'abord fallu cross-compiler puis installer les outils user-space PCMCIA Card Manager Service, afin que la carte réseau soit prise en compte. Puis il a fallu cross-compiler et installer le service ifup, pour le montage automatique des interfaces réseaux. Ensuite, il a fallu monter /dev/pts, afin de disposer de suffisament de TTY virtuels pour les serveurs telnet et ftp. Finalement, nous avons du explicitement spécifier à proftpd la méthode d'authentification des clients ("à la unix": avec le fichier /etc/passwd) pour que l'authentification des clients FTP fonctionne correctement.

Finalement, nous avions un système capable de lancer les services minimaux nécessaires à la bonne suite du développement:

[7] http://www.busybox.net/downloads/BusyBox.html voir "Usage"
[8] http://www.proftpd.org/
[9] http://www.gnu.org/software/inetutils/inetutils.html

- Kernel 2.4.25-hrt
- Support des composants matériels (Pcmcia, NIC)
- BusyBox : les commandes unix minimales
- Telnetd : pour tester notre code à distance
- ProFTPD : pour uploader nos programmes

Il nous restait à rédiger un script d'initialisation pour configurer tous nos services au démarrage.

3.5 Init scripts

Nous avons choisi de ne pas utiliser le système complet SysVInit, car nos besoins de flèxibilité n'étaient pas très important. Un script rédigé par nous même s'est révelé suffisant pour toutes nos fonctionnalités.

```
#!/bin/sh

#mount file-systems
/bin/mount -a

#start pcmcia service
/etc/rc.d/rc.pcmcia start

sleep 2

#mount all NICs
/sbin/ifup -a

#set hostname
hostname pekee

#french keymap
/sbin/loadkmap < /usr/lib/key/fr.map
```

Nous avons finalement recu la carte PC pour l'installation finale, peu après la fin de nos essais sur l'ordinateur portable. Il a été plutôt facile de finir la transition en copiant notre installation sur le nouveau disque dur. Les premiers transferts se sont même révélés plus faciles qu'avec l'ordinateur portable, car le disque de la nouvelle carte PC est sur carte Flash, il nous a suffi de brancher un lecteur sur la machine de développement pour installer le système. Nous avons ensuite configuré Grub depuis la machine de développement pour rendre la carte flash bootable. Il a fallu reconfigurer la carte réseau, puis nous avons eu la surprise de découvrir que tous les services que nous avions précédemment installé sur la machine de test fonctionnaient correctement sur le Pekee, sans besoin de reconfiguration. L'espace total utilisait par notre installation est d'environ 35 mo. Nous avons pu allouer 28 mo pour le swap. L'espace utilisé par l'installation de windows sur la carte embarquée précédente était d'environ 100 mo.

3.6 Suite du travail, Remerciements, Conclusion

3.6.1 Suite du travail

Mon travail ici aura complété principalement deux objectifs: l'implémentation de la classe "interface de ressource physique du cycab pour Orccad", et l'installation de linux sur le Pekee. Ces deux objectifs auront

été documentés sur le WiKi[10] de l'intranet du projet SED. Ainsi, notre travail pourra être consulté voire repris par la suite, par d'autres membres du projet.

Pour ce qui concerne le travail sur orccad,la suite logique est de continuer l'intégration de la ressource physique, en connectant ses driver ports vers d'autres modules du système robotique global Cycab, afin de modéliser un comportement du système tel que la conduite autonome en évitant les obstacles repérés grâce à la caméra embarquée. Ce comportement nécessite de modéliser les capteurs (avec des drivers ports en sortie) concernant les informations recueillies par la caméra. Il faudra aussi implémenter un module algorithmique concernant la reconnaissance des obstacles, et le calcul de la trajectoire.

Pour le travail sur le Pekee, la suite consiste à écrire un driver commandant le bus OPP de communication entre le PC et le robot, puis un logiciel pour implémenter un comportement similaire d'évitement d'obstacles à celui du cycab. Ce travail est en train d'être effectué par Amaury, et il semble que ses travaux soient déjà bien avancés. Il a pu nous montrer que le comportement attendu était quasiment opérationnel.

3.7 Remerciements

J'aimerais remercier tous ceux qui m'ont permis d'effectuer ce stage formidable dans le cadre du projet SED, il s'agit de toute l'équipe SED, en particulier Gérard Pissard-Gibollet, mon maître de stage, qui m'a accordé sa confiance dès le début de mon travail ici, et dont les conseils ont été précieux pour m'initier au monde de la robotique, ainsi que pour comprendre l'utilité d'outils tels qu'Orccad. J'aimerais aussi remercier Gérard Baille, qui était présent lors de mon premier rendez-vous oú ils m'ont présenté les objectifs, et qui m'a aidé à m'installer le premier matin. Je remercie aussi Sébastien Jardé, pour sa disponibilité lorsque je lui faisais part de mes problèmes techniques relatifs au développement en C++ (il m'a aidé à compiler le simulateur de cycab sur ma machine, et à maîtriser emacs en tant qu'outil de développement logiciel). Mes remerciements vont également à Soraya Arias, qui m'a elle aussi grandement aidé lors de ma tentative de compilation du simulateur de cycab. Le problème auquel j'étais confronté, qu'elle a résolu, me semblait assez obscure: gcc se plaignait à propos de macros indéfinies: INFINITY et NAN, alors que je pouvais vérifier la chaîne des #include et #define, qui me disaient que ces macros étaient correctement mises en place. Il manquait un -D_ISOC99_SOURCE aux flags de compilation pour qu'elle s'effectue sans problèmes sur ma machine. Mes remerciements finaux vont à Hervé Mathieu et Jean-Francois Cuniberto, auxquels j'ai pu régulièrement demander de l'aide sur des sujets moins techniques, et à Claudie Marchand, pour s'être occupée des formularités administratives relatives aux stagiaires.

3.7.1 Conclusion

Finalement, j'aimerais préciser que ce stage a été pour moi une grande expérience, qui complète efficacement mon premier stage effectué dans un laboratoire de recherche à la fin de l'IUT[11]. Je pense que travailler dans un laboratoire de recherche est un formidable moyen de faire avancer l'informatique, et que leurs méthodes de développement sont en accord avec mes convictions. De plus, le matériel mis à notre disposition pour travailler était de grande qualité: machines de développement puissantes, machines de tests variées au niveau des configurations matérielles et logicielles. J'ai pu discuter avec d'autres stagiaires de leurs sujets respectifs et m'impliquer dans un sujet un peu différent de mon sujet initial, ce qui a été une grande expérience. J'ai pu approfondir mes connaissances dans de nombreux domaines, le premier étant la robotique, domaine dont je ne connaissais pas toutes les implications auparavant. J'ai appris beaucoup sur leurs méthodes de développement, ainsi que sur des logiciels intéressants: Orccad bien sûr, mais aussi toute la procédure d'installation de "linux from scratch". Cette méthode d'installation, dont la distribution Gentoo semble la plus proche, est un excellent moyen de comprendre toutes les étapes nécessaires pour avoir un système complètement opérationnel, et nous avons ainsi beaucoup appris sur les problèmes d'installation de tous ces

[10] what is wiki according to wikipedia : http://fr.wikipedia.org/wiki/Wiki

[11] Scripts de déploiement de la plate-forme Diet à l'ENS-Lyon : http://graal.ens-lyon.fr/ diet/ et http://www.ens-lyon.fr/

logiciels, mais aussi de manière plus globale, sur l'interaction de toutes les parties logicielles du système. Finalement, j'ai pu me familiariser un peu plus avec le formidable outil qu'est LaTeXpour rédiger ce rapport.