Quentin QUADRAT
quadra_q, UID 17115, promo 2007

EPITA
École Pour l'Informatique et les Techniques Avancées

RAPPORT DE STAGE

Développement d'un train virtuel de CyCabs avec Scilab/Scicos/SynDEx

1er Janvier, 1er Juillet 2007

Supervisé par

Yves SOREL
INRIA - Domaine de Voluceau - Rocquencourt B.P.105
78153 Le Chesnay Cedex - France
Tél : (1) 39 63 52 60 - email : Yves.Sorel@inria.fr

INRIA
Institut National de Recherche en Informatique et en Automatique

---

Table des matières

Chapitre 1

Introduction

1.1 Contexte et but du stage

Ce rapport présente le travail réalisé dans le cadre du stage de 6 mois de fin d'étude pour l'EPITA, intitulé "Développement d'un train virtuel de CyCabs avec Scilab/Scicos/SynDEx" et s'est déroulé à l'INRIA Rocquencourt sous la direction de Yves Sorel, responsable du projet AOSTE dont l'équipe travaille sur des modèles et des méthodes pour l'analyse et l'optimisation des systèmes temps réel embarqués. L'équipe AOSTE est en relation avec l'équipe IMARA qui travaille sur des projets de route automatisée.

L'équipe AOSTE développe le logiciel de CAO SynDEx qui met en oeuvre la méthodologie AAA pour le prototypage rapide et l'optimisation de la mise en oeuvre d'applications distribuées temps réel embarquées. A partir d'un algorithme et d'une architecture donnés sous forme de graphe, SynDEx génère une implémentation distribuée de l'algorithme en macro-code. La validation de l'algorithme a été faite avec les logiciels de modélisation/simulation Scilab/Scicos, développés par l'équipe SCILAB/METALAU.

Le CyCab est un véhicule électrique commandé manuellement par un joystick. Il sert d'application directe au logiciel SynDEx. C'est ainsi qu'a été généré le code distribué sur les processeurs embarqués du CyCab pour la conduite manuelle. A présent, le but est d'effectuer un train de CyCabs liés électroniquement grâce à une caméra à bas-coût.

Le but de mon stage a été de poursuivre le travail mené sur le suivi automatique de CyCabs. Il faut implémenter un algorithme d'estimation de la distance entre deux véhicules basé sur de la détection de contours dans des images. Il faut déterminer l'asservissement longitudinal du CyCab suiveur en simulation avec le logiciel Scilab. Il faut enfin enfin générer, grâce à la chaîne d'outils Scilab/Scicos/SynDEx le code temps réel, avec ses communications et synchronisations distribuées sur chacun des processeurs embarqués.

Ce rapport se compose de huit parties. Dans la première partie, on présente l'architecture matérielle du CyCab (les deux noeuds à coeur MPC, le PC embarqué, reliés par un bus CAN). Dans la deuxième partie, on présente le système d'exploitation du PC embarqué basé sur un Linux temps réel. Dans les parties trois et quatre, nous rappellerons les éléments d'automatique et de traitement d'images utilisés. Dans les parties cinq et six, nous présentons les logiciels de développement : Scilab/Scicos et SynDEx. Enfin, dans les parties sept et huit nous parlons des logiciels générés par Syndex pour faire de la conduite manuelle puis automatique.

1.2 Travail effectué

J'ai du exploré plusieurs domaines :

- La rétro-ingénierie :
  - En espionnant les états du CyCab,
  - En traduisant l'application de conduite manuelle SynDEx vers Scicos et où j'ai été confronté à des problèmes des erreurs entre les signaux observés et simulés dus à des problèmes d'arrondis.
  - En traduisant des morceaux de code assembleur en langage Scilab/Scicos.
- L'électronique :
  - En dichotomisant les noeuds MPC afin d'isoler le problème des pannes des cartes.
  - En faisant de la maintenance électronique (pannes à détecter, fils à ressouder, etc.)
  - En espionnant la circulation des données temps réel sur un bus CAN,
  - En choisissant et remplaçant les cartes du PC embarqué.
- Le traitement d'images :
  - En récupérant les images d'une caméra FireWire,
  - En lisant des documents sur le traitement de l'image.
  - En appliquant des filtres de traitement d'images simples comme Sobel, Laplace, ...
  - En faisant une IHM pour visualiser/débuger le traitement de l'image servant à l'estimation de distance entre les véhicules.
- L'automatique :
  - En déterminant le régulateur de suivi de véhicules,
  - En appliquant le filtrage de Kalman au suivi de contours.
- La programmation :
  - En découvrant le Linux temps réel RTAI (installation, configuration et l'utilisation),
  - En approfondissant ma connaissance du noyau et des modules Linux,
  - En découvrant la bibliothèque Xlib.
  - En apprenant à me servir des logiciels Scilab, Scicos et SynDEx.

Première partie

Structure d'accueil et contexte

Chapitre 2

Structure d'accueil

2.1 INRIA Rocquencourt

Faisant suite à l'IRIA créé en 1967, l'INRIA est un établissement public à caractère scientifique et technologique (EPST) placé sous la double tutelle du ministre chargé de la Recherche et de l'Industrie.

Les missions qui lui ont été confiées sont :
- entreprendre des recherches fondamentales et appliquées,
- réaliser des systèmes expérimentaux,
- organiser des échanges scientifiques internationaux,
- assurer le transfert et la diffusion des connaissances et du savoir-faire,
- contribuer à la valorisation des résultats de recherches,
- contribuer, notamment par la formation, à des programmes de coopération avec des pays en voie de développement,
- effectuer des expertises scientifiques.

L'objectif est donc d'effectuer une recherche de haut niveau, et d'en transmettre les résultats aux étudiants, au monde économique et aux partenaires scientifiques et industriels.

L'INRIA accueille dans ses 6 unités de recherche situées à Rocquencourt, Rennes, Sophia Antipolis, Grenoble, Nancy et Bordeaux, Lille, Saclay et sur d'autres sites à Paris, Marseille, Lyon et Metz, 3 500 personnes dont 2 700 scientifiques, issus d'organismes partenaires de l'INRIA (CNRS, universités, grandes écoles) qui travaillent dans plus de 120 "projets" (ou équipes) de recherche communs. Un grand nombre de chercheurs de l'INRIA sont également enseignants et leurs étudiants (environ 950 préparent leur thèse dans le cadre des projets de recherche de l'INRIA).

Les chercheurs en mathématiques, automatique et informatique de l'INRIA collaborent dans les cinq thèmes suivants :

1. systèmes communicants,
2. systèmes cognitifs,
3. systèmes symboliques,
4. systèmes numériques,
5. systèmes biologiques.

Un projet de recherche de l'INRIA est une équipe de taille limitée, avec des objectifs scientifiques et une thématique relativement focalisés, et un chef de projet qui a la responsabilité mener et coordonner les travaux de l'équipe. Toutes ces équipes sont très souvent communes avec des établissements partenaires.

Le sujet de ce stage s'inscrit dans les activités du projet AOSTE : Modèles et Méthodes pour l'Analyse et l'Optimisation des Systèmes Temps-Réel Embarqués. Ce projet est bilocalisé à Rocquencourt et Sophia Antipolis. La partie située à Rocquencourt s'intéresse plus particulièrement à l'optimisation des systèmes distribués temps réel embarqués.

2.2 Projet AOSTE Rocquencourt

AOSTE est l'acronyme pour "Modeling Analysis and Optimisation of Systems with real-Time and Embedded constraints".

Les travaux de l'équipe concernent quatre axes de recherche :

- la modélisation de tels systèmes à l'aide de la théorie des graphes et des ordres partiels,
- l'optimisation d'implantation à l'aide :
  - d'algorithmes d'ordonnancement temps réel dans le cas monoprocesseur,
  - d'heuristiques de distribution et ordonnancement temps réel dans le cas multicomposant (réseau de processeurs et de circuits intégrés),
- les techniques de génération automatique de code pour processeur et pour circuit intégré, en vue d'effectuer du co-développement logiciel-matériel,
- la tolérance aux pannes.

Tous ces travaux sont réalisés avec l'objectif de faire le lien entre l'automatique et l'informatique en cherchant à supprimer la rupture entre la phase de spécification/simulation et celle d'implantation temps réel, ceci afin de réduire le cycle de développement des applications distribuées temps réel embarquées.

Ils ont conduit d'une part à une méthodologie appelée AAA (Adéquation Algorithme Architecture) et d'autre part à un logiciel de CAO niveau système pour l'aide à l'implantation de systèmes distribués temps réel embarqués, appelé *SynDEx*.

Chapitre 3

Contexte

3.1 Systèmes réactifs temps réel embarqués

On s'intéresse dans ce document à la programmation de systèmes informatiques pour des applications de commande et de traitement du signal et des images, soumises à des contraintes temps réel et d'embarquabilité [5]. Dans ces applications, le système commande son environnement en produisant, par l'intermédiaire d'actionneurs, une commande qu'il calcule à partir de son état interne et de l'état de l'environnement, acquis par l'intermédiaire de capteurs.

Les systèmes informatiques étant numériques, les signaux d'entrée acquis par les capteurs, ainsi que ceux de sortie produits par les actionneurs, sont discrétisés (échantillonnage-blocage-quantification), aussi bien dans l'espace des valeurs que dans le temps. La précision de la commande dépend de la résolution de cette discrétisation.

Réagir trop tard peut conduire à des conséquences catastrophiques pour le système lui-même ou son environnement. Une analyse mathématique utilisant la théorie de la commande permet de déterminer d'une part une borne supérieure sur le délai qui s'écoule entre deux échantillons (cadence), et d'autre part une borne supérieure sur la durée du calcul (latence) entre une détection de variation d'état de l'environnement (stimulus) et la variation induite de la commande (réaction).

En plus de ces contraintes temps réel, l'application est soumise à des contraintes technologiques d'embarquabilité et de coût, qui incitent à minimiser les ressources matérielles (architecture) nécessaires à sa réalisation (l'architecture peut être composée de plusieurs processeurs, et de circuits spécialisés, pour satisfaire les contraintes temps réel).

Dans ce document, comme nous nous intéressons uniquement aux processeurs, plutôt qu'aux circuits intégrés spécialisés, l'implantation de l'algorithme sur l'architecture consiste donc à traduire (coder) l'algorithme de commande en programmes à charger dans les mémoires des processeurs pour que ceux-ci les exécutent.

3.2 Parallélisation

Pour une architecture monoprocesseur, l'algorithme serait traduit en un seul programme, c'est-à-dire codé en un ensemble d'instructions exécutées séquentiellement par le séquenceur d'instructions du processeur. Pour une architecture multiprocesseur, composée de plusieurs séquenceurs d'instructions opérant en parallèle, ainsi que de médias de communication leur permettant d'échanger des données, il faut partitionner l'ensemble des instructions en fonction du nombre de séquenceurs d'instructions, allouer un séquenceur d'instructions à chacun des sous-ensembles d'instructions et enfin ajouter des instructions de synchronisation et de trans-

fert de données, et leur allouer des médias de communication, pour supporter les dépendances de données entre les instructions de l'algorithme qui sont exécutées par des séquenceurs d'instructions différents.

Un partitionnement simple, par découpage linéaire du programme séquentiel en étapes successives exécutées chacune par un séquenceur d'instructions différent, permet rarement une exploitation efficace du parallélisme disponible de l'architecture, car les dépendances de données entre étapes sont alors souvent telles que les séquenceurs d'instructions passent une partie importante de leur temps à exécuter les instructions de synchronisation ajoutées pour supporter les dépendances de données entre étapes. Pour permettre une exploitation plus efficace du *parallélisme disponible* de l'architecture, il faut étendre l'*ordre total*, d'exécution des instructions du programme séquentiel monoprocesseur, à un emphordre partiel extrait par une analyse de dépendances de données entre instructions, exhibant le emphparallélisme potentiel de l'algorithme, et permettant une distribution (partitionnement ou "allocation spatiale") et un réordonnancement ("allocation temporelle", limitée au respect de l'ordre partiel) individuel des instructions.

La parallélisation est donc un problème d'allocation de ressources, que l'on désire réaliser de manière efficace, où les ressources sont les séquenceurs d'instructions et les médias de communication inter-séquenceurs, et où les tâches à allouer à ces ressources sont les instructions de l'algorithme ainsi que celles de synchronisation et de communication.

3.3 Synchronisation dans les architectures

La synchronisation n'est pas simple entre opérateurs (séquenceurs d'instructions et/ou de communications) car chacun peut séquencer ses opérations indépendamment des autres, sauf dans les deux cas suivants :

1. Lorsque deux opérateurs requièrent simultanément, pour leur micro-opération en cours, un accès à une même ressource (partagée), comme par exemple un bus mémoire, les deux accès doivent être séquentialisés, donc l'un des deux opérateurs doit, avant de commencer son accès, attendre que l'autre opérateur ait terminé le sien, ce qui rallonge d'autant la durée d'exécution de la micro-opération mise en attente ;
2. Lorsqu'une macro-opération consomme en donnée le résultat produit par une autre macro-opération exécutée par un autre opérateur, il faut que ce dernier termine l'exécution de la macro-opération productrice avant que l'autre opérateur ne commence l'exécution de la macro-opération consommatrice, et de plus, comme les algorithmes réactifs sont par nature répétitifs, il faut que la macro-opération consommatrice soit terminée avant que la macro-opération productrice soit à nouveau exécutée lors de l'itération suivante de la séquence répétitive, tout ceci afin que les données ne soient pas modifiées pendant leur utilisation.

Dans les deux cas, des opérations qui auraient pu être exécutées concurremment doivent être exécutées séqueniellement, mais leur ordre d'exécution est sans influence sur le résultat fonctionnel des opérations dans le premier cas, alors qu'il doit être imposé dans le second cas.

C'est la raison pour laquelle dans le premier cas il n'est pas nécessaire de spécifier les synchronisations, d'autant plus qu'elles doivent être faites au niveau de la micro-opération, "invisible" au niveau macroscopique (et même au niveau de l'instruction), et que leurs dates d'occurrence dépendent des durées relatives d'exécution des micro-opérations exécutées concurremment.

Par contre dans le second cas, les synchronisations doivent être spécifiées au niveau macroscopique, insérées entre les macro-opérations.

Ce niveau “opérateur” de décomposition de l’architecture correspond à un grain adéquat pour le problème d’optimisation d’allocation de ressources : chaque opérateur séquence des macro-opérations de calcul et/ou de communication, et de synchronisation.

3.4 Logiciel de modélisation et de simulation Scilab/Scicos

3.4.1 Différents types de logiciels scientifiques

Il existe deux types de programmes scientifiques : – les logiciels algébriques faisant essentiellement du calcul symbolique (Maple, Mathematica, Maxima, Axiom, et MuPad), – les logiciels de calcul scientifique faisant essentiellement de l’analyse numérique (Scilab, MATLAB).

Scilab [7, 2] est un logiciel libre pour le calcul scientifique. Scilab est un interpréteur de langage manipulant des objets typés dynamiquement. Il inclut de nombreuses fonctions spécialisées pour le calcul numérique organisées sous forme de librairies ou de boites à outils qui couvrent des domaines tels que la simulation, l’optimisation, et le traitement du signal et du contrôle.

Une des boites à outils les plus importantes de Scilab est Scicos [7, 3]. Scicos est un éditeur graphique de bloc diagramme permettant de modéliser et de simuler des systèmes dynamiques. Il est particulièrement utilisé pour modéliser des systèmes où des composants temps-continu et temps-discret sont inter-connectés (systèmes hybrides) comme le montre la figure (3.1).

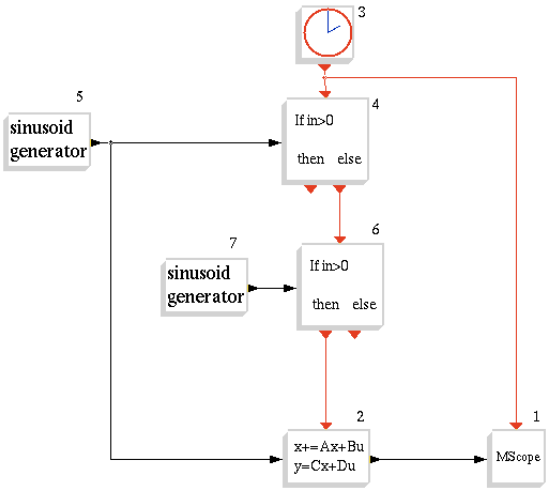


FIG. 3.1 – Un système hybride.

Ce système est hybride car : – les blocs 3, 5 et 7 sont continus et – les blocs 4, 6 et 2 sont discrets.

3.4.2 Mise en place de nouveaux blocs Scicos

Les blocs prédéfinis dans les palettes Scicos permettent de construire des schémas très divers et de créer des systèmes dynamiques hybrides mais dans certains cas on a besoin d’une fonctionnalité que Scicos ne possède pas. Dans notre cas, nous avons besoin d’obtenir des images à partir d’une caméra FireWire. Ces blocs peuvent être définis de plusieurs façons (Scilab, C, Fortran), mais dans tous les cas, Scicos a besoin de deux types de fonctions : – une fonction d’interface, presque toujours écrite en Scilab, pour gérer l’interface avec l’éditeur Scicos, – une fonction de simulation réalisant le comportement dynamique du bloc.

Pour ce stage, j'ai construit de nouveaux blocs permettant d'obtenir des images à partir d'une caméra FireWire et de quelques filtres. J'explique ici comment créer les fonctions d'interface et de simulation, qui servent de patron–exemple pour créer d'autres blocs Scicos.

La fonction d'interface

La fonction d'interface d'un bloc détermine non seulement sa géométrie, sa couleur, le nombre et la taille de ses ports d'entrée-sorties, etc., mais aussi les états initiaux et ses paramètres. Elle gère une fenêtre de dialogue qui permet de ses propriétés, ce que l'utilisateur peut faire en cliquant sur le bloc.

Les fonctions d'interface suivent à peu près toujours le même patron :

```
// Fichier: CAMERA_FIREWIRE.sci

function [x,y,typ]=CAMERA_FIREWIRE(job,arg1,arg2)
x=[];y=[];typ=[]
select job

case 'plot' then
  standard_draw(arg1)
case 'getinputs' then
  [x,y,typ]=standard_inputs(o)
case 'getoutputs' then
  [x,y,typ]=standard_outputs(o)
case 'getorigin' then
  [x,y]=standard_origin(arg1)

case 'set' then
  x=arg1;
  graphics=arg1.graphics;
  exprs=graphics.exprs;
  model=arg1.model;
  while %t do
    [ok,height,width,exprs]=getvalue(..
['Camera FireWire (IEEE 1394)';
 '';
 'Donne une image RGB de taille sous';
         'forme de vecteur de taille 3 x height x width'],..
['Height';
'Width'],..
 list('vec',1,'vec',1),exprs)
    if ~ok then break,end //user cancel modification
    graphics.exprs=exprs;
    if ok then
        model.ipar=[height,width]
        graphics.exprs=exprs;
        x.graphics=graphics;
        model.out = 3 * height * width;
        x.model=model;
        break
```

```
    end
  end

case 'define' then
  height = 240
  width  = 320
  model  = scicos_model()
  model.sim = list('scicos_camerafirewire',4)
  model.evtin = 1;
  model.out = 3 * 240 * 320;
  model.ipar=[height,width]
  model.blocktype='d'
  model.dep_ut=[%f %t]
  exprs=[string(height); string(width)]
  gr_i=['txt=[''Camera'';''FireWire''];';
    'xstringb(orig(1),orig(2),txt,sz(1),sz(2),''fill'')']
  x=standard_define([4 2],model,exprs,gr_i)
end
endfunction
```

La figure (3.2) montre ce que l'on peut obtenir après compilation. Selon la valeur de `job` :

FIG. 3.2 – La fonction d'interface obtenue après compilation.

On a utilisé ici les fonctions standard pour dessiner un bloc rectangulaire avec les cas `plot` (dessiner le bloc), `getinputs`, `getoutputs` (retourner les coordonnées des entrées et des sorties).

Les cas `define` et `set` doivent être adaptés. Le premier définit les valeurs initiales des paramètres et le deuxième gère la fenêtre de dialogue avec l'utilisateur. Les variables `graphics` et `model` sont des structures représentées sous forme de liste. `graphics` contient des informations sur l'aspect du bloc, comme sa taille, son emplacement, ... et `model` des informations nécessaires pour la simulation comme le nom de la fonction de simulation et son type, le nombre et les tailles de ports d'entrées-sorties, les valeurs des états, des paramètres, etc.

Dans le cas `define`, nous créons deux variables `height` et `width` qui définissent la taille par défaut de l'image. `model.out` est le port de sortie qui est un vecteur de taille $3 \times 240 \times 320$. Nous avons aucune entrée `model.in` car nous définissons un capteur. `model.evtin` indique que nous avons qu'une entrée d'horloge. `model.blocktype` indique que nous avons un bloc de type discret. L'élément `[%f %t]` indique que ce bloc ne contient pas de dépendance directe d'entré-sortie, mais qu'il est temps dépendant.

Dans le cas `set`, nous créons une liste de dialogue qui permet à l'utilisateur de modifier les variables `height` et `width`. Les valeurs de ces deux variables seront affichées, ainsi qu'un titre.

Lorsqu'on clique sur le bouton `ok` on change la taille du port de sortie en fonction des nouvelles valeurs de `height` et `width`.

La fonction de simulation

Le patron de la fonction de simulation en langage C est plus simple que pour le patron de la fonction d'interface. En effet elle effectue les tâches suivantes selon la valeur d'un paramètre `flag` :

- *initialisation* : Scicos appelle ce cas une seule fois et au tout début de la simulation pour lui permettre d'initialiser ses états initiaux ou ouvrir un le port de la caméra.
- *terminaison* : Scicos appelle ce cas une seule fois et à la fin de la simulation pour lui permettre par exemple de libèrer de la mémoire ou de fermer le port de la caméra.
- *calcul des sorties* : la fonction calcule ses sorties en fonction des valeurs de ses entrées et de ses états.
- Il existe d'autres cas comme, la mise à jour des états, calcul des dates des événements de sortie, calcul de la dérivée de l'état continu, ...

```
/* Fichier: scicos_camera_firewire.c */

#  include <scicos/scicos_block.h>

void scicos_camera_firewire(scicos_block *block, int flag)
{
  switch (flag)
    {
    case INITIALISATION:
      camera_firewire_open();
      break;
    case TERMINAISON:
      camera_firewire_close();
      break;
    case CALCUL_DES_SORTIES:
      camera_firewire_get_new_image(block);
      break;
    default: break;
    }
}
```

La valeur de `flag` est mise à jour par Scicos. J'ai caché la valeur des identifiants des tâches en utilisant des `define` même si les développeurs de Scicos préfèrent manipuler directement la valeur littérale de `flag`.

```
#define CALCUL_DES_SORTIES      1
#define INITIALISATION          4
#define TERMINAISON             5
```

On fera attention, au bouton `stop` du menu Scicos qui ne termine pas la simulation, mais la met en pause. Par conséquent la fonction `camera_firewire_close` sera appelée que si on clique ensuite sur le bouton `restart`.

Nous n'avons pas encore parler de la structure `scicos_block`. Elle contient toutes les informations utiles du blocs, comme les port d'entrées, sorties, les paramètres, les états :

```
typedef struct {
int nevprt; /* binary coding of activation inputs, -1 if internal ly activated */
voidg funpt; /* pointer: pointer to the computational function */
int type; /* type of interfacing function, current type is 4 */
int scsptr; /* not used for C interfacing functions */
int nz; /* size of the discrete-time state */
double *z; /* vector of size nz: discrete-time state */
int nx; /* size of the continuous-time state */
double *x; /* vector of size nx: continuous-time state */
double *xd; /* vector of size nx: derivative of continuous-time state */
double *res; /* only used for internal ly implicit blocks. vector of size nx */
int nin; /* number of inputs */
int *insz; /* input sizes */
double **inptr; /* table of pointers to inputs */
int nout; /* number of outputs */
int *outsz; /* output sizes */
double **outptr;/* table of pointers to outputs */
int nevout; /* number of activation output ports */
double *evout; /* delay times of output activations */
int nrpar; /* number of real parameters */
double *rpar; /* real parameters of size nrpar */
int nipar; /* number of integer parameters */
int *ipar; /* integer parameters of size nipar */
int ng; /* number of zero-crossing surfaces */
double *g; /* zero-crossing surfaces */
int ztyp; /* boolean, true only if block MAY have zero-crossings */
int *jroot; /* vector of size ng indicating the presence and direction of crossings */
char *label; /* block label */
void **work; /* pointer to workspace if al location done by block */
int nmode; /* number of modes */
int *mode; /* mode vector of size nmode */
} scicos block;
```

Voici un exemple de fonction pour le calcul des sorties :

```
void     camera_firewire_get_new_image(scicos_block *block)
{
  int  i;
  static struct s_device_firewire device;
  static unsigned char rgb[3 * IMAGE_WIDTH * IMAGE_HEIGHT];

  /* block->ipar[0] <==> height */
  /* block->ipar[1] <==> width  */
  camera_firewire_get_frame(&device, rgb);
  for (i = 0; i < 3 * block->ipar[0] * block->ipar[1]; ++i)
    block->outptr[0][i] = rgb[i];
}
```

Compilation des fonctions d'interface et de simulation

Voici un script Scilab qui permet de compiler les fonctions de simulation et de les lier aux fonctions d'interfaces.

```
// Fichier: builder.sce

comp_fun_lst = ['scicos_camera_firewire'];
c_prog_lst   = ['camera_scicos.c'];
prog_list    = strsubst(c_prog_lst, '.c', '.o');
lib_list     = ['libraw1394','libdc1394_control'];

ilib_for_link(comp_fun_lst, prog_list, lib_list, 'c');
genlib('lib_firewire', pwd());
```

La variable `comp_fun_lst` est une liste de chaîne de caractères indiquant les noms des fonctions à utiliser. `c_prog_lst` stocke tous les noms des fichiers C à compiler. `prog_list` stocke les noms des fichiers objets et dont l'extension se termine par o. `lib_list` stocke les noms de librairies dynamiques nécessaires à la compilation. Ces librairies se trouve dans le répertoire `usr/lib/` et doivent être partagées (pour être ouverte lors de l'exécution de la simulation).

Le script suivant permet de lancer la compilation et, si tout va bien, permet de créer une palette nommée `myblock.cosf`. Attention, je n'ai pas bien compris comment créer une palette `scicos_pal` donc il faut utiliser le menu `Open as palette` pour créer une palette valide avec le fichier `myblock.cosf`.

```
exec('builder.sce');
create_palette(pwd());
load lib;
exec loader.sce;
scicos_pal($+1, ["IEEE 1394"; "myblock.cosf']);
```

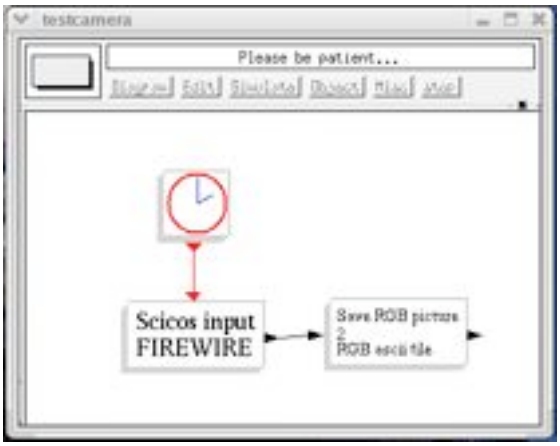


FIG. 3.3 – Les fonctions d'interface et de simulation obtenues pendant une simulation.

Résultat

Si on doit re-modifier la fonction d'interface ou de simulation, il faut faire attention à ce que Scilab intègre bien la dernière version, ce qui n'est pas toujours le cas. Par exemple si on

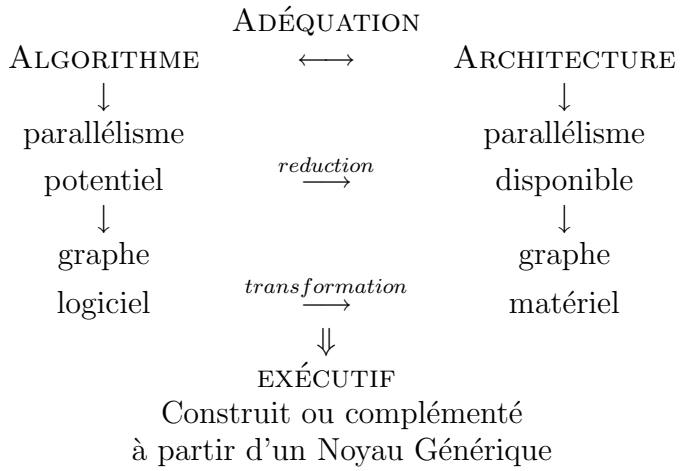
change la fonction d'interface, on doit obligatoirement détruire le bloc de la simulation pour en créer un nouveau, sinon l'ancienne version ne laisse pas la place à la nouvelle.

Les figures (3.2) et (3.3) montrent ce que l'on peut obtenir avec des fonctions d'interface et de simulation.

3.5 Logiciel d'implantation SynDEx

3.5.1 Méthodologie AAA

La méthodologie d'Adéquation Algorithme Architecture est basée sur des modèles de graphes pour spécifier d'une part l'algorithme et d'autre part l'architecture matérielle. La description de l'algorithme permet de mettre en évidence le parallélisme potentiel tandis que celle de l'architecture met en évidence le parallélisme disponible. Cette méthode consiste en fait à décrire l'implantation en terme de transformations de graphes. En effet, le graphe modélisant l'algorithme est transformé jusqu'à ce qu'il corresponde au graphe matériel modélisant l'architecture. L'implantation de l'algorithme sur l'architecture consiste donc à réduire le parallélisme potentiel au parallélisme disponible tout en cherchant à respecter les contraintes temps réel. Toutes ces transformations effectuées avant l'exécution en temps réel de l'application, correspondent à une distribution et à un ordonnancement des différents calculs sur les processeurs et des communications sur les liaisons physiques inter-processeurs. C'est à partir de ces allocations spatiales et temporelles qu'un exécutif va pouvoir être généré et permettre l'exécution de l'algorithme sur l'architecture construite avec des processeurs. Cependant, pour que cette implantation soit vraiment efficace, il est nécessaire de réaliser une adéquation entre l'algorithme et l'architecture. Celle-ci consiste à choisir parmi toutes les transformations proposées celle qui optimise les performances temps réel. Cette méthodologie a été concrétisée dans un logiciel appelé SynDEx.



3.5.2 SynDEx

Comme il a été dit en introduction, SynDEx est un outil de développement pour l'implantation optimisée d'algorithmes respectant des contraintes temps réel sur des architecture distribuées. A partir de graphes flot de données (la description hiérarchique d'opérations est possible) et d'un graphe d'architecture matérielle, des heuristiques sont mises en œuvre afin d'en déduire une distribution et un ordonnancement optimisé des opérations satisfaisant les contraintes. L'adéquation est réalisée en fonction des paramètres des opérations tels que temps estimé de calcul ou un impératif sur le type de ressources (processeurs, media de communication) où l'opération doit être exécutée. L'approche y est formelle et fondée sur la sémantique

des langages synchrones. Le code issu de l'adéquation est un macro-code (m4) qui est ensuite traduit par le macroprocesseur standard M4 utilisant des noyaux d'exécutif (confère section 3.5.4) dépendant des processeur spécifiés sur le graphe d'architecture (figure 3.4).

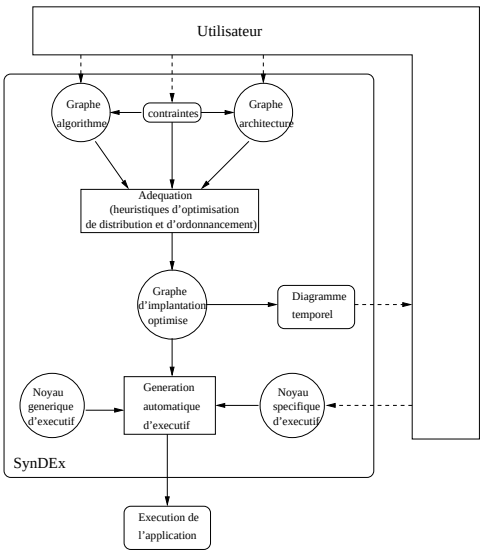


FIG. 3.4 – Principe de SynDEx

Modèles d'algorithmes

Un algorithme est modélisé par un graphe flot de données éventuellement conditionné (il s'agit d'un hypergraphe orienté), qui se compose de sommets et d'arcs. Un sommet est une opération et un arc un flot de données, c'est-à-dire un transfert de données entre deux opérations.

Une opération peut-être soit un calcul, soit une mémoire d'état (retard), soit un conditionnement ou encore une entrée-sortie. Les sommets qui ne possèdent pas de prédécesseur sont des interfaces d'entrée (capteurs recevant les stimuli de l'environnement) et ceux qui ne possèdent pas de successeur représentent des interfaces de sortie (actionneurs produisant les réactions vers l'environnement). Dans le cas d'une opération de calcul, la consommation des entrées précède la production des sorties. La figure (3.5) donne un exemple de la description d'un algorithme via SynDEx.

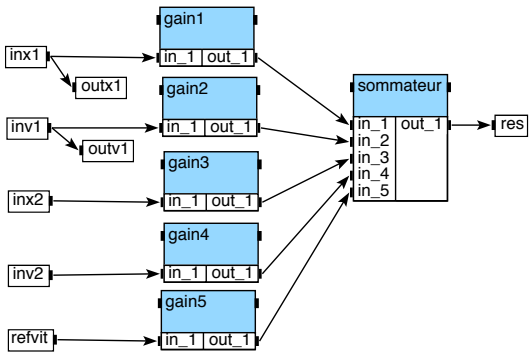


FIG. 3.5 – Un algorithme d'un régulateur simple sous SynDEx.

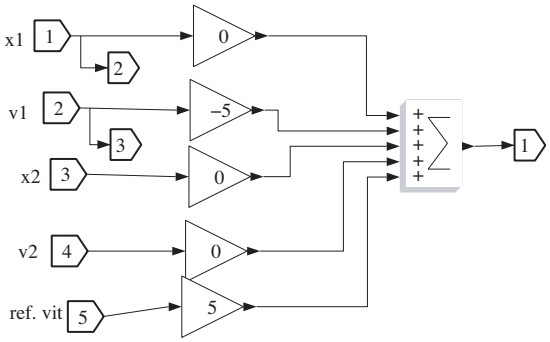


FIG. 3.6 – Le même régulateur mais sous Scicos.

Les algorithmes peuvent être simulés dans un premier temps avec le logiciel Scilab et son éditeur de schéma bloc Scicos [2, 3] puis être convertis vers un algorithme SynDEx grâce à une passerelle [1].

Modèles d'architectures

Une architecture est modélisée par un graphe dont chaque sommet représente un processeur ou un média de communication, et chaque arc représente une connexion entre un processeur et un média de communications (SAM ou RAM). On ne peut connecter directement deux processeurs ou deux médias. Chaque sommet est une machine séquentielle qui séquence soit des opérations de calcul pour les processeurs, soit des opérations de communications pour les médias de communications.

La figure (3.7) montre 5 processeurs de type PC en relation avec un média de communication TCP/IP.

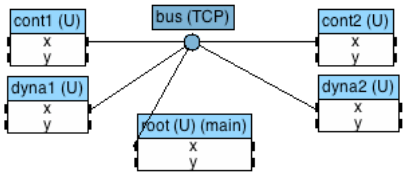


FIG. 3.7 – Graphe d'architecture.

3.5.3 Heuristique de distribution et d'ordonnancement

Une fois les spécifications de l'algorithme et de l'architecture effectuées, il est nécessaire de réaliser l'adéquation. Celle-ci est chargée de respecter d'une part l'ordre des événements vérifiés lors de la spécification de l'algorithme et d'autre part les contraintes temps réel. Pour cela, est choisie parmi toutes les transformations de graphes possibles, celle qui optimise les performances temps réel de l'implantation en terme de latence. La latence ou temps de réponse R est la longueur du chemin critique du graphe logiciel, dont les sommets sont valués par les durées d'exécution des opérations correspondantes y compris celles des communications inter-processeurs.

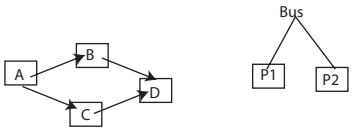


FIG. 3.8 – Exemple simple d'algorithme et d'architecture.

Supposons qu'on ait l'algorithme et l'architecture comme montré sur la figure (3.8) et que l'on veuille trouver une distribution de l'algorithme. On suppose que chaque opération (tâche) et que chaque communication s'exécute en une unité de temps.

A n'ayant pas de prédécesseur, on doit exécuter l'opération A en premier. A l'étape 1, on la place soit sur le processeur P_1, soit sur le processeur P_2 (figure (3.9)). Si on choisit la première solution, on peut placer l'opération B, soit sur P_1, soit sur le P_2 (figure 3.10)). Si on choisit la deuxième solution, il faut ajouter une unité de temps pour la communication. De la même façon, si on choisit la deuxième solution, et que l'on place la tâche C sur un processeur, on peut obtenir la figure (3.11). On continuera de la même manière pour le placement de la tâche D.

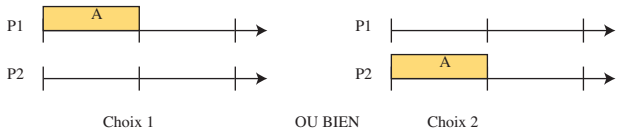


FIG. 3.9 – Etape 1.

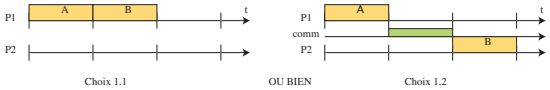


FIG. 3.10 – Etape 2.

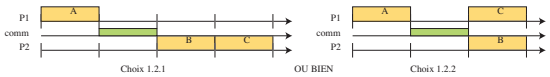


FIG. 3.11 – Etape 3.

Parmi tous ces choix possibles, il faut choisir la meilleure solution, à savoir il faut sélectionner le meilleur choix parmi les solutions possibles qui vont du choix 1.1.1.1 au choix x.x.x.x . La résolution de ce problème est NP-difficile. Afin de résoudre ce problème d'optimisation du temps de réponse, une heuristique a été développée. Il s'agit d'un algorithme glouton dont chaque étape alloue une opération à un processeur, route les éventuelles communications inter-processeurs c'est-à-dire crée des opérations de communication et alloue chacune d'elles à une liaison physique. L'ordonnancement des opérations de calculs ou de communication est directement déduit de l'ordre dans lequel elles sont allouées.

Cette méthode consiste donc à faire progresser au long du graphe une coupe séparant les opérations déjà placées sur des processeurs de celles qui ne le sont pas encore. La progression se fait en respectant les précédences du graphe logiciel. De toutes les opérations à distribuer sur la coupe et de tous les processeurs, on choisit la paire qui optimise une fonction locale de coût prenant en compte :

- les différences entre dates locales d'exécution au plus tôt et au plus tard (schedule flexibility),
- l'allongement du temps global d'exécution : le temps de réponse (latence),
- le rythme d'entrée (cadence),
- la capacité mémoire.

Afin d'illustrer l'adéquation, la figure (3.12) montre le graphe temporel d'exécution d'un algorithme, tiré du tutoriel SynDEx, sur l'architecture de la figure (3.7). Le temps se déroule de haut en bas avec une colonne par processeur (`root`, ...) ainsi qu'une colonne par média de communication (`bus`). Chaque opération de calcul est représentée par une boîte dont la hauteur est proportionnelle à la durée d'exécution de l'opération. Chaque communication inter-processeurs est représentée par une boîte dont la taille est proportionnelle à la durée de la communication. La communication commence dès que l'opération qui a fournit la donnée à transmettre est terminée, l'opération qui a besoin de la donnée transférée commence dès que la communication est terminée. La valeur de la durée d'une itération du graphe est, quant à elle, donnée dans la fenêtre principale de SynDEx.

3.5.4 Noyaux d'exécutif

Génération d'exécutif

Les transformations de graphes modélisant le processus d'implantation de l'algorithme sur l'architecture, permettent de produire automatiquement des exécutifs temps réel optimisés, déchargeant ainsi l'utilisateur des tâches fastidieuses de programmation bas niveau et autorisant du même coup une meilleur concentration sur les problèmes directement liés au programme applicatif.

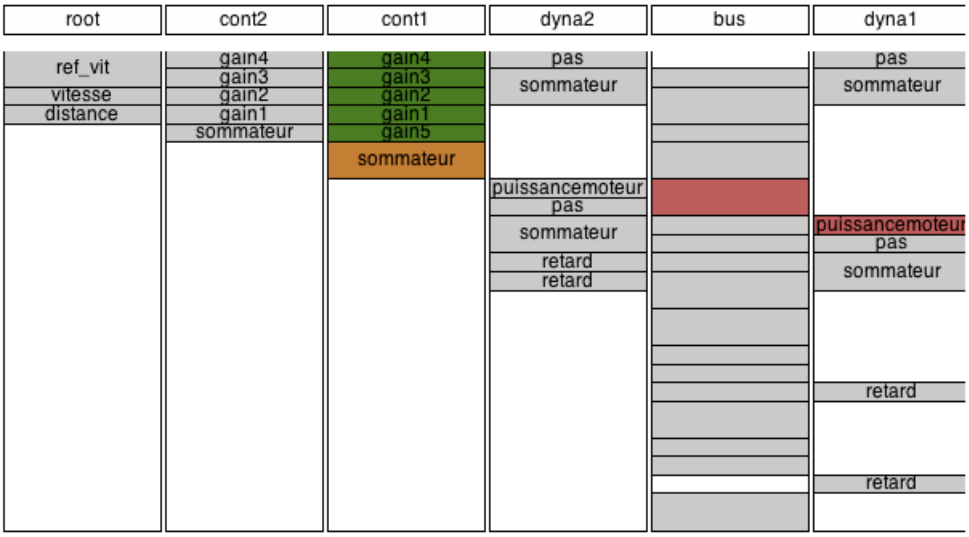


FIG. 3.12 – Graphe temporel pour l'application du tutorial.

Un exécutif a pour rôle d'allouer les ressources de l'architecture matérielle (unités de calcul, de mémoire et communication) au programme d'application. Les exécutifs peuvent être classés en fonction de leur manière d'arbitrer l'allocation des ressources. Nous avons vu précédemment que cette allocation est à la fois spatiale (distribution) et temporelle (ordonnancement). Si les optimisations et les décisions que doit prendre l'exécutif sont effectuées à l'exécution, on dit que l'allocation est dynamique. Par contre, si cela est fait avant l'exécution, on dit que l'allocation est statique (il faut connaître les durées d'exécution). Dans le cas d'une allocation statique, les exécutifs sont les moins pénalisants car dans le cas d'une allocation dynamique, l'exécutif consomme une partie des ressources pour effectuer ses décisions d'arbitrage et d'optimisation. C'est pourquoi SynDEx génère des exécutifs statiques. Il y en a autant que de processeurs dans l'architecture.

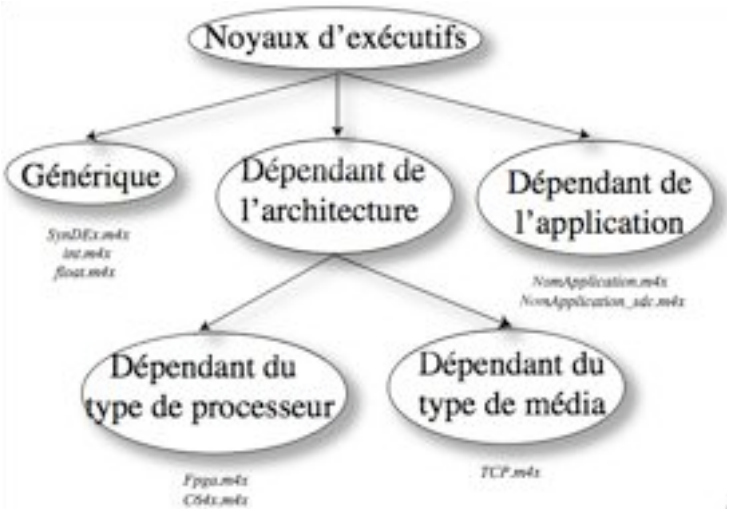


FIG. 3.13 – Arborescence des exécutifs SynDEx

Comme les exécutifs sont générés automatiquement, l'utilisateur n'a pas d'autre code à écrire que celui de ses opérations de calculs et d'entrée/sortie. Tout le reste, c'est-à-dire la distribu-

tion, l'ordonnancement, les appels des opérations de calcul et d'entrée/sortie, les allocations de la mémoire nécessaires aux communications inter-opérations est généré automatiquement à partir des graphes logiciel et matériel, des résultats de l'optimisation et d'un noyau générique. En effet, pour chaque processeur, l'exécutif est constitué par un assemblage d'éléments d'un noyau générique d'exécutifs (tirés d'une bibliothèque système) qui gère les communications inter-processeurs. Chaque exécutif généré par SynDEx est un macro-code indépendant de l'architecture. Ce macro-code est ensuite macro-processé avec des noyaux d'exécutifs dépendant de l'architecture et de l'application, afin d'obtenir du code source, qui après compilation sera exécutable.

Arborescence des noyaux d'exécutifs

Les noyaux d'exécutifs sous SynDEx sont divisés en différents groupes comme le montre la figure (3.13). Le travail de ce stage qui a consisté à aider à la conception des noyaux applicatifs (dépendants de l'application) sera expliqué plus en détail dans la deuxième partie de ce rapport.

Deuxième partie

Architecture matérielle et OS

Chapitre 4

Architecture matérielle d'un CyCab

4.1 Histoire de la conduite automatique

Dans le cadre de la route automatisée, l'INRIA a imaginé un système de transport original de véhicules en libre-service pour la ville de demain. Ce système de transport public est basé sur une flotte de petits véhicules électriques spécifiquement conçus pour les zones où la circulation automobile doit être fortement restreinte. Pour tester et illustrer ce système, un prototype, nommé CyCab (contraction pour Cybar Cab), a été réalisé (Figure 4.1).

Fig. 4.1 – Un Cyber Cabi.

Les chercheurs de l'INRIA et de l'Inrets (Institut National de Recherche sur les Transports et leur Sécurité) travaillent depuis 1991 sur de nouveaux moyens de transport intelligent pour la ville. Ils étudient en particulier le concept du libre-service et celui de la voiture automatique. Les premiers résultats de recherche ont débouché sur le projet Praxitèle (1993-1999), qui était en exploitation à Saint-Quentin-en-Yvelines. Les partenaires industriels du projet étaient CGFTE (la filiale transports publics de Vivendi), Dassault Electronique, EDF et Renault.

Dans le cadre du projet Praxitèle l'INRIA a démontré la faisabilité de la conduite automatique sous certaines conditions : créneau et train de véhicule expérimenté sur une Ligier électrique instrumentée à cet effet.

Pour des raisons de législation et de responsabilité ces automatismes de conduite n'ont pas pu être implémentés sur les Clio électriques de Saint-Quentin-en-Yvelines. Le CyCab a ensuite été développé par l'INRIA avec l'aide de l'Inrets, de EDF, de la RATP et de la société Andruet S.A. pour montrer le potentiel de l'informatique dans la conduite de véhicules. Le CyCab est un

véhicule électrique à quatre roues motrices et directrices avec une motorisation indépendante pour chacune des roues et pour la direction. Pour contrôler et commander les 9 moteurs du CyCab (4 de traction, 1 de direction et 4 de frein), une architecture matérielle a été choisie. Elle est constituée de noeuds intelligents pouvant gérer les différents moteurs du CyCab et répartie autour d'un bus de terrain CAN (Controller Area Network), très répandu dans le monde de l'automobile.

Le rôle des noeuds est d'asservir les moteurs en fonction des consignes de vitesse et de braquage qui transitent sur le bus CAN soit en provenance de la position du joystick, soit par un programme de planification de trajectoires. Le noeud doit donc non seulement être capable de fournir la puissance nécessaire aux moteurs, mais aussi exécuter les boucles d'asservissement de vitesse ou de position. Pour ce faire il doit prendre en compte un certain nombre d'informations en provenance des capteurs proprioceptifs : état, odomètrie, fins de course, mesures de température, de courant, ...

Un train de véhicules est constitué d'un véhicule de tête conduit par un chauffeur et d'autres véhicules automatisés, chacun suivant celui qui le précède. Ainsi le premier véhicule est suivi par le deuxième qui à son tour est suivi par le troisième ... C'est donc une procession de véhicules. Ce type d'automatisation a été pensé pour les conduites sur autoroute ou dans les périphériques. Ce procédé a l'avantage de maximiser la vitesse des véhicules ainsi que leur nombre tout en minimisant les accidents.

4.2 Travail effectué

Un des problèmes que j'ai rencontré pendant ce stage fût le manque de documentation. J'ai ajouté dans ce rapport ma propre documentation. J'ai également effectué une mise à jour matérielle et logicielle du PC embarqué et fait quelques maintenance sur les noeuds MPC.

4.3 Architecture matérielle

4.3.1 Caractéristique générale d'un CyCab

Il existe différents types de CyCab. Celui utilisé a les caractéristiques suivantes :

- longueur : 1,90 m
- largeur : 1,20 m
- poids total avec batteries : 300 kg
- 4 roues motrices et directrices
- vitesse théorique maximale : 20 km/h
- autonomie : 2 heures d'utilisation continue
- capacité d'accueil : 2 personnes
- conduite manuelle ou automatique.

La figure (4.2) montre une vue en coupe de l'architecture du CyCab qui est constituée de :

- 1 ensemble de batteries avec un gestionnaire automatique de charge (10) et un bouton arrêt d'urgence qui est soit de type poussoir (2) soit de type radiocommandé (1).
- 2 cartes électroniques (5) et (6) d'acquisition de données comprenant chacune un microprocesseur 32 bit Power PC (appelés MPC555). Chaque carte permet de contrôler 2 roues du CyCab. Nous reviendrons plus tard sur l'architecture de ces cartes que l'on appellera par la suite *noeuds*.
- 1 PC embarqué au format rack (taille 2U), placé sous le siège (2), possédant un processeur Intel cadencé à 3 GHz, avec un Linux temps réel, RTAI. L'ensemble est alimenté par une tension d'entrée de -48V (350W) et non de 220V. L'écran est situé en (3).

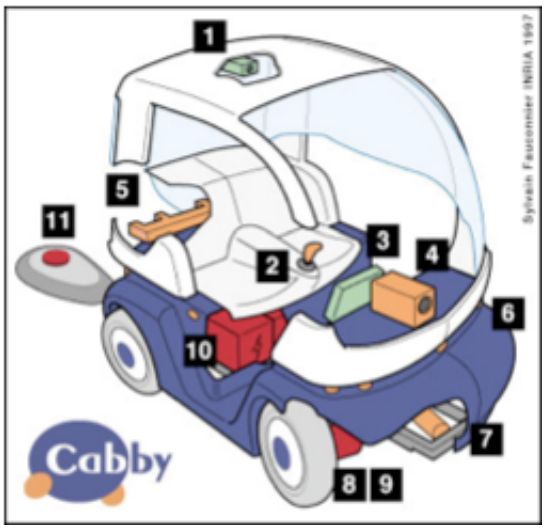


Fig. 4.2 – Architecture d'un CyCab.

- 2 bus CAN indépendants : le bus CAN 0 permet la communication entre les 2 MPC555 et le PC embarqué, alors que le bus CAN 1 permet d'ajouter d'éventuels futurs composants électronique.
- 4 moteurs et leurs freins électriques (8) et (9) contrôlés par 4 contrôleurs de moteur appelés Curtis PMC 1227 (9) servant d'amplificateurs de puissance pour contrôler la vitesse des roues. La consigne de vitesse est donnée par une tension de 0 à 5V aux Curtis qui fourniront des signaux PWM adéquats aux moteurs. Les Curtis protègent les noeuds des contre-courants des moteurs, quand par exemple, on les arrête brusquement.
- 4 décodeurs incrémentaux donnant la vitesse des roues (8).
- 1 vérin de direction électrique alimenté par signal PWM (7) faisant pivoter les 4 roues.
- 1 encodeur absolu avec sortie SPI et donnant l'angle des roues.
- 1 joystick (2) fournissant deux courants indiquant : – la consigne de vitesse des roues, – la consigne de direction des 4 roues.
- Depuis ce stage, le CyCab possède une caméra type webcam (4) se branchant sur un port FireWire du PC embarqué.

Nous verrons, dans les prochains chapitres, comment ces capteurs et actionneurs sont utilisés pour faire rouler le CyCab que ce soit pour de la conduite manuelle ou de la conduite automatique.

4.3.2 L'architecture du système

La figure (4.3), montre en formalisme UML, l'architecture complète du CyCab avec ses moyens de communication, à savoir les deux noeuds, le PC embarqué et les deux bus CAN.

Comme nous l'avons expliqué dans le chapitre précédent, SynDEx va distribuer le programme de conduite sur les 2 noeuds et sur le PC embarqué qui communiqueront grâce au bus CAN 0. Le passager (ou acteur en formalisme UML) du CyCab peut communiquer avec le PC embarqué (clavier/souris/écran, USB, FireWire, Ethernet, CAN 1) mais n'a accès aux noeuds que par l'intermédiaire du bus CAN 1.

Comme nous le verrons, chapitre 5, une partie du programme de conduite (manuel ou automatique) va tourner sur un Linux temps réel (RTAI) servant essentiellement de timer 10 ms aux deux noeuds MPC. La partie LXRT va gérer les images de la caméra, appliquer le traitement de l'image et communiquer avec RTAI via une mémoire partagée. Un programme Linux peut

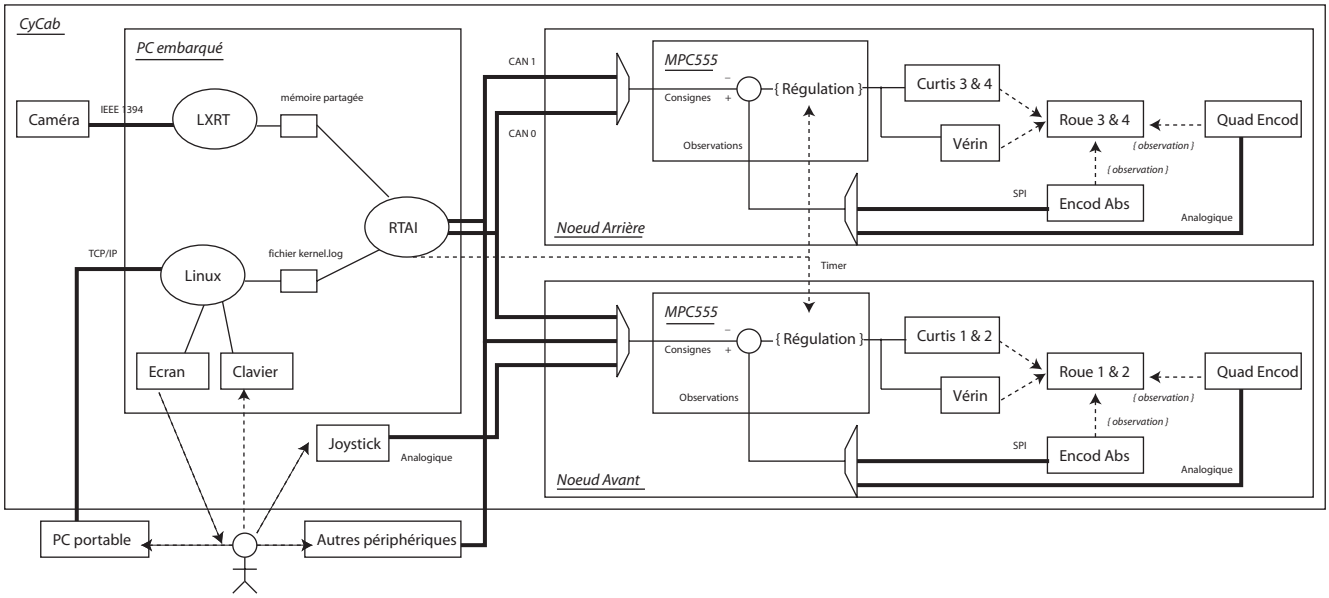


FIG. 4.3 – Architecture d'un CyCab.

observer le flot de données pour, par exemple, pouvoir les faire rejouer en simulation.

Comme nous le verrons, chapitre 7, le programme de conduite s'exécute sur les deux noeuds. Il lit les données fournies par les capteurs (direction des roues, vitesse des roues, ...). Il calcule la régulation et puis envoie le résultat aux différents actuateurs gérant les quatre roues du CyCab (traction et direction). La communication se fait soit par sortie série SPI, soit par lecture analogique ou optique.

4.3.3 Noeuds à coeur MPC

Les noeuds du CyCab sont constitués de quatre parties démontables, dont nous allons expliquer l'utilité :

1. une carte mère,
2. une carte fille s'emboîtant sur la carte mère,
3. un micro-contrôleur Motorola MPC555,
4. une petite boite, de couleur noire.

Les entrées-sorties de la carte mère (à gauche, figure (4.4)), sont :

1. L'emplacement du MPC555. Par convention avec les autres CyCab de l'équipe IMARA, on donnera l'identifiant 4000 au processeur du noeud arrière et l'identifiant 4001 au processeur du noeud avant.
2. L'emplacement de l'alimentation. Selon l'ancienneté du CyCab, la tension délivrée par l'alimentation est soit de 24 VDC, soit de 48 VDC. La plupart des cartes mères ont été modifiées pour fonctionner avec ces 2 types de tension. Elles sont aussi censées être protégées électriquement de l'extérieur grâce à des optocoupleurs (composants de couleur blanche sur la figure (4.4)) mais des problèmes ont été détectés à ce niveau.
3. C'est l'entrée de la boite de couleur noire. Sans cette boite le CyCab ne peut démarrer. Selon les dires de Robosoft, elle permet de remettre à zéro le MPC555 lorsque le bouton arrêt d'urgence est enfoncé.
4. L'entrée du joystick (uniquement pour le noeud avant du CyCab).

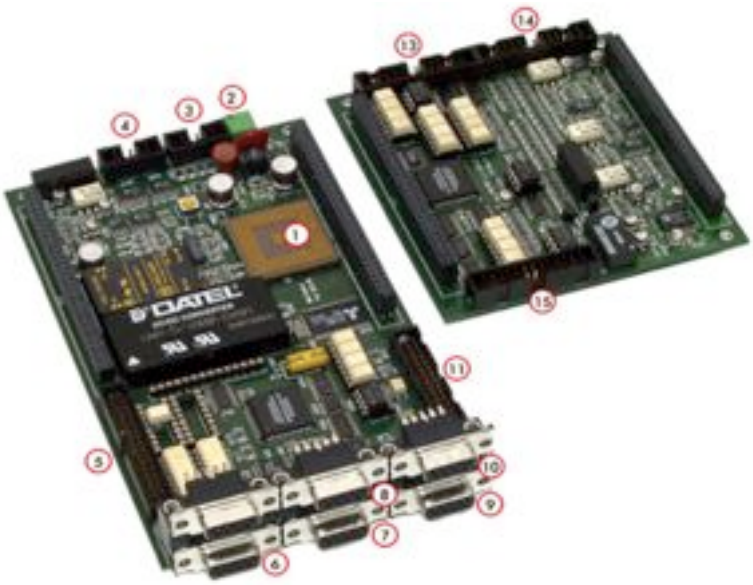


Fig. 4.4 – Carte mère (à gauche) et fille (à droite).

5. Une prise électronique permet de connecter des fils et d'en propager d'autres sur l'axe 11. Ces fils sont la masse et un signal 5V.
6. L'entrée/sortie sérielle de type SPI vers l'encodeur absolu. Seul le connecteur du bas est utilisé.
7. Ne sert pas.
8. Ne sert pas.
9. L'entrée/sortie CAN 0 permettant la communication avec l'autre noeud et le PC embarqué. Par convention avec les autres CyCab de l'équipe IMARA, la vitesse du bus CAN est de 800 Kbit/s.
10. L'entrée/sortie CAN 1 permettant de brancher des équipements externes au CyCab (PC portable par exemple).
11. Masse et un signal 5 V.

Je n'ai pas analysé les entrées/sorties de la carte fille, figure (4.4) à droite. Mais il faut brancher les axes 13, 14 et 15 aux câbles dont les numéros correspondent et qui proviennent du châssis.

Après réparation d'une paire de noeud, l'application de conduite manuelle dont nous parlerons plus en détail chapitre 7, s'arrêtait brusquement lorsque les roues se mettaient à tourner. Robosoft, nous a dit que l'actuateur de direction des roues générait de forts champs électromagnétiques (CEM) qui perturbent la carte mère. Il nous a conseillé de faire un rappel de masse, à savoir brancher un fil connectant la masse des capteurs de direction avant et arrière à la carcasse du CyCab. L'application du chapitre 7 a re-fonctionnée depuis.

4.4 PC embarqué

Le CyCab contient un PC embarqué, de forme horizontale (format rack taille 2U), placé sous le siège, qui communique avec les 2 noeuds via le bus CAN 0. Avant mon stage, le PC possédait un processeur Intel à 233 MHz, un Linux temps réel (kernel 2.4, RTAI 2.4 et dont le

serveur X ne pouvait se lancer), ce qui était suffisant pour faire tourner l'application de conduite manuelle (dont nous parlerons plus en détail chapitre 7). Mais sa vitesse est trop faible pour faire du traitement de l'image, et donc par conséquent, de faire du suivi automatique basé sur une caméra.

J'ai remis à jour, l'architecture du PC embarqué. Il est désormais constitué de :

- Le même châssis alimenté par une tension -48 V.
- Une carte mère contenant un Pentium 4 cadencé à 3 GHz et 512 Mo de mémoire vive.
- Une carte de fond de panier, référence PCI-5SDA-RS-R30 à 2 slots PCI et 4 ISA. Une nouvelle version contient 4 slots PCI (confère figure (4.5)).
- Une carte SPI pour la communication CAN.
- Une carte SPI pour la communication IEEE 1394.
- Une carte graphique PCI NVidia GX 5200, la puce graphique incluse dans la carte mère est suffisante pour un affichage confortable.
- Un disque dur IDE à 20 Go.

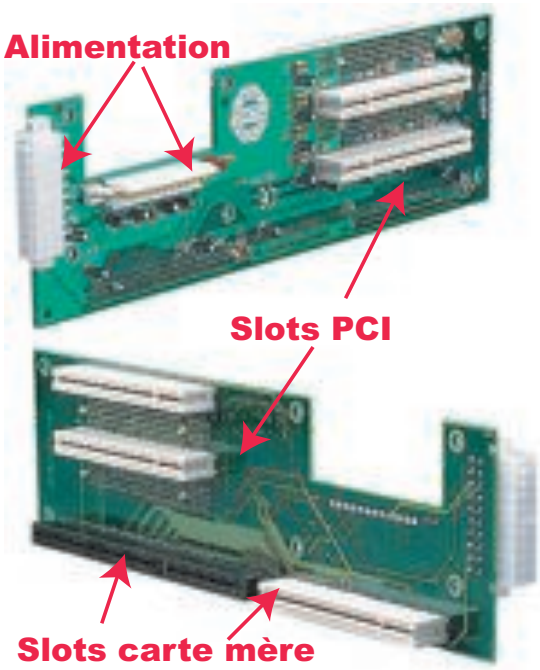


FIG. 4.5 – Carte riser.

FIG. 4.6 – Carte mère.

Le système d'exploitation du PC est un Linux temps réel dont la distribution est une Debian 4.0 release 0. Le noyau Linux installé par défaut par Debian est le 2.6.18. Un noyau 2.6.18-52 a été compilé et patché pour le temps réel avec RTAI 3.4. Le chapitre 5 expliquera plus en détail son fonctionnement.

Il est intéressant de noter que le format de la carte mère du PC embarqué n'est pas identique au format ATX des cartes mères des PC personnels. Sa largeur est la moitié d'une carte mère d'un PC standard et contient que le processeur et la mémoire vive. Elle possède 2 slots mâles qui lui permettent de s'insérer dans une carte de fond de panier, ou *riser* en anglais, pour pouvoir ajouter des cartes PCI ou ISA. Voir figure (4.5).

4.5 Capteurs et actuateurs

4.5.1 Caméra FireWire

La caméra FireWire que nous utilisons est une Fire-I fabriquée par UniBrain (4.7), sa taille est 35×65 mm. Sa résolution maximale est de 640×480, elle peut donner des images au format monochrome, RGB ou YUV à des fréquences de 30, 15, 7.5 et 3.5 images par seconde.

Fig. 4.7 – Caméra UniBrain Fire-I.

Nous utilisons les librairies libraw1394 et libdc1394 [18] pour obtenir les images. Les configurations que nous avons choisies sont : format YUV422, taille 320×240, 7.5 images par seconde. Ces options peuvent facilement être modifiées.

Les termes FireWire, IEEE 1394 ou i.Link, sont synonymes. L'article [17], nous dit que ce bus est un bus série plug and play, pouvant supporter jusqu'à 63 périphériques et permettant différentes topologies alors que l'USB ne permet que des configurations en étoile. Le débit maximum de la norme 1394 le plus répandu est de 400 Mbit/s, ce qui permet de fournir des débits plus élevés que celui de la norme USB 2.0.

Il existe 2 modes distincts pour le bus 1394 : – le mode asynchrone et – le mode isochrone. Le mode isochrone est le plus rapide car il permet l'envoie de paquets de taille fixe à intervalle de temps régulier et donc il n'existe plus d'accusé de réception. Même si le débit théorique est de 400 Mbit/s, le cadencement des paquets d'une transmission isochrone fait que le débit utile est de 256 Mbit/s.

On fera donc attention à ne pas dépasser ce débit maximum si on veut brancher, par exemple deux caméras sur une même carte pour faire par exemple, du traitement de l'image avec des images stéréo. Pour cela on jouera sur les paramètres suivants : – la taille de l'image, son format, – son taux de compression, – le nombre d'images par seconde. Par exemple, une image YUV422 de taille 640×480 à 30 images par seconde aura un débit de 147 Mbit/s. Il n'est donc pas possible de mettre deux caméras sur la même carte, il faut les brancher sur deux cartes différentes.

4.5.2 Contrôleur Curtis

La figure (4.8) montre une photo de l'avant d'un CyCab sans sa coque. L'arrière est la réplique exacte de l'avant. Nous ne voyons pas sur la photo les roues gauche et droit, mais nous voyons leurs suspensions, le vérin qui permet la direction, le contrôleur PWM qui gère le vérin, le contrôleur Curtis gérant la motricité des roues, ainsi que le noeud à coeur MPC.

Comme le but de mon stage était de faire du suivi longitudinal de CyCab, je me suis uniquement documenté sur le contrôleur Curtis PMC 1227. Dont voici le résumé du fonctionnement.

Le datasheet du Curtis PMC 1227 [27], nous indique que ce composant est un contrôleur pour moteur électrique. Il fournit au moteur une tension comprise entre 24 et 48 V (allant jusqu'à 200 A) et il programmable grâce à une sonde.

La sonde, nous indique que les paramètres suivants on été mis.

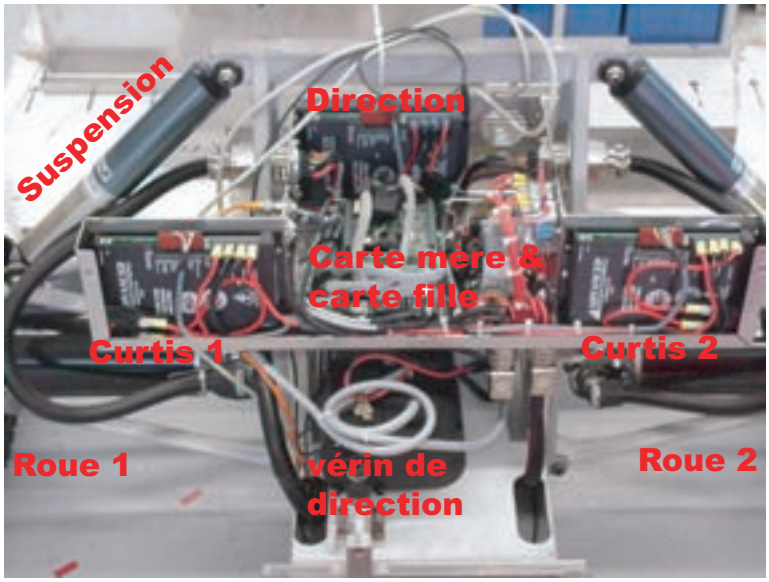


FIG. 4.8 – CyCab sans sa coque, vu de l'avant.

Nom	valeur (%)	Nom	valeur (%)
Throttle	0	M1,2 rec decel	0
Spd limit pot	100	M1,2 max speed	100
Batt volt	48.5	M1,2 min speed	0
Mode input A	on	M1,2 main C/L	60
Frwrd input	off	M1,2 IR coef	0
Reverse input	off	reverse speed	100
Inhibit in	on	Ramp shape	50
main cont	off	Creep speed	0
em brake drvr	off	Brake DLY	1.0
push enable in	off	Thrttle type	0
Thrtl autolocal	off	direct	0
M1, M2 accel rate	0	Thrttle gain	100
M1, M2 decel rate	0	thrttle deadband	5.0
high pedal dis	on		

Avec ses informations et en se rapportant au manuel, on en déduit la figure (4.9) qui, à gauche, montre la correspondance entre la plage de tension entrant dans le Curtis et à droite, la sortie PWM (signal carré dont le rapport cyclique est variable) alimentant le moteur DC.

Dans la plage de tension 2.4V et 2.6V, le moteur ne tourne pas, c'est la zone neutre. De 0.58V à 2.4V et de 2.6V à 4.44V la sortie PWM varie linéairement par rapport à la tension.

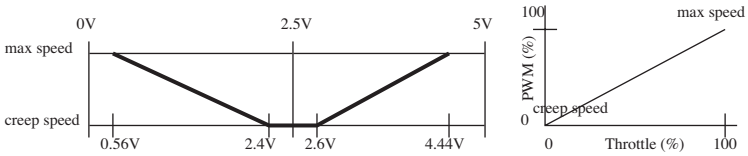


FIG. 4.9 – Comportement du Curtis.

4.5.3 Vérin de direction

On a remarqué que les roues du CyCab n'étaient pas bien centrées. Le vérin doit être réglé.

4.6 Problèmes rencontrés

4.6.1 Noeuds MPC

J'ai eu un problème électronique avec les cartes mères. Lors d'un test, une carte refusait de communiquer. La réparation de la carte par la société Robotsoft a duré 2 mois. La nouvelle carte, ne fonctionnait pas correctement, non plus.

4.6.2 PC embarqué

Lors du remplacement du PC embarqué, une barrette mémoire RAM était défectueuse. Il est intéressant de savoir diagnostiquer ce problème. Dans mon cas elle empêchait l'installation de certaines distributions Linux et quand bien même, l'installation réussissait, des messages d'erreur du type `*** glib detected *** corrupted double-linked list : 0xXXXXX` apparaissaient. Une méthode pour diagnostiquer ce problème consiste à installer l'image `memtest` qui se comporte comme une image noyau Linux et donc se lance avec `grub` et teste la mémoire.

4.6.3 Risque potentiel avec le joystick

Un problème grave peut arriver si le joystick se casse. Les plages de valeur des signaux analogiques du joystick sont comprises entre 0 et 5V. Le zéro est codé avec la tension 2.5V. Ce centrage est très dangereux car si le joystick a une défaillance, la tension devient nulle et le contrôleur croit que la consigne est au maximum et donc emballe les roues.

Chapitre 5

Système d'exploitation Linux

5.1 Travail effectué

J'ai profité de la mise à jour des cartes électroniques du PC embarqué pour installer un système d'exploitation Linux plus récent. Je lui ai appliqué un patch pour le faire fonctionner avec le temps réel.

J'explique ici quelques notions de Linux comme le but du noyau, les modules, la différence entre espace noyau et utilisateur, la différence entre LXRT et RTAI. Dans les annexes V, j'explique comment installer RTAI et surtout comment configurer et compiler le noyau Linux. Dans le chapitre 10, j'explique le travail que j'ai effectué, à savoir : corriger un problème dans la communication entre LXRT et RTAI.

5.2 Rappels

On définit un programme comme étant une séquence d'instructions qui doit être exécutée dans un certain ordre. Un logiciel ou application est un ensemble de programmes, qui permet à un ordinateur ou à un système informatique d'assurer une tâche ou une fonction en particulier. Un processus est un programme en train de s'exécuter. En informatique, on ajoute certaines propriétés à la définition précédente. En l'occurrence, un processus se compose d'un texte de programme (du code machine) et d'un état du programme (point de contrôle courant, valeur des variables, pile des retours de fonctions en attente, descripteurs de fichiers ouverts, etc.). Une tâche est un concept abstrait que l'on rapproche d'un processus dans le sens d'un travail ou d'une responsabilité qui est alloués et qui doit être exécutée.

Le noyau est le composent central de la plupart des systèmes d'exploitation. Il est responsable de la gestion des ressources du système et de la communication entre le matériel et les logiciels utilisateurs (confère figure (5.2)). Il fournit la couche d'abstraction la plus basse (confère figure (5.1)) des ressources (comme la mémoire, le processeur et les dispositifs d'entrée-sortie). Cela permet d'avoir accès aux ressources physiques par des mécanismes de communication inter processus et par des appels systèmes. Cette encapsulation du matériel libère les développeurs de logiciels de la gestion complexe des périphériques et d'écrire des applications génériques.

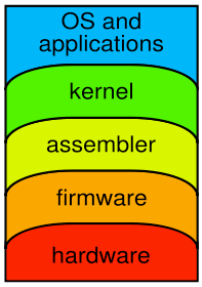


Fig. 5.1 – Abstractions [16].

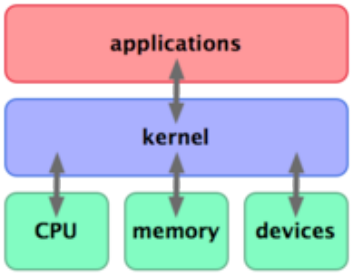


Fig. 5.2 – Connexions du noyau [16].

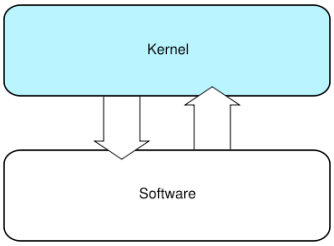


Fig. 5.3 – Noyau monolithique [16].

Il existe différents types de noyau, comme par exemple les noyaux monolithiques, les micro-noyaux, etc. Celui qui nous intéresse ici est le noyau de Linux qui est un noyau monolithique (confère figure (5.3)). Sa mémoire est divisée entre l'espace utilisateur et l'espace noyau. L'espace utilisateur désigne la région mémoire dédiée aux applications, à l'exclusion du noyau lui-même, qui fonctionne dans son propre espace mémoire. L'espace noyau est l'endroit où le code réel du noyau réside après son chargement, et où la mémoire est allouée pour les opérations qui prennent place à son niveau. Ces opérations incluent l'ordonnancement, la gestion des processus et des signaux, des entrées/sorties assurées par les périphériques, de la mémoire et de la pagination.

Cette partition entre espace utilisateur et espace noyau permet d'obtenir une certaine forme de sécurité : les applications de l'espace utilisateur ne peuvent, par accident ou intentionnellement, accéder à une zone mémoire ne leur appartenant pas : une telle action déclenche immédiatement une trappe du noyau, qui doit envoyer un signal particulier au programme et, généralement, le terminer. Pour que ce mécanisme fonctionne, il faut que les processeurs disposent d'une unité de gestion mémoire (MMU) exploitable par le noyau.

Un module est une partie du noyau qui peut être intégrée pendant le fonctionnement du système d'exploitation. C'est une alternative aux fonctionnalités compilées dans le noyau qui ne peuvent être modifiées qu'en relançant le système.

5.3 Différences entre modules du noyau Linux et applications Linux

Il existe des différences entre un module du noyau et une application (outre le fait de leur séparation de leur espace mémoire [25]). Une application (espace utilisateur) accomplit généralement une tâche unique du début jusqu'à la fin de sa vie. Un module (espace noyau) s'enregistre auprès du noyau pour servir les futures requêtes mais sa fonction d'initialisation se termine immédiatement. En d'autres termes, le but de la fonction d'initialisation *module_init* est de s'enregistrer au près du noyau afin de préparer la future invocation de la fonction du module, puis s'endort. La fonction d'arrêt du module *module_exit* est invoquée juste avant que le module soit déchargé du noyau. Elle indique au noyau que le service qu'elle fournit ne sera plus actif.

Une autre différence entre module du noyau et une application est que l'arrêt de l'application peut se faire de manière *fainéante* en libérant pas ses ressources (mémoire, descripteur de fichier, ...). L'arrêt d'un module devra se faire correctement sinon les ressources resteront présentes jusqu'au redémarrage du système.

Les développeurs savent que quand ils développent une application, ils peuvent appeler des fonctions non connues mais qui le seront au moment de *link* la résolution des références

externes vers les bibliothèques appropriées. Par exemple la fonction *printf* est une des fonctions définie dans la bibliothèques *libc*. Un module, quand à lui, est lié seulement avec le noyau, et les seules fonctions qu'il peut appeler sont celles qui sont exportées par le noyau. Il n'y a pas de bibliothèques avec qui on peut se lier. La fonction *printk* est une version de la fonction *printf* définie dans le noyau et exportable dans un module. Remarquons que *printk* ne supporte pas l'affichage des nombres en virgule flottante.

Les modules du noyau Linux sont généralement placés dans /lib/modules. Ils utilisent l'extension .ko depuis la version 2.6. Voici un exemple de création d'un module, tiré de [25] :

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");
static int hello_init(void)
{
    printk(KERN_ALERT "Hello, world\n");
    return 0;
}
static void hello_exit(void)
{
    printk(KERN_ALERT "Goodbye, cruel world\n");
}
module_init(hello_init);
module_exit(hello_exit);
```

Après compilation, on obtient un fichier hello.ko que l'on charge dans le noyau avec la commande `insmod ./hello.ko`. On retire le module avec la commande `rmmod hello`.

5.4 Linux temps réel

5.4.1 RTAI

Le projet RTAI, pour Real Time Application Interface, a pour origine le département d'ingénierie aérospatiale DIAPM de l'école polytechnique de Milan. Pour des besoins internes, Paolo Montegazza du DIAPM entreprit de développer un produit inspiré de RTLinux dont la stratégie est de faire cohabiter le noyau Linux avec un noyau auxiliaire basé sur un vrai ordonnanceur temps réel à priorités fixes. Les tâches temps réel sont gérées par ce noyau RTAI et le traitement d'autres tâches est délégué au noyau Linux, lui-même considéré par le noyau temps réel comme étant une tâche de plus faible priorité (*idle task*. La figure (5.4) montre cette architecture.

En effet, le code de masquage des interruptions a été réécrit toutes les interruptions hard sont initialement traitées par le noyau temps réel et transmises au noyau Linux seulement si l'interruption ne correspond pas à une tache temps réel. En bref, les générations d'interruptions softs sont laissées à Linux. L'ensemble des services tel que : les ordonnanceurs, les FIFO ou encore les allocations dynamiques de mémoire est fourni par des modules du noyau, que l'on charge dynamiquement en utilisant simplement les commandes standard de Linux : *insmod* ou *modprobe*.

5.4.2 LXRT

Les développeurs de RTAI ont récemment fait un effort particulier sur une extension appelée LXRT (LinuX Real Time) permettant de développer des tâches temps réel dans l'espace utili-

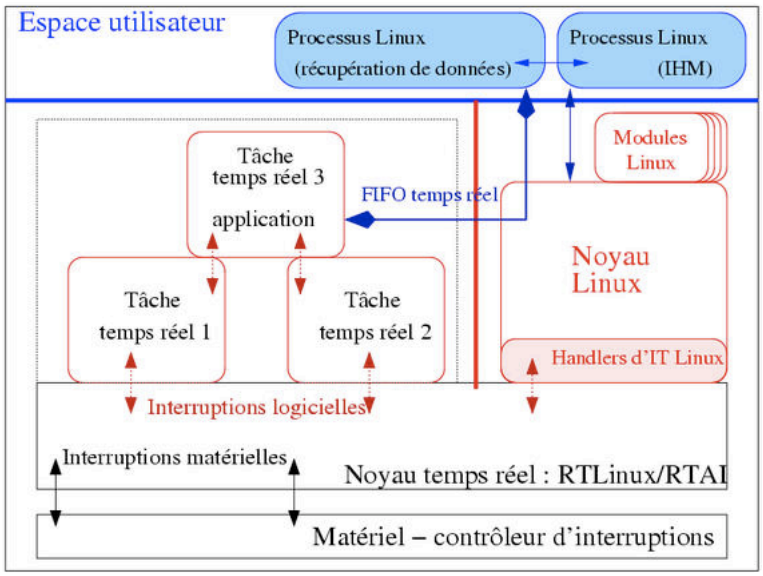


FIG. 5.4 – Architecture d'un Linux temps réel.

sateur et non plus dans l'espace noyau. Ce dernier point est extrêmement intéressant au niveau de la facilité de développement car nous savons que développer dans l'espace noyau est relativement plus complexe (la mémoire de l'espace noyau ne possède pas de mécanismes de protection contre les accès invalides ou encore de programmes pouvant amenés à un panne du système. La modification d'une zone mémoire non désirée cause la corruption générale du système).

Au niveau des performances, les résultats par rapport au même programme développé dans l'espace noyau sont très honorables et ce bien que LXRT soit encore en cours d'évolution.

5.4.3 Ordonnanceurs

L'ordonnanceur se trouve être le coeur de RTAI. Il fournit à travers une série de mécanismes les capacités temps réel. En utilisant l'ordonnanceur RTAI, le processus peut satisfaire des contraintes temps réel dures en exécutant les tâches de façon déterministe .

RTAI fournit des services inter et intra espace utilisateur et noyau temps réel durs symétriques. Un tel support passe à travers deux ordonnanceurs, qui à l'heure actuelle sont nommées *rtai_lxrt* pour LXRT et *rtai_sched* pour RTAI. Ils peuvent fonctionner aussi bien dans l'espace utilisateur que dans celui du noyau et ils se distinguent seulement par les objets qu'ils peuvent ordonnancer. Cela signifie que : – *rtai_lxrt* est un co-ordonnanceur GNU/Linux, – *rtai_sched* supporte non seulement le temps réel dur (pour l'ensemble des objets Linux ordonnançable, comme les processes/threads/kthreads) mais aussi les propres tâches noyau RTAI.

5.4.4 Comparaison de Linux et RTAI/LXRT

Finalement on distinguera 4 types de tâches selon leurs contraintes temporelles et l'espace mémoire dans lequel elles vivent :

- les *processus* qui s'exécutent dans l'espace utilisateur sans contrainte temporelle dure,
- les *modules* qui s'exécutent dans l'espace noyau sans contrainte temporelle dure,
- les *tâches RTAI* qui s'exécutent dans l'espace noyau avec contraintes temporelles dures,
- les *tâches LXRT* qui s'exécutent dans l'espace utilisateur avec contraintes temporelles dures.

Type de tâche	Avantages	inconvénients
Processus	• Elle laisse plus de ressources temporelles aux autres tâches. Car considéré par le noyau temps réel comme étant une tâche de plus faible priorité • On a accès aux fonctions Linux standards comme les fonctions (*open*, *write*). • Protection contre les violations aux accès mémoire (on a un segmentation fault plutôt qu'un kernel panic de l'OS) et outil de débugage.	• Très souvent préemptée.
Tâche LXRT	• Protection contre les violations aux accès mémoire (on a un segmentation fault plutôt qu'un kernel panic de l'OS) et outil de débugage. • On a accès aux fonctions Linux standards comme les fonctions (*open*, *write*).	• Moins réactive que RTAI car un appel système doit traverser la couche utilisateur.
Module	• Souplesse dans le développement de driver, car peut être chargé/déchargé dynamiquement, ce qui évite de redémarrer à chaque fois le système d'exploitation.	• Il faut charger/décharger des modules avant et après exécution du programme.
Tâche RTAI	• Souplesse dans le développement de driver, car peut être chargé/déchargé dynamiquement, ce qui évite de redémarrer à chaque fois le système d'exploitation. Plus réactive que LXRT.	• La programmation doit être soigneuse, car étant dans l'espace noyau, la mémoire n'est plus protégée : l'accès en écriture peut se faire partout et donc tuer l'OS entier. • On n'a pas accès aux fonctions Linux standards comme les fonctions (*open*, *write*, ...).

5.5 Noyau d'exécutif SynDEx pour RTAI

Le logiciel SynDEx possède un noyau d'exécutif pour RTAI. Comme on l'a expliqué précédemment ces exécutifs générés sont des modules, que l'on charge dans le noyau avec la commande `insmod`. Dans le cas de la génération de l'exécutif pour la conduite manuelle, on obtient un module appelé `root.ko` que l'on lance avec la commande `insmod`. L'arrêt de l'application, se fait, d'abord en appuyant sur le bouton rouge d'arrêt d'urgence du CyCab, puis en laçant la commande `make stop`.

Dans le cas de la génération de l'exécutif pour la conduite automatique, c'est plus compliqué car SynDEx ne possède pas de noyau exécutif pour LXRT. Il faut compiler et exécuter un programme LXRT qui communiquera avec le module `root`. La synchronisation de la communication n'est donc pas générée automatiquement.

Troisième partie

Conduite manuelle du CyCab

Chapitre 6

Notions d'automatique

Dans ce chapitre, des éléments d'automatique sont présentés rapidement afin de faciliter la compréhension des chapitres suivants où des applications SynDEx et Scicos sont données.

6.1 Rappel de quelques éléments de l'automatique

Considérons un bateau [8] ayant un pilote automatique recevant en permanence le cap actuel α du bateau et le cap désiré α_c. En utilisant ces informations le pilote automatique génère au cours du temps des ordres de positionnement ϵ du gouvernail de façon à ce que l'erreur de cap $e = \alpha_c - \alpha$ soit maintenue aussi faible que possible sachant que le bateau reçoit des perturbations extérieures (vent, ...). La figure (6.1) montre le schéma bloc du modèle comme nous l'avons montré dans la section **??**.

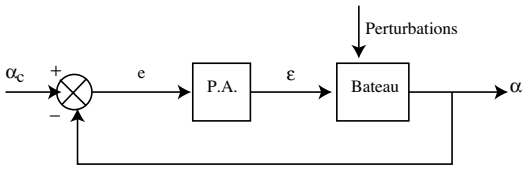


FIG. 6.1 – Pilote automatique de bateau (P.A.)

Pour ce faire, une loi de commande (bloc P.A.), calculant $\epsilon = f(e)$ en fonction des informations disponibles, pourrait être :

- par à-coups :
$$\epsilon = \begin{cases} +\epsilon_m & \text{si } e > 0 \ , \\ -\epsilon_m & \text{si } e < 0 \ . \end{cases}$$
Ca a l'avantage d'être simple, mais n'est pas très efficace. Le système va osciller (un coup trop à gauche, un coup trop à droite, ...)
- proportionnelle :
$$\epsilon = Ke \ ,$$
Cette méthode à l'inconvénient de faire apparaître une erreur asymptotique.
- proportionnelle et intégrale :
$$\epsilon = Ke + 1/\rho \int^t e(\alpha) \, d\alpha \ ,$$
Le terme intégrale permet d'additionner les erreurs du systèmes. La sommation des erreurs va continuer jusqu'à ce que la valeur corresponde à la valeur de la consigne. L'inconvénient

du terme intégrale est que la réponse à une perturbation devienne plus lente que le seul terme proportionnel.

- proportionnelle et dérivée :

$$\epsilon = Ke + K'de/dt \ ,$$

  Le terme dérivé permet une réponse plus rapide lors d'un changement rapide de l'erreur. En théorie, ce terme aide le système a être plus réactif, mais en pratique, les bruits blancs (vibration, erreur de lecture des capteurs, ...) vont rendre instable le système (on dérive du bruit).
- proportionnelle et intégrale et dérivé etc.

C'est la théorie des asservissements qui permettra, dans un cas particulier de choisir la loi de commande la mieux adaptée. Le document AVR221 d'Atmel explique plus en détail et de façon très simple, l'implémentation d'un régulateur PID (dans le cas temps discret) et la méthode Ziegler-Nichols pour trouver les bons gains [10] car les systèmes deviennent instables quand les valeurs des gains sont trop forts.

Dans les paragraphes suivants on va introduire les outils permettant d'analyser la classe des systèmes linéaires temps invariant.

6.2 Représentation sous forme de matrice d'état

Un système linéaire temps invariant, peut s'écrire sous la forme de matrice d'état, comme pour l'exemple suivant, du mouvement d'un pendule inversé avec les conditions initiales nulles. On peut écrire le système avec x_1 l'angle et x_2 sa position, et u l'entrée u système et y la sortie (l'observation) :

$$\begin{aligned}\dot{x_1} &= x_2 \\ \dot{x_2} &= \frac{mgl}{J}\sin x_1 + \frac{ml}{J}u\cos x_1\end{aligned}$$

On peut écrire $\dot{x_2} = f(x_1, x_2, u)$. Grâce à la formule de Taylor :

$$\dot{x_2} \simeq f(0,0,0) + \delta x_1 f'_{x_1}(0,0,0) + \delta x_2 f'_{x_2}(0,0,0) + \delta u f'_u(0,0,0)$$

Ce qui donne :

$$\begin{aligned}\delta\dot{x_2} &\simeq \delta x_1\left(\frac{mgl}{J}\right) + \delta u\left(\frac{ml}{J}\right) \\ \delta\dot{x_1} &= \delta x_2\end{aligned}$$

Finalement, en changeant de notation $\delta x \to x, \delta u \to u, \delta y \to y$, on obtient le systeme suivant sous la forme matricielle :

$$\begin{aligned}\dot{x} &= \begin{bmatrix} 0 & 1 \\ \frac{mgl}{J} & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{ml}{J} \end{bmatrix} u \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} u\end{aligned}$$

Soit de la forme générale :

$$\begin{aligned}\frac{dx}{dt} &= Ax + Bu \\ y &= Cx\end{aligned}$$

6.3 Transformée de Laplace et transformée en z

Deux transformations permettent de ramener l'analyse des systèmes dynamiques linéaires à du calcul algébrique. Ce sont la transformée de Laplace et la transformée en z. Elles transforment des fonctions du temps en une fonction d'une variable dont la partie imaginaire s'interprète en terme de fréquence. Ces transformations permettent de résoudre les équations différentielles linéaire à coefficients constants.

6.3.1 Transformée de Laplace

Définition

La transformée de Laplace est définie de la manière suivante : soit $f(t)$ une fonction du temps définie pour $t > 0$. Alors :

$$\mathcal{L}[f(t)] \equiv F(s) \equiv \lim_{\substack{T\to\infty \\ \epsilon\to 0}} \int_{\epsilon}^{T} f(t)\, e^{-st} dt = \int_{0^+}^{\infty} f(t)\, e^{-st} dt \quad 0 < \epsilon < T$$

où s est un variable complexe défini par $s \equiv \sigma + j\omega$.

Transformée de Laplace de dérivées

Montrons que la transformée de Laplace de la dérivée df/dt d'une fonction $f(t)$ vaut :

$$\mathcal{L}\left[\frac{df}{dt}\right] = s\mathcal{L}f - f(0^+)$$

En intégrant par partie, on obtient :

$$\left[e^{-st} f(t)\right]_{0^+}^{\infty} + s \int_{0^+}^{\infty} f e^{-st} dt$$

Finalement :

$$\mathcal{L}\left[\frac{df}{dt}\right] = s\mathcal{L}f - f(0^+)$$

Autre exemple

Une autre formule utilisée dans l'application du chapitre suivant est la transformée de Laplace de la double dérivée d^2f/dt^2 d'une fonction $f(t)$:

$$\mathcal{L}\left[\frac{d^2f}{dt^2}\right] = s^2F(s) - sf(0^+) - \frac{df}{dt}\Big|_{t=0^+}$$

6.3.2 Transformée en z

La transformée en z est utilisée pour décrire des signaux en temps discret. Soit $\{f(k)\}$ dénote une séquence de valeur réelle $f(0), f(1), f(2), \ldots$ ou bien $f(k)$ pour $k = 0, 1, 2, \ldots$. Alors on définit la transformée en z par :

$$\mathcal{Z}\{f(k)\} \equiv F(z) = \sum_{k=0}^{\infty} f(k) z^{-k}$$

où z est un variable complexe défini par $z \equiv \sigma + j\omega$.

6.4 Placement de pôles

Un système linéaire temps invariant (LTI) ayant un nombre fini d'états s'écrit généralement de la façon suivante :

$$\frac{dx}{dt} = Ax + Bu$$
$$y = Cx$$

Où la A matrice est une matrice $n \times n$, B une matrice $n \times p$, C une matrice $q \times n$, . Sous block diagramme il se dessine comme sur la figure 6.2.

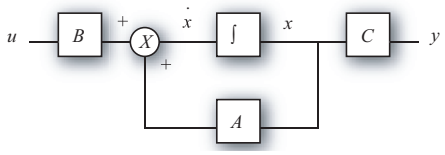


FIG. 6.2 – u est l'entrée, y la sortie, x l'état (ou mémoire)

La fonction de transfert d'un système linéaire temps invariant mono-entrée mono-sortie est une fonction rationnelle s'écrivant :

$$G(s) = C(sI - A)^{-1}B = \frac{N(s)}{D(s)} ,$$

où N et D sont deux polynômes.

On appelle pôles les zéros du polynôme $D(s)$, ils sont aussi les valeurs propres de la matrice A. La présence du pôle λ implique que la sortie du système $y(t)$ contient une composante de la forme $e^{\lambda t}$. Alors, si $Re(\lambda) > 0$ la sortie tend vers l'infini lorsque t tend vers l'infini. Le système est dit instable. Un système est asymptotiquement stable si et seulement si toutes les valeurs propres vérifient : $Re(\lambda_i) < 0$.

Considérons le système bouclé (u en feedback sur l'état) dans lequel $u = Kx + v$ le nouveau système s'écrit

$$\frac{dx}{dt} = (A + BK)x + Bv ,$$
$$y = Cx .$$

Le feedback étant à notre disposition on peut choisir ses coefficients de façon à placer les pôles du système bouclé où l'on veut. Par exemple rendre stable par feedback un système instable.

Un théorème indique sous quelles conditions sur les matrices (A, B, C) il est possible de placer les pôles du système où l'on veut.

6.5 Discrétisation d'une intégrale

SynDEx manipule seulement des modèles temps discret (et donc, pas de temps continu) et il n'est pas capable de manipuler des boucles algébriques implicites. C'est pourquoi, une boucle SynDEx doit contenir au moins un délais $(1/z)$. Par conséquent, notre application Scicos qui est un système dynamique temps continu doit être convertie en temps discret pour pouvoir être utilisée dans SynDEx.

L'équation différentielle $\dot{x} = u$ est discrétisée d'une façon simple en utilisant un schéma de Euler. Notons h le pas de discrétisation et x_0 une valeur initiale arbitraire. Le système discret s'écrit :

$$x_{n+1} - x_n = uh \tag{6.1}$$

Finalement, le système (6.1) est donné en Scicos (SynDEx) par la figure. 6.3 (6.4). Notons que la variable h, stockée dans le contexte de Scicos, est utilisée dans l'entrée du gain et dans la définition de l'horloge. Dans SynDEx, h est défini en tant que paramètre dans la définition d'un gain et l'horloge est utilisée directement dans le code source des opérations.

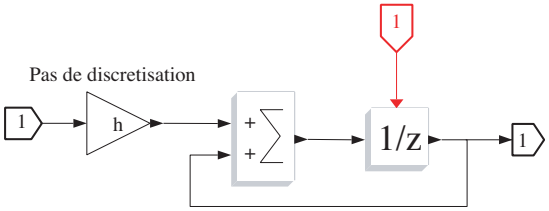


FIG. 6.3 – Une intégrale discrétisée dans Scicos (temps discret).

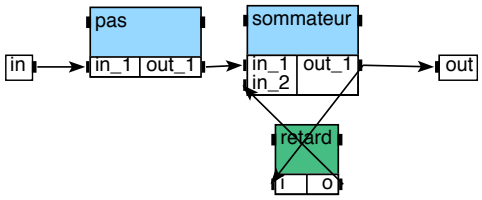


FIG. 6.4 – Une intégrale discrétisée dans SynDEx.

Chapitre 7

Modélisation de la conduite manuelle du CyCab

7.1 Observation des états du CyCab

7.1.1 Travail effectué

Il existe au moins deux applications SynDEx de conduite manuelle du CyCab : l'application *Manu*, et l'application *Robucar*. *Robucar* est l'application développée par la société Robosoft. Elle est utilisée par l'équipe IMARA comme base de développement. Elle est plus récente que l'application *Manu*. *Robucar* et *Manu* sont des régulateurs, après leur téléchargement sur l'architecture du CyCab (à savoir deux noeuds MPC et le PC embarqué) il est possible de conduire le CyCab avec le joystick.

A mon arrivée, j'avais deux applications SynDEx : *Robucar* originelle et l'application *Robucar* modifiée par Lionel Durand, le stagiaire précédent. Dans ce chapitre, on expliquera le contenu et le fonctionnement du *Robucar* fait par Robosoft. La reprise du travail de Lionel sera expliquée dans la section 11.6. Comme nous l'avons expliqué dans la section 3.5.2, un algorithme sous SynDEx est un ensemble de blocs interconnectés, mais au premier abord l'application *Robucar* n'est pas facile à comprendre : – les blocs appellent du code assembleur MPC555 (que ce soit pour les calculs sur la régulation, de communication ou de gestion des capteurs/actuateurs), – enfin, on ne connaît pas les plages de valeur de ces signaux.

Un des buts de mon stage était de remplacer certains blocs de régulation de la vitesse des roues qui sont en relation avec le joystick par des blocs régulant la vitesse mais à partir d'une caméra. Mais il est difficile de trouver les bons gains si on ne connaît pas le domaine des valeurs correspondant à ces blocs. J'ai donc du traduire l'application SynDEx *Robucar* en une application Scicos, logiciel de modélisation et de simulation, et par la même occasion trouver un modèle physique du CyCab. Ceci a permis de simuler la régulation *Robucar* en toute sécurité, pour moi et pour les stagiaires suivants.

7.1.2 Observation des états

Afin de savoir si la traduction de Robucar sous Scicos sera correcte, il est nécessaire de faire jouer la simulation avec des valeurs qui correspondent à la réalité. Avant de commencer le travail de traduction, j'ai du trouver un moyen d'espionner les données du CyCab.

Le premier problème rencontré fut qu'une partie de l'algorithme, appelé *root*, de l'application Robucar tourne sous RTAI. Passer par RTAI pour sauver des données dans des fichiers n'est pas ce qui y a de plus simple. Comme nous l'avons vu dans le chapitre 5, fonctionner avec RTAI

a un inconvénient majeur : celui de ne pas pouvoir accéder aux fonctions Unix de gestion des fichiers (comme *open*, *write*, *fprintf*, ...).

Une première solution consisterait à transformer le programme *root* en une application LXRT car on pourrait accéder aux fonctions *open*, *write*, *fprintf* interdites sous RTAI.

Une deuxième solution consisterait à créer un programme tournant sous LXRT en parallèle avec RTAI et transmettre les données par mémoire partagée et protéger les accès concurrents par sémaphores nommées. Ceci est assez pénalisant, car assez lourd a mettre en place et est peu esthétique.

Une troisième solution, la plus simple mais assez douteuse, consiste à espionner le fichier *kernel.log* par un script shell. Dans l'application *Robucar* sous SynDEx on ajoute des blocs appelant la fonction *rt_printk*, devant tous les blocs de type actuateurs et capteurs.

Cette fonction du noyau qui permet d'écrire des chaînes de caractères de façon équivalente à *printf* (sauf qu'elle ne permet pas l'affichage de flottants). Ces chaînes de caractères sont déroutées vers l'écran et/ou un fichier log nommé *kernel.org*, selon configuration. Ce fichier peut-être intégralement lu grâce à la commande shell *dmesg*. Le paramètre optionnel '-c' permet d'effacer ensuite le fichier *kernel.log*. Il faut s'avoir que ce fichier log est un fichier cyclique qui peut contenir, dans notre cas, environ 400 lignes. Donc *rt_printk* écrit dans ce fichier à la position courante modulo 400.

J'ai écrit un petit script shell qui recopie le contenu de ce fichier *kernel.log* dans un fichier texte de taille infinie :

```
#! /bin/bash
while [ 1 ]
do
   dmesg -c >> CyCab.log
done
```

Comme RTAI peut tourner à 1 KHz et qu'il travaille peu, on peut supposer qu'il reste suffisamment de temps pour Linux d'exécuter ce script sans trop de problème avant que les 10 ms ne s'écoulent.

En faisant attention au nombre d'appel à *rt_printk*, on ne perd pas de données par débordement des 400 lignes. Comme un fichier log récupère également des messages d'erreur du système, ils vont se mélanger aux données du CyCab (comme le démarrage, et arrêt du CyCab, ...). on ajoute un identifiant pour pouvoir les reconnaître. On ajoutera le flag *IN*= pour les données sortant des capteurs et le flag *OUT*= pour les données entrant dans les actuateurs.

Il ne reste plus qu'a lancer ce script, compiler et exécuter l'application *Robucar*. On obtient le fichier CyCab.log

7.1.3 Traitement des états

J'ai écrit un petit script shell qui va filtrer le fichier *CyCab.log*, supprimer les messages du noyau, garder les données des signaux et les séparer en 2 fichiers textes, lisibles par Scicos. Car par défaut, Scicos s'attend à lire des données au format $7(e10.3, 1x)$.

```
#! /bin/bash
function call_scilab
{
    cat <<EOF > SCISCRIPT.txt
A=fscanfMat('/afs/inria.fr/rocq/home/aoste/qquadrat/DATA.txt');
A(:,$) = (A(:,$) - A(1,$)) / 1000;
```

```
write('$1', A, '(7(e10.3,1x))');
exit
EOF

    scilab -nw -ns -nb -f SCISCRIPT.txt
    echo "Creation du fichier $1 [OK]."
    rm -fr SCISCRIPT.txt DATA.txt 2> /dev/null
}

if [ "$1" = "" ]
then
    echo "Donner le nom du fichier log CyCab"
    exit 1
fi

if [ "`which scilab 2> /dev/null`" = "" ]
then
    echo "Il faut installer Scilab"
    exit 1
fi

rm -fr IN.txt OUT.txt 2> /dev/null

cat $1 | sed -e '/^IN=/!d;s/[ \t][ \t]*/ /g' | cut -d" " -f2,4,6,8,10,12 > DATA.txt
call_scilab IN.txt

cat $1 | sed -e '/^OUT=/!d;s/[ \t][ \t]*/ /g' | cut -d" " -f2,4,6,8,10 > DATA.txt
call_scilab OUT.txt
```

La première partie consiste à séparer les signaux des capteurs des actuateurs dans 2 fichiers grâce aux commande *sed* et *cut*. Puis on transforme le format des signaux en signaux compréhensibles par Scicos grâce à un script Scilab.

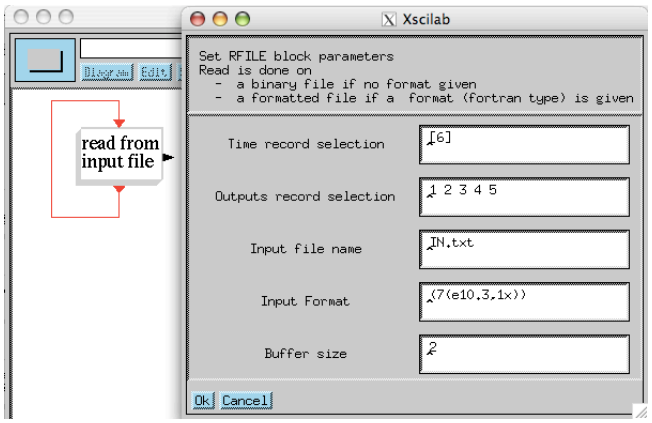


FIG. 7.1 – Le bloc *Read from input file*.

La lecture des signaux sous Scicos se fait grâce à un bloc *Read from input file* comme sur la figure (7.1). On branchera la sortie de l'horloge (en rouge) sur l'entrée de l'horloge pour le mettre en échantillonneur bloqueur. Ceci permet aux signaux de ne pas se désynchroniser au

cas où des données viendraient à manquer (pas lues par l'espion). Dans la figure (7.1), on voit une configuration possible de ce bloc :

- on indique le numéro de la colonne où est stocké le temps.
- on indique les numéros des colonnes où sont stockées les données. Ici les colonnes 1 2 3 4 5.

Il faut faire attention, car c'est bien les numéros des colonnes qu'il faut indiquer et non pas la taille du bus de données, comme on peut le retrouver dans les autres configurations des blocs Scicos.

7.2 Notions d'assembleur MPC555

La traduction de l'application Robucar SynDEx en un application Scicos, n'est pas la partie la plus compliquée car le code assembleur est assez simple à comprendre, nous expliquerons les instructions les plus importantes.

La partie où j'ai eu le plus de problème fut de comprendre pourquoi les signaux obtenus avec la simulation de Scicos ne correspondait pas aux signaux réels. Il faut rappeler que les MPC555 travaillent sur des entiers 32 bit alors que Scicos travaille avec des nombres à virgule flottantes et que des problèmes de dérive sont apparus, dus aux problèmes d'arrondis des signaux simulés. Après l'ajout à certains endroits des blocs qui gèrent les arrondis (à savoir les blocs *quantization*) à permis de simuler le travail sur des entiers, car on ne peut indiquer a scicos de travailler sur des entiers.

L'application Robucar est constitué de blocs appelant des morceaux de code assembleur ressemblant à :

```
(1)    B(lwz r30,$1);
(2)    cmpwi r30,1;
(3)    bge 0f;
(4)    mulli r30,r30,-1;
(5)    0: B(stw r30,$2)
```

Ce code mélange assembleur 555 et macro code M4. Dans le chapitre (**??**) nous avons expliqué à quoi servait le langage M4. Nous allons expliqué les instructions 555.

La première ligne `B(lwz r30,$1)` est une macro M4 qui affecte au registre r_{30} la valeur contenue dans la variable M4 `$1`.

Les lignes `(2)` et `(3)` testent si le registre r_{30} est plus petit que 1. Si c'est le cas, on saute à la ligne `(5)`, sinon on passe à la ligne `(4)`. L'instruction `cmpwi` compare le contenu d'un registre avec un mot (word ou `w`) entier (`i`). L'instruction `bge 0f` signifie : *Branch into label* `0f` *if* r_{30} *is Greater or Equal to 1.* La ligne `(4)` multiple le contenu du registre r_{30} par -1 et affecte le résultat dans le registre r_{30}. Enfin, la ligne `(5)` permet de sauver le registre r_{30} dans la variable M4 de `$2`, elle joue le rôle de `return` en langage C.

7.3 Modélisation en Scicos de la conduite manuelle

La figure 7.2) montre le résultat de la traduction de l'application Robucar en une application Scicos.

Dans la figure (7.2), on peut voir deux superblocs : – à droite : le régulateur *Robucar* SynDEx traduit en formalisme Scicos (couleur bleue) et – à gauche : un superbloc *Capteurs & Actuateurs & Plot* permettant de lire les signaux espionnés des capteurs et des actuateurs du CyCab, puis de visualiser les signaux obtenus par simulation sous la forme d'un graphique (couleur grise) et voir s'ils correspondent à ceux espionnés.

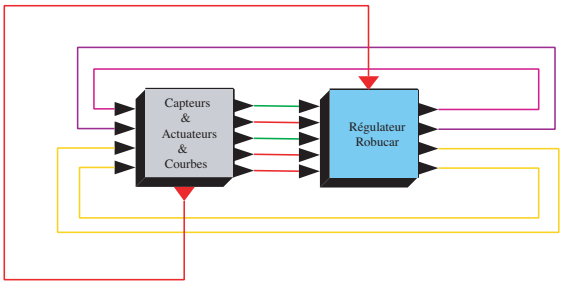


FIG. 7.2 – L'application *Robucar* sous Scicos.

Nous verrons dans la section (8) que le superbloc *Capteurs & Actuateurs & Plot* peut être remplacé par une fonction de transfert simulant le fonctionnement des capteurs et actuateurs. Rappelons (confère section (**??**)) qu'il existe un traducteur automatique de bloc Scicos vers des blocs SynDEx et qu'il suffit, dans l'IHM de Scicos de sélectionner le super bloc *Régulateur Robucar* pour obtenir une application SynDEx.

7.3.1 Superbloc *Capteurs, actuateurs, plots*

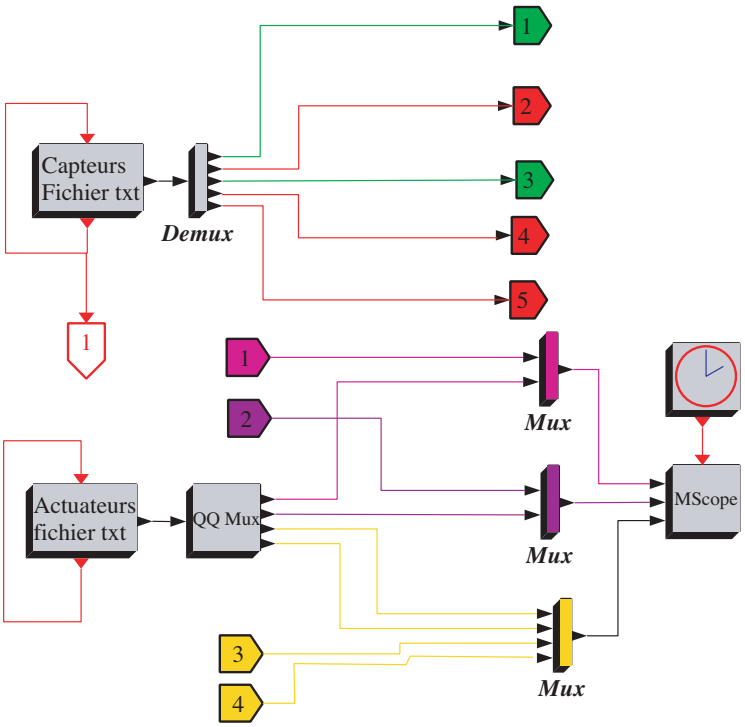


FIG. 7.3 – Le superbloc gérant l'affichage et la lecture des signaux.

Le contenu du superbloc de gauche (*Capteurs & Actuateurs & Plot*) est représenté figure (7.3). Dans la section 7.1.2, nous avons vu comment obtenir un fichier ascii contenant la valeur des signaux (ainsi que le temps) observés lors d'un test réel de déplacement sur un CyCab. Nous allons exploiter ces données grâce aux deux blocs *Capteurs* et *Actuateurs* qui sont du type *Read from input file* (figure (7.1)).

Les signaux observés sortant des capteurs sont respectivement :

1. La conversion analogique du joystick pour la direction des roues,

2. La valeur de l'encodeur absolue pour la direction des roues arrières,
3. La conversion analogique du joystick pour la vitesse des roues,
4. La valeur moyenne des quatre décodeurs en quadrature indiquant la vitesse moyenne des roues,
5. La valeur de l'encodeur absolue pour la direction des roues avants,
6. Le temps (sortie horloge).

Les signaux observés entrant dans les actuateurs sont respectivement :

1. La consigne de direction des roues avants,
2. La consigne de direction des roues arrières,
3. La tension de consignes pour la vitesse des roues avants,
4. La tension de consignes pour la vitesse des roues arrières,
5. Le temps (sortie horloge).

Tous ces signaux observés sont multiplexés avec les signaux obtenus par simulation puis sont dessinés sous de formes de graphiques (bloc *MScope* pour Multi Scopes). Un exemple de graphique obtenu est figure 7.4. En noir les signaux espionnés et en rouge et bleu, les signaux obtenus par simulation.

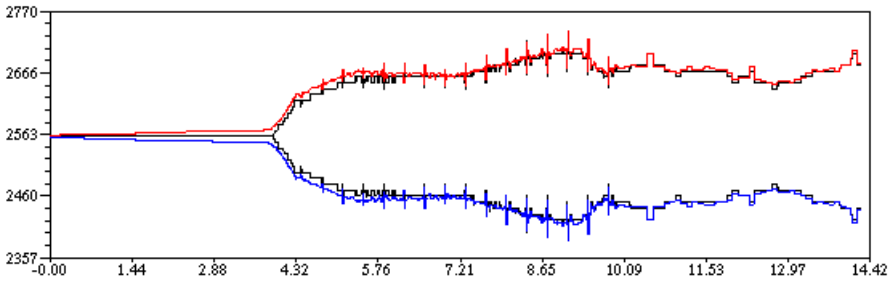


FIG. 7.4 – Signaux simulés et réels.

7.3.2 Coeur du régulateur de conduite manuelle

La figure (7.5) montre le superbloc *Régulateur Robucar* SynDEx traduit en formalisme Scicos (superbloc droit de la figure (7.2)).

On voit en entrée les signaux des 5 capteurs précédemment cités dans la figure (7.4). On voit également 5 superblocs colorés en 3 couleurs différentes (vert, jaune et marron). En effet certains blocs accomplissent presque le même travail, d'où leur regroupement par couleur :

- En vert : des filtres sur les signaux analogiques du joystick,
- En jaune : la régulation sur la vitesse des 4 roues,
- En rouge : la régulation sur la direction des 4 roues.

Nous allons détailler chacun de ces superblocs.

7.3.3 Superbloc *Adoucissement Joystick motricité*

La figure (7.6) est le contenu du superbloc *Adoucissement Joystick Avant/Arrière* montre le filtre sur le signal du joystick pour la vitesse des roues (motricité).

La plage de valeur du signal du joystick est comprise entre +115 et +775. On commence par recentrer ce signal en 0 en lui soustrayant 445. C'est le rôle des blocs verts à gauche de l'image.

Ensuite, on filtre le bruit du signal grâce aux blocs roses :

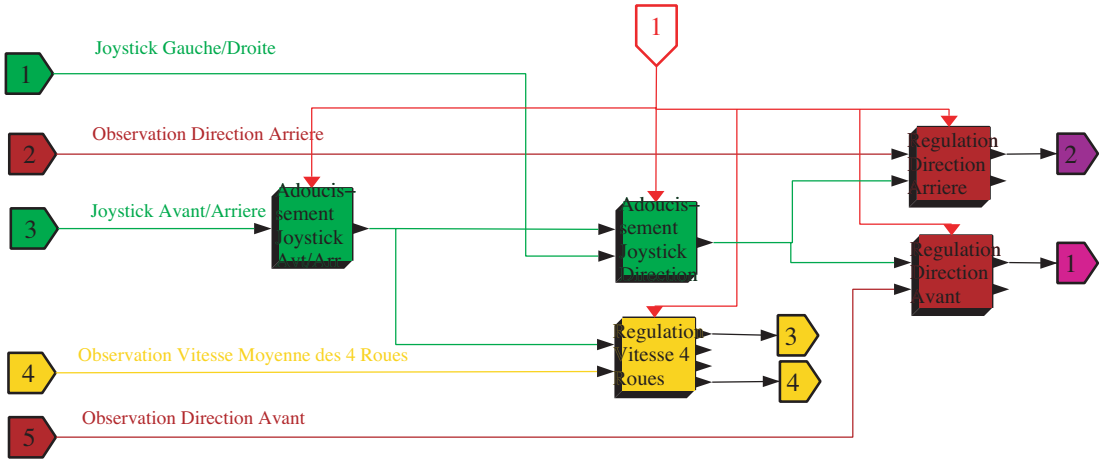


FIG. 7.5 – *Robucar* main

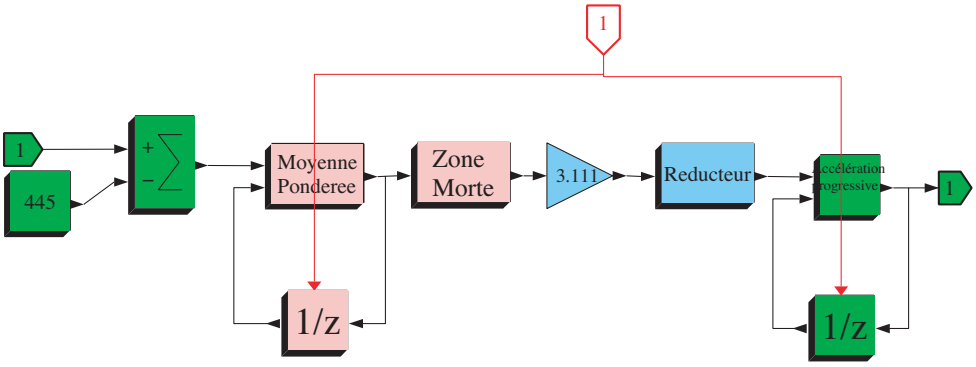


FIG. 7.6 – Filtrage du joystick gérant la motricité.

- d'abord par une moyenne pondérée des 32 dernières valeurs grâce à la fonction dynamique $y = f(u, y)$, où u est l'entrée, y la sortie et f la fonction qui calcule $\frac{u+31y}{32}$.
- Ensuite, avec le bloc *Zone Morte*, si le signal est compris entre les valeurs -30 et +30, il est réduit à 0 ; sinon tronqué d'une valeur 30. Les extremums du signal sont donc -300 et +300.

Le signal est multiplié par une constante 3.111 (bloc bleu) qui n'est d'autre qu'une conversion d'une vitesse en une autre vitesse. La documentation du CyCab nous dit que la vitesse maximum d'un moteur est de 7330 mm/s, que le diamètre d'une roue est de 400 mm, que le réducteur moteur a un ratio de 8 et que la révolution d'une roue fait 2000 impulsions après décodage en quadrature et que la période de lecture est 10 ms. Donc la constante est :

$$\frac{7330 \times 8 \times 2000 \times 0.01}{400\pi \times 300} = 3.11$$

Le deuxième bloc bleu *Réducteur* divise uniquement le signal négatif par 4. Je ne suis pas sure mais c'est à cause d'un autre réducteur moteur pour la marche arrière.

Ensuite, vient le bloc vert *Accélération progressive* il protège le moteur des changements de vitesse trop grands, ce qui empêche l'apparition de pics de courant importants dans les moteurs, risquant d'endommager l'électronique ou les moteurs.

La fonction $f(u_1)$ est de la forme $f(u_1) = (u_1 + 30)^2/933$ et est bornée par deux fonctions :

- $u_2 - 7 < f(u1) < u_2 + 1.5$ quand $u1 >= 0$.
- $u_2 + 7 > f(u1) > u_2 - 1.5$ quand $u1 < 0$.

7.3.4 Superbloc *Régulation de la vitesse des roues.*

La figure (7.7) montre le contenu du bloc jaune *Régulation Vitesse 4 Roues* de la figure (7.5).

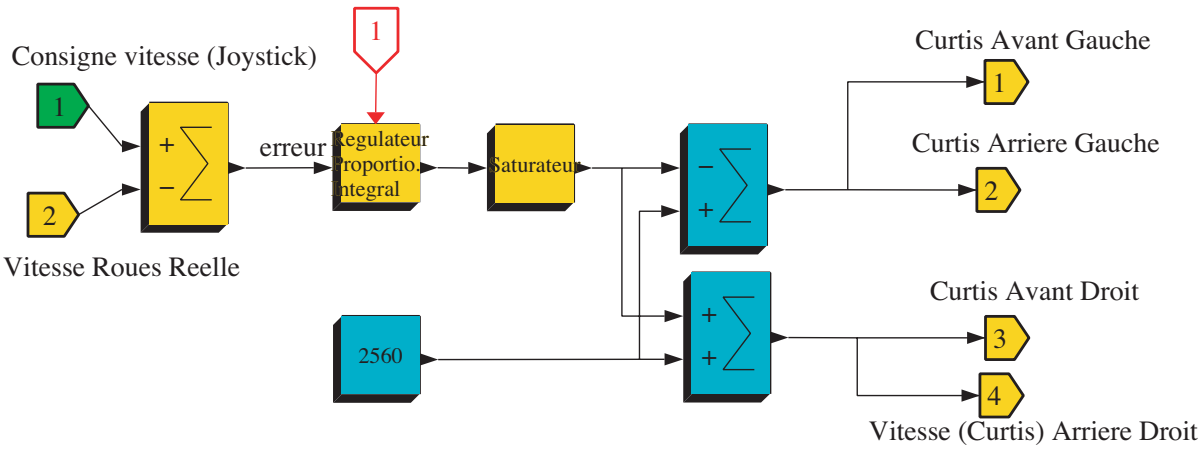


FIG. 7.7 – Régulation de la vitesse des roues.

Il permet la régulation de la vitesse des quatre roues en fonctionnant de l'observation de la vitesse réelle des roues et de la consigne de vitesse qui est la sortie du superbloc de la figure (7.6).

Les blocs en jaunes montrent l'erreur (entre la consigne de vitesse fournie par le joystick et l'observation de la vitesse réelle des roues) entre dans un régulateur *Proprotionel–Intégral* puis est saturée dans la plage de valeur $[-400; +400]$. Les blocs en bleu font que le signal s'additionne ou se soustrait avec la constante 2560.

Comme nous l'avons vu dans la section 4.5.2, ce signal correspond à une tension a fournir au Curtis. 400 correspond à 0.4V et 2560 correspondant à 2.5V, la valeur neutre où les roues ne tournent pas. Comme les quatre roues doivent tourner toutes dans le même sens, et que les moteurs à gauche du CyCab sont inversés par rapport à ceux du côté droit, on doit changer le signe (valeur < 2560).

7.3.5 Filtrer le signal du joystick de direction

La figure 7.8 montre le deuxième bloc vert filtrant le signal du jystick.

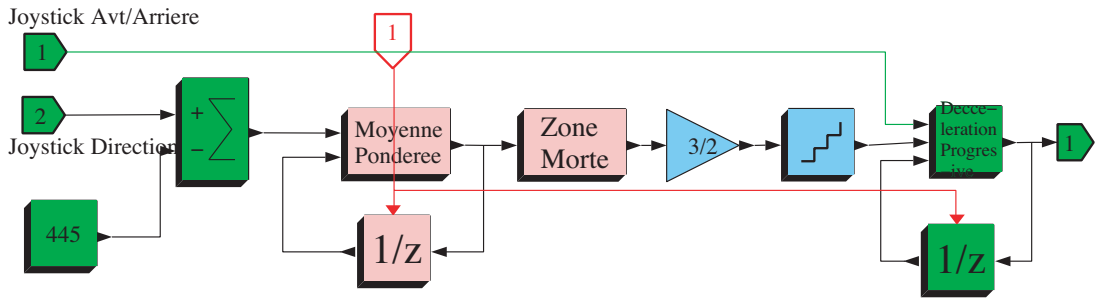


FIG. 7.8 – Filtrage du joystick gérant la direction.

Ce signal indique une consigne de direction pour les roues. Son fonctionnement est identique au superbloc 7.6 : même filtre pondéré, même élimination du bruit par troncature. La différence

vient du bloc vert *Décélération Progressive* qui permet de contrôler la direction des roues en fonction de leur vitesse (pour éviter de tourner trop brusquement à grande vitesse).

La fonction est de la forme $4(1 + \frac{400}{40+u_1})$ et comme pour le bloc *Accélération Progressive*, on empêche le signal de sortir entre deux bandes qui sont des fonctions dynamiques.

7.3.6 Superbloc *Adoucissement Joystick de direction*

Enfin, les deux derniers blocs en rouge sur la figure 7.5 permettent de réguler la direction des roues. L'un des deux est montré figure (7.9).

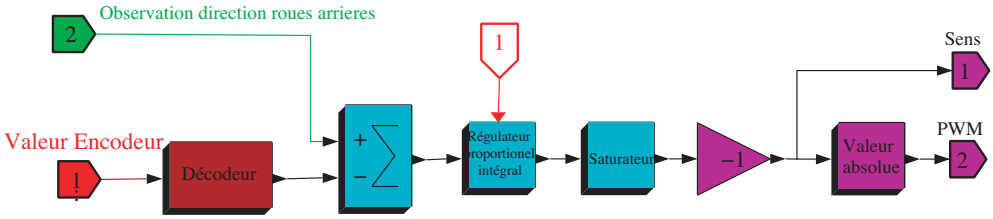


FIG. 7.9 – Régulation de la direction des roues arrières.

Les blocs bleus permettent de calculer l'erreur entre la consigne de direction des roues et de leur observation. On applique un régulateur proportionnel intégral à l'erreur et on sature la valeur entre -900 et +900. Enfin on envoie au vérin le sens et une tension permettant de faire tourner les roues.

7.4 Note importante sur l'application de conduite manuelle

Il faut savoir que l'application *Robucar* pour fonctionner correctement en situation réelle, doit avoir au moins un bloc tournant sur RTAI. En effet, RTAI sert de timer de période 10 ms. Il permet de ralentir les deux noeuds du CyCab pour la régulation. Supprimer cet unique bloc aura pour conséquence une mauvaise régulation.

Une autre remarque importante, est que l'on constate qu'au démarage de la régulation, RTAI tourne à 3 ms au lieu de 10 ms puis revient à 10 ms. Je ne sais pas d'où vient ce phénomène.

Chapitre 8

Modélisation du process CyCab

8.1 Principe des moindres carrés

Maintenant que nous avons traduit le régulateur du CyCab de SynDEx en Scicos, il est nécessaire de se donner un modèle du CyCab pour pouvoir tester le régulateur. Dans notre cas, on aimerait connaître, la fonction de transfert qui nous donne la vitesse des roues en fonction des sorties du régulateur (entrées du Curtis) c.a.d. la fonction de transfert Curtis – Quadrature Encoder. Grâce aux données espionnées lors d'un test sur le CyCab, on ajuste dans un premier un système lineaire temps discret MIMO d'ordre un par la méthode des moindres carrés.

On suppose donc que le modèle du CyCab est le système, $x_{n+1} = ax_n + bu_n$ et $y_n = x_n$ où les entrées sorties y_n et u_n sont connus et les paramètres a et b à ajuster de façon à minimiser l'erreur entre le modèle et les données observées. Soit :

$$\min_{a,b} \mathcal{L}(a,b) = \min_{a,b} \sum_{n=0}^{N} (x_{n+1} - ax_n - bu_n)^2$$

Donc on résout le système : $\frac{\delta L}{\delta a} = 0$ et $\frac{\delta L}{\delta b} = 0$ et on trouve :

$$\frac{\delta L}{\delta a} = -\sum x_n x_{n+1} + b \sum u_n x_n + a \sum x_n^2 = 0$$

$$\frac{\delta L}{\delta b} = -\sum u_n x_{n+1} + a \sum u_n x_n + b \sum u_n^2 = 0$$

Soit sous forme matricielle :

$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum x_n^2 & \sum u_n x_n \\ \sum u_n x_n & \sum u_n^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum x_n x_{n+1} \\ \sum u_n x_{n+1} \end{bmatrix}$$

8.2 Programme Scilab

Il ne reste plus qu'à écrire un petit script Scilab pour obtenir le modèle de notre CyCab.

```
function [W]=Modelise(file_IN, col_in, file_OUT, col_out)
  M   = fscanfMat(file_IN);
  u   = M(:, col_in);

  M   = fscanfMat(file_OUT);
  x   = M(:, col_out);
```

```
  N   = min(size(u, 1), size(x, 1));
  i   = [1:1:N-1];

  a11 = sum(x(i)^2);
  a12 = sum(u(i) .* x(i));
  a21 = a12;
  a22 = sum(u(i)^2);
  A   = [a11, a12; a21, a22];

  b11 = sum(x(i) .* x(i + 1));
  b21 = sum(u(i) .* x(i + 1));
  B   = [b11; b21];

  W   = A \ B;
endfunction

//Modele vitesse roues droites
Roue_droite = Modelise("IN.txt", 4, "OUT.txt", 3)

//Modele vitesse roues gauches
Roue_gauche = Modelise("IN.txt", 4, "OUT.txt", 4)
```

Une version alternative d'écriture de la fonction `Modelise` plus courte est possible. Elle utilise le produit scalaire :

```
function [W]=Modelise(x, u)
  xx  = x(1:$-1)
  uu  = u(1:$-1)
  A   = [uu * uu', xx * uu'; xx * uu', uu * uu'];
  B   = [[x,0] * [0,x]'; [0,u] * [x,0]'];
  W   = A \ B;
endfunction
```

8.3 Résultat

Nous en déduisons la fonction de transfert $F(z) = b/(1/z - a)$ Curtis–Encodeur. Nous pouvons, ensuite, modifier notre application en remplaçant le bloc `Capteurs & Actutateurs & Plot` de la figure (7.2) par le bloc qui représente la fonction de transfert $F(z)$ comme le montre la figure (8.1).

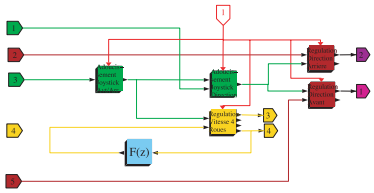


FIG. 8.1 – *Robucar* main avec la fonction de transfert Curtis-décodeur.

Quatrième partie

Conduite automatique du CyCab

Chapitre 9

Notions de traitement d'image

Avertissement : afin de réduire le poids de ce document PDF, les images présentes ont du être comprimées.

9.1 Formats de couleurs utilisés

Nous allons présenter trois formats de couleur qui sont utilisés pour le traitement de l'image concernant la détection de CyCab : – l'image, donnée par la caméra, au format YUV422 qui est une compression du format YUV, – le format YUV et enfin, – le format RGB. Nous allons décrire ces trois formats.

9.1.1 Codage RGB

Le format RGB (RVB en français) attribue un octet par composante de couleur rouge, verte ou bleue par pixel. Ces trois composantes primaires additives servent dans les écrans d'ordinateur et de télévision pour reconstituer toutes les couleurs sur l'image visualisée. Par exemple, dans un ordinateur la valeur d'une composante est un entier compris entre 0 et 255 inclus. L'absence de couleur correspond au noir (valeur 0) alors que le blanc correspond à la fusion des 3 couleurs (valeur 255). Comme on représente une couleur sur les trois composantes, on obtient plus de 16 millions de possibilités théoriques de couleurs différentes, l'œil humain n'en voyant que 2 millions.

9.1.2 Codage YUV

D'après l'article [17] on sait que l'oeil humain est plus sensible au signal de luminance qu'aux signaux de couleurs. Comme le format RGB n'en tire pas partie, on utilisera alors le format YUV qui a l'avantage de séparer une composante moyenne des signaux RGB, appelée luminance (Y à savoir le noir et blanc), des deux signaux de couleur (U et V qui correspondent au rouge et bleu) appelés chromances. La couleur verte se retrouve alors par combinaison entre Y, U et V. Figure (9.1) montre une photo et la décomposition de ses composantes Y, U, V. Les images proviennent de [16].

Du point de vu historique, le codage YUV provient de la transmission de la vidéo analogique pour la télévision. Le Conseil Supérieur de l'Audiovisuel n'ayant accordé à la télévision couleur qu'une bande passante de 8 MHz (au lieu de 32 MHz provenant des trois bandes de couleurs et de la bande de la luminescence), des ingénieurs ont dû trouver une méthode pour comprimer les données des images RGB pour la télévision hertzienne. Le système est donc compatible avec les téléviseurs noir et blanc.

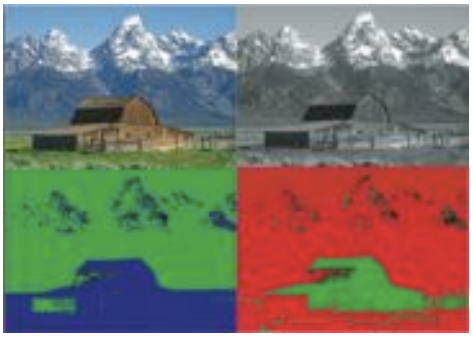


FIG. 9.1 – Une image et ses composantes Y (haut, droit), U (bas gauche) et V (bas droit).

De nombreuses équations, très proches les unes des autres, permettent de passer du codage YUV au codage RGB.

$$R = Y - 0.0009267(U - 128) + 1.4016868(V - 128)$$
$$G = Y - 0.3436954(U - 128) - 0.7141690(V - 128)$$
$$B = Y + 1.7721604(U - 128) + 0.0009902(V - 128)$$

9.1.3 Codage YUV422

Ce format est celui fournie par notre camera FireWire, il permet de compresser le débit vidéo tout en ne détérioraient pas trop l'information contenue. Les chiffres 4, 2 et 2 indiquent le nombre d'échantillons de luminance Y et de chrominance U et V pour coder l'information d'une image YUV de taille 4×4. Cela permet d'avoir 2 octets par pixel au lieu de 3.

Par exemple, si on reçoit le flux d'entrée YUV422 :

$$Y_1\ Y_2\ ...\ Y_{16}\ U_1\ U_2\ ...\ U_4\ V_1\ V_2\ ...\ V_4$$

On en déduit l'image 4×4 YUV décompressée :

$$Y_1U_1V_1\ \ Y_2U_1V_1\ \ Y_3U_2V_2\ \ Y_4U_2V_2$$
$$Y_5U_1V_1\ \ Y_6U_1V_1\ \ Y_7U_2V_2\ \ Y_8U_2V_2$$
$$Y_9U_3V_3\ \ Y_{10}U_3V_3\ \ Y_{11}U_4V_4\ \ Y_{12}U_4V_4$$
$$Y_{13}U_3V_3\ \ Y_{14}U_3V_3\ \ Y_{15}U_4V_4\ \ Y_{16}U_4V_4$$

D'autres formats YUV compressés comme les formats YUV420, YUV411 dont le débit total de ce dernier est de 1.5 celui de la luminance. Remarquons que le format YUV est aussi nommé YUV444.

9.2 Opérations de base sur les pixels

Comme nous l'avons vu dans la section précédente, une image est un ensemble de composants chromatiques consécutifs (3 pour RGB et YUV ; 1 pour BW). En regroupant ces composants on forme des pixels qui sont ordonnancés par leur position (x, y).

On appelle taille d'une image le couple (L, l), le nombre de pixel en longueur L et en largeur l qui la constituent. Une représentation mémoire d'une image est un vecteur de $3lL$ pixels pour

des images YUV et RGB et un vecteur de lL pixel pour les images blanches et noires. Chaque pixel est codé sur un octet.

On liera les images d'abord de gauche à droite puis de haut en bas. On notera $p(x, y)$ un pixel de cordonnées $(x, y) \in (l, L)$. Par exemple en format RGB, $p(x, y)$ retournera le triplet $(R(x, y), G(x, y), B(x, y))$ dont :

$$p(x, y) = \begin{bmatrix} R(x, y) \\ G(x, y) \\ B(x, y) \end{bmatrix} = \begin{bmatrix} 3(yl + x) \\ 3(yl + x) + 1 \\ 3(yl + x) + 2 \end{bmatrix}$$

Il existe des opérations de base qui manipulent composante par composante les pixels d'une image. Ils ont la forme suivant :

$$p(x, y) = p_1(x, y) \oplus p_2(x, y)$$

$$p(x, y) = \begin{bmatrix} R(x, y) \\ G(x, y) \\ B(x, y) \end{bmatrix} = \begin{bmatrix} \max(0, \min(255, R_1(x, y) \oplus R_2(x, y)) \\ \max(0, \min(255, G_1(x, y) \oplus G_2(x, y)) \\ \max(0, \min(255, B_1(x, y) \oplus B_2(x, y)) \end{bmatrix}$$

où p_1 et p_2 sont des pixels de même coordonnées x, y mais provenant de deux images. p est le pixel résultant de l'opération. La fonction $\oplus$ peut-être : – une opération arithmétique de base comme, l'addition, la soustraction (on veillera à ce que la valeur de p reste comprise entre 0 et 255 en saturant la valeur du résultat) ; – des opérations booléenne comme l'opération *non*, *ou*, *et*, *ou exclusif*.

9.2.1 Masque

Dans notre cas, concernant le CyCab, seule l'opération *et* binaire (opérateur & en langage C) nous intéresse, car il va nous permettre de masquer certaines zone de l'image. Le site [15] donne des exemples de ces opérations, dont voici un exemple avec le *et*.

FIG. 9.2 – Image 1.

FIG. 9.3 – Image 2.

FIG. 9.4 – Et binaire.

9.2.2 Transformation d'une image en niveaux de gris

La détection des contours se fait sur une image noir et blanc par un produit de convolution. La première étape est alors de transformer l'image de type YUV ou RGB en provenance de la caméra en une image en niveaux de gris.

Transformation d'une image RGB en une image en niveaux de gris

Cette transformation est un simple moyenne pondéré des trois composantes de couleurs. La formule générale est donc la suivante, pour un pixel $p(x, y)$:

$$p(x, y) = \frac{R(x, y) + G(x, y) + B(x, y)}{3}$$

Transformation d'une image YUV en une image en niveaux de gris

Comme précisé dans la section 9.1, la composante Y correspond au niveau de gris. Voici cependant la formule qui permet d'obtenir la composante Y à partir des composantes RGB :

$$Y = 0,299R + 0,587G + 0,114B$$

Les coefficients ne sont pas les mêmes (on s'attendait à avoir $0,333$). Cette différence ne va pas nous gêner puisqu'elle correspond à une réalité physique (le vert est plus lumineux que le rouge qui est à son tour plus lumineux que le bleu) et puisque l'algorithme qui va être utilisé n'est pas sensible à ces coefficients linéaires.

On prendra alors dans notre programme la composante Y pour niveaux de gris, tout simplement.

9.3 Filtrage matriciel

Cette technique consiste à appliquer une matrice M de taille $I \times J$ (généralement de taille 3×3 à chaque pixel $p(x, y)$ de l'image à transformer de taille $m \times n$. Le calcul est le suivant :

$$p(x, y) = \sum_{i=1}^{I} \sum_{j=1}^{J} M(i, j)\, p(y + i, x + j) \tag{9.1}$$

avec : $x \in [1 \,..\, L - i - 1]$ et $y \in [1 \,..\, l - j - 1]$.

On veillera à ce que la valeur du pixel p reste comprise entre 0 et 255 en saturant la valeur. Nous allons voir dans les section futures qu'elles sont les formes que peut avoir M et quels sont les résultats que l'on obtient.

9.3.1 Filtres passe-bas

Nous ne nous attarderons pas sur les filtres passe-bas dont le but est de réduire les parasites (bruits de mesure). Ils agissent par moyenne sur un voisinage et suppriment donc les détails.

9.3.2 Filtre passe-haut de Sobel

Les filtres passe-haut nous serons utiles pour le CyCab car ils ont pour but d'augmenter le contraste et de mettre en évidence les contours. Les contours sont une discontinuité locale de l'intensité lumineuse. Les techniques permettant de détecter un contour sont basées sur l'utilisation de gradients.

La détection de contours du CyCab utilise principalement le filtre de Sobel dont la matrice est :

$$M_h = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Puisque ce filtre est basées sur l'utilisation de gradients, les coefficients 1, 2 et 1 ne sont que l'approximation de la dérivée. Ce filtre permet de déterminer les contours horizontaux. On obtient un filtre qui détermine les contours verticaux par rotation de $\pi/2$ de la matrice M_h. On obtient alors :

$$M_v = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Le résultat du produit matriciel (formule (9.1)) de M par le pixel $p(x, y)$ contiendra des valeurs négatives et positives ce qui correspondent aux signes du gradient. Comme nous travaillons sur des octets non signé, il faut recentrer la plage de valeur à 128. La visualisation graphique du résultat nous donne une image grisée (ce qui correspond à 128) avec des parties claires ou foncées.

FIG. 9.5 – Image RGB.

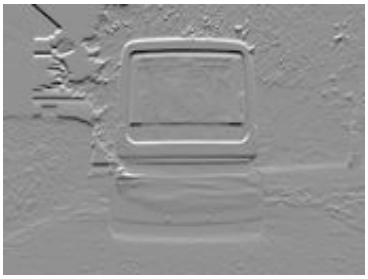


FIG. 9.6 – Sobel hor. signé.

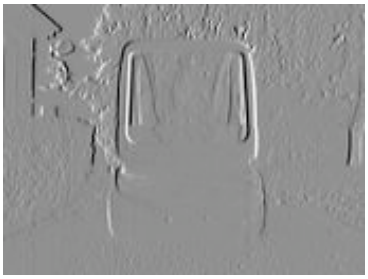


FIG. 9.7 – Sobel ver. signé.

Par rapport à la formule 9.1, on prendra la valeur absolue de la valeur du résultat du pixel $p(x, y)$, puis on divisera le résultat par un coefficient k. Cette division permet de supprimer le bruit (haute fréquence).

Les images (9.11), (9.12) et (9.10) donne un exemple de filtrage de Sobel sur les contours. Il est intéressant de noter que le logiciel libre *The Gimp* contient une bibliothèque de détection de contours (Sobel (non) signé, Laplace, ...).

FIG. 9.8 – Image RGB.

FIG. 9.9 – Sobel horizontal.

FIG. 9.10 – Sobel vertical.

9.3.3 Histogrammes horizontal et vertical

L'histogramme est, dans notre cas, un vecteur H de hauteur le nombre de lignes de l'image traitée. Chaque élément (i) du vecteur correspond à la moyenne des pixels de la ligne i. Voici la formule générale pour une image :

$$h(y) = \frac{\sum_{x=1}^{L} p(x, y)}{L}$$

$$v(x) = \frac{\sum_{y=1}^{l} p(x, y)}{l}$$

Les lignes horizontales les plus grandes de l'histogramme horizontal correspondent le plus souvent aux contours du CyCab.

FIG. 9.11 – Sobel Horizontal.

FIG. 9.12 – Histo horizontal.

Malheureusement ce n'est pas toujours le cas : une haie d'arbres bien taillée ou la façade d'un bâtiment peuvent créer une ligne horizontale supplémentaire.

FIG. 9.13 – Haie coupée.

FIG. 9.14 – Pied du batiment.

9.4 Problématique sur le sélection des raies

Dans ce chapitre, nous avons vu quelques briques de bases concernant le traitement de l'image. Nous avons vu qu'un filtre de Sobel non signé suivit d'un histogramme horizontale nous permettait d'obtenir des lignes horizontales (raies) du contours du CyCab.

Lors de mes tests avec le traitement de l'image que j'ai hérité, j'ai remarqué trois choses :

- L'environnement pouvait perturber la détection de CyCab, en créant des raies supplémentaires suffisamment grandes dans l'histogramme pour perturber l'estimation de la distance qui sépare les deux CyCab.
- L'algorithme de détection re-découvrait les positions des raies à chaque nouvelle image, sans utiliser la détection du passé.
- L'algorithme calcule la distance qui sépare les deux CyCab avec une méthode basée sur le calcul de rapport invariant avec les positions des raies. Si l'idée d'utiliser les rapports invariants est intéressant du point de vue théorique, il est difficile à mettre au point un programme, car il utilise un certain nombre de raies pouvant aller de 2 à 5.

Dans le prochain chapitre, nous décrirons une méthode améliorée de détection de CyCab.

Chapitre 10

Régulation de la distance

10.1 Régulation de la distance

Soit le diagramme Scicos (10.1) représentant la simulation du régulateur de distance d'un CyCab suiveur et de la simplification de sa dynamique.

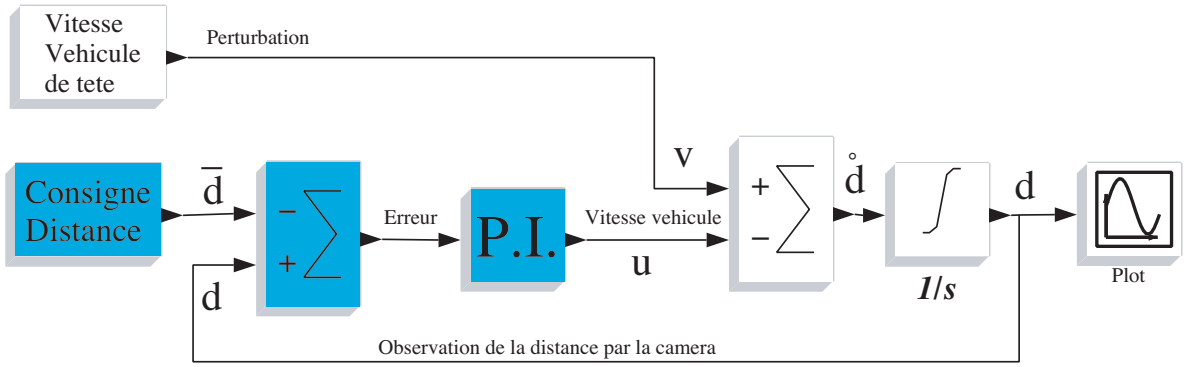


FIG. 10.1 – Régulation de la distance

On nommera :

- $\bar{d}$, La consigne de distance que notre CyCab suiveur doit maintenir par rapport au CyCab qui le précède (par exemple 5 mètres).
- d, la distance observée par la caméra du CyCab suiveur.
- v, La vitesse du CyCab de tête, qui est a priori inconnue. Nous la voyons comme une perturbation à minimiser.
- u, La vitesse de notre CyCab.

Le but de notre régulateur est de maintenir d à la valeur $\bar{d}$ et ce quelque soit la vitesse du CyCab de tête.

Le bloc intégrale (ou $1/s$) est une simplification de la dynamique d'un CyCab. Le bloc P.I. est un régulateur de type proportionnel intégral. Nous allons expliquer pourquoi ce choix est le meilleur.

10.1.1 Cas d'un régulateur proportionnel

Si on prenait un régulateur proportionnel (au lieu d'un proportionnel intégral), nous aurons une erreur asymptotique. On explique ici pourquoi. D'après le schéma (10.1), on a :

$$\dot{d} = v - u \ . \tag{10.1}$$

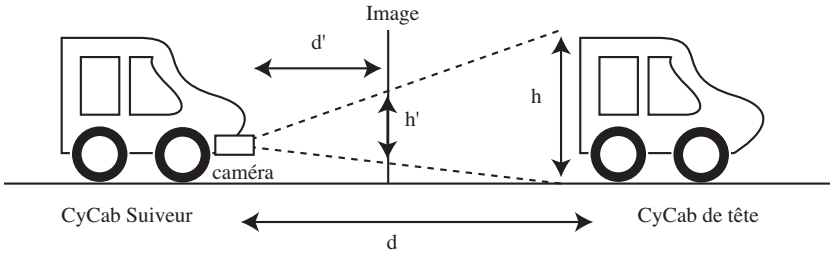


FIG. 10.2 – Suivi.

Or, si on considère que notre régulateur est de type proportionnel (de gain k_p), alors u s'écrit :

$$u = k_p(d - \bar{d}) \ . \tag{10.2}$$

De (10.1) et de (10.2), en déduit :

$$\dot{d} = v + k_p(\bar{d} - d) \ . \tag{10.3}$$

Le système est stable des que $k_p > 0$. L'erreur asymptotique est obtenu en calculant le point d'équilibre ($\dot{d} = 0$) soit $\epsilon = d - \bar{d} = v/k_p$. On aimerait que ϵ soit le plus petit possible quelque soit v inconnu donc il faut prendre k_p positif et grand. Mais, il reste un biais ϵ que l'on aimerait annuler. C'est le terme intégral qui permet de rendre l'erreur assymptotique à nulle.

10.1.2 Cas d'un régulateur proportionnel et intégral

Si notre régulateur est un proportionel intégral, l'équation (10.2) devient :

$$\dot{d} = v + k_p(\bar{d} - d) + k_i \int_0^t (\bar{d} - d)dt \ .$$

Si $k_p > 0$ et $k_i > 0$ le système est stable. En dérivant, on obtient :

$$\ddot{d} = -k_p\dot{d} + k_i(\bar{d} - d)$$

En régime stationnaire, on a $\ddot{d} = 0$ et $\dot{d} = 0$ ce qui implique $d = \bar{d}$. Nous avons reussi, du point de vue théorique, à maintenir une distance fixe, sans connaître la vitesse du CyCab précédent.

La figure (10.3) montre résultat de la simulation. En bas, en rouge nous voyons la vitesse de perturbation de la voiture de tête. En haut, en noir, la consigne de distance (3 mètres) et en vert la distance séparant les deux CyCab. Seul, les blocs bleus seront générés pour SynDEx et seront présents dans le nouveau régulateur Robucar.

10.2 Principe d'obtention de la distance

Nous avons montré dans la section précédente, que nous pouvons stabiliser autour d'une distance fixe entre deux CyCab, quelque soit la vitesse du CyCab de tête, à condition de bien régler les gains du régulateur P.I. Comme il n'est pas facile d'estimer la distance d à partir d'une image, nous allons utiliser l'observation de la hauteur h du CyCab sur l'image, comme observation. En effet, il est très aisé de compter le nombre de pixel sur une image plutôt que de calculer la distance correspondante.

Le régulateur de hauteur du CyCab dans l'image reste le même que la figure (10.2), à la différence qu'il faut changer les signes de la rétroaction. En effet la hauteur du Cycab est

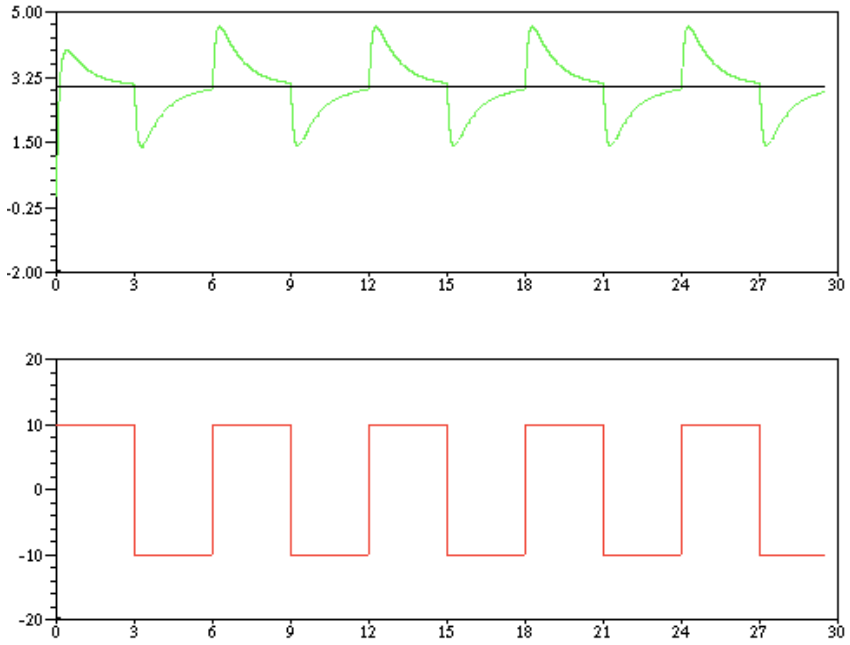


FIG. 10.3 – Résultat de la simulation.

inversement proportionnel à la distance.

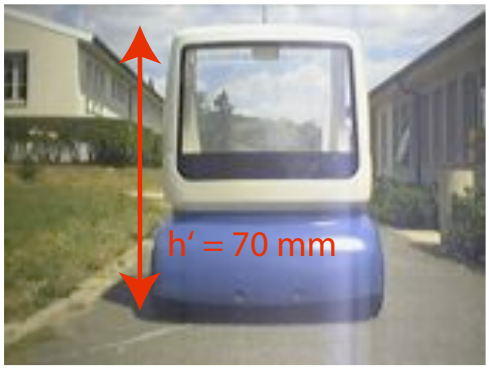


FIG. 10.4 – d est petit, h est grand.

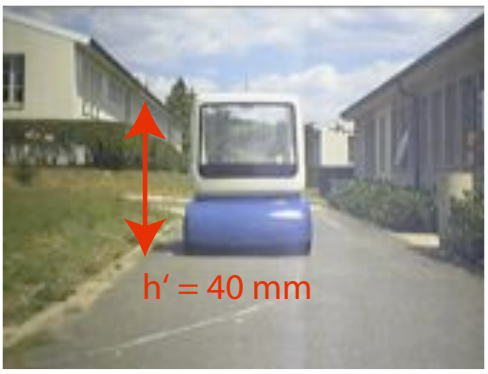


FIG. 10.5 – d est grand, h est petit.

Pour déterminer la hauteur de consigne on place le Cycab à la distance de consigne et on regarde sur l'image la différence de hauteur de deux lignes horizontales. Le traitement d'image aura pour but de retrouver automatiquement et rapidement ces deux lignes horizontales sur les images fournies par la caméra.

Pou pouvoir acquérir le plus rapidement possible la position de ces deux lignes on propose dans la suite un algorithme de suivi de ces lignes basé sur le filtrage de Kalman.

Dans la suite lorsque nous parlerons de distance ce sera en fait de la distance entre les deux lignes horizontales de l'image et non pas de la distance entre les deux véhicules.

Chapitre 11

Estimation de la distance

11.1 Distinguer les lignes

1. Nous transformons l'image (11.1) en une image BW et nous appliquons le filtre de Sobel signé horizontal (11.2) et filtre de Sobel horizontal (11.3) que nous stockons dans deux buffers d'images différents.
2. A partir de l'image de Sobel signé (11.2), nous faisons apparaître le signe du gradient dans une image rouge et bleue. La force du gradient ne nous intéresse pas, on veut distinguer les signes. Le signe négatif est en rouge ($p(x, y) <= 128$) et le signe positif en bleu ($p(x, y) > 128$). Nous obtenons l'image (11.3).
3. Nous appliquons un masque entre l'image (11.3) et (11.2). Le masque consiste à appliquer un *et binaire* entre les composantes de couleurs de chaque pixel comme expliqué dans le chapitre 9. Nous obtenons l'image finale (11.5).

Fig. 11.1 – Image RGB initiale.

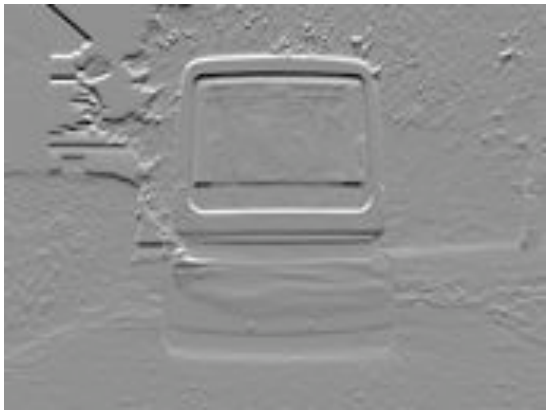


Fig. 11.2 – Filtre de Sobel signé horizontal.

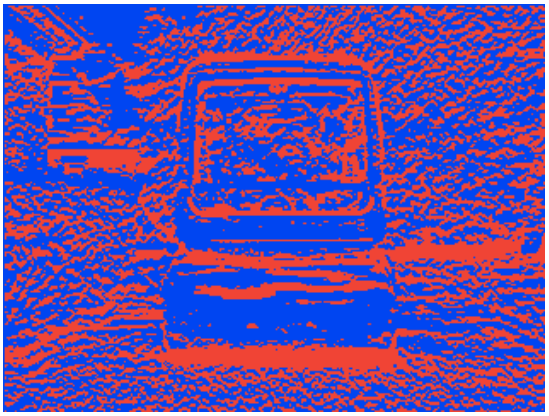


Fig. 11.3 – Le signe du filtre de Sobel.

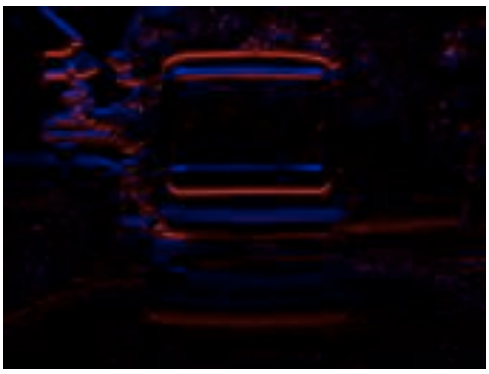


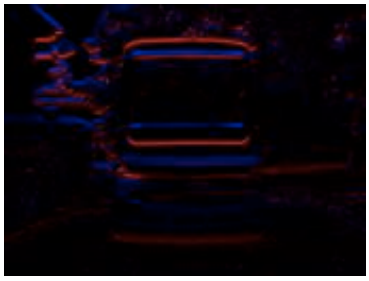
FIG. 11.4 – Filtre de Sobel horizontal. FIG. 11.5 – Résultat final.

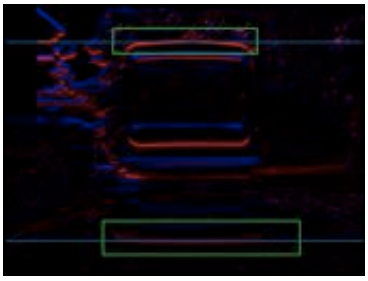
Sur l'image (11.5), nous remarquons que les lignes rouges et bleues sont alternées, ce qui permettra de bien distinguer les lignes entre elles.

11.2 Sélectionner les lignes sur une image statique

Au lieu de calculer un histogramme des pixels blancs, nous allons calculer l'histogramme sur les pixels rouges et bleus dans certaines portions de l'image, délimitées par des lassos.

Nous allons encadrer à la main les zones où les lignes sont intéressantes. La figure 11.7 nous montre la sélection de deux lignes rouges (ce choix est arbitraire on aurait pu choisir une couleur bleue et rouge). En dehors du lasso l'environnement est éliminé, à l'intérieur du lasso, nous calculons l'histogramme (figure(11.8)). Ceci permet de trouver plus facilement le contours du CyCab.





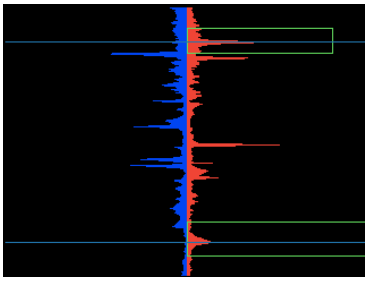


FIG. 11.6 – Sobel couleur. FIG. 11.7 – lasso. FIG. 11.8 – Histogramme.

11.3 Sélectionner les lignes sur une vidéo

La section 11.2, nous a permis de sélectionner à la main les lignes qui nous intéressent sur une image fixe. On le fait une seule fois : c'est la phase d'initialisation. Les deux lignes sélectionnées nous serviront à construire l'observateur de distance entre les deux véhicules.

11.4 Suivi des raies sélectionnées

Sur le flux vidéo la voiture de tête va arbitrairement accélérer ou freiner, il faut donc suivre le déplacement des lignes sélectionnées. Pour cela on va utiliser un filtre de Kalman estimant au mieux la position de la raie dans l'image. Pour cela on a besoin d'un modèle du déplacement de la raie dans l'image. Supposons que l'on veuille suivre une raie horizontale Appelons v la vitesse

de déplacement vertical de cette raie dans l'image. On suppose que la différence de vitesse entre les deux véhicules est localement constante et donc le modèle est

$$\dot{v} = 0.$$

Si on appelle x la position verticale de la raie on a donc

$$\dot{x} = v.$$

On suppose faire une observation y bruité de cette position soit

$$y = x + w.$$

où w désigne un bruit blanc de variance N. On en déduit le modèle en temps discret suivant (où h dénote le pas pas en temps) :

$$\begin{cases} v_{n+1} = v_n, \\ x_{n+1} = x_n + hv_n, \\ y_n = x_n + w_n. \end{cases} \tag{11.1}$$

L'observateur optimal donné par le filtre Kalman est alors :

$$\begin{cases} \hat{v}_{n+1} = \hat{v}_n + k_1\{y_{n+1} - (\hat{x}_n + h\hat{v}_n)\}, \\ \hat{x}_{n+1} = \hat{x}_n + h\hat{v}_n + k_2\{y_{n+1} - (\hat{x}_n + h\hat{v}_n)\} \end{cases} \tag{11.2}$$

où k_1 k_2 sont donnés par la formule suivante :

$$K = \begin{bmatrix} k_1 \\ k_2 \end{bmatrix}, \quad \text{avec } K = \begin{bmatrix} \Sigma_{12} \\ \Sigma_{22} \end{bmatrix} (\Sigma_{22} + N)^{-1}$$

et Σ est solution de l'équation de Riccatti stationnaire suivante :

$$\Sigma = \begin{bmatrix} 1 & 0 \\ h & 1 \end{bmatrix} \Sigma^+ \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix}, \qquad \Sigma^+ = (I - K \begin{bmatrix} 0 & 1 \end{bmatrix})\Sigma\,.$$

En re-écrivant la formule 11.4 sous forme matricielle, on vérifie que les valeurs propres de A sont comprises dans le cercle unitaire. Le calcul de K est obtenu par la fonction Scilab `Riccati`. Nous obtenons $k_1 = 0.33$ et $k_2 = 0.76$.

11.5 Résultat de la prédiction de la position des lignes

Les figures (11.9) et (11.10), montre que les résultats obtenus sont bons arrive à maintenir le cadre dans la bonne portion d'image, comme nous le montre les images suivantes (11.9) et (11.10) tirées d'une expérience avec un suivi de déplacement de lignes horizontales. Le Kalman est plus robuste avec des lignes de différentes couleurs car, sur le haut du CyCab, il y a trop de contours trop proches. Kalman peut se tromper dans la sélection d'une ligne.

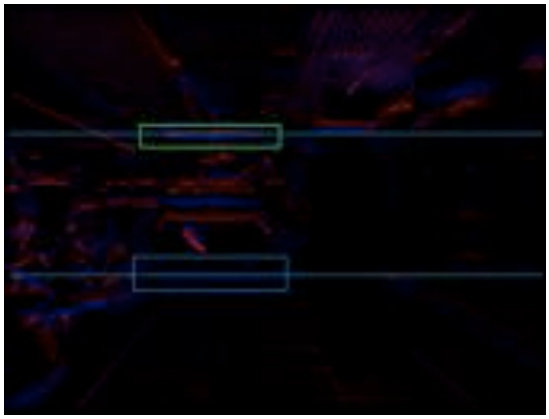


FIG. 11.9 – Début de l'expérience.

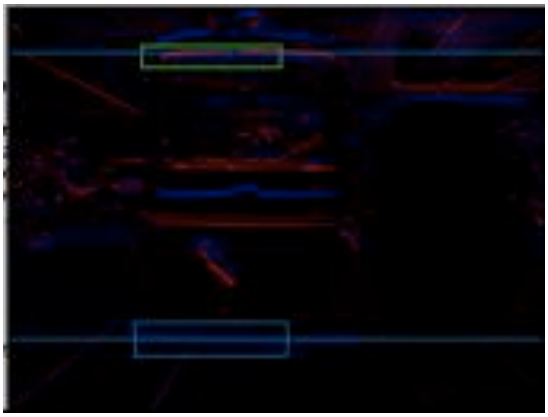


FIG. 11.10 – Fin de l'expérience.

11.6 Communication entre LXRT et RTAI sur le logiciel de conduite automatique

Maintenant que nous savons comment obtenir, la hauteur h du CyCab. Il faut pouvoir, depuis une tâche RTAI, l'envoyer aux MPC555 afin qu'ils puissent l'exploiter dans la régulation. Or comme nous l'avons signalé, Il n'est pas possible de faire le traitement d'image dans une tâche temps réel RTAI. Il doit être fait dans une tâche LXRT.

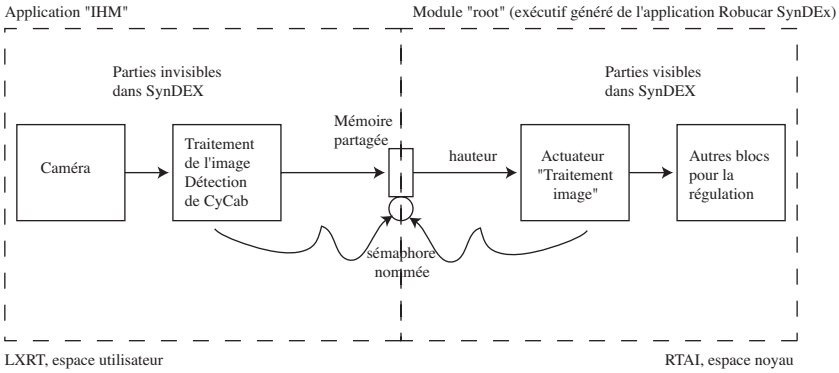


FIG. 11.11 – Communication entre l'exécutif Robucar et IHM.

Dans la figure (11.11), le rectangle en pointillé à gauche, présente la tâche LXRT qui permet de gèrer la caméra FireWire et la détection de CyCab. Cette tâche n'est pas visible depuis l'application SynDEx de conduite automatique. Le résultat du traitement de l'image, c.a.d. h est stocké dans une mémoire partagée.

Le rectangle en pointillé à droite, présente la tâche RTAI. Les blocs qu'elle contient sont les blocs que l'ont voit dans l'application SynDEx de conduite automatique (confère figure (11.12)).

Le sémaphore nommé joue le rôle de mutex pour gérer les accès concurrents de lecture et d'écriture dans la mémoire partagée. Le terme "nommé" indique que le module SynDEx et le programme LXRT que le sémaphore soit commun.

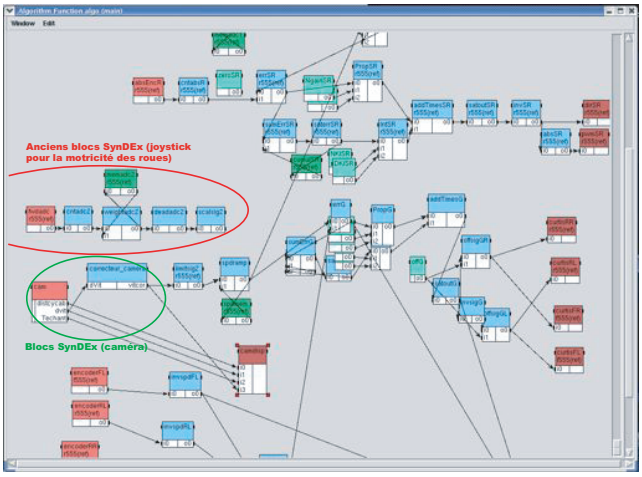


FIG. 11.12 – L'application Robucar modifiée pour la conduite automatique.

11.7 Récapitulatif du fonctionnement de l'application de conduite automatique

La figure (11.13) montre le diagramme *flow chart* récapitulant le fonctionnement de l'algorithme du suivi longitudinal de CyCab. En italique et en pointillé, la partie de l'algorithme pour le suivi latéral. Cette partie n'a pas encore été réalisée mais peut facilement être mise en place en "copiant/collant" le code pour le suivi longitudinal et où la seule différence consiste à utiliser un filtre de Sobel vertical au lieu d'un filtre de Sobel horizontal.

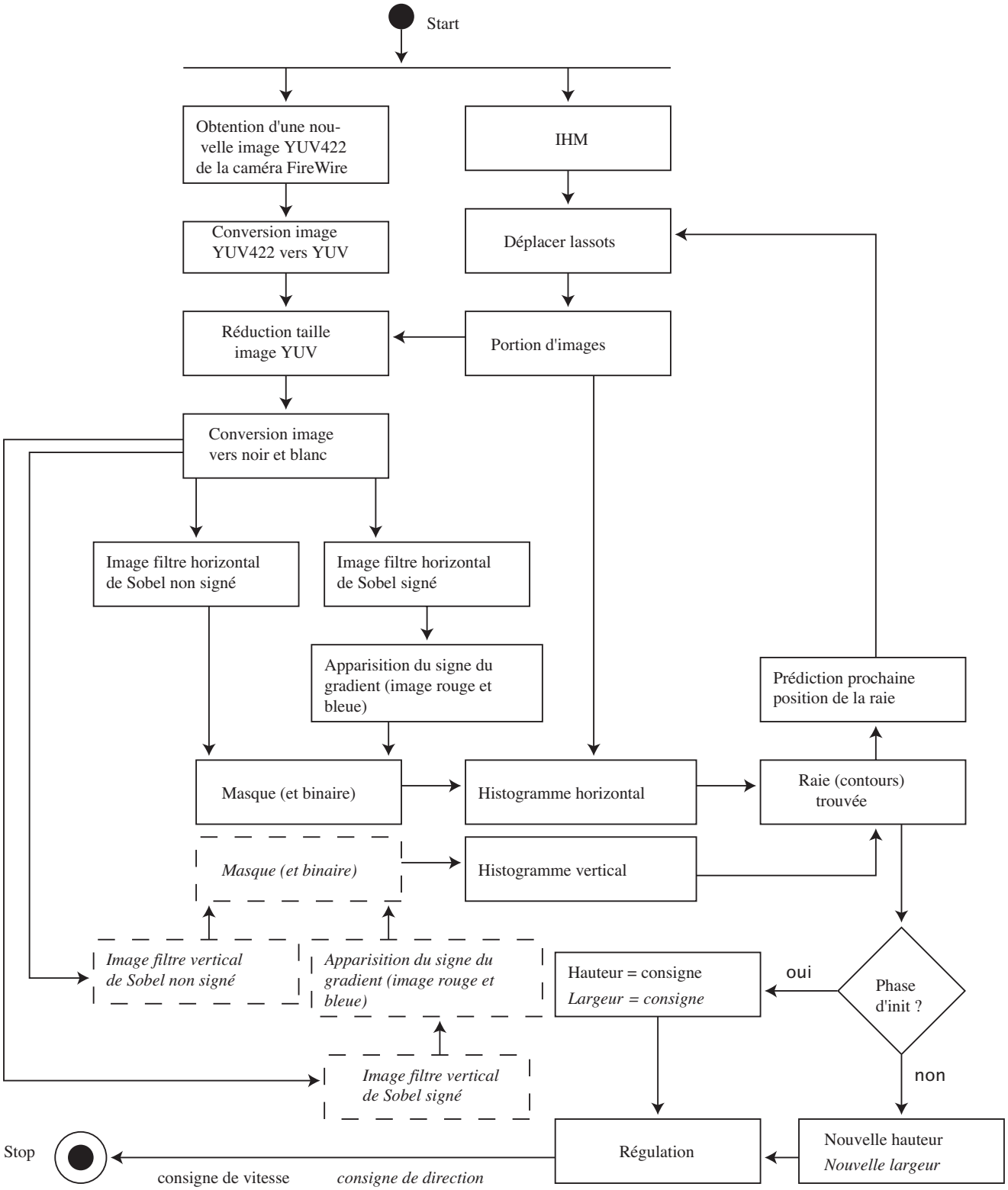


FIG. 11.13 – Diagramme State Flow de l'algorithme du traitement de l'image.

Chapitre 12

Conclusion

Malgré les nombreuses pannes du matériel (noeuds MPC, barrette mémoire, mon ordinateur portable), ce stage a été très bénéfique sur plusieurs plans. J'ai eu la responsabilité de réaliser une application temps réel complète de niveau industriel. J'ai donc du abordé de nombreux domaines : modélisation, traitement d'image automatique, informatique distribué, système, programmation dans plusieurs langage, électronique etc. Ce stage m'a aussi familiarisé avec les difficultés de la réalisation d'un système matériel industriel qui nécessite la coopération de nombreux acteurs externes au projet.

Au moment de l'écriture de ce rapport la partie simulation du système complet et l'expérimentation du traitement d'image est achevé et fonctionne de façon satisfaisante. La campagne d'expérimentation du régulateur débute et devrait pouvoir être terminée d'ici la fin du stage fin juillet si d'autres pannes matérielles ne retardent pas l'atteinte de cet objectif.

La seule régulation longitudinale sera sûrement insuffisante pour le suivi effectif des véhicules. Le suivi 2D peut être réalisé très rapidement en adaptant la méthodologie utilisée ici (il suffit de suivre des lignes verticales liés au Cycab et de les maintenir au centre de l'image) mais il demande encore un peu de travail pour être mis en oeuvre effectivement.

Cinquième partie

Annexes

Chapitre 13

Installer Linux Debian et RTAI

13.1 Installer RTAI

Vu que le hardware du PC embarqué a été mis à jour pendant ce stage, j'en ai également profité pour mettre à jour l'OS. Afin de faciliter la tâche des futurs stagiaires, je décris ici, comment installer une distribution Debian, compiler un nouveau noyau et installer RTAI.

13.2 Etape 1 : installer Debian

L'installation de Debian

J'ai opté pour l'installation d'une Debian 4.0 NetInst release 0, car Debian est réputée pour sa fiabilité. Elle contient un noyau 2.6.18. Son installation ne pose pas de problème, mais il faut lancer l'installation en mode *expert* (non testé en mode *expertgui*) car en mode *normal* Debian choisit arbitrairement un réseau connecté à un serveur DHCP alors qu'à l'INRIA, il impératif de posséder une adresse IPv4 statique.

Vu que j'ai choisît une distribution légère (150 Mo), l'installation des logiciels se fait par le réseau (d'où le nom de Debian NetInst). Il faut laisser Debian se connecter à un serveur http pour installer tous les packages utiles, sinon on n'a pas serveur X.

Faire reconnaître la puce graphique intégrée de la carte mère

Debian reconnaît très bien la carte graphique NVidia, mais pas la puce graphique Intel de la carte mère car il faut choisir l'une ou l'autre dans le BIOS (touche *delete* lors du boot du PC, puis choisir le menu XX et (dés)activer XX).

Une fois Debian la Debian installée, on édite le fichier */etc/X11/xorg.conf* et on ajoute les lignes suivantes :

```
Section "Device"
    Identifier   "Intel 82865G"
    Driver        "i810"
EndSection
```

On cherche la `Section "Screen"` et on ajoute la ligne suivante à côté de la ligne `Device NVidia`.

```
#    Device    "Intel 82865G"
```

Le `#` sert à commenter l'utilisation de ce device, car on préférera l'utilisation de la carte graphique NVidia.

Finir d'installer les logiciels manquants

Une fois Linux démarré, on se logue en tant que utilisateur *aoste* et on finit d'installer les packages manquants grâce à l'utilitaire *Synaptic*. On installe les packages suivants : demon ssh, *gcc-3.4* (on créera un lien symbolique gcc avec la commande *sudo ln -s gcc-3.4 /usr/sbin/gcc*), *autotools*, *auto-make*, *ncurses*, *emacs*, *libraw1394-dev*, *libdld394-dev*, *glade2-dev*, et pour tester comment ce comportement de RTAI au stress du CPU) : *lxdoom*.

13.3 Etape 2 : compiler un nouveau noyau

Sur le site de RTAI [19] on télécharge la version 3.4 de RTAI dans le répertoire */usr/src*, on la décompresse et on créer un lien symbolique *rtai* par les commandes :

```
$ cd /usr/src/
$ tar jxvf rtai-3.4.tar.bz2
$ ln -s rtai-3.4 rtai
```

La version actuelle du noyau Debian est une 2.6.18 or il ne semble pas exister de patch RTAI pour 2.6.18 bien qu'il existe un patch pour 2.6.17 ou pour 2.6.19 ! On télécharge sur le site [21] la version 2.6.16-52.

```
$ cd /usr/src/
$ tar jxvf linux-2.6.16-52.tar.bz2
$ ln -s linux-2.6.16-52 linux
```

La dernière étape est utile. On lance le configure graphique (il faut avoir installer la libncurses), je n'ai pas tester les autres alternatives (gtk, ...)

```
$ make menuconfig
```

Après avoir sélectionner les options du noyaux, comme on l'explique dans la section suivante (13.7), on compile les sources et on installe l'image du noyau et les modules :

```
$ make
$ sudo make install
$ sudo make modules_install
```

Je n'ai rencontré aucun problème de compilation. Donc, si tout va bien on retrouve, dans le répertoire */boot/*, la nouvelle image du noyau *vmlinux-2.6.16-52-rtai*. C'est tout à fait normal qu'il n'y a pas de fichier *initrd-2.6.16-52-rtai*. On va updater *grub* par :

```
$ update-grub
```

Ou bien manuellement par :

```
$ emacs /boot/grub/menu.lst
```

13.4 Etape 3 : redémarrer avec le nouveau noyau installé

On redémarre la machine et au démarrage du *grub*, on choisit le nouveau noyau 2.6.16-52-rtai. Si, le démarrage échoue, c'est que les options du noyaux n'ont pas été mises comme il faut, notamment les deux importantes : – par défaut le système de fichier ext3 est mis en tant que module, je l'ai inclus entièrement dans le noyau, et – ne pas mettre la gestion des bus SPI en module. Normalement, après ces deux vérifications, Linux se lance sans erreur. Sinon, il faut activer/désactiver d'autres options du *configure* et recompiler le noyau.

13.5 Etape 4 : compiler RTAI

On revient au dossier de rtai, on lance un configure graphique (un peu comme pour le kernel), on compile et on installe RTAI.

```
$ cd /usr/src/rtai
$ make menuconfig
$ export PATH=PATH:/usr/realtime/bin
$ make
$ sudo make install
$ sudo make modules_install
```

On sauvegardera la variable *PATH* dans le *.bashrc*.

13.6 Etape 5 : tester RTAI au stress

Maintenant, on doit vérifier que RTAI fonctionne correctement, en utilisant le test `latency` fournis avec les sources RTAI.

```
$ cd /usr/realtime/testsuite/kern/latency/
$ ./run
```

Il faut également lancer des applications gourmandes en CPU et qui vont stresser le système, par exemple une application 3D. Ces applications "latency killers" causent des retards non prédictibles et sont incompatibles avec le concept de temps réel. Il faut vérifier que la colonne `overuns` reste toujours à 0, sinon il faut enlever les modules susceptibles de causer des retards par la commande `insmod` et/ou supprimer des options du noyau puis recompiler le noyau et RTAI (faire un `make distclean` dans les sources RTAI).

13.7 Etape 2prim : choisir les options pour la compilation du noyau

Je conseille d'utiliser mon fichier `.config` pour activer les bonnes options. J'indique ici, les options que j'utilise pour mon kernel 2.6.16.52 patché avec RTAI 3.4.

- *Code maturity level options* on sélectionne "Prompt for development ..."
- *General setup* on met le nom `-rtai` à "Local version".
- *Loadable module support* on sélectionne "Enable module support", "Module unloading" et "Automatic kernel module loading". Déselectionner "Module versioning support".
- *Processor type and features* Mettre "Subarchitecture Type" à `PC-Compatible` et `Processor family` à Pentium 4. Selectionner "Preemption Model (Preemptible kernel (Low-Latency Desktop))". Mettre "High Memory Support" à 4GB. Déselectionner "Use register arguments (EXPERIMENTAL)", "kexec system call (EXPERIMENTAL)", "kernel crash dumps (EXPERIMENTAL)", "Symetric multi-processing support" et "Local APIC support on uniprocessors". Mettre "Timer frequency" le plus élevé possible à savoir 1 Khz.
- *Power Management options* Enlever "ACPI Support". Déselectionner "APM BIOS Support" et "CPU Frequency scaling".
- *Bus options* Activer "PCI support" mais supprimer "PCI Express support" et "ISA support" car le PC embarqué n'en possède pas ou n'en utilise pas. Supprimer "PCI hotplug Support".

- *Networking* enlever "Amateur Radio support" et tout ce qui concerne le "Bluetooth" et "802.11".
- *Device Drivers*
  - *Generic driver options* garder les options par défaut.
  - *Memory Technology Devices (MTD)* à enlever.
  - *Parallel port support* Peu être activé ou désactivé.
  - *Plug and Play support* garder par défaut.
  - Enlever *SCSI device support*, *SPI support*, *I2C support*, *USB support*, *Sound* et *Multi-device support (RAID and LVM)*. Activer *IEEE 1394 support*.
- *File systems* Activer "Second extended fs support" et "Ext3 extended attributes".
- *Instrumentation Support* désactiver tout.
- Garder les autres options par défaut.

13.8 Etape 4prim : choisir les options pour la compilation de RTAI

Le choix des options pour RTAI est plus simple que le choix des options du noyau.

- *Menu General* Mettre "Installation directory" à `/usr/realtime` et "Kernel source directory" à `/usr/src/linux`.
- Mettre *Number of CPUs* à 1.

Table des figures

Bibliographie

Logiciels

[1] La page principale de SynDEx : http ://www-rocq.inria.fr/syndex/

[2] La page principale de Scilab : http ://www-rocq.inria.fr/syndex/

[3] La page principale de Scicos : http ://www.scicos.org/

[4] Il existe 20 articles de prise en main avec les logiciels Scilan/Scicos. Ils sont téléchargeables sur `http ://www.saphir-control.fr/articles/Articles_lm.htm`

Automatique et Ordonnancement

[5] Thierry Grandpierre, Christophe Lavarenne, Yves Sorel, *Modèle d'exécutif distribué temps réel SynDEx*, INRIA, 1998.

[6] Andreas Ermedahl, *Schedulability Analysis Assignment*, 2004.

[7] Stephen L. Campbell, Jean-Philippe Chancelier and Ramine Nikoukhah, *Modeling and Simulation in Scilab/Scicos*, Springer, 2005.

[8] Pierre Faure et Michel Depeyrot, *Eléments d'automatique*, Dunod, 1974.

[9] Karl Johan Åström *Control System Design* ME155A.

[10] *Discrete PID controller* ATMEL, `http ://www.atmel.com/dyn/resources/prod_documents/doc`

[11] La page principale de Karl Johan Åström : http ://www.control.lth.se/∼kja/

[12] Philipe Baptiste, Emmanuel Neron, Francois Soud *Modèles et algorithmes en ordonnancement*, Ellipses 2004.

Traitement de l'image

[13] `http ://www-sop.inria.fr/icare/personnel/malis/software/ESMapplications.html` Quelques applications basées sur du traitement de l'image, dont un algorithme de suivi de CyCab.

[14] `http ://perso.enst.fr/ maitre/BETI/` Bibliothèque d'Exemples de Traitement des Images.

[15] `http ://iup3gmi.univ-fcomte.fr/IG/TraitementImages/TraitementImages.htm` Site d'introduction sur le traitement d'image.

[16] Certaines images servant d'exemple dans ce document, proviennent du site `http ://www.wikipedia.org/`

Bus IEEE 1394

[17] *Acquisition vidéo au moyen d'une caméra IEEE 1394*, Renaud Dardenne et Marc Van Droogenbroeck. Linux Magazine N 69, Février 2005, page 54 – 59.

[18] http ://www.linux1394.org/

Linux, RTAI, kernel

[19] Le site officiel de RTAI https ://www.rtai.org/

[20] Site fournissant de la documentation et des exemples de programmes RTAI et LXRT http ://www.captain.at/rtai.php

[21] Où télécharger un noyau Linux http ://www.kernel.org/pub/linux/kernel/v2.6/

[22] *RTAI-Lab tutorial : Scilab, Comedi, and real-time control*, Roberto Bucher, Simone Mannori, Thomas Netter. https ://www.rtai.org/RTAILAB/RTAI-Lab-tutorial.pdf

[23] Cours de RTAI http ://www.courseforge.org/courses/fr/rtai1/

[24] RTAI 3.4 User manual https ://www.rtai.org/index.php ?module=documents &JAS_DocumentManager_op=downloadFile&JAS_File_id=46

[25] Le livre *Linux Device Drivers 3rd edition*, Jonathan Corbet, Alessandro Rubini et Greg Kroah-Hartman, O'Reilly 2005.

Matériel

[26] Les cartes du PC embarqué sont commandées chez le fournisseur Aplus http ://www.aplus-sa.com/fr/produit.asp.

[27] Datasheet et manuel du contrôleur Curtis PMC 1227 http ://curtisinst.com/index.cfm ?fuseaction=cDatasheets.dspListDS&CatID=1