Projet Icare/RobotVis
INRIA Sophia
2004 Route des Lucioles, BP 93
06902 Sophia Antipolis Cedex, France

ArGiMoGe : Nouvelle version

Soraya ARIAS

20 décembre 1996

1 Introduction

Ce rapport présente le travail qui a été effectué consécutivement à mon stage de DEA portant sur l'étude et le développement d'un prototype de vision robotique temps réel. Il constitue le point de départ pour mon travail de thèse.

Base de ce travail : l'application développée lors du stage de DEA consiste à contrôler un système de robot avec caméra, appelé ARGÈS. Pour ce faire, l'utilisateur pilote le système ARGÈS et lance des traitements de vision tel que la détection de mouvement avec suivi de cibles. Ces actions perceptives sont contrôlées par un automate ESTÉREL à travers une interface Homme/Machine.

Extensions apportées :

1. Au niveau de la fonctionnalité de l'application :
   Ajout d'un module de suivi de mouvement basé sur un asservissement linéaire.
2. Introduction du système ORCCAD :
   - d'un point de vue conceptuel : au niveau de la spécification et de la validation des entités mises en oeuvre pour l'application;

- d'un point de vue implémentation : réutilisation d'une version du système ORCCAD qui pour l'occasion a été portée sous POSIX pour répondre à un besoin d'efficacité et de portabilité.

3. Utilisation du langage de mission MAESTRO :

- au niveau de la manipulation des différentes actions spécifiées pour programmer l'application;
- validation comportementale de l'application;
- validation expérimentale des primitives du langage.

Présentation de l'application : Le prototype qui a été mis-en-oeuvre s'inspire du travail effectué par [1] sur une tête artificielle binoculaire et dont le but était de :

- faire de l'observation passive d'une scène avec stabilisation d'image ;
- faire de la détection de cible.

De manière plus effective que cette étude faite par [1], le prototype que l'on a conçu gère un système à une caméra, le système *Argès*, et fournit à l'utilisateur une interface graphique à partir de laquelle il peut piloter le système de caméra, lancer des requêtes telles que la détection de mouvement avec suivi dans la scène perçue à travers la caméra.
Ce pilotage est sous le contrôle d'un automate qui a été généré grâce au langage de programmation de mission MAESTRO et il est géré de façon temps réel grâce au système ORCCAD, qui a été porté sous POSIX dans le cadre de cette étude.

Plan du document : Ce document se décompose de la manière suivante : dans une première partie on explicitera les méthodes de spécification qui ont été utilisées lors de l'élaboration de l'application. Ensuite on envisagera la partie de validation de ces spécifications avec les outils disponibles pour faire les vérifications. Enfin, on présentera les détails de l'implémentation de l'application.

2 Spécifications

Le prototype construit durant le stage de DEA a montré qu'il était nécessaire :

- de contraindre la programmation si on souhaite une gestion moins lourde des programmes
- de pouvoir automatiser l'élaboration et valider plus facilement des applications.

C'est pour cela qu'il nous est apparu intéressant d'utiliser une méthodologie qui a déjà fait ses preuves dans le domaine des applications robotiques : le système ORCCAD[2].

Du point de vue conceptuel, ce système considère principalement les deux niveaux d'abstraction suivants :

1. le niveau fonctionnel.
   Ce niveau considère des **tâches robot** ou TRS.
   Ces TRS sont la représentation de la spécification complète et paramétrée :

   - d'un ensemble d'algorithmes qui servent à expliciter une action (par exemple : une loi de commande, un traitement d'image) qui est invariant pendant la durée de l'exécution de l'action.
   - d'un comportement logique déterminé par un ensemble de signaux qui peuvent survenir avant, pendant et après l'exécution de la TR.

2. le niveau contrôle.
   Ce niveau considère des **procédures robot** ou PRS.
   Ces PRS sont la représentation de la spécification d'un arrangement au niveau logique et temporel de plusieurs TRS, dans le but de réaliser un objectif global, ou plus généralement une application.

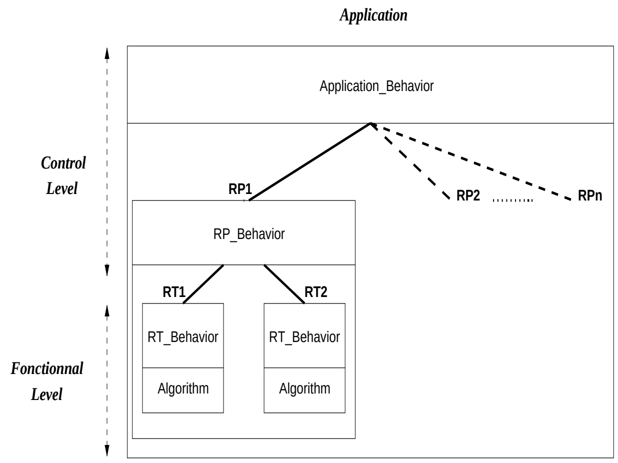

Figure 1: Hiérarchie des Spécifications dans ORCCAD

Pour chacun des niveaux ci-dessus, nous allons détailler la structuration en TRs (entités algorithmiques et entités de contrôles associées), et en PRs de notre application dérivée du travail de DEA.

2.1 Spécification des modules de vision

Dans ce paragraphe, on se place au niveau fonctionnel et on va s'intéresser à l'aspect algorithmique formant une partie de la définition des TRs.
En particulier, on va étudier les entités algorithmiques qui participent à la réalisation de l'application et qui concernent les modules de vision suivants :

- détection de mouvement.
- suivi de mouvement.

Détection de mouvement Ce module a été emprunté à la librairie `AcVis` [3] (vision active) du projet ROBOTVIS.
Il permet une détection rapide de mouvement se basant sur :

- différence entre deux images ;
- lissage de l'image différence utilisant un filtre normalisé exponentiel de la forme $H(u) = \frac{1}{2}(1-a)a^{|u|}$ avec $a = (1/10)^{1/(SmoothWindow+1)}$
- détermination du point de plus grand mouvement, i.e. dont la différence est la plus grande ;
- détermination de la zone de mouvement autour de ce point.

Ce module fournit comme paramètres de retour :

- le centre (x_0, y_0) et
- la taille de la zone en mouvement.

Suivi de mouvement Après détection d'une zone en mouvement dans l'image, on va centrer l'image sur cette zone. Par conséquent, ce module permet de modifier l'orientation de la caméra, uniquement le pan et le tilt, de manière à ce que la zone en mouvement se retrouve au centre de l'image observée par la caméra.
Ici, le degré de translation de la caméra n'est pas considéré, car sur le système de caméra envisagé, la translation prend un temps trop important pour garantir le respect des contraintes temps réel de l'application.

Le changement d'orientation est obtenu par une méthode d'asservissement linéaire déterminée de la manière suivante :

- Equation à résoudre :

$$X_{center} = x_0 + K_p p_0$$
$$Y_{center} = y_0 + K_t t_0$$

$$\text{avec} \begin{cases} X_{center}, Y_{center} & : \text{coordonnées du centre de l'image} \\ x_0, y_0 & : \text{coordonnées du point à centrer} \\ p_0, t_0 & : \text{valeurs courantes du pan et du tilt} \\ K_p, K_t & : \text{facteurs d'échelle du pan et du tilt} \end{cases}$$

- Pour déterminer les facteurs d'échelle :
  - on a repéré plusieurs points sur l'image à traiter ;
  - on a modifié le pan et le tilt de façon à centrer ces points sur l'image ;
  - on a ensuite fait une moyenne des différentes valeurs de K_p et K_t pour obtenir les valeurs des facteurs d'échelle que l'on va employer par la suite.

Ce travail a été fait à la main, une fois pour toute. Mais il devrait être à refaire dans le cas où le zoom variait - ceci pose le problème de l'automatisation de cette tâche.
Le choix de cette méthode, assez élémentaire et approximatif, s'explique par le fait que l'on veut limiter le plus possible la complexité du suivi de mouvement pour garantir le respect des contraintes temps réel.

Discussion sur la spécification des entités algorithmiques Les entités algorithmiques envisagées sont vues sans utilisation d'aucun formalisme précis pour les spécifier.
Cependant, le projet ROBOTVIS cherche à développer des objets appelés : IMPS, qui vont permettre de formaliser la spécification des algorithmes de vision. Ces IMPS pourraient être ensuite inclus dans le formalisme des TRS.

Sous leur forme inerte, un IMP est un objet du type :

$$InObj = \{attr_1 = val_1, attr_2 = val_2, ...\}$$

C'est une liste d'attributs avec valeurs, qui correspond au formalisme des frames [4].
Cette syntaxe permet de définir facilement des listes, des tableaux, des ensembles, ou encore des valeurs numériques.

Mais surtout, cet objet peut être interfacé avec MAPLE à travers des primitives définies dans `http://www.inria.fr/robotvis/personnel/vthierry/mascotte/main.html` et par conséquent bénéficier des facilités d'évaluation de MAPLE.

Dans notre application, les IMPS peuvent être vus comme les composants des modules de détection et de suivi de mouvement, mais ils ne seront utilisés que dans un second temps.

Un grand effort est donc à faire pour élaborer la formalisation qui va permettre d'utiliser ces IMPS *au niveau fonctionnel.*

2.2 Spécification des actions

Dans le paragraphe précédent, nous avons présenté les deux principales entités algorithmiques de vision qui vont être utilisées dans l'application.
Mais pour répondre complètement à la spécification des TRs définissant une action élémentaire, il faut adjoindre à ces entités leur comportement logique. On pourra ensuite obtenir la vue abstraite de chacune de ces actions que le niveau de contrôle pourra manipuler.

Pour spécifier le comportement logique d'une TR, ORCCAD utilise des événements et des traitements typés pour faciliter et automatiser cette spécification. A ce niveau, on distingue 3 classes de signaux :

1. les pré-conditions qui conditionnent l'activation d'une tâche ;
2. les exceptions qui se répartissent en 3 types :

Exception	Type 1	Type 2	Type 3
Conséquence	Paramétrisation pendant l'exécution de la TR courante	Arrêt de laTR courante Appel à la PR au niveau supérieur	Mise en sécurité du système

3. les post-conditions qui signalent et conditionnent la bonne terminaison de la TR.

Une fois ceci défini pour chaque TR, il est aisé de déterminer pour chacune de ces actions leur vue abstraite qui sera utilisée directement par le niveau de contrôle par l'intermédiaire des PRs, et du langage de programmation de mission MAESTRO.

On a pour les différentes TRs définies pour l'application :

- Commande du système ARGÈS
  - Fonctionnalité : commander le mouvement, les caractéristiques visuelles de la caméra ARGÈS via une commande formulée par l'utilisateur sous forme d'une chaîne de caractères, ou d'un appel de routine.
  - Spécification sous forme TR : voir la figure 2
- Détection de mouvement
  - Fonctionnalité : détecter une zone en mouvement dans l'image et retourner en paramètre la taille et le centre de la zone en mouvement.
  - Spécification sous forme TR : voir la figure 3

```
RT_Command
    Preconditions :
        ARGES_ON = OK
        ARGES_XIL = OK
    PostConditions :
        ARGES_COMM_OK
    Exception Events :
        T1_type :   RobotCommand == Acqui && IMAGE_Buff == NULL
                    Processing : Try_Robot ++ ; New_Robot
        T3_type :   (RobotCommand == Acqui && IMAGE_Buff == NULL
                    &&  Try_Robot == MAX_TRY)
                    || (ARGES_STATE == KO)
                    Processing :      Abort_Robot
    Parameters :
        WhichArgesComm : STRING
    Return :
        ArgesCommOK : BOOLEAN
--------------------------------------------------------------
    Algorithm :
        argesComm(WhichArgesComm) : ArgesCommOK
```

Figure 2: TR commande robot

```
RT_Detect
    Preconditions :
        ARGES_ON = OK
        ARGES_XIL = OK
    PostConditions :
        MOTION = {YES,NO}
    Exception  Events :
        T1_type :   IMAGE_Buff == NULL
                    Processing : Try_Acqui ++ ; New_Acqui
        T2_type :   CurrentBlobThres < BLOBTHRES
                    Processing : Restart_BlobDetect
        T3_type :   IMAGE_Buff == NULL &&  Try_Acqui == MAX_TRY
                    Processing :      Abort_Acqui
    Parameters :
        SmoothWin : INT
        GradThres  : INT
    Return :
        BlobType : STRING
--------------------------------------------------------------
    Algorithm :
        loop
            RT_ACQUI
            DIFF
            SMOOTH(SmoothWin)
            MaxMotion
            BlobDetect(GradThres) : BlobType
        endloop
```

Figure 3: TR Détection de mouvement

- Suivi de mouvement
  - Fonctionnalité : à partir de la taille et du centre de la zone en mouvement détectée, calculer le déplacement en pan et tilt de la caméra pour centrer cette zone dans l'image; bouger la caméra suivant le pan et le tilt déterminé et réitérer le processus de détection de mouvement.
  - Spécification sous forme TR : voir la figure 4

```
RT_Follow
   Preconditions :
      ARGES_ON = OK
      ARGES_XIL = OK
   PostConditions :
      NO_MOVEMENT
   Exception Events :
      T1_type :   IMAGE_Buff == NULL
                  Processing : Try_Acqui ++ ; New_Acqui
      T2_type :   CurrentBlobThres < BLOBTHRES
                  Processing : Restart_BlobDetect
      T3_type :   IMAGE_Buff == NULL && Try_Acqui == MAX_TRY
                  Processing :   Abort_Acqui
   Parameters :
      BlobType : STRING
   Return :
-----------------------------------------------------------
   Algorithm :
      loop
           PanTiltCalc(BlobType) : Pos
           Move(Pos)
           RT_Detect
      endloop
```

Figure 4: TR Suivi de mouvement

Pour les vues abstraites, une syntaxe a été créée [5] permettant de définir les vues abstraites d'une TR. La syntaxe est la suivante :

```
TaskAbstract    →   "NAME" ":" <TaskName>
                    "PARAM" ":" paramypelist "RETURN" ":" <paramtypelist>
                    "ROBOT" ":" <RobotName>
                    precond
                    postcond
                    exceptionT1
                    exceptionT2
                    exceptionT3
paramtypelist   →   <void> | paramtype  "," paramtype
paramtype       →   "INT" | "STRING" | "BOOL" | "FLOAT"
precond         →   "PRECOND" <SignalName>
postcond        →   "POSTREL" <SignalName>
exceptionT1     →   "T1" <T1Name>
exceptionT2     →   "T2" <T2Name>
exceptionT3     →   "T3" <T3Name>
```

Voici un exemple d'illustration :

NAME :	RTDetectMotion
PARAM :	INT,INT
RETURN :	STRING
ROBOT :	ARGES
PRECOND	RTDetectMotion_Prec1
POSTREL	RTDetectMotion_Movement
T1	NO_IMAGE
T2	LOWER_THRES
T3	MAX_ACQUI

La partie implémentation des TRs est faite à travers les *Tâches Modules* ou TMs. Ces TMs sont des tâches temps réel qui implémentent de manière optimale les fonctions élémentaires qui interviennent dans les TRs.
Ces TMs seront traités dans la section 5.2.

2.3 Spécification de l'application

Une fois les différentes TRs spécifiées et les vues abstraites associées établies, il faut pouvoir élaborer des missions complexes à partir de ces différentes actions.

Cet arrangement de TRs se fait au niveau contrôle à travers le formalisme des Procédures Robot ou PRs et l'utilisation du langage MAESTRO.

Les PRs représentent la spécification d'un arrangement de plusieurs TRs dans le but de réaliser un objectif global. Cette gestion des TRs est assurée par la spécification des PRs sous forme d'un comportement logique.

La traduction du comportement logique des PRs se fait en langage ESTEREL, et de manière automatique grâce au langage de programmation de mission : MAESTRO [5].

Ce langage utilise directement les vues abstraites des différentes actions et permet grâce à une syntaxe définie dans [5] de les arranger afin d'élaborer une mission, dans notre cas notre application.

Une fois la mission programmée, en utilisant la syntaxe de MAESTRO, un automate est généré qui va contrôler le comportement de l'application.
Cet automate pourra être ensuite utilisé pour établir des vérifications du comportement de l'application et pour prouver que certaines propriétés des spécifications sont bien vérifiées.

La figure 5 illustre l'interface graphique utilisée par le langage de programmation de mission pour définir le comportement des PRs, i.e. les différents enchaînements d'actions à gérer.

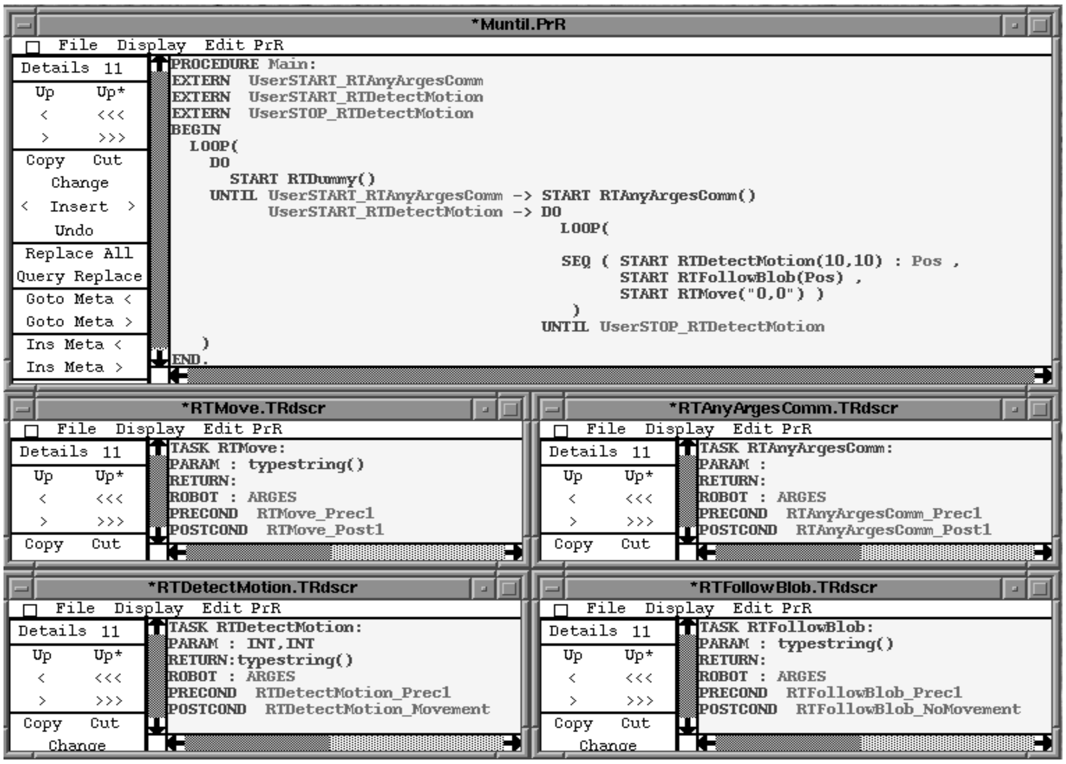

Figure 5: Interface de programmation de MAESTRO

Discussion sur la méthodologie Pour définir notre mission, nous avons envisagé deux principales possibilités pour arranger plusieurs actions permettant de mettre en oeuvre la fonctionnalité de détection et suivi de mouvement :

1. Soit :

```
do
  loop (
       SEQ ( RTDetect(Pos)
             RTMove(Pos)
           )
  )
UNTIL No_Movement
```

2. Soit : Exemple :

```
do
   loop (
        SEQ ( RTDetect(Pos)
              RTFollow(Pos)
              RTMove(0,0)
            )
   )
```

Dans le 1er cas :

+ il y a réutilisation des TRs.

− cette structure force des réactions de l'automate à la cadence de l'horloge. Et par conséquent l'automate transite de trop nombreuses fois, et sans lien avec une modification du comportement logique de l'application.

Dans le 2ème cas :

+ cette structuration respecte la logique globale de l'application. L'automate transite en fonction des changements de comportement de l'application : détection puis suivi avec détection de mouvement puis repositionnement de la caméra dans sa position initiale quand il n'y a plus de mouvement.

− pas de réutilisation des TRs : la tâche robot RTFollow englobe dans sa spécification même l'aspect détection de mouvement et déplacement de la caméra, i.e la boucle de poursuite, sans fait appel explicitement aux tâches robot RTDetect ou RTMove.

3 Validation: logique

Dans le paragraphe précédent, nous avons présenté l'ensemble des spécifications de l'application. Nous avons vu que le langage MAESTRO est transformé en programme ESTÉREL qui se compile sous forme d'un automate.
On peut alors envisager de faire des vérifications sur cet automate, et de valider un certain nombre de propriétés logiques.

Ce travail se fait avec les outils de vérification automatique dédiés à l'analyse d'automate finis : AUTO/GRAPH [6].

Ces outils permettent d'extraire de l'automate global un automate réduit plus facilement vérifiable pour valider des comportements particuliers. Avec l'interface graphique de l'outil AUTOGRAPH, on peut visualiser les automates résultants.

On peut donc grâce à ces outils détecter des comportements anormaux, ou vérifier qu'une propriété particulière est bien réalisée.

La figure 6, illustre une partie de l'automate, élaboré à travers le langage de programmation de mission et visualisé grâce à l'outil AUTOGRAPH. Cette portion de l'automate correspond à la partie de contrôle de la requête de détection et de suivi de mouvement.
Remarque sur la lecture de l'automate :

- les signaux d'entrée sont préfixés par ?
- les signaux de sortie sont préfixés par !.

Supposons que l'on souhaite vérifier que l'automate suit la politique de transition entre deux TRS, schématisée par la figure 7, et qui concerne la requête de détection et suivi de mouvement.

On va analyser les transitions de l'automate et on vérifie que l'on retrouve bien l'enchaînement des signaux `InitOK_ARGES`, `ReadyToStop_ARGES`, `Activate`, `EndKillArges` spécifiés pour la transition entre deux TRS dans cet automate.

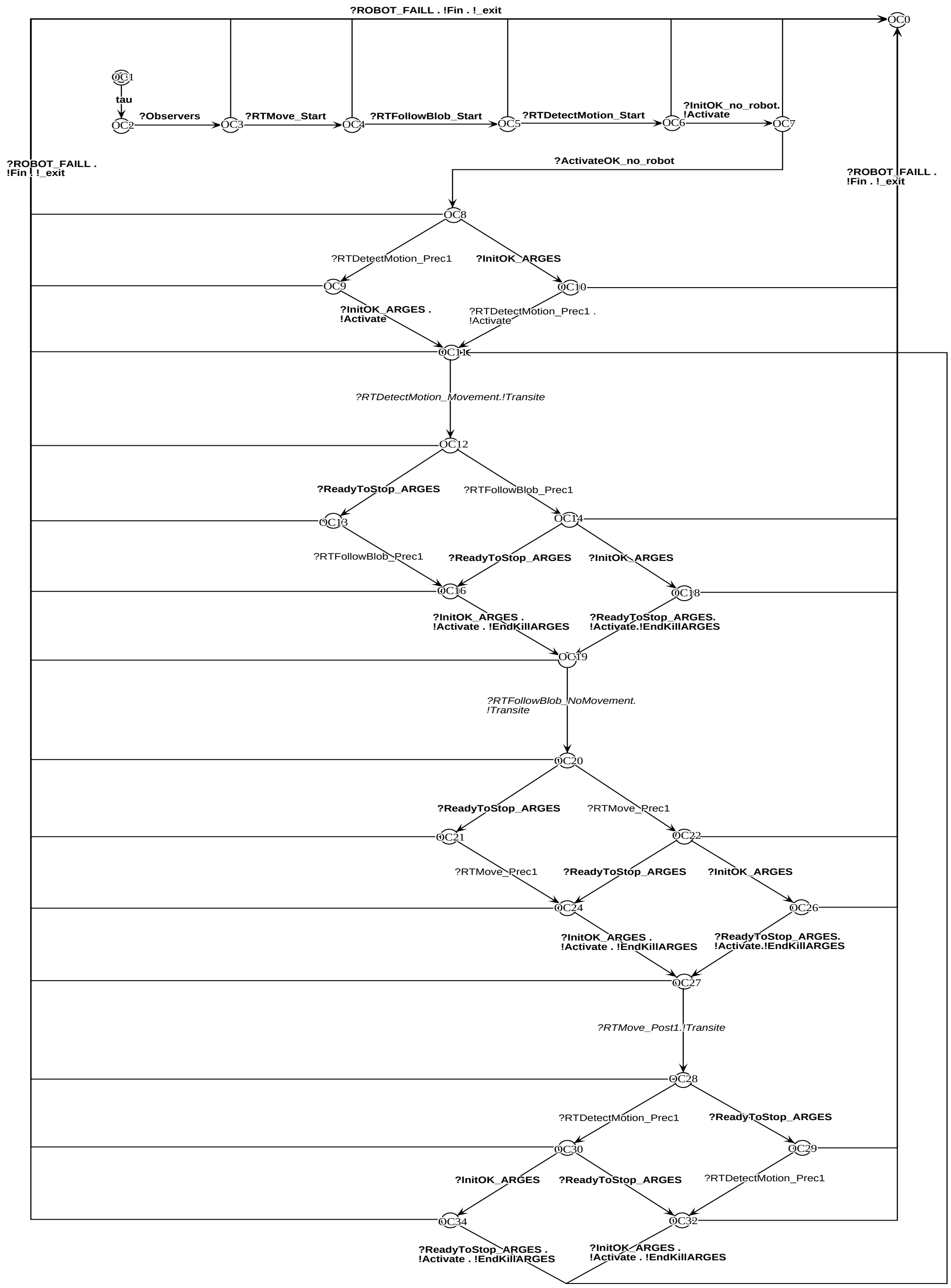

Figure 6: Fragment d'automate généré par ESTEREL

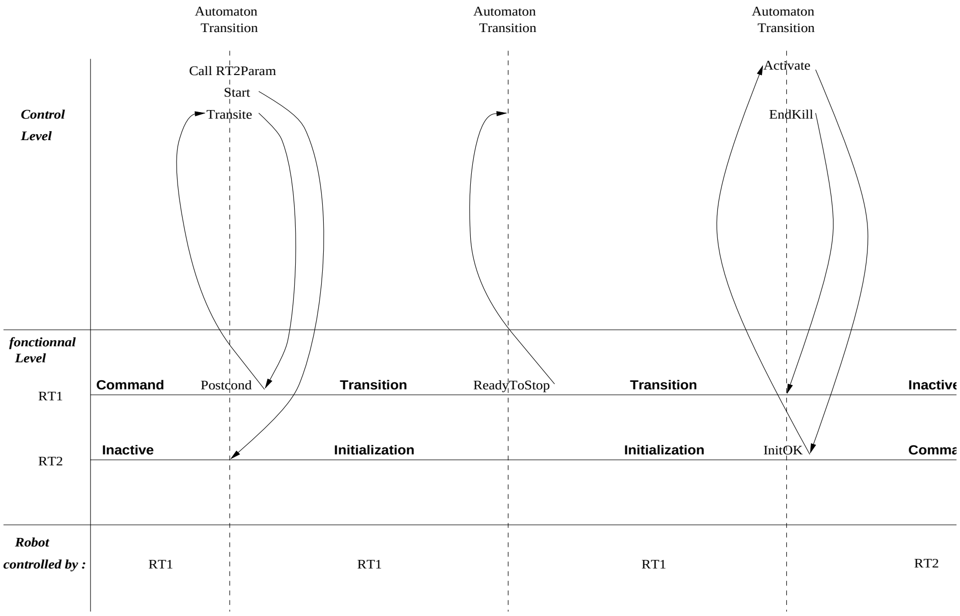

Figure 7: Déroulement d'une transition entre deux TRs

4 Implémentation du prototypage

Dans cette section, nous allons envisager l'implémentation des modules de vision décrits au paragraphe 2.1.

Ensuite, nous envisagerons la partie qui va gérer dans un contexte temps réel la partie contrôle ESTÉREL avec les modules de vision : le système ORCCAD. Nous allons étudier la façon dont le portage POSIX s'est effectué.

4.1 Les modules de vision

- Implémentation du module de commande de la caméra :
  C'est une routine founie par le driver du systéme ARGÈS qui a été employée.

  C'est une fonction générique, dans le sens qu'elle traite toutes les commandes robot indépendamment de leur nature, qui lance au système *Argès* la commande adéquate selon la chaîne de caractères correspondant à la requête formulée par l'utilisateur,

- Implémentation du module de détection de mouvement :
  Cette routine implémente l'enchaînement d'algorithmes décrits au paragraphe 2.1.
  La routine de base (routine `acvMotionBlob`) provient de la librairie `AcVis` du projet ROBOTVIS . Elle a été légèrement modifiée pour retourner la taille et le centre de la zone en mouvement.

- Implémentation du module de suivi de mouvement

Nous avons employé la méthode empirique décrite dans le paragraphe 2.1 sur une image de taille 48x36.
Les valeurs des facteurs d'échelle de pan et tilt obtenues sont les suivantes :
$K_{pan} = 1.3$, $K_{tilt} = 1.4$.

On a élaboré une fonction qui combine à la fois détection et suivi de mouvement. Si une zone de mouvement est rencontrée, le suivi est calculé et le changement d'orientation de la caméra est enclenché.
Cette routine rend la main lorsqu'au bout de 6 images consécutives aucun mouvement n'a été déterminé.

4.2 ORCCAD et la Portabilité Temps-Réel

4.2.1 Modification de la version d'ORCCAD

La version du système ORCCAD [7] été implémentée explicitement pour tourner sur le système d'exploitation VXWORKS.

Pour permettre de gérer les fonctionnalités temps réel, ORCCAD utilise les objets suivants :

- les Sémaphores :
  Un sémaphore est un objet qui possède une valeur entière et un ensemble de processus bloqués associés à lui :

si $sem >= 0$	aucun processus n'est bloqué sur le sémaphore ;
si $sem < 0$	$\|sem\|$ = le nombre de processus bloqués sur le sémaphore

  On distingue deux types de sémaphore :

  - *le sémaphore binaire* avec $sem \in [0, 1]$.
    Ce sémaphore sert comme mécanisme élémentaire de synchronisation entre deux processus ou pour protéger une ressource partagée par plusieurs processus.
  - *le sémaphore à compte* avec $sem \in I\!N$.
    Ce sémaphore permet de garder la trace de la disponibilité des ressources, en mémorisant le nombre d'appel qu'il a reçu pour une ressource donnée.

  Dans le système ORCCAD, ces sémaphores servent à protéger des ressources critiques telles que l'accès aux paramètres d'une TR, et pour permettre la synchronisation des différentes TRS suivant un schéma de transition du type de la figure 7.

- les Queues de Messages :
  Une queue de messages est un objet qui permet le passage d'un nombre arbitraire de

données entre processus d'une application.
Les caractéristiques d'une queue de messages sont les suivantes :

- un nom permet l'identification d'une queue de message. Ce nom est similaire à un nom de fichier.
- une longueur maximale impose la taille des messages qui transitent par une queue de message.
- une priorité est assignée à chaque message. Cette priorité est interprétée par le système pour déterminer l'ordre de délivrance des messages dans une queue de message.
- une demande d'ouverture de la queue de message doit être faite par tout processus qui veut communiquer avec cette queue de message.
- un signal peut être émis lors de l'arrivée d'un message dans une queue de message.

Dans le système ORCCAD, ces messages servent à la gestion des signaux asynchrones par l'automate synchrone ESTÉREL.

- les Sous-Processus ou Threads :
Un `thread` est une entité d'exécution interne à un processus et qui possède un environnement simplifié.
Les threads n'existent qu'à l'intérieur d'un processus. Tous les threads d'un processus partagent l'ensemble des ressources de ce processus, permettant une programmation simple et efficace.

Les caractéristiques d'un thread sont les suivantes :

- un appel bloquant exécuté par un thread ne bloque pas les autres threads du processus.
- le changement de contexte entre threads d'un même processus est rapide car l'environnement d'un thread est simple.
- le processus lui-même est considéré comme un thread. Ainsi on pourra parler de processus *monothreadé* ou de processus *multithreadé* suivant la capacité à gérer les threads.
- Du point de vue de l'ordonnanceur :
en mode global, le thread représente une tâche indépendante et de même niveau que le processus.
- Du point de vue du processus englobant le thread :
le thread est vu comme un sous-programme autonome avec lequel le processus englobant partage ses données.

Dans le système ORCCAD, les threads servent à exécuter les tâches suivantes : l'interface graphique, le gestionnaire d'événements asynchrones, le contrôleur des TRs et les actions à exécuter TMs sur le processus principal.

- une horloge et un timer :
  Ce timer va servir à cadencer les différentes actions pour assurer que les contraintes temporelles soient bien garanties entre les différentes tâches.
  On programme un timer de façon à ce qu'à chaque expiration du délai, une nouvelle action soit enclenchée.

4.2.2 La norme POSIX

Comme nous l'avons dit au paragraphe précédent, la version d'ORCCAD de [7] n'est utilisable que sur les systèmes VXWORKS. Or, pour être intéressant au niveau industriel, il est très important que les applications complexes puissent tourner sur n'importe quel système. Il faut donc que l'application soit portable.

Pour atteindre cette portabilité au niveau des systèmes temps-réel, une norme a été créée : POSIX ou **P**ortable **O**perating **S**ystem **I**nterface [8]

Cette norme propose la spécification d'un ensemble de fonctions permettant de solliciter les services de base d'un système d'exploitation. L'objectif est de garantir le développement d'applications portables au niveau du code source, entre les systèmes d'exploitation conformes à la norme.

Cette norme ne constitue pas la définition d'un système d'exploitation. Elle fournit une liste de points d'accès aux services du système : pour chaque fonction, le comportement attendu dans les différentes circonstances susceptibles de se produire est complètement défini. Le but de cette norme est de masquer les spécificités des systèmes sous-jacents.

POSIX fournit des moyens, sous forme de bibliothèques, pour rendre portables des applications écrites pour un système d'exploitation donné, et de rendre l'écriture d'un programme plus simple.

Cette norme a été standardisée par ANSI et ISO sous le nom POSIX 1003.1X, où le X correspond au type de l'extension envisagée :

- POSIX.1003.1b : extension temps réel
- POSIX.1003.1c : extension thread

Connaissant les caracréristiques de la norme POSIX, nous avons travaillé sur une version de ORCCAD que nous avons rendue compatible POSIX.
Ce travail s'est traduit par la transcription de tous les appels relatifs aux entités propres

à la gestion du temps réel (sémaphores, queues de messages, threads, timers, signaux) en appels à des routines POSIX gérant ces mêmes entités.

4.2.3 Problèmes de portabilité sur SOLARIS 2.5

Le système d'exploitation SOLARIS 2.5 sur lequel le système ARGÈS est installé ne supporte pas certaines définitions POSIX. C'est le cas pour :

- les sémaphores
- les queues de messages
- les threads

Par conséquent, nous avons dû utiliser la définition donnée par SOLARIS pour chacune des trois entités précédentes.
Le système ORCCAD que nous avons modifié reste cependant compatible avec la norme POSIX : toutes les entités concernées ont été définies selon la norme, et pour celles qui ne sont pas supportées par SOLARIS, les deux définitions sont installées et contrôlées par un `#define SOLARIS` qui détermine à la compilation le type de système d'exploitation sur lequel tournera l'application :

- sémaphore :

```
#ifdef SOLARIS
         (sema_init(DeleteTasksPrR, 0, USYNC_THREAD, 0) != OK) ||
         (sema_init(DeletePrR, 0, USYNC_THREAD, 0) != OK) )
         // USYNC_THREAD -> intra-process and counter = 0 {
#else
         (sem_init(DeleteTasksPrR, 0, 1) != OK) ||
         (sem_init(DeletePrR, 0, 1) != OK) )
         // 0 -> intra-process{
#endif
```

- thread :

```
#ifdef SOLARIS
if (thr_create(NULL, NULL, (FUNCPTR)interface,
               NULL, NULL, &InterfaceId) != OK)  {
   printf("error in thr_create interface,stack_size=%d\n",
          THR_MIN_STACK) ;
```

```
}
#else
if (pthread_create(InterfaceId, NULL,
                   (FUNCPTR)interface,0 ) != OK) {
  printf("error in pthread_create interface \n") ;
}
#endif
```

5 Application : le Prototype ArgiMoge

Comme spécifié dans l'introduction, l'application qui a été conçue gère un système à une caméra, le système *Argès*, et fournit à l'utilisateur une interface graphique à partir de laquelle il peut piloter le système de caméra, lancer la requête de détection avec suivi de mouvement.

Dans les paragraphes précédents, nous avons vu comment les modules de pilotage de caméra et de détection et de suivi de mouvement ont été formalisés puis implémentés. Nous avons présenté le portage qui a été fait sur ORCCAD pour être compatible POSIX.

Dans cette section nous allons montrer sur quel matériel et comment se fait l'intégration de ces différents modules pour fournir une application.

5.1 Hardware Utilisé

Au niveau hardware, l'application utilise le système monoculaire ARGÈS. C'est une plateforme expérimentale développée dans le projet ROBOTVIS pour tester les algorithmes de vision active 3D.

Le système ARGÈS est formé des éléments suivants :

- un caméra couleur CCD Acom1 PAL, contrôlée par ordinateur. Ses caractéristiques sont les suivantes :
  - une focale variable allant de $f = 5.9$ à $f = 47.2$, permettant de faire du zoom ;
  - un auto-focus numérique codé sur 10 bits ;
  - une interface vidéo et une interface rs232C ;
  - un rapport $S/N > 46dB$;
  - une résolution supérieure à 450x400.

- une tourelle Pan-Tilt de chez RobotSoft qui utilise une interface rs232C, pour piloter le moteur.
- un degré de liberté linéaire de CharlyRobot, avec une résolution de 0.1 mm.
- un rail de $85cm$ sur lequel peut se déplacer la tourelle Pan-Tilt.
- un panneau d'acquisition SunVideo basé sur la librairie xil.

Le système est piloté à travers un driver logiciel défini dans le fichier d'interface `arges.h`. On peut donc commander la caméra, modifier le pan, le tilt, la position de la tourelle sur le rail, acquérir des images que l'on pourra ensuite traiter.

5.2 Réalisation

5.2.1 Du point de vue Utilisateur

L'application se présente comme une interface graphique homme–machine, grâce à laquelle un utilisateur peut lancer quatre types de procédures :

- le pilotage du système ARGÈS. L'utilisateur a à sa disposition :
  - le bouton `"Acqui"` : qui lance une acquisition vidéo qui va s'afficher sur la fenêtre de gauche de l'interface ;
  - le bouton `"Init"` : qui lance la procédure d'initialisation du système ARGÈS;
  - une ligne de commande `"arges >:"` situé en bas de la fenêtre d'acquisition, qui permet à l'utilisateur de taper directement les commandes pour ARGÈS.
- la procédure de détection et de suivi de mouvement. Pour l'enclencher, l'utilisateur clique à la souris le bouton noté "Motion Detection". Une fois cliqué, ce bouton change de couleur et est noté "Stop Detect". Ainsi, pour stopper la procédure de détection, l'utilisateur a juste besoin de recliquer sur ce bouton.
  Pour cette procédure, l'utilisateur a deux paramètres à initialiser:
  - "SmoothWin", qui correspond à la fenêtre de lissage désiré, qui va servir lors du processus permettant d'éliminer le bruit dans l'image à traiter ;
  - "GradThres", qui correspond au seuil limite du gradient à employer sur l'image lors du processus de détection de mouvement.

  Il est à noter que le positionnement de ces paramètres peut se faire tout au long du processus de détection, car toutes les actions sont traitées en temps-réel.

- faire appel à un fichier d'aide : en cliquant sur le bouton "Help". Une fois ce bouton cliqué, l'application NETSCAPE est lancée et la page d'aide apparaît sous forme d'une page WWW.

- quitter l'application : ceci est fait en cliquant sur le bouton "Quit".

5.2.2 Du point de vue de l'implémentation

C'est le système ORCCAD qui orchestre le déroulement de l'application.
C'est lui qui va lancer l'automate de contrôle, qui va enclencher le timer qui va cadencer l'application et l'interface graphique. et lancer les actions que l'utilisateur aura programmées en fonction des requêtes reçues par l'intermédiaire de l'interface graphique.

L'interface graphique L'interface a été créée en utilisant la librairie ACTIVEWINDOWS développée au sein du projet ROBOTVIS, et qui permet de programmer des interfaces XWINDOW.

Cette interface est lancée sous forme d'un thread par le processus principal qui est en fait le systéme ORCCAD.

A chaque bouton créé est associé à une fonction *call back* qu'ORCCAD va gérer en fonction de l'état de l'automate.

Cette interface est mise à disposition du système ORCCAD sous la forme d'une librairie qui est "linkée" avec l'application lors de l'édition de lien des sources de l'application.

Implémentation des actions ou TMS L'utilisateur de ORCCAD doit implémenter chaque action ou TM comme la composition des fichiers suivants :

- `inc.h` : ce fichier contient la liste des `include` nécessaire à l'implémentation de l'action.

- `var.h` : ce fichier contient la liste des variables nécessaires à l'implémentation de l'action.

- `param.h` : ce fichier récupère les paramétres d'entrée de l'action fournis par l'intermédiaire de l'automate.

- `init.c` : ce fichier implémente les traitements à faire comme initialisation de l'action à exécuter. Ce fichier est exécuté après que le signal de déclenchement de l'action a été reçu par l'automate.
  Ce traitement se termine par l'émission à l'automate du signal de précondition, indiquant ainsi que l'initialisation de cette action a bien eu lieu.

- `corps.c` : ce fichier implémente comme son nom l'indique le corps de l'action à traiter. Ce fichier est exécuté après que le signal de précondition a été reçu par l'automate et que l'état de l'automate permette le lancement du corps de l'action
  Le corps se termine par l'émission à l'automate du signal de postcondition.

- `end.c` : ce fichier implémente le traitement correspondant à la terminaison de l'action. Ce fichier est exécuté après que l'automate a reçu le signal de postcondition.

Pour chaque action, l'utilisateur devra créer ces cinq fichiers.
Ces fichiers devront être créés pour chaque action à implémenter et dans des répertoires différents à partir du répertoire `MT` compris dans l'arborescence du système ORCCAD.

Ensuite, pour chacune des actions à implémenter, il faudra donner le chemin d'accès de ces différents fichiers dans le fichier `mts.C` contenu dans le répertoire `Prr/nom_application/` de l'arborescence du système ORCCAD.

Le protocole indiqué ci-dessus est le seul qu'un utilisateur qui souhaite implémenter des actions sur ORCCAD a à faire pour élaborer son application, une fois que la mission en MAESTRO a été créée et sans compter la programmation de l'interface graphique.

Performence temps réel Le cadencement des actions se fait sur une période de 1.2secondes, pour des traitements sur une image 48x36.

Des erreurs ont été détectées au niveau de la librairie SOLARIS XIL qu'utilise le driver de ARGÈS pour stocker une image, et qui sont responsablea des délais parfois trop longs lors des acquisitions.

6 Conclusion et Perspectives

Dans ce document, nous avons analysé comment spécifier, valider et implémenter une application mixant les aspects robotiques et vision, en accord avec les formalismes (TRS, PRS) et outils (MAESTRO, AUTO/GRAPH) introduits dans le système ORCCAD.

Nous avons présenté les modifications apportées à ORCCAD pour le rendre compatible à la norme POSIX ainsi que les problèmes que causent une telle migration sur un système SOLARIS.

Cette application a montré que l'utilisation du système ORCCAD pour le développement d'applications de vision était tout-à-fait réalisable.

Cependant, l'application développée n'est pas encore suffisament significative au niveau de la complexité des actions de vision notamment. De plus, les performances temps-réel ne sont pas non plus très satisfaisantes.

Aussi, les perspectives sont nombreuses.
Tout d'abord, améliorer le driver ARGÈS au niveau de la vitesse d'acquistion. Pour se faire, les gens de SUN ont été contactés en vue de corriger les erreurs de la librairie XIL.

Une application plus réaliste et plus complexe au niveau des algorithmes de vision est en voie de spécification. Ceci va permettre d'appréhender plus en détail les problèmes inhérents à la formalisation des actions de vision pour obtenir une utilisation plus haut niveau de ces actions. En particulier, il sera intéresant d'étudier ce que peut apporter un formalisme basé sur les IMPS vus à la fin du paragraphe 2.1.
De plus, au niveau de la validation logique de l'application, le travail de vérification va se faire sur un jeu de signaux et d'actions plus complet. Et on pourrait envisager de construire la spécification de l'application avec l'interface graphique d'ORCCAD.

Il est aussi intéressant d'étudier ce que les langages synchrones de flot de données tels que SIGNAL ou les environnements basés sur des flots de données tels que PTOLEMY peuvent apporter notamment au niveau de la spécifications des actions de vision.

References

[1] T. Viéville, E. Clergue, R. Enciso, and H. Mathieu. Experimentating with 3D vision on a robotic head. *Robotics and Autonomous Systems*, 14(1), 1995.

[2] D.Simon, B. Espiau, E. Castillo, and K. Kapellos. Computer aided design of a generic robot controller handling and real-time control issues. *IEEE Transaction on Control Systems Technology*, 1(4):213–229, December 1993.

[3] T. Viéville. The acvis package documentation version 2.1. Technical Report RR-?, INRIA, 1994.

[4] G. Massini, A. Napoli, D. Colnet, D. Léonard, and K. Tombre. *Les langages à objets*. InterEditions, 1st edition, 1989.

[5] Ève Coste-Manière N. Turro. Mission programming language. Technical Report Final Report of the WorkPackage 2 Task 2.2 of the Union, Esprit Basic Research Project, July 1996.

[6] V. Roy and R. de Simone. Auto and autograph. In *Workshop on the Computer Aided Verification*, New Brunswick, June 1990.

[7] R. Pissard-Gibollet, K. Kapellos, P. Rives, and J.J. Borrelly. Real-time programming of a mobile robot actions using advanced control technics. In *ISER'95 International Symposium on Experimental Robotics*, Stanford, USA, May 1995.

[8] Bill O. Gallmeister. *POSIX.4, Programming for the Real World*. O'Reilly & Associates, inc., 1st edition, January 1995.