République Algérienne Démocratique et Populaire

Université BADJI MOKHTAR ANNABA  **جامعة باجي مختار عنابة**

Faculté des Sciences de l'Ingénieur
Département d'électronique

MEMOIRE

En Vue De l'Obtention Du Diplôme De **MAGISTERE**

Intitulé

Modélisation des systèmes complexes parallèles Application : ROBOTS mobiles autonomes évoluant dans un environnement parsemé d'obstacles mobiles.

Option
Automatique Industrielle

Présenté par
HOUDA Kelthoum

Président	: DEBBACHE N.	Pr.	U. ANNABA
Directeurs de mémoire	: DJEGHABA M.	Pr.	U. ANNABA
	: LAKEL R.	M.C	U. ANNABA
Examinateurs	: ABASSI H.A.	Pr.	U. ANNABA
	: KIMOUR.M.	M.C	U. ANNABA

Année 2006

	Introduction générale
Chapitre 1 :	Etat de l'art sur la robotique mobile et sur la coopération entre robots
Chapitre 2 :	Planification de trajectoire et navigation
Chapitre 3 :	Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes
Chapitre 4 :	Implémentation des processus parallèles à l'aide du langage de programmation ADA
Chapitre 5 :	Conception d'une plateforme de robots mobiles autonomes et début de réalisation
Chapitre 6 :	Simulation
	Conclusion générale
	Bibliographie
	Annexe et Appendice

Résumé.

L'une des principales motivations du développement de la robotique mobile demeure la substitution de l'être humain en milieu hostile. Les robots mobiles autonomes qui réalisent des tâches sans intervention d'un opérateur sont nécessaires dans plusieurs domaines d'application militaire, d'exploration spatial ou sous-marine…
Généralement, on cherche à minimiser les risques à l'être humain et à réduire les coûts des opérations. Ainsi, parmi les tendances actuelles en robotique, on note la mobilité et la coopération en environnement dynamique.

Ce mémoire a pour objet la conception d'une plate forme de robotique mobile et l'étude présentée est composée de 3 parties essentielles :

- Une planification de trajectoire basée sur la méthode des champs de potentiels fictifs pour obtenir une carte 2D de potentiel sans minimum local, excepté le point-cible.

- L'implémentation dans le langage ADA des problèmes de synchronisation et de communication dans un système de robots mobiles autonomes avec des points de rendez-vous fixes. L'analyse des invariants du réseau de pétri, modélisant l'ordonnancement des rendez-vous, permet de mettre en évidence les machines à états communicantes appelées processus.

- Et enfin la conception de la plate forme qui comporte une partie mécanique et une partie électronique embarqué et essentiellement un système de localisation qui est basé sur l'utilisation de balises infrarouges, principe choisi pour sa simplicité.

	Introduction générale
Chapitre 1 :	Etat de l'art sur la robotique mobile et sur la coopération entre robots
Chapitre 2 :	Planification de trajectoire et navigation
Chapitre 3 :	Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes
Chapitre 4 :	Implémentation des processus parallèles à l'aide du langage de programmation ADA
Chapitre 5 :	Conception d'une plateforme de robots mobiles autonomes et début de réalisation
Chapitre 6 :	Simulation
	Conclusion générale
	Bibliographie
	Annexe et Appendice

Abstract.

One of the principal motivations of the development of mobile robotics remains the human being substitution in hostile environment. The autonomous mobile robots which carry out tasks without intervention of an operator are necessary in several military applications, space or underwater exploration… Generally, one seeks to minimize the human being risks and to reduce the costs of the operations. Thus, among the current tendencies in robotics, there are notes mobility and co-operation in dynamic environment.

This memory has for aim, the design of mobile robotics floor plan and the study presented is made up of 3 main parts:

- A planning of trajectory based on the method of the fields of fictitious potentials to obtain a 2D chart of potential without local minimum, except the target- point.

- Implementation in language ADA of the problems of synchronization and communication in a system of autonomous mobile robots with fixed rendezvous points. The analysis of the invariants of the Petri-nets, which the scheduling of the rendezvous, makes the existence of the communicating states-machines (called process.) obvious.

- And finally the design of the floor plan which includes a mechanical part, an embarked electronic part and mainly an infra-red based localization system for its simplicity

	Introduction générale
Chapitre 1 :	Etat de l'art sur la robotique mobile et sur la coopération entre robots
Chapitre 2 :	Planification de trajectoire et navigation
Chapitre 3 :	Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes
Chapitre 4 :	Implémentation des processus parallèles à l'aide du langage de programmation ADA
Chapitre 5 :	Conception d'une plateforme de robots mobiles autonomes et début de réalisation
Chapitre 6 :	Simulation
	Conclusion générale
	Bibliographie
	Annexe et Appendice

ملخص

إن الحاجات و الدوافع الرئيسية لتطوير مجال الروبوتيك تتمثل في تعويض الإنسان بالألة في البيئة المعادية له. الإنسان الآلي المستقل داتيا الدي يحقق عمله من دون تدخل أي مساعد, له أهمية كبيرة في ميادين عدة منها التطبيقات العسكرية, الإستكشافات الفضائية و ما تحت البحر... عموما نبحث على تجنيب الإنسان ما قد يؤذيه. فهكذا من بين الإتجاهات الحالية في مجال الروبوتيك نجد إنتقالا و تعاونا في بيئة ديناميكية.

هذه المذكرة تهدف إلى إنجاز هيكلة للروبوتيك المتحركة, والدراسة المقدمة مكونة من ثلاثة أجزاء أساسيه :

- طريقة تخطيط مسار مركزة على حقول طاقة خيالية للحصول على بطاقة ذات بعدين من دون الوقوع في الحضيض الأدنى ، إلا النقطة المستهدفة.

- تنفيذ برنامج ADA لحل مشاكل المزامنة و الإتصال في نظام الإنسان الآلي المستقل ذاتيا عبر نقاط تلاقي ثابتة.

- و أخيرا إنجاز هيكلة متكونة من جزء ميكانيكي و من جزء إلكتروني محمل و خوصوصا نظام تحديد المكان مركز على المعلم الذي يستخدم الأشعة ما تحت الحمراء.

Tables des matières

Chapitre 2.

Planification de trajectoire et navigation

Chapitre 3.

Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes.

Chapitre 4.

Implémentation des processus parallèles à l'aide du langage de programmation ADA.

Chapitre 5.

Conception d'une plateforme de robots mobiles autonomes et début de réalisation.

Chapitre 6.

Simulation.

Conclusion générale.

Bibliographie.

Annexe et Appendices.

	Introduction générale.
Chapitre 1 :	Etat de l'art sur la robotique mobile et sur la coopération entre robots.
Chapitre 2 :	Planification de trajectoire et navigation.
Chapitre 3 :	Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes.
Chapitre 4 :	Implémentation des processus parallèles à l'aide du langage de programmation ADA.
Chapitre 5 :	Conception d'une plateforme de robots mobiles autonomes et début de réalisation.
Chapitre 6 :	Simulation.
	Conclusion générale.
	Bibliographie.
	Annexe et Appendice.

Introduction générale.

Objectifs.

Depuis quelques années, un intérêt croissant est porté au sein de la communauté robotique au développement de systèmes intelligents autonomes et de plates-formes expérimentales dans le cadre de la robotique d'intervention. Un tel intérêt peut être perçu comme une conséquence logique à l'apparition d'applications potentielles et au désir de mettre les robots sur des tâches nouvelles telles que les opérations d'intervention sur sites accidentés, la manipulation sur sites sensibles en l'occurrence nucléaires, ou l'exploration de sites maritimes ou planétaires.

Pour la réalisation de certaines tâches, il est plus sûr d'utiliser un groupe de robots qu'un robot unique. Pour cela chacun des robots navigue de manière autonome en réalisant une tâche qui lui est propre et communique avec les autres robots en des points de rendez-vous fixes. C'est la tendance actuelle de la robotique où l'accent est mis sur les problèmes de compétition et de coopération en environnement dynamique.

La plateforme robotique est constituée d'une plateforme matérielle et logicielle. Actuellement, aucune plateforme robotique n'est universelle. Il existe une multitude de robots, chacun se distinguant par sa cinématique, sa puissance de calcul, ses périphériques embarqués, sa taille ou son autonomie par exemple. Dans le cadre de ce mémoire, quelques critères prévalent : le prix, la construction, la fiabilité et les prérequis.

Parmi les plateformes existants, nous pouvons citer les MilliBots [KHO 99], les CotsBots [BER 01], les Crickets [CRI 02], les Khepera [KHE 03] et les LEGO Mindstorms [LEG 97]. Toutes ces plateformes répondent aux critères cités ci-dessus excepté le prix et la disponibilité.

Notre objectif est de concevoir une plateforme expérimentale de robotique mobile de type modulaire et ouverte, pouvant intégrer les nouveaux modules au fur et à mesure de leurs développements.

Le projet comporte essentiellement trois aspects :

- Un aspect pratique avec la réalisation dans une première phase de trois (03) robots mobiles au sein du laboratoire (LASA) du département électronique de l'université ***Badji Mokhtar Annaba***. Ces robots sont de types étagés et sont composés de divers modules qui s'imbriquent les uns sur les autres. Pour notre application, on a retenu deux modules de bases : le module comportant la partie motrice et la partie d'alimentation électrique d'une part, et le module de contrôle intégrant toutes l'électronique embarqué bâti autour d'un microcontrôleur d'autre part.

  Dans cette phase de réalisation pratique les aspects de navigation, de perception de l'environnement ainsi que la réalisation de cartes intelligentes embarquées sont étudiés.

- Un aspect lié à la modélisation de l'ordonnancement des rendez-vous entre robots dans lequel les problèmes de compétition et de coopération sont traités.

  La compétition entre les robots est liée aux conflits générés par la gestion des ressources limitées alors que la coopération est liée à la tâche commune qu'ils doivent réaliser.

  Les réseaux de Pétri, issues d'une théorie mathématique en continuelle évolution [HAD 88], permettent de modéliser l'ordonnancement des déplacements des robots tel qu'il est spécifié dans le cahier des charges.

  Les possibilités de validation qu'ils offrent constituent un atout fondamental, ainsi ils peuvent, grâce à la notion de vivacité, évaluer de façon certaines les risques de blocage d'un système.

- Un aspect très important lié à l'implémentation de la décomposition du modèle réseau de Pétri obtenu à l'aide d'un langage dédié à la programmation multi-tâches tel que Ada.

Plan du mémoire.

Ce document est constitué de six (06) chapitres, et des annexes détaillant les caractéristiques techniques liées à notre application, et des Appendices permettant d'indexer les différentes définitions, figures et tableaux.

Le ***chapitre 1*** introduit de manière générale tout ce qui a trait à la robotique mobile autonome. En particulier, nous présentons un aperçu historique sur la robotique mobile, ainsi que les différents types de systèmes de robots mobiles autonomes existants, nous décrivons aussi dans ce chapitre les différents problèmes posés dans le domaine de la robotique mobile tel que la planification de trajectoire, la navigation, l'évitement d'obstacle.
La conclusion de ce chapitre est réservée à la présentation des composants matériels et logiciels qui ont été retenus pour constituer l'ossature de départ de notre plateforme expérimentale.

Le ***chapitre 2*** décrit en détail l'algorithme développé pour obtenir une planification de trajectoire, ça consiste à établir une carte 2D de potentiel sans minimum local, excepté le point-cible, afin d'aboutir à une planification de trajectoire par une méthode globale basée sur l'établissement des champs de potentiels fictifs.

Le ***chapitre 3*** est consacré à la démarche de caractérisation des différents objets pertinents pour la génération G-Objets (Graphe Objet) à partir de l'analyse sémantique du réseau de Pétri modélisant l'ordonnancement des déplacements des robots. Chaque ensemble de G-objets définit des fonctionnalités qui sont regroupées dans des modules. L'ensemble de ces modules compose le squelette du prototype.
La production du squelette de l'application repose sur l'analyse des invariants du réseau de Pétri en vue de détecter des machines à états [KOR 92] communicantes.
Le modèle est décomposé selon les types de G-Objets suivants :

- Un *Processus* est le résultat de l'interprétation d'un flot conservatif déterminé par l'analyse du modèle. Il regroupe plusieurs objets de type Etat_Processus et de type Action.
- Un *Etat_Processus* caractérise un état possible d'un Processus. Il est tiré d'une place du modèle.

- Une *Ressource* privée ou partagée (donnée mémoire, message, article de fichier, …) correspond à une place du modèle.
- Une *Action* décrit le traitement à exécuter. Elle correspond à une transition du modèle. Selon le nombre de processus qui l'exécutent simultanément, une action sera synchronisée ou simple. Elle sera également gardée si son exécution est conditionnée par l'état des Ressources.

Le ***chapitre 4*** décrit, l'implémentation des différents G-Objets obtenu dans un langage multi tâche tel que Ada. Et nous décrivons les caractéristiques propres au prototype Ada centralisé.

Le ***chapitre 5*** englobe l'aspect pratique de notre travail. Nous présentons l'architecture générale de notre plate-forme ainsi que ces différents modules.

Le ***chapitre 6*** présente une simulation des deux algorithmes de planification de trajectoire ainsi que de l'algorithme d'implémentation de processus parallèle sur langage ADA.

Enfin, une ***conclusion générale***, dans laquelle les orientations futures ainsi que les perspectives offertes pour ce projet sont présentées.

	Introduction générale.
Chapitre 1 :	Etat de l'art sur la robotique mobile et sur la coopération entre robots.
Chapitre 2 :	Planification de trajectoire et navigation.
Chapitre 3 :	Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes.
Chapitre 4 :	Implémentation des processus parallèles à l'aide du langage de programmation ADA.
Chapitre 5 :	Conception d'une plateforme de robots mobiles autonomes et début de réalisation.
Chapitre 6 :	Simulation.
	Conclusion générale.
	Bibliographie.
	Annexe et Appendice.

Chapitre 1.

Etat de l'art sur la robotique mobile et sur la coopération entre robots.

1. Robotique mobile.

Il existe diverses définitions du terme robot, mais elles tournent en général autour de celle-ci :

Un robot est une machine équipée de capacités de perception, de décision et d'action qui lui permettent d'agir de manière autonome dans son environnement en fonction de la perception qu'il en a.

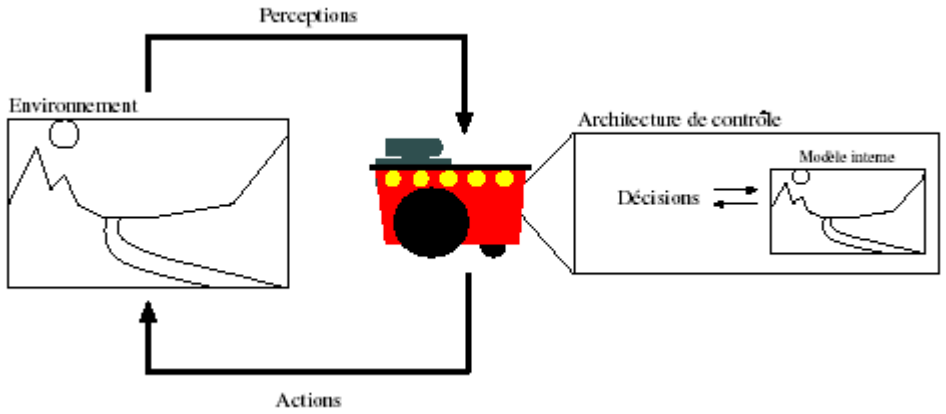


Figure 1.1 : Schéma des interactions d'un robot avec son environnement.

Cette définition s'illustre par un schéma classique des interactions d'un robot avec son environnement (voir figure 1.1).

Ces différentes interactions correspondent au cycle *Perception / Décision / Action*. La manière dont un robot gère ces différents éléments est définie par son *architecture de contrôle*, qui peut éventuellement faire appel à un *modèle interne* de l'environnement pour lui permettre alors de planifier ses actions à long terme.

2. Aperçu historique.

Le terme de robot apparaît pour la première fois dans une pièce de Karel Capek en 1920 : Rossum's Universal Robots. Il vient du tchèque 'robota' (servitude) et présente une vision des robots comme serviteurs dociles et efficaces pour réaliser les tâches pénibles mais qui déjà vont se rebeller contre leurs créateurs. La Tortue construite par GreyWalter dans les années 1950, est l'un des tout premiers robots mobiles autonomes. Grey Walter n'utilise que quelques composants analogiques, dont des tubes à vide, mais son robot est capable de se diriger vers une lumière qui

marque un but, de s'arrêter face à des obstacles et de recharger ses batteries lorsqu'il arrive dans sa niche. Toutes ces fonctions sont réalisées dans un environnement entièrement préparé, mais restent des fonctions de base qui sont toujours sujets de recherche pour les rendre de plus en plus génériques.

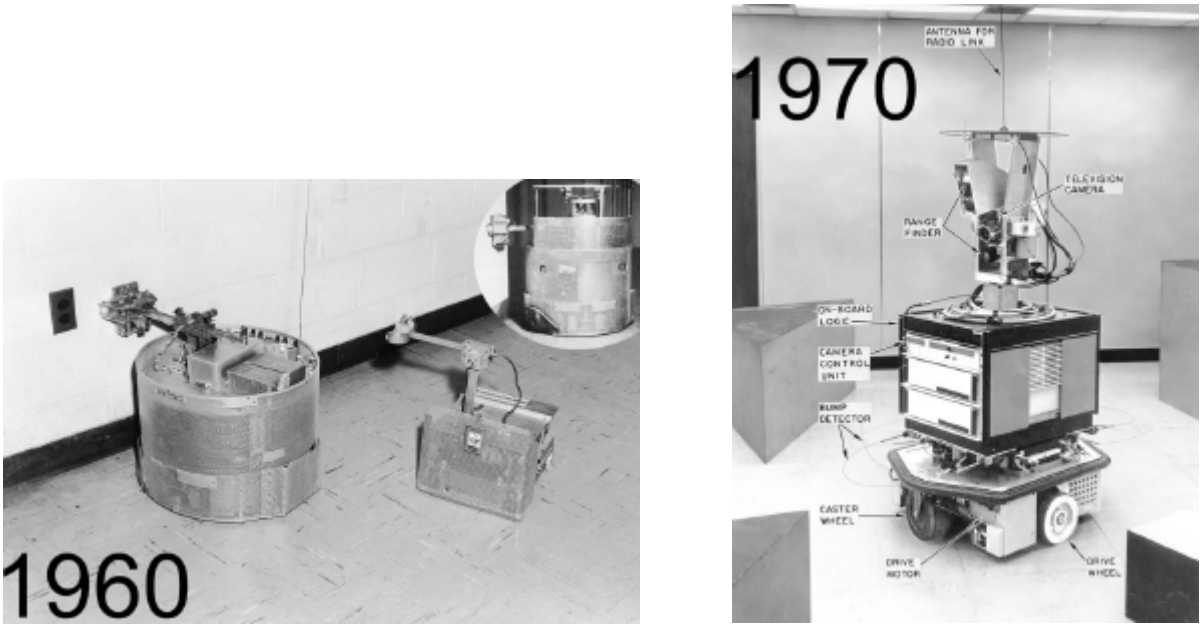


Figure 1.2 : A gauche : Robot "Beast" de l'université John Hopkins dans les années 1960. A droite : Le robot Shakey de Stanford en 1969 a été une plateforme de démonstration des recherches en intelligence artificielle.

Dans les années 60, les recherches en électronique vont conduire, avec l'apparition du transistor, à des robots plus complexes mais qui vont réaliser des tâches similaires. Ainsi le robot "Beast" (voir figure 1.2) de l'université John Hopkins est capable de se déplacer au centre des couloirs en utilisant des capteurs ultrason, de chercher des prises électriques (noires sur des murs blanc) en utilisant des photodiodes et de s'y recharger.

Les premiers liens entre la recherche en intelligence artificielle et la robotique apparaissent à Stanford en 1969 avec Shakey (figure 1.2). Ce robot utilise des télémètres à ultrason et une caméra et sert de plate-forme pour la recherche en intelligence artificielle, qui à l'époque travaille essentiellement sur des approches symboliques de la planification. La perception de l'environnement, qui à l'époque est considérée comme un problème séparé, voire secondaire, se révèle particulièrement complexe et conduit là aussi à de fortes contraintes sur l'environnement.

Ces développements se poursuivent avec le Stanford Cart dans les années 1980 (figure 1.3), avec notamment les premières utilisations de la stéréo-vision pour la détection d'obstacles et la modélisation de l'environnement.

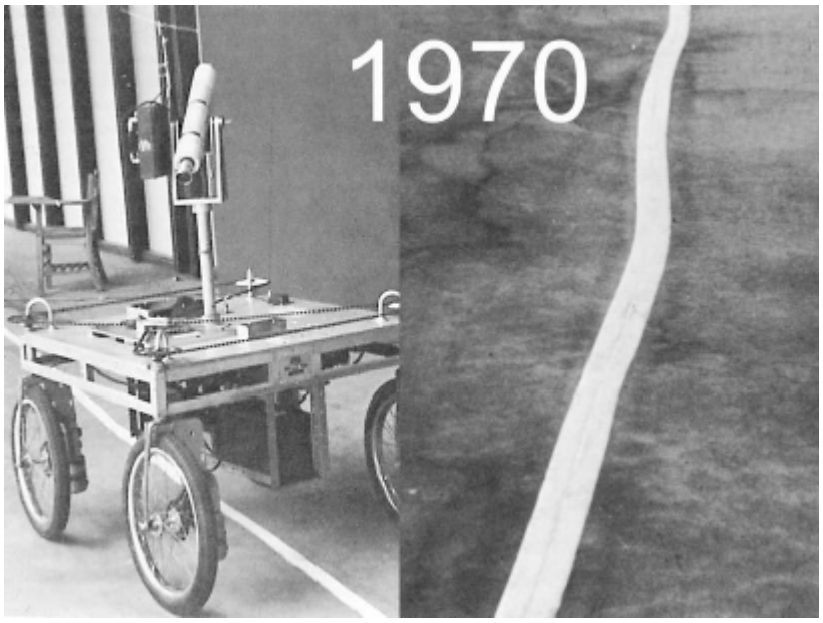


Figure1.3 : Le Stanford Cart date des années 1980.

Une étape importante est à signaler au début des années 1990 avec l'apparition de la robotique réactive, représentée notamment par Rodney Brooks. Cette nouvelle approche de la robotique, qui met la perception au centre de la problématique, a permis de passer de gros robots très lents à de petits robots, beaucoup plus réactifs et adaptés à leur environnement. Ces robots n'utilisent pas ou peu de modélisation du monde, problématique qui s'est avérée être extrêmement complexe.

Ces développements ont continué depuis et l'arrivée sur le marché à partir des années 1990 de plates-formes intégrées a permis à de très nombreux laboratoires de travailler sur la robotique mobile et à conduit à une explosion de la diversité des thèmes de recherche. Ainsi, même si les problèmes de déplacement dans l'espace restent difficiles et cruciaux, des laboratoires ont pu par exemple travailler sur des approches multi-robot, la problématique de l'apprentissage ou sur les problèmes d'interactions entre les hommes et les robots [FILL 04].

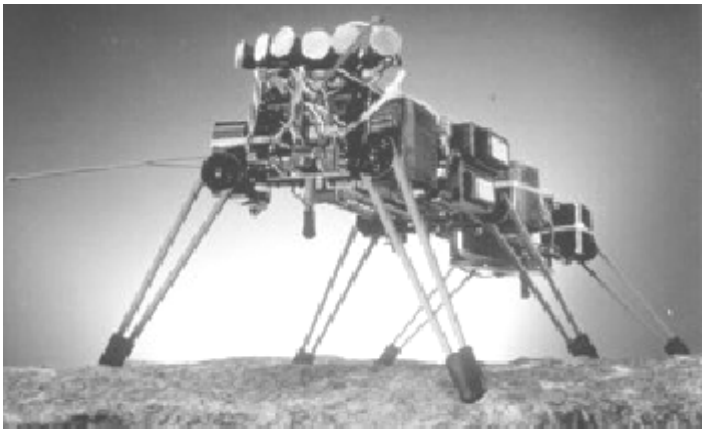


Figure 1.4 : Genghis, développé par Rodney Brooks au MIT au début des années 1990.

3. Présentation des robots mobiles autonomes.

On distingue différents types de systèmes, tels que les machines téléguidées, téléopérées ou les robots mobiles.

- Avec une *machine téléguidée*, l'homme commande directement les tensions d'entrée des moteurs de celle-ci grâce à un ou deux mini manches (joysticks). Cependant cette opération ne peut être réalisée que lorsque la machine est à vue d'œil d'un opérateur entraîné.
- L'utilisation en *téléopération* exclut la nécessité de la vue directe car l'opérateur reçoit des informations sur le comportement interne de la machine et sur son environnement : il envoie ainsi des consignes ou des commandes directes à la machine. Cependant, ce type d'opération ne peut s'effectuer que si la liaison entre l'opérateur et la machine est possible.
- Le *robot* se différencie essentiellement des autres systèmes par s*a simplicité de commande* : l'opérateur donne l'ordre au robot d'accomplir une certaine tâche à un endroit précis, alors qu'il doit communiquer aux autres systèmes des consignes (de vitesse, de cap).
- Pour sa part, un *robot mobile autonome* n'a pas besoin de communiquer avec l'opérateur, pourvu qu'il dispose de capacités de perception de l'environnement (capteurs en tous genres). Il n'est donc commandé qu'à partir d'ordres synthétiques (du type : « aller du lieu 1 au lieu 2 »). [GIR 04]

4. Mode de programmation d'un robot.

La simplification des ordres est permise soit par l'apprentissage soit par la programmation du robot [GIR 04].

- *L'apprentissage* se fait en déplaçant à la main le robot ou un appareil plus léger de même structure. Il est possible d'enregistrer tous les mouvements ou seulement quelques points particuliers de ceux-ci, l'unité de commande se chargeant d'extrapoler les mouvements intermédiaires (trajectoires et vitesses), ainsi que les arrêts.
- *La programmation au clavier* consiste à commander manuellement à l'aide de boutons ou de leviers les mouvements, et à enregistrer le nombre nécessaire de positions. Les vitesses et les arrêts sont programmables de la même façon.
- *La programmation directe* s'effectue sur des unités de calcul appelées microcontrôleurs. Elle permet le figeage de la partie commande par les informaticiens.

5. Définition opérationnelle.

Un robot mobile autonome est un *système automoteur*, disposant de moyens de *traitement de l'information* (permettant une capacité décisionnelle suffisante) et de *moyens matériels* adaptés (charge utile), de façon à pouvoir exécuter, sous contrôle humain réduit mais nécessaire, un certain nombre de tâches dans un environnement variable inconnu.

Il faut donc que le robot puisse embarquer assez d'énergie pour se déplacer et soit capable d'acquérir des informations sur son environnement et de dialoguer avec un opérateur. [GIR 04]

6. Décomposition canonique d'un robot mobile autonome.

A partir de la définition vue précédemment, on peut décomposer un robot mobile autonome en plusieurs sous-systèmes [GIR 04] :

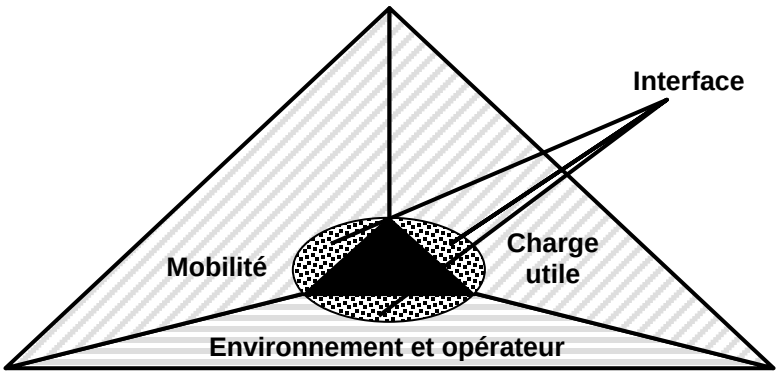


Figure 1.5 : Sous-système d'un robot mobile autonome.

Les trois principaux systèmes sont les sous-systèmes *mobilité*, *charge utile*, *contrôle/commande/communication/décision* (voir figure 1.5) :

- Le premier associe l'ensemble des éléments mécaniques et électriques permettant *l'autonomie de mouvements* nécessaires au robot. On intègre dans ce sous-système la génération d'énergie.
- Le deuxième sous-système est plus spécifique à la *mission du robot* : ce peut être une caméra, un outil de nettoyage etc.
- Le troisième sous-système regroupe l'ensemble des moyens d'*acquisition*, de *traitement*, et de *calcul* qui permettent au robot d'avoir une certaine *autonomie décisionnelle*.

7. Les problèmes posés.

7.1. La planification de trajectoire.

On voit ainsi au travers de cette première approche assez théorique apparaître un problème essentiel : *la planification de la trajectoire*. Comment en effet construire un robot capable de se mouvoir dans un environnement extérieur en décidant de sa direction ? Différentes approches sont envisageables selon que le robot évolue en milieu connu ou inconnu :

- L'évolution en territoire cartographié *simplifie* évidemment la tâche des concepteurs : une fois la carte de la zone d'évolution introduite dans la

mémoire d'un ordinateur communiquant avec le robot ou bien dans une mémoire intégrée au robot lui-même, des algorithmes de routage permettent de diriger le robot.

Il en va tout autrement dans le cas de l'évolution en territoire inconnu. Le robot doit alors *analyser* son environnement au moyen de différents *capteurs* (voir le paragraphe *Les problèmes liés à l'évitement d'obstacles*), détecter sa *position* par rapport à son but, et *décider* de sa trajectoire. Cette localisation peut s'effectuer par différentes méthodes :

- triangulation de signaux émis par des balises déposées au cours du déplacement
- repérage d'obstacles à distance, et construction d'une carte du site
- mesures odométriques (vitesse de déplacement, dérive par rapport à une position donnée) et estimation de la position.

On applique ensuite des algorithmes complexes pour diriger le robot. Ceux-ci peuvent amener des résultats plus ou moins heureux, le principal problème étant la *non-convergence* de certaines boucles de déplacement. Si aucun algorithme de secours n'a été prévu, l'*intervention humaine* est alors nécessaire. [GIR 04]

7.2. L'évitement des obstacles.

La détection et l'évitement des obstacles sont les étapes fondamentales de l'évolution d'un robot en territoire inconnu. On dispose à cet effet de plusieurs types de capteurs :

- Caméras ; un programme d'analyse des images étant alors nécessaire.
- Capteurs laser
- Capteurs infrarouges
- Capteurs à ultrasons

Une fois les obstacles repérés, le robot peut effectuer plusieurs actions, par exemple : cartographier le site sur lequel il évolue, vérifier si sa distance à l'obstacle est supérieure ou non à une distance limite, et dans le cas contraire, éviter l'obstacle… [GIR 04].

7.3. La navigation.

7.3.1. Définitions.

Un *mouvement* est une application définie en fonction du temps *t*, reliant un point initial à l'instant t_0 à un point final à l'instant t_f. Une *trajectoire* est le support d'un mouvement. Il s'agit donc d'une courbe paramétrée par une variable s quelconque. L'évolution du paramètre s en fonction du temps t est appelée *mouvement sur la trajectoire*. [BAY 06]

Le problème de navigation d'un robot mobile consiste de la manière la plus générale à trouver un mouvement dans l'*espace des configurations sans collisions*, traditionnellement noté C_{free}. Ce mouvement amène le robot d'une configuration initiale $q_0 = q(t_0)$ à une configuration finale $q_f = q(t_f)$. On peut néanmoins donner des définitions différentes de la tâche de navigation à accomplir, selon le but recherché : par exemple on peut souhaiter seulement placer le robot dans une zone donnée ou relâcher la contrainte d'orientation, etc.

La tâche de navigation ainsi définie est donc limitée à un seul mouvement. Il existe néanmoins une très grande variété de travaux et de méthodes permettant d'aborder ce problème difficile. Pour différencier les techniques de navigation, on peut de manière classique distinguer deux approches :

- la première consiste à planifier le mouvement dans l'espace des configurations et à l'exécuter par asservissement du robot sur le mouvement de consigne (*schéma planification - exécution*) ;
- la seconde consiste à offrir un ensemble de primitives plus réactives. Elles correspondent alors à des sous - tâches (suivre un mur, éviter un obstacle) dont on estime que l'enchaînement est du ressort d'un planificateur de tâches ayant décomposé la tâche globale.

7.3.2. Méthodes réactives.

Les *méthodes* dites *réactives* n'utilisent pas de phase de planification et s'apparentent le plus souvent à des méthodes de commande dédiées à tel ou tel capteur et fonctionnant dans des conditions bien particulières. Ces méthodes présentent généralement une bonne réactivité (comme leur nom l'indique) et une certaine robustesse [KHA 97].

8. Matériels courants en robotique.

8.1. Les effecteurs.

Nous présentons ici les différents types de bases mobiles utilisées en robotique [FILL 04] :

8.1.1. Les plates-formes différentielles.

Une des configurations les plus utilisées pour les robots mobiles est la configuration différentielle, elle comporte deux roues commandées indépendamment. Une ou plusieurs roues folles sont ajoutées à l'avant ou à l'arrière du robot pour assurer sa stabilité (Figure 1.6). Cette plate-forme est très simple à commander, puisqu'il suffit de spécifier les vitesses des deux roues, et permet de plus au robot de tourner sur place [FILL 04].

Figure 1.6 : Exemple de plate-forme différentielle.

Les vitesses de translation *v* et de rotation w sont en effet données par :

$$v = \frac{\omega_1 r + \omega_2 r}{2}$$
$$\omega = \frac{\omega_1 r - \omega_2 r}{2l} \qquad (1.1)$$

Où :

r : représente le rayon de la roue

l : représente la moitié de la distance entre les deux roues motrices

8.1.2. Les plates-formes omnidirectionnelles.

Les plates-formes omnidirectionnelles permettent de découpler de manière encore plus nette le contrôle de la rotation et de la translation d'un robot. Elles utilisent pour cela 3 ou quatre roues qui tournent à la même vitesse pour fournir une translation et un mécanisme qui permet d'orienter simultanément ces roues dans la direction du déplacement souhaitée (Figure 1.7).

Un robot mobile est dit *omnidirectionnel* si l'on peut agir indépendamment sur les vitesses : vitesse de translation selon les axes $\vec{x}$ et $\vec{y}$ et vitesse de rotation autour de $\vec{z}$ (voir figure 1.7).

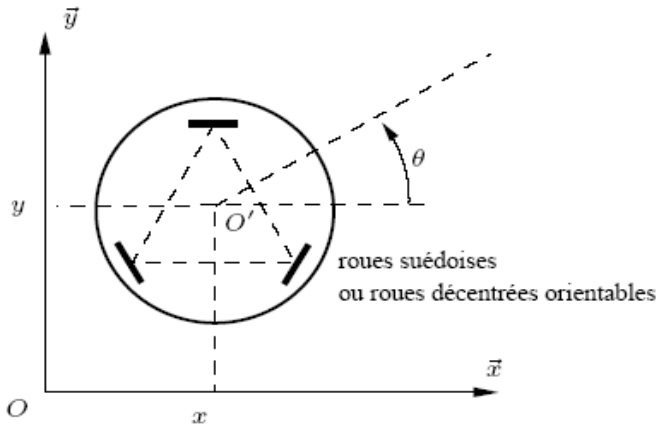


Figure 1.7 : Exemple de plate-forme omnidirectionnelle.

8.1.3. Les plates-formes type voiture.

Des plates-formes de type voiture, sont également utilisées en robotique mobile (voir figure 1.8). Ces plates-formes sont toutefois plus difficiles à commander car elle ne peuvent pas tourner sur place et doivent manœuvrer.

Figure 1.8 : Exemple de plate-forme de type voiture.

Ainsi, un système de type voiture a classiquement trois variables de configuration (voir figure 1.9) : deux pour repérer sa position dans le plan et une qui mesure son orientation.

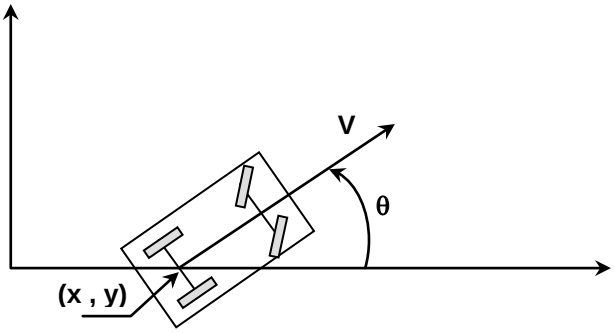


Figure 1.9 : Cinématique d'un robot de type voiture.

Où :

x, y : représente la position du robot

θ : représente l'angle d'orientation du robot

8.1.4. Les plates-formes à pattes.

Des plates-formes à deux, quatre ou six pattes peuvent également être utilisée. Les plates-formes à six pattes sont relativement pratiques car le robot est en équilibre permanent, ce qui facilite le contrôle.

8.2. Les capteurs.

Nous présentons dans cette section les capteurs les plus couramment utilisés en robotique mobile pour les besoins de la navigation [FILL 04].

8.2.1. Les capteurs proprioceptifs.

Les capteurs proprioceptifs permettent une mesure du déplacement du robot. Ce sont les capteurs que l'on peut utiliser le plus directement pour la localisation, mais ils souffrent d'une dérive au cours du temps qui ne permet pas en général de les utiliser seuls. On trouve l'odomètrie, Les systèmes radar Doppler et Les systèmes inertiels,

8.2.2. Les télémètres.

Il existe différents types de télémètres, qui permettent de mesurer la distance à l'environnement, utilisant divers principes physiques. On trouve les télémètres à ultrason, les télémètres à infrarouge et les télémètres laser [BAY 06].

8.2.3. Les cameras.

L'utilisation d'une caméra pour percevoir l'environnement est une méthode attractive car elle semble proche des méthodes utilisées par les humains. Le traitement des données volumineuses et complexes fournies par ces capteurs reste cependant difficile à l'heure actuelle, même si cela reste une voie de recherche très explorée.

8.2.4. Autres capteurs.

Il y a d'autre type de capteur comme les capteurs tactiles (ou capteur de choc), les boussoles, les balises et le GPS.

9. Groupes de robots mobiles.

L'avantage du travail collectif sur le travail individuel n'est plus à démontrer. Les exemples justificatifs ne manquent pas : travailler en groupe diminue le temps nécessaire pour effectuer une tâche dés que celle-ci est parallélisable ou encore permet d'exécuter des tâches impossibles à réaliser par un individu seul. Le fait que le travail collectif permette de dépasser les limitations propres aux membres d'un groupe est donc un des moteurs qui pousse à s'organiser et à travailler en groupe.
Cela est utilisé par les êtres humains depuis la création des premières sociétés. Ainsi, que se soit pour alléger notre charge de travail ou pour augmenter les profits d'une entreprise, nous tendons à faire travailler les robots à notre place. Or les capacités de ces derniers, bien qu'étant potentiellement largement supérieures aux nôtres, resteront toujours limitées. Malgré les progrès technologiques, un robot ne pourra jamais être à deux endroits en même temps ou être infaillible. Pour pallier à ces limitations, il est important de mettre au point des approches permettant aux robots de travailler en groupe, entre eux et avec nous [RAÎ 04].

Effectuer une tâche de manière collective implique, pour les membres du groupe, d'interagir les uns avec les autres. Le travail en groupe possède également des inconvénients : lorsqu'une tâche est partagée entre plusieurs agents, humains ou artificiels, il apparaît inévitablement des conflits entre eux au niveau des ressources qu'ils partagent. Par exemple comme deux agents ne peuvent pas être au même endroit au même moment, le partage de l'espace où s'effectue la tâche est une source de conflit.

9.1. Formation de groupe de robots mobiles.

Trois catégories de caractéristiques peuvent être associées aux diverses stratégies utilisées pour réaliser une formation de robots [LEM 03].

9.1.1. Perception.

Cette catégorie permet de caractériser une approche par la manière dont les robots perçoivent leur environnement. Comme caractéristiques de perception affectant les approches de formations, il y a tout d'abord la visibilité des robots entre eux. Cette visibilité peut être *complète* (chaque robot sait où les autres

robots se trouvent) ou *limitée* (chaque robot perçoit, via ses capteurs, les robots à proximité).

Le système de référence affecte aussi les perceptions des robots : dans un système référentiel *absolu*, la position des robots dans leur environnement s'évalue par rapport à un seul point commun, tandis que dans un système référentiel *relatif*, chaque robot évalue la situation par rapport à sa propre position dans l'environnement. Enfin, les perceptions des robots sont aussi affectées par la présence ou non d'informations communiquées. Ces informations peuvent être *globales* (informations portant sur l'ensemble de la formation) ou *locales* (informations propres à un sous-groupe de robots en particulier).

9.1.2. Formation.

Cette catégorie regroupe les caractéristiques des formations que le groupe de robots peut réaliser. Ces formations se distinguent par trois facteurs importants. D'une part, nous retrouvons les types de formations qu'un groupe de robots peut effectuer. Parmi toutes les formations arbitraires possibles, nous retrouvons les types de formations : *centrées* (le conducteur est positionné dans le milieu de la formation) tel que les formations en cercle, en diamant, en coin (*wedge*), en ligne, en triangle, en rectangle, en flèche, en hexagone ; *non-centrées* (le conducteur est positionné à une extrémité de la formation) tel que la formation en colonne. D'autre part, la référence de positionnement utilisée par chacun des robots pour que la formation se réalise est une autre caractéristique de la formation.

Trois façons de faire se démarquent : le positionnement *centre-référencé* signifie que les robots dans le groupe se positionnent par rapport au centre de la formation ; le positionnement *point-référencé* veut dire que les robots se positionnent par rapport à un point quelconque (tel le conducteur de la formation ou un point «virtuel » associé à la formation) dans leur environnement; le positionnement *voisin-référencé* où chaque robot maintient sa position dans la formation par rapport à un ou deux robots à proximité. Finalement, il est possible de différencier les formations par leurs contraintes de structure, à savoir si la formation est *rigide* (c'est-à-dire qu'elle ne peut pas se déformer en cas de contraintes d'espace ou d'obstacles) ou encore *flexible* (c'est-à-dire

qu'elle permet au groupe de robots de déformer sa formation afin de s'ajuster aux différentes contraintes de l'environnement).

9.1.3. Contrôle.

Cette catégorie caractérise le processus de décision pour la mise en place de la formation. Ce processus peut être *centralisé* (c'est-à-dire qu'un seul robot prend les décisions) ou *distribué* (tous les robots participent à la prise de décision). La catégorie caractérise aussi la dépendance au temps pour le contrôle de la formation, où les robots peuvent être *avec mémoire* (les robots sont influencés par ce qu'ils ont fait dans le passé) ou *sans mémoire* (les robots ne tiennent pas compte de leurs décisions antérieures). Finalement, la stratégie utilisée pour contrôler un groupe de robots peut être basée sur des *lois de contrôle*, sur une approche de type *comportementale* ou encore sur une stratégie *hybride* (une stratégie combinant plusieurs mécanismes de contrôle).

Trois autres termes distincts sont utilisés pour identifier les différents comportements qu'auront les robots selon leur position dans la formation.

Conducteur. Le conducteur est le robot qui se trouve à la tête du groupe de robots mobiles lorsqu'ils sont en formation. Le conducteur mène la formation à l'endroit désiré.

Meneur. Le meneur est un robot qui guide celui qui le suit, nommé suiveur. Autre que pour le conducteur, un meneur est aussi un suiveur pour le robot qui le précède dans la formation.

Suiveur. Le suiveur est n'importe quel robot qui n'est pas à la tête d'un groupe de robots en formation. Un suiveur a comme rôle de suivre son meneur à une certaine distance et à un certain angle par rapport à son meneur, selon la formation désirée et sa position dans la formation.

10. Conclusion.

Après cette présentation de la robotique mobile autonome avec tous ses différents problèmes posés tel que la planification de trajectoire, l'évitement d'obstacles ainsi que les différents types de plateformes existantes ; l'ossature de départ de notre plateforme expérimentale comporte des robots mobiles autonomes circulaires étagés, répondant au principe de la locomotion différentielle, qui est la structure la plus répandue dans le domaine de la robotique mobile. Nous avons opté pour cette structure pour sa simplicité tant dans sa réalisation mécanique que dans sa commande. En effet, le contrôle du déplacement et de l'orientation est étroitement lié au contrôle des vitesses des deux roues motrices. Ces robots mobiles sont contrôlés par un microcontrôleur *PIC* ; et ils seront alimentés par des batteries 12 V ; les moteurs utilisés sont des moteurs industriels à courant continu.

La partie principale de raisonnement de déplacement se fera par le biais d'une méthode mixte associant :

- Une méthode globale permettant de planifier une trajectoire basée sur l'utilisation des champs de potentiel et sera développée en détail dans le chapitre suivant.

- Une méthode locale basée sur :
  - Une partie matérielle constituée de trois balises infrarouges actives fixées dans l'environnement dans lequel évoluent les robots mobiles et d'une tourelle comportant un récepteur infrarouge commandé par un moteur Pas à Pas.
  - Une partie logicielle bâtie autour d'un algorithme de triangulation chargé dans la mémoire flash du microcontrôleur ; ce système de localisation est développé dans le chapitre cinq.
  - Un module d'évitement d'obstacle qui est bâti autour d'un radar infrarouge qui détecte en continu la présence d'obstacle et qui avertie le robot en conséquence.

	Introduction générale.
Chapitre 1 :	Etat de l'art sur la robotique mobile et sur la coopération entre robots.
Chapitre 2 :	Planification de trajectoire et navigation.
Chapitre 3 :	Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes.
Chapitre 4 :	Implémentation des processus parallèles à l'aide du langage de programmation ADA.
Chapitre 5 :	Conception d'une plateforme de robots mobiles autonomes et début de réalisation.
Chapitre 6 :	Simulation.
	Conclusion générale.
	Bibliographie.
	Annexe et Appendice.

Chapitre 2.

Planification de trajectoire et navigation

1. Méthodes de planification de trajectoire.

1.1. Présentation du problème.

Parmi les principales causes limitant le développement des systèmes robotisés, et en particulier dan le cas de robotique mobile, on trouve la complexité de la programmation. Cette complexité provient du nombre élevé de cas potentiels auxquels peut être confronté le robot. En robotique mobile, le problème est d'autant plus ardu que les déplacements sont importants et que les obstacles sont nombreux.

La planification des mouvements d'un robot mobile autonome, c'est-à-dire l'établissement de la liste des opérations à effectuer pour aller d'un lieu donné à un autre lieu défini comme but, doit se faire sans intervention humaine. La notion de planification de trajectoire sans collision peut être formulée de la manière suivante :

*Soit un système robotique **R** et un environnement **E** ; il faut déterminer*
*une trajectoire **T** permettant à **R** de se déplacer de la position initiale **Pi***
*à la position finale **Pf** en évitant toute collision avec **E***

La trajectoire correspond à une suite continue de situations géométriques successivement occupées par le robot durant son déplacement.

Pour cela il est nécessaire de fournir au robot un descriptif aussi précis que possible de l'environnement dans lequel il évolue. Les informations que l'homme recueille sur l'environnement sont bien entendu beaucoup trop nombreuses pour qu'elles puissent être communiquées ainsi au robot. Par conséquent, il est nécessaire dans un premier temps d'effectuer un filtrage de ces informations afin de ne communiquer que les informations pertinentes en extrayant les points importants. Il faut donc définir aussi complètement que possible l'univers du robot en minimisant les informations à mémoriser.

L'univers d'un robot comprend le terrain sur lequel il évolue, les objets ou obstacles qui l'entourent et bien entendu le robot lui-même. Pour tout robot mobile évoluant dans un environnement dynamique, le sol est généralement modélisé par un plan horizontal ou par un plan incliné, auquel est attaché un repère de référence. Les

obstacles sont représentés par une projection sur le sol de tout le volume de l'obstacle qui pourrait interférer avec le robot. La représentation des obstacles dépend donc du robot considéré comme c'est présenté sur la figure 2.1.

La planification doit générer l'évitement d'obstacles. Pour cela, il faut commencer par définir un itinéraire. L'itinéraire est une suite discontinue de positions que doit occuper le robot au cours de son déplacement. Le calcul de la trajectoire sera alors une procédure itérative qui s'appliquera entre deux points de passages sur l'itinéraire (figure 2.1).

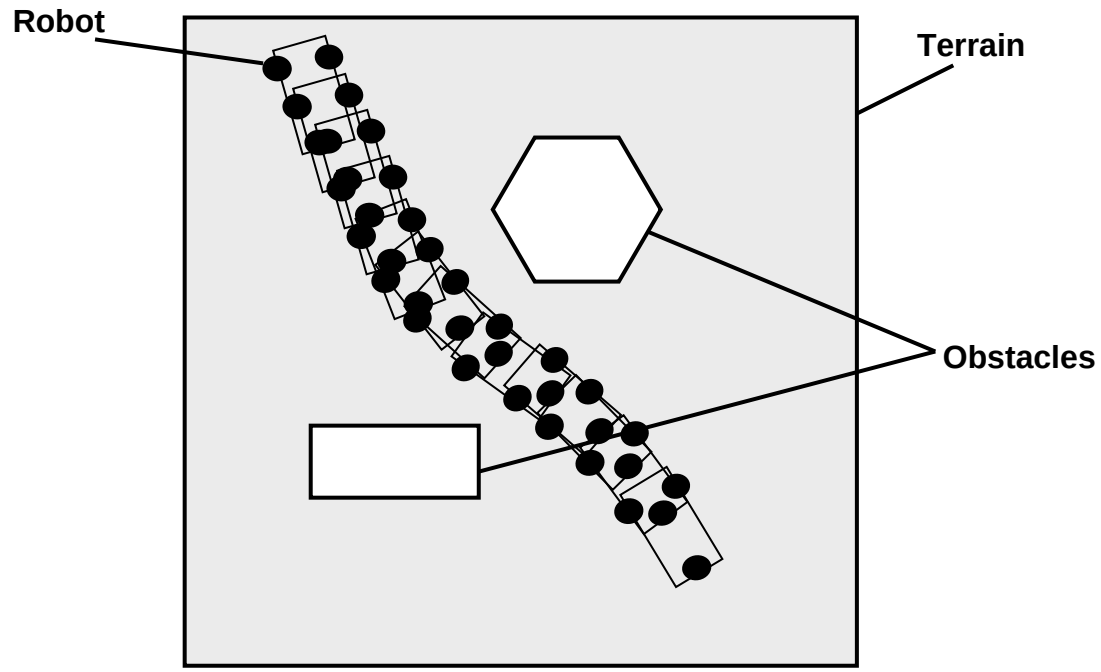


Figure 2.1 : Univers du robot.

La navigation, ou détermination de l'itinéraire et de la trajectoire, constitue un des grands problèmes de la robotique mobile. Parmi les solutions existantes :

- Une méthode globale supposant connu l'ensemble de l'univers d'évolution du robot.
- Et une méthode locale qui génère un déplacement à partir des informations recueillies par le robot sur son environnement proche.

Dans les deux cas, il sera nécessaire de modéliser les connaissances propres du robot. En général, on trouve dans ces modèles une partition entre les lieux occupés par les obstacles et les lieux qualifiés de libre et permettant aux robots de circuler.

1.2. Méthodes globales.

Elles disposent pour leur recherche d'une représentation globale du monde.
Ces méthodes nécessitent la construction d'un espace de recherche qui permet l'obtention implicite d'un graphe valué. A l'aide de ce graphe, une recherche heuristique permet de trouver le trajet qui minimise les fonctions de coût utilisées.

1.2.1. Définitions de l'espace des configurations.

La notion d'espace des configurations a été introduite par Udupa [UDU 77] qui a fourni l'un des premiers algorithmes de planification de trajectoires sans collision pour un robot. Elle a ensuite été généralisée et formalisée par Sharir-Schwartz [SCH 83], Elle consiste à transformer l'environnement, soit dans sa totalité c'est-à-dire espace libre et espace occupé, soit partiellement (espace libre) en un ensemble de cellules discrètes connexes, l'espace libre est alors caractérisé par un graphe, chaque noeud étant une cellule. Donc au lieu de trouver un chemin dans l'espace des configurations, il suffit de le trouver dans un graphe.

1.3. Méthodes locales.

Par opposition aux méthodes globales, les méthodes locales n'utilisent pas le modèle complet de l'environnement, mais l'environnement immédiat du robot (les obstacles « visibles ») le plus souvent obtenu à partir des informations des capteurs.

Ces méthodes utilisent des propriétés locales de l'environnement (distance aux obstacles, potentiels…) pour trouver localement un chemin. Elles peuvent être qualifiées de « *méthodes orientées pilotage* », car elles ne permettent pas l'optimisation du chemin.

Leur principe consiste à engendrer progressivement des incrémentations locales sur les contraintes imposées par l'environnement.

Une des versions les plus simples de ce type d'approche correspond à des méthodes de type « génère, test et corrige ». Ces méthodes consistent, après avoir fait l'hypothèse d'une trajectoire simple, à la suivre tant que le système ne détecte pas de risques de collision avec les obstacles. L'utilisation d'heuristiques appropriées

permet de corriger localement la trajectoire, la méthode est appliquée récursivement jusqu'à la convergence vers la position désirée.

On reconnaît à ces méthodes les avantages suivants :

- Rapidité pour l'obtention d'une solution,
- Possibilité de prendre en compte rapidement les obstacles imprévus,
- Obtention d'une trajectoire continue (non segmentée) qu'il n'est pas nécessaire de lisser.

Cependant, il nous faut remarquer que dans certains cas de figures ces méthodes échouent alors qu'une solution existe.

1.3.1. Méthodes des potentiels fictifs.

L'utilisation de champs potentiels a été développée principalement et indépendamment par Khatib [KHA 86] et Krogh [KRO 86].

Dans la méthode d'évitement d'obstacles par champs de potentiels, le robot est assimilé à une particule se déplaçant suivant les lignes de courant d'un potentiel créé en fonction de l'environnement perçu par ce dernier. Ce potentiel traduit différents objectifs tels que l'évitement d'obstacles ou une direction de déplacement préférée. Il est calculé par sommation de différentes primitives de potentiels traduisant chacun de ces objectifs (Figure 2.2). Ces différents potentiels peuvent avoir une étendue spatiale limitée ou non (par exemple, n'avoir une influence que près des obstacles) et leur intensité peut dépendre ou non de la distance.
Le gradient de ce potentiel donne, en chaque point de l'espace, la direction de déplacement du robot (Figure 2.2). Comme c'est ce gradient, et non la valeur absolue du potentiel, qui nous intéresse, il est possible de calculer directement en chaque point sa valeur par une simple somme vectorielle en ajoutant les valeurs issues des différents potentiels primitifs. Ainsi, pour un robot se déplaçant en ligne droite en espace ouvert et évitant les obstacles qu'il peut rencontrer, nous obtenons par exemple les lignes de courant illustrées figure 2.3.

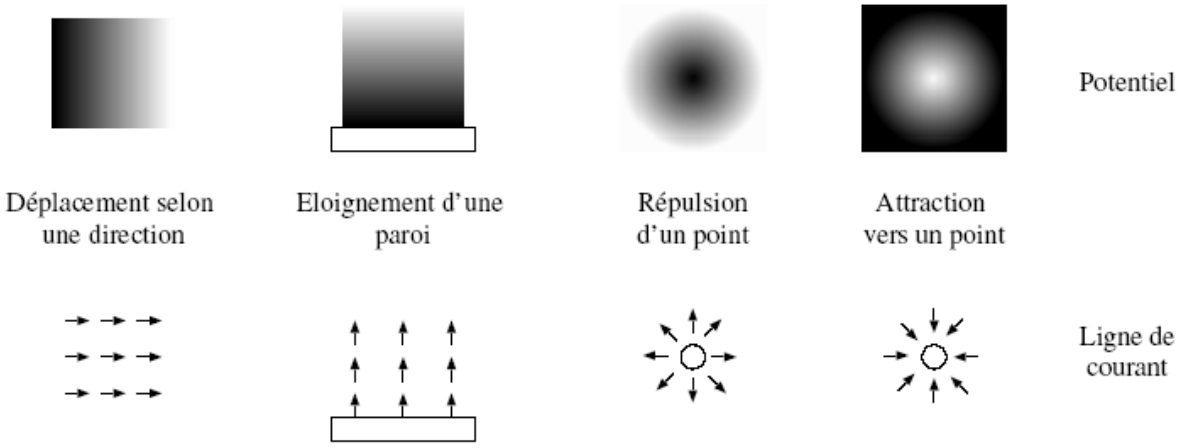


Figure 2.2 : Illustration de potentiels primitifs dont la combinaison guide les déplacements du robot. Le robot se déplacera selon les lignes de courant.

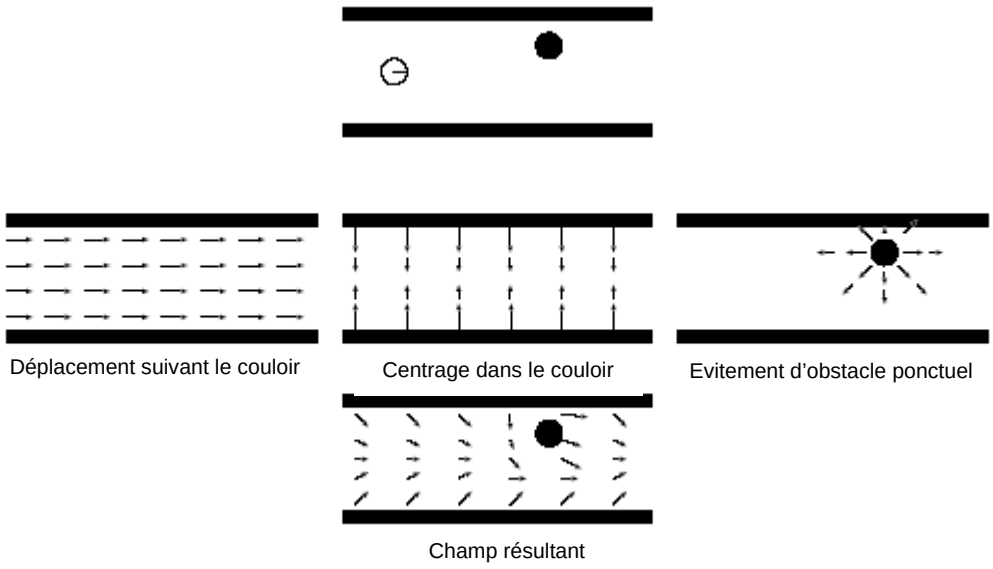


Figure 2.3 : Illustration de la combinaison de différents potentiels primitifs.

De plus, dans la pratique, pour l'évitement d'obstacles, le potentiel est en général calculé dans l'espace relatif au robot et ne sert qu'à décider de la vitesse et de la direction courante. Il n'est donc nécessaire de l'estimer que pour la position courante du robot, en sommant simplement la contribution des différents éléments perçus (Figure 2.4).

Le principal inconvénient de cette méthode d'évitement d'obstacles est l'existence, pour certaines configurations d'obstacles (relativement courantes) de minimum locaux du potentiel qui ne permettent pas de décider de la direction à prendre

(Figure 2.5). Ce problème peut être traité de différentes façons. Il est par exemple possible de déclencher un comportement particulier lorsque l'on rencontre un tel minimum (déplacement aléatoire, suivi de murs ....). Il est aussi possible d'imposer que le potentiel calculé soit une fonction harmonique, ce qui garanti qu'il n'ait pas de minima, mais complexifie beaucoup son calcul.

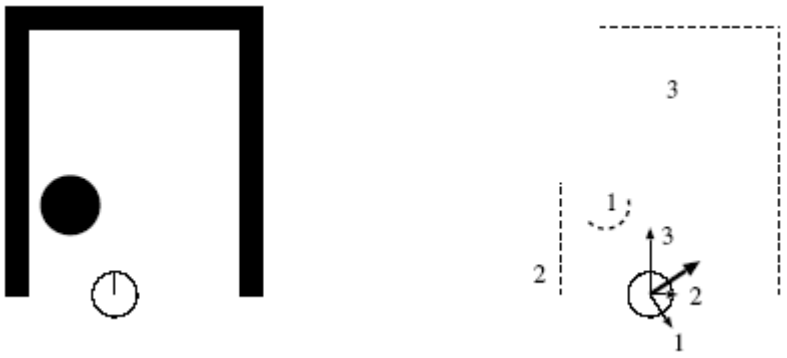


Figure 2.4 : Illustration de la combinaison de différents potentiels primitifs dans l'espace relatif au robot.

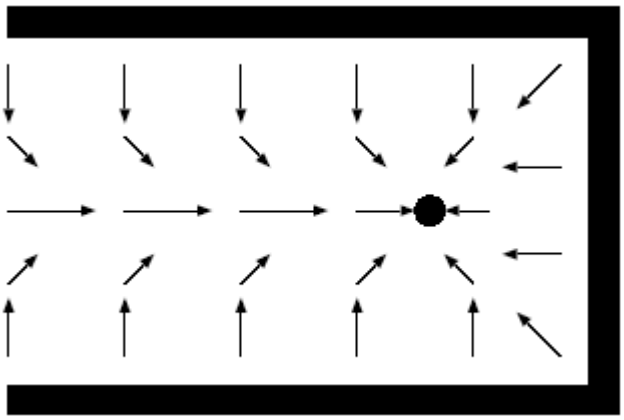


Figure 2.5 : Exemple de minimum local dans un champ de potentiel.

1.3.2. Méthodes des contraintes.

L'essentiel de cette méthode consiste à traduire les interactions entre solides susceptibles d'entrer en collision, en des contraintes sur les déplacement (vitesse, orientation) exprimées dans l'espace de configuration du système.

Cette méthode permet de séparer clairement d'une part la description de la tâche, qui consiste en un ensemble de mesures du problème à minimiser, et éventuellement le respect de contraintes géométriques, et d'autre part les contraintes anti-collision.

La distance entre les solides ne doit pas décroître trop rapidement lorsque celle-ci est inférieure à la somme des distances d'influence de chacun des solides.

A l'approche d'un obstacle, le champ de potentiel repoussera le robot, tandis que le robot commandé par la méthode des contraintes reste passif. La méthode des contraintes permet d'éviter les phénomènes de rebond sur les obstacles successifs observés dans un champ de potentiel.

1.4. Vers des méthodes mixtes.

Barraquand [BAR 91] et Latombe [LAT 99] proposent une approche qui combine une méthode locale basée sur les champs de potentiel avec une représentation hiérarchique par « discrétisation ». où l'environnement est modélisé par une carte 2D de potentiel avec un seul minimum (Cellule-cible) associée à une table de distance. Chaque cellule disponible à la navigation est modélisée par une structure de données comportant deux champs : le premier contient le niveau de potentiel par rapport à la cellule cible et le deuxième comporte la distance de cette cellule par rapport à l'obstacle le plus proche. Les deux taches demandées à un générateur de trajectoire sont : faire évoluer le robot mobile vers la cellule cible et éviter les obstacles. Ceci mène souvent à des situations de blocage.

L'approche que nous avons développé s'intéresse au problème difficile de l'établissement d'une carte 2D unique de potentiel de l'environnement complet, sans minimum local, excepté le point-cible, en intégrant les obstacles. Elle permet d'avoir une planification globale de trajectoire (descente du gradient) basée sur l'utilisation des champs de potentiel sans aucun risque de blocage.

C'est un problème d'un grand intérêt et qui reste d'actualité. Nous avons développé deux algorithmes, ils permettent d'éviter les conflits générés par l'agencement des deux tâches : progression vers la cellule cible et évitement d'obstacles [HOU 05].

Le premier algorithme est basé sur la définition des points caractéristiques de l'environnement et du graphe de visibilité, alors que le second algorithme exploite les notions de voisinage et de contournement d'obstacles.

Pour la modélisation de l'environnement et la planification de trajectoire nous adoptons les hypothèses suivantes ; figure 2.6 :

- L'environnement dans lequel évolue le RMA est divisé en petites cellules carrées ayant une taille fixe en relation directe avec la taille du robot.
- Le centre de chaque cellule est localisé par deux coordonnées x et y
- L'environnement est structuré en deux sous-espaces l'un contenant les cellules libres et disponibles pour la navigation, l'autre contenant les cellules occupées par les obstacles et donc indisponibles pour la navigation
- Une cellule occupée partiellement est considérée comme totalement occupée et une cellule exempte de tout obstacle est considérée comme libre
- Chaque cellule a quatre cellules voisines
- La trajectoire optimale proposée tient compte des aspects géométriques des mouvements et néglige les aspects dynamiques et de contrôle.

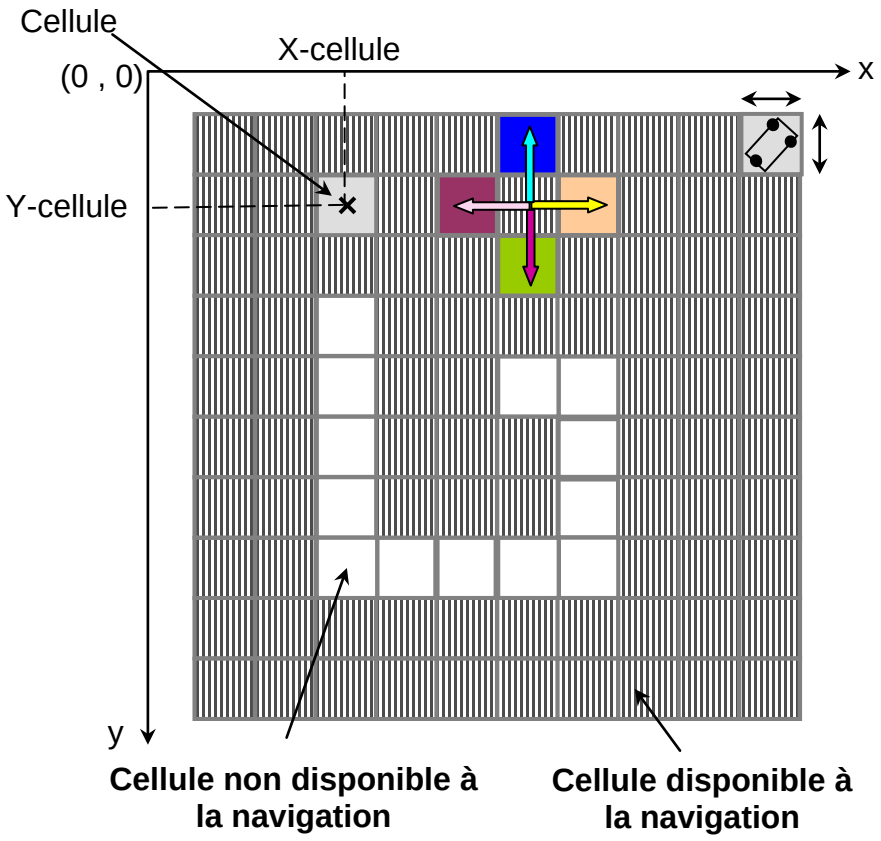


Figure 2.6 : Modélisation de l'environnement.

2. Algorithme basé sur la définition des points caractéristiques de l'environnement.

Nous considérons uniquement le cas des obstacles polygonaux. On sait pertinemment que cela n'entraîne aucune perte de généralité par rapport au cas plus réaliste d'un environnement courbe [LAV 99].

2.1. Définition des points caractéristiques de l'environnement.

Soit ***O*** un obstacle de l'environnement et ***A = { a_i}*** l'ensemble des arrêtes de ***O*** et ***S = { s_i}*** l'ensemble des sommets de ***O***.

On dit que le sommet ***s_i*** de ***O*** est un point caractéristique de l'environnement si l'angle interne de ***O*** formé par les arrêtes ***a_i*** *et* ***a_{i-1}*** est inférieur à π ; figure (2.7)

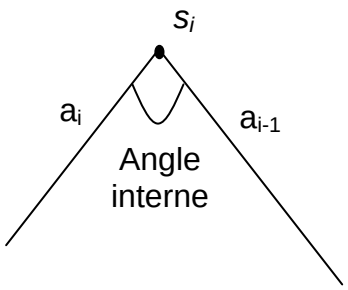


Figure 2.7: Angle interne d'un obstacle.

La figure 2.8 présente les points caractéristiques d'un obstacle polygonal.

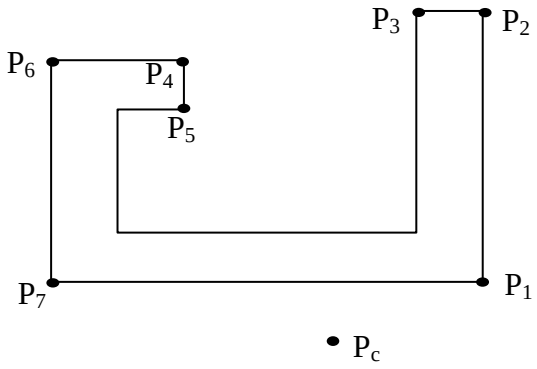


Figure 2.8 : Identification des points caractéristiques d'un environnement.

Le point cible est un point caractéristique particulier de l'environnement. Il est le seul point dont le niveau de potentiel est connu à priori, il est porté à ***zéro*** et c'est le seul minimum admis.

L'état de l'environnement en un point caractéristique peut être modélisé par l'ensemble des points caractéristiques visibles en ce point d'une part et auquel on associe un niveau de potentiel d'autre part.

2.2. Etablissement du niveau de potentiel des points caractéristiques.

Le graphe de visibilité fini G_v dont les nœuds sont les points caractéristiques est généré à partir du point-cible. A partir d'un point caractéristique donné on crée une arête le reliant à un autre point caractéristique chaque fois que ce dernier est visible depuis le point considéré. Ces arêtes sont valuées par la distance séparant les deux points caractéristiques.

Le graphe G_v (G_v = [X , E]) est un graphe connexe orienté tel que $X = \{P_c, P_1, \ldots, P_n\}$ sont les n + 1 points caractéristiques et $E = \{e_1, \ldots, e_m\}$ sont les m arêtes.

La figure 2.9 présente le graphe de visibilité de l'obstacle de la figure 2.8.

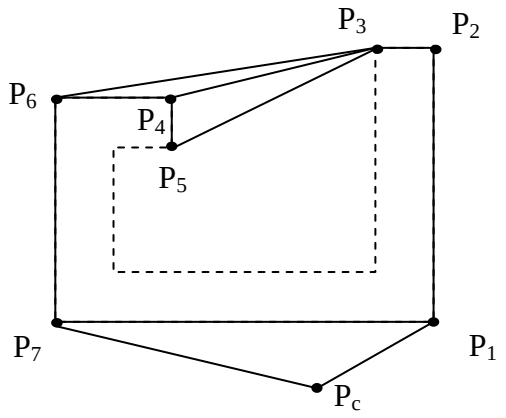


Figure 2.9 : Graphe de visibilité.

A partir du graphe de visibilité et du potentiel du point - cible, on crée le graphe orienté de diffusion du potentiel.

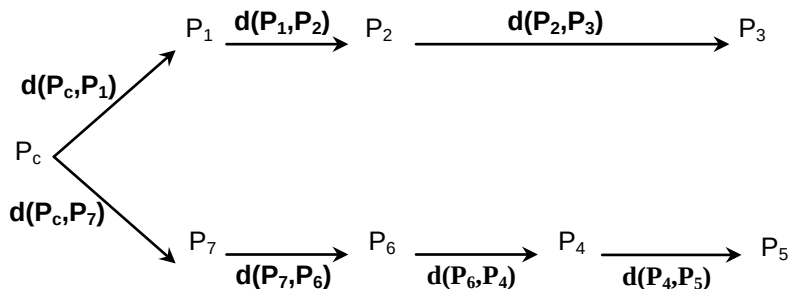


Figure 2.10 : Diffusion du potentiel du point-cible vers tous les points caractéristiques.

L'algorithme :

Initialement le potentiel du point-cible ***(Pot (P_c))*** est égale à zéro et le potentiel d'un point caractéristique quelconque est porté à l'infini.

Pot (P_c) = 0

Pot (P_i) = ∞ ; i = 1,… ,n : nbr de points caractéristiques

La diffusion du potentiel se fait à partir du point-cible de la manière suivante :

Pot (P_i) = Pot (P_c) + d (P_c , P_i)

Et P_i est inséré dans une file d'attente

avec i = 1, …, l_c nombre de points caractéristiques visibles à partir de P_c.

L'algorithme se poursuit en extrayant de la file d'attente le point caractéristique placé en tête et en l'autorisant à diffuser son potentiel. Les points caractéristiques visibles à partir de ce point et qui ont vu leur niveau de potentiel se modifier sont insérer à leur tour dans la file d'attente.

L'algorithme s'arrête lorsque la file d'attente ne contient plus aucun élément en attente de diffusion.

Cette file d'attente est gérée avec un système de priorité privilégiant les points caractéristiques ayant le plus faible niveau de potentiel [GON 95].

2.3. Etablissement de la carte 2D de potentiel.

De même, l'état de l'environnement en une cellule disponible pour la navigation est modélisé par une structure de données comprenant l'ensemble des points caractéristiques visibles en cette cellule et de sa valeur de potentiel.

Le potentiel de la cellule C_k ($Pot(C_k)$) est donné par :

$$Pot(C_k) = \min_i \left(Pot(P_i) + d(P_i, C_k) \right) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.1)$$

$i = 1, \ldots, l_{Ck}$ nombre de points caractéristiques visibles à partir de C_k

3. Algorithme basé sur la notion de voisinage et contournement d'obstacles.

3.1. Notion de voisinage et de distance.

Chaque cellule disponible pour la navigation va diffuser son potentiel sur ses quatre voisins figure 2.11.

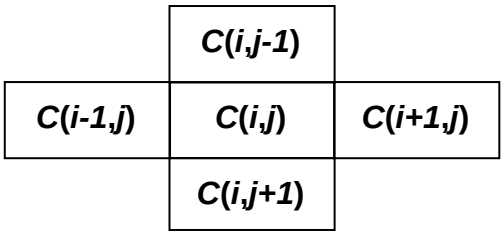


Figure 2.11 : Notion de voisinage.

Et la distance séparant deux cellules voisines est égales à 1.

3.2. Diffusion du potentiel.

Un arbre hiérarchique se développe au fur et à mesure de l'exécution de l'algorithme et le premier nœud de l'arbre correspond à la cellule-cible figure 2.12.

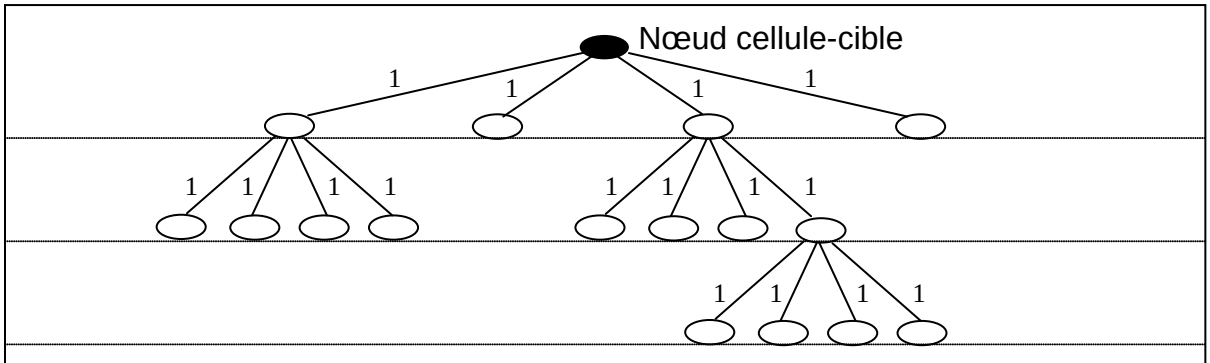


Figure 2.12 : Arbre hiérarchique de diffusion de potentiel.

Chaque cellule donne naissance à quatre nœuds fils de niveau hiérarchique inférieur, l'estimation du potentiel dans une cellule est égale au potentiel du nœud père incrémenté d'une unité.

Cette technique de diffusion de potentiel appelé aussi *Wave front propagation* procède à l'aide d'expansion par vagues et contournement d'obstacles de la manière suivante :

Initialement le potentiel de la cellule cible est porté au potentiel **0** et le reste des cellules disponibles à la navigation est porté au potentiel infini **(∞)**.

Un nœud ne donnera pas de nœud fils lorsqu'il correspond soit à une cellule non disponible à la navigation ou bien à une cellule dont le potentiel est inférieur ou égale à celui du nœud père incrémenté d'une unité.

La procédure continue jusqu'à un certain niveau où aucun nœud ne donnera de nœuds fils.

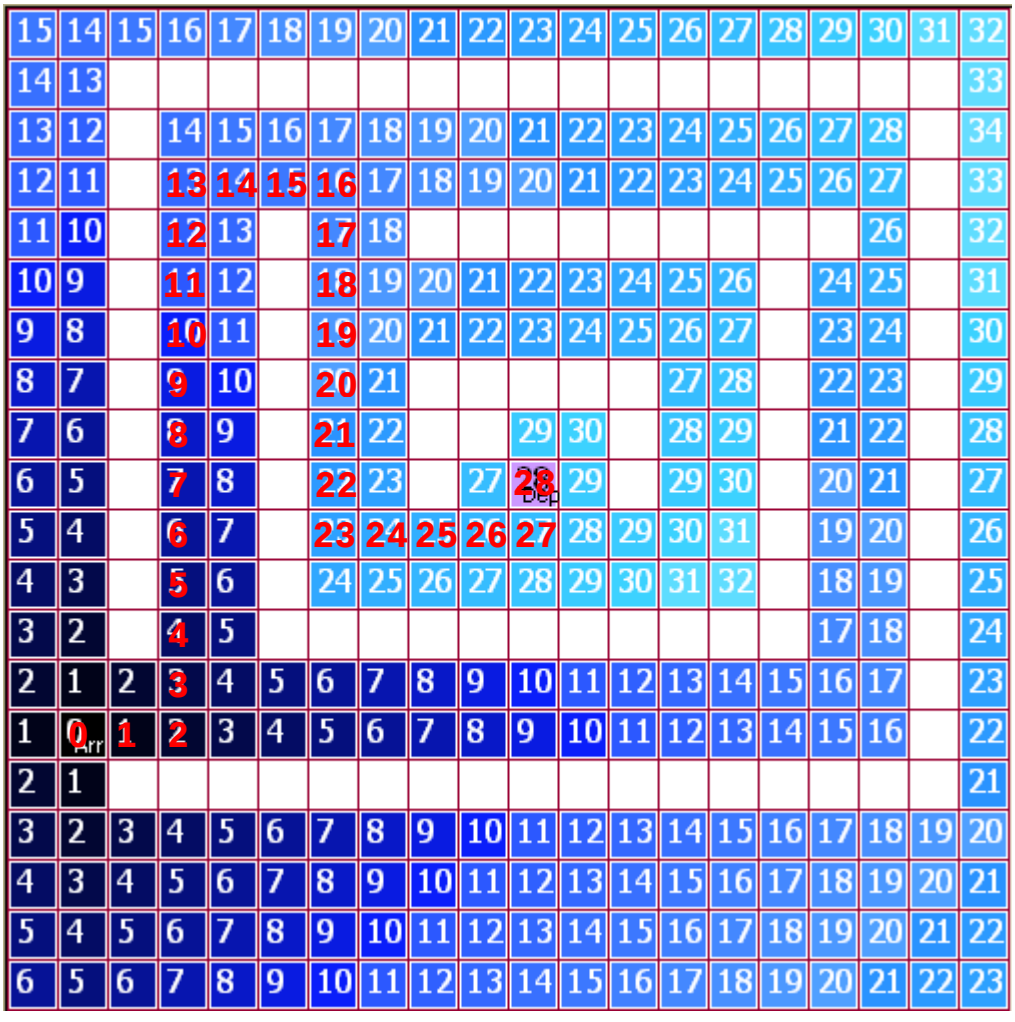


Figure 2.13 : Diffusion de potentiel et choix de la trajectoire.

Sur la figure 2.13 est présenté un exemple d'obstacle de forme labyrinthe avec le point de départ au milieu de ce dernier et le point cible à l'extérieur. Après diffusion de potentiel sur tout l'environnement on remarque qu'il existe un chemin (une solution au problème) liant le point de départ au point d'arrivée et ceci en réalisant une descente du gradient, c'est-à-dire quelque soit la configuration de l'espace de travail dans lequel le robot évolue on abouti toujours à une solution si elle existe.

4. Conclusion.

Ces deux algorithmes donnent des résultats similaires. Une simulation avec un commentaire des résultats obtenus est présentée dans le chapitre 6.

Les méthodes mixtes associant une méthode globale pouvant fournir une trajectoire avec une méthode locale pouvant générer des stratégies de contournement d'obstacles ont actuellement la préférence des chercheurs et sont de plus en plus utilisées dans les plates formes de robotiques mobiles.

	Introduction générale.
Chapitre 1 :	Etat de l'art sur la robotique mobile et sur la coopération entre robots.
Chapitre 2 :	Planification de trajectoire et navigation.
Chapitre 3 :	Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes.
Chapitre 4 :	Implémentation des processus parallèles à l'aide du langage de programmation ADA.
Chapitre 5 :	Conception d'une plateforme de robots mobiles autonomes et début de réalisation.
Chapitre 6 :	Simulation.
	Conclusion générale.
	Bibliographie.
	Annexe et Appendice.

Chapitre 3.

Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes.

1. Introduction.

Dans une plate forme de robotique mobile, l'ordonnancement des déplacements des robots, ainsi que la synchronisation des rendez-vous demeurent des problèmes cruciaux, d'où la nécessité de maîtriser parfaitement toutes les phases de conception et notamment celle dite préliminaire dans laquelle on inclut les spécifications fonctionnelles, la modélisation et l'évaluation du comportement du système est primordiale. Elle permet d'éviter d'apporter des corrections, à grand frais, à des erreurs se soldant par des dysfonctionnements [HAD 91].

De nombreux outils existent pour aider le concepteur dans cette phase critique ; a notre connaissance les réseaux de Pétri sont le seul ensemble d'outils qui permet d'intégrer à la fois : la spécification fonctionnelle, la modélisation et l'évaluation. Si on rajoute à cela la force d'expression du support graphique qu'il offre, il devient tout simplement irremplaçable.

L'ensemble des outils offert par les réseaux de Pétri va des réseaux de Pétri élémentaires très riches en propriétés mathématiques jusqu'aux réseaux de Pétri colorés et à Prédicats offrants une grande puissance dans la simulation.

Une partie de ce chapitre est dédiée à la présentation des réseaux de Pétri et de certaines de leurs propriétés notamment celles qui leurs confèrent une puissance d'analyse aussi bien structurelle que comportementale.

2. Définition de base.

Un réseau de Pétri (RdP) est un graphe biparti fait de deux types de sommets : les places et les transitions. Des arcs orientés relient certaines places à certaines transitions, ou certaines transitions à certaines places. Un arc ne relie jamais deux sommets de même nature. Généralement, les places sont représentées par des cercles et les transitions par des rectangles (ou des barres). Chaque place peut contenir un ou plusieurs jetons, représentés par des points. Ces jetons vont permettre de modéliser la dynamique du système [PRO 95].

De manière plus formelle, nous dirons qu'un réseau de Pétri est un 5-uple $N = \left(P, T, Pré, Post, M_0\right)$ où :

$P = \left\{p_1, p_2, \ldots, p_n\right\}$ est un ensemble fini de places,

$T = \left\{t_1, t_2, \ldots, t_q\right\}$ est un ensemble fini de transitions,

$Pré : \left(P \times T\right) \rightarrow N$ est une application incidence avant

$Post : \left(P \times T\right) \rightarrow N$ est une application incidence arrière

$M_0 : P \rightarrow \left\{0, 1, 2, \ldots\right\}$ est le marquage initial.

Notons que $P \cap T = \varnothing$.

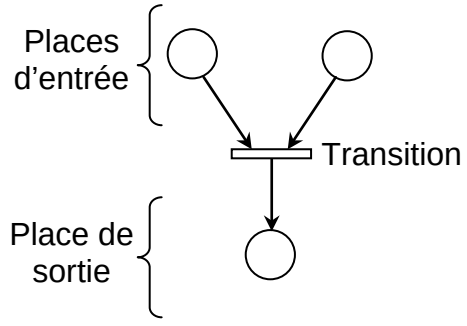


Figure 3.1 : Composants d'un réseau de Pétri.

2.1. Matrice d'incidence.

***Définition 3.1*: Matrice d'incidence d'un réseau de Pétri**

Etant donné un réseau de Pétri Ordinaire *N*, sa matrice d'incidence *W* est une application *PxT* dans Z définie par :

$$\forall\ p \in P, \forall\ t \in T, W(p,t) = post(p,t) - pré(p,t)$$

2.2. Règles de fonctionnement d'un RdP.

- Une transition est *validée* ssi chacune des ses places d'entrée contient au moins une marque
- Une transition validée peut alors être *tirée*. Cette opération dite de *mise à feu*, consiste à enlever une marque dans chacune des places d'entrée, et à ajouter une marque dans chacune des places de sortie.

***Définition 3.2* : franchissement d'une transition**

Si *t* est une transition franchissable pour le marquage *M*, le franchissement de *t* conduit à un nouveau marquage M' tel que :

$$\forall\ p \in P\ : M'(p) = M(p) - Pré(p,t) + Post(p,t)$$
$$= M(p) + W(p,t)$$

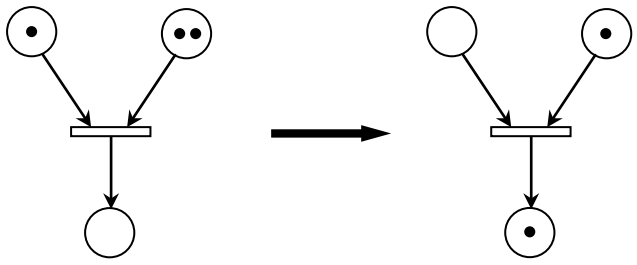


Figure 3.2 : Franchissement d'une transition.

2.3. Propriété qualitative des réseaux de Petri.

2.3.1. Propriété comportementale.

2.3.1.1. Atteignabilité.

La dynamique d'un système modélisé à l'aide d'un RdP est représentée par l'évolution des marquages obtenus, à partir du marquage initial M_0, par le franchissement de transitions. $R(M_0)$ représente l'ensemble des marquages qu'ils est possible d'atteindre à partir de M_0. Dans l'étude d'un système dynamique, il est souvent souhaitable de savoir si un état donné peut être atteint ou, à l'inverse, si un état non désiré risque d'être atteint. Ce problème est appelé problème d'atteignabilité.

***Définition 3.3*: Atteignabilité**

Etant donné un réseau de Pétri Ordinaire $N = \left(P, T, Pré, Post, M_0\right)$, et un marquage *M*, le problème d'**atteignabilité** consiste à vérifier s'il existe une séquence de transitions σ qui transforme M_0 en *M*

2.3.1.2. Bornitude.

Un réseau de Pétri est borné pour un marquage M_0, si pour tout marquage accessible, le nombre de marques dans chaque place est $\leq$ **k** (entier). Vu que le franchissement d'un nombre de transitions peut provoquer la modification du nombre de marquage du réseau, dans le cas particulier il arrive que le nombre de marquages s'accroît infiniment d'où la notion du réseau non borné.

***Définition 3.4* : Bornitude**

Un réseau de Pétri $N = \left(P, T, Pré, Post, M_0\right)$ est dit **k-borné** si le nombre de jetons dans chaque place ne dépasse pas k

- $M(p) \leq k, \forall\ p \in P\ et\ \forall\ M \in R\left(M_0\right)$

Un réseau de Pétri est dit **borné** s'il est k-borné pour un certain nombre entier k > 0

2.3.1.3. Vivacité et blocage.

Un réseau de Pétri est dit vivant pour un marquage initial M_0 si à partir de tout marquage successeur de M_0 on arrive à franchir toutes les transitions du réseau. Le bon fonctionnement de nombreux systèmes est exprimé par la propriété de la vivacité d'une transition, cette caractéristique est notamment importante vu qu'elle assure que le réseau est sans blocage et également que toute partie du système de commande représenté est accessible.

***Définition 3.5* : Vivacité et blocage**

Une transition *t* d'un réseau de Pétri $N = \left(P, T, Pré, Post, M_0\right)$ est dite **vivante** si elle peut être franchie quel que soit le marquage atteint.

- $\forall\ M \in R\left(M_0\right), \exists\ M' \in R\left(M\right)$ tel que *t* soit franchissable pour M'

Un réseau de Pétri $N = \left(P, T, Pré, Post, M_0\right)$ est dit **vivant** si chacune de ses transitions est vivante

2.3.2. Propriété structurelle.

Les propriétés structurelles ne dépendent que de la structure du réseau de Petri, et non de la manière dont les jetons évoluent dans le réseau.

2.3.2.1. Vivacité structurelle.

***Définition 3.6* : Vivacité structurelle**

Un réseau de Pétri est dit **structurellement vivant** s'il existe un marquage initial M_0 tel que le réseau de Pétri marqué soit vivant.
Un Rdp vivant est structurellement vivant, mais la réciproque est fausse.

2.3.2.2. Conservation.

***Définition 3.7* : Conservation**

Un réseau de Pétri est dit **conservatif** s'il existe un vecteur colonne X qui associe à chaque place *p* un poids entier positif x(p) ≥ 0 de telle sorte que :

$$X^t.M = X^t.M_0,\ \forall\ M_0\ et\ \forall\ M \in R\left(M_0\right),\ où\ X = \left[x\left(p_1\right), x\left(p_1\right), \ldots, x\left(p_q\right)\right]^t$$

2.3.2.3. Répétition.

Définition 3.8 **: Répétition**

> Un réseau de Pétri est dit **répétitif** s'il existe un marquage initial M_0 et une séquence σ de transition franchissables dans laquelle chaque transition apparaît un nombre illimité de fois.

2.3.2.4. Consistance.

Définition 3.9 **: Consistance**

> Un réseau de Pétri est dit **consistant** s'il existe un marquage initial M_0 et une séquence σ de transition franchissables qui contient au moins une fois chaque transition et dont le franchissement conduit à nouveau au marquage initial M_0.

3. Méthode d'analyse.

Pour chaque robot r_i un réseau de Pétri modélise le parcours local entre des points de rendez-vous. Le produit synchronisé des réseaux locaux est un réseau de Pétri global modélisant l'état global du système à tout instant [HAD 03]. Le réseau de Pétri de la figure (3.3) représente le comportement d'un robot r_i, et se définit par w le tuple $N_i = \left(P_i , T_i , \Pr é_i , Post_i , MO_i \right)$ où :

- $P_i = \left\{ p_{(i,1)} ,..., p_{(i,j)} ,..., p_{(i,n_i)} \right\}$ est un ensemble fini et non vide de places représentant soit l'ensemble des points de rendez-vous ou bien l'ensemble des déplacements. La présence d'un jeton dans la place $P(i, j)$ signifie que r_i est soit en déplacement vers son j^{ieme} point de rendez-vous, soit présent sur ce dernier mais "en attente" de son homologue.

- $T_i = \left\{ t_1 ,..., t_j ,..., t_{ni} \right\}$ est un ensemble fini et non vide de transitions. La condition de franchissement d'une transition t_j est que soit les deux robots concernés par le j^{ieme} point de rendez-vous soient présents ou bien que le robot ri soit arrivé à son rendez-vous.

– $Pré_i$ est la matrice d'incidence arrière, définie de $P_i \times T_i$ dans $\{0, 1\}$ comme suit :

$$Pré_i(p,t) = \begin{cases} 1 & si\ p = p_{(i,j)}\ et\ t = t_{f(i,j)} \\ 0 & sinon \end{cases}$$

– $Post_i$ est la matrice d'incidence avant, définie de $P_i \times T_i$ dans $\{0, 1\}$ comme suit :

$$Post_i(p,t) = \begin{cases} 1 & si\ p = p_{(i,(j+1) \bmod ni)}\ et\ t = t_{f(i,j)} \\ 0 & sinon \end{cases}$$

– Le marquage initial, noté MO_i, définit le nombre de jetons contenus dans toutes les places du réseau. $MO_i(p_{(i,j)})$ définit le marquage initial de la place $p_{(i,j)}$ comme suit :

$$MO_i(p_{(i,j)}) = \begin{cases} 1 & si\ j = 0 \\ 0 & sinon \end{cases}$$

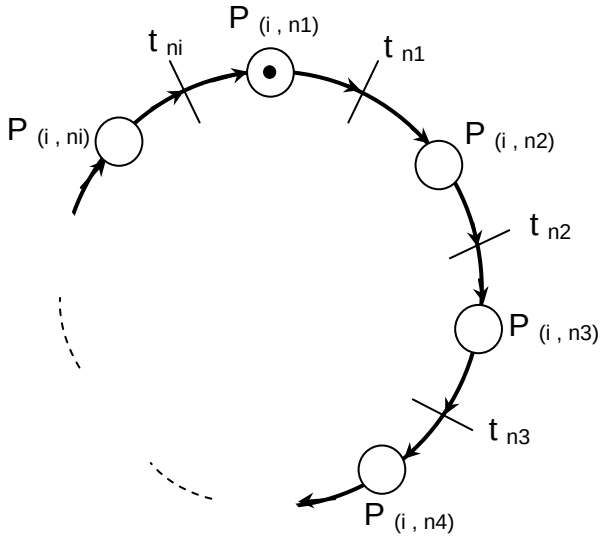


Figure 3.3 : Le cycle de visite du robot r_i.

Considérons un système constitué de 3 robots et 3 points de rendez-vous. La figure (3.6) représente le réseau de Pétri global correspondant à ce système. En ordonnant les points de rendez-vous de chacun des robots, nous obtenons le tableau suivant :

Robots	r1	r2	r3
Points rendez vous	l_1 l_3	l_2 l_3	l_1 l_2

Tableau 3.1 : Ordonnancement des rendez-vous

Voici le RdP modélisant le parcours local de chaque robot individuellement.

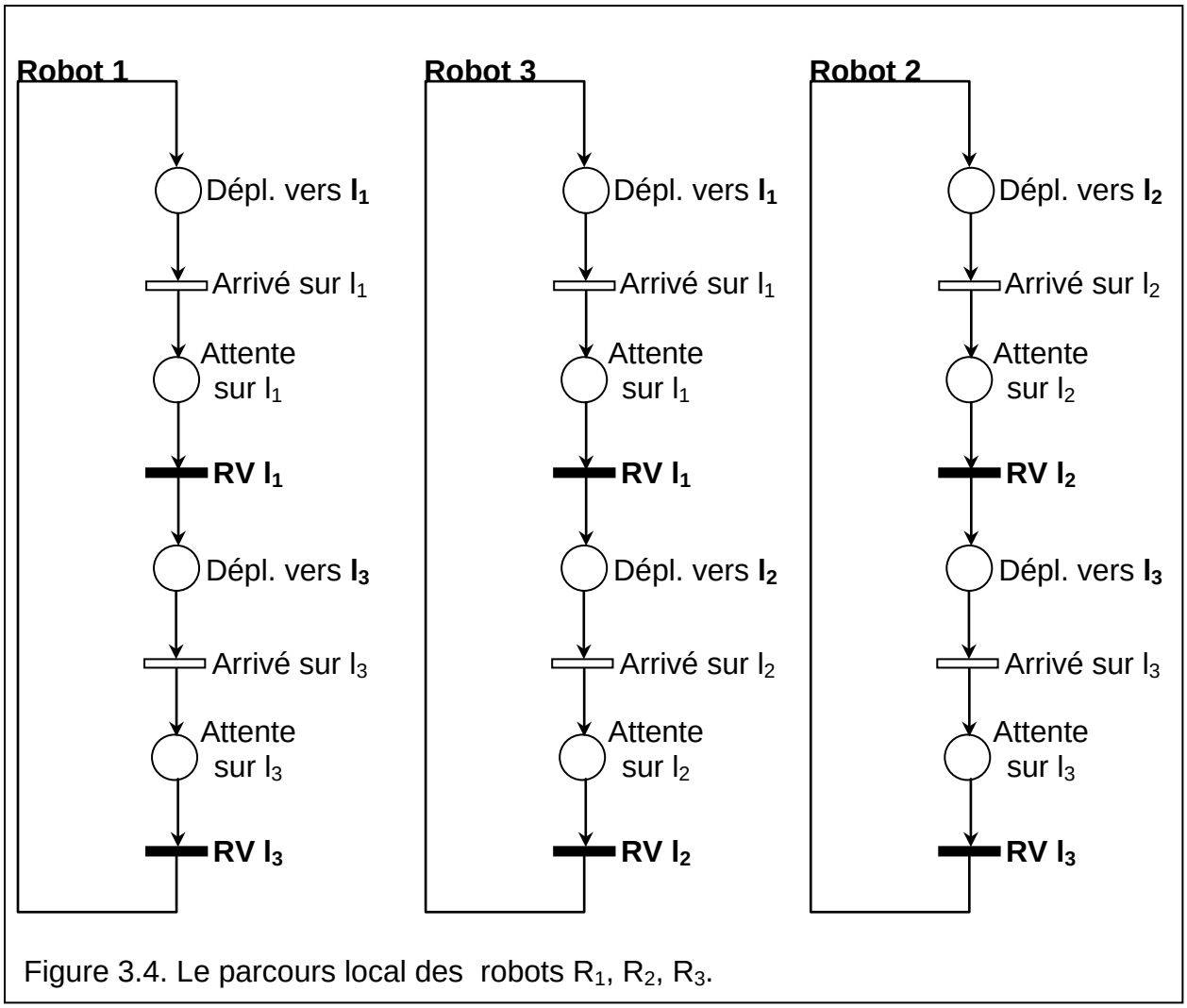


Figure 3.4. Le parcours local des robots R_1, R_2, R_3.

D'après le tableau 3.1 on remarque que le premier rendez-vous établit sera entre les deux robots $\mathbf{r_1}$ et $\mathbf{r_3}$ sur le point de rendez-vous $\mathbf{l_1}$ comme suit :

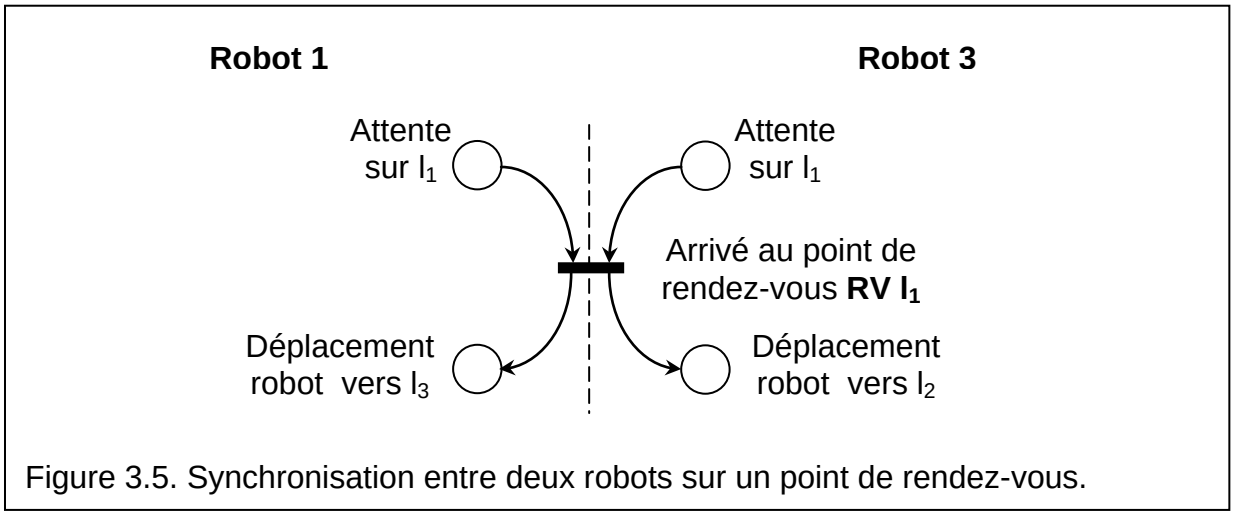


Figure 3.5. Synchronisation entre deux robots sur un point de rendez-vous.

Après synchronisation de tous les réseaux locaux à partir du tableau 3.1 on obtient le réseau de Pétri global modélisant l'ordonnancement de tous les points de rendez-vous présenté sur la figure 3.5.

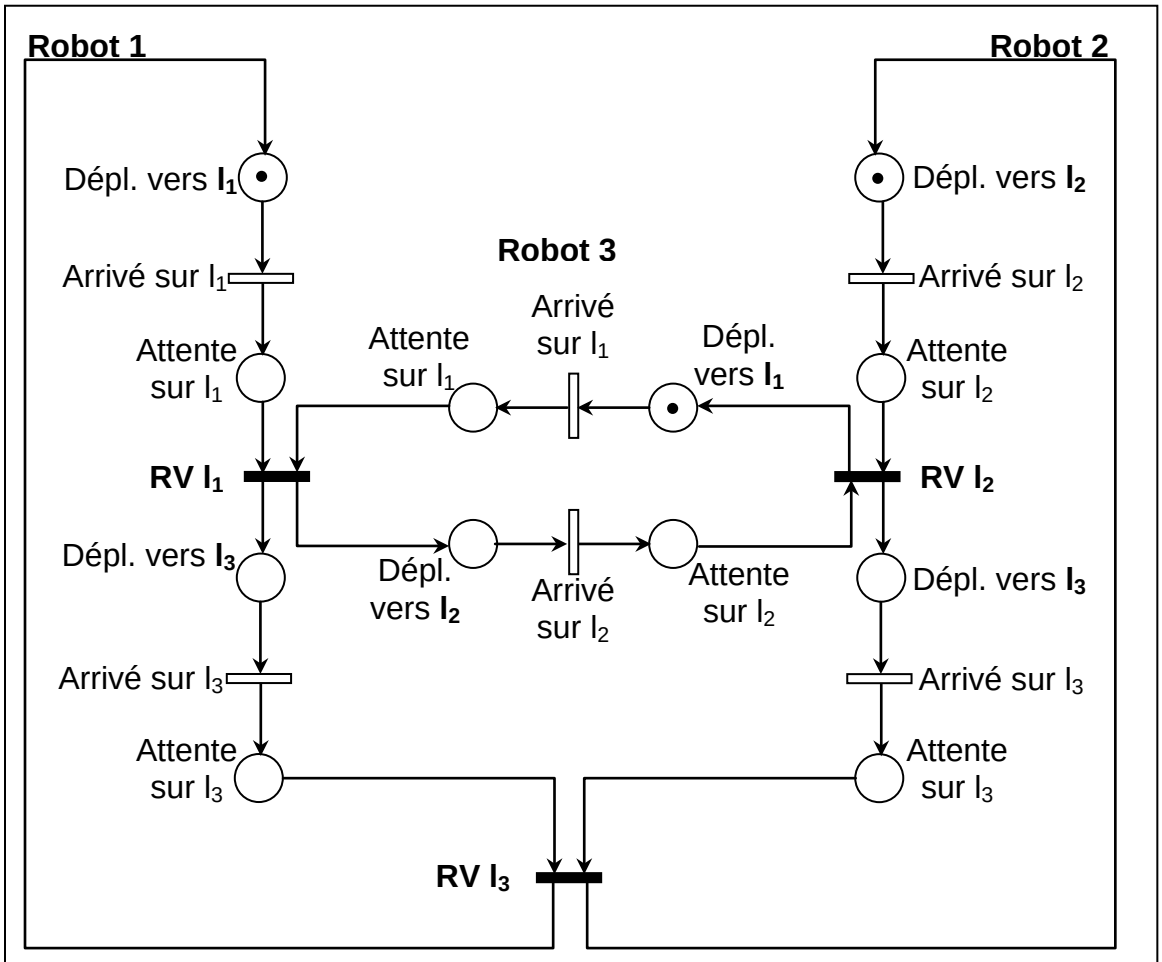


Figure 3.6 : Le réseau de Pétri modélisant les visites des points de rendez-vous.

4. Présentation des composants G-Objets.

L'approche retenue pour faciliter l'implémentation des déplacements des robots consiste à réaliser la décomposition du réseau de Pétri modélisant l'ordonnancement en Graphes-Objets (G-objets). Elle repose sur les actions suivantes :

- Analyse du modèle ouvert d'un système, car il correspond à la spécification du système que l'on réalise. Il nous permet de générer le squelette du programme.
- Interprétation de la représentation des activités concurrentes; à cette fin, nous introduisons les *G-objets* qui constituent des types de composants logiciels.

Définition 3.10 **: G-objets**

> Sous-ensemble d'un réseau de Pétri (modèle ouvert). A partir des G-objets, selon leur nature, des séquences spécifiques de code seront générées.

Nous avons utilisé les quatre types de G-objets :

- Les *Processus* sont constitués par des sous-ensembles du réseau de Pétri assimilables à une machine à états [HAC 74].
- Les *Actions* sont associées aux transitions du modèle et correspondent aux traitements effectués par le système. Une Action peut également réaliser un mécanisme de synchronisation entre Processus.
- Les *Etats_Processus* correspondent à certaines places du modèle et représentent les états d'un Processus.
- Les *Ressources* correspondent aux places du modèle qui ne sont pas des Etats_Processus et représentent des zones de stockage de données.

4.1. G-objet de type processus.

Cette section propose de construire les *Processus* à partir des ensembles séquentiels. De tels ensembles représentent des modèles de comportement séquentiels, liés à la notion de machine à états dans laquelle aucun parallélisme n'est possible. Nous les interprétons comme des *classes* de Processus.

Chaque classe est instanciable : les processus sont munis de leur contexte unique et d'un comportement propre défini dans la classe de processus.

Dans les systèmes d'exploitation, un processus est caractérisé par son contexte d'exécution [KRA 85]. Il regroupe l'ensemble des informations courantes du processus, c'est-à-dire les registres (processeur et utilisateurs). Nous définissons ainsi les notions de *compteur ordinal* et de *mot d'état*.

4.1.1. Compteur ordinal d'un processus.

Pour désigner l'Action à exécuter, il est nécessaire de transposer la notion de compteur ordinal : il s'agit de la transition activée (traitement à effectuer).

Définition 3.11 **: Compteur ordinal d'un processus**

> Le compteur ordinal désigne une étape dans le comportement du Processus. Ces étapes correspondent, soit à des Actions, soit à des Etats_Processus

4.1.2. Mot d'état d'un processus.

Un mot d'état caractérise les données manipulées par un Processus instancié.

Définition 3.12 **: Mot d'état d'un processus**

> Ensemble des informations manipulées durant les étapes de comportement d'un Processus.

4.1.3. Processus instancié.

La notion de Processus permet de définir des modèles de comportement séquentiels reposant sur la notion de machine à états. Un processus instancié s'exécute en concurrence avec les autres.

Le calcul du nombre de processus instanciés s'effectue à partir du marquage initial du modèle :

- Le nombre de processus instanciés est égal à la cardinalité du marquage de l'ensemble des places reliant les transitions du Processus entre elles
- La valeur du compteur ordinal de chaque processus instancié à son démarrage est indiquée par la position de la marque initiale définissant l'instance
- Le mot d'état initial de chaque processus instancié est défini, s'il y a lieu, par la nature de la marque repérant l'instance

4.2. G-objet de type action.

Une Action représente une partie du traitement effectué par un Processus. Dans le modèle, elle correspond à un niveau de description atomique. Elle est naturellement associée aux transitions.

Conformément à la théorie des réseaux de Pétri [BRA 82], "l'exécution" d'une transition se décompose en :

- Evaluation de la précondition (éventuellement bloquante).
- Réalisation du traitement associé
- Production du marquage postcondition

Les différents Processus du modèle sont des entités séquentielles concurrentes dont l'exécution est soumise à des contraintes liées aux Actions. Ces contraintes mettent en œuvre une séquentialisation des Actions des différents processus instanciés. Elles s'expriment :

- Directement : l'exécution d'une Action implique plusieurs processus instanciés qui réalisent ainsi une synchronisation.
- indirectement : l'exécution d'une Action est liée au contenu de Ressources, c'est-à-dire de zones de stockage contenant des données produites par d'autres processus instanciés; il y a réalisation d'une communication asynchrone.

Les contraintes directes définissent des Actions dont les conditions d'activation dépendent du contexte de plusieurs processus instanciés. Elles sont exprimées par des arcs reliant des Etats_Processus de différents Processus à l'Action.

Les contraintes indirectes définissent des Actions dont "l'activabilité" est soumise au contexte d'un seul processus instancié. Elles sont exprimées par des arcs reliant des Ressources à l'Action.

Définition 3.13 **: Action (simple, synchronisée)**

Une Action est une transition du modèle. Elle correspond à un traitement.
Si le traitement est local au Processus (une seule contrainte directe), l'Action est dite *simple*.
Dans le cas contraire (plusieurs contraintes directes), elle correspond à un rendez-vous entre plusieurs Processus; l'Action est alors *synchronisée.*

4.3. G-objet de type état_processus.

Certaines places du modèle relient les Actions d'un Processus entre elles. Elles sont appelées Etat_Processus.

Dans un modèle validé, un Etat_Processus sans successeur est un état d'accueil [BRA 82] pour le Processus. Il est associé à une terminaison du traitement. Un Etat_Processus ayant plusieurs successeurs est considéré comme un point de choix.

Définition 3.14 **: Etat_Processus (simple, alternatif, de terminaison)**

Un Etat_Processus est une place définissant une contrainte directe entre plusieurs Actions d'un Processus. S'il ne comporte pas de successeur, il est dit de *terminaison*. S'il comporte un seul successeur, il est *simple*. Dans les autres cas, il est qualifié d'*alternatif*.

4.4. G-objet de type ressources.

Certaines places du modèle représentent des zones de stockage de données. Elles sont interprétées comme des supports de mécanismes de communication asynchrone entre les Processus du modèle. Nous appelons de telles places des Ressources.

Définition 3.15 **: Ressource**

Une Ressource est une place qui n'est pas contenue dans un Processus. Elle correspond à une zone de stockage de données accessible par un ou plusieurs Processus.

4.5. Semi flots et G-objet.

Après avoir défini les différents G-objets d'un modèle, nous allons les relier à la théorie des réseaux de Pétri, et, en particulier, à la notion de semi-flot [BRA 82], [HAD 88].

Pour caractériser les activités concurrentes représentées par les Processus, nous utilisons des propriétés structurelles du réseau de Pétri.

4.5.1. Semi-flots d'un réseau de Pétri Ordinaire.

Les semi-flots sont des invariants définissant des relations entre le marquage de certaines places d'un réseau de Pétri.

Les semi-flots d'un réseau de Pétri Ordinaire sont actuellement parfaitement définis d'un point de vue théorique. Et plusieurs algorithmes de calcul ont été proposés [TRV 89].

Dans un réseau de Pétri, tout franchissement d'une transition entraîne une modification du marquage (Définition 3.2). Cette différence est exprimée par la matrice d'incidence (Définition 3.1) [BRM 82], [HAD 88].

L'équation de changement d'état d'un réseau de Pétri exprime que la différence entre deux marquages accessibles reste dans l'image de la matrice d'incidence *W*.

Nous définissons les flots comme des formes linéaires annulant la fonction d'incidence. Les flots définissent alors des invariants linéaires pour les réseaux de Pétri.

Définition 3.16 **: Flot d'un réseau de Pétri**

Soit ***R*** un réseau de Pétri Ordinaire de matrice d'incidence ***W*** et ***f*** un vecteur de places. ***f*** est un flot de ***R*** si et seulement si :

- $f^T \cdot W = 0$

Les flots à coefficients positifs ont un intérêt particulier : ils impliquent que les places à coefficient non-nul dans le flot restent bornées.

Définition 3.17 **: Semi-flot d'un réseau de Pétri**

Un semi-flot de ***R*** est un flot à coefficients positifs.

Une famille génératrice de semi-flots est le plus petit ensemble de semi-flots permettant d'exprimer, par combinaison linéaire, tous les semi-flots d'un réseau de Pétri.

Définition 3.18 **: Famille génératrice de semi-flot**

Soit ***R*** un réseau de Pétri Ordinaire, et $F = \{ f_1 , ... , f_n \}$ une famille génératrice de semi-flots. Nous avons :

- $\forall\ f$ semi-flot de ***R*** $\left| f \notin F , \exists\ a_1 , ... , a_n \in Q^+ \right| f = a_1 . f_1 + ... + a_n . f_n$

F est une famille génératrice de support minimal de semi-flots si et seulement si :

- $\forall\ f \in F , \exists a_1 , ... , a_n \in Q^+ \left| f = a_1 . f_1 + ... + a_n . f_n \right.$?

4.5.2. Modèle structurel.

Un ensemble séquentiel déterminant un Processus est constitué de places et de transitions. Dans un tel ensemble, un invariant de places, représentant les étapes du comportement d'un Processus est directement lié à la notion de machine à états. Le mécanisme d'instanciation dynamique des Processus préserve naturellement la cohérence des invariants.

Comme nous recherchons une décomposition optimale en Processus, nous devons la calculer à partir d'un ensemble minimal de semi-flots (une *famille génératrice de support minimal*).

4.6. Algorithme de décomposition en processus.

Ce paragraphe présente un algorithme de décomposition d'un réseau de Pétri en ensembles de Processus, d'Actions, d'Etats_Processus et de Ressources.

Il permet de décomposer un modèle en un ensemble de processus communicants (machine à état). Pour cela, il faut pouvoir extraire de la famille génératrice de semi-flots de support minimal du modèle structurel associé au moins un sous-ensemble de semi-flots $\boldsymbol{F_p}$ vérifiant les quatre propriétés énoncées ci-après.

Propriété 1 **: Couverture de l'ensemble des Actions**

$\Gamma(F_P) = T$: les semi-flots composant F_P couvrent l'ensemble des transitions du modèle.

Aucune transition ne doit être écartée de la décomposition. Si tel est le cas, les transitions isolées ne pourront être incluses dans des Processus.

Propriété 2 **: Indépendance des processus d'un modèle**

$\forall$ p1 p2 $\in F_P$, p1 $\cap$ p2 = $\varnothing$: les semi-flots de F_P n'ont aucune place en commun.

Les places précondition ou postcondition d'Actions appartenant à différents Processus sont des Ressources.

Propriété 3 **: Conservation des processus instanciés**

$\forall$ p, t $\in \Gamma(F_P)$, alors |val (p,t)| = |val (t, p)| = 1 : il ne transite qu'une seule marque sur chaque arc reliant les composantes des semi-flots de F_P aux transitions d'un Processus.

Cette propriété est étroitement liée à la notion de compteur ordinal. Par définition, ce compteur ordinal étant unique, il y a conservation du nombre de marques; un processus instancié ne peut se terminer que lorsque la marque matérialisant le compteur ordinal se situe dans une place correspondant à un Etat_Processus de terminaison.

***Propriété 4* : Cohérence comportementale du modèle**

$\forall\ p \in \Gamma(F_P)$, $|M_0\ (Sup^2\ (F_P))| \geq 1$: tout Processus de la décomposition doit contenir au moins une marque initiale dans les places du semi-flot qui lui sont associées.

Conformément à la théorie des réseaux de Pétri, un modèle qui ne vérifie pas la propriété 4 n'est pas vivant car il comporte une composante conservative vide.

4.6.1. Description de l'algorithme.

La définition des Actions, Etats_Processus et Ressources repose sur la notion de Processus. Le calcul des décompositions valides d'un réseau de Pétri permet donc de déterminer l'ensemble des objets qui le composent.

L'algorithme comporte les étapes suivantes :

1. Création du modèle structurel associé au réseau de Pétri.

2. Calcul de F, une famille génératrice de semi-flots de support minimal du modèle structurel.

3. Suppression des semi-flots ne vérifiant pas les propriétés 3 ou 4, le résultat est noté F'.

4. Etude de toutes les combinaisons de semi-flots possibles dans F' afin de déterminer celles qui vérifient les propriétés 1 et 2. Nous obtenons de la sorte tous les sous-ensembles FP possibles.

5. Caractérisation des G-objets.

Pour réduire la durée d'exécution de l'algorithme, il est important de commencer par vérifier des propriétés 3 et 4 car elles se rapportent à des semiflots. Ceci permet de réduire le nombre de combinaisons à étudier dans la phase 4, les propriétés 1 et 2 s'appliquant à des combinaisons de semi-flots.

5. Exemple de décomposition en G-Objets.

Deux trains circulent dans le même sens sur un tronçon de voie circulaire découpé en sept sections. Il est spécifié que deux trains ne pourront jamais, pour des raisons de sécurité, se situer sur deux segments contigus. Des feux gèrent l'accès à chacune des sections ; cet exemple est tiré de [KOR 92].

Le réseau de Pétri de la Figure 3.7 modélise ce problème : les sections sont représentées par les places *Section 1* à *Section 7*. La présence d'une marque dans l'une de ces places indique qu'un train s'y trouve. Les feux sont modélisés par les places *F1* à *F7*. La présence d'une marque signifie que le feu est au vert, l'entrée dans la section qu'il garde est alors possible. Le passage d'une section x à une section y s'effectue à l'activation de la transition *x vers y*.

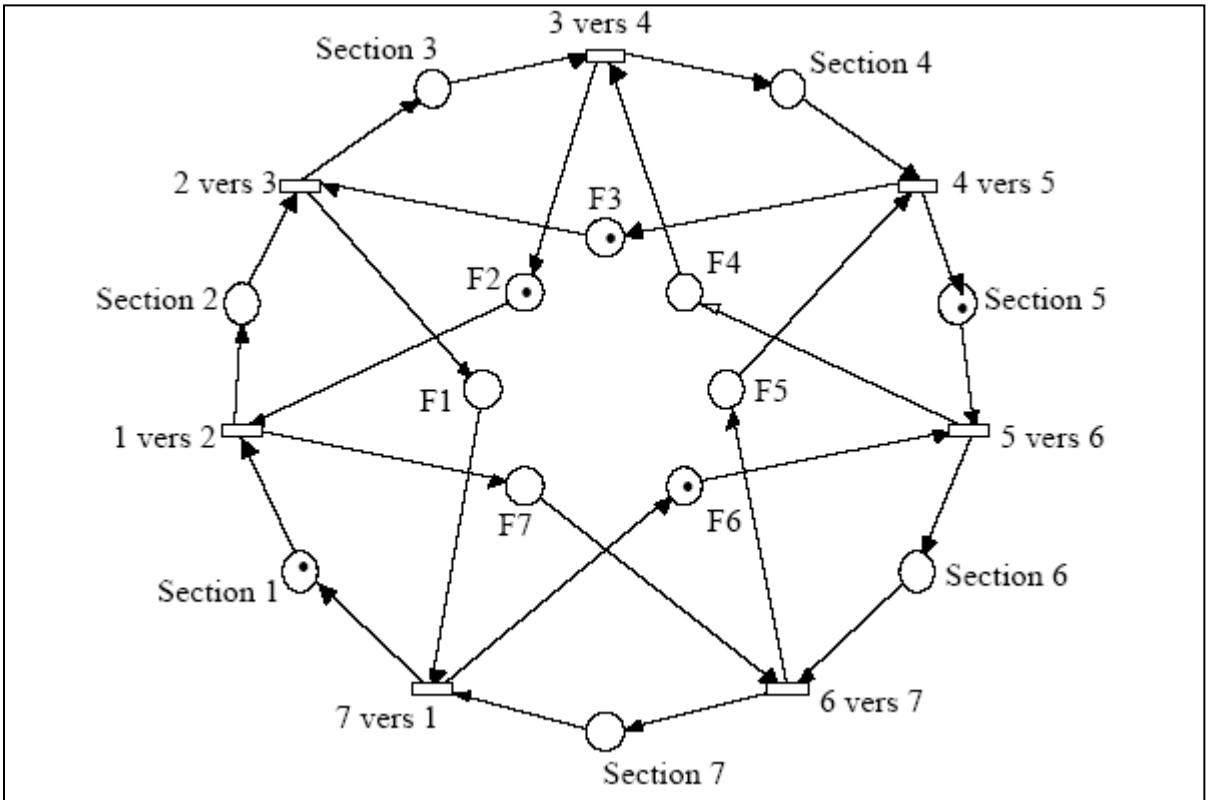


Figure 3.7. Réseau de Petri associé au modèle des trains.

Détaillons l'algorithme de décomposition en Processus sur le modèle défini dans l'exemple précédent :

Etape 1 :

Le modèle structurel est donné en figure 3.7

Etape 2 :

Le calcul des semi-flots du modèle structurel donne F, composée de :

01 $\Rightarrow$ F1 + F2 + F3 + F4 + F5 + F6 + F7.
02 $\Rightarrow$ Section 1 + Section 2 + Section 3 + Section 4 + Section 5 + Section 6 + Section 7.
03 $\Rightarrow$ Section 1 + F1 + F3 + F5 + F7.
04 $\Rightarrow$ Section 2 + F1 + F2 + F4 + F6.
05 $\Rightarrow$ Section 3 + F2 + F3 + F5 + F7.
06 $\Rightarrow$ Section 4 + F1 + F3 + F4 + F6.
07 $\Rightarrow$ Section 5 + F2 + F4 + F5 + F7.
08 $\Rightarrow$ Section 6 + F1 + F3 + F5 + F6.
09 $\Rightarrow$ Section 7 + F2 + F4 + F6 + F7.
10 $\Rightarrow$ Section 1 + Section 2 + F1.
11 $\Rightarrow$ Section 2 + Section 3 + F2.
12 $\Rightarrow$ Section 3 + Section 4 + F3.
13 $\Rightarrow$ Section 4 + Section 5 + F4.
14 $\Rightarrow$ Section 5 + Section 6 + F5.
15 $\Rightarrow$ Section 6 + Section 7 + F6.
16 $\Rightarrow$ Section 7 + Section 1 + F7.

Etape 3 :

Tous les semi-flots calculés vérifient les propriétés 3 et 4, F = F'.

Etape 4 :

- {1} vérifie les propriétés 1 et 2,
- {2} vérifie les propriétés 1 et 2,
- {3} ne vérifie pas la propriété 1. En essayant de l'associer avec d'autres semi-flots,

nous découvrons que la combinaison {3, 11, 13} vérifie les propriétés 1 et 2.

- etc...

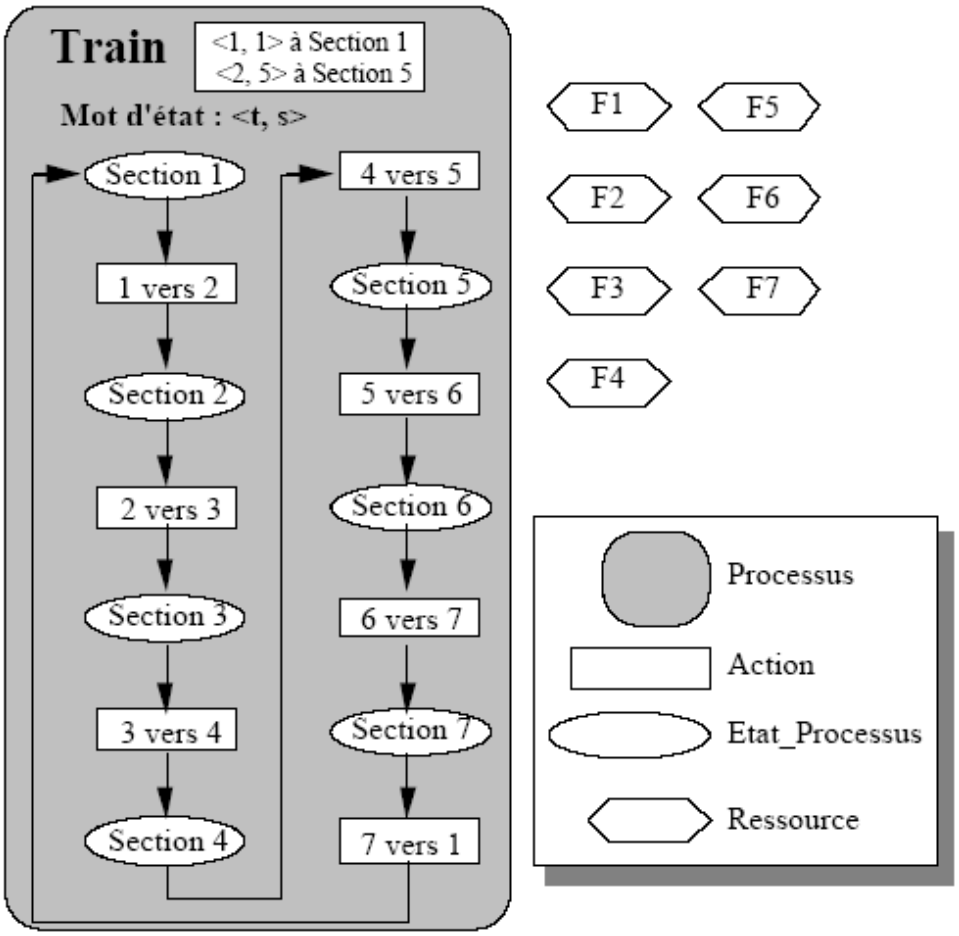


Figure 3.8. Décomposition en G-objets du modèle défini dans l'exemple du train.

Il est probable que le concepteur d'un tel modèle aura envisagé de modéliser le cheminement des trains sur la voie de chemin de fer. Dans ce cas, il choisira la décomposition {2}. Cependant, les autres décompositions peuvent s'avérer intéressantes, si l'on considère un autre point de vue (la gestion de portions de voies par exemple), ou pour une architecture particulière. Considérons que le concepteur du système choisisse la seconde décomposition. Il nomme l'unique Processus "train".

Etape 5 :

La figure 3.8 présente la décomposition en G-objets du modèle pour la décomposition choisie. L'unique Processus ne comporte que des Actions simples et des Etats_Processus simples.

6. Conclusion.

L'interprétation d'un réseau de Pétri, en vue de son implémentation, repose sur la notion de Processus. Après l'avoir effectuée, nous pouvons caractériser les différents G-objets. Cette décomposition en G-objet peut se faire soit d'une manière intuitive c'est-à-dire c'est le concepteur lui-même qui la réalise ou bien en utilisant un algorithme de caractérisation de processus basé sur la notion des semi-flots d'un réseau de Pétri.

L'analyse sémantique du réseau de pétri permet la caractérisation des différents objets pertinents. Chaque ensemble d'objets définit des fonctionnalités qui sont regroupés dans des modules, l'ensemble de ces modules compose le squelette de l'application. La génération de ces objets repose sur l'analyse des invariants du réseau de pétri en vue de détecter des machines à état communicantes.

Une fois la décomposition en Processus est connue, les autres G-objets sont automatiquement identifiables :

- Les Etats_Processus sont donnés par le support des places des semi-flots décrivant les Processus. Leurs attributs (de terminaison, simple ou alternatif) sont calculés en fonction du nombre de successeurs des places ainsi désignées.
- Les Ressources sont désignées par les places n'appartenant à aucun semi-flot décrivant les Processus.
- Les Actions correspondent aux transitions du modèle. Les attributs qui les caractérisent se déduisent de la décomposition en Processus (Action partagée par plusieurs Processus, gardée par une Ressource...).

	Introduction générale.
Chapitre 1 :	Etat de l'art sur la robotique mobile et sur la coopération entre robots.
Chapitre 2 :	Planification de trajectoire et navigation.
Chapitre 3 :	Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes.
Chapitre 4 :	Implémentation des processus parallèles à l'aide du langage de programmation ADA.
Chapitre 5 :	Conception d'une plateforme de robots mobiles autonomes et début de réalisation.
Chapitre 6 :	Simulation.
	Conclusion générale.
	Bibliographie.
	Annexe et Appendice.

Chapitre 4.

Implémentation des processus parallèles à l'aide du langage de programmation ADA.

1. Introduction.

La nécessité de maintenir dynamiquement les interactions de multitudes de programmes G-Objets oeuvrant en parallèle, communiquant, coopérant et partageant des ressources communes est primordial [KOR 92].

Un fonctionnement réparti, empêchant toute vision globale, aggrave les difficultés de contrôle des problèmes sous-jacents de communication (synchrone ou asynchrone).

Ces problèmes se compliquent lorsqu'on exige que les vérifications nécessaires pour garantir que des situations indésirables (erreurs, défaillances, pannes, …) soient effectués le plus tôt possible.

Chaque programme G-Objet peut être approché par un modèle. Il en existe deux types : modèle fermé et modèle ouvert.

Un modèle fermé peut être vu comme une "boîte noire" contenant le module G-Objet et son environnement. Donc un modèle fermé est un modèle muni de la représentation abstraite de l'environnement dans lequel il s'exécute.

Par contre le modèle ouvert représente le programme G-Objet sans son environnement. Ce modèle ouvert doit :

- préciser les services requis, nécessaires à l'exécution du système;
- définir proprement les services offerts à l'environnement;
- expliciter les relations entre les différents types de services.

Un modèle ouvert est contenu dans un modèle fermé figure (4.1). Le modèle ouvert décrit formellement un système. Les hypothèses et les propriétés attendues du système peuvent ainsi être validées formellement à l'aide du modèle fermé.

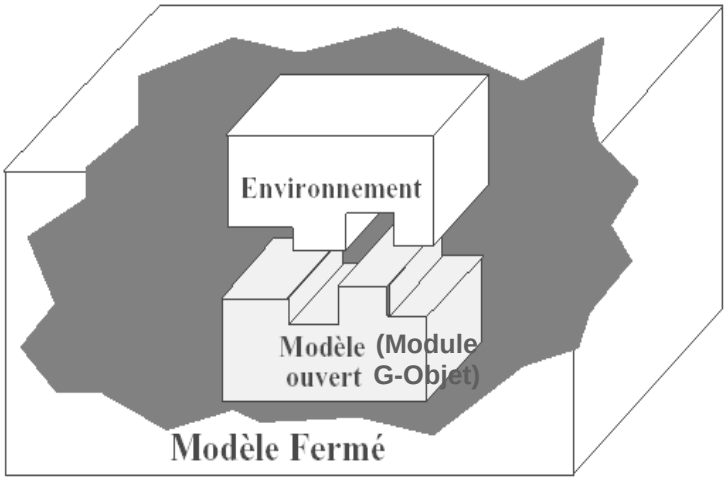


Figure 4.1 : Décomposition d'un modèle fermé en un modèle ouvert plus la représentation de son environnement.

Le langage Ada répond en grande partie aux problèmes soulevés par la mise en œuvre de plusieurs programmes en parallèles avec des interactions prenant des formes différentes (communications, coopération et compétition) avec les concepts de tâches et de rendez-vous.

Ada propose avec le concept d'exception, un outil puissant des gestions des erreurs.

La structure générale d'un programme Ada, divisée en partie visible contenant les clauses de visibilités [Ada 83] (les services offerts) et en partie privée contenant les traitements, correspond à la structure d'un modèle ouvert. Pour toutes ces qualités, Ada est le langage le mieux approprié pour implémenter ce type d'application répartie.

Avant de détailler les modules G-Objets, on présente dans le paragraphe suivant de ce chapitre les points forts du langage Ada 95.

2. Ada 95 un langage de programmation.

2.1. Historique.

En janvier 1975, le Ministère Américain de la Défense (DOD) a convoqué un groupe d'experts, le High Order Language Working Group (HOLWG), pour trouver une solution aux problèmes de qualité et de coût des logiciels militaires. Les plus gros frais de maintenance, était pour les systèmes temps réels embarqués [HAL 04].

L'objectif est d'avoir un langage de haut niveau permettant le développement de très grosses applications industrielles, ainsi que d'applications critiques (avionique, signalisation des chemins de fer, contrôle de processus et d'applications médicales...).

Au printemps de 1977, une équipe française, dirigée par Jean Ichbiah, remporta l'appel d'offre. Le langage fut alors baptisé Ada, du nom d'Augusta Ada Byron, Comtesse Lovelace (1815 - 1852), le premier programmeur sur la "machine analytique" de Babbage.

Après quelques modifications de moindre importance, le langage fut standardisé par l'ANSI en 1983.

Il fallut attendre 1987 pour obtenir une norme française satisfaisante et la standardisation ISO.

En 1988 le processus de révision de la norme a été relancé par, encore une fois le ministère américain de la Défense (DOD), et la révision a été conduite par l'Ada 9X Project Office, dirigé par Chris Anderson pour aboutir à l'approbation simultanée par l'ANSI et l'ISO au tout début de 1995.

Un module (programme) Ada est composé de trois parties :

- Partie visible : dans laquelle un programme Ada offre des services aux autres modules.
- Partie privée : dans laquelle on réalise les services offerts dans la partie visible.
- Gestion des erreurs : dans cette partie Ada offre un gestionnaire d'erreur permettant au programmeur de lever des exceptions et de les gérer à un certain niveau du programme.

Dans les trois sections suivantes on présente la structure générale d'un programme Ada ainsi que les concepts fondamentaux de tâches, de rendez-vous et d'exceptions.

2.2. Structure d'un programme Ada.

Un programme ADA s'articule autour d'un programme principal qui utilise le plus souvent des *paquetages* correspondant à des unités cohérentes :

- regroupement par fonctionnalité: entrées/sorties, bibliothèques mathématiques, ...
- type abstrait: nombres rationnels, listes, ...
- etc.

```
with  Nom_Des_Librairies_Utilisées;
use  Nom_Des_Librairies_Utilisées;
procedure Nom_Du_Programme is

        Partie déclarations

begin

        Partie instructions

end Nom_Du_Programme;
```

Partie déclaration.

Elle comporte les déclarations de constantes, variables, types, sous-programmes (procédure ou fonction), tâches.

Partie instruction.

C'est la séquence d'instructions du programme principal (principalement appels de sous-programmes déclarés dans la partie déclarations ou de sous-programmes de librairies)

La clause **with** permet d'accéder aux fonctionnalités du paquetage Nom_Des_Librairies_Utilisées.

2.3. Le multi - tâche en Ada.

Souvent sous-exploitées, les capacités du langage Ada dans le domaine du multitâche sont impressionnantes de facilitées. Nul besoin d'utiliser des pointeurs et autres mécanismes délicats à coder, tout est pris en compte par le langage.
Outre le fait premier de faire tourner plusieurs tâches avec une sensation qu'elles s'exécutent concurremment ou réellement concurremment pour les heureux possesseurs de système multi-processeurs, le multi-tâche permet facilement de lancer des actions asynchrones qui demandent des contorsions de programmation pas possible sinon.

La section 9 de la norme Ada95 [ADA 95] offre la possibilité de créer des tâches, et offre des mécanismes de synchronisations et de préemption. Ada95 offre une implémentation de systèmes temps réel beaucoup plus agréable à développer, moins lourde à écrire qu'avec l'interface programmatique d'un système d'exploitation traditionnel.

Une tâche Ada se présente syntaxiquement comme un paquetage. Elle est constituée d'une spécification décrivant l'interface présentée aux objets externes et d'un corps décrivant son comportement dynamique :

- - spécification : **task** La_tâche **is** ... **end** La_tâche;	*- - corps :* **task body** La_tâche **is** ... **begin** ... **end** La_tâche;

La première fonction d'une tâche est le **parallélisme**, en effet il est avantageux de considérer l'application comme la mise en parallèle de plusieurs activités, comme par exemple des traitements, des surveillances d'événements clavier/souris, un dialogue avec un serveur … etc.

- **Activation d'une tâche.**

L'activation d'une tâche est le plus souvent automatique. Une tâche est généralement activée dès après le **begin** du cadre (sous programme, programme principal, bloc) qui l'a déclarée. Si le cadre est une unité de bibliothèque, la tâche sera activée à l'élaboration de cette unité, c'est à dire lors de l'élaboration du programme principal.

Le seul moyen de maîtriser le moment d'activation d'une tâche est l'allocation dynamique via un pointeur de tâche :

```
task type La_tache is
   ...
end La_tache;

type une_tache is access La_tache;

cette_tache : une_tache;
   ...
begin
   ...
cette_tache := new La_tache;
   ...
```

- **Terminaison d'une tâche.**

Une tâche se termine lorsqu'elle atteint son **end**. L'unité qui a déclaré la tâche est dépendante de la tâche : elle ne peut se terminer que si la tâche est terminée ou a atteint une alternative **terminate**.

Les tâches peuvent interagir, principalement de deux façons : directement, en envoyant des données en paramètres les unes vers les autres, et indirectement, en accédant à des données partagées. Les données en paramètres sont passées directement entre les tâches en Ada par un mécanisme de synchronisation appelé « ***rendez-vous*** ». [ZAF 99] Le mécanisme de rendez-vous Ada permet à la fois la *synchronisation* et la communication, nécessaires à la solution des problèmes de compétition et de coopération entre tâches réparties.

La communication entre tâches est réalisée par la transmission de paramètres typés dans les deux sens (in, out et in out), lors de l'occurrence d'un rendez-vous. Le mécanisme d'un rendez-vous simple peut être décrit comme suit :

> *Il y a rendez-vous entre une tâche* Appelante *et une tâche dite* Appelee, *lorsque la tâche* Appelante *appelle une* Entree *de la tâche* Appelee *et que cette dernière accepte cet appel. Le rendez-vous est terminé lorsque les actions associées à une instruction spécifique* accept *de la tâche* Appellee *ont été exécutées. Les deux tâches reprennent alors leur exécution respective, indépendamment l'une de l'autre.*

La forme de **communication** qui en découle est donc **synchrone** (bloquante). La figure (4.2) illustre deux cas. Dans le cas (a) la tâche *Appelee* est prête après que la tâche *Appelante* ait effectué une demande de rendez-vous sur une *entrée*. Alors que dans le cas (b) c'est la tâche *Appelee* qui est prête avant la demande de rendez-vous. Lors d'un appel, si la tâche *Appelee* attend sur l'instruction *accept*, le rendez-vous a lieu immédiatement. Sinon, la demande de la tâche *Appelante* est mémorisée et la tâche mise en attente *passive* de l'acceptation du rendez-vous par la tâche *Appelee*.

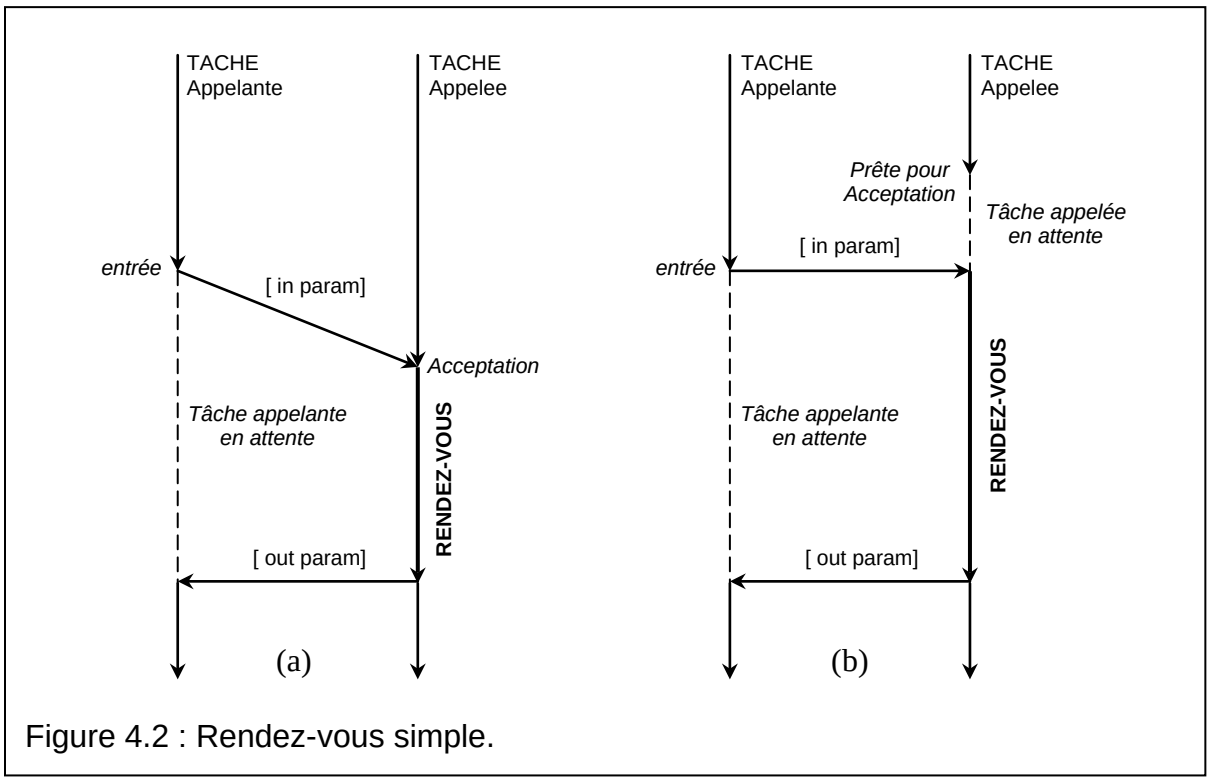


Figure 4.2 : Rendez-vous simple.

Si une tâche *appelante* effectue également une demande de rendez-vous sur la même *Entree* de la tâche *Appelee*, sa demande est également mémorisée, à la suite de la première. L'ordre de mémorisation est donc constitué par l'ordre d'arrivée (philosophie FIFO)

Dans un rendez-vous entre deux tâches, les deux entités réalisent un échange puis poursuivent indépendamment leur activité. Un rendez-vous entre deux tâches arrive comme conséquence d'un appel de l'entrée d'une tâche par une autre. Une entrée est déclarée dans une spécification de tâche comme on déclare une procédure dans une spécification de paquetage :

```
task La_tâche is
  entry E( ... );
  ...
end La_tâche;
```

Une entrée peut avoir des paramètres de mode ***in, out*** et ***in out***. On appelle une entrée comme une procédure : La_tâche.E(…) ;

Les instructions à effectuer pendant le rendez-vous sont décrites dans le corps de l'instruction **accept** correspondante à l'entrée activant la tâche :

```
accept E (...) do
  ...
end E;
```

La principale différence entre un appel d'entrée et un appel de procédure est la suivante : dans le cas d'une procédure, la tâche qui appelle la procédure exécute aussi immédiatement le corps de la procédure, alors que dans le cas d'une entrée, une tâche appelle l'entrée mais c'est la tâche à laquelle appartient l'entrée qui exécute l'instruction **accept** correspondante. De plus l'instruction **accept** ne sera exécutée que lorsqu'une tâche appelle l'entrée et que la tâche possédant l'entrée aura atteint l'instruction **accept**.

Les deux tâches s'attendent. Quand cela se produit, la suite d'instructions contenue dans l'instruction **accept** est exécutée pendant que la tâche appelante reste suspendue. On appelle cette interaction un rendez-vous. Quand la fin de l'instruction

accept est atteinte, le rendez-vous est fini et les deux tâches poursuivent leur exécution indépendamment (exemple : pour la tâche, exécution des instructions qui suivent le rendez-vous).

Chaque entrée possède une file d'attente des tâches qui sont en attente d'un rendez vous pour cette entrée : cette file est normalement traitée avec une politique **FIFO** (First In First Out).

L'instruction **select** permet à une tâche de choisir un rendez-vous parmi plusieurs. L'instruction **select** commence par le mot réservé **select** et se termine par un **end select**; Elle contient deux options ou plus, séparées par **or**. Une partie de code alternatif peut être insérée à l'instruction **select**, cette partie de code précédée du mot clé **else** est exécutée si les files d'attentes liées aux entrées proposées par le **select** sont vides :

```
select
  accept E1 ( ... ) do
     ...
  end E2;
or
  accept E2 ( ... ) do
     ...
  end E2;
else
  ...
end select ;
```

Le mécanisme de rendez-vous requiert parfois l'adjonction d'une tâche supplémentaire, un composant actif, comme par exemple une tâche tampon de messages.

2.4. Les exceptions en Ada.

Une *exception* est une situation inhabituelle, qui ne s'accorde pas avec le cas général prévu par le programme, la procédure ... et qui demande donc une gestion spécifique. L'avantage du mécanisme d'exception est qu'il permet un traitement uniforme des erreurs au sein du programme ADA.

Exceptions prédéfinies.

- CONSTRAINT_ERROR : activée quand on ne respecte pas une contrainte d'intervalle, de tableau
- STORAGE_ERROR : problème de mémoire
- PROGRAM_ERROR : activée quand on appelle un sous-programme inexistant, quand l'appel à une fonction se termine sans résultat retourné (return)
- NUMERIC_ERROR : activée quand une variable numérique est trop grande
- TASKING_ERROR : quand on a un problème avec une tâche.

3. Interfaces des modules composants l'ensemble G-Objets généré.

Compte tenu de nos hypothèses portant sur les Actions et le domaine d'application visé, le langage cible que nous considérons est procédurale et autorise la *manipulation du parallélisme*.

Nous détaillons dans cette partie les différents modules composant l'ensemble G-Objets. Pour chacun de ces modules, nous nous attachons à décrire précisément, conformément à la notion de composant, les services offerts et requis ; un schéma synthétise l'organisation des différents modules :

- Les modules-processus, supportant l'exécution des Processus du modèle ouvert;
- Le module de gestion des Ressources, réalisant les services liés aux données partagées entre les Processus;
- Le module de gestion des Actions synchronisées, mettant en œuvre les techniques de rendez-vous entre Processus;
- Le module de contrôle qui gère l'initialisation et la terminaison du prototype.

L'ensemble de G-Objets obtenu est constitué d'un ensemble de modules coopérant. Chaque module est une entité séquentielle;
Le comportement de chaque module respecte des règles qui peuvent être définies à l'aide de machines à états. Ce comportement doit tenir compte d'événements internes et externes.

Les liens entre un module et ses homologues sont réalisés au moyen de *services offerts* et de *services requis*. Les services offerts sont utilisés par les autres parties du G-Objets pour stimuler le module (événements externes). Ce dernier peut avoir besoin de services requis, fournis par d'autres modules.

Chaque type de G-objet caractérise une classe de fonctionnalités qui doit être implémentée :

- Les Ressources du modèle correspondent à des liens de communication asynchrones.
- Les Actions synchronisées du modèle définissent des mécanismes de communication synchrones. Nous choisissons de les gérer dans une unité dédiée.
- Les Etats_Processus et les Actions simples sont inclus dans des Processus. Un Processus définit une entité de traitement séquentielle parallélisable. Chaque Processus est réalisé sous la forme d'un module processus.

Il existe deux catégories de modules :

- Les modules-processus, associés aux Processus issus de la décomposition du modèle. Ils exécutent les traitements modélisés dans le réseau de Petri. Ils sont *actifs*.
- Les modules de service, associés à la gestion des Ressources et des Actions synchronisées. Ils réalisent, sur demande des modules-processus, les communications dans le prototype. Ils sont *passifs*.

Aucun module n'a de vision complète de l'état du fonctionnement de l'application. Le contrôle du prototype est donc confié à une unité dédiée qui intervient :

- A l'initialisation de l'application : elle initialise l'ensemble des modules en leur transmettant leurs paramètres de démarrage.
- Dans la gestion des cas d'exception : elle provoque la fin prématurée de l'application.
- A la terminaison de l'application : elle est chargée de comptabiliser les instances des processus qui disparaissent. Lorsque les modules-processus ont achevé leur exécution, elle termine l'application.

Dans les sections suivantes, nous allons présenter la réalisation des différents modules composant l'application.

3.1. Interface modules-Processus.

Un module-processus réalise le comportement d'un processus du réseau de Pétri, Deux modules-processus ne communiquent jamais directement entre eux mais uniquement par l'intermédiaire d'un module de service et s'exécute en parallèle (concurrence) comme indiqué figure 4.3. Le module de contrôle initialise les différentes instances du processus. Chaque instance signale la fin de son exécution, qu'elle soit normale ou anormale.

Les liens avec le module de gestion des Ressources n'existent que si le processus comporte au moins une Action avec précondition ressource. Il en est de même avec le module de gestion des Actions synchronisées : la relation n'existe que si le processus se synchronise au moins une fois.

Un processus regroupe plusieurs objets de types Etat-processus et de type Action.

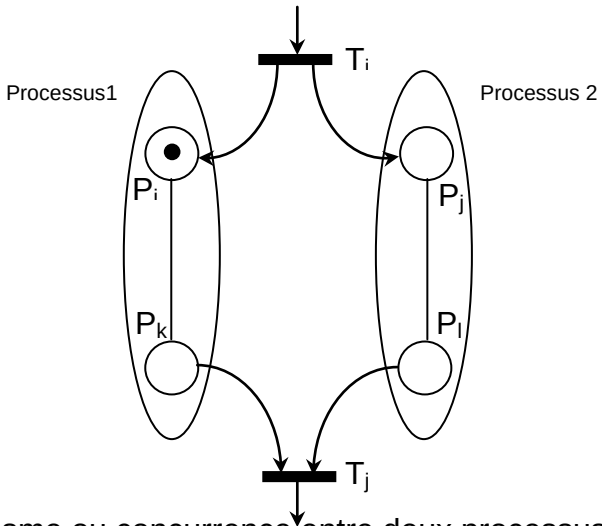


Figure 4.3 : Parallélisme ou concurrence entre deux processus.

Le réseau ci-dessus modélise deux processus, qui s'exécutent en parallèle. Après franchissement de T_i , le processus 1 = P_i … P_k et le processus 2 = P_j … P_l évoluent de manière parallèle, indépendante et chacun à son rythme.

3.1.1. Services requis et offerts.

Ces modules correspondent au déplacement d'un robot vers tous ces points de rendez vous de manière cyclique. On trouve dans notre application un seul processus qui sera instancié ***« Nombre de robots » fois*** ; il est implémenté dans un paquetage sous le nom Gestion_Du_Module_Processus_Robot.

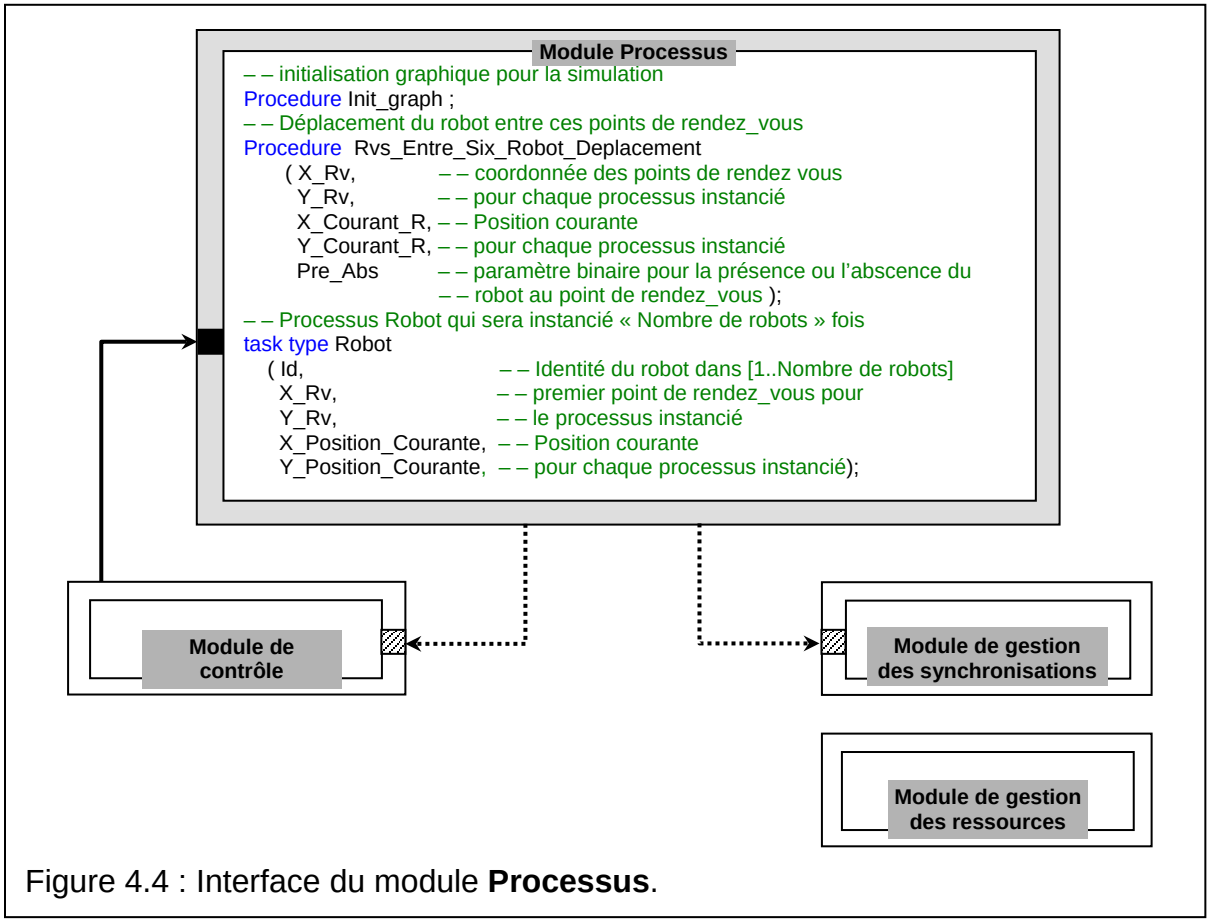


Figure 4.4 : Interface du module **Processus**.

Il est implémenté sous forme d'une tâche qui sera activée ou détruite par le module contrôle, et il offre dans sa partie visible les paramètres à manipuler suivants ; voir figure 4.4. :

- Identité du robot **Id** qui prend des valeurs dans **1 . . Nombre de robots**
- Coordonnées des points de rendez vous pour chaque processus instancié **X_RV**, **Y_RV**
- La position courante pour chaque processus instancié **X_Position_Courante**, **Y_Position_Courante**

Et manipule une primitive :

- nommée Rvs_Entre_Six_Robot_Deplacement permet à un robot de se déplacer entre ses points de rendez vous par l'intermédiaire des champs suivants :

- Un champ pour sa position courante (***X*** , ***Y***)
- Et un champ pour son futur rendez-vous (***LX*** , ***LY***)

Dans sa partie privée il réalise tous les traitements nécessaires pour réaliser les déplacements des robots.

3.2. Interface Module de gestion des synchronisations.

Ce module correspond à l'implémentation des actions synchronisées qui sont en réalité les transitions du modèle. Et les arcs d'entrées de ses transitions, provenant de processus différents, expriment des contraintes de synchronisation. Ainsi les conditions d'activation de ces actions dépendent du contexte à plusieurs processus instanciés.

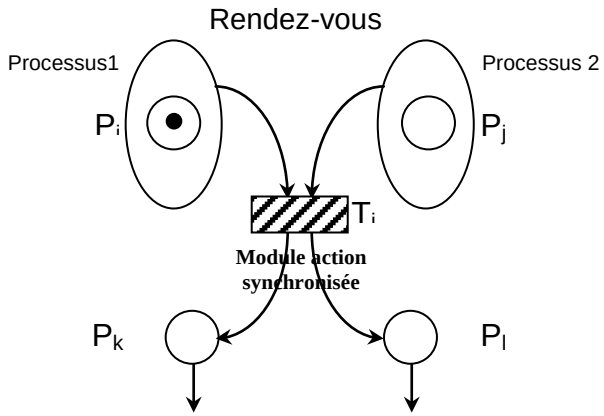


Figure 4.5 : Communication synchrone entre processus par rendez-vous.

Ce module établit un lien de communication synchrone entre deux processus dans un point de rendez vous considéré comme un nœud de communication, c'est-à-dire le franchissement de T_i (communication synchrone) ne peut se faire que lorsque P_k et P_l contiennent chacune une marque (voir figure 4.5).

3.2.1. Services requis et offerts.

Ce module correspond à une action synchronisée. Dans sa partie visible, il offre comme service l'établissement d'un lien de communication entre deux processus. Il reçoit des requêtes à partir de processus souhaitons communiquer, établit un lien physique, gère la communication et libère les processus en leur transmettant des acquittements.

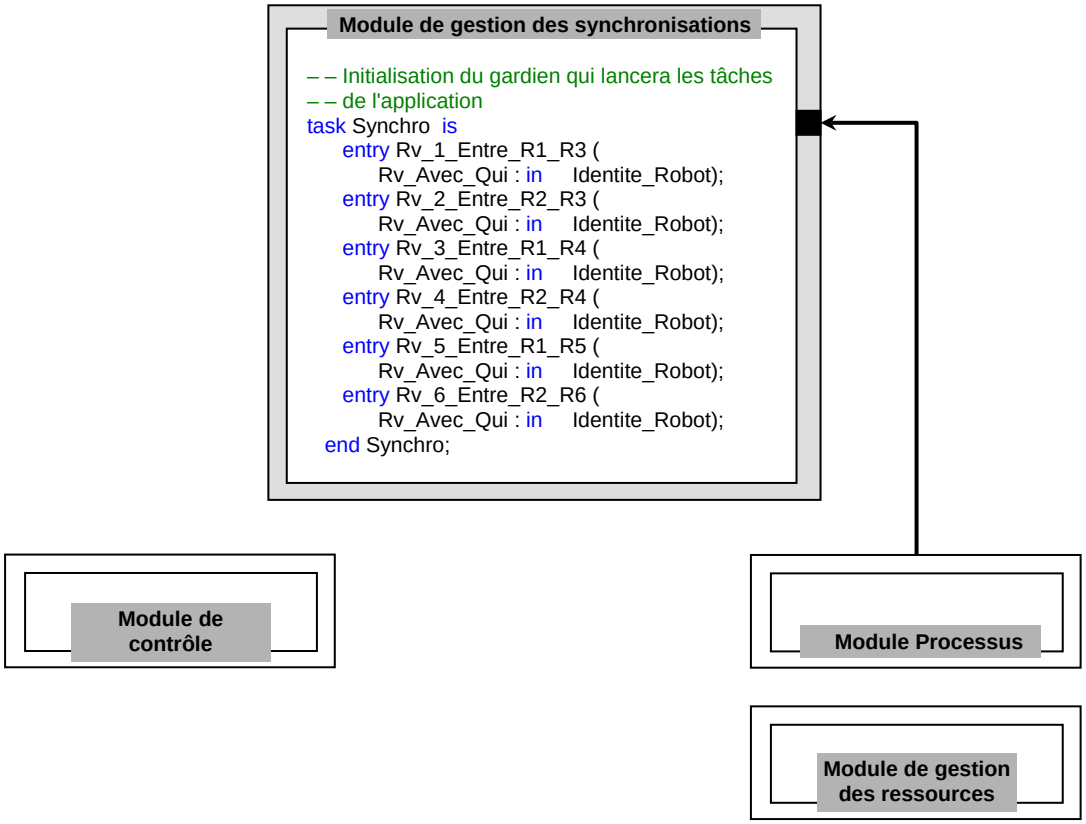


Figure 4.6 : Interface du module **de gestion des synchronisation**.

Il est implémenté sous forme d'une tâche de synchronisation qui contient 6 entrées pour un exemple de six robots et six points de rendez-vous ; voir figure 4.6. Ces entrées bloquerons un robot (processus) jusqu'à l'arrivé de son homologue, et il offre dans sa partie visible six primitives de type :

Rv_N°l_Entre_Ri_Rj : rendez-vous numéro *l* entre le robot i et le robot j avec le champ :

- Rv_Avec_Qui qui prend des valeurs dans **1 . . Nombre de robots**

3.3. Interface Module de contrôle.

Aucun module n'a de vision complète de l'état du fonctionnement de l'application, c'est pour cela que le contrôle est confié à une unité dédiée qui intervient :

- *A l'initialisation de l'application* : initialise l'ensemble des modules en leur transmettant leurs paramètres de démarrages
- *Dans la gestion des cas d'exception* : Provoque la fin prématurée de l'application

- *A la terminaison de l'application* : lorsque les modules_processus ont achevé leur exécution, elle termine l'application

3.3.1. Services requis et offerts.

Le contrôle de l'application est réalisé par un seul gardien. Ce gardien est initialisé au démarrage de l'application par le programme principal, il est implémenté dans un paquetage sous le non Gestion_Du_Controle_Robot.

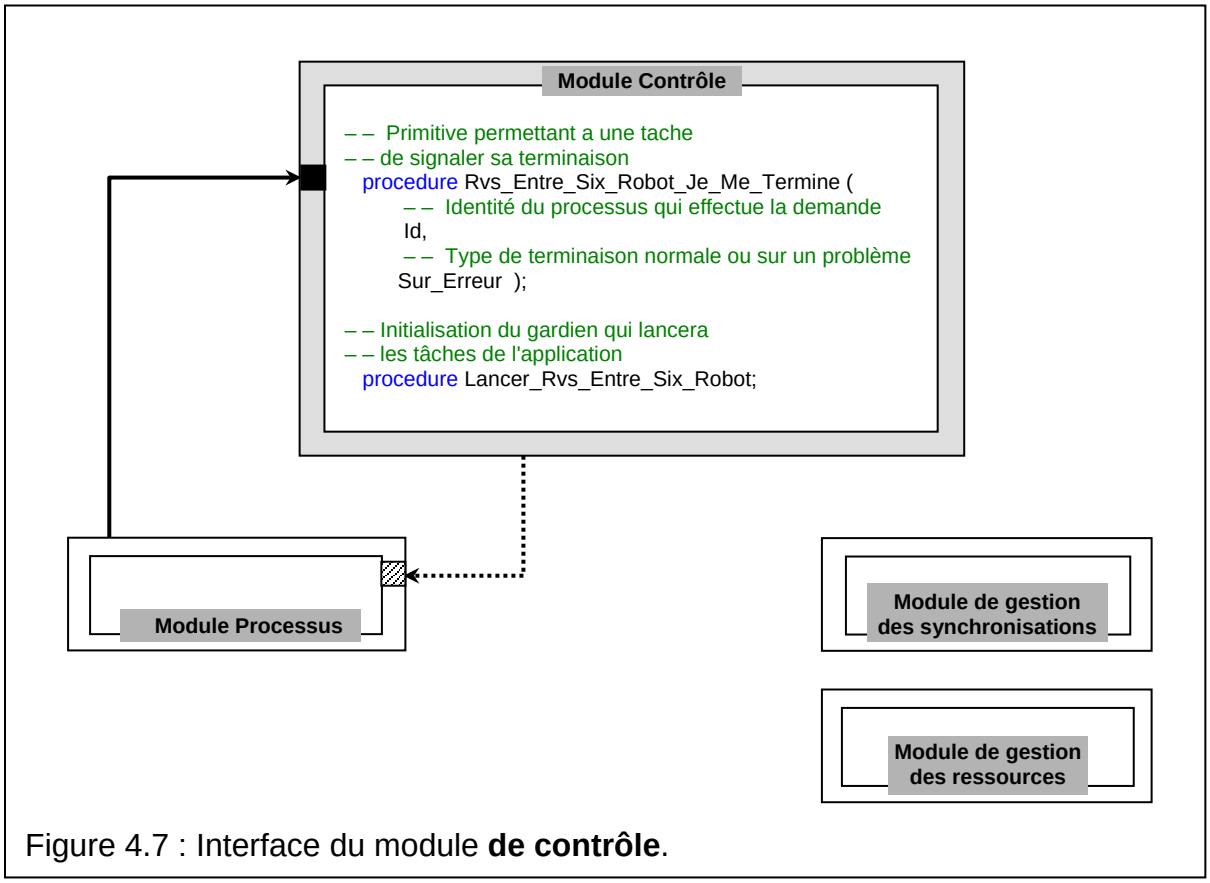


Figure 4.7 : Interface du module **de contrôle**.

Dans sa partie visible le module contrôle offre comme services deux primitives ; figure 4.7 :

- La première nommée Rvs_Entre_Six_Robot_Je_Me_Termine permet à une tâche de signaler sa terminaison par l'intermédiaire des deux champs :
  - Identité du processus qui effectue la demande
  - Le type de terminaison (normale ou sur un problème)
- La seconde nommée Lancer_Rvs_Entre_Six_Robot permettant l'initialisation du gardien qui lancera les tâches de l'application

Dans sa partie privée il réalise les services mentionnés dans sa partie visible (spécification).

4. Conclusion.

Nous avons décrit dans ce chapitre, les caractéristiques essentielles des modules Ada composant l'ensemble G-Objets générés à partir de la décomposition du réseau de Pétri modélisant l'ordonnancement des déplacements des robots. Nous avons choisi le langage Ada, d'une part parce qu'il respecte les hypothèses de base nécessaires à notre application (il est procédural, permet la manipulation du parallélisme et gère les notions de rendez-vous) [HAL 04] mais aussi pour les aspect suivants :

- La notion d'*unité de programme*, permettant de diviser l'application générée en paquetages avec des parties privées et visibles;
- La notion d'*exception*, permettant d'élaborer un mécanisme de détection des erreurs fort utile;
- La notion de *typage fort* : Plus une erreur est diagnostiquée tôt, moins elle est coûteuse à corriger. Le langage fournit des outils permettant de diagnostiquer beaucoup d'erreurs de cohérence dès la compilation.

	Introduction générale.
Chapitre 1 :	Etat de l'art sur la robotique mobile et sur la coopération entre robots.
Chapitre 2 :	Planification de trajectoire et navigation.
Chapitre 3 :	Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes.
Chapitre 4 :	Implémentation des processus parallèles à l'aide du langage de programmation ADA.
Chapitre 5 :	Conception d'une plateforme de robots mobiles autonomes et début de réalisation.
Chapitre 6 :	Simulation.
	Conclusion générale.
	Bibliographie.
	Annexe et Appendice.

Chapitre 5.

Conception d'une plateforme de robots mobiles autonomes et début de réalisation.

1. Conception de la plateforme de robotique mobile.

1.1. Introduction.

La plate forme de robotique mobile que nous souhaitons concevoir (figure 5.1), comporte dans une première phase, trois robots mobiles étagés se déplaçant entre des points de rendez-vous et échangeant des messages lorsque le rendez-vous entre deux robots a lieu effectivement. Elle est de type modulable avec la capacité d'intégrer de nouveaux modules au fur et à mesure du développement et de l'extension de la plate forme.

Le développement futur de la plateforme peut prendre les aspects suivants :

- Intégration de nouveaux éléments matériels (robots, système de localisation, etc…)
- Intégration de nouvelles applications :
  - Réordonnancement des déplacements après une panne définitif (défaillance d'un robot) (défaillance du logiciel)
  - Réordonnancement des déplacements après une panne temporaire (défaillance du logiciel)

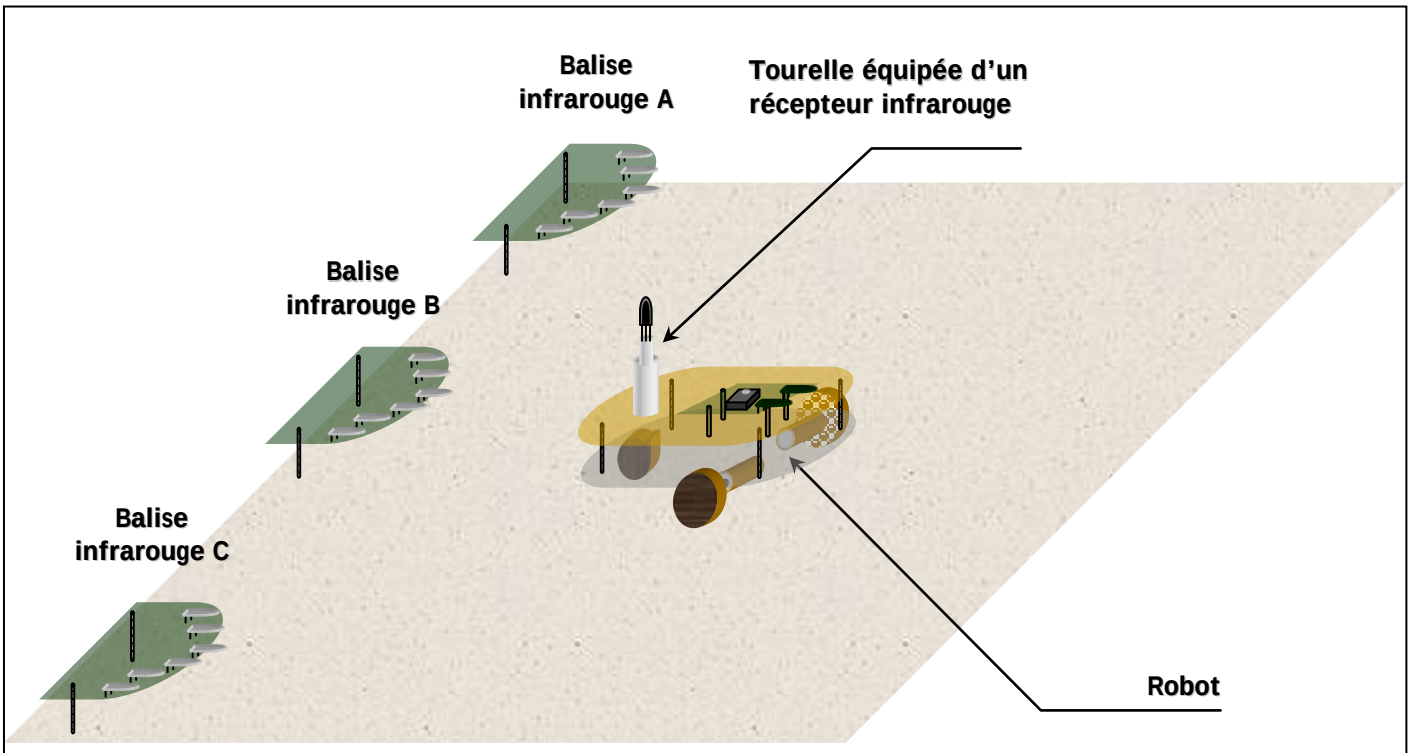


Figure 5.1 : La plateforme.

1.2. Conception des robots.

Les robots conçus sont des robots étagés autonomes mobiles et programmables. Ils sont équipés d'un système avec carte cerveau à base de microcontrôleur (16F84A) avec extensions possibles au niveau électroniques qui rendra l'électronique du robot beaucoup plus propre et donc plus fiable.

La taille du robot est d'environ (25 X 25 X 20 cm) ; sa vitesse est de 10 cm/s et en ce qui concerne l'intelligence, le robot doit être capable de faire face de façon autonome aux événements apparaissant dans son environnement.

Voici les composants de notre robot :

- Châssis en aluminium.
- Etage pour l'électronique embarquée
- 2 Roues motrices en caoutchouc et une roue libre.
- 2 moto-réducteur.
- 1 moteur Pas à Pas.
- 1 bloc d'alimentation.
- Et d'une tourelle.

Le robot est équipé d'un microcontrôleur PIC 16F84A dont le rôle est d'exécuter le code objet présent dans la mémoire *flash*, La simplicité de son architecture et sa taille raisonnable en font une cible de choix notamment pour les applications étudiant le comportement d'une grande collectivité de robots. De plus, son interaction continuelle avec son environnement permet de le considérer comme un système réactif. Il est programmé en langage assembleur ou bien en langage Basic, c'est ce qui rend la mise en œuvre du robot facile car il est composé d'un nombre de composants électroniques très réduit.

1.2.1 Le châssis.

Pour réaliser le châssis, on a utilisé de l'aluminium de 1mm d'épaisseur, découpé sous forme de disque de diamètre 25 cm. Ensuite on a découpé de chaque côté sur 15 cm l'emplacement pour les roues comme présenté sur la figure 5.2.

On a choisi l'aluminium car d'une part il présente l'avantage d'être facilement usinable et d'autre part il rend le robot léger ce qui le rend plus mobile (meilleur rapport poids / puissance des moteurs).

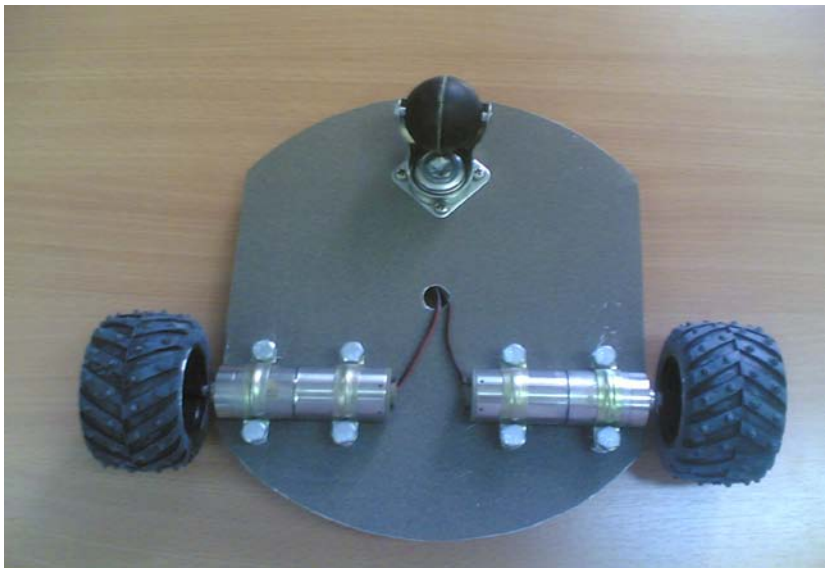


Figure 5.2 : Châssis du robot vue de dessous.

1.2.2 L'étage pour l'électronique embarquée.

Pour réaliser l'étage qui embarque l'électronique du robot, on a utilisé le bois de 7mm d'épaisseur, découpé sous la même forme que celle du châssis, on a choisi le bois pour une meilleure fixation des cartes électroniques (voir figure 5.3).

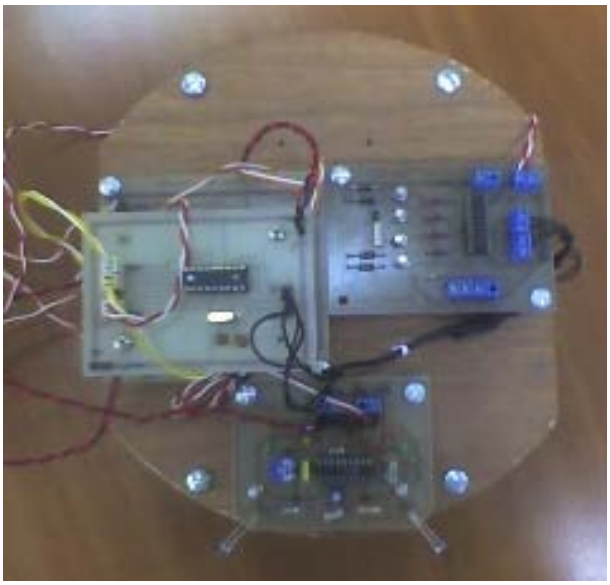


Figure 5.3 : Etage pour l'électronique du robot.

1.2.3 Les roues.

On a deux types de roues ; deux roues motrices et une roue libre :

1.2.3.1 Roues motrices.

On a utilisé deux roues de forme ronde de 50 mm de diamètre, rendues solidaire de l'axe du moto réducteur.

Le pneu est constitué d'une chambre à air permettant à la roue d'adhérer au sol (indispensable), le diamètre des roues peut être augmenté si l'on souhaite donner au mobile une vitesse plus importante ; figure 5.4.

Figure 5.4 : Roues motrices.

1.2.3.2 Roue libre.

Les deux points d'appui des roues n'étant pas suffisants pour assurer la stabilité, il était nécessaire d'employer une roue libre pour la construction du robot. Elle est ajoutée à l'arrière du robot pour assurer sa stabilité ; figure 5.5.

Figure 5.5 : Roue libre.

1.2.4 Les actionneurs.

1.2.4.1. Moto réducteur.

Les moteurs utilisés pour la propulsion du robot sont des moteurs industriels à courant continu. Ce sont des moteurs de très grande qualité et très grande fiabilité. Ils sont alimentés en 3V. Cependant, il est possible de les utiliser sans les détériorer à des tensions et puissances plus élevées ; figure 5.6.

Chaque moteur est couplé à un réducteur afin de diminuer la vitesse de sortie (et donc d'augmenter le couple).

Figure 5.6 : Moto réducteur.

1.2.4.2. Moteur Pas à Pas.

Le moteur Pas à Pas utilisé pour la commande de la tourelle est un moteur d'imprimante avec six fils (figure 5.7). Possédant un double - stator : à chacune de ces deux parties, est associé un bobinage avec un point milieu et deux phases ; en alimentant l'une ou l'autre des phases, on peut ainsi inverser l'aimantation au niveau du stator correspondant. et un rotor à 12 paires de pôles : il pourra donc occuper 48 positions différentes, la valeur de chaque pas étant de $\frac{360°}{48} = 7,5°$.

Figure 5.7 : Moteur Pas à Pas.

1.2.5. Bloc d'alimentation.

Un autre aspect important est l'alimentation du robot qui doit pouvoir être opérationnel pendant une durée d'au moins cinq heures afin d'avoir le temps de réaliser sa tâche.

Une alimentation est constituée de deux éléments : la batterie et le montage de régulation.

1.2.5.1. La batterie.

Le choix de la batterie idéale pour un robot est conditionné par différents critères.

- Tension de sortie [V] : la tension de sortie de l'accumulateur doit être supérieure ou égale à la tension d'entrée minimale de l'étage de régulation.
- Capacité de l'accus [mAh] : La capacité d'un accus est exprimé en milliampère/heure. Ce paramètre exprime quel courant il faut faire débiter à l'accus pour le vider en heure.

Leurs caractéristiques sont les suivantes (voir figure 5.8) :

Voltage	3,6 Volts
Ampérage	700 mAh

On a utilisé deux batteries montée en série qui utilise la technologie « Lithium-ion », c'est une technologie très courantes, on la retrouve dans la plupart des appareils électroniques récents demandant une grande capacité tel que : téléphones mobiles, ordinateurs portables … etc,

Chacune délivre 3,6 volts en théorie et 4,11 volts en pratique et a une capacité de 700 mAh.

Figure 5.8 : Batterie « Lithium-ion ».

1.2.5.2. Montage de régulation.

Pour l'alimentation du robot les deux batteries montée en série sont associée à un régulateur 7805.

1.3. L'électronique du robot.

Notre ambition est de mener ce projet sur plusieurs années, c'est ce qui nous a incité à construire une architecture logicielle modulable et facilement réutilisable. Les différents systèmes électroniques décrits ci-dessous sont donc reliés au processeur central selon un réseau Maître - Esclave. On peut donc rajouter de nouveaux modules facilement.

L'architecture se présente donc sous la forme suivante figure 5.9 :

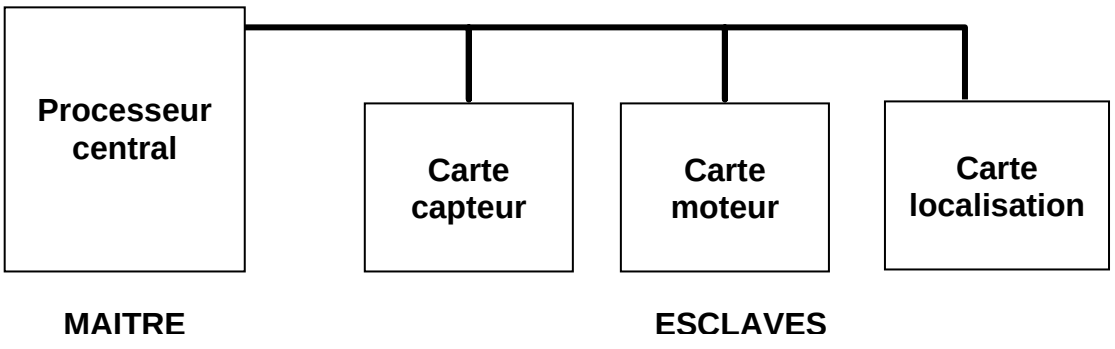


Figure 5.9 : Allure générale de l'électronique du robot.

Le processeur central (le Maître): contient toute l'intelligence du robot et interroge les différentes cartes (cartes capteurs, localisation) pour prendre ses décisions, et leur donne des ordres (carte moteurs) en conséquence.

Les différentes cartes électroniques embarquées sur le robot sont :

- Une carte électronique avec le PIC16F84 comme cerveau.
- Une carte de puissance pour les deux moto-réducteur.
- Un système de détection infrarouge IR.
- Un système de localisation à base de balise infrarouge.

1.3.1 Cerveau du robot.

Toutes les cartes comprises sur le robot sont conçues autour d'un microcontrôleur de type PIC 16F84A. Ces composants sont très simples à programmer, fiables, robustes et ne nécessitent que très peu d'électronique autour, pour pouvoir fonctionner. Les instructions qu'ils exécutent sont écrites dans la mémoire *flash* sous forme de mots binaires codé sur 14 bits. Leur programmation s'effectue en assembleur (en basic éventuellement).

Sur les cartes du robot, ces composants sont programmés pour attendre des ordres émis par la carte centrale, et exécutent les commandes correspondantes.

Le PIC possède une broche de reset qui doit être tenue à l'état haut pour que le programme contenu dans le pic puisse s'exécuter correctement. Le quartz muni de deux condensateurs est branché entre les broches OSC1 et OSC2. Il détermine la vitesse (fréquence) de travail du microcontrôleur (voir figure 5.10).

Le schéma électronique de la carte Cerveau du robot est présenté en annexe.

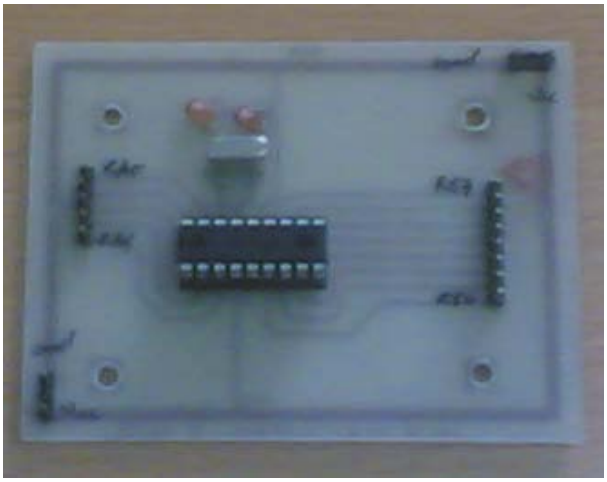


Figure 5.10 : Cerveau du robot.

1.3.2. Carte moteur : Une carte de puissance pour les moto-réducteur.

Centralise les commandes destinées aux moteurs. La commande des moteurs nécessite quatre signaux provenant du microcontrôleur. Pour chaque moteur, on peut faire varier le sens de rotation et la vitesse, le circuit **L293** est une interface de puissance spécialisée dans la commande de petits moteurs de faible puissance. Le courant maximum est de 600 mA par voie.

Le L293 est configuré pour réaliser deux ponts en H afin de piloter deux moteurs. Pour cela on utilise deux inverseurs 74HC04 ; ceux-ci ne sont pas indispensables,

cependant ils offrent l'assurance que la paire de transistors verticaux ne sera jamais passante en même temps (voir figure 5.11).

Le schéma électronique de la carte de puissance pour les moto-réducteurs est présenté en annexe.

Figure 5.11 : carte de puissance pour les moto réducteur.

1.3.3. Carte capteur : Un système de détection infrarouge IR.

Toutes les informations fournies par le capteur infrarouge sont centralisées sur cette carte.

Pour détecter un obstacle se trouvant devant le robot nous allons utiliser un radar infrarouge (voir figure 5.12).

Le schéma électronique de la carte de détection infrarouge IR est présenté en annexe.

Figure 5.12 : carte de détection infrarouge IR.

1.3.3.1. Principe de fonctionnement.

Le radar se compose de deux led émettrices infrarouge (une de chaque côté à l'avant) et d'un récepteur infrarouge au centre. Les led émettent à tour de rôle un faisceau lumineux infrarouge. Au cas où le faisceau est réfléchi, il revient percuter le récepteur infrarouge. En fonction de la led qui émettait, on sait de quel côté se trouve l'obstacle et l'on peut engager une manoeuvre de contournement (voir figure 5.13).

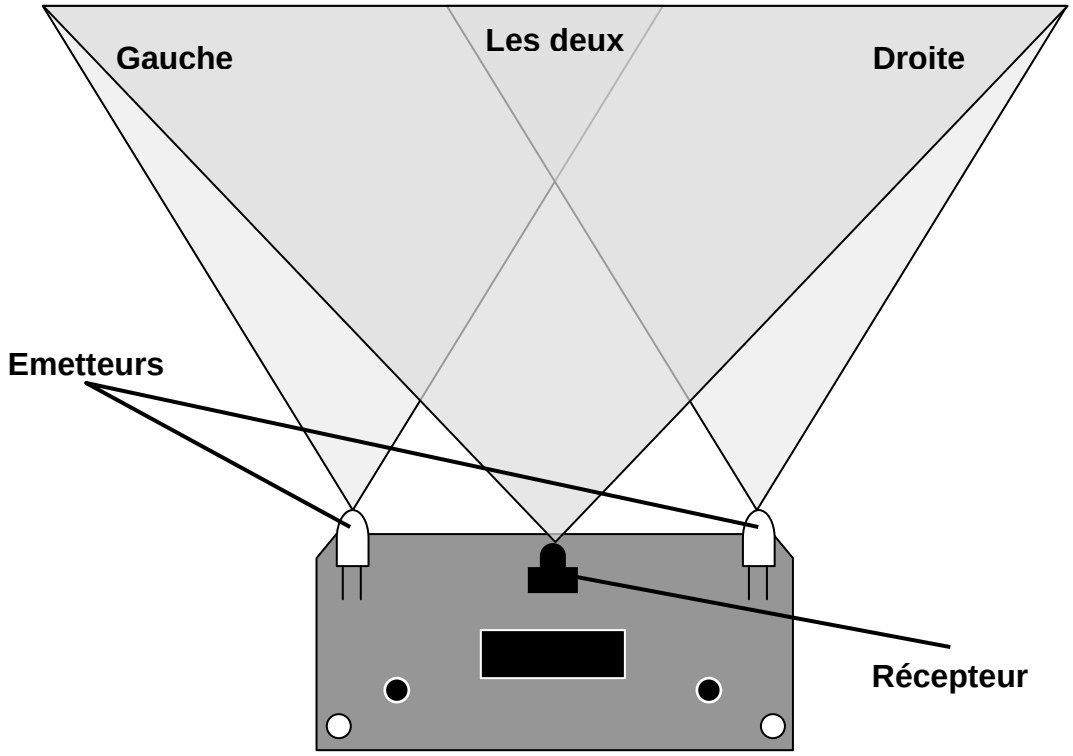


Figure 5.13 : Principe de détection de la carte infrarouge.

1.3.3.2. Le récepteur infrarouge.

Figure 5.14 : Récepteur infrarouge.

Le récepteur infrarouge est une cellule à trois pattes, elle ce compose essentiellement d'une photodiode, préamplificateurs avec commande automatique de gain ainsi qu'un filtre passe-bande. La figure 5.14 montre le boîtier du composant. Le récepteur réagit à un faisceau infrarouge modulé à une fréquence de 40 khz. Sa sortie passe au niveau 0 lors de la réception. L'avantage de travailler avec une fréquence de 40 khz est qu'on se prémunisse contre toute lumière parasite.

1.3.3.3. But à atteindre.

Il va falloir qu'on génère des trains d'impulsions de 40khz vers les diodes émettrices et tester si la sortie du récepteur passe à 0. En cas d'obstacle, on allume une LED du côté correspondant.

1.3.4. Carte localisation : Un système de localisation à base de balise infrarouge.

Gère toute la partie localisation du robot par triangulation à l'aide de balises infrarouges figure 5.15.

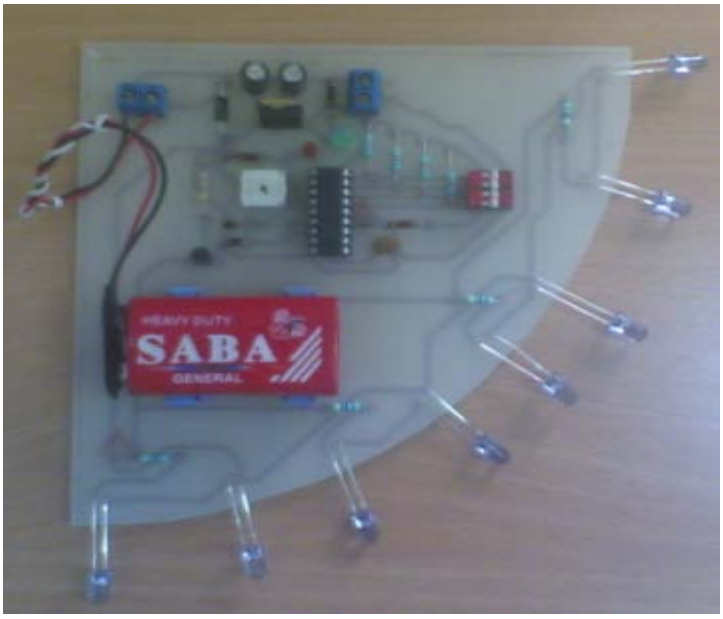


Figure 5.15 : Balise infrarouge active.

Le schéma électronique de la carte Balise infrarouge active est présenté en annexe.

1.3.4.1. Principe de fonctionnement.

Ce système de positionnement utilisera **3 balises** fixes qui seront positionné comme indiqué sur la figure 5.14. A partir de la connaissance de l'emplacement de ces 3 balises on pourra en déduire la position absolue du robot par mesure d'angles ou de distances. Pour la communication entre le robot et les balises on peut utiliser différentes ondes : acoustiques (ultrasons), électromagnétiques (radio), ou lumineuse telles que le LASER ou les infrarouges. Le LASER possède l'énorme avantage de sa précision mais son prix élevé nous a détourné de ce système. Nous avons décidé d'utiliser les **infrarouges** (**IR**) qui, modulés, permettent de ne pas être brouillés par les lumières parasites. Ensuite nous avons privilégié des balises autonomes qui ne reçoivent aucun signal : elles ne font qu'émettre un signal modulé composant un code propre à chaque balise et diffusant dans tout l'espace. C'est ainsi que nous avons débouché sur ce concept :

Nous disposons de trois balises sur le contour de l'environnement dans lequel naviguent les robots. Ces balises ont en particulier : Une couronne de LED diffusants sur 180° ou sur 90°, un système de commandes (PIC 16f84a) des batteries pour alimenter ces LED et quatre microswitchs permettant de définir un numéro propre à la balise. Les numéros sont attribués de façon croissante dans le sens inverse des aiguilles d'une montre. Nous avons ainsi trois "phares".

Le robot lui est équipé d'une tourelle actionnée par un moteur pas à pas et tournant sur elle même. Sur cette tourelle sera positionné un récepteur **infrarouge** (**IR**) très directif (cache). Un système de commande (PIC 16f84a) permettra donc d'associer une balise à un angle relatif à la direction principale du robot.

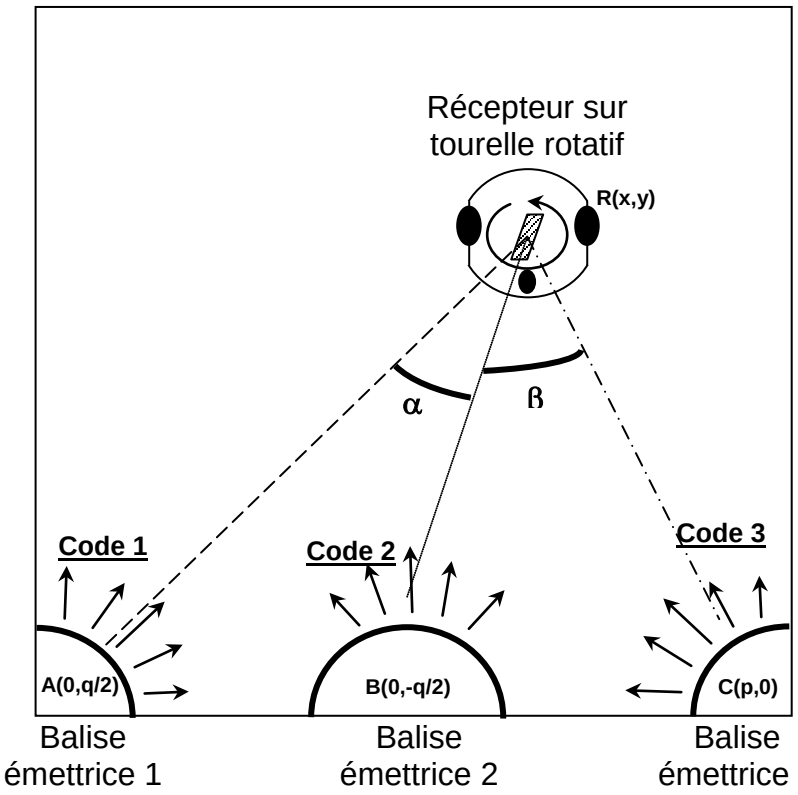


Figure 5.16 : Un système de localisation à base de balises infrarouges.

1.3.4.2. Algorithme de triangulation.

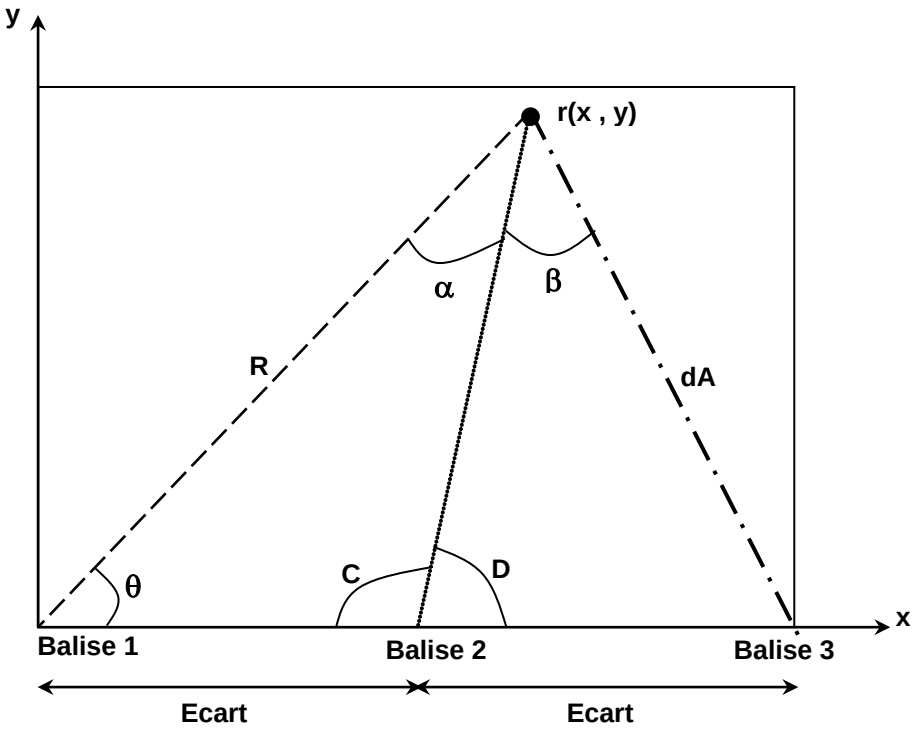


Figure 5.17 : Principe de triangulation.

L'algorithme :

- La position du robot R est donnée par les coordonnées x et y.
- Les angles α et β sont mesurés entre la Balise 1 et la Balise 2 et entre la Balise 2 et la Balise 3 respectivement
- Soit Balise1, Balise2, Balise3 les positions des balises 1, 2 et 3.
- Soit écart les distances entre la Balise 1 et la Balise 2 et entre la Balise 2 et la Balise 3.
- Soit dA la distance entre R et Balise 3.
- Soit D l'angle forme par R, Balise2, Balise3.
- Soit C l'angle forme par R, Balise2, Balise1.

Les formules du triangle rectangle nous donne naturellement :

$$x = R \bullet \cos(\theta)$$

$$y = R \bullet \sin(\theta)$$

D'après les formules du triangle quelconque, on peut écrire :

$$\frac{R}{\sin(C)} = \frac{ecart}{\sin(\alpha)} \quad \text{donc} \quad R = \frac{ecart \bullet \sin(C)}{\sin(\alpha)}$$

Or :

$$C = 180 - (\alpha + \theta) \quad \text{et} \quad \sin(180 - angle) = \sin(angle)$$

Donc :

$$R = \frac{ecart \bullet \sin(\alpha+\theta)}{\sin(\alpha)}$$

D'après les formules du triangle, on peut écrire :

$$\frac{dA}{\sin(\theta)} = \frac{2 \bullet ecart}{\sin(\alpha+\beta)} \qquad dA = \frac{2 \bullet ecart \bullet \sin(\theta)}{\sin(\alpha+\beta)}$$

$$\Rightarrow$$

$$\frac{dA}{\sin(D)} = \frac{ecart}{\sin(\beta)} \qquad dA = \frac{ecart \bullet \sin(D)}{\sin(\beta)}$$

Donc :

$$\frac{2 \bullet \sin(\theta)}{\sin(\alpha+\beta)} = \frac{\sin(D)}{\sin(\beta)}$$

Or :

$$D = 180 - C = \alpha + \theta$$

Ce qui nous amène à :

$$2 \bullet \sin(\theta) \bullet \sin(\beta) - \sin(\alpha + \beta) \bullet \sin(\alpha + \theta) = 0$$

$$2 \bullet \sin(\theta) \bullet \sin(\beta) - \sin(\alpha + \beta) \bullet [\sin(\alpha) \bullet \cos(\theta) + \cos(\alpha) \bullet \sin(\theta)] = 0$$

$$\sin(\theta) \bullet [2 \bullet \sin(\beta) - \cos(\alpha) \bullet \sin(\alpha + \beta)] = \sin(\alpha + \beta) \bullet \sin(\alpha) \bullet \cos(\theta)$$

$$\frac{\sin(\theta)}{\cos(\theta)} = \frac{\sin(\alpha) \bullet \sin(\alpha+\beta)}{2 \bullet \sin(\beta) - \cos(\alpha) \bullet \sin(\alpha+\beta)}$$

Donc :

$$\theta = \arctan\left[\frac{\sin(\alpha) \bullet \sin(\alpha+\beta)}{2 \bullet \sin(\beta) - \cos(\alpha) \bullet \sin(\alpha+\beta)}\right]$$

1.3.4.3. Tourelle commandée par un moteur pas-à-pas unipolaire.

Pour réceptionner les signaux infrarouges émis par les trois balises actives on a placé notre récepteur infrarouge sur une tourelle commandé par un moteur Pas à Pas pour balayé toutes la surface désirée ; et pour récupérer les deux angles α ; β indiqués sur la figure 5.16 qui seront introduit dans un algorithme de triangulation pour calculer la position courante du robot mobile.

Principe de fonctionnement.

C'est un moteur possédant un nombre important de positions stables successives séparées entre elles par un Pas, le nombre de Pas définissant le moteur correspondant à un tour. Dans notre cas, le moteur est un 48 Pas, c'est-à-dire qu'il faudra 48 impulsions dans une ou plusieurs bobines du moteur pour exécuter un tour complet. [CRO 97]

Chacune des 4 bobines du moteur devra être commander dans un ordre bien établit, cet ordre définissant le sens de rotation du moteur.

Pour faire tourner un moteur Pas à Pas, il faut alimenter ses bobines suivant des séquences bien définies.

Il existe deux types de séquences :

- Les séquences en Pas complet : on obtient 7,5° par Pas, et il faudra faire 12 fois le tableau de séquences (tableau 5.1) pour faire un tour complet (ou 48 Pas) du moteur.
- Les séquences en demi-Pas complet : on obtient 3,75° par Pas, et il faudra faire 12 fois le tableau de séquences (tableau 5.2) pour faire un tour complet (ou 96 Pas) du moteur.

	Enroulement				
Numéros des séquences	**RB0/Jaune1**	**RB1/Jaune2**	**RB2/ Gris2**	**RB3/Gris1**	**Valeur BCD**
1	0	1	0	1	5
2	0	0	1	1	3
3	1	0	1	0	10
4	1	1	0	0	12
1	0	1	0	1	5

Tableau 5.1. Rotation du moteur par séquence en Pas complet dans le sens anti-horaire, angle de Pas : 7,5°

Pour faire tourner le moteur dans le sens horaire, il suffit de commencer par la séquence n°4, pus n°3, etc…

	Enroulement				
Numéros des séquences	**RB0/Jaune1**	**RB1/Jaune2**	**RB2/ Gris2**	**RB3/Gris1**	**Valeur BCD**
1	1	1	0	0	12
2	1	0	0	0	8
3	1	0	1	0	10
4	0	0	1	0	2
5	0	0	1	1	3
6	0	0	0	1	1
7	0	1	0	1	5
8	0	1	0	0	4
1	1	1	0	0	12

Tableau 5.2. Rotation du moteur par séquence en demi-Pas dans le sens horaire, angle de Pas : 3,75°

2. Conclusion.

Nous avons présenté dans ce chapitre les principales caractéristiques du robot qui sont également ses avantages à savoir :

- Un concept modulaire, permettant une flexibilité pour les développements et les nouvelles applications.
- Une électronique basée autour d'un PIC 16F84a qui a rendu l'application plus fiable du fait de l'utilisation d'un nombre réduit de circuits logiques.
- Une mécanique simple qui a rendu la commande du robot facile.
- Un système de localisation basé sur la triangulation

- Il reste à compléter cette configuration avec un module de communication permettant aux robots de connaître leurs positions relatives dans l'environnement
  - Cette communication peut être de nature centralisée en passant par un seul nœud de communication (un seul ordinateur)
  - Cette communication peut être de nature décentralisée (une liaison de type point à point)

	Introduction générale.
Chapitre 1 :	Etat de l'art sur la robotique mobile et sur la coopération entre robots.
Chapitre 2 :	Planification de trajectoire et navigation.
Chapitre 3 :	Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes.
Chapitre 4 :	Implémentation des processus parallèles à l'aide du langage de programmation ADA.
Chapitre 5 :	Conception d'une plateforme de robots mobiles autonomes et début de réalisation.
Chapitre 6 :	Simulation.
	Conclusion générale.
	Bibliographie.
	Annexe et Appendice.

Chapitre 6.

Simulation.

1. Planification de trajectoire.

Pour un environnement contenant l'obstacle de la figure (2.3) chapitre 2, les cartes 2D de potentiel obtenues par les deux algorithmes sont représentées par les figures (6.2) et (6.3) ci-dessous.

Dans le cas où la métrique choisie est de type *Minkowski* $d = (|x|^n + |y|^n)^{1/n}$ et pour *n* = 1 cette métrique devient $d = |x| + |y|$, alors on obtient les mêmes cartes de potentiel pour les deux algorithmes.

Pour un robot dont la position courante est [6,6] et la position but [6,9] plusieurs trajectoires sont possibles et celle qui est retenue réalise une descente du gradient du point courant vers le point cible en minimisant distance et changement de sens.

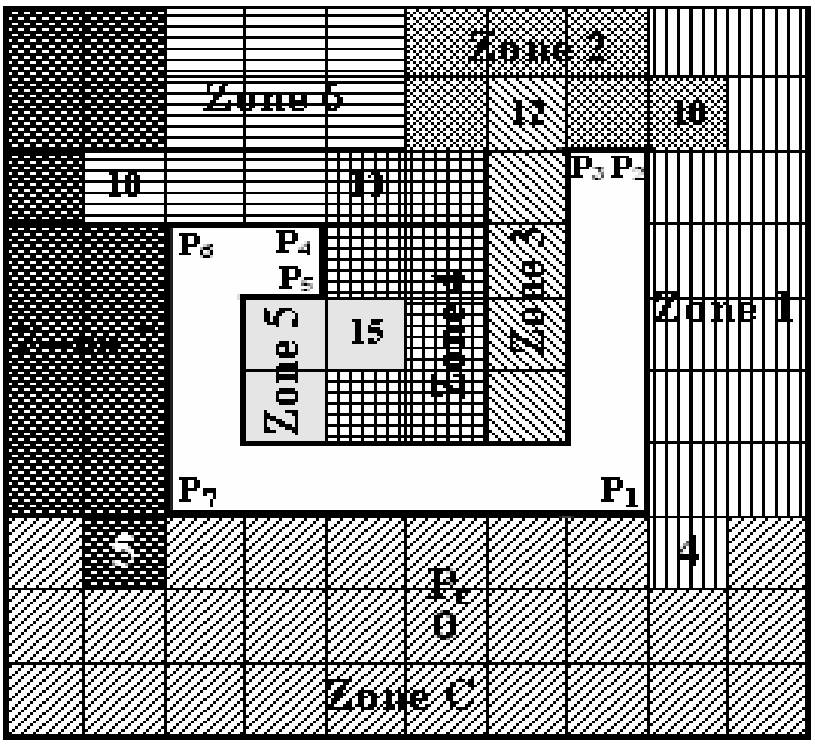


Figure 6.1 : Carte 2D du potentiel par l'algorithme basé sur la définition des points caractéristiques de l'environnement.

La figure 6.1 représente un environnement découpé en zones, où chacune d'elles subit l'influence d'un point caractéristique (c'est le potentiel minimum de cette zone). Ce qui correspond au graphe de diffusion du potentiel figure 2.5.

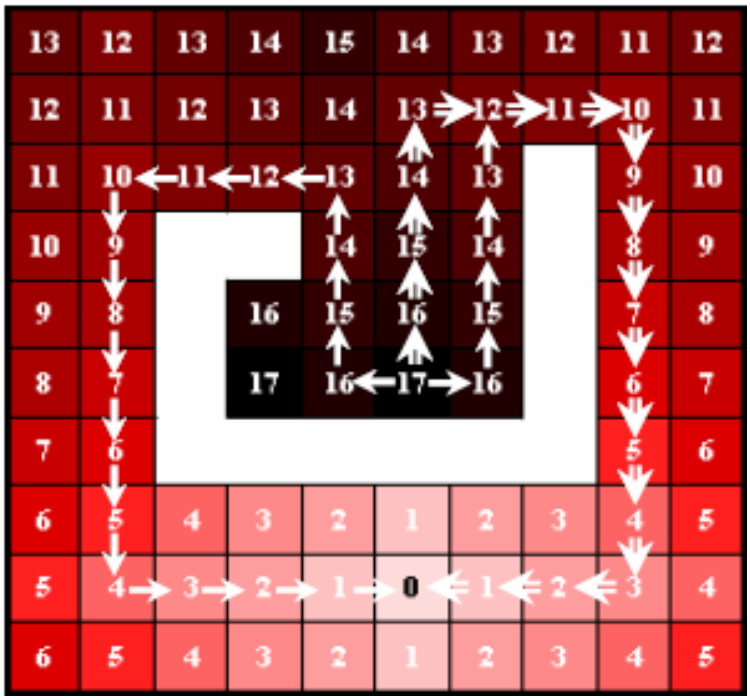


Figure 6.2: Carte 2D du potentiel par l'algorithme basé sur la notion de voisinage et contournement d'obstacles.

La figure 6.2 représente une carte complète du potentiel avec une matérialisation de la meilleure trajectoire selon les critères retenus (minimisation de distance et de changement de sens).

Et la figure 6.3 est une illustration 3D de la figure 6.2.

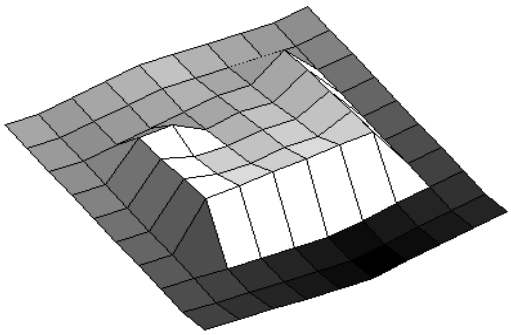


Figure 6.3 : Carte 3D du potentiel par l'algorithme.

1.1. Etablissement d'une trajectoire.

Connaissant une carte 2D du potentiel de l'environnement et la position du robot au sein de cette carte, il est possible de calculer une trajectoire pour rejoindre le but. Il s'agit tout simplement de réaliser une descente du gradient du point courant du

mobile vers le point but. Dans le cas où plusieurs trajectoires sont possibles, l'utilisation des techniques d'optimisation intégrant les contraintes géométriques permet de trouver la meilleure trajectoire possible parmi celles qui sont solutions du problème. Les fonctionnelles les plus utilisées sont celles qui sont basées sur la minimisation d'une distance et du nombre de changement de direction.
Une fois la trajectoire est générée, il est intéressant de prévoir son lissage afin de minimiser les accélérations.

1.2. Conclusion.

La taille raisonnable des cartes topologiques utilisées en robotique mobile montre que ces deux algorithmes sont suffisamment efficaces en pratique. Même si les difficultés (temps de calcul et espace de mémoire consommé) sont moins importantes pour la méthode de diffusion *Wave front propagation* que celles basées sur la propagation des potentiels à partir des points caractéristiques, on estime qu'elles sont de même grandeur et d'ordre polynomial. Toute fois, la méthode basée sur l'identification des points caractéristiques est mieux adaptée à un environnement dynamique, où la position courante d'un obstacle ne modifie que partiellement la carte de potentiel sans génération de minimums locaux.
Dans le cas où l'environnement est constitué d'obstacles fixe et mobile, la démarche à observer est la suivante :

- On établit une carte de potentiel 2D sans minimum local avec l'une des 2 méthodes en intégrant uniquement les obstacles fixes (environnement statique)
- intégrer les obstacles mobiles en identifiants les cellules et les zones qu'ils occupent.
- pour la première méthode : si la trajectoire retenue ne passe pas par les zones susceptibles d'être modifiées par la présence de l'obstacle, le robot mobile continue de se diriger vers la cellule cible en ayant une carte de potentiel 2D partiellement correcte mais suffisante pour atteindre son objectif. Dans le cas contraire, une actualisation de la carte de potentiel s'impose et elle se fait uniquement à partir de la zone contenant l'obstacle et les zones sous son influence.
- pour la deuxième méthode : si le niveau de potentiel de la cellule occupée par le robot mobile est inférieure au potentiel le plus bas des cellules occupées

par l'obstacle mobile, le robot va continuer sa trajectoire vers son objectif avec une carte de potentiel partiellement correcte. Dans le contraire la carte de potentiel est réactualisée à partir du niveau de la cellule occupée par l'obstacle et ayant le plus faible niveau de potentiel.

2. Implémentation des processus parallèles à l'aide du langage de programmation ADA.

Considérons un système constitué de 6 robots et 6 points de rendez-vous. La figure 6.1 représente le réseau de Petri global correspondant à ce système. En ordonnant les points de rendez-vous de chacun des robots, nous obtenons le tableau suivant :

Robots	r1	r2	r3	r4	r5	r6
Points rendez vous	L_1 L_3 L_5	L_2 L_4 L_6	L_1 L_2	L_3 L_4	L_5	L_6

Tableau 6.1 : Ordonnancement des rendez-vous

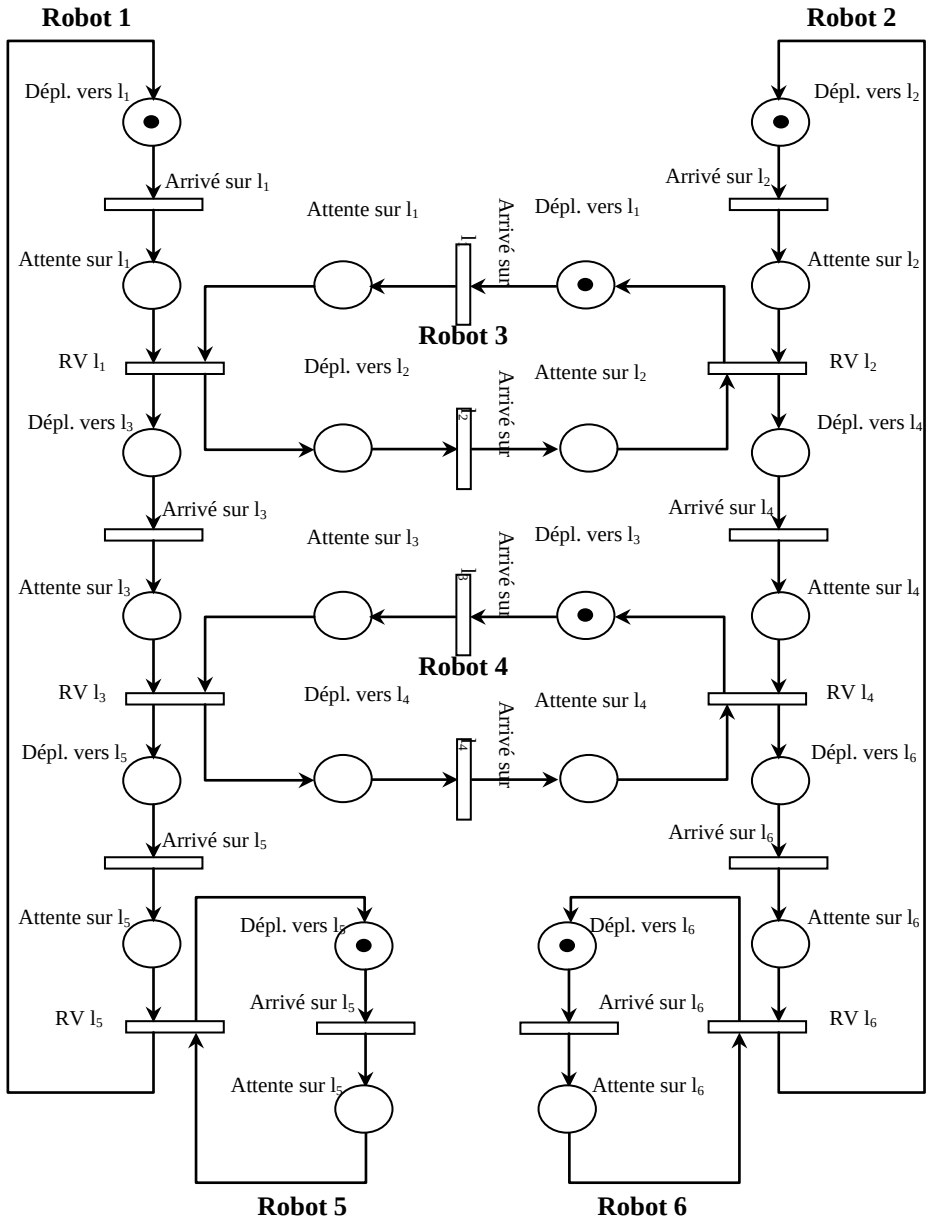


Figure 6.4 : réseau de Pétri modélisant les visites des points de rendez-vous.

Le calcul du nombre de processus instanciés s'effectue à partir du marquage du modèle initial. Le nombre de processus instanciés est égal à la cardinalité du marquage de l'ensemble des places reliant les transitions du processus entres-elles.
Les figures de (6.5 à 6.17) représentent l'exécution de l'application durant un cycle et ce pour les six robots.

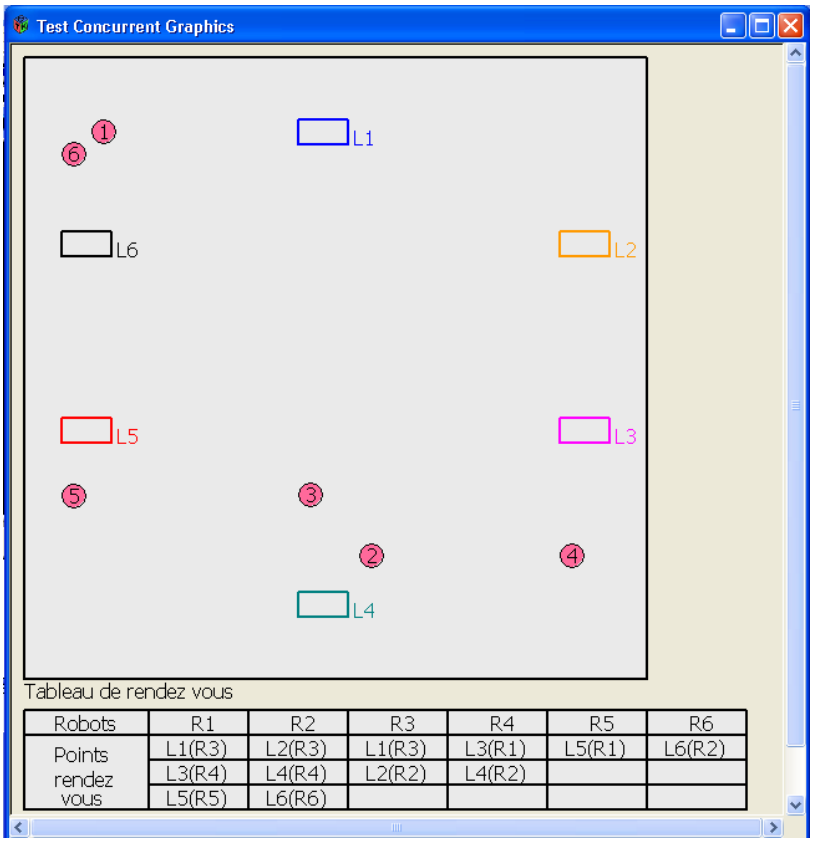


Figure 6.5 : Déplacement des robots vers leurs premiers rendez-vous.

La figure 6.5 est une image instantanée du déplacement des six robots vers leurs points de rendez vous respectifs comme il est indiqué dans le tableau des rendez-vous. Ceci correspond en réalité à l'activation des six tâches robots.

La figure 6.6 montre que le robot 5 a atteint son point de rendez-vous, alors que les autres robots sont toujours en déplacement.
La tâche relative au robot n° 5 est de signaler sa présence au point de rendez-vous au module de synchronisation, qui a son tour la met en position d'attente.

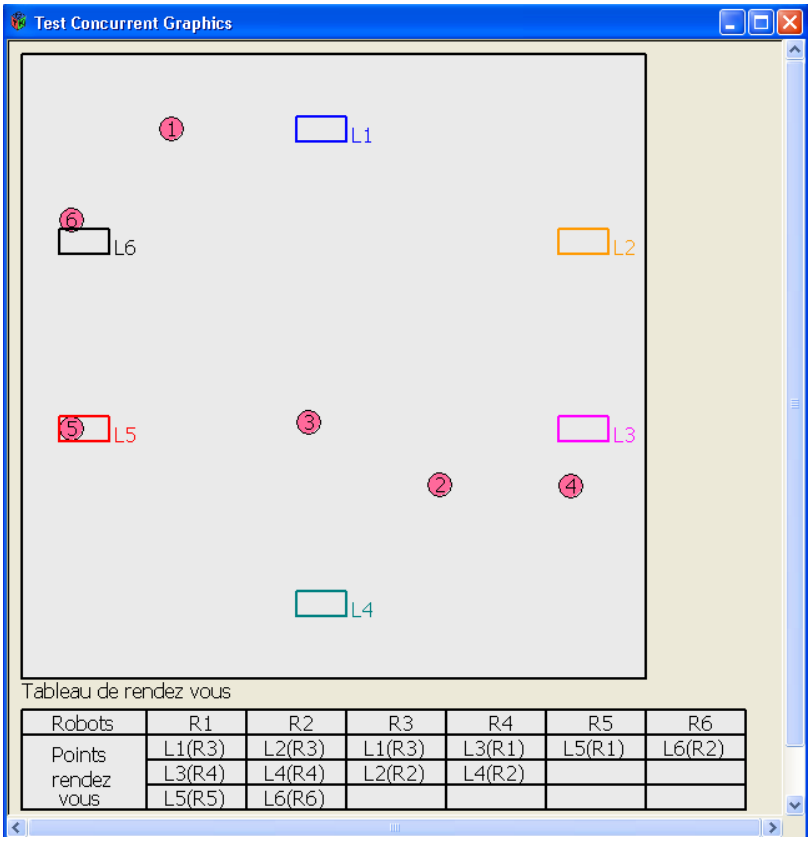


Figure 6.6 : Le robot R5 a atteint le premier son premier point de rendez-vous L5, il est en attente de son homologue le robot R1.

Les figures 6.7, 6.8, 6.9 et 6.10 montrent les arrivées respectives des robots R5, R4, R1 et R2 à leurs premiers points de rendez-vous et l'attente de leurs homologues respectifs.

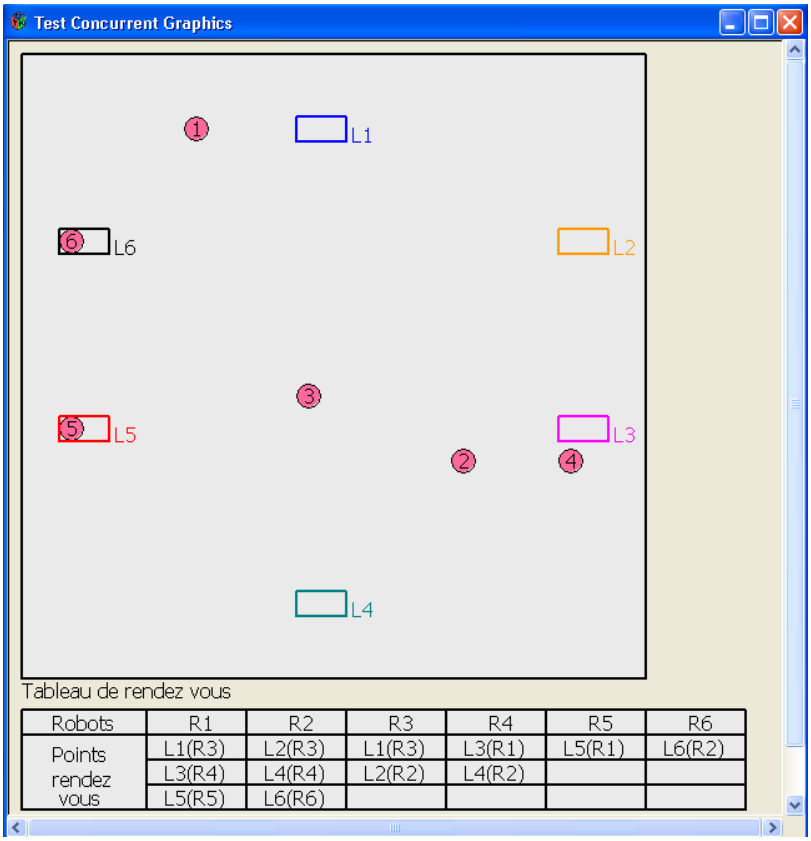


Figure 6.7 : Le robot R6 a atteint son premier point de rendez-vous L6, il est en attente de son homologue qui est le robot R2.

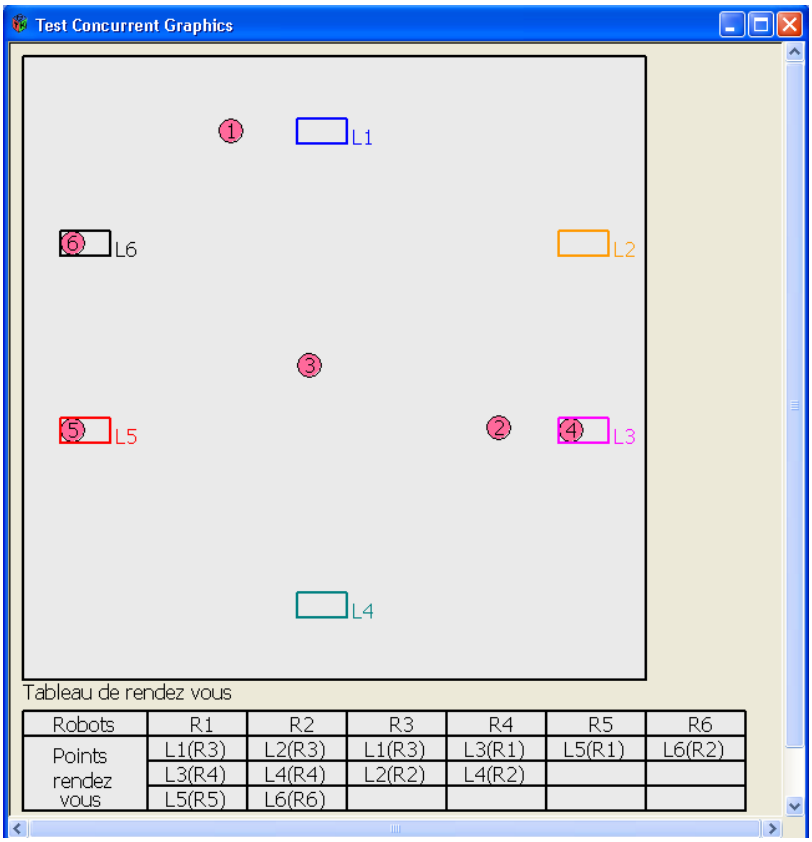


Figure 6.8 : Le robot R4 a atteint son premier point de rendez-vous L3 et il est en attente de son homologue le robot R1.

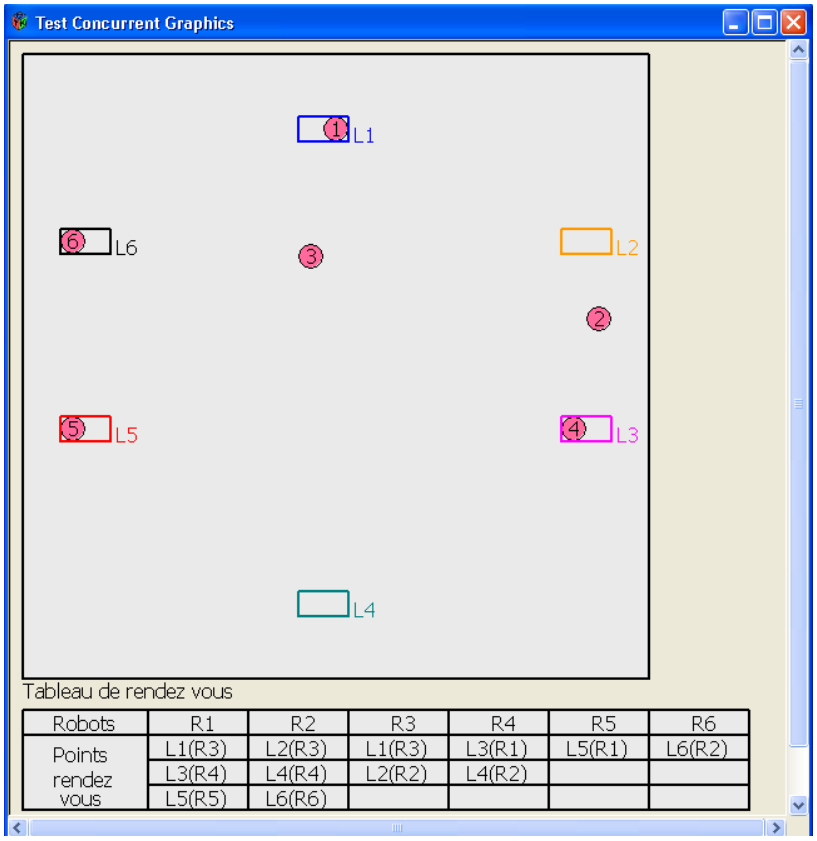


Figure 6.9 : Le robot R1 a atteint son premier point de rendez-vous L1 et il est en attente de son homologue le robot R3.

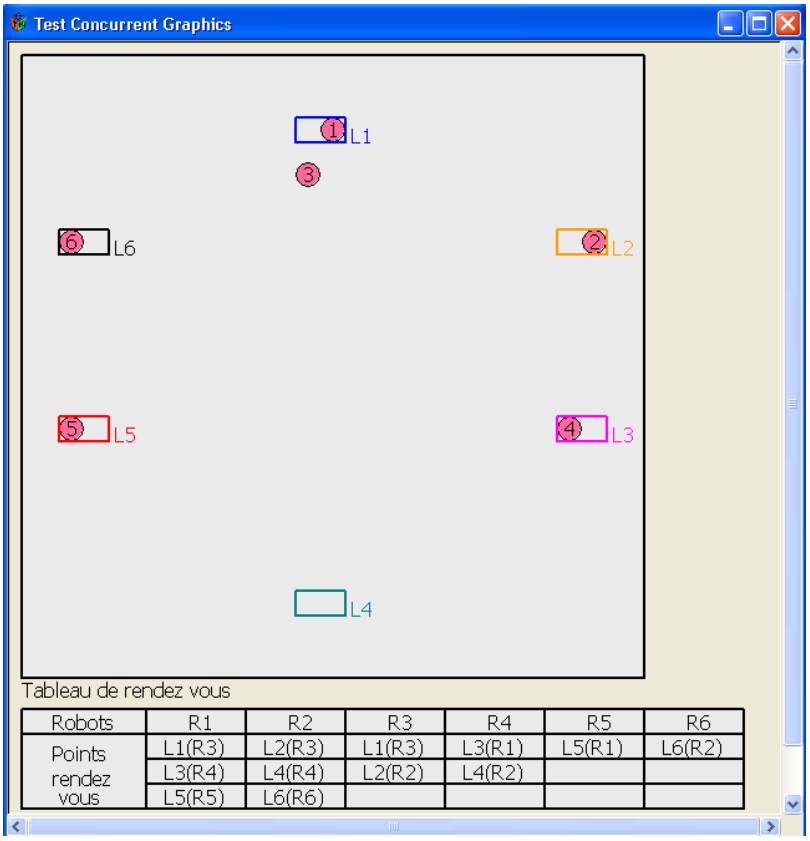


Figure 6.10 : Le robot R2 est arrivé à son premier point de rendez-vous L2, il est en attente de son homologue le robot R3.

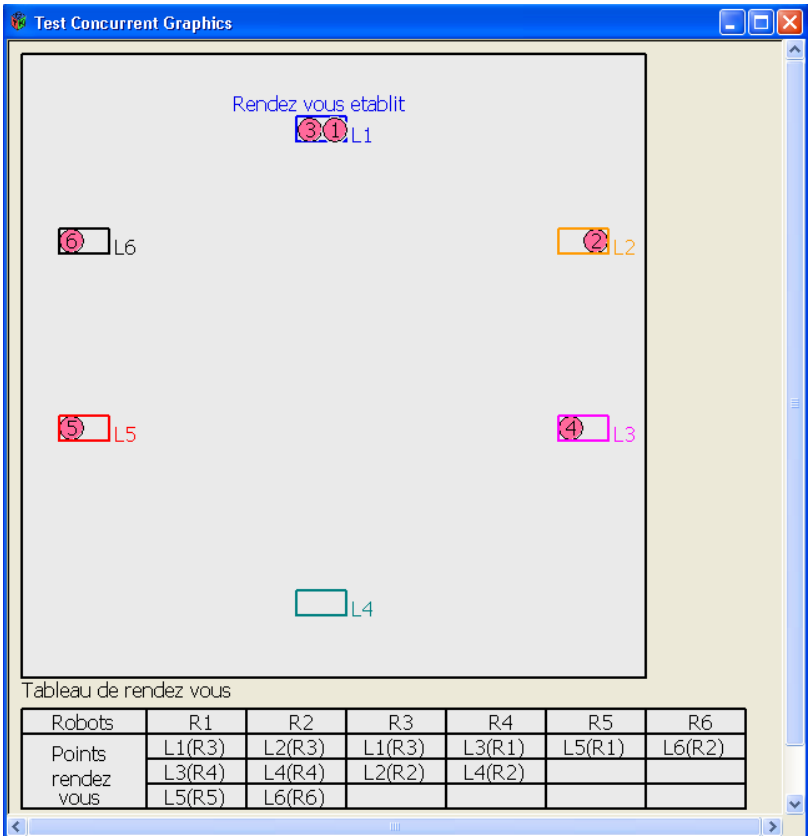


Figure 6.11 : Premier rendez-vous établit entre les deux robots R1 et R3 au point de rendez-vous L1.

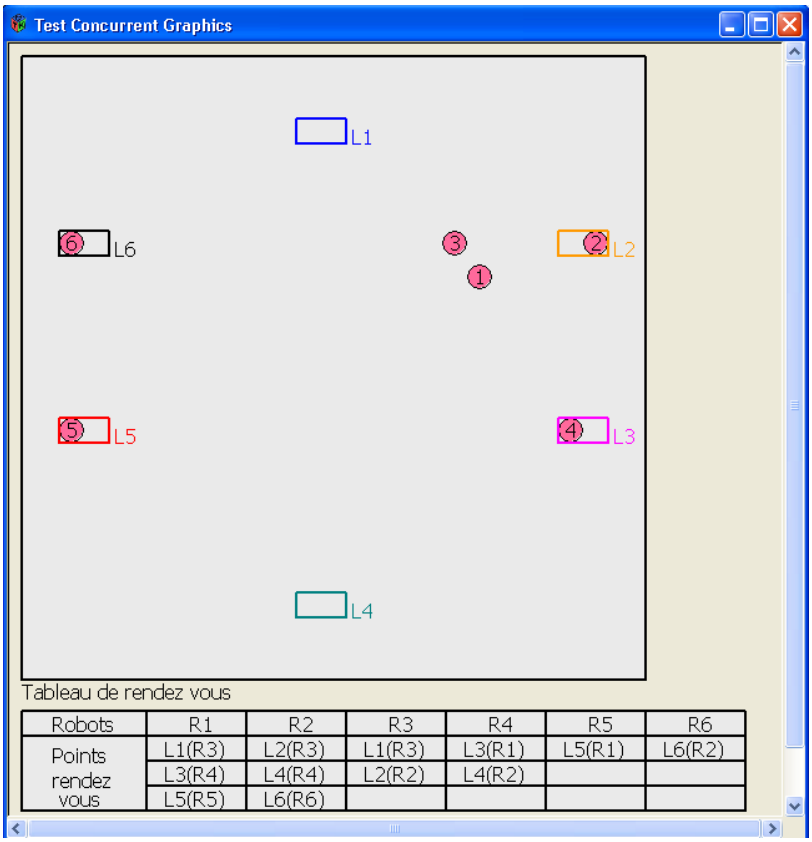


Figure 6.12 : Les deux robots R1 et R3 reprennent leurs chemins vers leurs prochains points de rendez-vous respectifs.

La figure 6.11 montre un rendez-vous établi entre les robots R1 et R3 au point de rendez-vous L1.
Le module de synchronisation établit la communication point à point entre les deux robots pour une liaison synchrone.

La figure 6.12 montre le déplacement des robots 1 et 3 vers leurs prochains points de rendez-vous après leurs libérations par le module de synchronisation.

Les figures 6.13, 6.14, 6.15, 6.16 et 6.17 montrent des rendez-vous réussits entre les couples respectifs de robots (R2,R3), (R1,R4), (R1,R2), (R1,R5) et (R2,R6)

A l'issu de cette série d'images, un nouveau cycle commence.

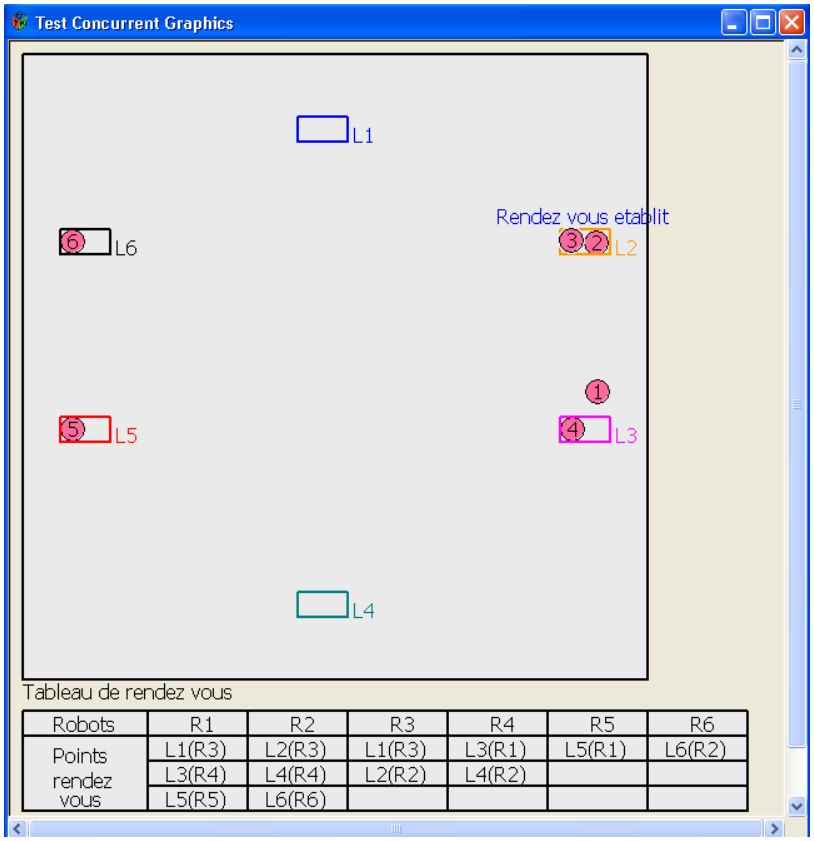


Figure 6.13 : Rendez-vous établit entre les deux robots R2 et R3 sur le point de rendez-vous L2.

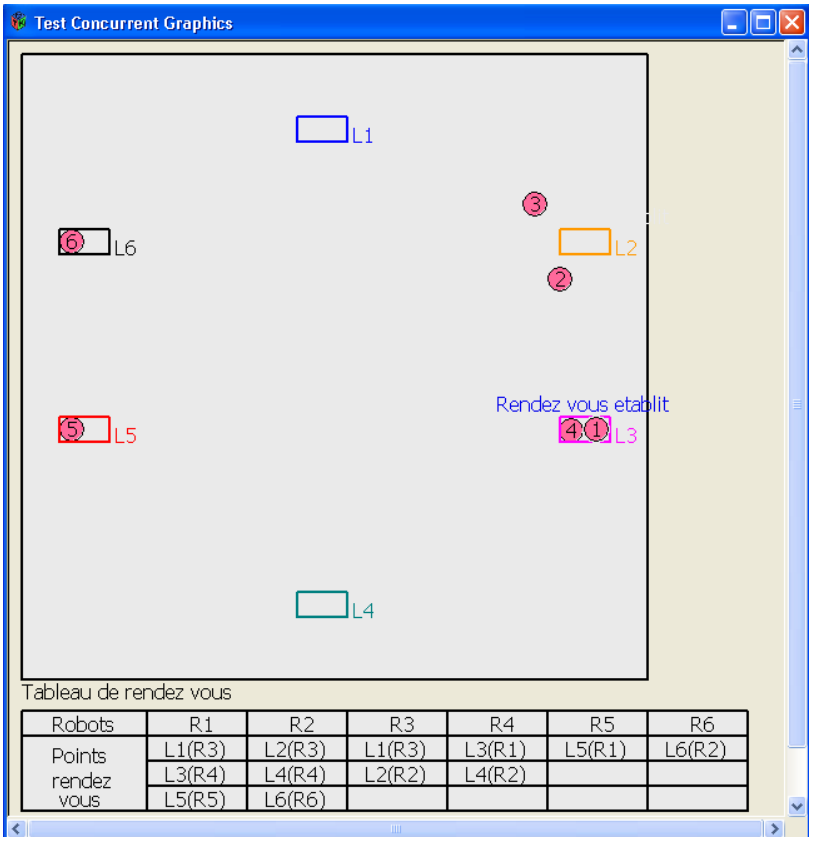


Figure 6.14 : Rendez-vous établit entre les deux robots R1 et R4 sur le point de rendez-vous L3.

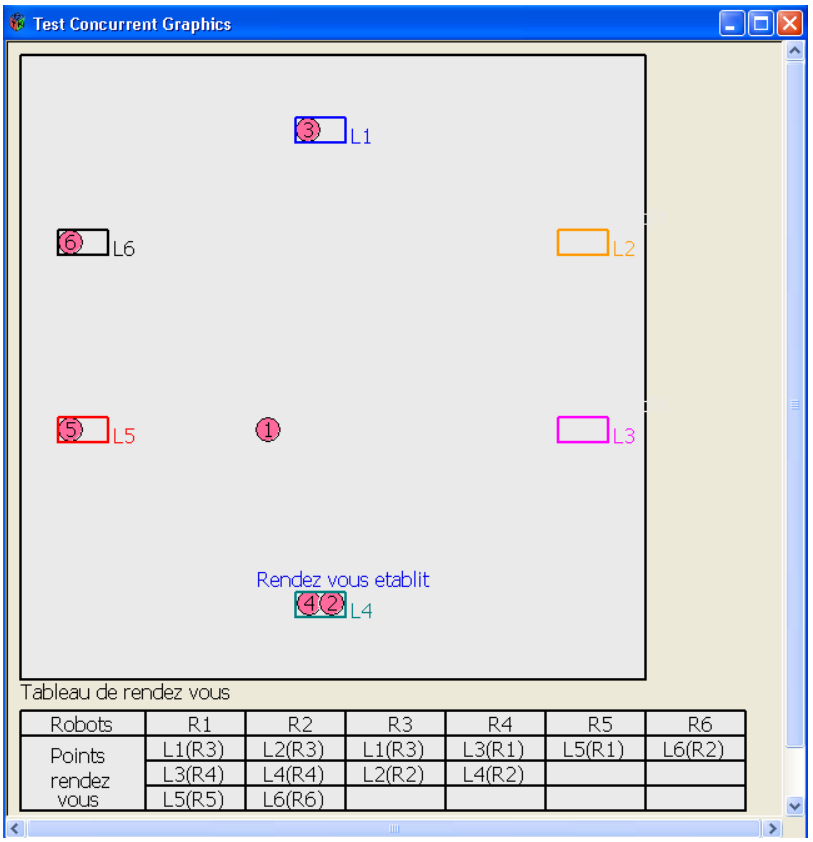


Figure 6.15 : Rendez-vous établit entre les deux robots R2 et R4 sur le point de rendez-vous L4.

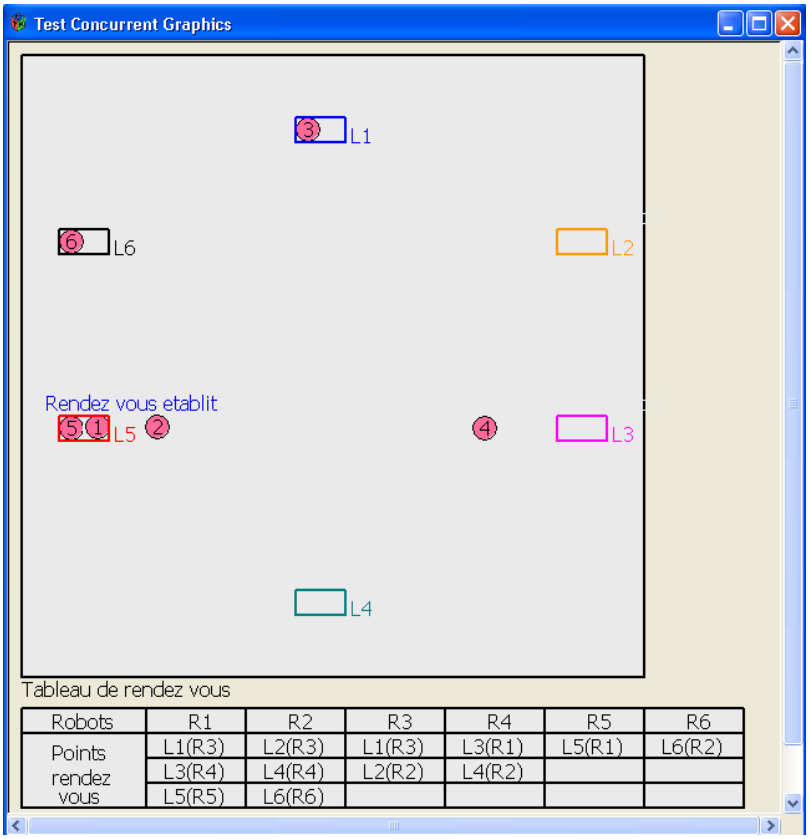


Figure 6.16 : Rendez-vous établit entre les deux robots R1 et R5 sur le point de rendez-vous L5.

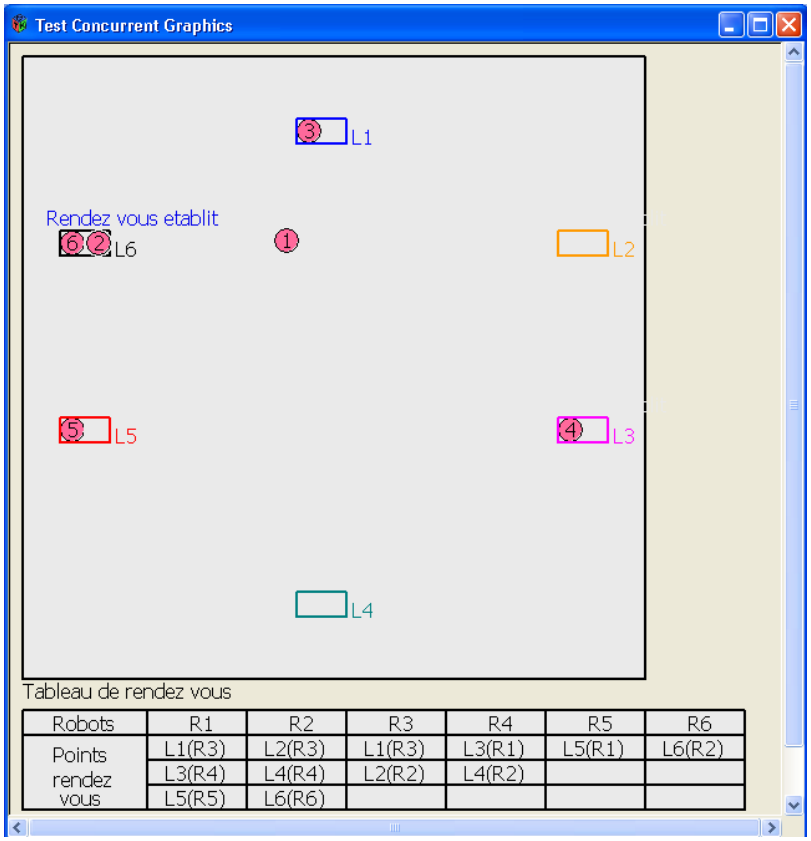


Figure 6.17 : Dernier rendez-vous établit entre les deux robots R2 et R6 sur le point de rendez-vous L6.

2.1. Conclusion.

Le contrôle de l'application est réalisé par un seul gardien ; les problèmes de mises à jour du contexte global de l'application sont donc facilement résolus puisque l'application s'exécute sur un seul processeur.

Pour améliorer la fiabilité du système il faut associer à chaque robot en attente de son homologue sur un point de rendez-vous un timer. Ceci permettra d'éviter une attente illimitée d'un homologue tombé en panne entraînant obligatoirement le blocage du système.

	Introduction générale.
Chapitre 1 :	Etat de l'art sur la robotique mobile et sur la coopération entre robots.
Chapitre 2 :	Planification de trajectoire et navigation.
Chapitre 3 :	Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes.
Chapitre 4 :	Implémentation des processus parallèles à l'aide du langage de programmation ADA.
Chapitre 5 :	Conception d'une plateforme de robots mobiles autonomes et début de réalisation.
Chapitre 6 :	Simulation.
	Conclusion générale.
	Bibliographie.
	Annexe et Appendice.

Conclusion générale et perspectives.

Les objectifs espérés de ce travail sont de mettre sur pied au sein du laboratoire *d'Automatique et de Signal Annaba (LASA)* une plateforme comportant un groupe de robots mobiles autonomes et coopératifs. C'est un problème complexe, il intègre :

1/ Tous les aspects de la robotique mobile autonome traitant de la planification de trajectoire, de navigation, d'évitement d'obstacle, etc…
2/ Les problèmes liés à la coopération entre robots mobiles traitant de l'ordonnancement des déplacements et des rendez-vous des robots
3/ Tous les problèmes de réalisation pratique de tous les modules allant de la conception des robots étagés avec toutes l'électronique et l'intélligence embarqués jusqu'au système de balises pour la localisation et la communication.

Pour cela nous nous sommes appuyés sur deux outils :

- Les réseaux de Pétri pour la modélisation de l'ordonnancement des déplacements et des points de rendez-vous
- Et le langage ADA pour l'implémentation des machines à états communicantes obtenues par la décomposition des réseaux de Pétri en processus parallèle

Les résultats obtenus en simulation sont encourageants et le travail de la plateforme mérite d'être poursuivi dans deux directions importantes :

- En étoffant la plateforme avec d'autres outils et d'autres robots
- En modélisant le réordonnancement des déplacements et des rendez-vous en cas de panne d'un robot de manière temporaire ou permanente.

La première direction concerne l'intégration dans la plateforme de nouveaux robots de même type et son renforcement par le développement de système de localisation et de perception performant permettant la fusion de données issues d'un système de vision, des codeurs optiques mesurant les déplacements et le système de balise radar déjà existant. Cette première direction permet de renforcer les capacités individuelles de chaque robot.

La deuxième direction concerne les possibilités coopératives de l'ensemble des robots au sein de la plate forme. Elle consiste à améliorer et à développer les systèmes de communication et le mécanisme de coopération au sein du système multi-robots en modélisant le réordonnancement des déplacements en cas de panne temporaire ou définitive d'un robot.

C'est un projet ambitieux, il est entamé et il mérite d'être poursuivi.

	Introduction générale.
Chapitre 1 :	Etat de l'art sur la robotique mobile et sur la coopération entre robots.
Chapitre 2 :	Planification de trajectoire et navigation.
Chapitre 3 :	Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes.
Chapitre 4 :	Implémentation des processus parallèles à l'aide du langage de programmation ADA.
Chapitre 5 :	Conception d'une plateforme de robots mobiles autonomes et début de réalisation.
Chapitre 6 :	Simulation.
	Conclusion générale.
	Bibliographie.
	Annexe et Appendice.

Référence bibliographique.

[ADA 83] Ada Join Program Office, *"Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A"*, U.S. Department of Defense, The Pentagon, Washington, January 1983.

[ADA 95] Ada95 Référence Manual

[BAR 91] Jérîme Barraquand , Jean-Claude Latombe, Robot motion planning: a distributed representation approach, International Journal of Robotics Research, v.10 n.6, p.628-649, Dec. 1991

[BAY 06] Bernard BAYLE, Robotique mobile, Ecole Nationale Supérieure de Physique de Strasbourg 2006

[BER 01] S. Bergbreiter and K. Pister. Cotsbots. http ://www-bsac.eecs.berkeley.edu/ sbergbre/CotsBots/cotsbots.html, 2001.

[BRA 82] G.W. BRAMS, *"Réseaux de Petri : Théorie et Pratique" Tomes I et II*, MASSON, Paris, 1982.

[CRI 02] Fred G. Martin. Crickets, http://lcs.www.media.mit.edu/people/fredm/projects/cricket/, 2002

[FILL 04] David FILLIAT. Cours robotique mobile. Cours C10-2. ENSTA. 2004

[GIR 04] Vincent Girard-Reydet http ://www.winchgr.freesurf.fr/tipe/index.html, 2004

[GON 95] Graphes et Algorithmes. 1995, Gondran, Hinoux. Collection de la direction des études et recherches Electricité de France 37

[HAC 74] M. Hack, *"Extended State-Machine Allocatable Nets (ESMA), an extension of free Choice Petri Net results"*, MIT, project MAC, Computation Structures Group, Memo 78-1, 1974.

[HAD 88] S. Haddad, "Les réseaux de Petri", Cours de D.E.A. de l'Université Pierre et Marie Curie, Paris, 1988.

[HAD 91] S. Haddad, *"Méthodes de vérification des systèmes parallèles"*, Rapport d'habilitation à diriger des recherches, Université P. & M. Curie, 4 place Jussieu, 75252 Paris Cedex 05, Décembre 1991.

[HAD 03] Joyce El Haddad, Algorithmes de communication autostabilisants dans un système de robots mobiles, 2003

[HAL 04] Halioui Karim, Contrôle Commande des Nacelles pointées à base de technologies standard, Centre National d'Etudes Spatiales, 2004

[HOU 05] K.Houda ; R.Lakel ; D.Messaoud ; « Méthode globale de planification de trajectoire pour un robot mobile autonome basée sur les champs de potentiels fictifs » ; Conférence Internationale sur la Productique ; CIP 2005 ; 22-24 Novembre 2005 ; pages 120-128 ; 2005

[HOU 06] K.Houda ; R.Lakel ; D.Messaoud ; « Décomposition d'un réseau de pétri en objets communicants et implémentation sur langage Ada » ; International Conference on Control, Modelling and Diagnostics ICCMD'06 ; 24-26 Avril 2006 ; N° 026_HOUDA_1 ; 2006

[KHA 86] O. Khatib. *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots*. International Journal of Robotic Research, vol. 5, no. 1, pages 90–98, 1986.

[KHA 97] M. Khatib, B. Bouilly, T. Sim´eon et R. Chatila. *Indoor Navigation with Uncertainty using Sensor-Based Motions*. In IEEE International Conference on Robotics and Automation, pages 3379–3384, Albuquerque, Etats-Unis, avril 1997.

[KHE 03] K-TEAM SA. Khepera.
http ://www.k- team.com/robots/khepera/index.html, 2003

[KHO 99] P. K. Khosla and C. J. J. Paredis. Millibots.
http ://www-2.cs.cmu.edu/ cyberscout/ millibots.html, 1999.

[KOR 92] F. KORDON. Prototypage de systèmes parallèles à partir de réseaux de Petri colorés. Application au langage ADA dans un environnement centralisé ou réparti, 1992

[KRA 85] S. Krakowiak, *"Principes des systèmes d'exploitation des ordinateurs"*, Dunod-informatique, Paris, 1985.

[KRO 86] B.H. Krogh, C.E. Thorpe, Integrated path planning and dynamic steering control for autonomous vehicles, IEEE International Conference on Robotics and Automation, San Francisco, USA, 1986, pp. 1664-1669.

[LAT 99] J.-C. Latombe. *Motion Planning : A Journey of Robots, Molecules, Digital Actors, and Other Artifacts*. The International Journal of robotics Research, vol. 18, no. 11, pages 1119–1128, 1999.

[LAV 99] LaValle S. M., Hinrichnsen J.,"Visibility-Based Pursuit-Evasion : The Case of Curved Environments ", *Proc.IEEE International Confrence on Robotics and Automation, 1999.*

[LEG 97] LEGO Company. Lego mindstorms o_cial page.
http ://mindstorms.lego.com, 1997.

[LEM 03] Mathieu LEMAY, ASSIGNATION DYNAMIQUE DE POSITIONS POURLES DEPLACEMENTS EN FORMATION DE GROUPE DE ROBOTS, Université de SHERBROOKE, Canda, 2003

[PRO 95] J.-M. PROTH, Les réseaux de Petri pour la conception et la gestion des systèmes de production, 1995

[RAÎ 04] Clément RAÎEVSKY, émotions artificielles pour la structuration sociale d'un groupe de robots, Université de SHERBROOKE, Canda, 2004

[ROS 05] J-P. Rosen , http://www.adalog.fr/biblio1.htm#LivreJPR, 2005

[SCH 83] J. Schawrtz and M. Sharir, “On the ‘piano movers’ problem: general techniques for computing topological properties of real algebraic manifolds,” *Advances in Applied Mathematics*, 1983.

[TRV 89] N. Trèves, *"Comparative study of different techniques for semiflow computation in P/T Nets"*, Lecture Notes in Computer Science, Advances in Petri nets Nº424, 1989.

[Udu77] S. Udupa : Coilision detection and avoidance in computer controlled manipulators. In Proc. of the Int. Joint Conf. On Artificial Intelligence, Cambridge, MA (US), aug 1977.

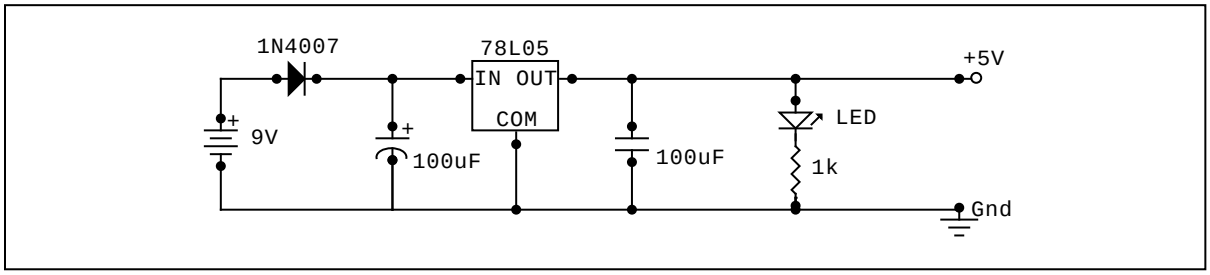
[ZAF 99] Luigi ZAFFALON, Programmation concurrentes et temps réel avec ADA9 95, Presses Polytechniques et universitaires romandes, 1999

Introduction générale.	
Chapitre 1 :	Etat de l'art sur la robotique mobile et sur la coopération entre robots.
Chapitre 2 :	Planification de trajectoire et navigation.
Chapitre 3 :	Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes.
Chapitre 4 :	Implémentation des processus parallèles à l'aide du langage de programmation ADA
Chapitre 5 :	Conception d'une plateforme de robots mobiles autonomes et début de réalisation.
Chapitre 6 :	Simulation.
Conclusion générale.	
Bibliographie.	
Annexe & Appendices.	

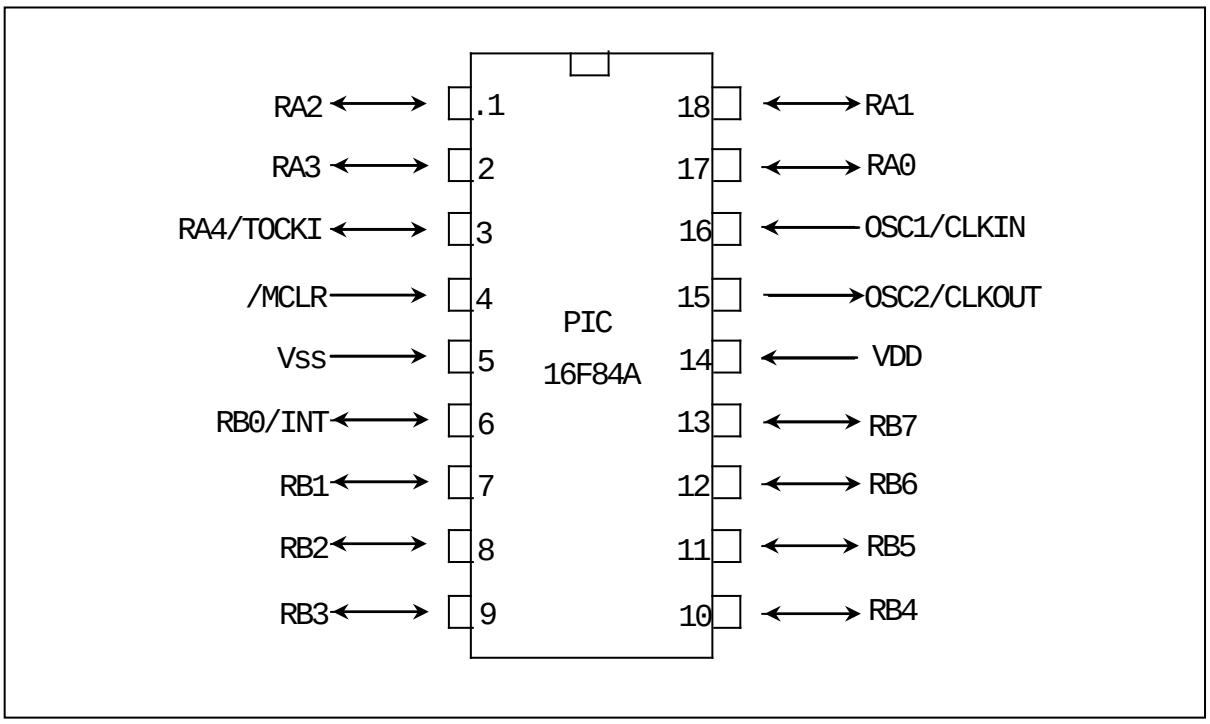
Annexe.

Schémas électroniques.

Alimentation stabilisée 5 Volts.



Brochage du PIC 16F84A.



Programmation du PIC 16F84A.

Pour programmer le PIC 16F84A il faut au minimum un kit de programmation, ce dernier est constitué de :

- Logiciel pour écrire le programme en mnémonique.

Chaque processeur ou microcontrôleur à son propre jeu d'instruction, le PIC 16F84A a une table à 37 instructions lui permettant d'établir un grand nombre d'applications. Le langage de programmation est l'assembleur (ou langage mnémonique).

Une fois le programme écrit, on passe à la phase d'assemblage

- Logiciel assembleur.

C'est un logiciel qui transforme les mnémoniques (fichier .ASM) d'un programme en langage compréhensible pour le microcontrôleur (fichier .hex) qui est un code hexadécimal destiné au PIC.

L'assembleur qu'on a utilisé est le logiciel « MPASM »,

Une fois le fichier assemblé (fichier .hex), on utilise un logiciel de programmation pour charger ce dernier dans la mémoire du PIC.

- Logiciel de programmation.

Il a pour rôle de stocker le contenu du fichier hexadécimal dans la mémoire flash du PIC via le port série du PC et une interface de programmation.

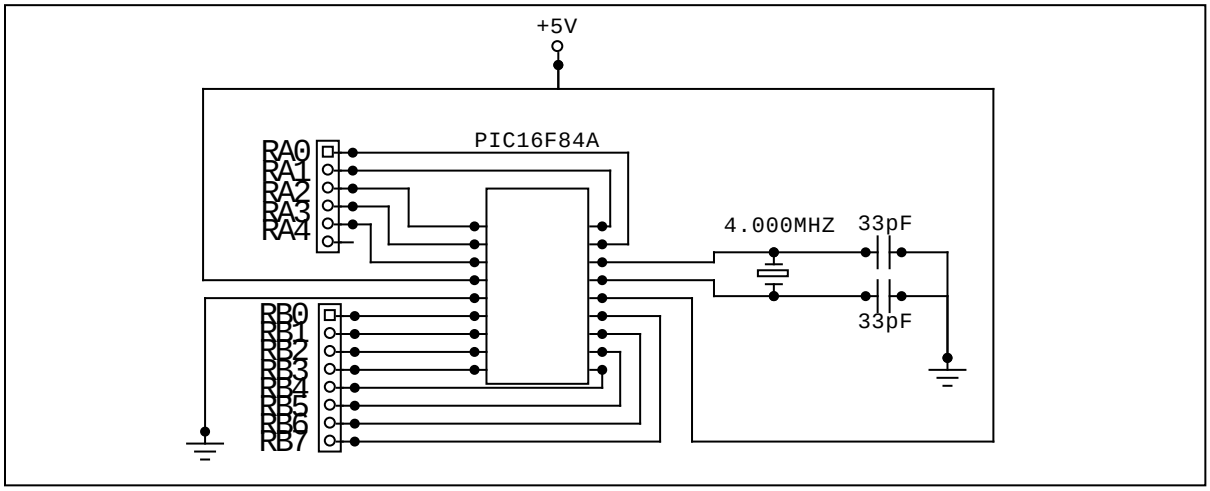
Le logiciel utilisé est le « ICPROG », il offre une grande souplesse d'utilisation, permet de charger un grand nombre de composants programmable (Eproms, microcontrôleurs, cartes à puces, …etc) et il travail sous windows XP ; il est téléchargeable gratuitement sur le site :

http://kudelsko.free.fr/prog_pic/PIC_XP.htm

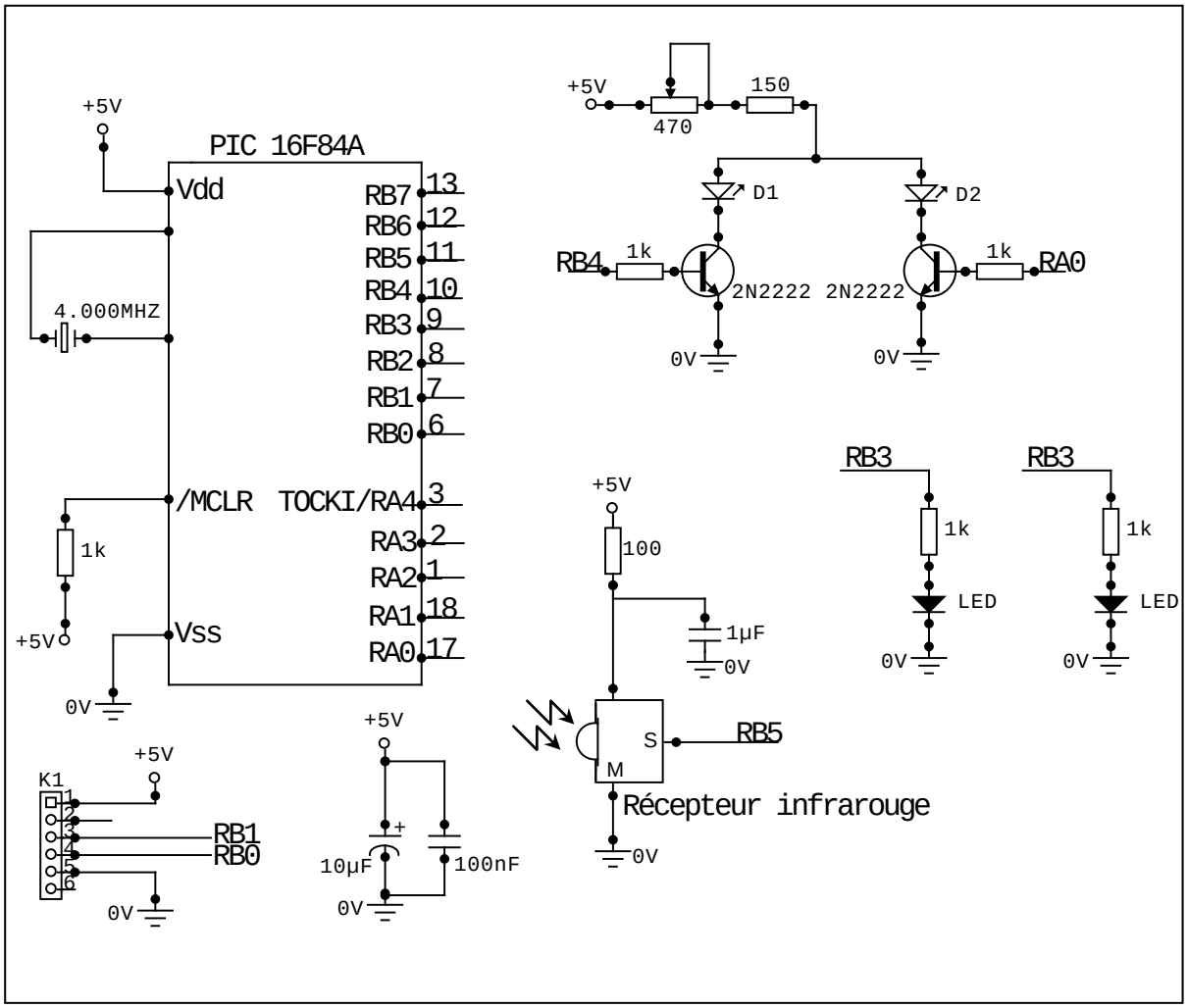
- Interface de programmation.

C'est un montage électronique, qui permet de charger ou de lire le contenu de la mémoire flash du PIC.

Carte cerveau du robot.



Carte radar infrarouge.



Carte de puissance pour les moto-réducteur.

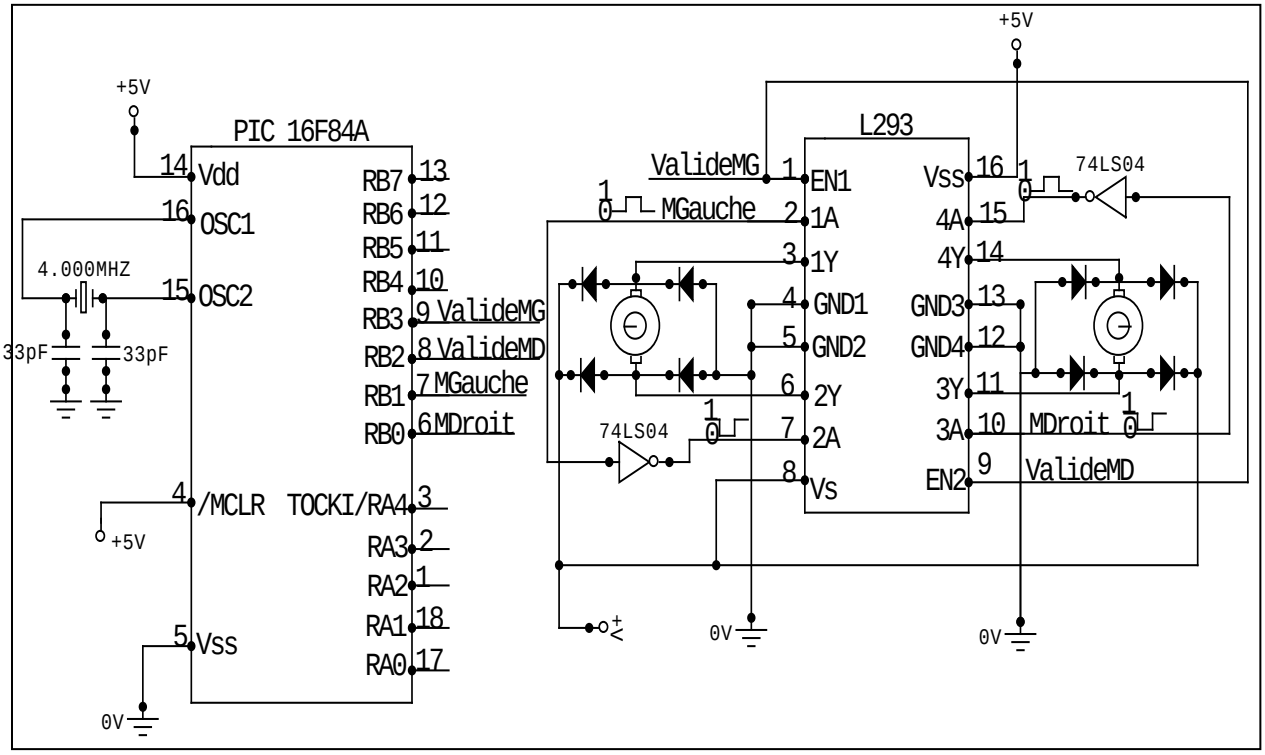
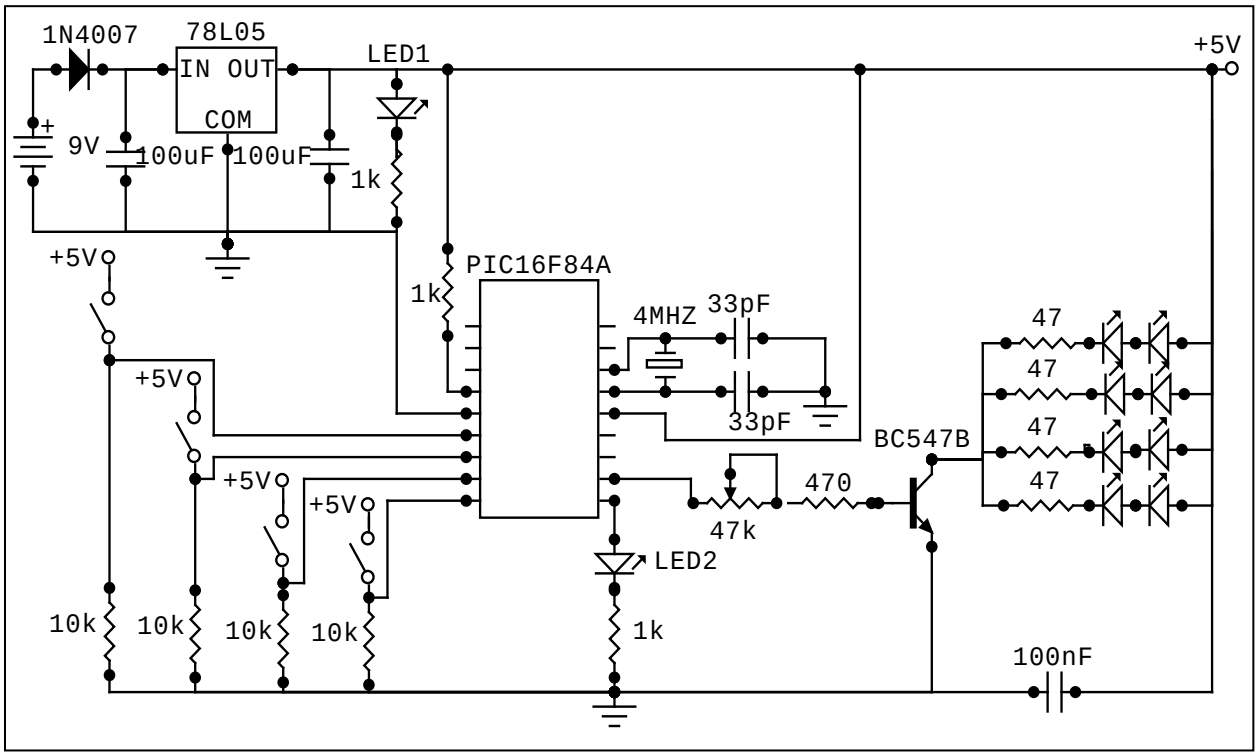


Table de vérité pour le fonctionnement des deux moteurs à courant continu.

Moteur Gauche		Mauteur Droit		Mode de fonctionnement
VALID_MG	MGAUCHE	VALID_MD	MDROIT	
1	1	1	1	Marche Avant
1	0	1	0	Marche Arrière
1	1	0		Marche Avant_Droit
0		1	1	Marche Avant_Gauche
1	0	0		Marche Arrière_Droit
0		1	0	Marche Arrière_ Gauche
0		0		Arrêt

Carte balise infrarouge.



Principe de fonctionnement d'un moteur Pas à Pas.

Il est constitué d'un rotor aimanté avec deux pôles, Nord et Sud, ainsi que d'un double - stator : à chacune de ces deux parties, est associé un bobinage avec un point milieu et deux phases ; en alimentant l'une ou l'autre des phases, on peut ainsi inverser l'aimantation au niveau du stator correspondant. La figure a représente le schéma d'un moteur pas-à-pas unipolaire à aimant permanent.

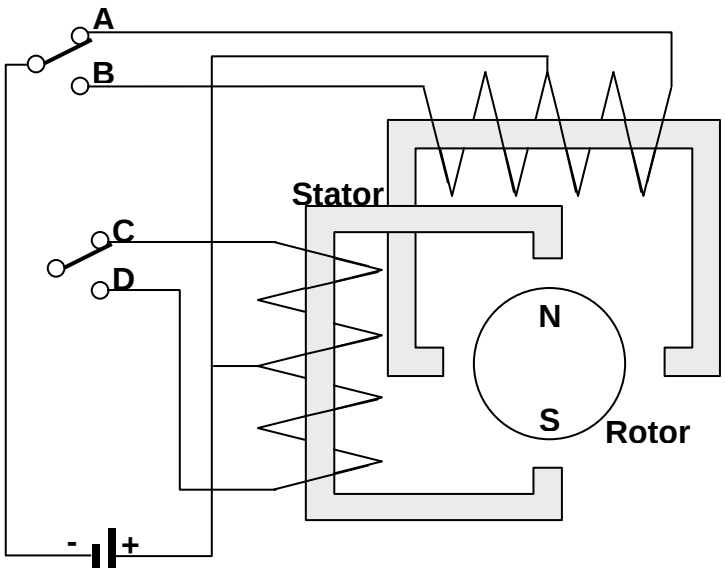


Figure a : Moteur unipolaire.

Sur cette figure, le moteur représenté possède un stator à deux enroulements à point milieu et un rotor constitué d'un aimant.

Les commutateurs permettent quatre combinaisons d'alimentation des enroulements. Ces différentes combinaisons et les champs magnétiques résultants vont déterminer la position du rotor comme indiqué sur la figure b. On remarque que ces combinaisons doivent être effectuées dans un ordre précis schématisé figure c, afin que le mouvement du rotor s'effectue uniformément (le sens de rotation est fonction du sens dans lequel on exécute cet ordre : 1234 ou 4321).

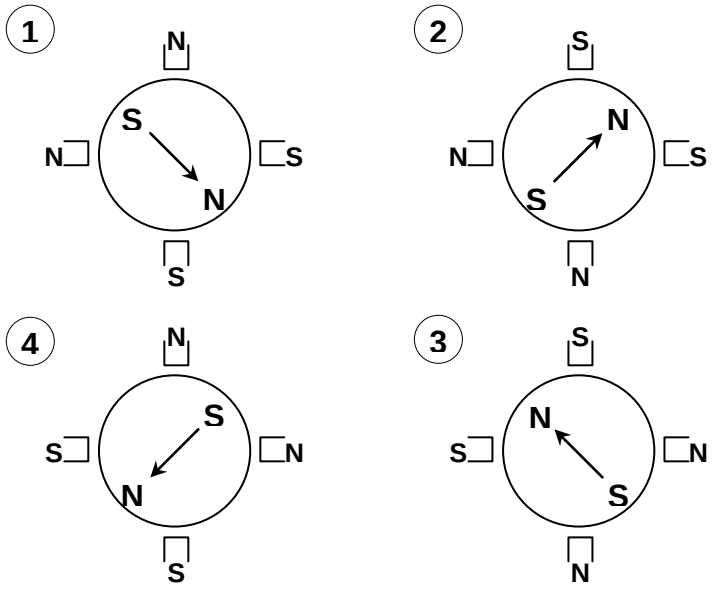


Figure b : Les quatre phases et positions d'équilibre du rotor.

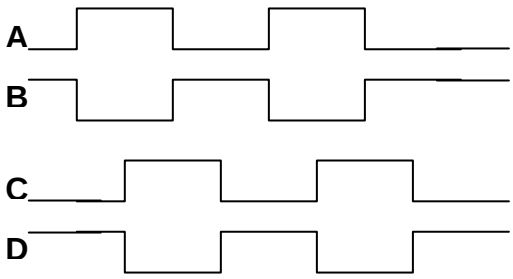
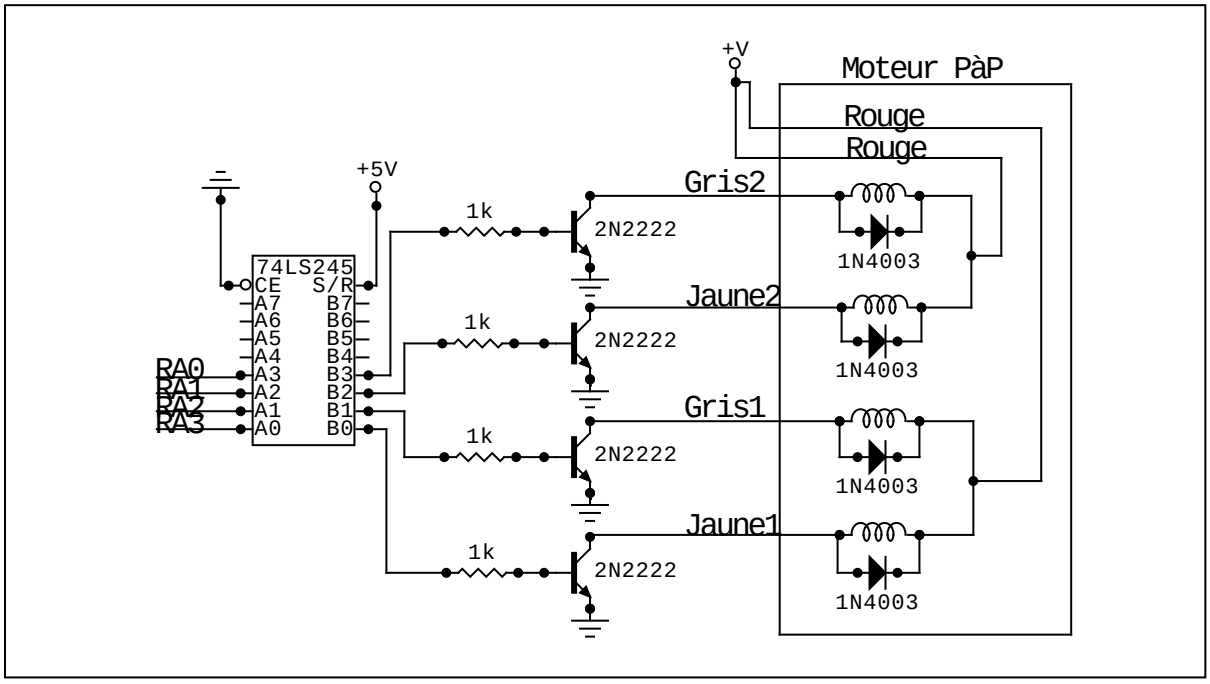


Figure c : Ordre d'application des tensions.

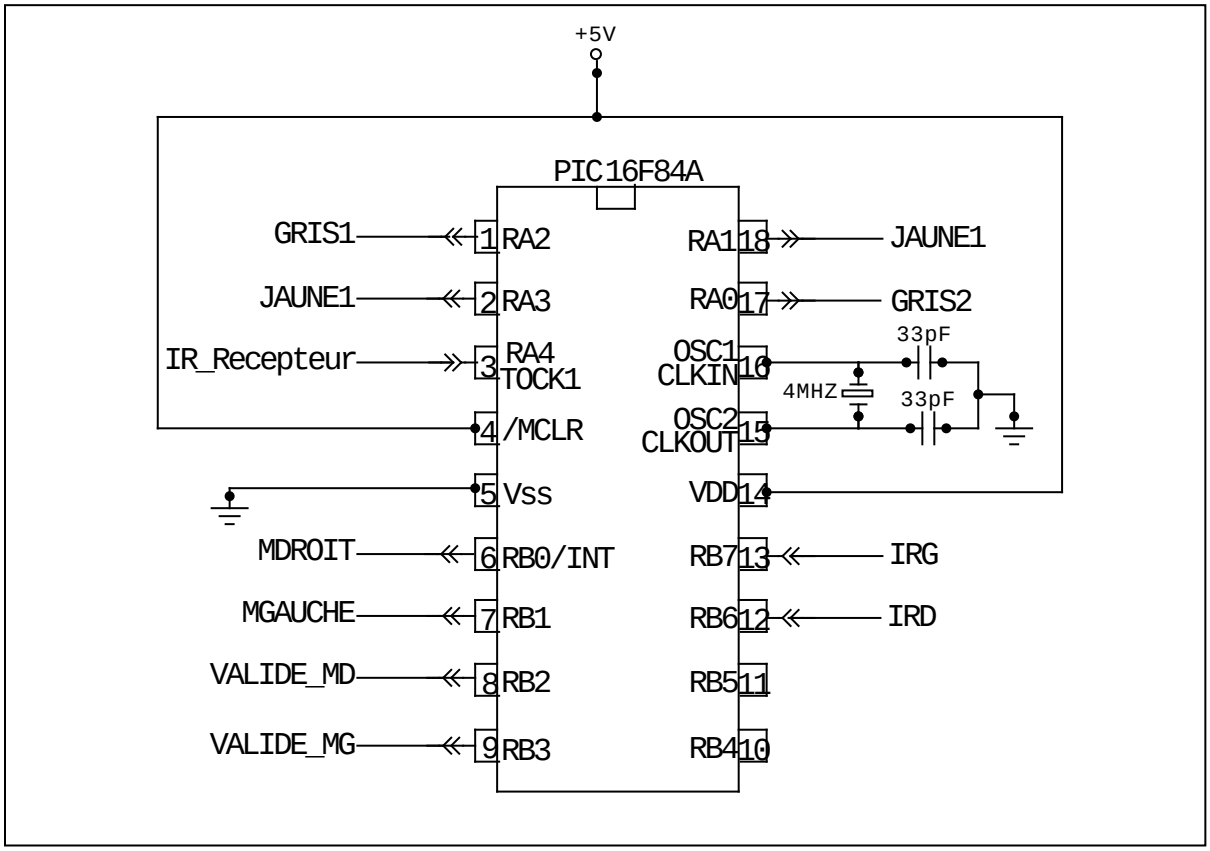
Sur ce moteur, le rotor ne peut prendre que quatre positions. Pour augmenter cette valeur, la solution générale consiste à augmenter le nombre de commutateurs et donc d'enroulements, et à augmenter le nombre de pôles N-S sur le rotor.

Carte puissance pour le moteur Pas à Pas.



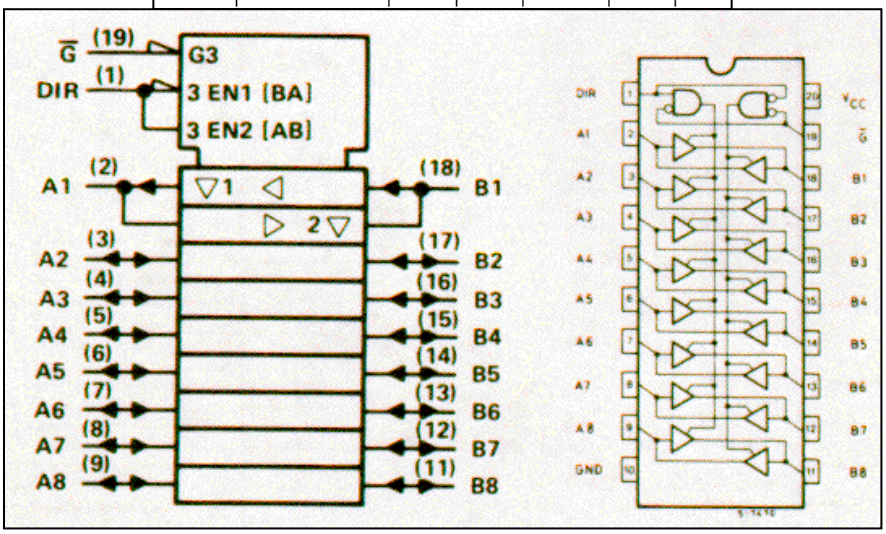
‘

Liens entre différents modules.



Brochage du 47Ls245.

	courant d'alim. (mA)	t_{PLH} (ns)	t_{PHL} (ns)	t_{enable} (ns)	sortance 0	sortance 1
LS245	64	8	8	27	30	60

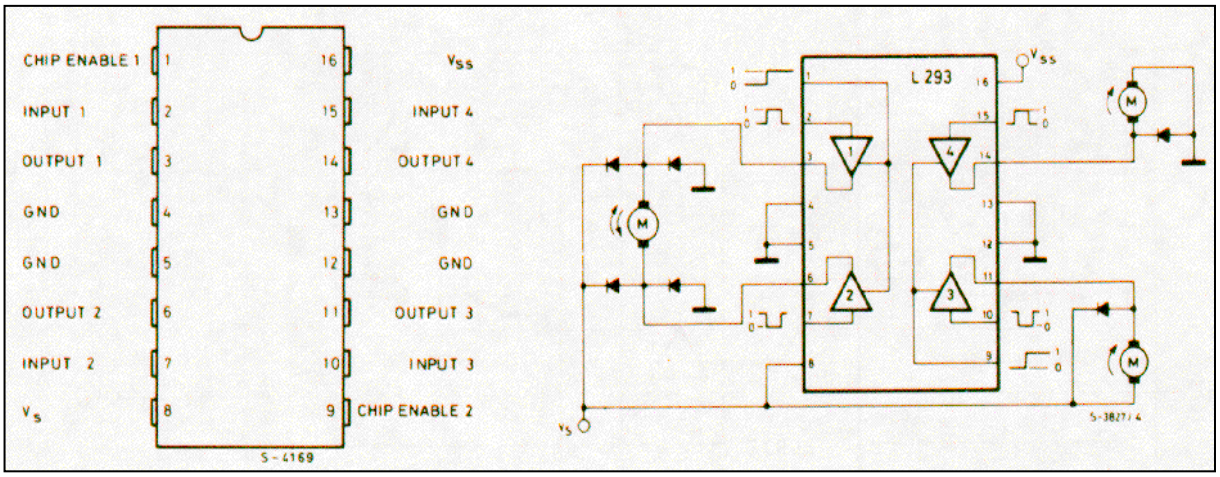


Description.

Ces circuits, composés de huit émetteurs-récepteurs, sont destinés aux communications asynchrones bidirectionnelles entre bus de données. La structure des fonctions de commande simplifie la logique extérieure nécessaire.

Le niveau logique appliqué sur l'entrée de commande de direction (DIR) détermine le sens de transmission des données. Un niveau logique bas sur cette entrée permet la transmission des données du bus B vers le bus A et un niveau logique haut permet la transmission du bus A vers le bus B. L'entrée de validation (G) sert à déconnecter le boîtier de telle manière que les deux bus soient isolés.

Brochage du L293.



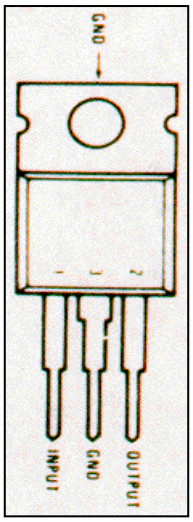
Caractéristiques.

Tension d'alimentation :	4,5 V à 36 V.
Consommation :	16 mA.
Puissance crête de sortie :	5 W.
Courant de sortie :	1A (600 mA pour L293D).
Entrées :	compatibles TTL.

Description.

Les L293, L293D et L293E sont des circuits intégrés contenant chacun quatre tampons dont chacun est en mesure de fournir un courant d'intensité nominale 1 A. Chaque tampon peut être commandé par une logique TTL puisque toutes les entrées logiques sont compatibles TTL. Il est possible de commander les tampons deux à deux par l'intermédiaire de deux entrées supplémentaires. La logique est alimentée indépendamment par une broche particulière du composant, elle peut donc fonctionner à un autre niveau de tension que la charge. C'est une façon de réduire notablement la puissance consommée. Le L293E est une version plus riche puisque chacun de ses tampons est pourvu d'une entrée de mesure particulière. Il est ainsi possible de contrôler l'intensité du courant à travers la charge. C'est surtout intéressant quand la charge doit être commandée par une tension découpée. Les sorties du L293D, à la différence de celles du L293, sont pourvues de diodes de roue libre de protection. Cette version sera donc utilisée pour des charges inductives. Toutes les variantes du L293 sont logées dans un boîtier DIL dont les quatre broches médianes peuvent servir à transmettre la chaleur produite à un radiateur.

Brochage du LM7805.



Description.

La série LM78MXX est disponible en boîtier plastique TO-202 qui permet au régulateur de fournir plus de 0,5 A avec un refroidisseur adéquat. La limitation en courant permet au courant de pointe de ne pas dépasser les limites de sécurité.

Une plage de sécurité est également prévue pour le transistor de sortie, ce qui permet de limiter la puissance interne dissipée. Si celle-ci devient trop importante pour le refroidisseur utilisé, le circuit de disjonction thermique est activé pour éviter une surchauffe du circuit intégré.

Des efforts considérables ont été faits pour rendre les LM78MXX faciles à mettre en oeuvre et pour réduire au maximum le nombre de composants extérieurs nécessaire. Il n'est pas obligatoire de découpler la sortie, bien que ceci améliore la réponse aux transitoires. Le découplage de l'entrée n'est nécessaire que lorsque le régulateur est éloigné du condensateur de filtrage de l'alimentation.

Photos du robot prototype.

Introduction générale	
Chapitre 1 :	Etat de l'art sur la robotique mobile et sur la coopération entre robots
Chapitre 2 :	Planification de trajectoire et navigation
Chapitre 3 :	Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes
Chapitre 4 :	Implémentation des processus parallèles à l'aide du langage de programmation ADA
Chapitre 5 :	Conception d'une plateforme de robots mobiles autonomes et début de réalisation
Chapitre 6 :	Simulation
Conclusion générale	
Bibliographie	
Annexe & Appendices	

Appendice A.

Index des figures.

Chapitre 1.

Chapitre 2.

Chapitre 5.

Chapitre 6.

Introduction générale	
Chapitre 1 :	Etat de l'art sur la robotique mobile et sur la coopération entre robots
Chapitre 2 :	Planification de trajectoire et navigation
Chapitre 3 :	Ordonnancements des déplacements des robots et leur décomposition en machines à états communicantes
Chapitre 4 :	Implémentation des processus parallèles à l'aide du langage de programmation ADA
Chapitre 5 :	Conception d'une plateforme de robots mobiles autonomes et début de réalisation
Chapitre 6 :	Simulation
Conclusion générale	
Bibliographie	
Annexe & Appendices	

Appendice B.

Index des Définitions.

Chapitre 3.