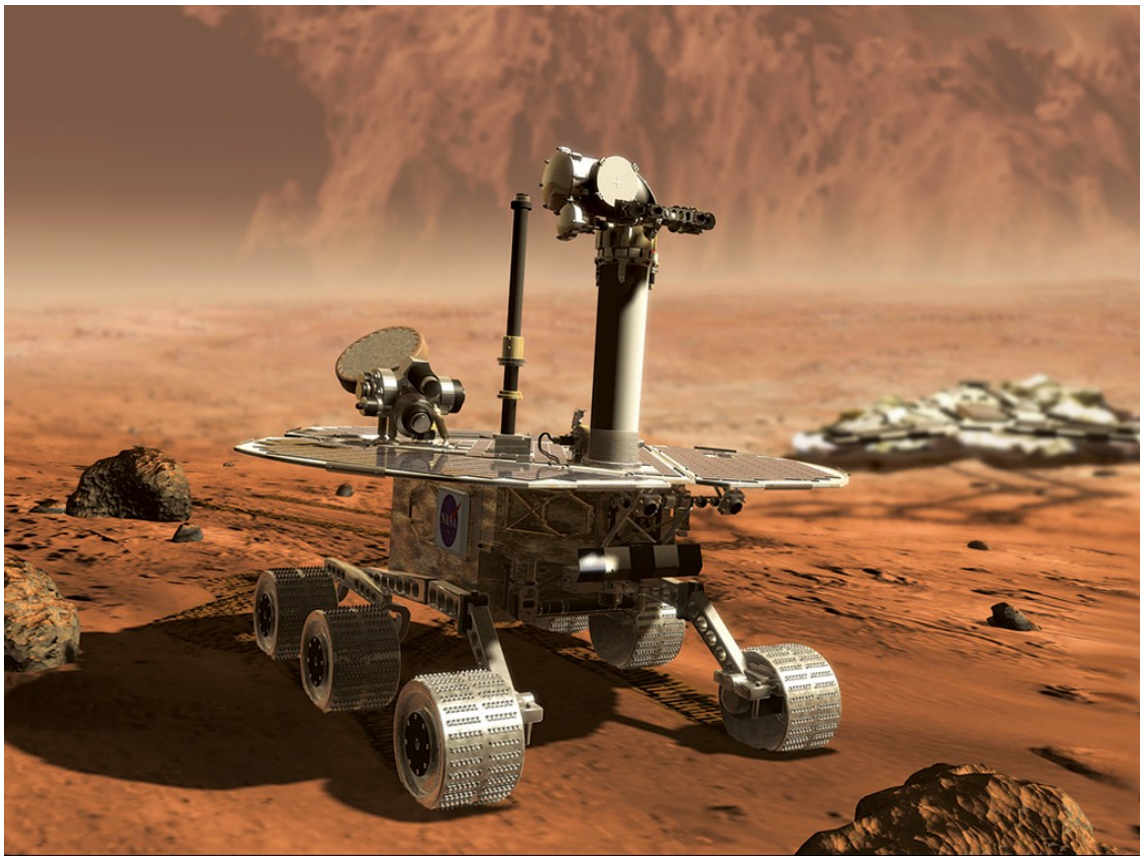
Laboratoire Informatique de l'École Polytechnique
Rapport de stage

Maître de stage : James REGIS

Contrôle d'un rover sur Mars - ICFP

Alexandre GORDIEN



Palaiseau, le 13 septembre 2009

Remerciements

Je remercie Monsieur James REGIS, informaticien et maître de stage, pour son enseignement, ses conseils et pour l'ensemble du stage.

Je remercie Monsieur Matthieu Guionnet, informaticien, pour le temps qu'il a passé à m'enseigner les bases du langage Python et pour tous ses conseils.

Je remercie Monsieur Marc Giusti, directeur de recherche, pour son aide dans la résolution d'un problème mathématique.

Je remercie également Monsieur le Directeur du Lycée Teilhard de Chardin, Pierre PELLÉ, pour la confiance qu'il a bien voulu m'accorder en me délivrant les conventions de stage pour l'École Polytechnique chaque fois que cela s'est avéré nécessaire.

Table des matières

Introduction

Après deux stages au Laboratoire Informatique de École Polytechnique, l'un sur la vision par ordinateur pour donner de la "vue" à un robot et l'autre sur l'électronique et la mécanique pour le doter de mouvements, il me restait à découvrir l'intelligence artificielle pour créer un robot autonome complet.

Du 6 au 31 juillet, j'ai intégré l'équipe systèmes et réseaux du laboratoire et travaillé avec Monsieur James REGIS, informaticien, ingénieur systèmes et réseaux et président de l'association de robotique dROBes.

L'intelligence artificielle est la "recherche de moyens susceptibles de doter les systèmes informatiques de capacités intellectuelles comparables à celles des êtres humains". Le sujet de l'ICFP 2008, concours de programmation mondial organisé chaque année par une université différente, constituait une bonne approche de ce domaine. Il s'agissait d'automatiser le comportement d'un robot sur Mars pour qu'il rejoigne sa base malgré des obstacles comme des cratères, des rochers et des Martiens.

Chapitre 1

L'entreprise

Situé au coeur du Centre de Recherche sur le campus de l'École Polytechnique à Palaiseau, le Laboratoire d'Informatique de l'X (LIX) est une UMR X-CNRS d'une centaine de membres dont une moitié de doctorants et une quarantaine de permanents équitablement répartis entre le CNRS, l'INRIA et l'X. Les activités du LIX se regroupent en trois grands domaines : algorithmique, réseaux, et méthodes formelles.

Parmi les dix équipes présentes dans les locaux du laboratoire, on compte 6 projets INRIA installés au LIX depuis la création du Pôle Commun de Recherche en Informatique du Plateau de Saclay, et une équipe commune avec le CEA LIST qui préfigure les cohabitations futures entre des équipes d'origine différente issues du RTRA Digiteo. Enfin, le LIX abrite la Chaire « Systèmes Industriels Complexes « financée par Thalès. À tous ces titres, il est devenu un acteur académique essentiel au sein du pôle de compétitivité System@tic.

Le LIX s'intéresse tout particulièrement au domaine des communications en réseau, afin de se doter dans ce domaine de compétences globales relatives au traitement du signal, au chiffrement et au routage des communications, à la distribution et à la mobilité des calculs ainsi qu'à la sécurité et à l'ingénierie des protocoles.

Au coeur de ces recherches, dont le but est de mettre au point des systèmes de communications fiables et efficaces, l'accent est mis sur les réseaux mobiles, qui deviendront une composante incontournable des systèmes embarqués futurs.

Le laboratoire est fortement impliqué dans les enseignements de l'École Polytechnique ainsi que dans plusieurs MASTER de la région parisienne. Il a des relations contractuelles avec des organismes publics (Ministère de l'Enseignement Supérieur et de la Recherche, Direction Générale de l'Armement, Agence Française pour l'Innovation, INRIA...) ou des organisations internationales.

Au cours des dernières années, le LIX s'est attaché à développer de nombreuses collaborations avec le monde industriel. Outre Thalès, on peut citer parmi ses grands partenaires internationaux Microsoft Research, Hitachi Labs, et la NASA.

Chapitre 2

L'ICFP 2008

2.1 Présentation du sujet

Les avancées technologiques récentes dans le domaine de la télécommunication permettront bientôt aux données d'être transférées entre Mars et la Terre quasi-instantanément. Ainsi, la NASA[1] est à la recherche d'exemples de logiciels de contrôle en temps réel de ses robots sur Mars. Sachant que le concours ICFP attire les meilleurs programmeurs du monde, la NASA a décidé de l'utiliser comme moyen de rassembler des prototypes logiciels pour leur nouveau système de commande. L'équipe gagnante du concours verra son algorithme utilisé pour contrôler le prochain robot envoyé sur la planète rouge.

L'objectif est de réaliser un programme — le *contrôleur* — capable de gérer de manière autonome le comportement du rover sur un terrain inconnu et hostile. L'intelligence artificielle doit permettre au rover d'adapter sa route en fonction des obstacles qu'il peut rencontrer : rochers, cratères et Martiens. La mission du rover est de rejoindre sa base en un minimum de temps, sans dommage.

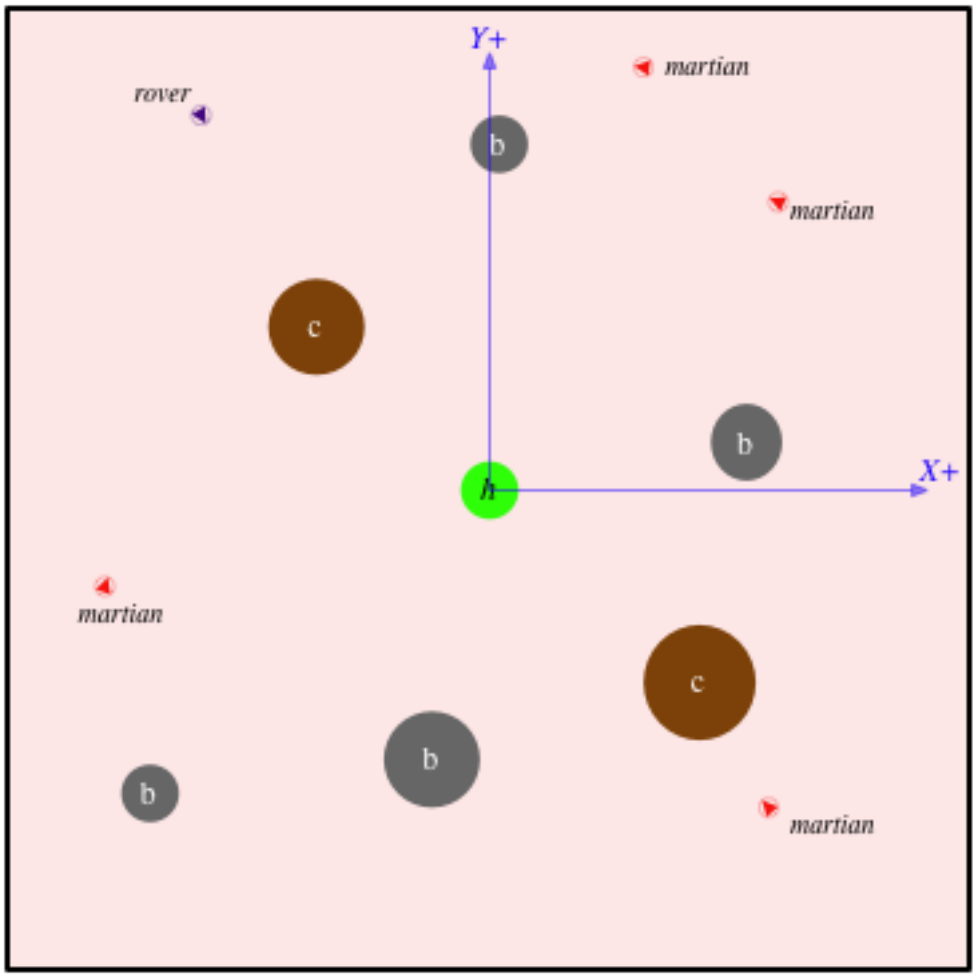
Les programmes sont évalués suivant leurs performances dans une série d'essais. Lors de l'évaluation, chaque exécution réalise cinq essais sur une carte aléatoire.

2.2 Le rover

Le rover est un véhicule de 0,5m de rayon que le contrôleur doit guider sur le terrain. Pour cela une connexion doit être établie entre le rover et le contrôleur sur une socket TCP/IP[2]. Cette socket est utilisée tout au long de l'exécution. Cependant, bien que la vitesse de communication soit très élevée, il y a un temps de connexion de l'ordre de 75 microsecondes.

[1] National Aeronautics and Space Administration, ("administration nationale de l'aeronautique et de l'espace"). Il s'agit de l'agence gouvernementale responsable du programme spatial des états-Unis.

[2] TCP/IP (Transmission Control Protocol / Internet Protocol) est le protocole le plus commun pour transmettre des informations autour d'un réseau. Chaque ordinateur d'un réseau TCP/IP doit avoir sa propre adresse IP.



Une fois la connexion établie, le contrôleur reçoit un premier message contenant les dimensions de la zone d'activité du rover et ses caractéristiques physiques. La NASA teste differents modèles de robots ayant des caractéristiques différentes. La zone est représentée suivant un repère orthogonal dont l'origine correspond au centre de cette zone. La *base* — destination du rover — est un cercle de 5m de rayon situé à l'origine. Plusieurs cartes sont disponibles.

Environ une seconde après le message d'initialisation, le premier essai démarre et le serveur commence à envoyer un flux de données télémétriques au contrôleur. Les données télémétriques comprennent la position, la vitesse et des informations sur le terrain. Dès la réception des données, le contrôleur peut envoyer des commandes au serveur pour diriger le véhicule vers la base.

Le rover doit faire face à trois types d'obstacle sur son chemin : - s'il entre en contact avec un rocher ou le bord de la carte, il rebondit et perd de la vitesse. La collision a lieu si la distance entre le centre du rocher et celui du rover est inférieure à la somme de leurs rayons ; - si le centre du rover entre dans un cratère, il tombe et explose ; - si le vehicule est bloqué par un Martien, il est détruit.

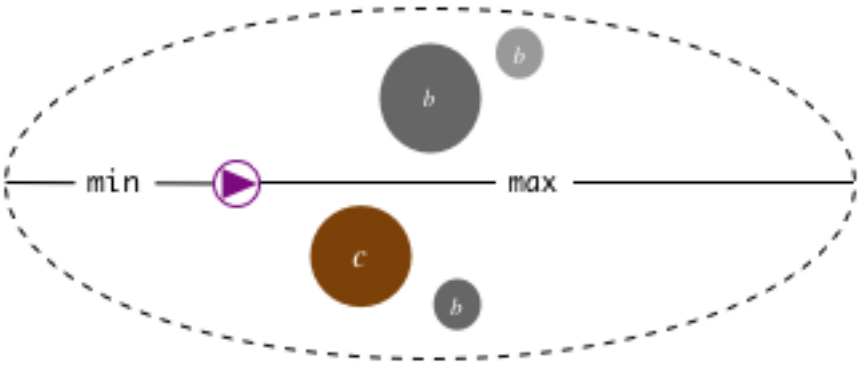
Les deux derniers cas entrainent la fin de l'essai.

Les Martiens, bien qu'hostiles, ne possèdent pas de caractéristiques physiques particulières : ils ne traversent pas les cratères, ne passent pas au travers des rochers et ne peuvent pas dépasser les

limites de la carte. Leurs caratériques physiques sont les mêmes que le rover si ce n'est qu'ils peuvent être plus rapides ou plus lents. Les Martiens sont légèrement plus petits : leur rayon est de 0.4m.

2.2.1 La vision

Les capteurs visuels du rover lui offrent un champs de vision de forme elliptique, étendue dans le sens de la marche du rover.



Cette illustration présente le rover orienté vers la droite. L'ellipse est définie par *min* et *max*. Le rover peut tout voir dans cette ellipse à l'exception des éléments cachés par les rochers.

2.2.2 La vitesse

La vitesse linéaire du rover au temps $t'(s'_t)$ est calculée en fonction de sa vitesse au temps t précédent selon la formule suivante :

$$t'_s = max(s_t + (t' - t)a - k(t' - t)s_t^2, 0)$$

Le dernier paramètre simule les forces de frottement s'exercant sur le véhicule. Remarquez que a peut être négatif si le robot freine. Les taux d'accélération et de freinage ne sont pas connus mais peuvent être déterminés par des tests. L'effet de frottement a pour but de réduire la vitesse maximale du rover. La vitesse maximale théorique (sans forces de frottements) est connue : elle est envoyée par le serveur dans le message d'initialisation. Mais ce n'est pas le cas du coefficient de la force de frottement.

2.2.3 La direction

Le rover dispose de deux vitesse de rotation, une moyenne et une rapide, dans les deux directions. Lorsque le rover reçoit une commande pour tourner à gauche, son état change d'un cran vers la gauche, de même pour la droite. L'accélération en rotation des véhicules est limitée et, par conséquent, il faudra un certain temps pour passer d'un mode de virage à l'autre.

2.3 Protocole de communication

La communication entre le serveur et le contrôleur se fera sur une socket TCP/IP avec des chaînes de caractères ASCII[3]. L'IP du serveur et le port de communication sont envoyés en ligne de commande au contrôleur. Le contrôleur, client TCP/IP, doit ensuite établir la connexion avec le serveur pour pouvoir recevoir les données télémétriques.

Un *message* est une séquence de blocs de caractères délimités par le caractère ASCII "espace" (0x20) et terminée par une virgule. Il renvoit diverses informations :

- les distances, longueurs et positions sont données en mètres dans un repère fixe et arrondies au millième ;
- les angles sont donnés en degrés et arrondis au dixième ;
- les vitesses angulaires sont données en degrés par seconde et arrondies au dixième ;
- les vitesses sont données en mètres par seconde et arrondies au millième ;
- les durées sont en millisecondes.

2.3.1 Messages du serveur vers le controleur

Il y a plusieurs types de messages, identifiés par le premier caractère de la chaîne.

Initialisation

Les caractéristiques exactes du rover ne sont pas precisées et peuvent varier suivant les essais mais des informations sont données au début de chaque exécution. Une fois la connexion au serveur établie, le contrôleur reçoit un message formaté comme suit :

I *dx dy time-limit min-sensor max-sensor max-speed max-turn max-hard-turn ;*

I est le caractere indiquant qu'il s'agit d'un message d'initialisation.

dx est la taille de la carte sur l'axe x (en mètres). Une carte avec $dx = 100.000$ s'étend de -50 à 50 sur l'axe des abscisses.

dy est la taille de la carte sur l'axe y (en mètres). Une carte avec $dy = 100.000$ s'étend de -50 à 50 sur l'axe des ordonnées.

time-limit est la durée maximale d'un essai (en millisecondes).

min-sensor est la plus petite longueur du champ de vision du rover (en mètres). Voir schéma ci dessus.

[3]ASCII, (American Standard Code for Information Interchange), est une norme de codage de caractères alphanumériques.

max-sensor est la plus grande longueur du champ de vision du rover (en mètres). Voir schéma ci dessus.

max-speed est vitesse maximale du véhicule (en mètres par seconde).

max-turn est la vitesse maximale en virage simple (degrés par seconde).

max-hard-turn est la vitesse maximale en virage serré (degrés par seconde).

Données télémétriques

Pendant un essai, le rover envoie regulièrement un flux de données télémétriques (environ toutes les 100 millisecondes). Ces données comprennent des informations sur l'état du véhicule (position, vitesse ...) ainsi que des informations sur son environnement (obstacles et ennemis).

T *time-stamp vehicle-ctl vehicle-x vehicle-y vehicle-dir vehicle-speed objects ;*

T est le caractère indiquant qu'il s'agit de données télémétriques.

time-stamp est le temps écoulé depuis le début de l'essai (en millisecondes).

vehicle-ctl est l'état actuel du rover. C'est un bloc de deux caractères : le premier indique s'il est en train d'accélérer (a), de freiner (b) ou s'il se déplace à vitesse constante ($-$) et le deuxième indique le sens et le type de virage (L pour un virage rapide à gauche, l pour un virage simple à gauche, ($-$) pour un déplacement rectiligne, r pour un virage simple à droite et R pour un virage rapide à droite).

vehicle-x est l'abscisse du rover.

vehicle-y est l'ordonnée du rover.

vehicle-dir est la direction du rover, donnée par la mesure de l'angle formé par son vecteur directeur et l'axe des abscisses, dans le sens trigonométrique.

vehicle-speed est la vitesse du rover (en mètres par seconde).

objects est une séquence contenant des informations sur la présence ou non d'obstacles et/ou de Martiens dans le champ de vision du vehicule. Est visible tout élément présent dans la zone blanche dont les caractéristiques sont données dans le message d'initialisation. Ces messages peuvent avoir deux formes différentes suivant le type de l'objet. Si l'objet est un rocher, un cratère ou la base, le format est le suivant :

object-kind object-x object-y object-r

object-kind est soit un b pour un rocher (boulder), c pour un cratère, ou h pour la base (home).

object-x est l'abscisse du centre de l'objet.

object-y est l'ordonnée du centre de l'objet.

object-r est le rayon de l'objet.

S'il s'agit d'un Martien, le message aura ce format :

m *enemy-x enemy-y enemy-dir enemy-speed*

Voici un exemple de message télémétrique :

```
T 3450 aL −234.040 811.100 47.5 8.450 b −220.000 750.000 12.000 m −240.000 812.000 90.0 9.100 ;
```

Ce message donne l'état du vehicule 3,45 seconde après le début de l'essai. Il est en train d'accélérer et d'effectuer un virage rapide à gauche. Il se situe au point de coordonnées $(-234,045; 811,100)$, sa direction est $47,5$ degrés (Nord-Est), sa vitesse est de $8,450$ mètres par seconde et il voit un rocher et un Martien.

Messages d'alerte

Il existe également des messages pour signaler des incidents. Ces messages ont le format suivant :

tag *time-stamp*

tag est l'un des caractères suivants :

B indique une collision avec un rocher ou le bord de la carte.

C indique que le rover est tombé dans un cratère, ce qui entraine la fin de l'essai.

K indique que le robot a été pris par un Martien, ce qui entraine également la fin de l'essai.

Message de reussite

Le serveur envoie le message "**S** ***t*** **;**" lorsque le véhicule atteint la base.

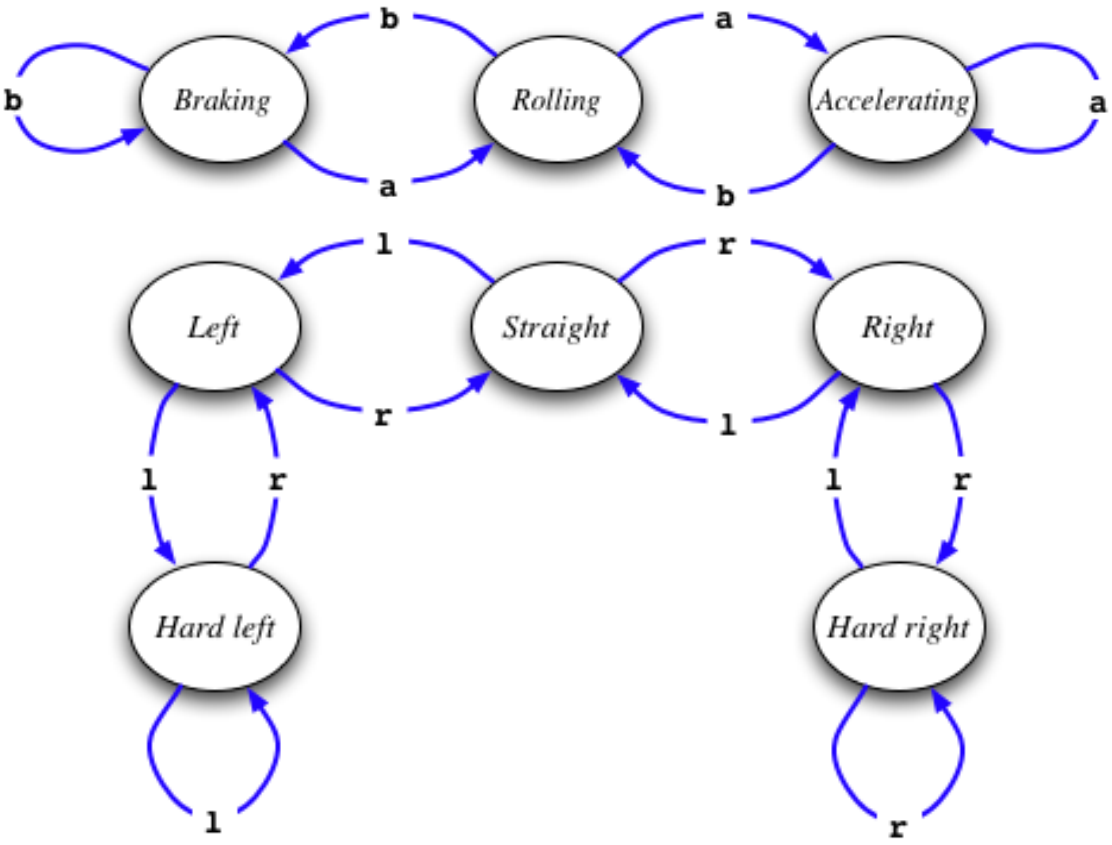
Message de fin

à la fin de chaque essai, le serveur envoie le message "**E** t s", ou t est le temps écoulé depuis le début de l'essai et s est le score (c'est à dire le temps d'exécution et les sanctions éventuelles).

Une fois que l'essai est terminé, il y a une pause d'une seconde avant le début du prochain essai. Le contrôleur ne doit donc pas s'arreter mais se préparer pour l'essai suivant. Notez que le message d'initialisation n'est envoyé qu'une fois par exécution. Lors d'une exécution, quelque soit le nombre d'essais, la même carte est utilisée même si la position initiale du rover et le nombre de Martiens peuvent varier.

2.3.2 Messages du controleur vers le serveur

Le comportement du rover est controlé par les messages envoyés par le contrôleur. Les schémas suivants résument l'interprétation de la commande envoyée :



Chaque commande est composée d'un caractère déterminant l'accélération (**a**) ou le freinage (**b**), suivi d'un autre indiquant le sens de rotation (**l** pour gauche et **r** pour droite) et est terminée par un point-virgule. Ainsi les messages pouvant être envoyés sont :

Messages : **; a ; b ; l ; r ; al ; ar ; bl ; br ;**

Aucun autre caractère ne doit être envoyé sur le flux de commande !

Bien que la communication avec le robot soit très rapide, il y a une certaine latence (moins de 20 millisecondes) dans le traitement des commandes. Le contrôleur peut envoyer des messages aussi souvent qu'il le veut mais la surchage du serveur peut avoir des effets négatifs sur les performances.

2.4 Règles et scores

L'évaluation se fait avec une exécution de cinq essais sur la même carte, la difficulté étant croissante. Chaque carte possède une limite de temps de n millisecondes.

Le score pour un essai est le temps qu'il a fallu au rover pour rejoindre la base ou être détruit ainsi que les sanctions.

- Si le rover arrive à la base avant la limite de temps ($t \leq n$), la période d'évaluation est t
- Si le rover ne parvient pas à atteindre la base dans la limite de temps, la période d'évaluation est donnée par l'équation $2n - t + p$ où t est le temps écoulé et p est la sanction :
  - 100 si le delai est dépassé
  - 600 si le rover a été detruit par un Martien
  - 1000 si le rover est tombé dans un cratère

Notez que t est au plus n dans cette formule car l'essai se clôture lorsque t est supérieur à n. Par conséquent $(2n - t)$ sera toujours compris entre n et $2n$ inclus.

Chapitre 3

Les outils

3.1 Le système d'exploitation : Linux

3.1.1 Qu'est-ce que Linux ?

Linux est un système d'exploitation de type UNIX, multi-tâches et multi-utilisateurs, compatible avec un très grand nombre de processeurs, ouvert sur les réseaux et les autres systèmes d'exploitation.
La principale singularité de Linux est d'être un logiciel libre, développé de façon collaborative et pour une grande part bénévole par des milliers de programmeurs répartis dans le monde.
Ce modèle de développement joue un grand rôle dans la qualité du résultat obtenu, qui est considéré par des analystes indépendants comme très supérieurs à des systèmes commerciaux similaires, par exemple Windows.

3.1.2 Avantages de Linux

Linux est un système :

- **Puissant**. Il permet de faire faire beaucoup de choses à sa machine.
- **Efficace**. Contrairement à des systèmes beaucoup plus répandus, il n'utilise pour ses besoins propres que très peu de ressources. Les logiciels que vous utilisez pour votre travail disposent donc de beaucoup plus de puissance pour fonctionner.
- **Fiable**. Une machine sous Linux peut fonctionner 24h/24 sans aucun problème à la condi-

tion d'avoir le matériel adapté, en particulier au niveau thermique.

- **Robuste**. Une erreur d'un utilisateur ou un "plantage" éventuel d'une application n'affecte pas le reste du système. D'autre part, il est exceptionnel de devoir l'arrêter : la quasi-totalité des opérations de configuration, mise au point, etc, ne nécessite pas l'arrêt du système.
- **Gratuit**. Linux est gratuit s'il est téléchargé via Internet (hors coût télécommunications) ou s'il est recopié depuis un CD-ROM prêté par un ami. Contrairement aux logiciels commerciaux, c'est légal et même recommandé.
- Enfin, Linux est conforme à la norme POSIX et aux standards du marché, en particulier de l'Internet. Cela signifie qu'un logiciel conçu pour un autre système de la même famille (Solaris de SUN, Digital Unix, AIX d'IBM, SCO Unix...) peut être rapidement porté sous Linux et vice-versa, ce qui assure une protection de l'investissement logiciel en cas d'obligation de changement de système.

Comme démontré, Linux est un système exceptionnel donnant satisfaction aussi bien sur des machines anciennes ou bas de gamme que sur des machines puissantes très sollicitées ou devant remplir des fonctions importantes.

3.1.3 Pourquoi Linux ?

Le serveur fourni par les organisateurs du concours et les outils du LIX fonctionnent sous Linux. J'aurais pu réaliser l'application client sous Windows mais Linux est un système d'exploitation à découvrir et qui fait de plus en plus parler de lui.

3.2 Le choix du langage de programmation

3.2.1 Le langage C

Les premières lignes de code du projet ont été rédigées en C car c'est un langage que j'utilise fréquement. Mais je me suis rapidement aperçu que la réalisation de ce projet en C serait longue et fastidieuse, notamment à cause de la gestion des types[1]. Ainsi, alors que le programme n'était capable que de se connecter, recevoir les données et *parser*[2] la chaîne de données, le code atteignait déjà les 300 lignes !

Nous avons donc pris la décision, au bout de la première semaine, de reprendre le projet depuis le début et d'utiliser un autre langage : Python.

[1] Les données manipulées en langage C sont typées, c'est-à-dire que pour chaque donnée que l'on utilise (dans les variables par exemple) il faut préciser le type de donnée, ce qui permet de connaître l'occupation mémoire (le nombre d'octets) de la donnée ainsi que sa représentation.

[2] Il s'agit de découper et distribuer dans des variables les informations contenues dans une chaîne de caractères.

3.2.2 Python

Python est un langage portable, dynamique, extensible, gratuit, qui permet (sans l'imposer) une approche modulaire et orientée objet de la programmation. Python est développé depuis 1989 par Guido van Rossum et de nombreux contributeurs bénévoles.
Quelles sont les caractéristiques du langage ?

- Python est portable, non seulement sur les différentes variantes d'Unix, mais aussi sur les OS propriétaires : MacOS, BeOS, NeXTStep, MS-DOS et les différentes variantes de Windows.
- Python est libre ; il peut être utilisé sans restriction dans des projets commerciaux.
- Python convient aussi bien à des scripts d'une dizaine de lignes qu'à des projets complexes de plusieurs dizaines de milliers de lignes.
- La syntaxe de Python est très simple et, combinée à des types de données évolués (listes, dictionnaires,...), conduit à des programmes à la fois très compacts et très lisibles. A fonctionnalités égales, un programme Python (abondamment commenté et présenté selon les canons standards) est souvent de 3 à 5 fois plus court qu'un programme C ou C++ (ou même Java) équivalent, ce qui représente en général un temps de développement de 5 à 10 fois plus court et une facilité de maintenance largement accrue.
- Python est orienté-objet. Il supporte l'héritage multiple et la surcharge des opérateurs. Dans son modèle objets, et en reprenant la terminologie de C++, toutes les méthodes sont virtuelles.
- Python intègre, comme Java ou les versions récentes de C++, un système d'exceptions, qui permettent de simplifier considérablement la gestion des erreurs.
- Comme Scheme ou SmallTalk, Python est dynamiquement typé. Tout objet manipulable par le programmeur possède un type bien défini à l'exécution, qui n'a pas besoin d'être déclaré à l'avance.
- Python est extensible : comme Tcl ou Guile, il peut facilement être interfacé avec des bibliothèques C existantes. il peut aussi être utilisé comme d'un langage d'extension pour des systèmes logiciels complexes.
- La bibliothèque standard de Python et les paquetages contribués, donnent accès à une grande variété de services : chaînes de caractères et expressions régulières, services UNIX standards (fichiers, pipes, signaux, sockets, threads...), protocoles Internet (Web, News, FTP, CGI, HTML...), persistance et bases de données, interfaces graphiques.

– Python est un langage qui continue à évoluer, soutenu par une communauté d'utilisateurs enthousiastes et responsables, dont la plupart sont des supporters du logiciel libre. Parallèlement à l'interpréteur principal, écrit en C et maintenu par le créateur du langage, un deuxième interpréteur, écrit en Java, est en cours de développement.

J'ai donc appris ce langage de programmation. Ce choix s'est révélé être très efficace : les 300 lignes en C donnaient à peine 20 lignes en Python et avec un *parser* complet et fonctionnel.

Chapitre 4

Communiquer

L'envoi et la réception des données nécessitent une connexion entre le serveur et le contrôleur.

4.1 Le réseau, brèves notions théoriques

La communication entre deux programmes nécessite trois éléments indispensables :
- connaître l'**addresse IP** de l'ordinateur sur lequel se trouve le programme avec qui nous voulons communiquer ;
- utiliser un **port** libre et ouvert ;
- utiliser le même **protocole** de transmission des données.

4.1.1 L'addresse IP

Chaque ordinateur est identifié sur le réseau par ce qu'on appelle une adresse IP. Il s'agit d'une série de chiffres, par exemple :

86.216.28.119

Cette adresse représente un ordinateur.

Mais un ordinateur peut avoir plusieurs IP. On en compte en général trois :
- Une IP interne : le *localhost*, aussi appelé *loopback*. C'est une IP qui sert pour communiquer avec soi-même (127.0.0.1) ;
- Une IP du réseau local : si plusieurs ordinateurs sont en réseau privé, ils peuvent communiquer entre eux sans passer par internet grâce à ces IP. Elles sont propres à ce réseau ;
- Une IP internet : c'est l'IP utilisée pour communiquer avec tous les autres ordinateurs de la planète qui sont connectés à internet.

4.1.2 Les ports

Le concept du système de ports a été inventé afin de ne pas mélanger les informations arrivant sur une machine (mails, page web...). Un port est un nombre compris entre 1 et 65 536. Voici quelques ports standards :

- **21** : utilisé par les logiciels FTP pour envoyer et recevoir des fichiers ;
- **80** : utilisé pour naviguer sur le web par les navigateurs ;
- **110** : utilisé pour la réception de mails.

La plupart des ports dont les numéros sont inférieurs à 1 024 sont déjà réservés par la machine. Les ports de 1 024 à 65 536 sont librement utilisables.

4.1.3 Le protocole

Un protocole est un ensemble de règles qui permettent à deux ordinateurs de communiquer. Il faut impérativement que les 2 ordinateurs utilisent le même protocole pour que l'échange de données puisse fonctionner. Il existe des centaines de protocoles de communication différents. Ceux-ci peuvent être très simples comme très complexes, selon s'ils sont de "haut" ou "bas" niveau :

- Protocoles de "haut" niveau : par exemple le protocole FTP, qui utilise le port 21 pour envoyer et recevoir des fichiers, est un système d'échange de données de haut niveau. Son mode de fonctionnement est déjà écrit et documenté. Il est donc assez facile à utiliser, mais on ne peut pas lui rajouter de possibilités.
- Protocoles de "bas" niveau : par exemple le protocole TCP. Il est utilisé par les programmes pour lesquels aucun protocole de haut niveau ne convient. Les données qui transitent sur le réseau sont manipulées octet par octet.

Nous utiliserons le protocole bas niveau TCP. Il nécessite d'établir une connexion au préalable entre les ordinateurs. Un système de contrôle permet de demander à renvoyer un paquet[1] au cas où l'un d'entre eux se serait perdu sur le réseau. L'utilisation du protocole TCP permet d'avoir l'assurance que tous les paquets arrivent à destination, et dans l'ordre voulu.

4.2 Architecture du projet

Le projet sera de type *client-serveur*. C'est l'architecture réseau la plus classique et la plus simple à mettre en oeuvre. Les machines qui demandent la connexion sont appelées des "clients". En plus de ces machines, on utilise un autre ordinateur (appelé "serveur") qui se charge de répartir les communications entre les clients.

[1] Les données s'envoient sur le réseau morceau par morceau. On parle de paquets, qui peuvent être chacun découpés en sous-paquets

4.2.1 Le serveur

Le serveur est l'application qui génère le monde où se trouve le rover et qui collecte les informations le concernant. Cette application est fournie par les organisateurs du concours.
Nous le lançons à l'aide de la commande suivante :

```
./server -v -p 50000 ../sample-maps/map.wrld
```

4.2.2 Le client

Le client est l'application qui reçoit et interprète les informations du serveur. C'est le contrôleur. Il est lancé avec la commande suivante :

```
python rover4.py localhost 50000
```

Voici le code de base pour se connecter au serveur et recevoir les informations :

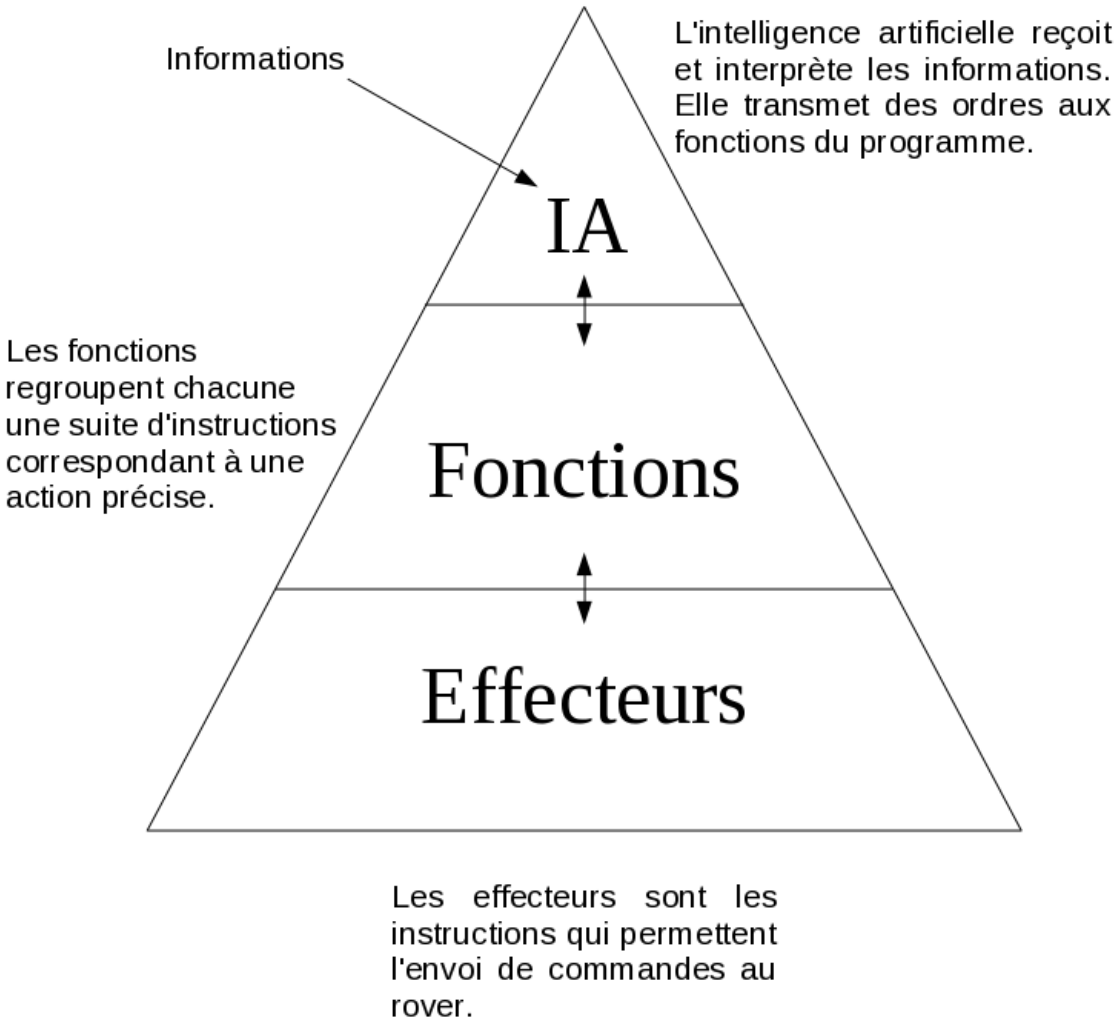
```
#!/usr/bin/python
from socket import *
import sys
import getopt

class Rover :

    data = ""
    data_field = ""
    s = ""

    def init(self, host, port) :
        print "----------Initialization"
        self.s = socket(AF_INET,SOCK_STREAM)
        self.s.connect((host, port))
        self.data = self.s.recv(1024)
        self.data_field = self.data.split(' ')
        print self.data_field

def connect_to_server(host, port) :

    end = 1

    #On cree un objet de la classe Rover()
    rover = Rover()

    #on appelle la methode d'initialisation
    rover.init(host, port)
    print "----------START !"
    while(end):

    # fin de connection
    rover.s.close()

def main():
```

```
    #Debut du programme

    optlist, llist = getopt.getopt(sys.argv[1:], '-h')
    host = llist[0]
    port = llist[1]
    #On appelle la fonction pour se connecter
    connect_to_server(host, int(port))

if __name__ == '__main__' :
    main()
```

Le programme débute dans la fonction *main* appelant la fonction *connect to server* . Cette dernière crée un objet de la classe *Rover* et appelle la méthode *init* ouvrant la connexion sur le serveur.

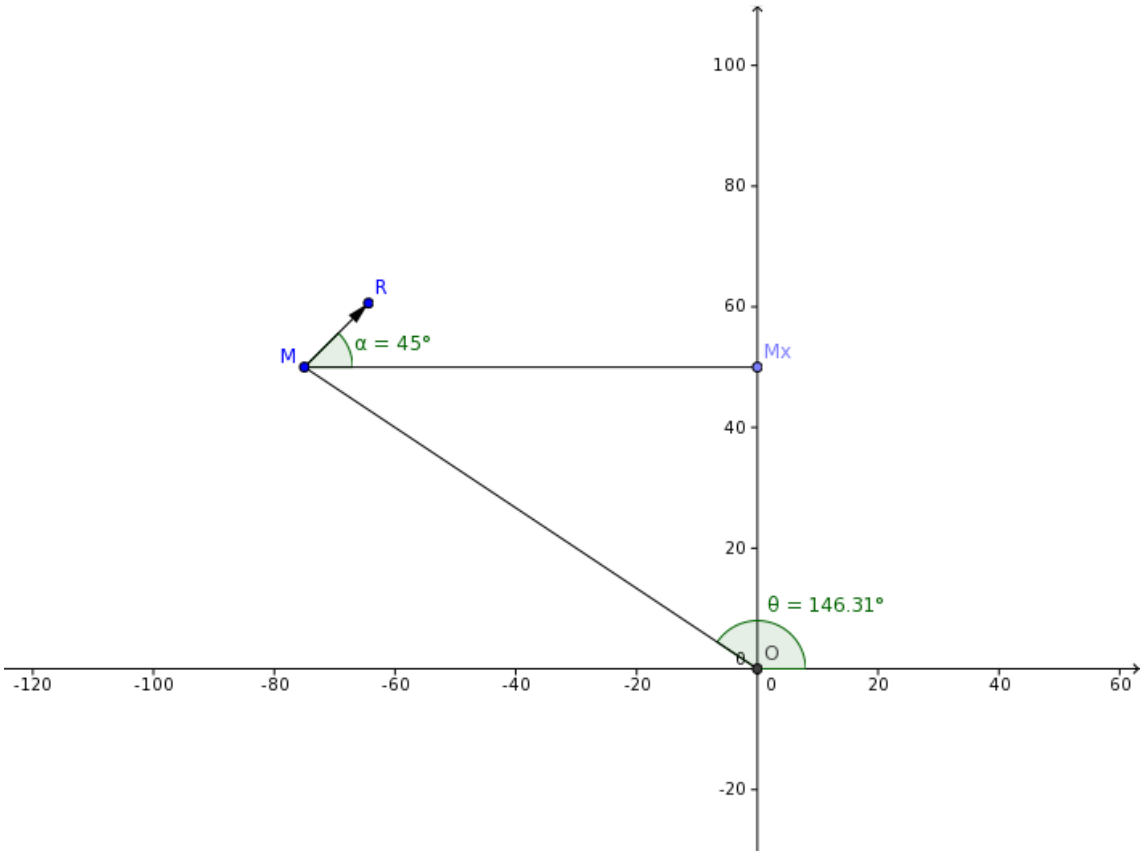
Chapitre 5

Algorithmique



5.1 Trouver la base

Soit, dans un repère orthonormé, le point $M(x;y)$ indiquant la position du rover, $\vec{MR}$ son vecteur directeur et $\vec{MO}$ le vecteur de la trajectoire voulue.



Pour que le rover se dirige vers la base, il doit effectuer une rotation afin que l'angle orienté $(\vec{MR}, \vec{MO})$ (ou φ) soit nul.

$\vartheta = (\vec{Ox}, \vec{OM}) \equiv \tan\vartheta = \frac{y_{\vec{(OM)}}}{x_{\vec{(OM)}}} \equiv \vartheta = \arctan\frac{y_{\vec{(OM)}}}{x_{\vec{(OM)}}}$

$\varphi = (\vec{MR}, \vec{MO})$

$\varphi = (\vec{MR}, \vec{Ox}) + (\vec{Ox}, \vec{MO})$

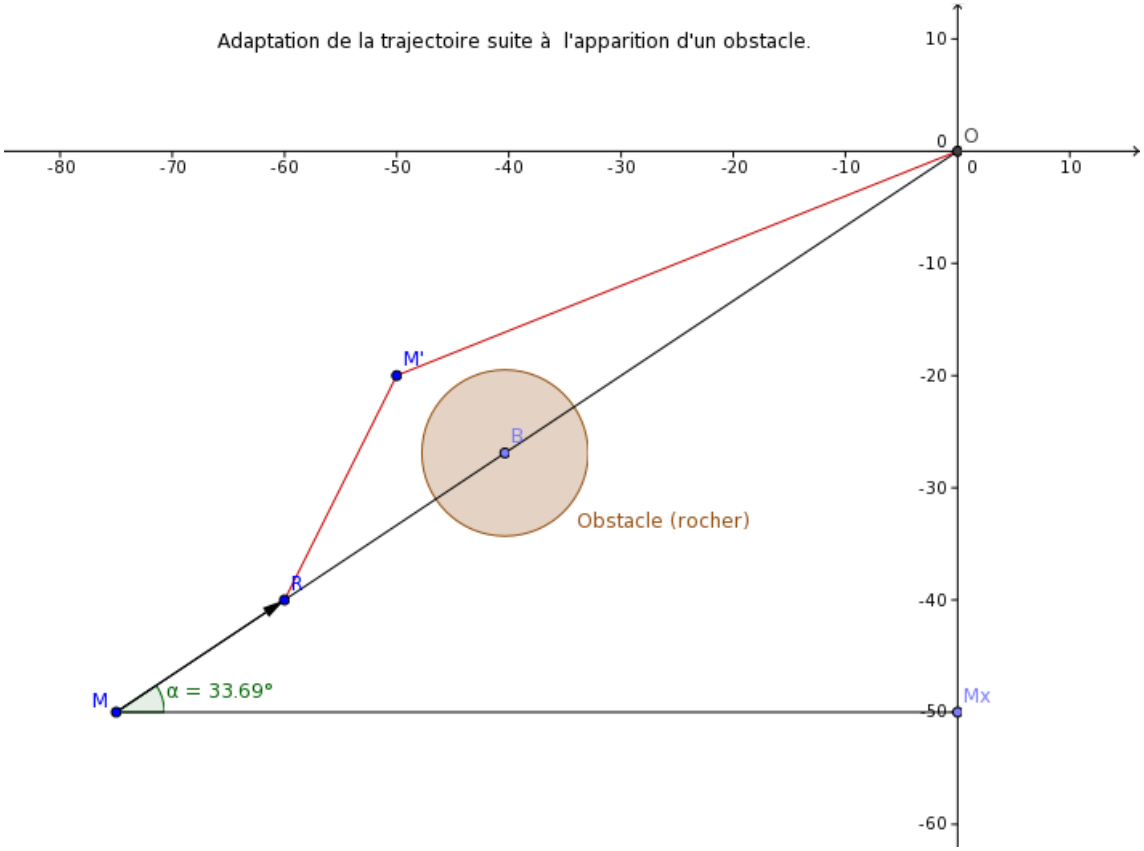
$\varphi = -(\vec{Ox}, \vec{MR}) + (\vec{Ox}, \vec{MO})$

$\varphi = -\alpha + (\vec{Ox}, \vec{OM}) + \pi$

$\varphi = -\alpha + \vartheta + \pi$

Le rover doit donc tourner jusqu'à ce qu'il ait effectué un angle de φ degrés. Cette valeur est envoyée à la méthode *turnRight* qui est chargée d'envoyer la commande de rotation et de l'arrêter une fois l'angle atteint.

5.2 Éviter les rochers



Les coordonnées de chaque élément se trouvant dans le champ de vision du rover sont envoyées par la chaîne de données télémétriques. La méthode *searchObstacle* parcourt cette chaîne jusqu'à trouver les caractères indiquant la présence d'un obstacle potentiel. On détermine la distance qui le sépare du rover avec le théorème de Pythagore :

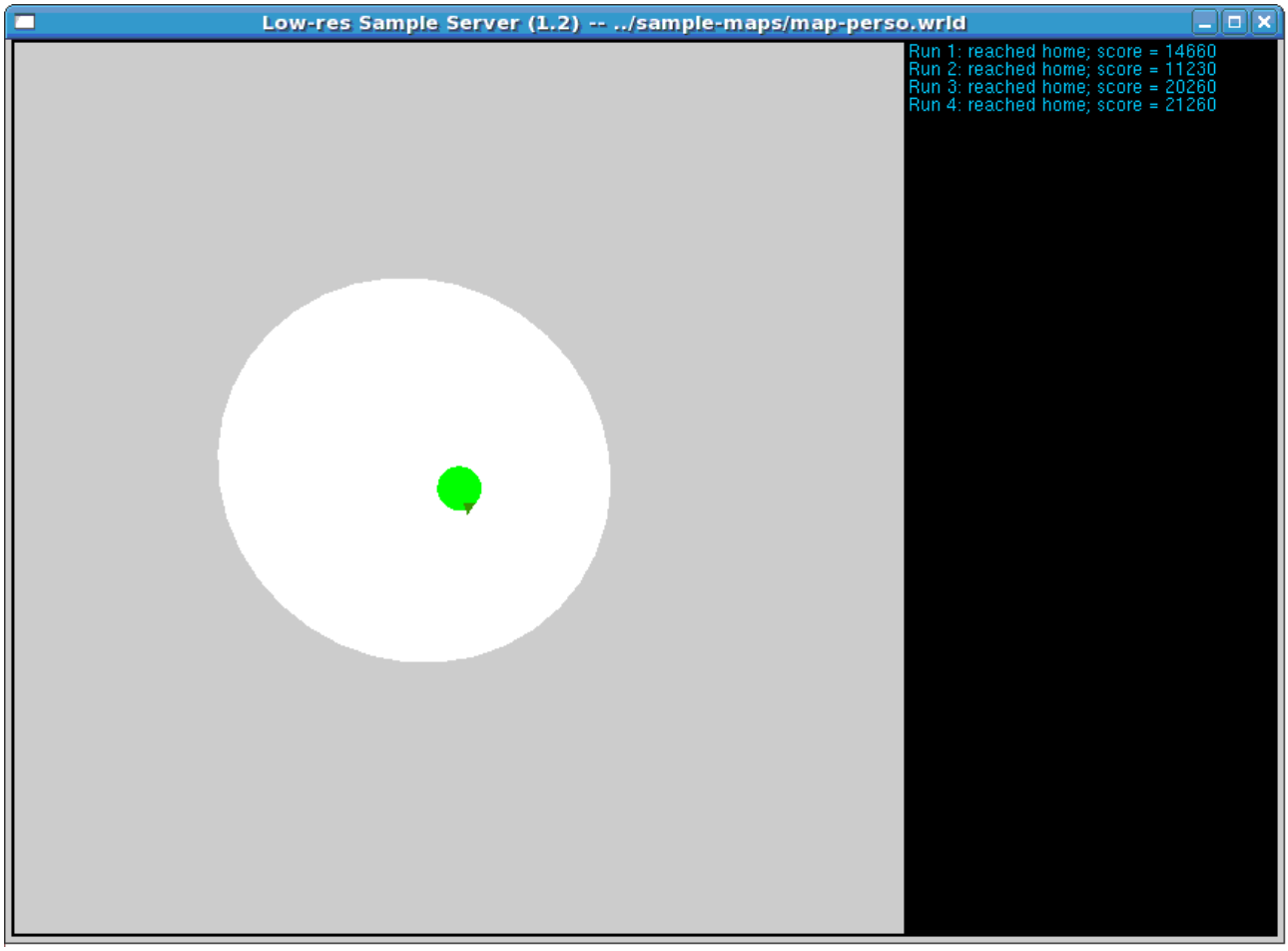
$$\sqrt{(x_{element} - x_{rover})^2 + (y_{element} - y_{rover})^2}$$

Cela nous permet de savoir quel élément est le plus proche du rover.
On cherche ensuite à savoir si cet élément est sur la trajectoire du rover. Il s'agit de trouver une équation de la droite de cette trajectoire à partir des coordonnées du vecteur directeur du rover. Sachant que $y = mx + p$, $m = \frac{y_{\vec{(OM)}}}{x_{\vec{(OM)}}}$ et $p = 0$ puisque le rover se rend à l'origine du repère. Si l'équation suivante est vérifiée, alors l'élément se trouve sur la trajectoire du rover :

$$y_{element} = m \times x_{element}$$

Le rover doit alors contourner l'obstacle et se réorienter vers la base.

Après une exécution de quatre essais, les résultats sont très satisfaisants :



Le rover a rejoint sa base quatre fois.

5.3 Éviter les Martiens

Les coordonnées de chaque Martien sont données de la même manière que celles des rochers et des cratères. On considère que la collision a lieu lorsque le rover et le Martien se trouvent au même point $I(x_I, y_I)$. Ce point est le point d'intersection des droites des trajectoires du rover et du Martien.
Soit Δ la droite de la trajectoire du rover.

$$\Delta : y = \frac{y_{\vec{(OM)}}}{x_{\vec{(OM)}}} x$$

Soit Γ la droite de la trajectoire du Martien. $\Gamma : y = mx + p$
Les positions successives du Martien sont enregistrées afin de pouvoir choisir deux séries de coordonnées consécutives et trouver une équation de la droite Γ. Ces points sont appelés $A(x_A, y_A)$ et $B(x_B, y_B)$. On cherche m et p :

$$m = \frac{y_B - y_A}{x_B - x_A}$$

et

$$p = y_A - m \times x_A$$

Les coordonnées du point I sont les solutions du système d'équations :

$$\begin{cases} y = \dfrac{y_{\vec{(OM)}}}{x_{\vec{(OM)}}} x \\ y = \frac{y_B - y_A}{x_B - x_A} x + y_A - \frac{y_B - y_A}{x_B - x_A} \times x_A \end{cases}$$

On néglige le temps qu'il faut à chacun pour rejoindre ce point et on considére le point I comme un obstacle pour le rover.

Chapitre 6

Programmation

6.1 Intelligence artificielle

6.1.1 Prendre une décision

La méthode *takeDecision* est chargée, comme son nom l'indique, de prendre une décision en fonction des informations reçues.

La méthode est définie dans la classe *Rover* avec en paramètres l'IP de l'ordinateur hébergeant le serveur et le port par lequel les informations doivent circuler. Avant de récupérer les informations du rover, on vérifie si la base est dans notre champ de vision, on s'oriente vers elle (*orientHome* cf section suivante) et on avance.

```
def takeDecision(self,host, port):

        is_base_in_sight = 0;
        self.orientHome(host,port)
        self.accelerate(host,port)
```

Le mot-clef **while** indique le début d'une boucle, c'est-à-dire une suite d'instructions à répéter tant qu'une condition est vérifiée. Ici, la boucle se fait tant que le premier caractère de la dernière chaîne d'informations reçue n'est pas un "E", indicateur de la fin de l'essai.

```
        while(self.data_field[0] != 'E'):
```

On commence par recevoir les données du serveur que l'on traite pour isoler les variables et pouvoir les traiter numériquement.

```
            self.data = self.s.recv(1024)
            self.data_field = self.data.split('_')
            self.s.send(";;")
```

Puis on pose les conditions qui détermineront la marche à suivre pour que l'essai soit un succès. Si le caractère d'information est un "S", le véhicule a rejoint sa base.

```
            if(self.data_field[0] == 'S'):
                print "----------YES ! The vehicle reached home
                    base safely !"
```

Si le caractère d'information est un "B", c'est que le rover est entré en collision avec un rocher. Il doit s'arrêter, se réorienter et repartir.

```
            if(self.data_field[0] == 'B'):
                print "----------WARNING !"
                self.stop(host,port)
                self.orientHome(host,port)
                self.accelerate(host,port)
```

Si la chaîne contient un "h" et si la base n'était pas en vue avant, c'est qu'elle vient d'apparaitre dans le champ de vision du rover.

```
            if((self.data_field.count('h') == 1) and is_base_in_sight == 0):
                print "----------BASE IN SIGHT !"
                is_base_in_sight = 1
```

Si le caractère "h" n'est pas présent dans la chaîne de données et si la base était en vue, c'est qu'elle n'est plus dans son champ de vision. Il doit s'arrêter, se réorienter et repartir vers la base.

```
            if((self.data_field.count('h') == 0) and is_base_in_sight == 1):
                print "----------BASE LOST !"
                is_base_in_sight = 0
                self.stop(host,port)
                self.orientHome(host,port)
                self.accelerate(host,port)
```

Si le premier caractère est un "E" alors la boucle ne s'exécute pas et c'est la fin de l'essai.

```
        print "----------END"
```

6.1.2 Trouver la base

La méthode *orientHome* se charge de calculer la mesure de l'angle φ et la transmet à la méthode *turnRight* pour orienter le rover vers la base.
On commence par définir la méthode :

```
    def orientHome(self, host, port):
```

On reçoit les données du serveur et on les affiche.

```
        self.data = self.s.recv(1024)
        self.data_field = self.data.split(' ')
        print "----------Orienting for home"
        print self.data_field
```

On prend les valeurs 3 et 4 de la chaîne pour les coordonnées du rover.

```
        X = float(self.data_field[3])
        Y = float(self.data_field[4])
```

Enfin, on calcule α, theta puis φ et on transmet φ à *turnRight*

```
        alpha = float(self.data_field[5])
        alpha=convertAlpha(alpha)

        theta = calculateTheta(X,Y)

        phi = calculatePhi(alpha,theta)

        self.turnRight(host, port, phi)


        print "--------------------------------------------------Going at home"
```

6.1.3 Éviter les obstacles

La méthode *searchObstacle* analyse les chaînes de données télémétriques et enregistre les positions des objets en vue.
On parcourt ensuite le tableau pour savoir quel objet est le plus proche du rover.

```
def searchObstacle(self,is_base_in_sight,host,port) :
        obstacles_number = self.data_field.count('b') + self.data_field.count('c')
        closest_element = 200;
        xclosest = 0
        yclosest = 0
        rclosest = 0
        position = 0

        X = float(self.data_field[3])
        Y = float(self.data_field[4])

        xE = -100
        yE = 100
        rE = 0

        i = 0

        len_data = len(self.data_field)
        while(i < len_data):
            if (self.data_field[i] == 'b' or self.data_field[i] == 'c'):
                xE = float(self.data_field[i+1])
                yE = float(self.data_field[i+2])

            print "++ xE %f | yE %f | X %f | Y %f" %(xE,yE,X,Y)

            length = math.sqrt(math.pow(xE-X,2)+math.pow(yE-Y,2))
            if (length < closest_element) :
                closest_element = length
                position = obstacles_number
                xclosest = xE
                yclosest = yE
            i+=1
```

```
        print "closest %f | position %s | lenght data %f | i %f" %(closest_element, position, len_data
            , i)
```

Puis on vérifie si l'objet est dans la trajectoire du rover. Si c'est le cas, on fait tourner le rover de façon à ce qu'il passe à coté de l'objet. Une fois celui-ci dépassé, il se réoriente et reprend son chemain vers la base.

```
        # y = ax + b
        xOM = 0 - X
        yOM = 0 - Y

        a = yOM/xOM

        if ((yclosest > a*xclosest-8) and (yclosest < a*xclosest+8) and (closest_element < 15)):
            print "----------------------------------------------------------ALERT ! OBSTACLES ON THE WAY !"
            self.data = self.s.recv(1024)
            self.data_field = self.data.split(' ')
            self.s.send(";l;")
            boucle = 1

            timefirst = float(self.data_field[1])

            while(boucle) :
                self.data = self.s.recv(1024)
                self.data_field = self.data.split(' ')
                X = float(self.data_field[3])
                Y = float(self.data_field[4])
                timesecond = float(self.data_field[1])
                xOM = 0 - X
                yOM = 0 - Y
                a = yOM/xOM
                print "Turning... %f" %(timesecond-timefirst)
                if((timesecond-timefirst) >=3000):
                    print "STOP"
                    boucle = 0

            self.s.send(";r;")
            self.stop(host,port)
            self.orientHome(host,port)
            self.accelerate(host,port)

            timefirst = float(self.data_field[1])

            boucle = 1
            while(boucle) :
                self.data = self.s.recv(1024)
                self.data_field = self.data.split(' ')
                timesecond = float(self.data_field[1])
                print "%f" %(timesecond-timefirst)
                if((timesecond-timefirst) >=2000):
                    boucle = 0;
```

6.2 Fonctions

Certaines méthodes de la classe *Rover* appartiennent au groupe des "fonctions" du programme. Elles organisent une seule et unique action comme "tourner à droite". Elle sont toutes construites de la même manière. Nous prendrons comme exemple la fonction *turnRight*. On obtient par les paramètres la valeur de l'angle de rotation à effectuer. On envoie donc au rover l'ordre de tourner à droite.

```
    def turnRight(self, host, port, phi_depart) :
        print "----------Turn right"
        self.data = self.s.recv(1024)
        self.data_field = self.data.split(' ')
        self.s.send(";r;")
        boucle = 1

        actual_phi = 0
```

Et on le laisse tourner jusqu'à ce que l'angle soit obtenu. Quand c'est le cas, on envoie la commande inverse pour que le rover reprenne une trajectoire rectiligne.

```
        while(boucle) :
            self.data = self.s.recv(1024)
            self.data_field = self.data.split(' ')
            if  (self.data_field[0] == 'T'):
                X = float(self.data_field[3])
                Y = float(self.data_field[4])

                alpha = float(self.data_field[5])
            alpha=convertAlpha(alpha)

            theta = calculateTheta(X,Y)

            actual_phi = calculatePhi(alpha,theta)

            #print "|alpha %f|actual_phi %f|r|" %(alpha,actual_phi)

            if X < 0 and Y > 0:
                if (actual_phi) < 1.5 and (actual_phi) > 0.5 :
                    boucle = 0

            elif X < 0 and Y < 0:
                if (math.fabs(actual_phi)) < 1 and (float(math.fabs(actual_phi))) > -1.0 :
                    boucle = 0

            elif X > 0 and Y > 0:
                if (math.fabs(actual_phi)) < 183 and (math.fabs(actual_phi)) > 179.5:
                    boucle = 0

            elif X > 0 and Y < 0:
                if (math.fabs(actual_phi)) < 180 and (math.fabs(actual_phi)) > 175:
                    boucle = 0

        self.s.send(";l;")
```

6.3 Effecteurs

Les effecteurs sont les ordres bruts envoyés au rover. Ils sont choisis par les fonctions. Pour exemple, l'effecteur $s.send(";ar;")$ ordonne au rover de tourner à droite et d'accélérer.

Chapitre 7

Conclusion

Les difficultés auxquelles j'ai été confronté durant la conception du programme résident dans la différence qu'il existe entre le modèle mathématique et la réalité. En effet, le modèle mathématique ne prend pas en compte les marges d'erreur alors que l'on manipule des variables de valeurs approximatives dans la réalité. Aussi, de nombreux tests ont été nécessaires avant d'obtenir les bons paramètres.

En cummulant les connaissances acquises lors des deux premiers stages et celles de celui-ci, je suis capable de fabriquer un robot en mesure d'analyser son environnement, prendre des décisions et se déplacer, et ce de manière complètement autonome.

Sources et documentation

Site web du Laboratoire Informatique de l'École Polytechnique :

`http://www.lix.polytechnique.fr`

Sujet du concours :

`http://smlnj.org/icfp08-contest/task.html`

Programmation :

`http://www.developpez.com`

`http://www.siteduzero.com`

`http://en.wikipedia.org/wiki/Python_(programming_language)`

Système :

`http://en.wikipedia.org/wiki/Linux`

Images :

Les images du logiciel sont des captures d'écran des exécutions réalisées durant le stage.

Les images ont été retravaillées à l'aide du logiciel de de dessin PaintDotNet

`http://www.getpaint.net/`

L'image en page de garde provient de la banque d'images de la NASA. Les images n'y sont soumises à aucun copyright si leur utilisation a un but éducatif ou informatif. [1]

`http://www.nasaimages.org/`

Les schémas mathématiques ont été conçus avec le logiciel Geogebra

`http://www.geogebra.org/`

Ce rapport a été rédigé en LaTeX.

`http://fr.wikipedia.org/wiki/LaTeX`

[1] Termes d'utilisation : "The NASA imagery offered on NASAIMAGES.ORG is generally not copyrighted. You may use this NASA imagery for educational or informational purposes, including photo collections, textbooks, public exhibits and Internet Web pages (personal or otherwise)."

Annexes

Le code complet du programme :

```
#!/usr/bin/python
from socket import *
import sys
import getopt
import math

class Rover :

    data = ""
    data_field = ""
    s = ""

    def init(self, host, port) :
        print "----------Initialization"
        self.s = socket(AF_INET,SOCK_STREAM)
        self.s.connect((host, port))
        self.data = self.s.recv(1024)
        self.data_field = self.data.split(' ')
        print self.data_field

    def stop(self, host, port) :
        print "----------Stop"
        self.data = self.s.recv(1024)
        self.data_field = self.data.split(' ')
        self.s.send(";b;")
        while(float(self.data_field[6]) > 0) :
            self.data = self.s.recv(1024)
            self.data_field = self.data.split(' ')
            self.s.send(";b;")
        self.s.send(";a;")

    def turnRight(self, host, port, phi_depart) :
        print "----------Turn right"
        self.data = self.s.recv(1024)
        self.data_field = self.data.split(' ')
        self.s.send(";r;")
        boucle = 1

        actual_phi = 0

        while(boucle) :
            self.data = self.s.recv(1024)
            self.data_field = self.data.split(' ')
            if  (self.data_field[0] == 'T'):
                X = float(self.data_field[3])
                Y = float(self.data_field[4])

                alpha = float(self.data_field[5])
            alpha=convertAlpha(alpha)

            theta = calculateTheta(X,Y)

            actual_phi = calculatePhi(alpha,theta)
```

```
            #print "|alpha %f|actual_phi %f|r|" %(alpha,actual_phi)

            if X < 0 and Y > 0:
                if (actual_phi) < 1.5 and (actual_phi) > 0.5 :
                    boucle = 0

            elif X < 0 and Y < 0:
                if (math.fabs(actual_phi)) < 1 and (float(math.fabs(actual_phi))) > -1.0 :
                    boucle = 0

            elif X > 0 and Y > 0:
                if (math.fabs(actual_phi)) < 183 and (math.fabs(actual_phi)) > 179.5:
                    boucle = 0

            elif X > 0 and Y < 0:
                if (math.fabs(actual_phi)) < 180 and (math.fabs(actual_phi)) > 175:
                    boucle = 0

        self.s.send(";l;")

    def turnLeft(self, host, port, phi_depart) :
        print "----------Turn Left"
        self.data = self.s.recv(1024)
        self.data_field = self.data.split(' ')
        self.s.send(";l;")
        boucle = 1

        actual_phi = 0

        while(boucle) :
            self.data = self.s.recv(1024)
            self.data_field = self.data.split(' ')

            X = float(self.data_field[3])
            Y = float(self.data_field[4])

            alpha = float(self.data_field[5])
            alpha = convertAlpha(alpha)

            theta = calculateTheta(X,Y)

            actual_phi = calculatePhi(alpha,theta)

            #print " |%s|alpha %f|theta %f|actual_phi %f|l" %(self.data_field,alpha,theta,actual_phi
                )

            if X < 0 and Y > 0:
                if (actual_phi) < 0 and (actual_phi) > -1 :
                    boucle = 0

            elif X < 0 and Y < 0:
                if (math.fabs(actual_phi)) < 1 and (float(math.fabs(actual_phi))) > -1 :
                    boucle = 0

            elif X > 0 and Y > 0:
                if (math.fabs(actual_phi)) < 180.5 and (math.fabs(actual_phi)) > 180:
                    boucle = 0
            elif X > 0 and Y < 0:
                if (math.fabs(actual_phi)) < 180.5 and (math.fabs(actual_phi)) > 180:
                    boucle = 0

        self.s.send(";r;")

    def accelerate(self, host, port) :
        print "----------Accelerate"
        self.data = self.s.recv(1024)
        self.data_field = self.data.split(' ')
        self.s.send(";a;")

    def orientHome(self, host, port):

        self.data = self.s.recv(1024)
        self.data_field = self.data.split(' ')
```

```
        print "----------------------------------------Orienting for home"
        print self.data_field

        X = float(self.data_field[3])
        Y = float(self.data_field[4])

        alpha = float(self.data_field[5])
        alpha=convertAlpha(alpha)

        theta = calculateTheta(X,Y)

        phi = calculatePhi(alpha,theta)

        self.turnRight(host, port, phi)


        print "----------------------------------------Going at home"


    def takeDecision(self,host, port):

        is_base_in_sight = 0;
        self.orientHome(host,port)
        self.accelerate(host,port)
        while(self.data_field[0] != 'E'):
            self.data = self.s.recv(1024)
            self.data_field = self.data.split(' ')
            self.s.send(";;")

            if(self.data_field[0] == 'S'):
                print "----------------------------------------YES ! The vehicle reached home base
                    safely !"
                is_base_in_sight = 0

            if(self.data_field[0] == 'B'):
                print "----------------------------------------WARNING !"
                self.stop(host,port)
                self.orientHome(host,port)
                self.accelerate(host,port)

            if((self.data_field.count('h') == 1) and is_base_in_sight == 0):
                print "----------------------------------------BASE IN SIGHT !"
                is_base_in_sight = 1

            if((self.data_field.count('h') == 0) and is_base_in_sight == 1):
                print "----------------------------------------BASE LOST !"
                is_base_in_sight = 0
                #self.stop(host,port)
                #self.orientHome(host,port)
                #self.accelerate(host,port)

            if((self.data_field.count('b') >=1)):
                self.searchObstacle(is_base_in_sight,host,port)


        print "----------------------------------------END"

    def searchObstacle(self,is_base_in_sight,host,port) :
        obstacles_number = self.data_field.count('b') + self.data_field.count('c')
        closest_element = 200;
        xclosest = 0
        yclosest = 0
        rclosest = 0
        position = 0

        X = float(self.data_field[3])
        Y = float(self.data_field[4])

        xE = -100
        yE = 100
        rE = 0
```

```
        i = 0

        len_data = len(self.data_field)
        while(i < len_data):
            if (self.data_field[i] == 'b' or self.data_field[i] == 'c'):
                xE = float(self.data_field[i+1])
                yE = float(self.data_field[i+2])

            print "++ xE %f | yE %f | X %f | Y %f" %(xE,yE,X,Y)

            length = math.sqrt(math.pow(xE-X,2)+math.pow(yE-Y,2))
            if (length < closest_element) :
                closest_element = length
                position = obstacles_number
                xclosest = xE
                yclosest = yE
            i+=1

        print "closest %f | position %s | lenght data %f | i %f" %(closest_element,position,len_data
            ,i)

        # y = ax + b
        xOM = 0 - X
        yOM = 0 - Y

        a = yOM/xOM

        if ((yclosest > a*xclosest-8) and (yclosest < a*xclosest+8) and (closest_element < 15)):
            print "----------ALERT ! OBSTACLES ON THE WAY !"
            self.data = self.s.recv(1024)
            self.data_field = self.data.split(' ')
            self.s.send(";l;")
            boucle = 1

            timefirst = float(self.data_field[1])

            while(boucle) :
                self.data = self.s.recv(1024)
                self.data_field = self.data.split(' ')
                X = float(self.data_field[3])
                Y = float(self.data_field[4])
                timesecond = float(self.data_field[1])
                xOM = 0 - X
                yOM = 0 - Y
                a = yOM/xOM
                print "Turning... %f" %(timesecond-timefirst)
                if((timesecond-timefirst) >=3000):
                    print "STOP"
                    boucle = 0

            self.s.send(";r;")
            self.stop(host,port)
            self.orientHome(host,port)
            self.accelerate(host,port)

            timefirst = float(self.data_field[1])

            boucle = 1
            while(boucle) :
                self.data = self.s.recv(1024)
                self.data_field = self.data.split(' ')
                timesecond = float(self.data_field[1])
                print "%f" %(timesecond-timefirst)
                if((timesecond-timefirst) >=2000):
                    boucle = 0;

def convertAlpha(alpha):
    #alpha = alpha+360
    return alpha

def calculateTheta(X,Y):
    xOM = 0 - X
    yOM = 0 - Y
```

```
    #theta = (>Ox,>OM)
    theta = math.degrees(math.atan(yOM/xOM))

    return theta

def calculatePhi(alpha,theta):
    #phi = -alpha + theta + 180
    phi = (alpha-(2*alpha)) + theta + 3.14*2
    return phi

def connect_to_server(host, port) :

    end = 1

    rover = Rover()

    rover.init(host, port)
    print "--------------------------------------START !"
    while(end):
        rover.takeDecision(host, port)

    # fin de connection
    rover.s.close()

def main():
    optlist, llist = getopt.getopt(sys.argv[1:], '-h')
    host = llist[0]
    port = llist[1]
    connect_to_server(host, int(port))

if __name__ == '__main__' :
    main()
```