2011

Etude du Middleware robotique ROS

Robot Operating System

Stage technicien : 6 semaines

Tuteur de stage : Etienne PERRET

Maître de stage : Nicolas TURRO

Daniel COLY

Etude du middleware robotique ROS

02/09/2011

Introduction

Introduction

Mon stage a eu lieu à l'INRIA Grenoble Rhône-Alpes, un centre de recherche spécialisé en informatique situé dans la zone industrielle de Montbonnot où réside de nombreuses entreprises et laboratoires de recherches comme celui de l'ENSIMAG qui appartient d'ailleurs au groupe Grenoble INP.

Mon stage s'est déroulé en 6 semaines. Le but de mon stage est d'étudier l'utilisation d'un middleware robotique nommé ROS (Robot Operating System) à la place d'Hugr, un middleware « maison » développé par l'institut.

L'INRIA

L'institut national de recherche en informatique et en automatique (INRIA) est un établissement public à caractère scientifique et technologique créé en 1967. Son ambition est de mettre en réseau les compétences et talents de l'ensemble du dispositif de recherche français, dans le domaine des sciences et technologies de l'information et de la communication. L'INRIA est composé de 8 centres de recherche autonomes répartis sur tout le territoire Français.

Présentation du service SED

Mon tuteur de stage Nicolas TURRO est un ingénieur SED. Le SED (Service Expérimentation et Développement) est une équipe d'ingénieurs qui font des développements et des expérimentations avec les équipes de recherche. Leur but est de développer des outils matériels et logiciels pour les chercheurs du centre. Leur travail leur permet de mener à bien leurs expérimentations.

Cadre de travail

Le centre se situe à quelques centaines de mètres de mon logement ce qui est pratique en soi puisque je ne suis pas soumis aux contraintes des transports en communs. Une bibliothèque est ouverte quasiment tous les jours. Elle recueille des ouvrages de tous types. J'ai pu en profiter pour me documenter sur le langage C++ lorsque mon maître de stage s'est absenté au milieu de mon stage.

Une cafétéria est mise au profit de tous, l'ambiance y est très conviviale. Il est très courant de croiser des ingénieurs et chercheurs dans les couloirs, j'ai pu discuter avec quelques-uns d'entre eux sur leur vécu et leur parcours au sein et en dehors de l'institut.

Environnement de travail

Mon poste de travail se trouve dans un hall robotique. Dès mon entrée, j'ai été agréablement surpris par la présence d'une Toyota Lexus Hybride. D'autres modules électroniques et robotiques sont placés un peu partout dans le hall. J'ai pu reconnaître notamment le CyCab, un véhicule intelligent et non polluant.

Le hall robotique accueille également 3 autres stagiaires. Deux d'entre eux entrent en 3ème année à l'ENSIMAG, le troisième passe en Master 2 Maths Info à l'université Joseph Fourier.

J'ai accès à un ordinateur sur lequel je travaille sous l'OS Fedora, une distribution linux similaire à Debian (utilisée à l'ESISAR).

La grande majorité des commandes restent les mêmes. Mon maître de stage a son poste de travail à côté du mien, je peux donc lui poser des questions si besoin est.

Contexte

Dans le cadre du projet Arosdyn d'assistance à la conduite automobile, les modules logiciels haut niveau et les drivers de capteurs bas niveau communiquent à travers un middleware logiciel « maison » nommé Hugr. L'objet de ce stage est d'étudier l'utilisation du middleware logiciel ROS (Robot Operating System) à la place d'Hugr.

ROS est un middleware robotique open source possédant une large audience et en voie de devenir un standard de fait dans le milieu académique et hobbyiste. Son utilisation permettrait au centre d'accéder à une large bibliothèque d'algorithmes implémentés et distribués avec ROS, ainsi que de partager plus facilement nos modules logiciels avec des collaborateurs extérieurs à l'INRIA utilisant ROS.

Objectifs et déroulement du stage

ROS offre-t-il les mêmes fonctionnalités que Hugr en termes d'affichage et pilotage de l'expérimentation ? Mon travail a été découpé en trois parties.

2 semaines : Découverte et prise en main du logiciel ROS

- Installation du logiciel, lecture de la documentation.
- Compréhension du rôle d'un middleware robotique.
- Implantation des nombreux tutoriels, passage en revue des bibliothèques déjà disponibles.
- Prise en main de l'afficheur interactif 3D rviz.

2 semaines : Conception et développement

- Réalisation d'un programme qui à partir de positions x y z et du temps t fournies par un fichier, affiche en temps réel la courbe correspondante sur rviz.
- Réalisation d'un programme qui convertit des coordonnées GPS latitude longitude altitude en coordonnées cartésiennes et qui affiche la courbe correspondante sur rviz.
- Réalisation d'un programme qui récupère les données fournies en sortie du GPS via un port série, et envoie ces données vers un autre programme qui traitera les données pour y faire apparaître une courbe sur rviz.

  Le but est d'écrire un ensemble de programmes qui transféreront les données d'un capteur GPS placé dans un véhicule vers le middleware ROS de façon à faire apparaitre la trajectoire suivie par ce véhicule sur rviz.

2 semaines : Analyse de l'utilisation de ROS dans le contexte des expérimentations de la sed : le middleware rempli-t-il les besoins en terme de :

- Possibilité de sauvegardes temps réel des mesures capteurs, facilité de portage Hugr vers ROS.

Cette partie sera consacrée à ma prise en main du middleware ROS. Les deux premières semaines de mon stage ont été axées sur la compréhension de l'environnement de programmation de ROS, de son protocole de communication, ainsi qu'à la découverte de son afficheur interactif 3D rviz.

Un middleware

Un middleware est un logiciel qui crée un réseau d'échange d'informations entre différents modules robotiques.

Un environnement de programmation robotique

ROS (Robot Operating System) est un système d'exploitation comme son nom l'indique. Cependant, il sera préférable de considérer ROS comme étant un environnement de programmation robotique. En effet, ROS s'éloigne de l'image que nous avons d'un système d'exploitation comme windows.

Fonctionnement ROS/Hugr

EMP : Espace Mémoire Partagée

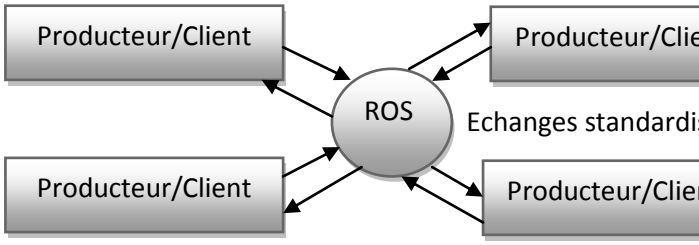


Figure 1 : Fonctionnement ROS

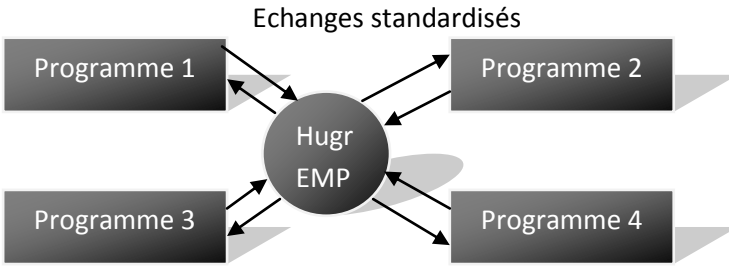


Figure 2 : Fonctionnement Hugr

Hugr est une application qui crée des variables dans un système de mémoire partagée afin de les rendre accessibles pour tous les programmes qui en auront besoin comme vous pouvez le voir sur le schéma conceptuel ci-dessus (Figure 2).

Il a l'avantage de s'affranchir des problèmes temps réel lorsque plusieurs processus insèrent des variables dans l'espace mémoire partagée.

Il permet un partage de données entre processus sur plusieurs plateformes. Par ailleurs, il permet de passer de la simulation au robot réel sans recompiler ce qui est pratique pour les phases de tests. Hugr communique avec les programmes par mémoire partagée alors que ROS lui communique avec ses programmes par réseaux. Pour les deux middleware, les échanges sont standardisés. Nous verrons sous quelles formes se présentent ces échanges un peu plus tard.

Les langages de programmation utilisés par ROS sont le langage C++ et le langage Python, deux langages de programmation orientés objets. Pour ma part, je n'ai utilisé que le langage C++, un langage qui reste tout de même procédural.

Dans la partie qui va suivre, nous aborderons la notion de paquet.

Paquets logiciels

ROS est un environnement de programmation robotique basé sur la notion de paquet. Chaque paquet possède un rôle différent et communique entre eux via ce qu'on appelle un démon.

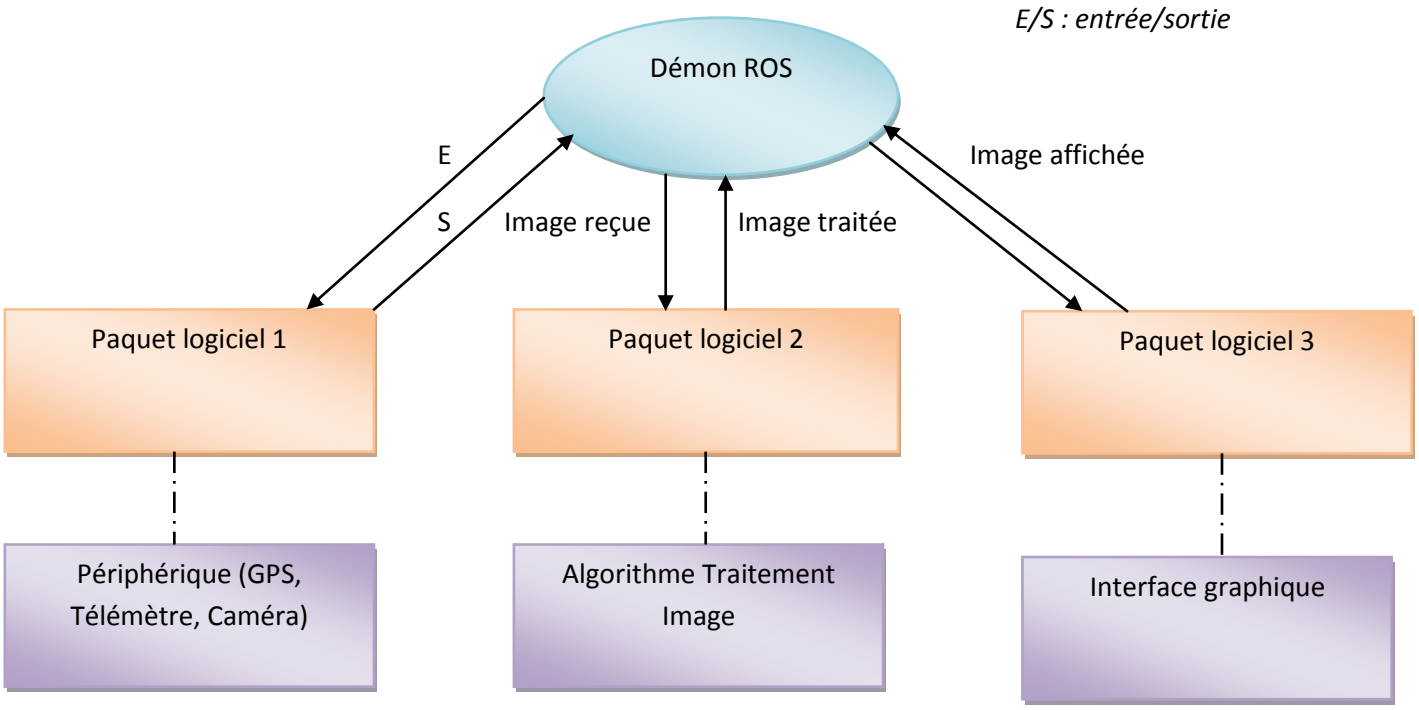


Figure 3 : Exemple d'application ROS

Le rôle du paquet P1 est de récupérer les données provenant d'un périphérique. Le paquet P2 récupère ces données via le ros démon et effectue un traitement sur l'image. Le paquet P3 récupère l'image traitée via le démon encore une fois et l'affiche sur une interface graphique.

Un paquet logiciel contient plusieurs éléments de programmation :

- Un code source en C++ ou python
- Un CMake caché qui s'apparente à un Makefile en plus complexe
- Un manifest, fichier xlm dans lequel sont précisées les dépendances que le paquet aura besoin.

En fait, les dépendances sont des librairies externes auxquelles un paquet peut avoir besoin. Lorsqu'on crée un paquet logiciel ROS, il faut lui préciser les dépendances qu'il aura besoin.

Nous nous intéresserons dans la partie suivante au système de communication qui relie les paquets avec le démon. Nous verrons comment les informations sont envoyées et reçues. En effet, ROS possède un protocole de communication qui lui est propre.

Des échanges standardisés

Le schéma qui suit permet de comprendre comment les informations sont envoyées et reçues par les paquets.

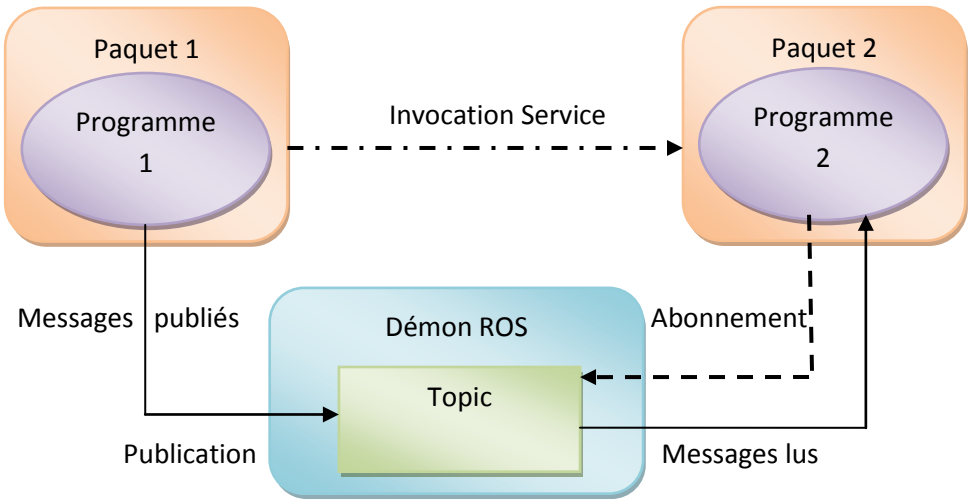


Figure 4 : Protocole de communication

Le principe est le suivant. Le programme 1 publie des messages sur ce qu'on appelle un topic. Le programme 2 s'abonne à ce topic pour pouvoir lire les messages publiés sur celui-ci.

Un topic c'est un « journal » sur lequel des messages sont publiés. Pour que son utilisateur puisse avoir accès à ce journal et donc aux messages qu'il recueille, il faut qu'il s'abonne. Le flux de données y est périodique.

De plus, on peut distinguer un deuxième chemin possible. Le client peut invoquer un service. Un service est définit dans un string (chaîne de caractère), dès lors qu'il est appelé par le client, une requête et une réponse s'effectue. Le flux de données est ponctuel.

La prochaine partie concernera les commandes ROS.

Commandes ROS

ROS possède ses propres commandes. Elles ressemblent grandement aux commandes linux avec le préfixe « ros ». Je vous donne quelques exemples de commandes de base que j'ai pu utiliser : **roscreate** pour créer un paquet ros, **rosmake** pour compiler un paquet, **rosrun** pour exécuter un paquet, **roscd** pour se déplacer d'un paquet à l'autre. Il existe d'autres commandes, mais je ne m'étalerai pas dessus, il faut juste en connaître l'existence. La partie suivante se concentrera sur les possibilités graphiques proposées par ROS.

Interfaces graphiques

Trouver une interface graphique permettant d'afficher ne serait-ce qu'une courbe faisait partie de mon travail. En effet, ROS met à disposition deux outils graphiques. Rxplot, un grapheur 2D/3D, et Rviz, un afficheur interactif 3D beaucoup plus puissant en termes d'affichage.

Grapheur 2D/3D

Rxplot est un grapheur 2D/3D. Son utilisation est basique et se limite à l'affichage d'une courbe.

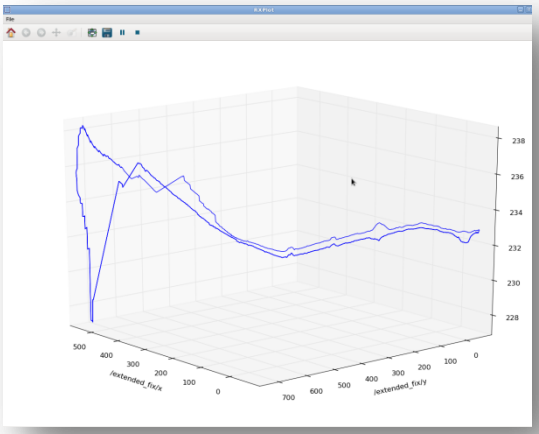


Figure 5 : Trajectoire de la Toyota Lexus

Afficheur interactif 3D

Rviz est un afficheur interactif 3D qui demande pas mal de ressources mémoires proportionnellement à ce que nous en faisons. Il permet d'afficher des formes diverses : sphère, cube, cylindre, point, ligne. Il a pour objectif de pouvoir visualiser et piloter des robots.

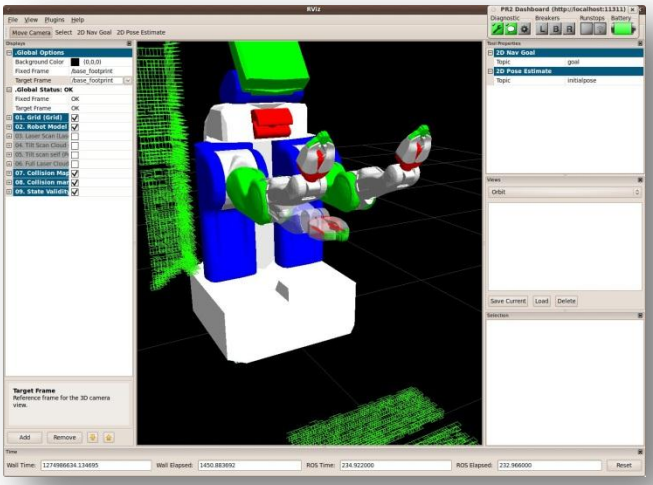


Figure 6 : Exemple d'application Rviz

Après avoir assimilé les concepts de ROS et découvert son environnement de programmation, mon objectif était de créer ma propre application ROS. Cette application ROS consiste à réaliser un ensemble de programmes qui auraient pour but de récupérer les données d'un capteur GPS, traiter ses données de façon à pouvoir afficher la courbe correspondante aux coordonnées reçues, sur l'afficheur interactif 3D rviz.

Démarche

Premièrement, je devais me familiariser avec le langage C++. Deuxièmement, il était nécessaire d'établir un plan avant de me lancer dans le code. C'est pourquoi j'ai établi un schéma de principe qui résume le travail que j'ai réalisé pendant mon stage.

Schéma de principe

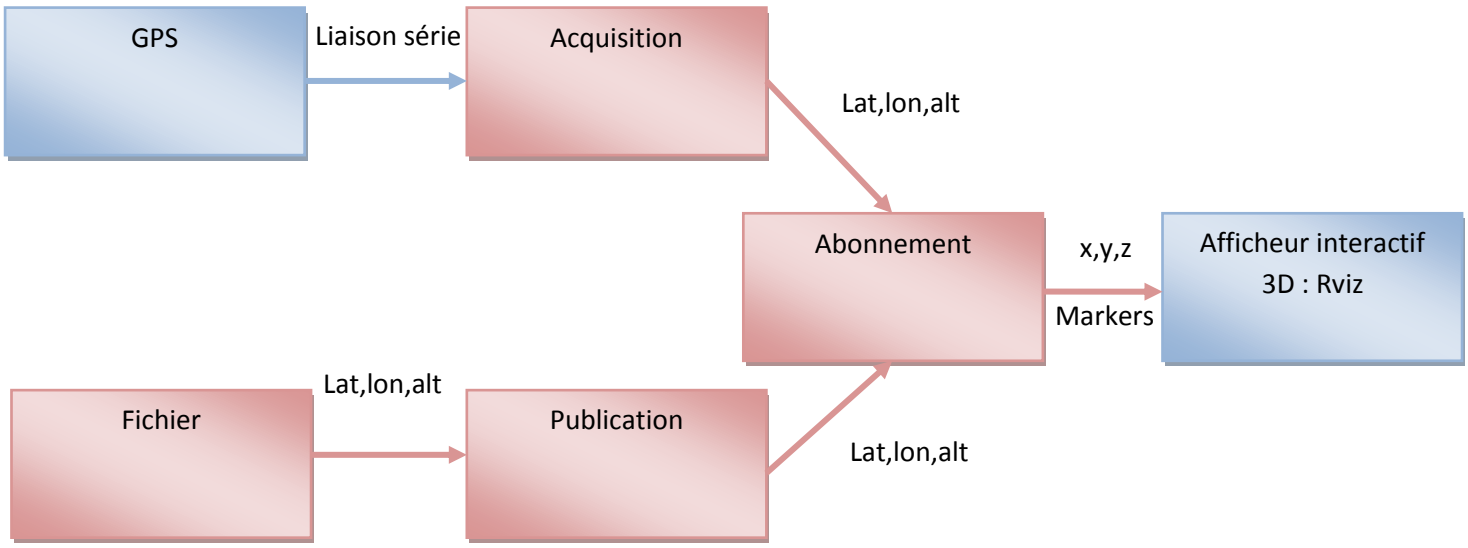


Figure 7 : Schéma de principe

On peut remarquer la présence de deux couleurs différentes. Les paquets bleus représentent ce qui m'a été fourni pour mon projet à savoir un vrai GPS, et l'interface graphique Rviz. Les paquets rouges sont les paquets que j'ai dû créer de moi-même dans son intégralité.

J'ai pu répartir mon travail en deux temps. Avant de traiter la partie acquisition de données GPS, je lisais dans un fichier .txt les coordonnées GPS latitude longitude altitude, et je publiais ces messages sur un topic de façon à ce que le programme contenu dans le paquet Abonnement puisse traiter ces données, à savoir convertir les coordonnées géographiques en coordonnées cartésiennes(j'ai dû faire des conversions UTM GeoPoint), produire des markers positions, vecteurs vitesse, et afficher ces éléments sur Rviz (voir signification marker annexe 2).

Une fois cette partie-là réalisée, je n'avais plus qu'à écrire un paquet acquisition qui récupère les données en sortie du GPS via une liaison série et publie ces données sur le topic sur lequel le paquet Abonnement, utilisé précédemment, s'abonne.

Messages typés

Dans ROS, les messages ont des types prédéfinis. C'est un travail que j'ai dû faire par moi-même, voir si pour les types d'informations que je voulais envoyer, un type prédéfini de message existait déjà. Par exemple, pour mes programmes, j'ai besoin de coordonnées géographiques, de coordonnées cartésiennes ainsi que de markers tels que des lignes et des points pour la courbe que je veux faire apparaître sur Rviz.

Voici l'exemple GeoPoint (figure 7) qui est le type de message prédéfini que j'ai utilisé pour les coordonnées géographiques que je récupère en sortie du GPS. J'ai inséré dans l'annexe les types de messages que j'ai utilisés.

```
geographic_msgs/GeoPoint.msg
float64 latitude
float64 longitude
float64 altitude
```

```
visualization_msgs/Marker.msg
uint8 POINTS=8
uint8 LINE_STRIP=4
uint8 LINE_LIST=5
```

Figure 7 : Message du type GeoPoint et Marker

Dans la partie qui va suivre, je consacrerai une petite partie pour expliquer le fonctionnement des deux programmes que j'ai créé.

Algorithme Traitement de données

Il s'agit du programme abonnement. Ce programme s'abonne au topic extended_fix sur lequel des messages de type GeoPoint (latitude,longitude,altitude) sont publiés. Après quoi, un traitement de conversion est effectué afin d'obtenir des coordonnées cartésiennes de façon à tracer ensuite la courbe sur rviz.

Algorithme Acquisition de données

Il s'agit du programme acquisition. Celui-ci a pour but de récupérer les trames de données à la sortie du GPS via un port série, cette trame de données se présente sur la forme suivante : latitude/longitude/altitude/***autres informations***

Ce programme publie les messages de type GeoPoint (Latitude, Longitude, Altitude) sur le topic /extended_fix, le topic sur lequel le programme abonnement s'abonne.

La trame GPS lue sur le port série est stockée dans un buffer_reception, il s'agit d'un tableau de caractère dont la taille adaptée aux trames envoyées par le GPS.

Les aspects communication temps réel

Le capteur GPS prend des mesures toutes les 100ms, dès lors qu'une mesure est faite, le programme d'acquisition publie le message sur le topic sur lequel le programme abonnement s'abonne de manière instantanée et affiche le point correspondant sur rviz.

Dans cette partie je ferais le bilan du travail que j'ai effectué, aussi bien les résultats que j'ai obtenus avec les phases de test de mes programmes mais aussi les difficultés rencontrées au cours de mon stage.

Résultats obtenus

A la fin de mon stage, mon maitre de stage m'a proposé de tester mes programmes dans la Lexus, un véhicule hybride prêtée à l'INRIA dans le cadre du projet Arosdyn d'assistance à la conduite automobile. La Lexus est équipée d'un ordinateur embarquée sur la banquette arrière du véhicule ce qui me permet de compiler et exécuter mes programmes pendant que la Lexus roule.

D'autre part, nous avons placé à l'arrière également un GPS Z-Max à précision centimétrique qui est relié via un port série à l'ordinateur embarqué Arosdyn ainsi qu'à une antenne GPS fixé sur le toit du véhicule.

Figure 8 : Ordinateur embarqué Arosdyn

Figure 9 : Antenne GPS

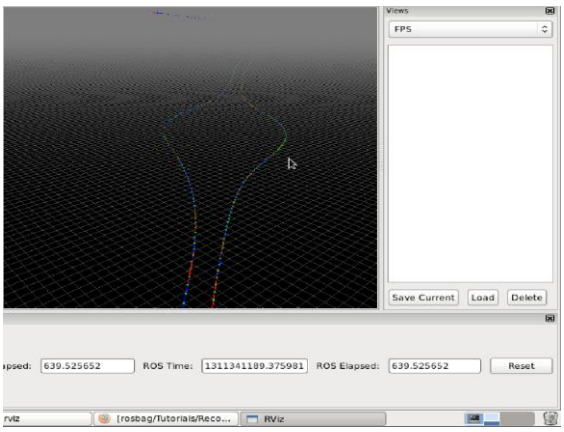


Figure 10 : Trajectoire Lexus, rond-point

Les résultats ont été concluants, j'affiche la trajectoire de la Lexus sur l'afficheur interactif 3D Rviz comme le montre l'illustration ci-des

Problèmes rencontrées

Au cours de mon stage j'ai été bloqué plusieurs fois, certaines parties étaient plus difficiles à mon goût que d'autres. J'ai trouvé les deux premières semaines de mon stage assez confortables, la prise en main du middleware ROS n'était pas fastidieuse car de nombreux tutoriels étaient mis à ma disposition.

Là où j'ai le plus peiné, c'est sur la partie développement car je n'avais aucune expérience avec le langage C++. Il m'a fallu plusieurs jours pour ne pas me mélanger les pinceaux avec les concepts de ROS (paquets, message typés) et les concepts propres au langage C++ (objet, classe, méthode). Par ailleurs, je vais vous présenter quelques erreurs ou problèmes que j'ai rencontrés lors des phases de test et pendant la compilation.

Pertes satellites

Nous avons réalisé plusieurs circuits avec la voiture pour tester mon programme. Nous avons tout d'abord effectué le tour du parking de l'inria, puis nous sommes partis un peu plus loin dans la zone industrielle de Montbonnot.

La première anomalie que j'ai remarqué, c'est que nous avions pris 2 mètres de hauteur en plus à l'arrivée par rapport au point de départ comme on peut le voir sur l'image qui suit.

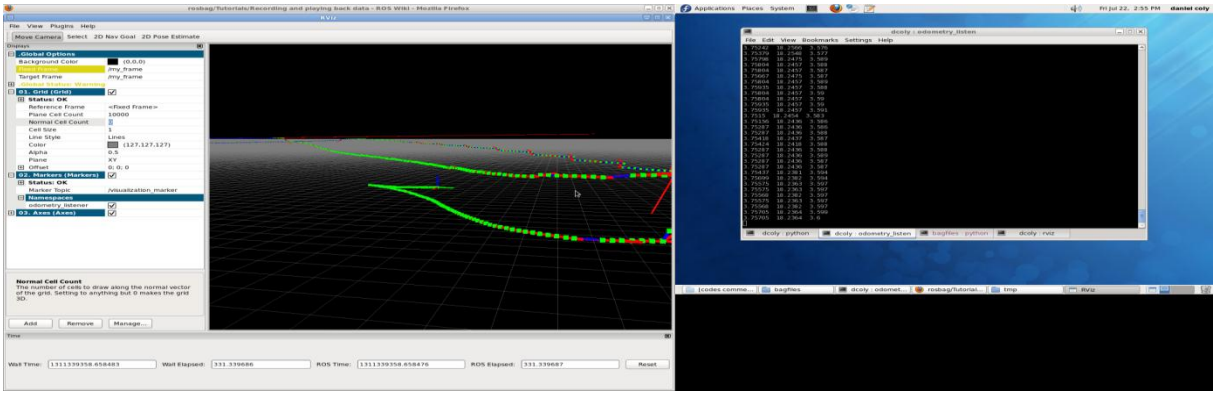


Figure 11 : Essai avec la Lexus le parking de l'INRIA

Cela s'explique par le fait qu'à un moment donné, le GPS ne recevait plus de satellites car j'ai vu que le voyant correspondant au nombre de satellites qu'il est capable de voir était rouge.

Lorsqu'on a fait le tour du parking, il y a eu une portion le long du bâtiment pour laquelle l'antenne de la lexus n'avait pas un champ de vision assez grand pour capter des satellites. Le GPS a perdu sa précision centimétrique, ce qui explique l'erreur en fin de parcours.

L'essai dans la zone industrielle a été plus concluant, l'angle de réception étant beaucoup plus dégagé. Nous avons emprunté une route à double sens de façon pouvoir valider la précision du GPS, du moins nous avons pu vérifier qu'il y avait bien environ 2 mètres qui séparés les deux trajectoires.

En fait, l'échelle est fixée par une grille qui peut être rajoutée dans rviz, les côtés de chaque case formant la grille correspondent à 1 mètre de longueur (voir figure 10).

Les dépendances

J'ai rencontré de nombreux problèmes avec les dépendances des paquets. Il faut bien faire attention dès lors qu'on crée des paquets à spécifier les dépendances qu'il aura besoin sans quoi une erreur de compilation apparaitra.

Distribution linux

La distribution linux que j'ai utilisé est Fedora, malheureusement c'est l'une des distributions linux la moins bien packagé pour ros. J'ai dû importer énormément de paquets linux notamment pour installer rviz qui a besoin de beaucoup de ressources graphiques (librairies).

Sauvegardes des données capteurs en temps réel

La sauvegarde en temps réel des données capteurs se fait dans des fichiers .bag qui enregistrent tous les messages publiés sur un topic au moyen de la commande **rosbag record name_topic**. En plus de cela, ros permet à ses utilisateurs de simuler l'envoi des messages sur le topic par la commande **rosbag play name_topic**.

C'est d'ailleurs ce que j'ai fait lorsque j'ai réalisé les tests de mes programmes sur la Lexus. Pour avoir accès à toutes les données qui ont été envoyées par le GPS, j'ai créé un fichier .bag de façon à pouvoir simuler cette publication de messages sur le topic, cela m'a permis de réaliser une vidéo de mon programme qui tourne sans pour autant réaliser la vidéo pendant la phase de test ce qui aurait été laborieux. De ce fait je peux révisionner l'affichage de la trajectoire de la Lexus en temps réel à tout moment.

Par ailleurs, ROS dispose d'un outil performant qui n'est pas présent dans Hugr. Il s'agit de Rxbag, un outil permettant de rejouer des séquences de publications. Voici un aperçu sur l'image qui va suivre :

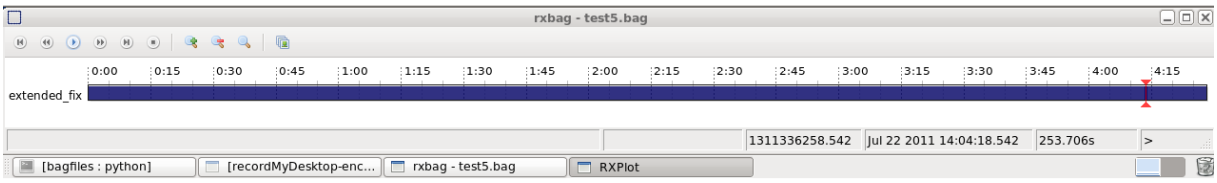


Figure 12 : Rxbag séquence de publication

Facilité de portage Hugr vers ROS

ROS reste un environnement de programmation robotique avec de nombreux paquets à installer. Passer de Hugr vers ROS n'est pas en soi déroutant, les concepts restent les mêmes, en revanche les noms changent. Néanmoins ROS n'est pas plus facile d'accès, il reste assez complexe avec les notions de dépendances, de paquets, etc.

On trouve deux grandes familles d'ingénieurs à l'INRIA à savoir ingénieur SED (Service Expérimentation et Développement) et ingénieur MI (Moyens Informatiques). J'ai pu fixer un rendez-vous avec Frédéric Saint-Marcel, ingénieur MI à l'INRIA.

Dans les deux prochains paragraphes, je développerai les aspects du métier d'ingénieur SED et MI et montrerai en quoi ils sont différents.

Ingénieur SED

L'INRIA est un institut de recherche, le principal atout qu'ils vont rechercher chez un candidat postulant chez eux est le contact avec le « scientifique ». La majorité des résidents du centre ont fait une thèse ou ont déjà travaillé avec des équipes de chercheurs. C'est une réalité d'INRIA, les ingénieurs sont au contact des équipes de recherche.

Nicolas TURRO, mon maitre de stage, est un ingénieur SED. Il développe des outils matériels et logiciels pour faire en sorte que les expérimentations des chercheurs se passent bien. On comprend que l'aspect « développement » est très présent, chose qu'on trouve quasiment pas dans le métier d'ingénieur MI. Les expérimentations qu'ils mettent en œuvre sont couteuses en temps d'ingénieur.

En deux mots, un ingénieur SED fait du développement expérimental et de la gestion de plateforme. Je dirai que son profil est celui d'un ingénieur recherche. Typiquement, il n'est pas rare de retrouver leurs noms dans des publications ou des conférences.

Leur domaine d'activité est large, la robotique étant un domaine qui recouvre de nombreux domaines tels que l'informatique, l'électronique, l'automatique, la vision par ordinateur, l'intelligence artificielle. Leurs domaines d'expertises sont proches des scientifiques : réseau, traitement d'images, calcul distribué.

Les activités des ingénieurs SED sont diverses et variées. Ils lisent des ouvrages scientifiques et techniques, assistent à des réunions scientifiques avec les chercheurs. Par ailleurs, il y a une spécialisation des ingénieurs SED par compétences scientifiques.

La principale chose à retenir concernant ce métier et qui forme en soi une nécessité pour intégrer l'INRIA est «le contact du scientifique » pour reprendre les dires de Frédéric St Marcel.

Ingénieur MI

Frédéric Saint-Marcel est un ingénieur MI (Moyens Informatiques) et comme tous ingénieurs MI, son rôle est de donner un service informatique pour tous les utilisateurs, aussi bien les chercheurs, que les administratifs et c'est bien là l'une des différences avec le métier d'ingénieur SED.

Autrement, leur domaine d'activité est centré sur l'informatique des systèmes et réseaux. En revanche, l'aspect développement est beaucoup moins présent que chez l'ingénieur SED, mais ça peut arriver qu'ils aient à faire du développement logiciel.

Leur rôle est de fournir des ressources informatiques pour que les gens puissent travailler, leur activité se décline selon 7 axes :

Réseaux : Déployer des solutions informatiques pour que le réseau fonctionne (Switch, baies informatiques, routeurs, téléphonie, DNS (donner un nom à chaque machine du centre), VPN, Wifi)

Système : gestion de serveurs, fournir des services de type web, mail, partage de documents

Ingénierie développement : Système de facturation, sous-traitance

Support aux utilisateurs : Aide sur des problèmes matériels

Confidentialité, sécurité

Intégrité des données (stockage)

Techniques de « Haute disponibilité », « Reprise d'erreur » (fail over), redondances à mettre en place pour assuer une bonne qualité de service. A contrario des plateformes expérimentales SED qui peuvent être non fonctionnelles sur de longues durées.

Par ailleurs, il y a une spécialisation des ingénieurs MI par domaines techniques et non pas compétences scientifiques comme pour les ingénieurs SED. D'autre part, il existe une infrastructure nationale, une coordination nationale entre les MIs (pratiquement absent chez les SED).

Pour résumé le tout, le métier d'ingénieur MI ne se situe ni dans le scientifique, ni dans l'expérimental, mais dans la production. Ce qu'on recherche, c'est surtout la qualité du service fourni. On peut retrouver ce service dans de nombreuses entreprises ayant besoin d'un service informatique alors qu'on ne retrouve pas de profil type ingénieur SED dans ces entreprises là, mais plutôt dans des centres de recherches.

Frédéric St Marcel a décidé d'intégrer l'institut pour justement se rapprocher du domaine de la recherche.

Ces deux métiers d'ingénieurs sont accessibles par la voie d'un concours publique.

Conclusion

Hugr et ROS sont deux middleware conceptuellement identiques. Leurs différences se situent au niveau de leur système de communication où ROS se caractérisera par une communication par réseaux tandis qu'Hugr communique par mémoire partagée.

Toutefois, les capacités graphiques de ROS restent très puissantes et répond à cette problématique en termes d'affichage et pilotage des expérimentations de la sed. Les capacités de Rviz sont immenses, pendant ce stage, je n'ai pu aborder qu'une fine partie. Cela m'a suffi pour dire que ROS offre les mêmes fonctionnalités que Hugr et fait parfois même meilleur que Hugr notamment au niveau de la sauvegarde des données capteurs en temps réel. ROS met à disposition l'outil Rxbag qui permet de rejouer des séquences de publication qu'on peut définir, ce qui n'est pas possible avec Hugr.

Je garderai un très bon souvenir de ce stage, l'ambiance y été chaleureuse et conviviale. J'ai appris un nouveau langage de programmation à savoir le langage C++, j'ai renforcé mes connaissances linux. Je suis passé par toutes les étapes d'un projet, de la recherche de la problématique à la réalisation de mes programmes pour enfin aboutir aux essais. Ce stage m'a beaucoup apporté au niveau des compétences techniques. Il m'a renforcé sur l'idée de devenir ingénieur informatique, et pourquoi pas dans le domaine de la robotique.

Remerciements

Je tiens à remercier tous les membres du service SED de l'INRIA Rhône-Alpes et plus particulièrement Bendehiba BOUKSARA qui m'a permis de réaliser mon stage à l'INRIA, ainsi que Nicolas TURRO, mon maitre de stage pour son aide, sa disponibilité et son accueil au sein du service SED.

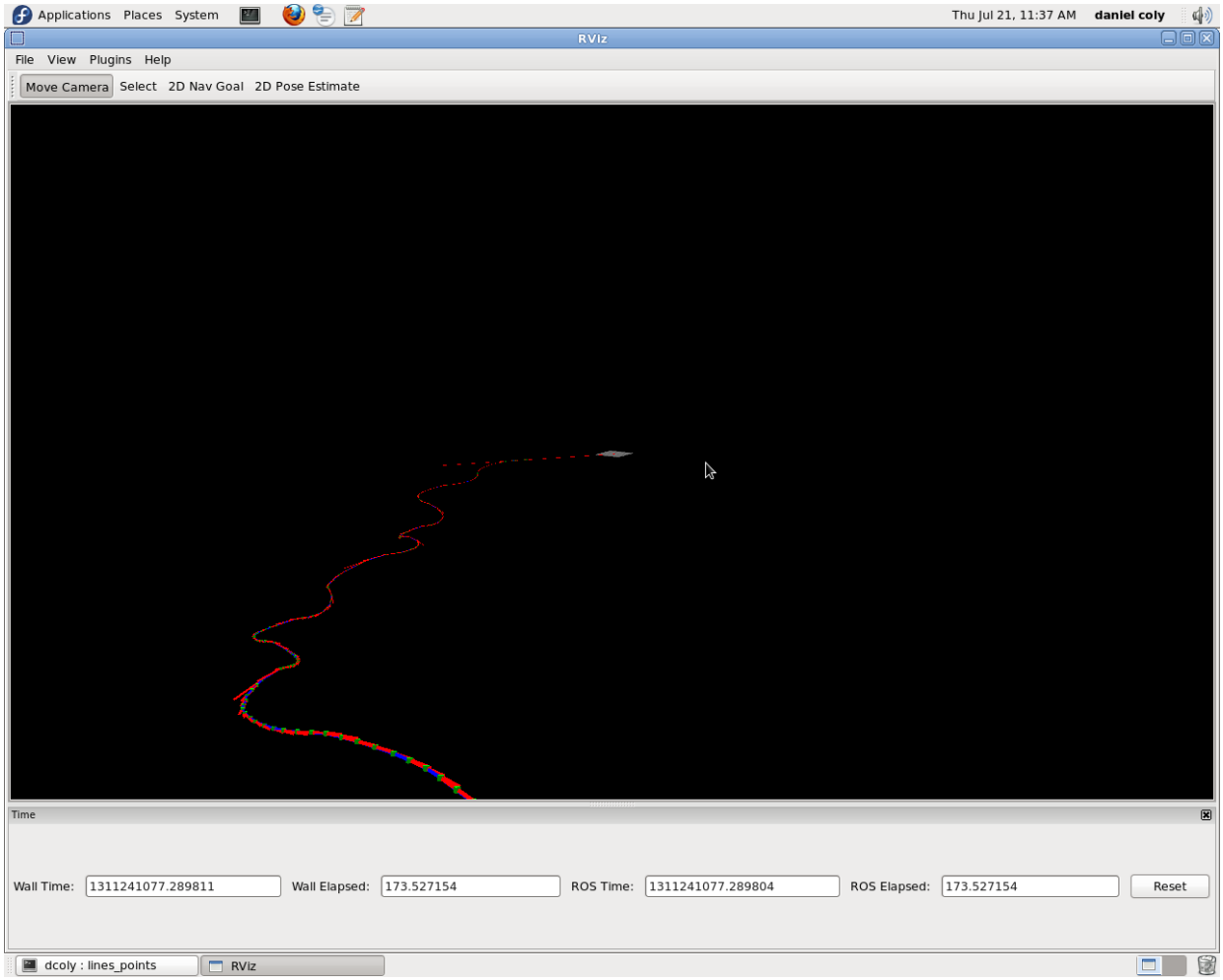
Je tiens à remercier également Jean-François CUNIBERTO pour l'installation de l'antenne GPS sur la Lexus qui m'a permis de tester mes programmes dans la Lexus.

Je remercie Frédéric SAINT MARCEL pour m'avoir consacré une petite heure d'entretien pour parler des métiers d'ingénieurs SED et MI.

Enfin, je remercie également Cyril LAMINE, étudiant en master 2 Maths Info à l'université Joseph Fourier pour sa précieuse aide concernant le langage C++.

Je remercie mon tuteur de stage pour son accompagnement tout au long de mon stage ainsi que toute l'équipe pédagogique de l'ESISAR qui m'ont permis de réaliser un stage technicien dans mon cursus préparatoire.

Annexe 1



Cette illustration est issue d'un programme que j'ai réalisé qui lisait les coordonnées x y z dans un fichier et qui affichait la courbe correspondante sur Rviz. Il s'agissait de coordonnées issues d'un skieur parcourant une piste. On peut y voir les markers positions suivants :

En vert, les points, en bleu, les lignes reliant deux points, et en rouge, les vecteurs vitesses. Pour réaliser ce vecteur vitesse, j'ai créé un algorithme qui donne la tangente en chaque point de la courbe. Ce n'est pas une méthode très précise, mais convenable pour le temps que je disposais.

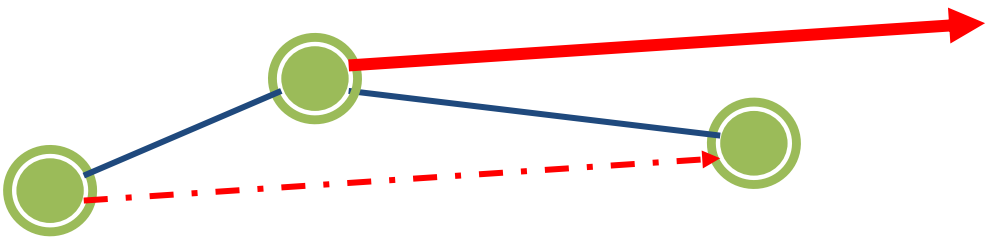


Schéma explicatif

Annexe 2

Message type Point de la classe geometry_msgs pour les coordonnées cartésiennes

File: geometry_msgs/Point.msg

```
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

Expanded Definition

```
float64 x
float64 y
float64 z
```

Message type UTM Point de la classe geodesy pour les coordonnées UTM

Public Member Functions

UTMPoint (double _easting, double _northing, double _altitude, uint8_t _zone, char _band)
UTMPoint (double _easting, double _northing, uint8_t _zone, char _band) UTMPoint (const geographic_msgs::GeoPoint &pt) UTMPoint (const UTMPoint &that) UTMPoint ()

Public Attributes

double altitude altitude above ellipsoid [meters] or NaN
char band MGRS latitude band letter.
double easting easting within grid zone [meters]
double northing northing within grid zone [meters]
uint8_t zone UTM longitude zone number.