Rapport de projet tuteuré

Robot explorateur

Etudiants : Jordan MARTIN & Jonathan TAWS
Tuteur : Michael MRISSA

IUT Lyon 1
l'excellence technologique

Remerciements

Nous tenons à remercier notre tuteur M. MRISSA, Maître de conférences, Professeur à l'université Lyon 1 et membre de l'équipe SOC au LIRIS[1], qui nous a à la fois guidés tout au long de notre projet et aidés à faire les bons choix. M. MRISSA nous a aussi permis d'inscrire notre travail dans un projet de grande ampleur.

Nous remercions également M. MEDINI, Maître de conférences, Professeur à l'université Lyon 1 et membre de l'équipe DRIM au LIRIS, qui nous a accompagné dans l'élaboration de l'architecture « ASAWoO ».

Nous remercions M. GUILLOU, Maître de conférences, Professeur à l'université Lyon1 et membre de l'équipe SAARA au LIRIS, pour son aide dans le montage de notre robot explorateur et pour le prêt des deux robots utilisés dans le projet.

Par ailleurs, nous remercions M. JALOUX, Professeur de mathématiques à l'université Lyon 1, pour son aide dans la résolution de problèmes mathématiques lié à notre algorithme de cartographie.

Nous remercions la Faculté des Sciences et Technologies de l'Université Lyon 1, l'université Lyon 1, et la région Rhône-Alpes.

Nous tenons enfin à remercier tout le corps enseignant de l'IUT Informatique Lyon 1 pour nous avoir donné l'opportunité de travailler en équipe sur un projet enrichissant à tous les niveaux, ainsi que pour son soutien tout particulier envers notre projet.

[1] Laboratoire d'InfoRmatique en Image et Système d'information situé sur le campus de la Doua à Villeurbanne.

Table des matières

1. Introduction 3
2. Présentation 4
   2.1. Présentation du projet 4
      2.1.1. Notre projet 4
      2.1.2. Le projet ASAWoO 5
      2.1.3. Nos missions 5
   2.2. Présentation de l'environnement technique 6
      2.2.1. Logiciels et outils utilisés 6
      2.2.2. Matériel utilisé 7
      2.2.3. Technologies utilisées 8
      2.2.4. Langages de programmation utilisés 9
3. Réalisation du projet 10
   3.1. Chronologie / Les grandes étapes 10
   3.2. Découpage du projet 11
   3.3. Fonctionnement général du projet 11
   3.4. Architecture ASAWoO 12
   3.5. Réécriture de l'API C# en Java 13
   3.6. Généricité : utilisation de plusieurs robots 14
   3.7. Le serveur Jetty 15
      3.7.1. Les Servlets 15
      3.7.2. Intégration dans le projet 16
   3.8. Live streaming 17
   3.9. Interface utilisateur 17
   3.10. Algorithme de cartographie autonome dans un environnement inconnu 19
4. Bilan personnel 21
   4.1. Jordan 21
   4.2. Jonathan 21
5. Conclusion 23
6. Glossaire 24
7. Annexes 25
   7.1. Comparatif diagrammes de Gantt 25
   7.2. Interfaces utilisateur 26
   7.3. Résultats obtenus 28
   7.4. Innorobo 28

1. Introduction

Lors de notre deuxième année à l'I.U.T. Informatique de Lyon, nous - Jordan MARTIN, Jonathan TAWS, et Nicolas LEBRUN - avons pu mettre en œuvre les connaissances acquises depuis le début de notre formation dans un projet de groupe supervisé par un tuteur pédagogique. Ce projet s'est déroulé sur une période d'environ six mois, de début Octobre 2013 à début Mars 2014, permettant une réelle montée en compétences. Les projets sont généralement proposés soit par nos enseignants, soit par des entreprises, et doivent répondre à un cahier des charges. Cependant, certains projets sont aussi à l'initiative des étudiants.

Ayant réfléchi à un sujet novateur, nous avons décidé de proposer notre propre idée de sujet à un tuteur potentiel, M. MRISSA, qui a accepté notre projet.

Notre idée était de créer un robot explorateur afin d'examiner un environnement de manière autonome. Il s'agissait de développer une architecture communicante entre un robot et un site web, qui permet de faire communiquer un robot avec Internet en lui envoyant des ordres et en récupérant les différentes données qu'il recueille de son environnement. Cette architecture met en place un concept dont on a beaucoup entendu parler ces derniers temps : le « Web des Objets », qui correspond à l'intégration de tout objet physique interrogeable ou contrôlable dans la sphère Internet. Dans notre cas, nous interrogeons et contrôlons le robot à travers notre interface web. Notre projet s'inscrit dans un projet de recherche intitulé « ASAWoO », que nous expliquerons plus en détails plus tard.

Nous avons également réalisé une interface web proposant différentes fonctionnalités telles que le pilotage manuel du robot, la visualisation en temps réel de l'environnement via une cartographie 2D et un flux vidéo provenant d'une caméra embarquée.

Ce projet pourrait servir par exemple à récolter des informations sur la structure d'un lieu difficile d'accès. On pourrait imaginer l'utilisation combinée d'une caméra et de la cartographie pour se rendre compte de l'état d'une pièce sans avoir à y rentrer. Dans le cas du projet « ASAWoO », notre robot explorateur pourrait servir à fournir des cartes d'environnement toutes prêtes à d'autres robots qui n'auront plus qu'à se déplacer en fonction des obstacles sur la carte. Si, par exemple, un robot doit apporter un objet dans un environnement inconnu, grâce à la carte établie par notre robot explorateur, il pourra déterminer son trajet facilement et arriver à bon port sans avoir à analyser tout l'environnement.

Afin de s'inscrire dans le projet de recherche, nous avons proposé une première implémentation de l'architecture « ASAWoO » appliquée à notre robot explorateur, qui servira, entre autre, à montrer les possibilités offertes par ce type d'architecture.

Jordan MARTIN avait déjà travaillé durant l'été 2013 sur une première ébauche d'un système permettant de contrôler à distance un robot à travers un site web. Nous avons donc repris les fondements de ce travail pour construire les bases de notre projet.

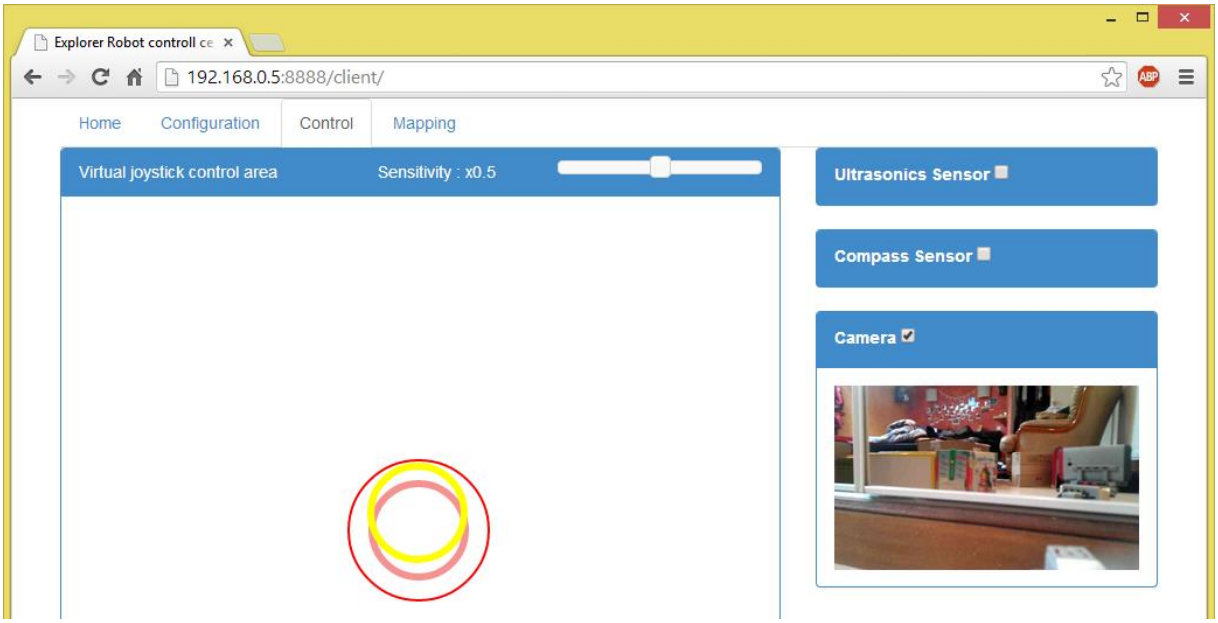
2. Présentation

2.1. Présentation du projet

2.1.1. Notre projet

Le sujet de notre projet a pour intitulé « Réalisation d'une architecture communicante entre un robot et une interface web permettant, entre autres, la cartographie d'un environnement ». Ce projet met en symbiose la communication entre trois composants pour obtenir un affichage en temps réel sur une interface web : le robot, la passerelle, et l'interface web.

La seule partie visible du projet pour l'utilisateur final est l'interface web, à partir de laquelle on peut contrôler le robot explorateur grâce aux différentes fonctionnalités proposées. Cette interface propose également l'affichage en direct du flux vidéo de la caméra embarquée sur le robot.



Une partie de l'interface web

Le site Internet est la seule partie visible pour le client ; derrière cette interface de nombreux composants travaillent de concert pour acheminer les commandes au robot et ensuite faire remonter les informations jusqu'à l'utilisateur. Une réelle architecture communicante a été mise en place afin de pouvoir échanger avec le robot sur un site Internet, et ce sans se soucier de la localisation du client qui utilise le site. Nous avons choisi la technologie « Bluetooth » pour communiquer par ordinateur avec le robot. Cet ordinateur représente la passerelle, sur laquelle un serveur a été mis en place pour héberger le site Internet et gérer les requêtes de l'utilisateur. Par exemple, si l'utilisateur décide d'enclencher le mode pilotage manuel, nous allons traiter chaque mouvement du « joystick » comme un message qui sera envoyé au serveur et qui, à son tour, traduira le message en une commande envoyée au robot, lequel l'exécutera sur le champ.

La passerelle permet donc de communiquer à la fois avec le robot et avec l'interface web. Sur cette passerelle se trouve la majeure partie des composants développés tout au long du projet. Parmi ces composants se trouvent l'algorithme de cartographie, encapsulé dans l'architecture « ASAWoO », mais aussi le service de « live streaming[2] », et surtout tout le processus de gestion des requêtes venant des clients sur l'interface web. Nous verrons plus loin comment se déroule cette communication. Le

[2] Envoi d'un contenu audio ou vidéo « en direct » en continu et à mesure qu'il est diffusé. Grâce à ce système, des utilisateurs peuvent suivre en direct un contenu proposé par un émetteur, généralement à travers Internet.

gestionnaire s'occupe d'aiguiller les requêtes en appelant les composants appropriés en fonction du type et du contenu de la requête, les composants appropriés. Si l'utilisateur veut lancer la cartographie, une requête sera envoyée de l'interface web jusqu'au serveur qui instanciera l'algorithme de cartographie.

2.1.2. Le projet ASAWoO

« ASAWoO », acronyme pour Adaptive Supervision of Avatar/Object Links for the Web of Objects, est un projet de recherche financé par l'Agence Nationale de Recherche (ANR) dans le contexte du programme INFRA[3] de l'ANR. Ce projet a pour objectif d'améliorer la pénétration et l'intégration des objets dans le Web. Il propose une architecture orientée fonctionnalités appliquée au « Web des Objets », en offrant un processus de collaboration entre des objets physiques hétérogènes. De nombreuses entités nationales se sont liées par partenariat à ce projet.

Le coordinateur de ce projet est M. MRISSA, professeur de l'université Lyon 1 et membre de l'équipe SOC du laboratoire LIRIS.

En quelques mots, le projet « ASAWoO » propose une architecture liant un avatar à Internet, cet avatar pouvant être un robot, un objet doté de capteurs, etc. L'avatar est une représentation simplifiée de l'objet qui a pour but de mettre en avant ce que sait faire cet avatar, à travers ses capacités et ses fonctionnalités.

Dans le cas de notre projet, nous avons implémenté une partie de l'architecture « ASAWoO » sur le serveur pour faire appel aux fonctionnalités de notre avatar, le robot explorateur. En participant à ce projet, nous avions pour mission de proposer une première implémentation de cette architecture dans le cas de notre projet de robot explorateur. Nous avons donc dû réfléchir à l'intégration en langage Java des concepts proposés par cette architecture et à l'adaptation de notre travail dans cette architecture.

2.1.3. Nos missions

Le but de ce projet était initialement de développer un processus de cartographie à l'aide d'un robot « explorateur » LEGO Mindstorms NXT dont le résultat pourrait être suivi sur une interface web. Nous avons eu la chance de pouvoir contribuer au projet de recherche « ASAWoO », et avons donc adapté notre travail pour mettre en œuvre certains concepts de ce projet. Cela revenait surtout à adopter et proposer un exemple du découpage logique des classes et de structurer en conséquence la communication entre ces classes. Le but de notre projet n'a donc pas été fondamentalement modifié suite à ce changement ; nous avons simplement dû réfléchir à l'implémentation de ces concepts dans l'architecture existante du projet.

Les missions :

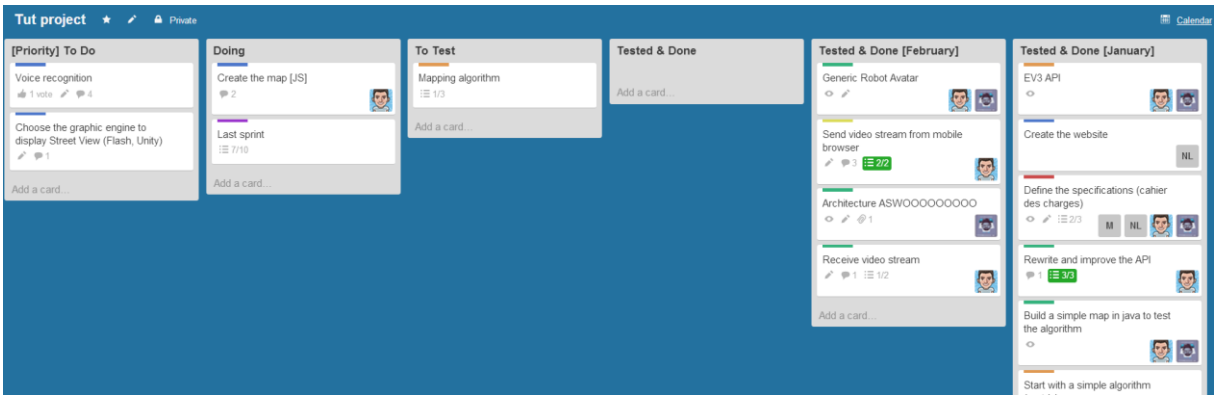
- ✓ Développer une architecture de communication entre un avatar et Internet
- ✓ Proposer une première implémentation du concept d'architecture ASAWoO
- ✓ Illustrer ce concept à travers des fonctionnalités, dont la cartographie
- ✓ Elaborer un algorithme de cartographie autonome dans un environnement inconnu
- ✓ Visionner le flux d'une caméra embarquée sur le robot sur l'interface web (live streaming)

[3] Le programme INFRA est un appel à projet de l'ANR qui a pour vocation de susciter et soutenir la recherche dans des domaines stratégiques que sont les infrastructures matérielles et logicielles pour l'internet du futur. Le projet ASAWoO a été choisi pour faire partie de ce programme.

2.2. Présentation de l'environnement technique

2.2.1. Logiciels et outils utilisés

Afin d'organiser le projet, nous avons opté pour une démarche agile du type « kanban », qui nous permette d'avoir un tableau de bord interactif proposé par le site Trello. Son système repose sur des cartes auxquelles on affecte des acteurs, que l'on peut assimiler à une tâche avec une échéance. Ainsi, sur la même page, nous pouvons savoir ce qu'il reste à faire, ce que nous devons tester, et ce que nous avons fait jusqu'à présent.



Aperçu du tableau de bord de notre projet sur Trello

Trello propose également un système de label de couleur pour savoir facilement à quel ensemble (dans notre cas le serveur, le client, et le robot) appartient la tâche.

Nous avons décidé de recourir à un autre logiciel pour le planning de notre projet. Alors que Trello propose une vision spatiale des tâches, Gantt project permet d'avoir une vue d'ensemble avec une vision temporelle sur les différentes tâches du projet.

GitHub est un service web d'hébergement de projet reposant sur le programme Git, un logiciel spécialisé dans la gestion de versions. GitHub permet de stocker le projet en ligne en gardant une chronologie des modifications réalisées, et garantit la synchronisation des fichiers à travers tout le projet. Cette plateforme nous a permis de pouvoir travailler en simultané sur le projet et de garantir la sauvegarde et la mise à disposition de nos fichiers. Afin d'avoir une interface plus plaisante que le terminal pour envoyer ou recevoir des modifications, Jonathan TAWS et Nicolas LEBRUN ont utilisé le logiciel GitHub (fourni par la plateforme du même nom), très simple d'utilisation, alors que Jordan MARTIN a préféré utiliser Git GUI, plus adapté pour la résolution des éventuels conflits entre versions des fichiers.

Nous avons essentiellement utilisé l'EDI Netbeans dans ses versions 7.3 et 7.4, nous permettant d'avoir une intégration avec Git pour la gestion des fichiers et des fonctionnalités avancées très intéressantes pour la programmation en Java telles qu'une navigation simplifiée dans les classes, l'auto-complétion du code, la génération automatique du code, le refactoring et « Matisse », l'outil graphique de Netbeans permettant de créer très rapidement des interfaces graphiques.

2.2.2. Matériel utilisé

Nous avons utilisé deux robots pour ce projet :

Un robot de la génération *Lego Mindstorms NXT* sortie en 2009. Ce dernier offre la possibilité de faire des montages très complexes et adaptés à n'importe quel besoin. Il est basé sur un processeur 32-bits de type ARM-7, et embarque également la technologie Bluetooth pour permettre la communication sans fil entre un ordinateur et le NXT.

Un robot de la dernière génération de robot proposée par Lego qui est sortie en Septembre 2013, sous le nom de Lego Mindstorms EV3. La différence majeure par rapport à la génération précédente réside dans l'architecture de son processeur qui passe d'ARM-7 à ARM-9[4]. Ce robot propose également des capteurs plus précis, ainsi qu'un port USB et un port SD. Le port USB permet, entre autres, la communication sur le Wifi via un dongle[5] adapté.

Nous avions choisi au début du projet de nous concentrer exclusivement sur des robots de la génération NXT, étant donné que les robots de dernière génération EV3 venaient tout juste de sortir et qu'aucune API[6] n'était proposée. De plus, l'API proposée pour le NXT était stable et très complète, ce qui nous permettait de développer des applications sans avoir à nous soucier d'éventuels problèmes liés au fonctionnement du robot.

Cependant, nous avons décidé, un peu plus tard dans le projet, de procéder à la réécriture en langage Java d'une API pour l'EV3 codée en C# qui venait tout juste d'être mise en ligne. Ainsi, notre projet a gagné en généricité : nous pouvons maintenant utiliser soit un NXT, soit un EV3, sans avoir à modifier drastiquement l'intégralité du code.

Pour ces deux robots, nous avons adapté le montage aux besoins du projet : pouvoir cartographier une salle, c'est-à-dire sonder les obstacles environnants, tout en filmant ce que « voit » le robot (caméra embarquée).

Pour gagner en efficacité, nous avons choisi de mettre trois capteurs qui ont pour mission de nous signaler tout obstacle venant de devant, de droite, ou de gauche du robot. En effet, nous pouvons ainsi cartographier simultanément dans trois directions mais aussi facilement choisir dans quelle direction aller lorsque la trajectoire du robot rencontre un obstacle. Nous utilisons également un gyroscope pour connaître à tout moment l'orientation du robot.

Nous avons également prévu un socle stable pour pouvoir mettre un téléphone qui a pour fonction d'envoyer à l'interface web ce que voit le robot. Pour cela, nous avons testé différents montages avec des chenilles à la place des roues pour gagner en stabilité. Mais finalement, le montage qui s'est révélé le plus adéquat fut le suivant : deux roues placées de part et d'autre du robot, centrées sur le centre de gravité, avec un « ball caster » derrière le robot, qui est une petite bille qui fait office de roue arrière et qui a la particularité de pouvoir se déplacer librement dans n'importe quelle direction.

[4] Processeurs doté d'une architecture RISC 32 bits ou 64 bits développés par la société ARM. Ces processeurs sont très populaires dans l'informatique embarquée pour leur faible consommation et leur simplicité d'utilisation.

[5] Composant matériel qui se branche sur un port d'entrées-sorties d'un ordinateur et qui peut permettre, selon son type, de proposer du stockage (clé USB), un accès à un réseau WiFi, Bluetooth, etc.

[6] Abréviation pour Application Programming Interface, ce terme est utilisé pour décrire un ensemble de classes et méthodes qui sont à disposition pour les programmeurs et visent à simplifier l'utilisation d'un composant.

2.2.3. Technologies utilisées

Nous avons utilisé les WebSockets[7] afin de pouvoir envoyer des informations entre l'interface web et le serveur, et entre le téléphone et le serveur. Les WebSockets représentent un tube de communication en full-duplex (bidirectionnel et simultané) reliant deux applications sur Internet. Les WebSockets reposant sur UDP, il n'y a pas de contrôle sur les données contrairement à TCP. Les données échangées sont également dépourvues de « header[8]», cela fait des WebSockets un moyen de communication très rapide et efficace. De plus, les WebSockets offrent l'avantage au serveur d'envoyer spontanément des données au client sans que ce dernier les ait demandées.

Jetty est un serveur web serveur qui permet d'utiliser des WebSockets et Servlets. Nous avons choisi Jetty pour ces hautes performances mais aussi parce qu'il est possible de l'intégrer dans une application (mode que l'on appelle « Jetty Embedded »), ce qui facilite son utilisation et la distribution de l'application finale : nous n'avons plus besoin de lancer le serveur indépendamment.

Nous avons dû utiliser deux API différentes pour pouvoir développer notre projet à la fois sur les robots NXT et sur les robots EV3. Les robots NXT étant sortis depuis un certain temps, nous utilisons une API stable et ayant fait ses preuves du nom de « LEJOS » pour contrôler le NXT. Concernant l'EV3, nous avons dû procéder à la réécriture complète en Java d'une API très récente développée en C# par une équipe de Microsoft qui s'avère très stable et optimisée.

Afin de récupérer le flux vidéo de la caméra d'un utilisateur directement à partir d'un navigateur, nous avons utilisé quelques parties des fonctionnalités proposées par WebRTC, une API JavaScript permettant la communication en temps réel entre navigateurs. Cependant, nous souhaitions récupérer le flux vidéo directement sur le serveur Jetty, c'est pourquoi nous n'avons utilisé que certaines parties précises de cette API. De plus, WebRTC sera rapidement compatible avec tous les navigateurs et n'est pas une technologie propriétaire, ce qui correspond bien à notre souhait que la diffusion du contenu soit possible quel que soit le type de téléphone.

Pour contrôler nos robots à distance, nous avons choisi d'utiliser la technologie Bluetooth qui est présente sur l'EV3 et le NXT. Bien que le Bluetooth soit contraint de communiquer sur une portée assez réduite, cela est suffisant pour notre utilisation et surtout nous ne sommes pas dépendants d'un réseau wifi qui devrait être fiable.

[7] Protocole réseau offrant un canal de communication bidirectionnel pour les navigateurs et serveurs web. Nous utilisons ce protocole pour communiquer entre notre serveur et l'interface web.
[8] Informations décrivant les données d'une requête (ou la requête elle-même).

2.2.4. Langages de programmation utilisés

Nous avons essentiellement développé en Java, langage objet et de haut niveau, dans notre projet pour la mise en place du serveur Jetty et l'intégration des WebSockets, ainsi que le développement des différentes fonctionnalités (la cartographie par exemple).

Bootstrap

Nous avons utilisé le langage JavaScript pour rendre l'interface web dynamique pour l'affichage en temps réel du flux vidéo et de la cartographie, mais aussi pour le joystick utilisé pour le pilotage manuel. De plus, nous avons utilisé JavaScript en corrélation avec Bootstrap, un framework HTML/CSS très en vogue, pour avoir une interface web ergonomique et facilement modulable. Nous avons utilisé également HTML5/CSS3 pour le contenu de la page et la mise en forme.

Pour utiliser un robot EV3, nous avons dû réécrire en Java une API écrite en C#, ce qui nous a permis de nous rendre compte des différences entre ces deux langages et de la propension forte du C# pour la programmation événementielle et asynchrone. Nous avons rencontré plusieurs problèmes au niveau du typage. Effectivement, certains types, comme les « *enum* », n'ont pas les mêmes caractéristiques en C# qu'en Java : en C#, on peut attribuer une valeur aux éléments d'un *enum*, alors qu'en Java, les éléments d'un *enum* ne peuvent pas avoir de valeurs. De plus, il existe en C# le type « *ushort* » définissant les entiers de 16 bits non signés. Or, en Java, il n'existe aucun type qui ne soit pas signé, nous sommes donc contraints d'utiliser soit un *char* pour se rapprocher de la taille de *l'ushort*, soit prendre plus simplement un *int*, qui est un entier signé de 32 bits.

3. Réalisation du projet

3.1. Chronologie / Les grandes étapes

Le projet s'est déroulé en plusieurs étapes qui nous ont permis, petit à petit, d'arriver à avoir un ensemble cohérent dépassant en bien des aspects nos attentes initiales. Nous allons retracer sommairement les moments forts du projet, d'octobre 2013 à mars 2014.

La mission initiale du projet était de développer un algorithme de cartographie pour un robot de la gamme LEGO Mindstorms NXT dont le résultat de la cartographie pourrait être suivi sur une interface web, avec la possibilité de visionner l'environnement grâce à la caméra embarquée sur le robot (téléphone relié à Internet). Nous sommes partis du travail qu'avait fait Jordan quelques mois auparavant.

Dès lors que le projet a été accepté en octobre, nous nous sommes directement répartis les tâches entre nous: Nicolas et Jonathan ont tout d'abord travaillé sur la recherche d'un algorithme pour la cartographie ; Jordan s'est occupé de réécrire l'API du robot sur le serveur afin de ne plus avoir à mettre de programme directement sur la brique du robot.

Nous avons constaté que les algorithmes de cartographie émanent pour la plupart de laboratoires de recherche. Nous nous sommes vite rendu compte que ces algorithmes étaient bien trop complexes et demandaient des connaissances très pointues en mathématiques, c'est pourquoi nous nous sommes résolus à développer nous-mêmes un algorithme de cartographie aussi simple que possible. Concernant la réécriture de l'API, nous ne voulions plus être contraints d'avoir à charger nos programmes de traitement des commandes directement sur la brique du robot, nous avons donc procédé à une réécriture de toute l'API d'exécution des commandes directement sur le serveur, ce qui permet de n'avoir plus qu'à envoyer les commandes toutes prêtes au robot via le « Bluetooth ».

De plus, à ce stade-là, nous avons procédé à un recentrage du projet avec le tuteur sur le développement d'une architecture de communication entre un avatar, dans notre cas le robot explorateur, et Internet qui soit stable et réutilisable. Ainsi, nous nous sommes surtout souciés d'avoir un algorithme de cartographie fonctionnel plutôt qu'un algorithme précis, la précision n'étant pas nécessaire pour illustrer cette architecture. Notre projet a donc ensuite eu pour mission de mettre en œuvre un premier essai d'architecture avec pour avatar un robot explorateur, illustré par des fonctionnalités telles que la cartographie autonome.

Nicolas a recherché des solutions pour afficher le résultat de la cartographie du robot directement sur l'interface web tandis que Jonathan élaborait un premier algorithme de cartographie, simple mais qui a servi de base. Cet algorithme s'est ensuite étoffé au fil des tests itératifs.

En Janvier, suite à la mise en ligne de l'API en C# pour les robots de nouvelle génération EV3, nous avons décidé de prendre le temps nécessaire au portage de cette API en Java et ainsi pouvoir utiliser les robots de nouvelle génération comme ceux de l'ancienne génération. D'autre part, les capteurs ultrasoniques de ces derniers sont précis au millimètre alors que ceux du NXT ne le sont qu'au centimètre, ce qui est un gain de précision non négligeable pour la cartographie.

Ensuite, nous avons principalement travaillé sur l'algorithme de cartographie et plus spécifiquement sur le problème de la localisation du robot. Nous avons également prévu une correction de trajectoire et un système de mémorisation des obstacles pour cartographier intelligemment (en évitant les zones déjà explorées).

La dernière partie à développer fut l'affichage du flux vidéo enregistré en direct par la caméra du téléphone fixé sur le robot. Il a fallu envoyer le flux vidéo de la caméra du téléphone au serveur sur l'ordinateur passerelle, qui reçoit le flux encodé et le renvoie à l'interface web, laquelle à son tour reconstruit le flux à partir des images reçues pour constituer une vidéo en direct (« live streaming »).

Le travail final de ce projet fut l'intégration de chacun des composants développés, pour permettre à l'utilisateur du site de pouvoir réaliser les actions suivantes : se connecter au robot, lancer la cartographie autonome d'un environnement, piloter manuellement le robot, ou tout simplement tester des fonctionnalités de base. Nous avons ajouté à chacune de ces fonctionnalités la possibilité de visionner en même temps ce que filme la caméra sur le robot.

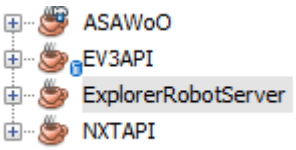
Finalement, nous avons eu l'opportunité de pouvoir présenter l'intégralité du projet au salon Innorobo à Lyon le 20 mars 2014, ce qui fut très valorisant.

3.2. Découpage du projet

Pour faciliter et optimiser la réalisation de notre projet, nous avons décidé de le découper en plusieurs sous-projets pour appliquer le concept du « Separation of Concerns » (séparation claire des problèmes).

Cela nous a permis de faciliter la navigation dans les différentes parties du projet et de modifier le code sans affecter le bon fonctionnement de l'application.

Nous avons donc 4 sous-projets :



Nos sous projets sur Netbeans

Le projet ASAWoO correspond à l'architecture dans laquelle nos robots sont représentés

Les projets EV3API et **NXTAPI** sont respectivement les APIs pour le contrôle des robots EV3 et NXT

Le projet ExplorerRobotServer représente notre serveur (HTTP et WebSocket).

Evidemment, pour notre application, tous nos projets sont étroitement liés. En effet l'architecture ASAWoO a besoin des APIs des robots pour représenter l'avatar de notre robot explorateur. De la même manière, le serveur utilise l'avatar du robot pour exposer ce dernier sur internet et permettre aux utilisateurs de le contrôler.

Néanmoins, nous avons développé en gardant à l'esprit que chaque brique de ce projet peut être réutilisée indépendamment pour un autre projet.

3.3. Fonctionnement général du projet

Nous pouvons visualiser cette architecture sous la forme de trois couches qui communiquent de manières contiguës entre elles : le serveur reçoit des messages venant de l'interface web contenant les commandes à envoyer et fait remonter au client les informations renvoyées par le robot, ce dernier reçoit des commandes venant du serveur situé sur la passerelle et lui renvoie les réponses aux commandes initiées par le client.

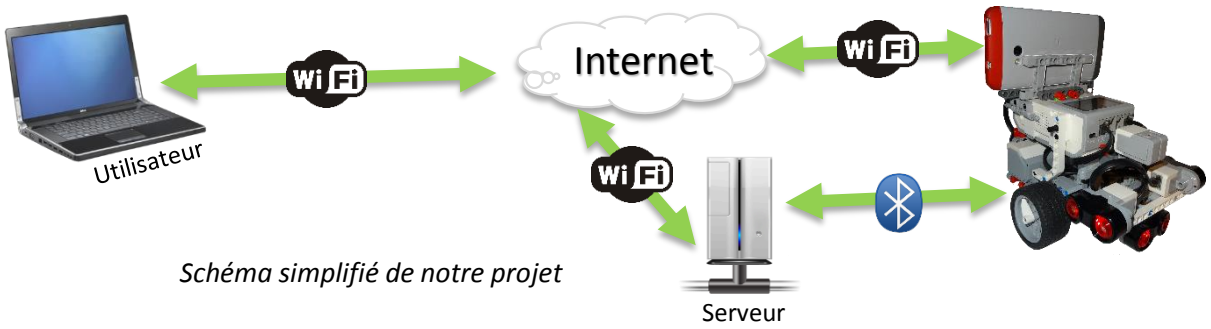


Schéma simplifié de notre projet

Ainsi, si le client demande la valeur du gyroscope sur l'interface web, un message est transmis du site pour arriver sur le serveur, le serveur réceptionne ce message à l'aide d'une WebSocket et le traduit sous forme d'une commande qui est envoyée au robot via un canal « Bluetooth » ; le robot répond au serveur en retournant la valeur du gyroscope sur ce même canal, et le serveur répond à l'interface web qui n'aura plus qu'à afficher au client la valeur actuelle du gyroscope.

Nous pouvons de ce fait à la fois gérer les requêtes de l'interface web arrivant sur le serveur et transmettre à l'interface web les informations du robot utiles à chaque fonctionnalité.

Les messages échangés entre l'interface web et la passerelle sont au format JSON[9], un format grandement utilisé pour faire communiquer des applications entre elles. Ce format de message très souple et abstrait permet de créer son propre formatage des messages, propre aux besoins de l'application concernée. Pour notre projet, nous utilisons des messages JSON pour caractériser les différents messages échangés entre la passerelle et le site : pour définir le type du robot (EV3, NXT) et les paramètres associés, pour faire appel aux fonctionnalités (cartographie, pilotage manuel, avancer tout droit, etc.), pour récupérer directement la valeur des capteurs, et ainsi de suite. La requête et la réponse sont des messages JSON qui sont respectivement envoyés et reçus par l'interface web.

Exemples de message JSON

Faire tourner les deux moteurs à 100% de leur puissance :

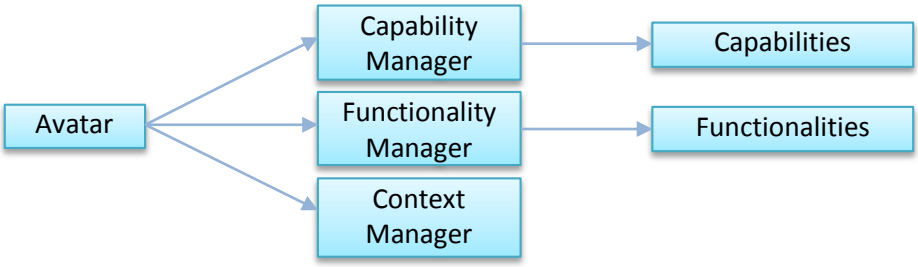
```
{ motor_command: [ {name: both, action: start,  speed: 100} ]  }
```

Demande l'exécution de la fonctionnalité « mapping » :

```
{ functionality_call : {name: "mapping", action: "start"} }
```

3.4. Architecture ASAWoO

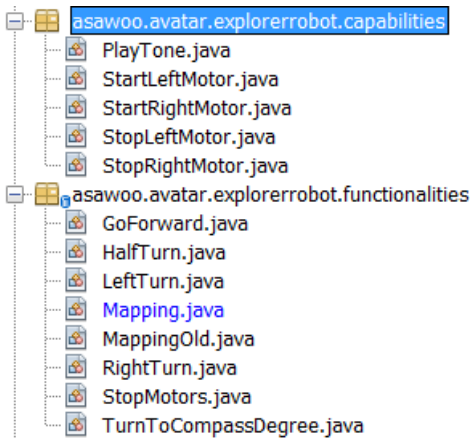
La partie de l'architecture ASAWoO que nous avons développée en langage Java suit un schéma d'organisation précis :



Implémentation de l'architecture ASAWoO

[9] Abréviation pour JavaScript Object Notation, format de données textuelles qui permet de structurer l'information et est utilisé par le langage standard ECMAScript, et mis en œuvre par des langages comme JavaScript ou ActionScript.

Le cœur de l'architecture et son originalité résident dans le découpage logique des classes utilisées par l'avatar : on part du principe que l'avatar, dans notre cas le robot explorateur, possède des capacités de bas niveau qui ne sont pas accessibles à l'utilisateur, par exemple démarrer le moteur gauche du robot. En revanche, ces capacités sont utilisées par des fonctionnalités de haut niveau accessible à l'utilisateur, par exemple la cartographie autonome d'un environnement : cette fonctionnalité utilise un grand nombre de capacités de bas niveau pour fonctionner correctement.



Découpage en capacités et fonctionnalités

Par exemple, pour la fonctionnalité « GoForward » (avancer tout droit), nous utilisons les capacités « StartRightMotor » et « StartLeftMotor » qui démarrent et font avancer respectivement le moteur droit et le moteur gauche. Pour permettre une telle utilisation des capacités dans les fonctionnalités, nous avons suivi un « design pattern[10] » se prêtant bien à ce problème : le Command pattern. Ce « design pattern » a la particularité de représenter chaque requête par un objet, par exemple la requête « arrêter le moteur droit » est représentée par l'objet « StopRightMotor », héritant d'une interface[11]. Cette interface propose la méthode « execute », dans notre cas l'interface est soit « Capability » soit « Functionality », à laquelle on passe des paramètres pour l'exécution de la requête. Grâce à cette encapsulation des requêtes dans des classes, nous pouvons utiliser dans notre projet des capacités directement dans les fonctionnalités, qui sont utilisées de la même manière par le serveur les appelant. Ainsi, la structuration est bien conforme à l'architecture « ASAWoO ».

Toutes ces capacités et fonctionnalités sont regroupées dans des listes de gestion (CapabilityManager et FunctionalityManager) permettant d'en ajouter ou d'en supprimer dynamiquement selon l'état de l'avatar, qui est supervisé par un gestionnaire de contexte (ContextManager). Ce dernier s'occupe, selon l'environnement et le contexte, d'activer ou de désactiver des fonctionnalités. On pourrait par exemple illustrer son action lorsqu'un robot rencontre un obstacle gênant sa progression en face de lui : le ContexteManager pourrait dynamiquement arrêter le robot et désactiver toute fonctionnalité permettant d'aller tout droit.

3.5. Réécriture de l'API C# en Java

Une API écrite en C# pour contrôler les robots de nouvelle génération, EV3, a été développée par une équipe Microsoft peu de temps après que ces robots soient disponibles sur le marché. Quand nous avons commencé le projet, l'API que nous utilisions pour contrôler le NXT n'existait pas encore dans sa version pour les nouveaux robots EV3, nous avons donc décidé de ne pas développer pour les EV3. Cependant, nous avons trouvé en Janvier cette API en C# pour les EV3 et avons décidé de la réécrire en Java, ce qui a eu pour effet de rendre le projet plus générique : nous pouvons maintenant contrôler soit un NXT, soit un EV3.

[10] Littéralement « patron de conception », désigne une solution générique reconnue comme bonne pratique à un problème de conception en développement logiciel.
[11] En Java, une interface est une classe qui définit un certain nombre de méthodes à implémenter pour une classe qui implémente cette interface. Généralement, une interface définit un comportement.

L'API C# est très performante car, d'une part tous les messages envoyés au robot sont écrits en base hexadécimale, et d'autre part elle propose de grouper les requêtes pour connaître les valeurs des capteurs et ainsi alléger le canal de communication « Bluetooth ».

```
sensors = robot.getSensorValue(FRONT_ULTRASONIC, LEFT_ULTRASONIC, RIGHT_ULTRASONIC, COMPASS);
```

Requête nous renvoyant les valeurs du gyroscope et des capteurs ultrasoniques de devant, de droite, et de gauche.

La réécriture de l'API s'est avérée être un travail complexe mais très enrichissant, car cette API manipule à plusieurs reprises des suites de bits dont la signification n'est pas toujours évidente. Nous avons également rencontré le fameux problème de l'Endianness[12], qui a été résolu après réflexion et plusieurs tests. De plus, l'API était incomplète et très peu commentée, nous avons donc eu besoin de consulter le code source du firmware[13] de l'EV3 pour comprendre les décisions des développeurs et savoir comment accéder aux fonctionnalités manquantes.

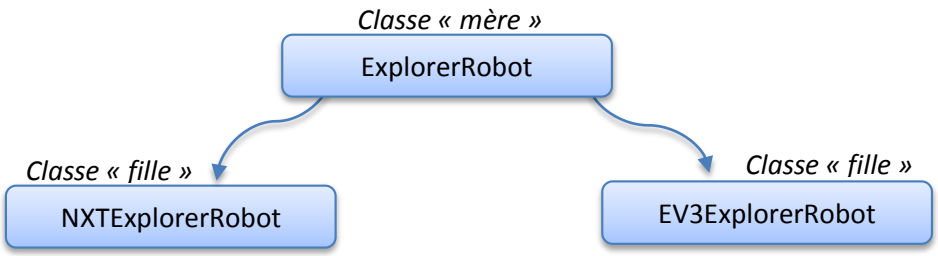
A travers la réécriture en Java de cette API, nous avons appris à manipuler directement les bits, ce qui nécessite beaucoup de concentration et une capacité à « traduire » ce que chaque suite de bits signifie pour notre programme.

Cette API étant une partie importante et distincte, nous avons décidé de créer un projet à part sur GitHub. Nous avons essayé de commenter au maximum l'API Java afin qu'elle soit facilement compréhensible et maintenable dans le futur. A terme, notre objectif est de publier cette API sur Internet.

3.6. Généricité : utilisation de plusieurs robots

Dès lors que nous avons procédé à la réécriture de l'API pour l'EV3, le projet a pris un tournant important : nous pouvons à la fois utiliser un NXT ou un EV3 dans le projet, à condition de faire les changements nécessaires dans notre programme.

Pour cela, nous avons suivi le schéma suivant :



Nous avons opté pour une relation d'héritage entre nos classes, avec une classe dite « mère » du nom de « ExplorerRobot » contenant toutes les méthodes qui seront à implémenter par chaque robot. Cette classe mère représente, dans le cas de l'architecture ASAWoO, notre avatar et hérite de la classe mère « Avatar ». Les deux classes dites « filles », représentant le NXT et l'EV3 (respectivement NXTExplorerRobot et EV3ExplorerRobot), vont hériter de cette classe mère et implémenter toutes les méthodes de la classe mère. Cette structuration permet ensuite de pouvoir utiliser dans notre

[12] Les données en informatique peuvent être représentées sur une suite d'octets ordonnée. Cependant, cet ordre n'est pas fixé et dépend du choix du développeur qui décide d'attribuer une certaine signification à un ordre qu'il fixe lui-même. Plus couramment, deux normes existent : le Big Endian, et le Little Endian.
[13] Micro logiciel intégré dans un matériel informatique et, dans notre cas, exécuté par l'EV3 pour qu'il puisse fonctionner.

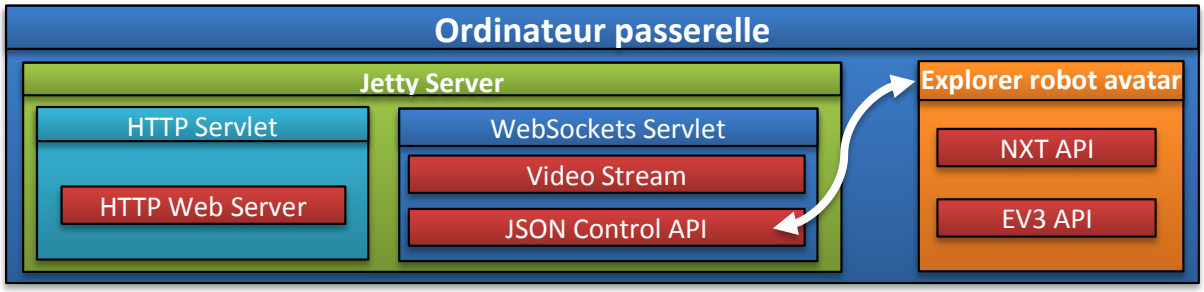
programme un objet du type ExplorerRobot sans se soucier du véritable type à l'exécution du programme. Selon le robot choisi par l'utilisateur, lorsque la fonctionnalité sera appelée, l'utilisateur fixera le type du robot : soit un EV3, soit un NXT, et ce type sera utilisé tout au long du programme.

En outre, cette structuration permet de pouvoir ajouter très facilement d'autres types de robots au projet très facilement : il suffit d'implémenter les méthodes fournies par la classe mère « ExplorerRobot » dans la classe qui représentera le nouveau type de robot. Néanmoins, pour être cohérent, le robot à ajouter doit pouvoir « explorer » un environnement, c'est-à-dire être muni de capteurs adéquats, à l'image de notre robot explorateur.

3.7. Le serveur Jetty

Dans notre projet, la passerelle constitue l'élément moteur de toutes les autres parties. C'est en réalité une machine qui exécute une application codée en Java contenant un serveur HTTP et de WebSocket.

Le schéma ci-dessous explique comment les différentes briques du projet sont orchestrées autour du serveur Jetty.



Architecture de l'application présente sur la passerelle

3.7.1. Les Servlets

Les Servlets sont des classes Java qui permettent l'interaction avec un utilisateur au travers d'un serveur HTTP et utilisées pour générer du contenu, typiquement une page HTML. Pour permettre cette interaction, les données qui transitent entre le serveur et l'utilisateur peuvent prendre la forme de contenu HTML, ou sont formatées en XML ou encore JSON dans le cas d'un service web.

Pour notre projet nous avons mis en place deux types de Servlets :

Servlet HTTP

Ce type de servlet repose sur le modèle de requête/réponse d'HTTP :

Communication HTTP client/serveur

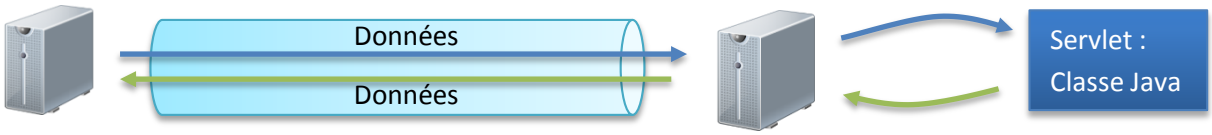
Un utilisateur envoie une requête à travers un navigateur, la Servlet reçoit et traite la requête et renvoie une réponse. Avec ce type de Servlet, nous avons créé un serveur web (comme Apache) pour rendre accessible les fichiers HTML/JS/CSS de nos applications web depuis un navigateur.

Pour notre serveur web, la classe Java, du schéma ci-dessus, est une application qui va récupérer le contenu d'un fichier demandé par une requête (par exemple *application_client.html*) et le retourner à l'utilisateur.

Dans le cadre du projet, nous avons deux applications web qui doivent être accessibles depuis un navigateur : celle destiné au client (*voir 3.9 Interface utilisateur p17*) et celle du broadcaster (*voir 3.8 Live Streaming p17*).

Servlet de WebSocket

Les Servlets de type WebSocket utilisent le protocole WebSocket pour communiquer avec un utilisateur. Contrairement à HTTP, les WebSocket ne reposent pas sur un principe requête/réponse, mais sur de l'événementiel, c'est-à-dire que le serveur peut spontanément décider d'envoyer des données à l'utilisateur sans que ce dernier les ai demandées et inversement. De même, après l'envoi de données, une réponse n'est pas requise (précédemment, pour HTTP, même si une requête n'attend pas de réponse, le serveur renvoie au minimum si la requête a bien été reçue/exécutée ou non).



Communication client/serveur avec les Websockets

Pour mettre en place un tel système « événementiel », une connexion persistante est ouverte entre le client et le serveur. Cette connexion est à l'initiative de l'utilisateur et acceptée (ou non) par le serveur. La phase de connexion est demandée par l'utilisateur avec le protocole HTTP. Cette demande est une requête HTTP qui comporte simplement un header demandant le changement de protocole : « **Upgrade : WebSocket »**. Le serveur répond alors avec le statut « **101 : Switching protocol »**. Dès lors, le tube de communication bidirectionnel est ouvert, ainsi les clients et le serveur peuvent s'envoyer des messages spontanément.

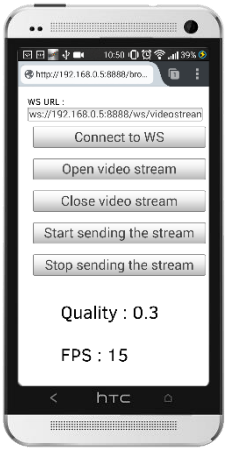
Nous avons utilisé les WebSockets pour deux applications : celle du broadcaster afin d'envoyer le flux vidéo et celle du client pour recevoir le flux vidéo, contrôler le robot et recevoir des informations de ce dernier.

3.7.2. Intégration dans le projet

Pour déployer un serveur Jetty dans une application, il existe deux possibilités. La première est d'exécuter une instance de serveur Jetty (qui n'est ni plus ni moins qu'une application Java) en passant en paramètre notre application. Cette solution nécessite de lancer manuellement le serveur avec une configuration statique (typiquement le port d'écoute du serveur).

La deuxième possibilité est d'intégrer le serveur Jetty dans notre application (mode appelé « standalone » ou « embedded »). Le principal avantage est que le serveur est piloté par notre application. Dès lors nous pouvons configurer, démarrer ou redémarrer le serveur à notre convenance. De plus, il n'est pas nécessaire d'avoir notre application et en plus le serveur Jetty à côté, ce qui est beaucoup plus portable et facile pour tester. C'est donc cette deuxième possibilité – « Jetty embedded » – que nous avons choisi pour notre projet.

3.8. Live streaming



Le visionnage de l'environnement dans lequel le robot évolue était une des idées que nous gardions de côté, n'étant pas sûrs d'avoir le temps de la mettre en œuvre. Grâce aux outils de gestion de projet, nous avons pu bien avancer et par conséquent développer cette fonctionnalité. Le visionnage en direct d'un flux vidéo émis par une source externe est plus généralement appelé « live streaming ». Afin de mettre en place du « live streaming », nous avons forcément besoin d'une source qui va servir de point d'entrée au flux vidéo, représentée par le dispositif qui filme. Dans notre cas, c'est le téléphone, muni d'une caméra, fixé sur le robot qui filme dans le sens de déplacement du robot. Nous devons d'abord nous connecter avec le téléphone à une application que nous avons développée pour faire du « live streaming », présente sur l'interface web.

Sur cette application, nous nous enregistrons en tant que « broadcaster » (émetteur) pour signaler que nous allons fournir le flux vidéo. Le flux vidéo va être envoyé en permanence au serveur via l'utilisation d'une WebSocket dédiée. Cependant, nous envoyons le flux vidéo sous la forme d'images encodées en base 64, correspondant à une suite de caractères. Ces images encodées sont donc envoyées à travers la WebSocket au serveur, qui va diffuser ces images à tous les spectateurs (clients) qui auront activé le « live streaming » sur le site. Effectivement, nous avons élaboré un véritable système de live streaming diffusant en direct le contenu de l'émetteur à tous les spectateurs.

Cela permet d'avoir plusieurs clients connectés à l'interface web qui peuvent visionner en direct le contenu de la caméra du robot, à l'image des vrais sites de « live streaming ».

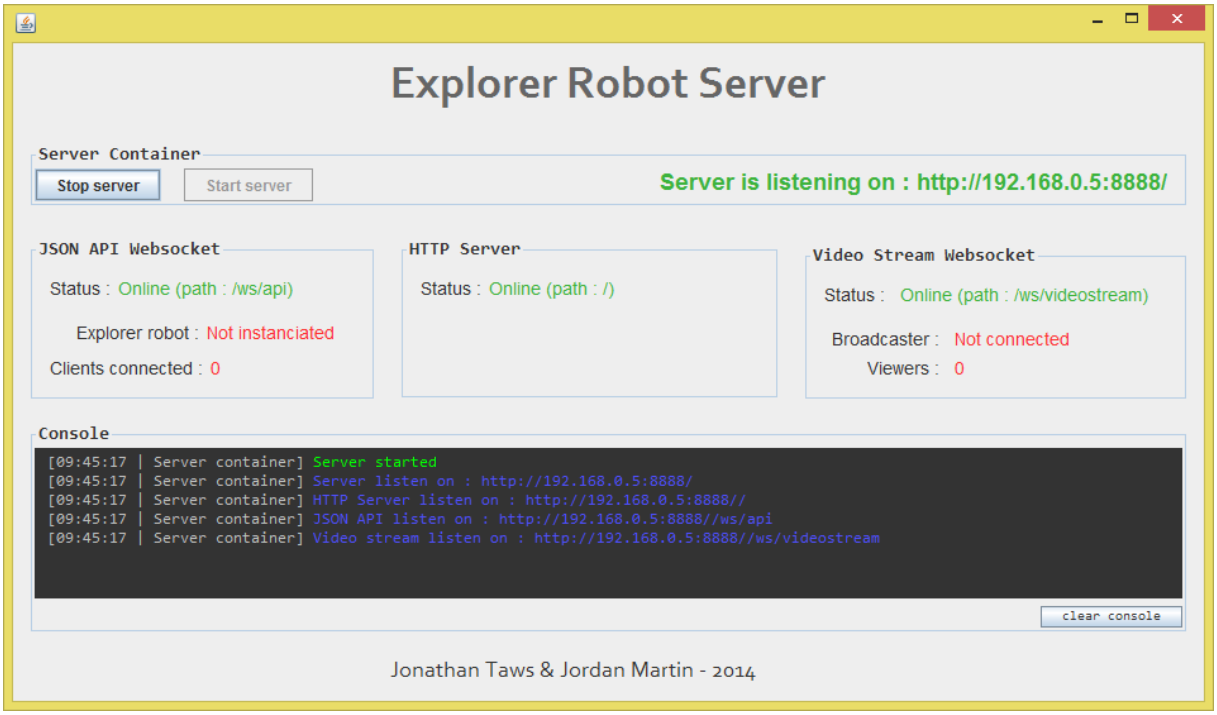
La diffusion des images à tous les clients est d'une telle rapidité que le client ne voit pas une suite d'images mais bel et bien un flux vidéo sans saccades. Cette diffusion se fait toujours sur le même canal de communication maintenu par la WebSocket.

Pour optimiser l'utilisation du serveur, nous avons décidé de n'utiliser qu'une seule WebSocket pour le « live streaming », sans séparer l'émetteur des spectateurs. Pour ce faire, nous enregistrons chaque connexion au service de streaming (un seul émetteur et un ou plus spectateur(s)), puis nous évaluons à chaque fois si un message sur la WebSocket provient de l'émetteur ou du spectateur, que nous redirigeons au bon endroit en fonction de son type.

3.9. Interface utilisateur

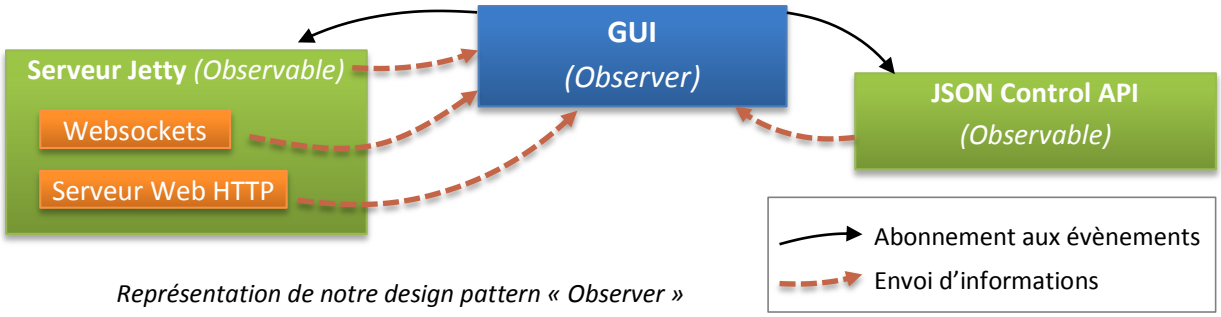
Pour piloter notre application, l'utilisateur dispose de deux interfaces graphiques (que l'on appellera dans la suite GUI pour Graphical User Interface).

La première est une GUI qui permet de lancer ou stopper et avoir des informations sur le serveur Jetty telles que l'adresse IP du serveur, le port d'écoute, le nombre de clients connectés, etc. Nous avons également décidé de centraliser tous les messages de log, de debug, et d'erreur sur cette interface.



GUI du serveur

Afin de récupérer tous ces messages, nous avons mis en place le design pattern **Observer** pour permettre à cette GUI de recevoir des informations des différentes classes de notre projet. Ce design pattern repose sur un mécanisme observateur/observé avec abonnement et permet à une classe d'être à l'écoute d'autres classes. Nous avons couplé ce design pattern à une classe dédiée à la gestion des messages afin de connaitre le type du message (erreur, log, information, debug…) et la classe émettrice. Cela permet d'obtenir des informations pertinentes et précises sur l'état de l'application en temps réel ainsi qu'un débogage rapide des éventuels problèmes.



Représentation de notre design pattern « Observer »

La deuxième interface, et aussi la plus importante, est le site web sur lequel l'utilisateur peut interagir avec un robot. Pour cela, l'utilisateur doit au préalable configurer son robot (choix du robot, port des moteurs et capteurs…) et se connecter au robot. Ensuite, l'utilisateur peut contrôler le robot via un joystick virtuel ou lancer et visualiser l'algorithme de cartographie en temps réel.

Sur ce site web, la cartographie est affichée à l'aide de deux canvas[14] superposés. Le premier permet d'afficher les obstacles vus par le robot ainsi que ces déplacements. Le second est utilisé pour afficher la position actuelle du robot.

[14] Composant HTML permettant de dessiner dynamiquement points, lignes, forme, etc. Le plus souvent, il est utilisé en corrélation avec JavaScript.

3.10. Algorithme de cartographie autonome dans un environnement inconnu

Malgré le nombre de ressources sur les algorithmes de cartographie autonome dans un environnement inconnu, plus communément connu sous l'acronyme SLAM, il a été très difficile de trouver un exemple d'un de ces algorithmes qui soit bien documenté pour une utilisation postérieure. En effet, ces algorithmes sont souvent élaborés par des laboratoires de recherche qui publient des documents expliquant le fonctionnement de leur découverte, plutôt que d'en livrer le code source. Après de longues recherches, nous avons essayé d'obtenir de l'aide de nos enseignants et de chercheurs sur des notions spécifiques abordées par ces algorithmes. Cependant, nous n'avons pas réellement trouvé de solutions satisfaisantes à ce problème et nous nous sommes donc résolus à développer notre propre algorithme de cartographie, en partant de considérations logiques.

Nous nous sommes rendus compte au fil des tests de cet algorithme que la théorie ne suffisait pas à construire un algorithme robuste et fonctionnel, et que seule l'observation de la pratique nous a réellement permis d'élaborer un algorithme répondant à nos besoins. En effet, nous avons dû patienter avant de pouvoir le tester car nous travaillions en parallèle sur la réécriture de l'API sur le serveur, l'algorithme ne reposait donc que sur des fondements théoriques. Néanmoins, dès que l'API fut fonctionnelle, nous avons grandement itérer sur des versions différentes de l'algorithme, en développant à chaque fois de nouvelles fonctionnalités.

Le fonctionnement général de l'algorithme repose sur l'évaluation des obstacles. Les trois capteurs ultrasoniques calculent en permanence la distance des obstacles dans leur champ de vision respectifs, permettant de les placer sur la carte. Nous évaluons si la distance du robot par rapport à ces obstacles gênera sa progression dans la seconde qui suit, et si cela est le cas, nous testons séquentiellement chaque direction jusqu'à ce que l'on trouve un obstacle qui ne gêne pas. Grâce aux trois capteurs, cette prise de décision est très rapide et semble quasiment instantanée pour un spectateur. Nous avons également mis en place un système de correction de trajectoire reposant toujours sur l'évaluation des distances renvoyées par les capteurs. Cela permet de remédier au problème de « l'angle mort » récurrent dans notre cas. En effet, si le robot se dirige de biais vers un mur ou obstacle, le capteur de devant ne verra pas d'obstacle (angle mort). Dans ce cas, nous utilisons les capteurs sur le côté pour évaluer l'angle d'incidence par lequel nous arrivons sur le mur et nous appliquons une formule afin de corriger la trajectoire du robot pour éviter l'obstacle.

Une version différente de l'algorithme, reprenant les fondements de l'algorithme précédemment énoncé, met en place le système de mémorisation des obstacles et de prise de décision en conséquence. Nous enregistrons toutes les positions du robot et des obstacles dans une matrice que nous évaluons localement pour déterminer dans quelle direction partir afin de cartographier efficacement. Nous prenons la position actuelle du robot, et nous regardons les obstacles déjà enregistrés dans la matrice pour ensuite choisir la direction la plus adéquate pour cartographier l'environnement le plus rapidement possible. Cependant, nous n'avons pas réussi à finaliser à temps cette version optimisée de l'algorithme.

Le véritable problème que nous avons rencontré avec cet algorithme venait du mécanisme permettant de déterminer précisément la position du robot. En effet, nous ne pouvons pas utiliser de GPS, qui n'est pas suffisamment précis pour de la cartographie à si petite échelle. De plus, le cœur du système de cartographie repose sur la position du robot : nous positionnons chaque obstacle en fonction de la position du robot. Nous avions d'abord réfléchi à une mise en relation des valeurs des capteurs ultrasoniques pour déterminer la position, mais cette méthode s'est révélée peu efficace à cause des situations où les capteurs ne renvoient aucune valeur. Après plusieurs essais infructueux d'autres

méthodes, nous avons utilisé ce que l'on appelle le positionnement par odométrie. L'odométrie correspond à la valeur des moteurs (que l'on appelle « le compte-tours»), qui s'exprime en degré. Nous pouvons ainsi dire, par exemple, que si le compte-tours affiche 360 degrés, nous avons avancé d'un centimètre. De plus, si un des compte-tours est plus important que l'autre, cela veut dire que nous avons commencé à tourner dans un sens. Nous avons utilisé cette donnée, en conjonction avec une relation sur le nombre de degrés à effectuer pour parcourir un centimètre, pour déterminer précisément la position du robot. Nous utilisons également le gyroscope pour être encore plus précis dans l'estimation de la position du robot, tout particulièrement lorsque l'on se déplace de biais.

4. Bilan personnel

4.1. Jordan

Je n'avais jamais vraiment travaillé sur un projet aussi important en équipe. Ce projet a donc été une réelle mise en situation professionnelle et m'a vraiment permis de me rendre compte que le travail en équipe est parfois indispensable.

Etant donné que nous avions nous-mêmes proposé et choisi le sujet de ce projet tuteuré, il me tenait très à cœur et m'a vraiment motivé pour me repousser. Je crois pouvoir dire que ce fut une réelle réussite. Je suis très content du résultat obtenu.

Ce projet a été très enrichissant car nous avons dû utiliser et mettre en relation de nombreuses technologies. Cela m'a aussi permis d'apprendre et d'approfondir certaines de ces technologies. Il est certain que tout ce que m'a apporté ce projet va m'être utile durant mon stage de fin de DUT mais aussi pour le reste de mon cursus universitaire et professionnelle.

J'ai tout particulièrement apprécié le travail de réécriture de l'API C# en Java, qui nécessitait de comprendre, à très bas niveau, le fonctionnement de l'EV3 et la manière dont les messages transitaient entre le robot et l'API. J'ai également pu me rendre compte du gain de performance qu'apporte une telle approche.

Pour finir, ce projet tuteuré a également permis de renforcer mon choix quant à mes poursuites d'études et le métier vers lequel je souhaite me diriger ; Développeur web en back-end avec la technologie J2EE.

4.2. Jonathan

Ce projet m'a personnellement permis une réelle montée en compétences dans divers domaines des métiers de l'informatique : l'analyse, le développement de logiciel, la gestion de projet et la communication.

Ce projet m'a permis, à travers toutes les étapes de développement, de réellement me perfectionner en langage Java. Effectivement, nous avons essentiellement codé en langage Java et appris ainsi à maîtriser en grande partie les concepts de ce langage. J'ai également eu l'opportunité d'apprendre et de maîtriser des technologies très récentes, telles que WebRTC pour le « live streaming ».

Je me suis rendu compte de l'importance des tests, et plus particulièrement des pratiques de développement piloté par les tests (Test Driven Developpement). La conception de l'algorithme de cartographie illustre bien cette pratique : c'est en appliquant une méthode de tests itératifs que nous avons pu bâtir et enrichir un algorithme fonctionnel répondant à nos besoins.

Je me suis également initié à l'utilisation de « patrons de conception » (design pattern), que nous avons largement utilisés dans la conception et le développement de l'architecture. Ces derniers proposent des solutions à des problèmes standards en programmation.

J'ai pu apprécier le travail en équipe sur ce projet de plusieurs mois. Cela nécessite d'établir un planning afin de gérer la charge de travail et de planifier les tâches afin de suivre notre avancement. Il est aussi intéressant de constater que nous n'avons pas réellement eu de désaccords problématiques tout au long du projet, chaque membre ayant su être force de proposition et faire

preuve d'écoute envers le reste de l'équipe, ce qui a permis d'avoir une avancée constante de nos travaux et une bonne atmosphère de travail.

Ayant une forte motivation pour ce sujet « innovant », ce projet tuteuré m'a fait découvrir d'une certaine manière le domaine du « Web des Objets », dont on parle tant, mais dont je ne connaissais pas l'envers du décor. J'ai maintenant une vision bien plus précise et claire des processus mis en jeu pour arriver à faire communiquer un objet avec une application sur Internet.

La rédaction du rapport de projet et sa présentation, étapes ultimes de notre projet m'a aussi permis d'analyser et de synthétiser notre travail, la communication sur un projet informatique étant un élément majeur dans la réussite d'un projet en entreprise.

5. Conclusion

Ce projet a été très formateur pour l'ensemble des membres de l'équipe puisque nous avons pu à la fois mettre en pratique les acquis de notre formation à travers les différents travaux menés et découvrir le monde de la robotique. Nous avons notamment pu largement nous perfectionner en langage Java et en développement Web.

Ce projet nous a également permis d'adopter une démarche projet que nous avions étudiée en cours : la méthodologie Agile. Nous avons ainsi pu nous rendre compte concrètement des tenants et aboutissements de cette méthodologie qui s'est avérée particulièrement efficace pour notre projet qui nécessitait une démarche de tests et de développement itératifs. De plus, cette méthodologie tend à être de plus en plus intégrée en entreprise pour la gestion de projet informatique, ce qui nous a permis d'avoir une idée plus précise du déroulement de tels projets.

Notre sujet, par son aspect innovant, était très intéressant car nous avons pu étudier diverses problématiques et ainsi élargir notre spectre de compétences. Effectivement, le fait d'avoir travaillé à la fois avec un robot, un serveur et un site internet, nous a permis de mettre en œuvre différentes techniques et d'acquérir ainsi une certaine polyvalence.

Nous avons également apprécié les différents échanges avec notre tuteur et les enseignants chercheurs du LIRIS avec qui nous avons partagé nos problématiques à certains moments clés du projet.

Finalement, ce fut un point fort du projet tuteuré d'avoir eu l'opportunité de présenter notre travail au salon Innorobo à Lyon. Nous avons pu expliquer au public, au travers de démonstrations, le fonctionnement de notre robot explorateur et nous avons aussi eu un retour positif et un regard extérieur sur ce projet ce qui nous a récompensé de nos efforts.

6. Glossaire

LIRIS : Laboratoire d'InfoRmatique en Image et Système d'information situé sur le campus de la Doua à Villeurbanne.

Live streaming : Désigne l'envoi d'un contenu audio ou vidéo « en direct » en continu et à mesure qu'il est diffusé. Grâce à ce système, des utilisateurs peuvent suivre en direct un contenu proposé par un émetteur, généralement à travers internet.

INFRA : Le programme INFRA est un appel à projet de l'ANR qui a pour vocation de susciter et soutenir la recherche dans des domaines stratégiques que sont les infrastructures matérielles et logicielles pour l'internet du futur. Le projet ASAWoO a été choisi pour faire partie de ce programme.

ARM : Processeurs dotés d'une architecture RISC 32 bits ou 64 bits développés par la société ARM. Ces processeurs sont très populaires dans l'informatique embarquée pour leur faible consommation et leur simplicité d'utilisation.

Dongle : Composant matériel qui se branche sur un port d'entrées-sorties d'un ordinateur et qui peut permettre, selon son type, de proposer du stockage (clé USB), un accès à un réseau WiFi, Bluetooth, etc.

API : Abréviation pour Application Programming Interface, ce terme est utilisé pour décrire un ensemble de classes et méthodes qui sont à disposition pour les programmeurs et visent à simplifier l'utilisation d'un composant.

WebSocket : Protocole réseau offrant un canal de communication bidirectionnel pour les navigateurs et serveurs web. Nous utilisons ce protocole pour communiquer entre notre serveur et l'interface web.

Header : Informations décrivant les données d'une requête (ou la requête elle-même).

JSON : Abréviation pour JavaScript Object Notation, format de données textuelles qui permet de structurer l'information et est utilisé par le langage standard ECMAScript, et mis en œuvre par des langages comme JavaScript ou ActionScript.

Design pattern : Littéralement « patron de conception », désigne une solution générique reconnue comme bonne pratique à un problème de conception en développement logiciel.

Interface : En Java, une interface est une classe qui définit un certain nombre de méthodes à implémenter pour une classe qui implémente cette interface. Généralement, une interface définit un comportement.

Endianness : Les données en informatique peuvent être représentées par une suite d'octets ordonnée. Cependant, cet ordre n'est pas fixé et dépend du choix du développeur qui décide d'attribuer une certaine signification à un ordre qu'il fixe lui-même. Plus couramment, deux normes existent : le Big Endian, et le Little Endian.

Firmware : Micro logiciel intégré dans un matériel informatique pour qu'il puisse fonctionner.

Canvas : Composant HTML permettant de dessiner dynamiquement points, lignes, forme, etc. Le plus souvent, il est utilisé en corrélation avec JavaScript.

7. Annexes

7.1. Comparatif diagrammes de Gantt

Diagramme de Gantt prévisionnel

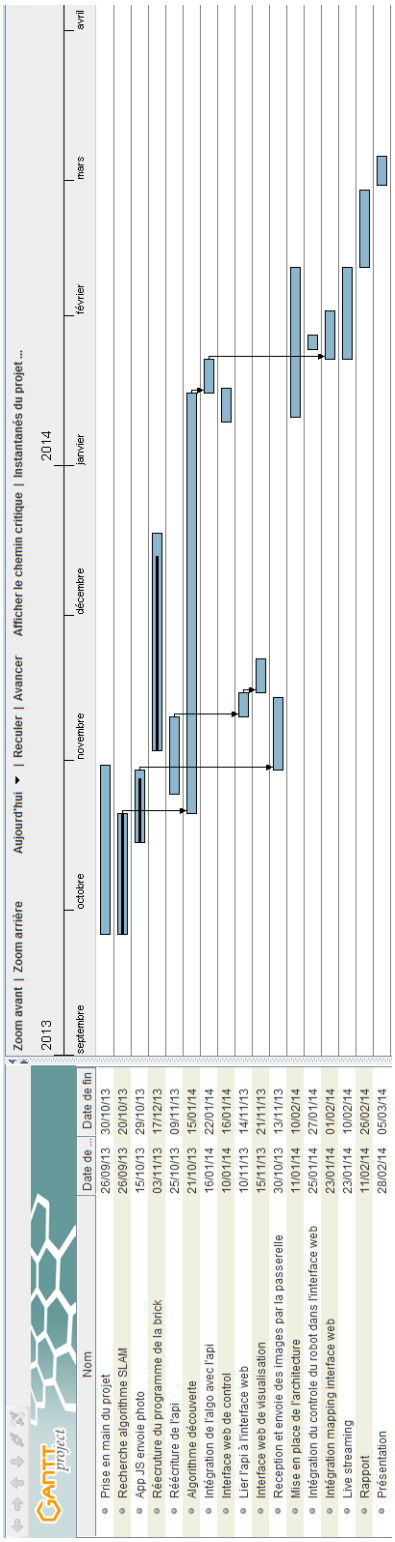
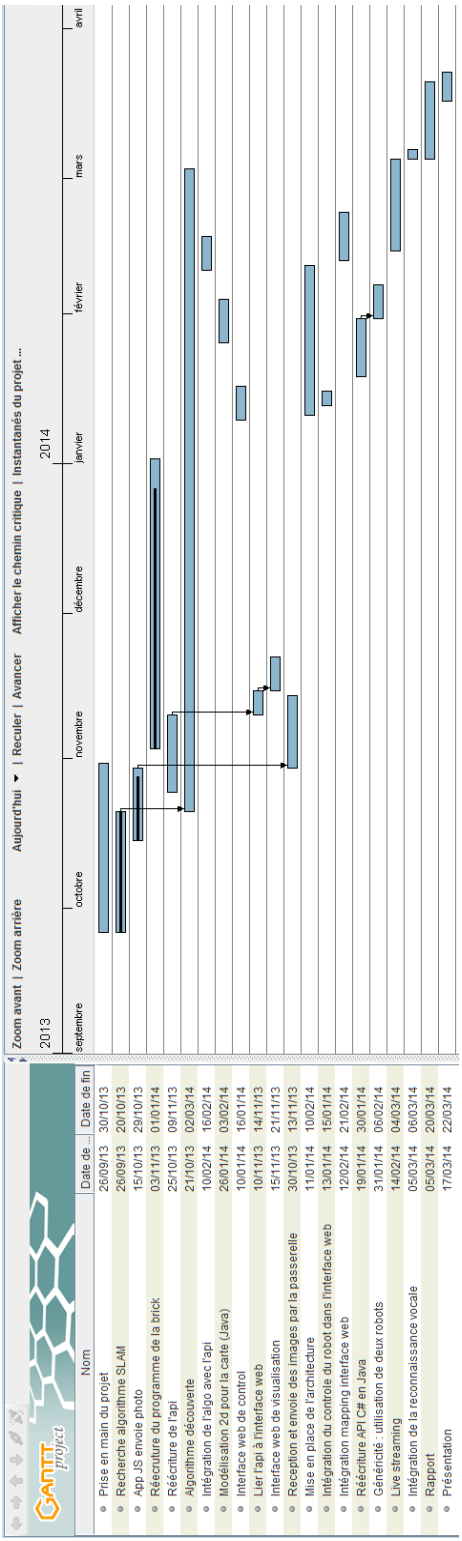
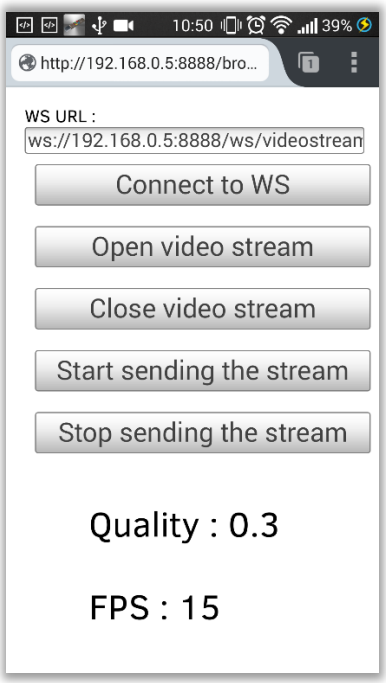


Diagramme de Gantt réel

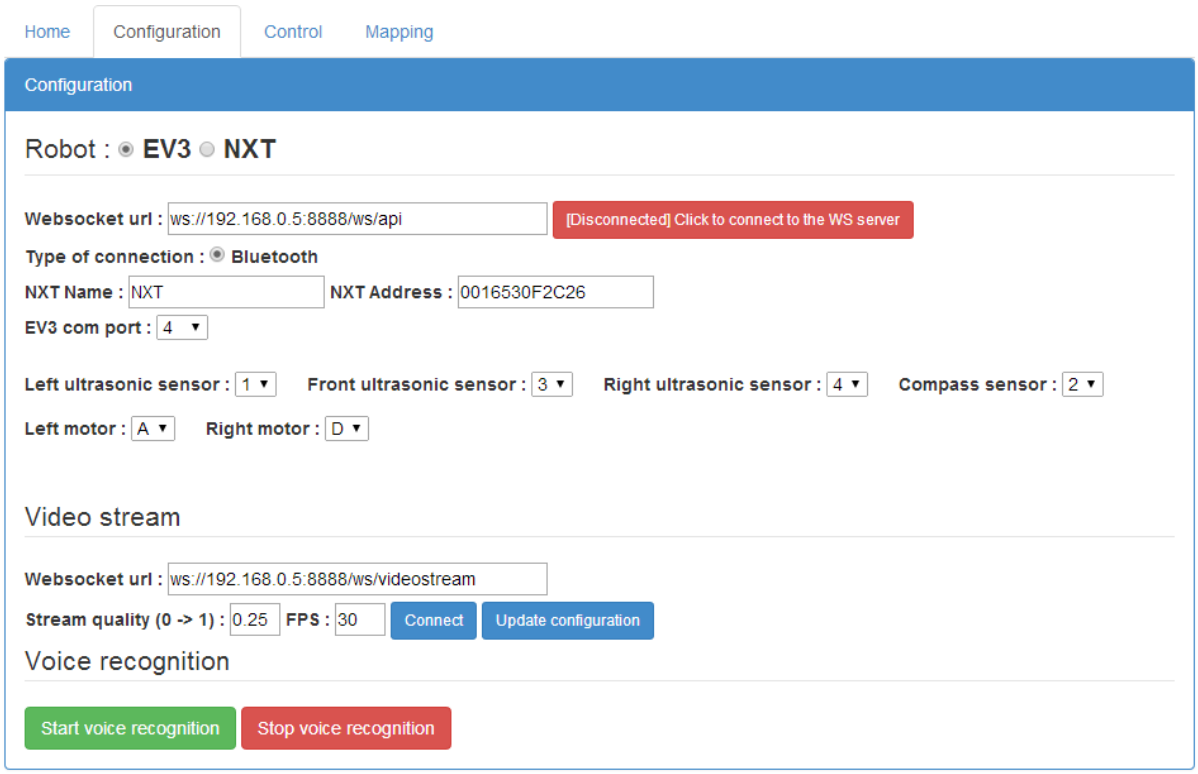


En comparant ces deux diagrammes de Gantt, nous pouvons remarquer plusieurs choses : de nombreuses tâches en plus ont été effectuées par rapport à nos ambitions initiales (API pour l'EV3 en Java, affichage intermédiaire de la cartographie directement en Java, mise en place de la « généricité » pour utiliser les deux robots), et l'algorithme d'exploration nous a pris beaucoup plus de temps que prévu.

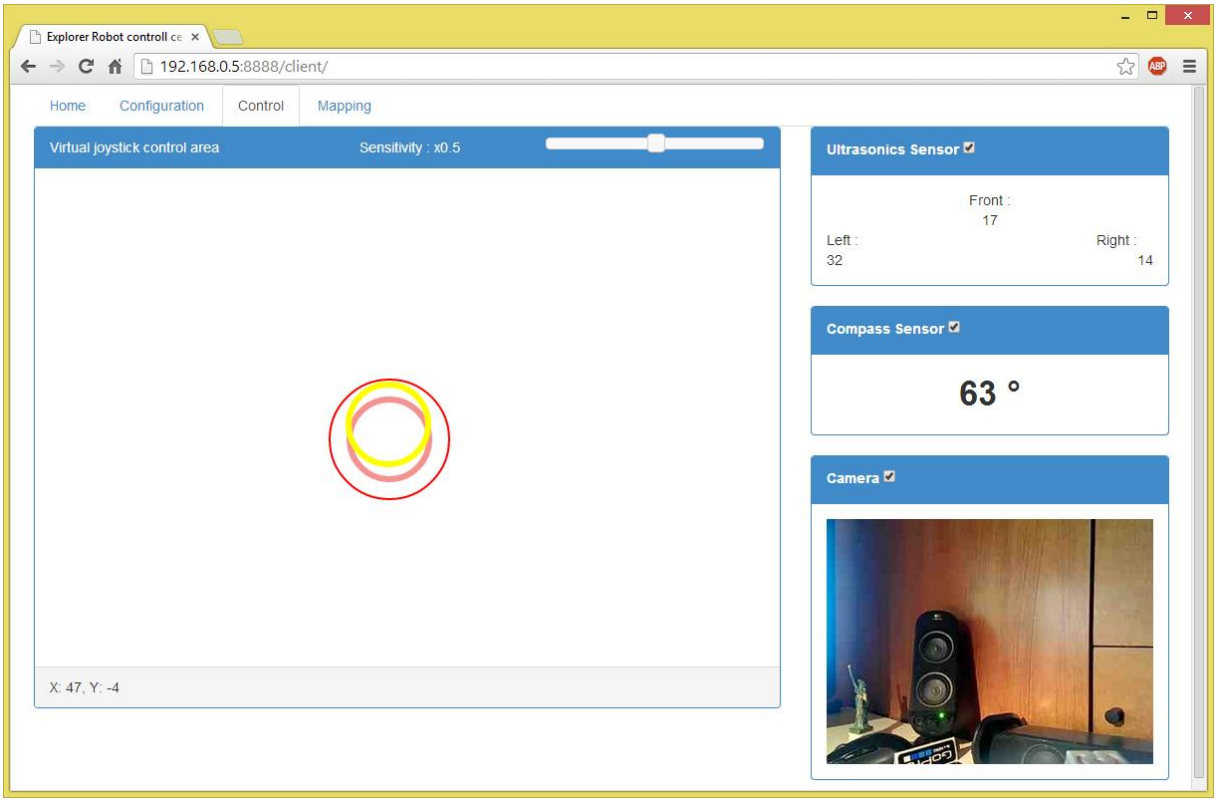
7.2. Interfaces utilisateur



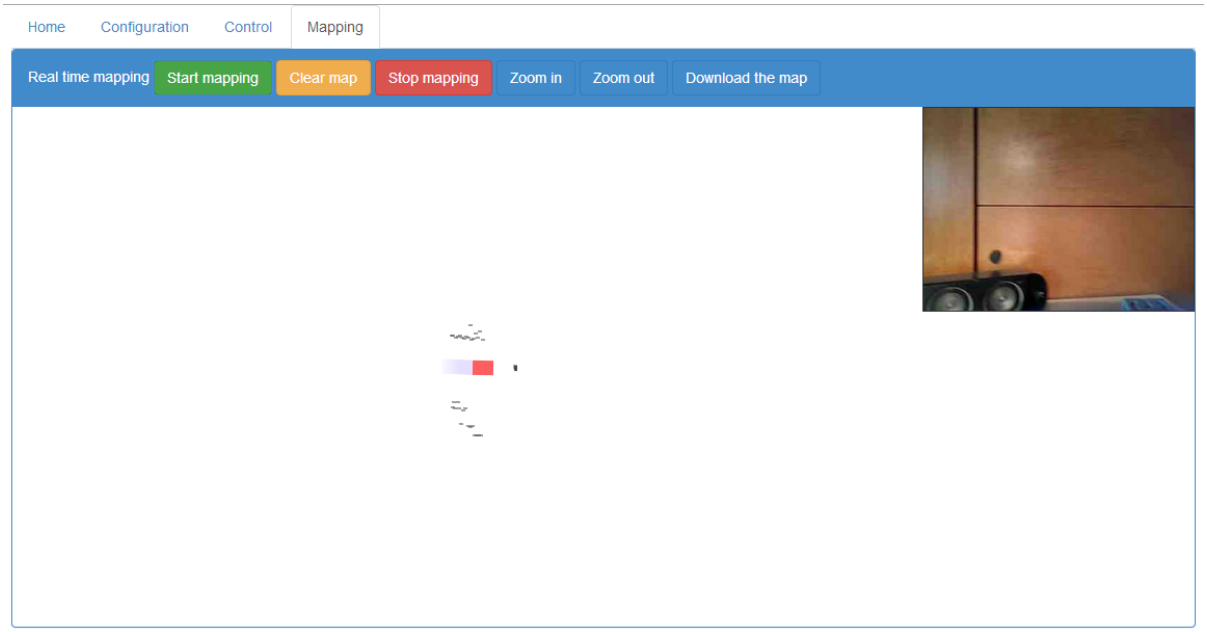
Application du smartphone pour l'envoi du flux vidéo



Panneau de configuration

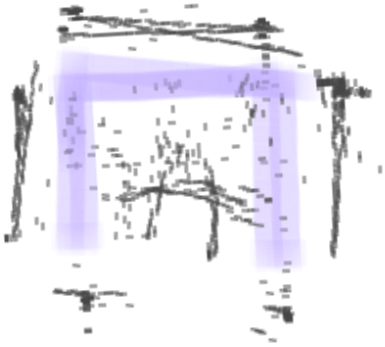
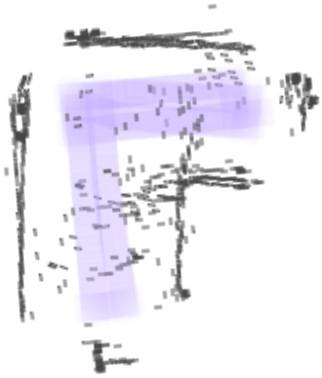


Interface permettant le contrôle via le joystick virtuel, l'affichage de la caméra et les valeurs des capteurs.



Visualisation de la carte générée avec la caméra

7.3. Résultats obtenus

Environnement réel	Représentation 2D
	
	

7.4. Innorobo

Photo prise par M. MRISSA, notre tuteur, au salon Innorobo, sur le stand où nous avons présenté notre projet et fait des démonstrations.