POLYTECH ANNECY-CHAMBÉRY

22 MAI 2018

PROJET CUSTODIA

APP RS-OC 2020 – SEMESTRE 6

FRANCK BATTU – TRISTAN GRUT – FABIEN LALANDE – THIBAUD MURTIN –
JOHANN PISTORIUS – ERWAN PROSPERT – ADRIEN RYBARCZYK
POLYTECH ANNECY-CHAMBERY

Table des matières

1. Introduction

Dans le cadre du module d'APP, nous avons choisi le thème de Robotique de Service et Objets Connectés (RS-OC). Avant de commencer à détailler l'ensemble de notre projet, nous allons vous expliquer en quoi consiste réellement l'APP.

Les principaux objectifs du module d'APP sont dans un premier temps, de développer nos compétences, d'améliorer nos compétences acquises lors de notre formation au sein de l'école mais d'aussi apprendre de nouvelles compétences dans des thèmes en lien avec notre formation (Santé / IE : Imagerie pour l'Environnement / RS : Robotique de Service / RS-OC : Robotique et Objets connectés /B I : Bâtiment Intelligent).

Dans un second temps, le projet d'APP nous permet d'être en situation réaliste proche de l'entreprise sur un travail à long terme afin d'apprendre le travail et la communication en équipe. Nous apprenons donc à travailler en autonomie avec des rôles particuliers importants (Chef de projet, animateur, secrétaire, responsable de sécurité). Nous avons donc pu développer notre organisation au sein d'une équipe de travail grâce à des réunions et l'élaboration de compte rendu à chaque fin de séance.

Ensuite, l'APP nous a permis de comprendre ce qu'est la gestion de projet avec la définition du cahier des charges et avec la mise en place d'une organisation de travail en planifiant et en se répartissant les différentes tâches grâce à différents outils (GANTT, bête à corne, etc.). Dans ce projet, nous avons à disposition des experts qui nous aident à nous aiguiller sur nos choix et les valider et ils nous apportent différentes connaissances sur des domaines que nous ne maitrisons que peu ou pas du tout. On a pu découvrir aussi les aspects économiques qu'impliquent la consultation des experts avec la rémunération en jeton.

De plus, durant ce projet, l'apprentissage et la documentation autonome sont des parts très importantes pour la réussite du projet. On a donc finalement une expérience similaire au travail d'ingénieur en entreprise.

Lors de ces 4 semestres de travail, notre équipe travaillera sur le thème de la Robotique de Service et des Objets Connectés. Par la suite, nous allons donc vous présenter notre projet **Custodia**. Nous allons vous présenter tout d'abord le cahier des charges de notre projet, puis la façon dont l'équipe s'est organisée, les problèmes rencontrés mais aussi la manière dont on les a résolus.

2. Présentation du matériel

Tout au long de notre projet, nous allons utiliser Robotino comme support matériel.

Robotino est un système robotique mobile disposant d'un système de déplacement omnidirectionnel, permettant de se déplacer dans toutes les directions. Il est muni de plusieurs types de capteurs (analogiques, binaires, et numériques) et dispose de différents ports permettant de raccorder des capteurs et actionneurs supplémentaires.
Robotino dispose d'un ordinateur de bord avec le système d'exploitation Linux. La programmation de Robotino est possible dans différents langages (C++, Java). Dans notre cas, nous allons utiliser le Java.

Robotino est très utilisé dans des formations d'informatique et de robotique. Il permet principalement d'apprendre à programmer complètement un robot.

Lors de ce projet, nous avons choisi Robotino pour sa flexibilité et sa simplicité de programmation.

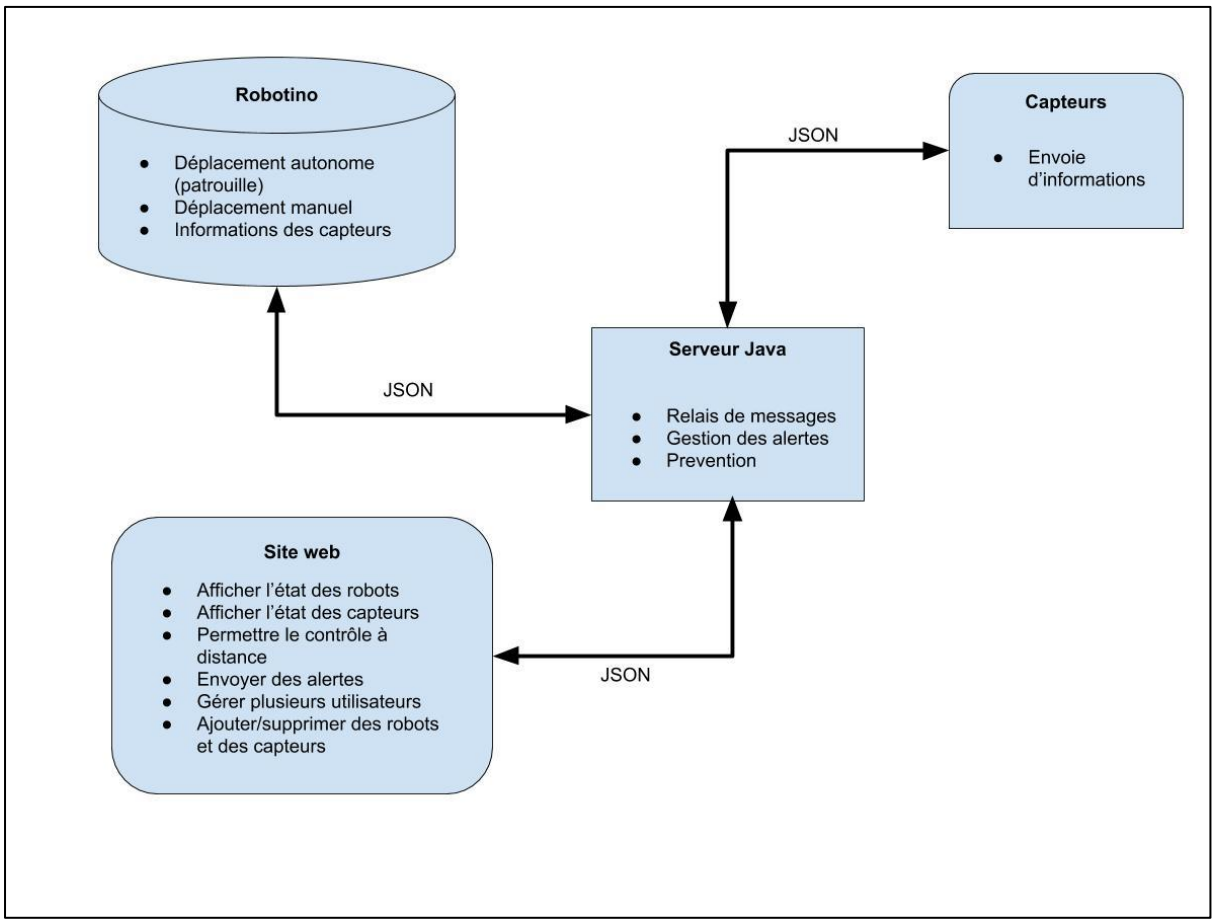
3. Cahier des charges

Pour ce projet APP en Robotique de Service et Objets Connectés 2020, le besoin principal du client est de permettre la surveillance d'un lieu à l'aide de plusieurs capteurs et de plusieurs robots. Ensuite, le client doit pouvoir accéder à l'ensemble de ces informations à l'aide d'un site web. Celui doit permettre de récupérer et visualiser les informations des capteurs et des robots, ainsi que permettre la patrouille de ces derniers et leurs interventions sur des lieux à l'aide des informations des capteurs. Il doit y avoir deux modes distincts : un mode automatique sous forme de patrouille, et un mode manuel où le robot est dirigé grâce à un joystick disponible sur le site web.

3.1. Objectifs

Notre principal objectif est de mettre en place une communication entre les différents éléments du projet. Nous avons donc décidé de créer un serveur en Java qui permettra d'échanger entre le robot et l'interface web. Le robot doit pouvoir remonter des informations au site web, et vice-versa, par l'intermédiaire du serveur Java. Concernant le robot, il devra être capable de se déplacer dans un lieu inconnu et détecter des obstacles grâce à un système de capteurs. Toutes ces informations seront envoyées au serveur pour être ensuite traité par le serveur, qui affichera les différentes informations du client. Cette interface web permettra également de contrôler le robot grâce à un joystick.

3.2. Schéma fonctionnel



3.3. Les fonctionnalités du projet

Les fonctionnalités prévues pour le projet sont les suivantes :

Navigation du robot :

- Navigation autonome
  - Patrouille dans une zone définie
  - Vérification de marqueurs d'alertes
- Communication
  - Statut des capteurs sur Robotino

Capteurs :

- Remontée des informations

Serveur Java :

- Communication
  - Relais des messages aux bons acteurs
  - Envoie d'alertes
  - Envoie d'informations au site web
  - Envoie d'ordre à Robotino
- Logique
  - Gestion des marqueurs d'alertes, priorité

Site web :

- Interface homme-machine
  - Contrôle de Robotino
  - Définition de zones de patrouilles
  - Affichage des informations des capteurs
  - Gestion de différents paramètres d'alertes
- Contrôle d'accès, de droits

Base de données :

- Enregistrement des utilisateurs et de leurs droits
- Enregistrement des capteurs
- Enregistrement des zones

4. Gestion du projet

4.1. Organisation

Tout d'abord, nous avons réfléchi aux grands axes du projet : objectif, sous-partie, calendrier assez large (au niveau du semestre). Il a été décidé de scinder notre groupe en différents sous-groupes qui travaillent chacun sur une partie spécifique du projet :

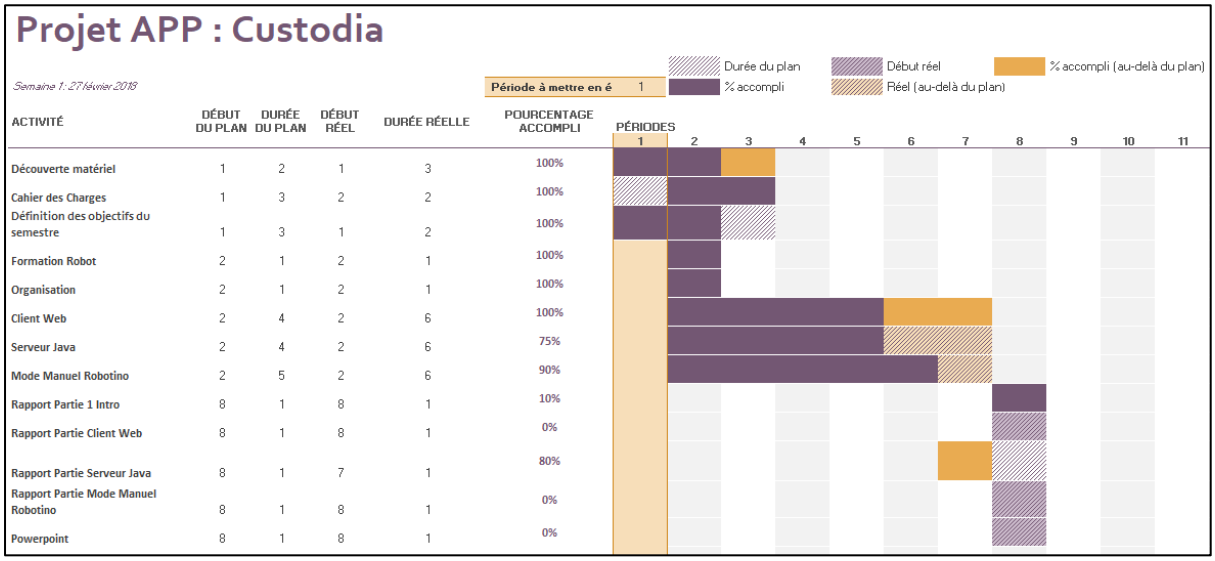
- Un groupe sur le serveur Java
- Un groupe sur la partie web
- Un groupe sur la partie Robotino

La gestion de la base de données a été mutualisé car plusieurs groupes en avaient besoin. Cette organisation nous a permis de mieux nous repartir les tâches et nous concentrer sur des buts précis. Le calendrier a été établi très largement avec des réalisations à faire pour le semestre, mais la longueur des tâches ou la gestion précise du temps de travail car cela a été délégué au groupe.

Une fois le travail commencé, nous avons tenté de mettre l'accent sur la communication interne : chaque séance d'APP a été l'occasion de prendre des nouvelles et de s'organiser entre les groupes ainsi que de vérifier l'avancement par rapport aux objectifs finaux.

4.2. Outils

Nous avons réalisé un GANTT, mais nous ne l'avons pas réellement utilisé au cours de ce semestre :



Nous avons ensuite mis en place un serveur de chat vocal et texte sur Discord afin de fluidifier les échanges. Cela a été choisi car la plupart des membres de notre groupe était plus présent sur ce réseau que sur Facebook ou autre. Il a toutefois été peu utilisé puisque l'on se croisait suffisamment souvent dans les différents cours pour pouvoir discuter rapidement.

Enfin, nous avons mis en place un GitHub, qui est un outil de versioning s très efficace pour les projets informatiques, afin de pouvoir y stocker les développement réalisés. Nous y avons mis 4 branches : une par groupe ainsi qu'une branche pour la "production". Ce GitHub sera passé à la fin du projet. Nous avons également mis en place un espace de stockage en ligne sur Google Drive.

5. Partie Java

Le semestre 6 est axé sur la découverte de l'environnement d'Apprentissage par Problème et par Projet ainsi que du matériel mis à disposition.
Afin de faciliter les échanges entre le robot, les capteurs et l'interface client, nous avons décidé de mettre en place un serveur Java.
La tâche qui a été assignée à Erwan, Fabien et Tristan consistait ce semestre à étudier la mise en place du serveur java central qui doit pouvoir transférer et gérer les connections entre les différents éléments du projet.

5.1. Organisation

Fabien était le codeur principal. C'est lui qui a écrit la plupart du code du serveur Java et l'a testé. Tristan et Erwan réalisaient et faisaient des modifications mineures en plus de tester et créer les liens avec le reste du projet.
Les objectifs de développement ont été donné au fur et à mesure de l'avancée du projet tout en tentant de garder une vision globale d'intégration avec le reste de l'APP.

5.2. Réalisations

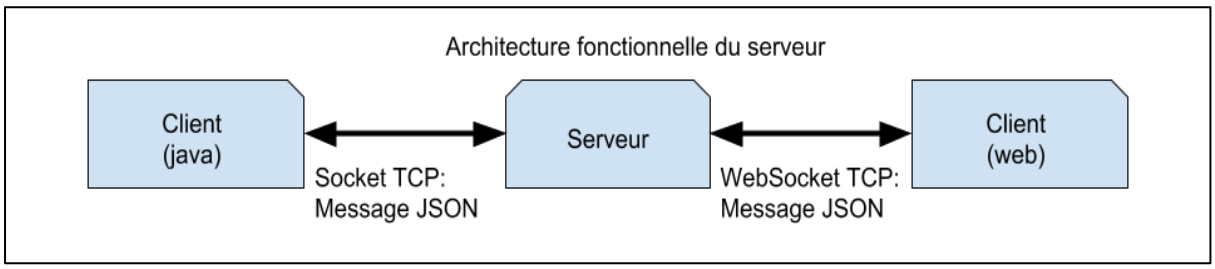
Ce semestre, le petit groupe a conçu un serveur java opérationnel répondant aux points suivants :

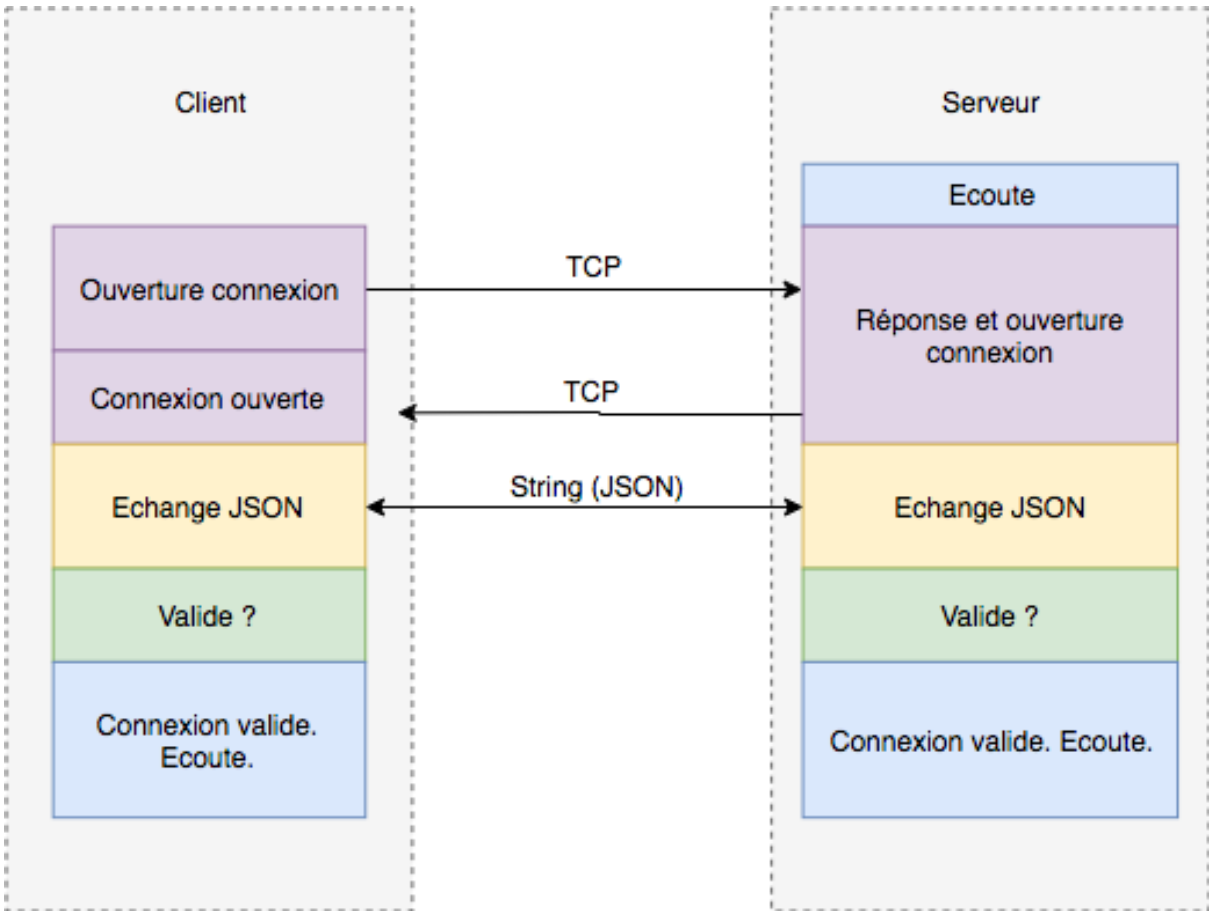
- Capacité à gérer de façon asynchrone plusieurs connexions java-socket <> java-socket
- Capacité à gérer de façon asynchrone plusieurs connexions java-socket <> web socket
- Capacité à traiter les messages JSON
- Différents traitements selon le contenu du message
- Ouverture de connexion
- Connexion simple entre le site et le serveur (Client Robotino.java) permettant de recevoir et d'envoyer des informations

Cependant, nous n'avons pas pu effectuer un test complet de communication avec les autres parties du projet, ni même faire une documentation complète du serveur.

5.3. Architecture

Actuellement le projet est organisé autour d'une douzaine de classes. Trois classes sont dédiées à créer un client pour le test et en tant qu'exemple pour les autres groupes. Le reste des classes est commun et/ou dédié au serveur. Nous avons fait le choix d'un modèle sans évènement mais multithread en utilisant les sockets de java.






```
//Connexion Client > Server
//Message envoyé par le client au serveur
{
   "type":"init",
   "infoInit":"Client-->Server  demande de connexion",
   "clientName":"NameOfClient",
   "clientType":"autre",
   "mdp":"PasswordOfClient"
}

//Connexion Server > Client
//Message envoyé par le serveur au client
{
   "type":"init",
   "infoInit":"Server->Client Connexion accepté",
   "serverName":"NameOfServer"
}
```

Un client se connecte au serveur qui écoute et gère les connexions. Le serveur et le client échangent des informations d'initialisation ou *drop* la connexion. Une fois ces informations échangées, la connexion est considérée comme valide et on peut échanger des messages sous format JSON. Un exemple des messages d'initialisation est donné ci-dessus.

5.4. Problèmes rencontrés

Pour la partie Java Server, nous avons dû énormément nous documenter. Ce semestre, une grande partie du temps a été utilisée dans la documentation. En effet, ce sujet étant nouveau pour certaines personnes de l'équipe, il a fallu découvrir cet aspect non abordé durant les cours.
Pour la partie programmation, dans un premier temps, nous avons eu un problème de socket. En effet, après le lancement du serveur, les sockets restaient ouvertes même après la fermeture du programme.

Le site web utilise des web sockets « protocole http » et le serveur java des java-sockets « protocole TCP ». Il a fallu établir une connexion et des échanges de données entre ces différentes sockets. La connexion http utilisant une connexion TCP, ainsi, il a fallu faire quelques modifications dans la partie java pour répondre et écouter une connexion http.

5.5. Conclusion, réflexion et perspectives

Ce semestre le groupe a réalisé de belles avancées. Nous avons appris à manipuler le type *runnable* et les sockets de java ainsi que les librairies externes. Le cours de programmation orienté objet en Java nous a bien aidé, mais est arrivé un peu tard pour certaines parties (notamment le modèle à évènement qui est arrivé un peu tard).
Toutefois, nous avons fait preuve d'un manque d'organisation et de répartition du travail en interne : le code a été créé au fur et à mesure sans prévoyance. Certaines classes sont trop chargées et la documentation manque. Nous n'avions aussi pas prévu le support pour les web sockets, utilisable par l'interface web et seule solution pour la transmission d'information et il a fallu adapter le code pour les gérer.
Pour le prochain semestre, il est prévu de reprendre ou réécrire une grande partie du serveur en "prenant notre temps", c'est-à-dire en se répartissant le travail, en créant un diagramme UML et en réfléchissant un peu mieux aux différentes choses dont on va avoir besoin. On sait notamment qu'il faudra intégrer les capteurs et la connexion avec ceux-ci. Il faudra notamment revoir les classes pour les décharger un peu et rendre le tout plus lisible.

6. Partie Web

L'objectif principal de ce semestre est la mise en place d'une interface web responsive permettant de contrôler manuellement Robotino. A long terme, ce site web sera utilisé pour contrôler l'ensemble du projet, y compris les divers capteurs prévus.

6.1. Réflexions initiales

Plusieurs pages ont ainsi été prévues initialement :

- Présentation du projet
- Retour vidéo de Robotino et contrôle manuel (joystick)
- Liste des capteurs disponibles avec interaction
- Carte interactive de Robotino

Pour réaliser un site web, plusieurs possibilités s'offrent à nous : soit le réaliser en "statique" avec HTML/CSS/JS, ou alors le réaliser en "dynamique" en ajoutant du PHP. Comme notre but est d'obtenir une gestion complète du projet depuis le site web, nous avons besoin d'un accès à une base de données pour récupérer les capteurs disponibles, ou encore, pour mettre en place un espace membre afin de sécuriser le site. Nous avons donc choisi de réaliser ce site en PHP.

6.2. Problèmes rencontrés

Initialement, nous souhaitons utiliser le framework PHP Symfony 3.4 de SensioLabs afin de faciliter le développement du site web en backend, et le framework Bootstrap de Twitter pour faciliter le frontend.

Pour ce faire, nous avons dû recourir à une machine virtuelle (VM) sous Ubuntu afin de pouvoir installer la version de PHP que nous avions besoin. En effet, le serveur interne de Polytech propose la version 5.2.11 de PHP, alors que nous avions besoin de la version 7.2.

Nous avons rencontré quelques difficultés pour installer la VM. En effet, la VM doit être installé sur un disque particulier de l'école, sinon elle risque d'être effacée. Ignorant cela, notre première VM a fini par être effacée, et nous avons dû en recréer un, perdant ainsi tout ce que nous avions mis dessus. Nous avons eu également quelques problèmes au niveau de la performance de la VM. En effet, nous n'avions pas assez alloué de RAM et de cores, ce qui faisait souvent saturer la VM, nous empêchant de télécharger les outils nécessaires et ainsi de travailler convenablement.

Nous avons également une autre difficulté qui nous a forcé à changer notre manière de développer : l'hébergement. En effet, nous n'avions pas pensé initialement que ce site pourrait être héberger au sein du serveur de Polytech. Malheureusement, lors de la création du site vitrine, nous avons appris que nous ne pourrions pas héberger notre site au sein du serveur de Polytech car notre site utilisant une version différente de PHP de celle de l'école et utilisant un framework PHP. Comme le site ne contenait alors que quelques pages, et que la base de données n'était pas encore créée, nous avons pris la décision d'arrêter le développement de ce site en utilisant Symfony, et de recommencer en utilisant la version de PHP disponible à Polytech. Ce nouveau développement a toutefois été assez rapide, car il suffisait d'adapter les pages déjà créées.

6.3. Mise en place d'un Joystick virtuel :

Le but de ce semestre était d'avoir une page web permettant de contrôler le robot. Nous avons décidé d'utiliser un joystick virtuel qui permettrait de déplacer Robotino dans toutes les directions.

Pour le joystick, après avoir tester plusieurs librairies, nous avons décidé d'utiliser la librairie *nippleJS* disponible cette adresse : https://github.com/yoannmoinet/nipplejs. Une fois implémenté sur le site web, ce joystick nous permet de récupérer plusieurs valeurs, comme par exemple :

- La position x et y du curseur
- La force *(plus la souris s'éloigne du centre, plus la force est importante)*
- L'angle entre le curseur et le centre *(en radians ou en degrés)*
- La direction de x, y, et de l'angle

Toutes ces informations sont utiles pour le déplacement de Robotino.

Nous avons également la possibilité de contrôler la récupération de ces informations. En effet, nous disposons de deux modes :

- Récupération de données à chaque mouvement du joystick
- Récupération de données quand le joystick dépasse un seuil :

```
//      \  UP /
//       \   /
// LEFT        RIGHT
//       /   \
//      /DOWN \
```

```
   UP                |
------         LEFT | RIGHT
 DOWN                |
```

Nous pouvons soit récupérer les données quand le joystick dépasse une diagonale (schéma de gauche), ou quand le joystick dépasse un plan horizontal ou vertical. Nous avons choisi dans notre cas le dépassement diagonal.

6.4. Communication avec un serveur Java :

Maintenant que nous pouvions disposer d'un joystick, il fallait envoyer les informations au serveur Java, afin de faire déplacer Robotino. Pour ce faire, nous avons mis en place un système de Web Socket en JavaScript. Le principe de conception est assez simple : une fois la page chargée, on ouvre une socket vers l'IP du serveur.

```
const socket = new WebSocket("ws://193.48.125.70:50007");
```

Une fois cette socket ouvert, il est possible par exemple d'écouter les événements suivants :

onopen	La connexion est prête à recevoir et émettre des données
onmessage	Un message a été émis par le serveur
onerror	Une erreur est survenue
onclose	La connexion a été fermée

Pour envoyer un message au serveur, il suffit d'employer la méthode **send**. Nous avons décidé que toute information envoyée au serveur et reçue par le serveur devait être en format JSON, car il s'agit d'un format universel et facile à lire ou écrire.

Pour envoyer les informations du joystick au serveur, on utilise la méthode suivante :

```
manager.on('dir', (event, data) => {
    let manuel = document.querySelector('#manuel');
    if (manuel.checked && serverOpened) {
        let json =
            {
                "type" : "commandeRobotino",
                "commande" : "setPositions",
                "data": {
                    "x": data.position.x,
                    "y": data.position.y,
                    "angle": {
                        "degree": data.angle.degree,
                        "radian": data.angle.radian
                    },
                },
                "destinataire" : {
                    "name" : "Server Robotino v1",
                    "ip" : "193.48.125.70:50007"
                },
                "expediteur" : {
                    "name" : "C1",
                    "ip" : "0.0.0.0"
                }
            };
        socket.send(JSON.stringify(json));
    }
});
```

Ici, **manager** représente le joystick. A chaque fois que le joystick change de direction diagonale, nous créons un JSON contenant les informations que l'on souhaite envoyer (les différentes caractéristiques permettant de caractériser une position précise). De plus, on spécifie le type de données que l'on envoie car il y aura différents envois (connexion, déconnexion). On envoie ensuite ce JSON au serveur sous la forme d'une chaîne de caractères, car l'envoi entre serveur et client ne peut se faire qu'en chaîne de caractères. Il suffit alors côté serveur de traduire cette chaîne de caractères en JSON à l'aide de méthodes appropriées.

Pour tester l'implémentation de ces sockets, nous avons créé un petit serveur Java utilisant les Web Sockets. Ce serveur ne peut pas être utilisé dans le projet final car il est construit en Web Socket, et l'utilisation de Robotino nécessite des Sockets. Mais, il est possible de faire un pont en Java entre les Web Socket et les Sockets.

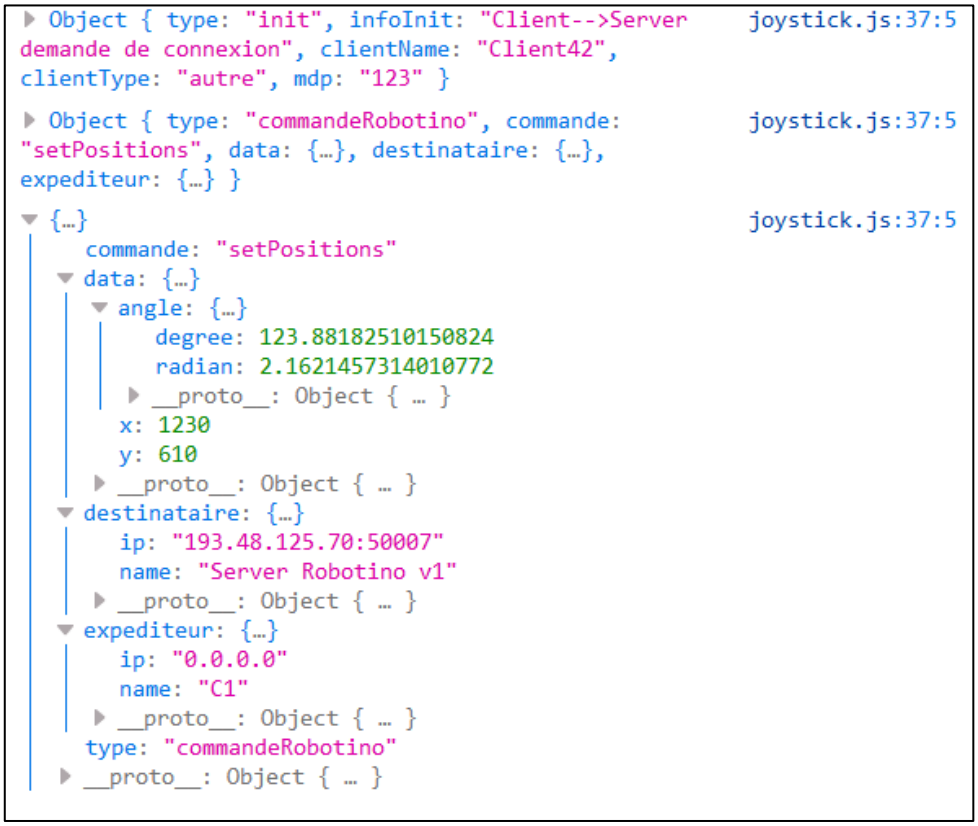
```java
public class WebsocketServer extends WebSocketServer {

    private static int TCP_PORT = 8888;

    private List<WebSocket> conns;

    public WebsocketServer() {
        super(new InetSocketAddress("193.48.125.64", TCP_PORT));
        this.conns = new ArrayList<WebSocket>();
    }

    public void onOpen(WebSocket webSocket, ClientHandshake clientHandshake) {
        conns.add(webSocket);
        System.out.println("Nouvelle connexion : " + webSocket.getRemoteSocketAddress().getAddress().getHostAddress());
    }

    public void onClose(WebSocket webSocket, int i, String s, boolean b) {
        conns.remove(webSocket);
        System.out.println("Connexion fermée : " + webSocket.getRemoteSocketAddress().getAddress().getHostAddress());

    }

    public void onMessage(WebSocket webSocket, String s) {
        System.out.println("Message du client :" + s);

        for (WebSocket sock : this.conns) {
            sock.send(s);
        }
    }

    public void onError(WebSocket webSocket, Exception e) {
        if (webSocket != null) {
            this.conns.remove(webSocket);
        }
        System.out.println("Erreur : " +  webSocket.getRemoteSocketAddress().getAddress().getHostAddress());
    }
}
```

Ce serveur n'a pas de réelle utilité car il renvoie la même donnée qu'il a reçue, mais on peut imaginer divers traitements possible. Nous souhaitions uniquement disposer d'un petit serveur Java pour tester notre application, le temps que le vrai serveur du projet soit finalisé.

Du côté du client Web, lorsqu'un message est reçu du serveur, on le traduit en JSON et on l'affiche. Pour le moment, on ne se contente que de l'afficher, mais on peut imaginer effectuer un traitement dessus ou filtrer les différents messages que l'on reçoit.

```
socket.onmessage = (event) => {
    let data = JSON.parse(event.data);
    console.log(data);
}
```

Pour tester, lorsqu'on fait bouger le joystick, on obtient côté client le message suivant :



Ce message correspond en réalité au message reçu du serveur une fois traduit en JSON.

6.5. Conclusion

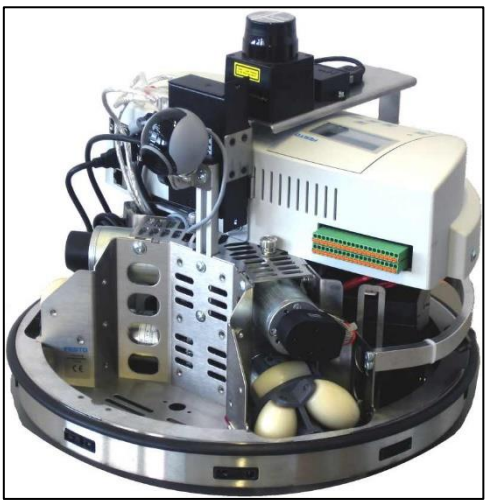
Cette application web est amenée à être modifié au fil du projet. Pour l'instant, son utilisation reste assez basique car elle permet simplement de communiquer à un serveur Java des informations pour contrôler Robotino manuellement. Cependant, la mise en place de cette application nous a permis de découvrir de nouvelles choses, comme notamment les sockets, que nous ne connaissions pas du tout. Les différents problèmes que nous avons rencontrés nous ont permis de prendre du recul et de réfléchir à l'implémentation de notre code. Par la suite, nous allons essayer de plus réfléchir sur comment implémenter notre code, les problèmes que l'on peut rencontrer et les conséquences que l'on peut obtenir.

Le code est disponible sur GitHub à l'adresse suivante :

https://github.com/APP-RS-OC-Polytech-2020/APP-RS-OC-Polytech-2020 (branche web pour l'application web et branche web socket pour le serveur Java).

7. Partie Robotino

Dans cette partie, nous étions chargés de prendre en main Robotino, découvrir son fonctionnement et être capable de programmer des déplacements. Au départ, nous avons consacré beaucoup de temps à la recherche pour voir ce qui se faisait avec Robotino. Pour cela, nous avons regardé ce qui a été fait dans les anciens projets, les codes trouvés sur internet ou encore ceux des exemples fournis avec l'API et ainsi trouver et comprendre les fonctions nécessaire au fonctionnement de Robotino.



Le groupe pour la partie robot, composé de Thibaud Murtin et Johann Pistorius, avait pour but de programmer le comportement du robot et de se renseigner sur ses capacités et son fonctionnement.

7.1. Matériel

Nous avons choisi de coder les fonctionnalités robot avec le langage de programmation JAVA, qui possède une API et divers exemples. Robotino possède 3 roues omnidirectionnelles, des capteurs infrarouges, des bumpers et plusieurs ports permettant un possible ajout de cartes électroniques, caméra ou autres capteurs.

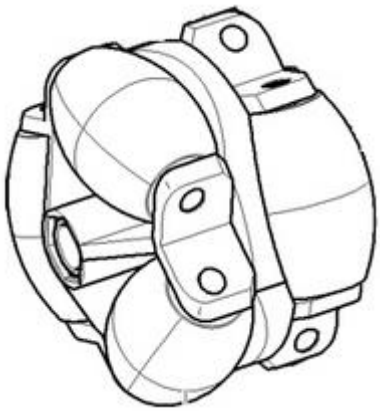


Image 2 - Roue omnidirectionnelle

7.2. Connexion au robot

```
ArrayList<String> hostname=new ArrayList<String>();
hostname.add("193.48.125.37");
hostname.add("193.48.125.38");
for(String host:hostname) {
        Robot r=new Robot(System.getProperty("hostname", host));
        r.start();
        if(r.isConnected()) {
                r.run();
                //Fenetre win=new Fenetre(r);
                break;
        }
}
```

Image 3 - Main

Pour se connecter au robot, nous avons mis toutes les adresses IP des robots dans des listes. On parcourt la liste et on teste chaque adresse pour voir si le robot est connecté. Si la connexion est un succès, on fait avancer le robot.

```
public void start()
{
    System.out.println("Robot started.");

    try
    {
        System.out.println("Initializing...");
        init();
        System.out.println("Connecting...");
        connect(hostname);
        System.out.println("Connected.");
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        disconnect();
    }

    System.out.println("Done.");
}
```

Image 4 - Fonction start

Ci-dessus, la fonction de connexion au robot. Ici, on calibre les moteurs omnidirectionnelles en appelant la fonction init et on teste la connexion.

7.3. Déplacement manuel

Plusieurs tests de déplacement et d'utilisation des bumpers ont été fait. Cependant, nous n'avons pas utilisé tous les capteurs accessibles sur le robot.
Pour faire avancer le robot, nous avons implémenté une fonction drive qui récupère un entier x, un entier y, un entier angle et un booléen status.

```
protected void drive(int x, int y, int angle, boolean status) throws InterruptedException{

    while (!Thread.interrupted() && com.isConnected() && false == bumper.value() && status == true)
    {
        omniDrive.setVelocity(x, y, angle);
    }
}
```

Image 5 - Fonction drive - mode manuel

Dans la fonction, tant que la connexion au robot n'est pas interrompue, les bumpers n'ont pas été touché et le status est à vrai, on appelle omniDrive et on lui donne une vitesse.
Malgré l'implémentation de cette fonction, nous n'avons pas pu être testé en parallèle avec les autres parties, faute de temps.

7.4. Problèmes rencontrés

À la suite d'une désynchronisation entre les équipes, nous avons choisi d'implémenter une interface graphique en Java pour tester le mode manuel.

```
this.buttonR.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
                            robot.drive(0,50,0,true);
                    } catch (InterruptedException e1) {
                            // TODO Auto-generated catch block
                            e1.printStackTrace();
                    }
    }
});
```

Image 6 - Bouton droit action performed

Pour chaque bouton, on a implémenté un ActionListener. Lorsque le bouton est cliqué, on appelle la fonction drive qui permet de faire avancer le robot.

Un incident est survenu lors de l'avant dernière séance : les deux Robotino ne répondaient plus lorsque l'on essayait de les accéder. Leurs fichiers de configuration réseau ont été endommagé. Nous n'avons donc pu tester notre code durant cette séance et la séance suivante.

7.5. Projet futur

Nous pensons maintenant être capable d'intégrer un mode automatique au prochain semestre en utilisant un point de calibrage, et de faire un système de patrouille à Robotino.
Conceptuellement, nous avons choisi de lui donner une cartographie de son environnement.
Le déplacement sera géré par un parcours d'arbre en utilisant des algorithmes de recherche tels que Dijkstra ou A*. De plus, l'utilisation d'une carte Arduino équipée de capteurs et l'ajout d'une caméra est aussi prévue, certainement accompagné d'une Raspberry Pi.

Dans des idées de projets plus ambitieux, nous pensions aussi intégrer une intelligence artificielle qui ferait du « machine learning » ou « deep neural networks ».

8. Conclusion générale

Le projet Custodia a bien avancé ce semestre. Nous avons réalisé des avancées dans les différentes parties mais également fait quelques erreurs bénéfiques. La gestion du projet et la communication devront être revu ce qui devrait nous permettre de fonctionner plus efficacement l'année prochaine.

Les différents sous-groupes ont rencontré quelques gros problèmes. Le groupe web s'est battu un long moment avec l'hébergement pour le serveur web. Les version de PHP et les modalité ont été mal comprises et certaines solutions techniques (VM par exemple) nous ont ralenti. Il a finalement fallu recoder une grande partie du site. Le groupe Robotino a eu beaucoup de problèmes à prendre en main le robot, et une importante défaillance logicielle et matérielle nous a empêché de tester les applications sur le robot. Enfin, le serveur java est encore à un stade de bêta et demanderait un important *refactoring* pour pouvoir être pérenne.

Il est clair que ces différents problèmes auraient pu être réglé un peu plus rapidement avec une meilleure communication et une meilleure allocation du temps et des ressources. Nous avions aussi une disparité assez importante de niveau technique parmi les différents membres mais celle-ci s'est comblé au fur et à mesure de l'avancement du projet.

Enfin, nous avons réalisé que beaucoup des développements réalisés ce semestre sont incomplets ou peu aisés à maintenir. Cela est normal car le projet s'est un peu cherché et nous avons découvert et appris beaucoup de choses qui nous seront utiles dans le suite du projet.

9. Perspectives

Pour le prochain semestre, il va être nécessaire de revoir l'organisation. Nous comptons utiliser plus des outils comme le Gantt pour allouer le temps et permettre un meilleur déroulement du projet. Nous allons aussi mettre en place un calendrier partagé avec les dates importantes. Enfin, nous allons encourager une utilisation plus importante des différent canaux de communication (Discord, email) afin de tenir tout le monde au courant. Parmi les idées, nous pensons par exemple d'envoyer le compte-rendu de séance par mail à la fin des séances afin que tout le monde puisse le relire avant une prochaine séance.

Pour la partie technique, le site web sera complété avec différentes fonctions et messages à communiquer et envoyer au serveur ainsi qu'une intégration avec le site développé en INFO642. Le serveur java sera réécrit avec une meilleur direction et une documentation complète et la partie Robotino devrait intégrer les commandes venant de la partie web et le mode automatique. Nous allons aussi commencer l'intégration avec les capteurs et les objets connectés.