POLYTECH ANNECY-CHAMBÉRY

09 JANVIER 2019

PROJET CUSTODIA

APP RS-OC 2020 – SEMESTRE 7

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

POLYTECH ANNECY -CHAMBERY

TABLE DES MATIERES

INTRODUCTION

Le projet Custodia est un projet de télésurveillance robotisée en utilisant des objets connectés. Pour rappel, notre objectif final est l'interaction entre un robot de surveillance, notre fidèle Robotino, et des objets connectés, dans notre cas un système de capteurs. Le robot doit naviguer en autonomie dans une pièce, équipée de capteurs. Quand un capteur renvoie une donnée jugée anormale, Robotino se déplace vers le capteur, et on peut observer ce qu'il se passe à l'aide d'une caméra placée sur le robot.

Lors du semestre 6, nous avions avant tout découvert notre environnement de travail, et mis en place nos objectifs. Nous avions défini trois grands axes de travail : la construction d'un serveur en Java, l'élaboration d'un site web pour contrôler et visualiser Robotino, et le développement du déplacement de Robotino.

1. OBJECTIFS DU SEMESTRE

Un objectif important de ce semestre était de mettre en relation tout ce que nous avions développé pendant le semestre dernier. En effet, à cause d'un problème de configuration réseau sur Robotino, nous n'avions pu effectuer aucun test de notre travail. Avant de trop avancer sur le projet sans tester, nous voulions absolument effectuer cette mise en relation.

Pendant ce semestre, nous souhaitions également mettre en place notre système de capteurs afin d'intégrer la partie objets connectés dans notre projet. Les données renvoyées par les capteurs devront être affichées sur le site web par l'intermédiaire de notre serveur, et sauvegardées dans une base de données.

De plus, nous voulions également mettre en place un refactoring de notre serveur actuel afin qu'il soit d'avantage maintenable, et également générer de la documentation pour chacune de nos parties (code du site web, de la navigation du robot, des capteurs et du serveur).

2. REPARTITION ET MANAGEMENT

Les responsables de ce semestre ont été :

- **Chef du groupe** : Franck Battu
- **Responsable sécurité** : Johann Pistorius & Adrien Rybarczyk
- **Scribe** : Tristan Grut

Nous nous sommes de nouveau répartis en plusieurs groupes, qui sont :

- **Team Robotino** : Johann Pistorius & Adrien Rybarczyk
- **Team Capteurs** : Thibaud Murtin, Tristan Grut (& Franck Battu)
- **Team Serveur** : Erwan Prospert & Fabien Lalande
- **Team Web** : Franck Battu

Ce semestre, nous ne souhaitions pas répéter les mêmes erreurs effectuées au semestre précédent, à savoir :

- Mauvaise répartition du travail entre les équipes
- Manque de communication entre les membres
- Projets parfois trop ambitieux dans les équipes
- Mauvaise gestion du planning

Pour résoudre ces problèmes, nous avons notamment décidé de mettre en place des réunions de 10-15 minutes à chaque séance pour voir l'avancé de chaque groupe, le travail à effectuer pendant la séance, les difficultés rencontrées, les solutions envisagées, etc. Nous avons constaté que ces réunions nous permettaient de progresser plus vite, et de résoudre plus facilement les problèmes rencontrés.

3. PARTIE SERVEUR

3.1. OBJECTIFS

De façon générale, notre objectif de ce semestre pour l'équipe serveur était de faire de la maintenance et de l'ajout de fonctionnalités pour les autres équipes.

- Éclaircissement du code / refactor
- Gestion du flux de la webcam
- Création d'un client de distribution de flux webcam
- Passage du code sous Maven

3.2. TRAVAIL EFFECTUE

Pendant ce semestre, nous avons pu développer les points suivants :

- Réception et envoie des données des capteurs vers la page web
- Refactor le serveur pour qu'il soit plus compréhensible et facile à maintenir
- Test du serveur
- Création d'un client de distribution webcam, d'abord sur OpenCV, puis avec WebCam API
- Prise en main de Maven pour gérer les dépendances
- Création d'une JavaDoc

Toujours dans l'objectif de simplifier la maintenance, nous avons pris le temps de documenter le code, notamment avec l'ajout d'un début de JavaDoc (qu'il reste à compléter et générer ensuite) ainsi que le passage sous le gestionnaire de dépendances Maven. Maven est particulièrement utile pour gérer les dépendances car il se charge tout seul d'aller chercher la version du paquet qui correspond ainsi que toutes les dépendances éventuelles de ce paquet.

3.3 PROBLEMES RENCONTRES

3.3.1. CREATION D'UN CLIENT DE DISTRIBUTION DE FLUX VIDEO

Après avoir mis en place le contrôle à distance du robot, nous nous sommes rendu compte qu'il était nécessaire de *voir* ce qu'on faisait avec le robot. Nous avons donc opté pour la mise en place d'une webcam qui serait sur le robot et connectée au serveur. Le but de cette webcam est de permettre la téléprésence mais aussi un traitement d'image pour détecter des intrus automatiquement. De même, elle sera aussi utilisée pour essayer de se géolocaliser dans l'espace.

Les critères que nous nous sommes fixés pour ce logiciel sont:

- Distribution rapide des images (quasi temps-réel)
- Ressources matérielles minimales

- Indépendances de plateforme
- Résilience aux erreurs

Première itération

Cette itération tentait d'implémenter Webcam API par *sarxos*[1], mais nous nous sommes vite heurtés à notre absence de connaissance et au manque d'exemples et de documentation. Tous les exemples que nous trouvions manquaient d'explication ou bien se reposaient sur des paquets très anciens et non maintenus.

Nous avons toutefois tenté d'exposer un flux MPEG transcodé (à l'aide de FFMPEG, un utilitaire pour convertir des formats vidéo). Ceci fonctionnait très bien sur un PC fixe relativement puissant, mais l'ordinateur embarqué de Robotino (en plus de demander un FFMPEG obsolète) s'est très vite retrouvé surchargé. Nous avons donc cherché une meilleure implémentation, moins gourmande en ressources.

Finalement, cette itération nous a permis de "tâter le terrain" et de comprendre les bases du transcodage et du streaming vidéo.

Deuxième itération

Cette deuxième mouture utilisait une implémentation récupérée sur le net et s'appuyant sur OpenCV. Elle avait l'avantage d'être très simple mettre en place et à utiliser. L'idée est d'ouvrir une connexion et d'écrire dedans au fur et à mesure les images. C'est beaucoup plus léger que de transcoder et de faire des paquets d'images (tel que fait MPEG). L'inconvénient, c'est que cela fait beaucoup de trafic réseau (plus que MPEG) mais on constate moins de latence.

Là encore, le code fonctionnait très bien sur Windows. Cependant, OpenCV n'existe pas pour Ubuntu Jaunty (9.04). Or, ceci est la version du système d'exploitation qui tourne sur Robotino. Nous avons d'abord tenté de recompiler OpenCV pour Ubuntu 9.04 (une expérience riche d'enrichissements dans les archives de paquet et la gestion de dépendances obsolètes) mais nous ne sommes pas allés au bout car nous avons appris qu'il avait été choisi de mettre un Raspberry Pi, plus puissante et avec un OS à jour, sur Robotino. Toutefois, nos collègues rencontrant des difficultés à faire fonctionner OpenCV sur le Raspberry Pi, nous avons tenté une autre approche.

Cette itération a permis de raffiner la connaissance du streaming vidéo ainsi qu'en apprendre plus sur les protocoles de transfert de média. Elle nous a aussi permis de voir comment se compile un paquet sous Linux.

Troisième itération

La version actuelle du logiciel utilise les enseignements acquis sur le code de OpenCV, pour implémenter Webcam API de façon légère et non-dépendante de la plateforme. Le code est très similaire (presque le même en fait) mais ne dépend plus d'OpenCV pour fonctionner, en plus de

[1] https://github.com/sarxos/webcam-capture

permettre un contrôle plus fin des drivers utilisés. Ce code est aussi performant que la 2e itération et produit le même trafic réseau.

Un gros avantage de ce code sur la 2e itération est une reconnexion automatique et le déploiement possible en jar autonome.

Toutefois, ce client est incapable de gérer plusieurs connexions simultanées. Ceci est un choix de design car on souhaite ne pas surcharger le robot (ou le RPi) avec plusieurs flux. Nous préférons que le serveur redistribue le flux vidéo à plusieurs clients. Ceci est aussi une option de sécurité. Le robot n'a pas besoin d'être connecté à Internet puisque c'est le serveur qui se charge de toutes les connexions entrantes et sortantes.

Finalement, cette itération a permis de voir le déploiement en paquets indépendants (jar) et de voir le fonctionnement d'un keepalive dans un contexte de distribution de flux vidéo.

3.3.2. GESTION ET REDISTRIBUTION DES FLUX VIDEO

La gestion du flux par le serveur est nécessaire afin de renvoyer le flux de la webcam à plusieurs utilisateurs sans surcharger Robotino, et il sera nécessaire de la mettre en place pour le prochain semestre.

Cette gestion du flux pose certains problèmes par la nature des données que le serveur doit analyser.

3.4. ÉVOLUTIONS

3.4.1. CLIENT VIDEO

A l'avenir, il serait utile que le client vidéo puisse être automatisé par scripts, c'est-à-dire qu'il puisse fonctionner comme un programme indépendant (un jar) avec ses options de lancement, ses codes de sorties, etc. Ainsi, il sera aisé de le lancer avec une tâche planifiée, de façon automatique. Ce sera un des chantiers du prochain semestre.

3.4.2. REDISTRIBUTION DES FLUX VIDEO

Nous souhaiterions rendre fonctionnelle la redistribution du flux, qui nécessitera une analyse de ce que la webcam envoie afin de connaître les données à envoyer au client web.

3.4.3. AUTRES

Au prochain semestre, il va aussi être nécessaire de retravailler et d'uniformiser la JavaDoc, ainsi que la compléter de fichiers d'aides additionnelles et la compiler. Nous devrons aussi rendre certaines parties du code un peu plus claire et/ou optimum, si le temps le permet. Enfin, il faudra toujours penser

à incorporer les autres solutions déjà faites, particulièrement celles qui arrivent à maturation (traitement d'image, capteurs), ce qui va nous prendre aussi beaucoup de temps.

3.5. CONCLUSION

Ce semestre nous a permis d'affiner la solution que l'on propose et de rendre les outils que nous utilisons un peu plus performants. Ce fut aussi l'occasion d'essayer de développer un logiciel (le client Webcam) plus spécialisé et avec d'autres critères que le serveur.

4. PARTIE WEB

4.1. OBJECTIFS

Les objectifs de ce semestre pour la partie web étaient les suivants :

- Contrôler le robot depuis le site à l'aide d'un joystick virtuel
- Affichage de la caméra
- Affichage des données des capteurs
- Enregistrements des données des capteurs dans la base de données

4.2. TRAVAIL EFFECTUE

Tout d'abord, l'objectif le plus important de ce semestre était de pouvoir contrôler Robotino à l'aide du joystick virtuel. Etant donné que ce dernier a déjà été implémenté au semestre dernier, il a fallu simplement adapter les données qu'il renvoyait pour assurer le bon fonctionnement du robot. Pour rappel, on envoie simplement une position x, une position y, un angle et une force. Ces données sont transférées par WebSocket à un serveur, qui les renvoie à Robotino. Toutes les informations sont au format JSON.

Nous avons également permis l'affichage de la caméra sur le site. Le site web affiche maintenant les données des capteurs présents dans une salle. Ces données sont reçues et sont stockées dans une base de données à l'aide de requêtes AJAX.

Ci-dessous une capture d'écran du contrôle de Robotino depuis le site web.

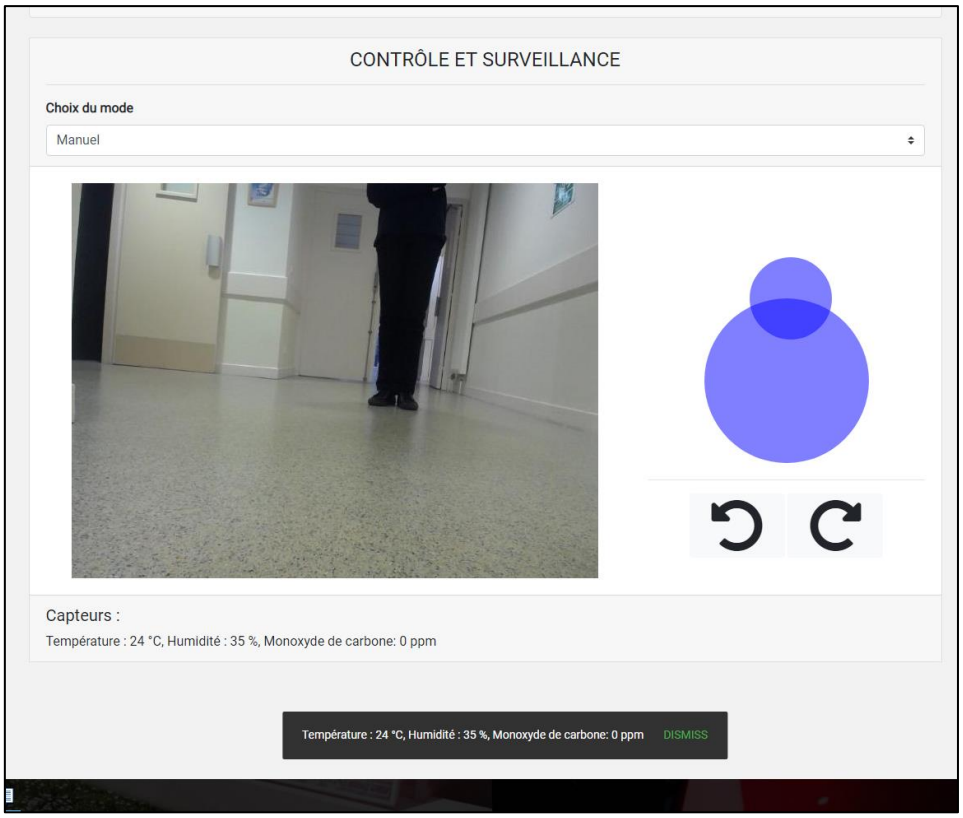


Figure 1 - Contrôle manuel de Robotino

Une refonte du design du site a également été effectuée. Le site n'a finalement pas subi beaucoup de modifications depuis le semestre dernier. Seules quelques améliorations ont été effectuées, dont l'ajout d'une page contenant toutes les données enregistrées par les capteurs.

4.3. DIFFICULTES RENCONTREES

L'affichage de la caméra a été le plus difficile à gérer. En effet, en tout premier, le serveur n'envoyait pas un flux vidéo à afficher mais une image. Nous devions ensuite gérer le rafraichissement de cette image en JS avec d'obtenir l'illusion d'une vidéo. Pour une vidéo fluide, il fallait rafraîchir au moins cette image 30 fois par seconde. Cette solution marchait, mais le navigateur était saturé au bout d'un moment à cause des requêtes multiples par seconde qu'il effectuait. Il a donc fallu gérer l'affichage de la caméra plus proprement à l'aide d'un flux vidéo développé par l'équipe serveur. Cette fois-ci, on affiche une vidéo directement, et par conséquence le navigateur est plus fluide car on ne n'effectue plus de rafraichissement.

4.4. TRAVAIL ENVISAGE

Il serait intéressant de permettre la gestion de plusieurs robots en même temps depuis le site. Pour le moment, il est uniquement possible de contrôler un seul robot. Pour gérer un second robot, il faut tout d'abord déconnecter le premier.

Il serait également intéressant d'effectuer un traitement sur les données des capteurs, comme par exemple l'affichage sous forme de graphique, effectuer des moyennes, etc.

4.5. CONCLUSION

Cette partie du projet n'a pas beaucoup avancé ce semestre car la plupart des implémentations nécessaires au contrôle de Robotino avait déjà été réalisées au semestre dernier. En résumé, il est maintenant possible de contrôler le robot depuis le site web à partir d'un joystick virtuel, de voir le retour de la caméra placée sur Robotino, et de lire les données renvoyées par nos capteurs. Le site web est maintenant opérationnel mais peut toutefois encore subir des améliorations au fil des semestres.

5. PARTIE CAPTEURS

5.1. OBJECTIFS

Nous décomposerons les objectifs du groupe capteurs en trois grands buts :

- D'une part la lecture des données dans l'environnement de Robotino, que ce soit la présence de gaz, le relevé de la température et l'humidité, ou d'autres types d'acquisition.
- Le montage de chaque capteur et de leurs procédés respectifs.
- La communication de nos cartes électroniques avec le serveur, et l'envoi de données vers le serveur et le site.

La première partie est donc orientée programmation Arduino avec le code dédié aux capteurs.

Pour la seconde partie, nous allons devoir préparer des montages pour chaque capteur en consultant leurs documentations pour ne pas les endommager et leurs fournir la tension nécessaire pour les alimenter.

Enfin, la dernière partie consiste à renvoyer les valeurs au serveur pour qu'elles soient stockées, traitées, et utilisées par Robotino.

5.2. COMMANDE DU MATERIEL

Tout d'abord, nous avons préparé et effectué notre première commande. Notre idée était de pouvoir faire l'acquisition de plusieurs grandeurs utiles pour notre projet et de l'envoyer à notre serveur. Nous avons donc commandé un ensemble de capteurs pour la température, l'humidité, et le gaz.

Ensuite, nous avons besoin de microcontrôleurs. Pour cela nous avons commandé des cartes ESP8266 accompagnées de leurs modules WIFI. Enfin, nous avons commandé des boitiers de piles pour pouvoir rendre le procédé portable avec une certaine autonomie.

5.2.1. LE MICROCONTROLEUR ESP8266

Pour la carte électronique, nous avons choisi l'ESP8266 NodeMCU fabriquée par le constructeur chinois Espressif. Cette carte est très pratique pour ce genre de projet : elle est compacte, possède un module Wifi intégré pour communiquer, et peut se programmer avec l'IDE Arduino. Il était nécessaire de télécharger la bibliothèque et l'environnement dédié à l'ESP8266 NodeMCU 0.9 sur l'IDE Arduino. Pour le reste tout fonctionne comme sur Arduino, en C++, et énormément de documentation est disponible sur le net.

Figure 2 - Carte ESP8266

5.2.2. CAPTEUR DE TEMPERATURE ET D'HUMIDITÉ

Pour ce faire nous avons choisi d'utiliser le capteur DHT11 : en effet ce capteur renvoie des valeurs numériques très facilement interprétables, et est fourni avec une librairie complète et facile d'utilisation. C'était un choix évident pour notre projet afin de récupérer les valeurs de températures et d'humidité d'un environnement. Toutefois, ce capteur ne peut pas être installé en extérieur car il ne fonctionne pas en dessous de 0°C. Etant dans un environnement clos à température ambiante, cela n'a pas été une contrainte à notre projet.

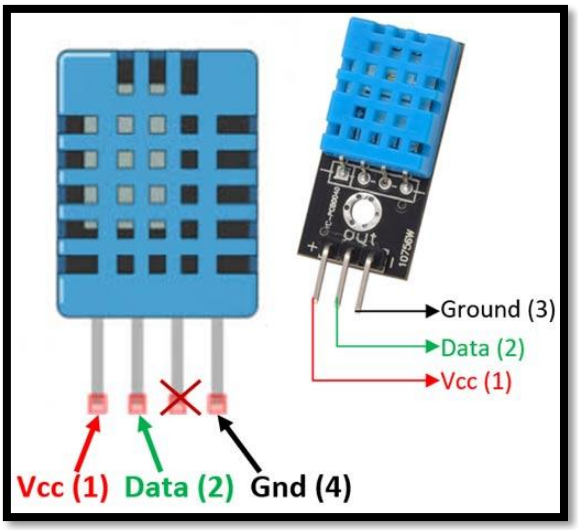


Figure 3 - Capteur DHT11

5.2.3. CAPTEUR DE GAZ

Concernant le capteur de gaz, nous avons choisi le capteur MQ2, un capteur très polyvalent mais complexe à utiliser. Il peut détecter des gaz issus de sept sources différentes, sous-réserve de calibration. De plus, le code constructeur permettant de calibrer le capteur s'est avéré difficile à maitriser et à comprendre. Ce capteur nous permet de détecter le monoxyde de carbone, issu d'une combustion incomplète et présent lors des incendies, ainsi que la fumée. Le résultat est affiché en ppm (partie par million).

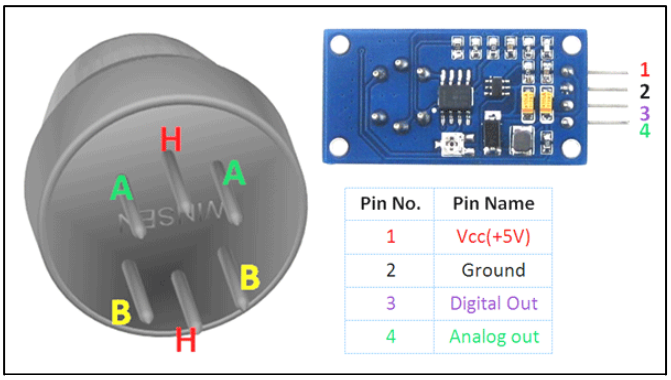
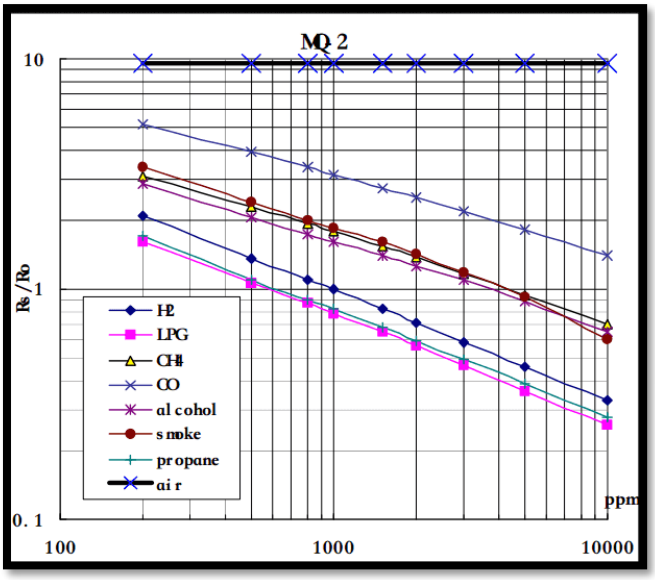


Figure 4 - Schéma du capteur MQ2

Figure 5 - Capteur MQ2

Les graphiques suivants sont fournis par le constructeur pour informer du fonctionnement et du calibrage du capteur. Nous avons donc seulement utilisé les courbes du CO (monoxyde de carbone) et du Gaz GPL (LPG en anglais).



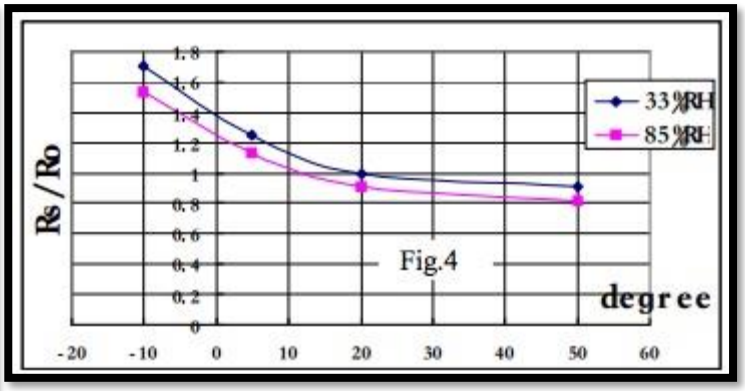


Figure 6 - Caractéristiques du capteur MQ2

5.3. PROTOTYPAGE

Une fois notre commande passée et reçue, nous avons pu commencer à prototyper notre projet. La seule difficulté fut la gestion des tensions : en effet, le MQ2 renvoie une tension entre 0V et 5V, alors que l'ESP8266 et son port analogique n'accepte qu'une tension entre 0V et 3.3V. Pour remédier à cela, nous avons utilisé un pont diviseur de tension. Cependant nous perdons en précision. C'est un souci moindre au sens où les valeurs seront soit très haute, soit très basse pour le gaz, permettant le déclenchement de l'alerte pour Robotino.

La partie acquisition de données a rencontré quelque difficulté au niveau du capteur de gaz : ce dernier fut difficile à calibrer. Le code et les instructions constructeurs n'étant pas très précises, le développement de cette partie fut rallongé.

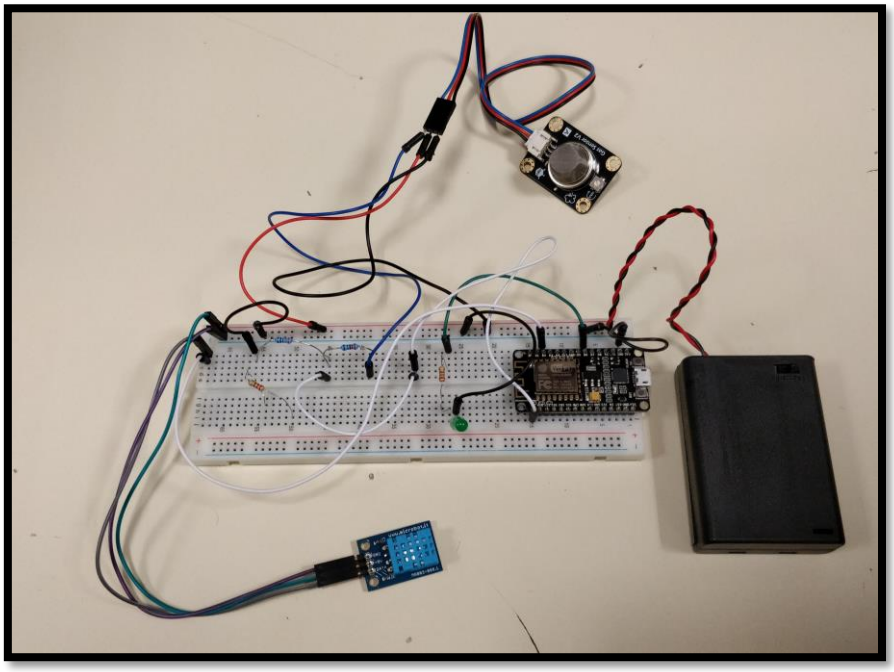


Figure 7 - Montage de test

Cependant, nous parvenons désormais à récupérer régulièrement nos données. Nous acquérons actuellement des valeurs toutes les 10 secondes. Il est possible que cet intervalle soit modifié plus tard.

De plus, nous avons réussi à lier nos cartes ESP8266 au serveur, ce qui nous permet d'afficher les données sur le site web. L'envoi se fait en JSON par sockets. Nous avons d'ailleurs rencontré quelques difficultés à connecter nos cartes au Wifi de la salle d'APP. En effet, la clé Wifi était mal gérée par nos cartes car cette dernière est trop longue pour fonctionner avec la bibliothèque utilisée. Il a donc fallu mettre en place une seconde borne Wifi avec une clé plus simple. Cette solution reste cependant temporaire jusqu'à que nous arrivions à résoudre le problème lié à la clé Wifi.

5.4. CONCLUSION

En résumé, nous avons eu un avancement très satisfaisant sur cette partie du projet. Toutefois, nous souhaiterions revoir notre calibrage sur les capteurs de gaz et intégrer la détection de présence. La suite de cette partie sera concentrée sur l'ajout d'autres types de capteurs et la fabrication d'un circuit plus propre : soudure sur platine de prototypage et si possible, créer des boitiers via SolidWorks, et ainsi les imprimer.

A terme, ces boitiers devront émettre vers le serveur pour pouvoir indiquer les sources d'anomalies et ainsi permettre au robot de se déplacer intelligemment. Ces boitiers ont pour vocation de devenir les yeux et les oreilles de l'environnement de Robotino, pour lui permettre une plus grande appréciation des évènements qui pourraient se produire autours de lui.

Nous avons eu des difficultés pour effectuer nos tests, notamment ceux du gaz, mais nous avons pu les faire chez nous, et non pas en présentielle. Il était assez facile d'avancer en dehors de la salle d'APP et les séances qui réunissaient le groupe permettaient de tester les cartes avec le WIFI dédié à l'APP. Ainsi, nous pouvions tester la connexion au serveur et l'envoi vers le site.

6. PARTIE ROBOTINO

6.1. OBJECTIFS

L'objectif de ce semestre était de tout d'abord de refactor le code, pour ensuite commencer à développer la partie automatique du déplacement du robot.

6.2. REFLEXIONS INITIALES

Tout d'abord, nous nous sommes intéressés aux capteurs que possédait le robot, ce qui nous aurait permis d'utiliser une ligne sur le sol qu'aurait suivi le robot pour se déplacer.

Par la suite, nous avons réfléchi à des alternatives possibles. Pour cela, nous avons trouvé la librairie OpenCV qui permet de faire de l'analyse d'image. Il fallait donc que la caméra sur le robot soit fonctionnelle.

6.3. TRAVAIL EFFECTUE

Pour concevoir cette partie du projet, nous avons utilisé un Raspberry Pi. Nous avons tout d'abord commencé par copier une image de Debian sur une carte micro SD. Ensuite, nous avons préparé l'environnement pour que nous puissions développer sous Java. Nous avons vérifié que Java était bien installé et ensuite nous avons procédé à l'installation d'Eclipse.

Pour OpenCV, il a fallu installer tout l'environnement nécessaire, ce qui a fallu environ 7 heures de téléchargement. Une fois tout cela accomplie, nous pouvions commencer le développement.

Nous allons utiliser OpenCV pour faire déplacer le robot. Puisque cette bibliothèque est nouvelle pour nous, nous avons commencé par nous former sur son fonctionnement. Malheureusement, il y a très peu de ressources sur OpenCV en Java en dehors de la documentation standard. La plupart des applications rencontrées se font en Python. Une fois que nous nous sentions assez confiant, nous avons créé une application de détection du bas du corps. Nous avons choisi la détection du bas du corps car le robot est très proche du sol et arrive seulement à distinguer les jambes de personne, à part si celui-ci se situe très loin du robot. Puisque notre robot fait de la télésurveillance, nous trouvons impératif que le robot puisse détecter des potentiels intrus pendant ses déplacements et par enchaînement informer le client d'une intrusion potentielle.

Voici, ci-dessous, un exemple d'exécution de notre programme. Pour que l'on puisse attester de son bon fonctionnement, nous avons afficher des rectangles verts autour de ce que le programme arrive à détecter.

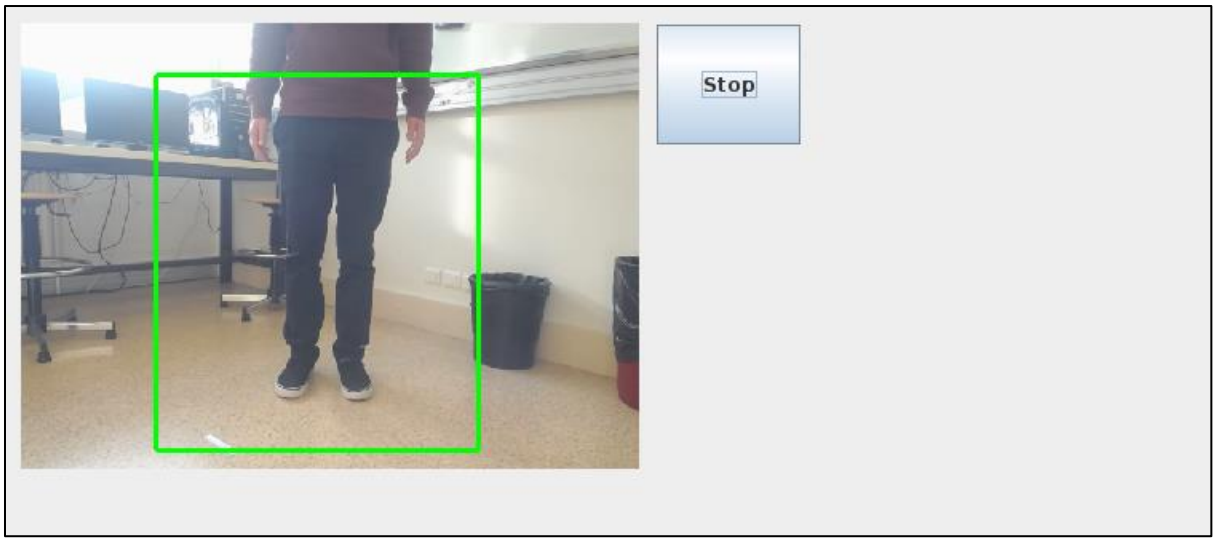


Figure 8 - Détection du bas du corps

6.4. DIFFICULTES RENCONTREES

Nous avons eu plusieurs problèmes durant ce semestre.

Tout d'abord, nous avons rencontré quelques difficultés avec les capteurs intégrés au robot. Nous n'avons pas réussi à récupérer des valeurs différentes de 0 pour les mesures, malgré le suivi de la documentation de Robotino.

Par la suite avec l'utilisation d'OpenCV, nous avons eu quelques problèmes avec l'image du Raspberry. On souhaitait avoir 2 sauvegardes de nos avancées. Mais, nous avons 2 cartes SD de taille différente. Nous avons dû recréer une nouvelle image et récupérer toutes les données de la 1re image.

6.5. TRAVAIL ENVISAGE

Dans le futur de l'application, nous pensions faire la localisation du robot avec OpenCV. A partir de logos que l'on collerait au mur, le robot pourrait estimer sa distance au logo et ainsi estimer sa position dans un espace. Il pourrait ainsi se déplacer de façon autonome.

6.6. CONCLUSION

Actuellement, le robot dispose de toutes les fonctionnalités de base nécessaire pour la suite. Le semestre prochain sera consacré au développement de la partie automatique, qui sera très longue à effectuer. Ces premières réflexions concernant la partie automatique nous ont appris à découvrir la librairie OpenCV, qui est très utilisée dans le milieu de la robotique.

7. CONCLUSION

Durant ce semestre, nous avons pu mettre en commun tout ce que nous avions développé le semestre précédent et nous en servir pour créer le mode manuel de Robotino. Nous sommes maintenant capables de déplacer le robot et visualiser la vidéo provenant de la caméra placée sur Robotino à partir du site web. Ce semestre, nous avons également mis en place les capteurs nécessaires à notre projet. Pour le moment, nous nous sommes limités à un capteur de température et d'humidité, et un capteur de gaz. Ces capteurs peuvent communiquer avec notre serveur via une carte ESP8266 et les données qu'ils renvoient sont affichées sur le site web et enregistrées dans une base de données.

Nous avons rencontré quelques difficultés pendant cette période : gestion de la caméra, problèmes de connexion des cartes au Wifi, configuration des capteurs, etc. Niveau organisation, nous avons beaucoup évolué par rapport au semestre dernier. Les réunions étaient plus régulières et plus efficaces. Globalement, nous sommes satisfaits de notre avancé ce semestre. Nous envisageons pour le semestre prochain de mettre en place le mode automatique du robot, qui est au cœur de notre projet.