

Eva RYCKBOSCH

Lionel SARDO VISCUGLIA

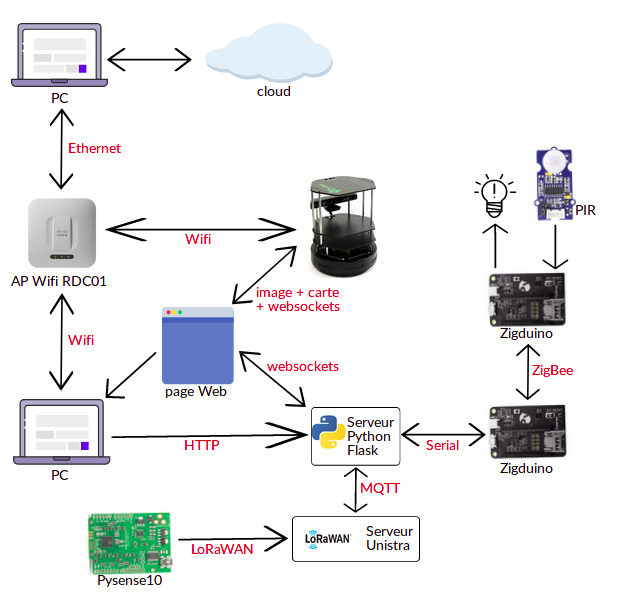
Jérémy TELLIEZ

Marie-Lou BRISMONTIER-T.

Rapport de Projet

Robotique & Domotique Communicant

# Introduction



Document 1: Schéma global

L’objectif de ce projet était d’implémenter une routine de déplacement pour le robot qui puisse être interrompu par des évènements extérieurs provenant de capteurs. Le turtlebot 2 doit se déplacer entre trois points définis dans la salle. Mais lorsque des informations des objets communicants, un Zigduino et une Lopy, lui sont transmisent, le robot doit interrompre sa routine pour se déplacer jusqu’à eux. Les protocoles de communication utilisées sont ZigBee, LoRaWan, WiFi et MQTT. Les informations transitent par un serveur Python et une page Web. Le point d’accès WiFi fait la liaison entre notre sous-réseau local et un ordinateur qui possède une connection internet et un serveur DHCP pour ce sous-réseau.

# Sommaire

[**Introduction**](#_b0jyrh2p26ws) **2**

[**Sommaire**](#_lva44uvmjtq) **3**

[**Zigduino**](#_v9n81nyywjxi) **4**

[**LoPy**](#_phyuo9z0kznl) **6**

[**Robot**](#_uy0xtvqez8iz) **7**

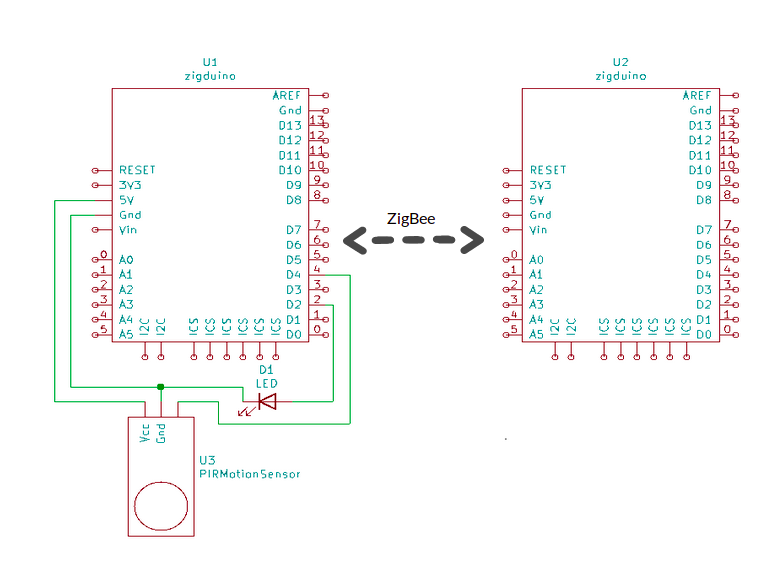
[Configuration du robot](#_daouv17z0k8f)

[Explication du script cir.py](#_ug6oklwmnssa)

[**Gateway**](#_bowh7nlb06z4) **9**

[**Conclusion**](#_bkg57iabav91) **11**

# Zigduino



Document 2: Schéma de montage des Zigduino

La communication entre les deux Zigduino se fait par le protocole ZigBee. La bibliothèque utilisée pour ZigBee est celle fournie dans le tp4.

Nous avons utilisé trois messages différents pour communiquer. La Zigduino qui sert de gateway peut envoyer à l’autre :

**‘0’** pour éteindre la led

**‘1’** pour l’allumer

L’autre carte peut envoyer :

**‘2**’ pour signaler une détection de mouvement

Elle ignore tous les autres messages en les considérant comme du bruit.

La carte reliée à la led et au détecteur de mouvement fait une acquisition de la valeur renvoyée par ce capteur à chaque boucle. Pour ne pas envoyer un message à chaque fois que le capteur envoi une valeur positive, nous avons introduit une variable globale *mem* qui stocke la dernière valeur lue. Cela permet de n’envoyer qu’un seul message à la première détection d’un mouvement sans bloquer l’écoute pour autant.

La deuxième carte communique directement avec la gateway via une connection série, agissant donc comme proxy qui transfère les messages. Dans premier temps, elle écoute sur le port série, et transmet d’éventuels messages provenant de la gateway à l’autre Zigduino. Elle écoute ensuite les messages Zigbee, et les transmet à la gateway en les recopiant sur le port série.

# LoPy



Document 3: Schéma communication LoPy-Gateway

Notre groupe a utilisé la carte Pysense-10.

Nous nous sommes basés sur l’exemple pysense\_lorawan\_cayenne donné. Dans le fichier de boot nous avions uniquement remplacé la clé OTAA. Quant au fichier main, nous l’avons modifié pour qu’il fonctionne de la même façon que le code pytrack\_cayenne\_eventdriven, c’est à dire détecter les mouvements par interruption de l’accéléromètre.

Les interruptions étaient trop nombreuses car la PySense est très sensible, nous avons donc en plus ajouté un test par rapport à une valeur de l’accéléromètre mesurée pendant le setup. Il est donc important que la PySense ne bouge pas pendant le setup. Les messages sont envoyés au serveur de l’unistra, et contiennent uniquement les valeurs de l’accéléromètre.

Ces messages sont récupérés par requête MQTT auprès du serveur de l’unistra. Nous utilisons la librairie Python Paho pour se connecter au serveur LoRa en tant que client depuis la gateway. On souscrit au topic application/pysense/device/pysense10# avec le nom d’utilisateur et le mot de passe (userdemo1 & icube01).

# Robot

## Configuration du robot

On connecte le robot au sous-réseau WiFi local afin de pouvoir se connecter en ssh dessus. A partir de ce moment, nous n’avions plus à intervenir physiquement sur le pc du robot. Nous avons utilisé **byobu** pour gérer plusieurs terminaux et partager certaines variables d’environnement.

Il faut configurer sur les terminaux du robot la variable d’environnement **ROS\_HOSTNAME** avec **export ROS\_HOSTNAME=ip\_robot**.

Sur le pc distant qui lancera rviz, il faut configurer les variables avec :

**export ROS\_HOSTNAME=ip\_pc**

**export ROS\_MASTER\_URI="http://ip\_robot:11311"**

Il faut ensuite installer le paquet ros **circuit** et penser à exporter le projet catkin dans lequel le paquet sera installé, dans notre cas il s’agissait de **catkin\_project** avec :

**source /home/turtlebot/catkin\_project/devel/setup.bash**

**source /home/turtlebot/catkin\_project/install/setup.bash**

Il faut lancer sur le robot les scripts suivants :

1. **roslaunch turtlebot\_bringup minimal.launch** : démarre le robot et permet aux autres scripts de fonctionner
2. **roslaunch turtlebot\_navigation amcl\_demo.launch map\_file:=/home/turtlebot/map\_clean.yaml** : démarre la navigation sur le robot et précise quelle carte il doit chargé
3. **roslaunch rosbridge\_server rosbridge\_websocket.launch** : permet au robot de communiquer ses topics avec le serveur web sur l’adresse : ip\_robot:9090
4. **rosrun web\_video\_server web\_video\_serve**r : commence la diffusion des topics de type image du robot (dont la caméra) sur l’adresse : ip\_robot:8080
5. **rosrun circuit cir.py** : lance le script qui fait le circuit, son fonctionnement sera détaillé par la suite

Et sur le pc distant qui fait office de poste de contrôle :

**rosrun rviz rviz**, il faut également penser à y charger le bon fichier de configuration : **tutlebot.rviz**

Au besoin, pour des raisons de debug, on peut utiliser :

**roslaunch kobuki\_keyop keyop.launch** : pour le piloter manuellement

**rosrun circuit subtest.py** : permet de s’abonner aux 3 topics que le script cir.py utilise pour avoir un historique des messages transmis sur ces topics

## Explication du script cir.py

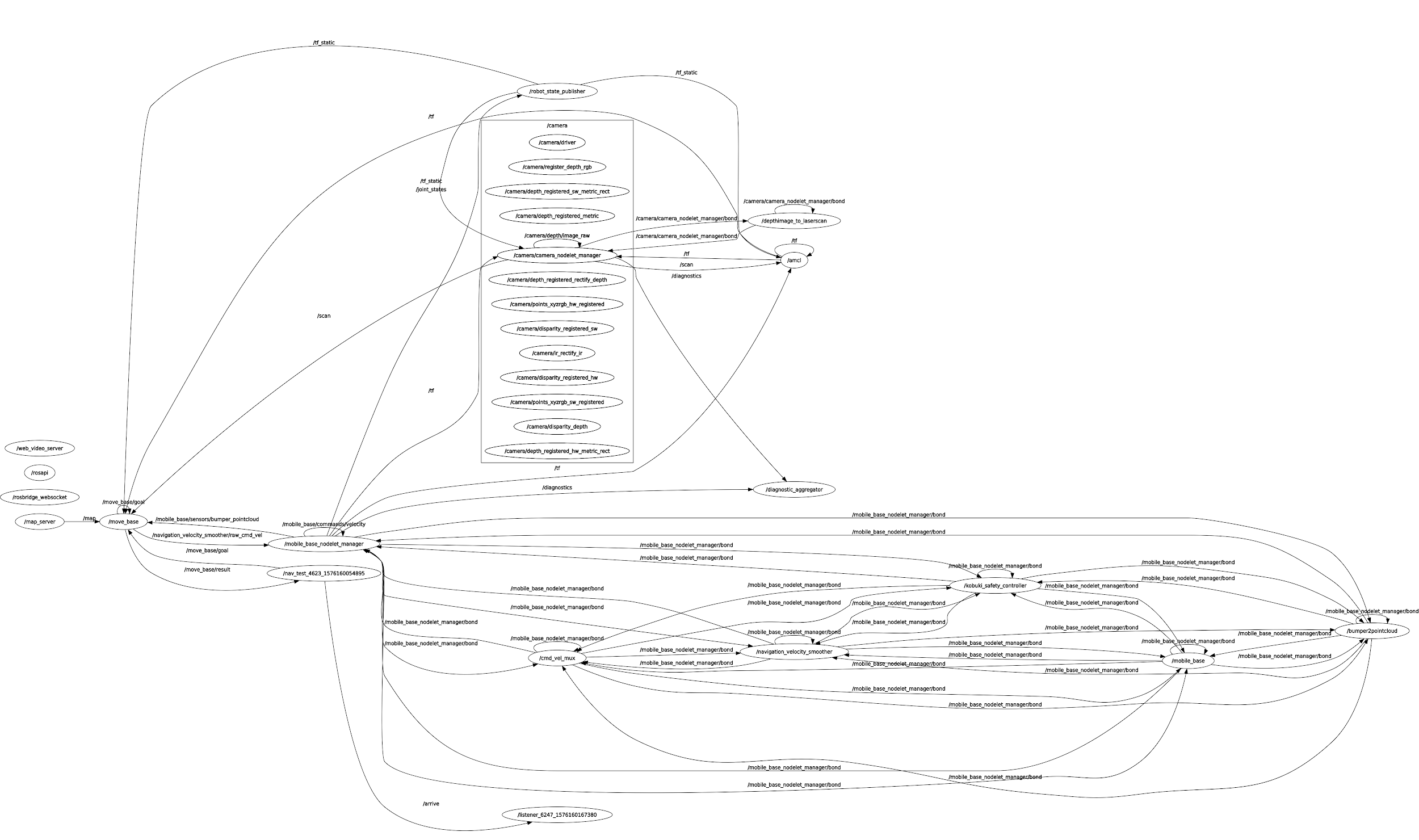
Le robot commence par attendre que l’opérateur lui indique sa position sur rviz.

Durant son fonctionnement de base, le robot va aux positions indiquées dans la liste *locations*. Il enchaîne les positions dans l’ordre et recommence lorsqu’il atteint la dernière position de la liste. Il est possible d’y ajouter autant de position que souhaité.

Le script s’abonne aux topics *zigduino* et *pysense*. Il devient un publisher sur le topic *arrive*.

Lorsqu’un message est reçu sur le topic zigduino, celui-ci est affiché et le robot annule son objectif actuel pour en redéfinir un autre : celui lié aux coordonnées de la Zigduino. Le fonctionnement est similaire pour la PySense.

Lorsque le robot atteint son objectif, quel qu’il soit, il l’indique au serveur web en publiant sur le topic *arrive*.

Document 4: Graphe Rqt-graph

# Gateway

La Gateway est composée de deux parties, le serveur Flask écrit en python, et la page HTML (avec son CSS et ses scripts Javascript).

Le serveur est composé de trois threads :

* Le thread principal gère le serveur avec les différentes routes possibles. Ainsi que les sockets entrants et sortants.
* Un thread lisant en boucle sur la liaison série et prévenant un websocket lié à la page web si elle lit un 2 correspondant à une alerte du capteur de mouvement.
* Un thread se connectant en MQTT au serveur LoRa de l’unistra et envoyant via websocket une alerte à la page web en cas de détection de mouvement par la Pysense.

Les threads sont nécessaires, car les deux fonctions d’écoute sont bloquantes ou tournent en boucle, et il est nécessaire de pouvoir servir des pages web pendant ce temps-là.

On lance le serveur via la ligne de commande, il faut fournir le port sur lequel est connecté la Zigduino ainsi que l’adresse IP du robot qui est donnée au moteur de template Jinja (fourni avec Flask) pour compléter la page Web.

La page web est le panneau de contrôle du système, elle accueille différentes fonctionnalités.

Tout d’abord on affiche le statut de la LED, ainsi que de la raspberry (si elle bouge ou non).

On peut allumer ou éteindre la lumière via des boutons ainsi que donner l’ordre au robot d’aller vers la Pysense ou la Zigduino à la main.

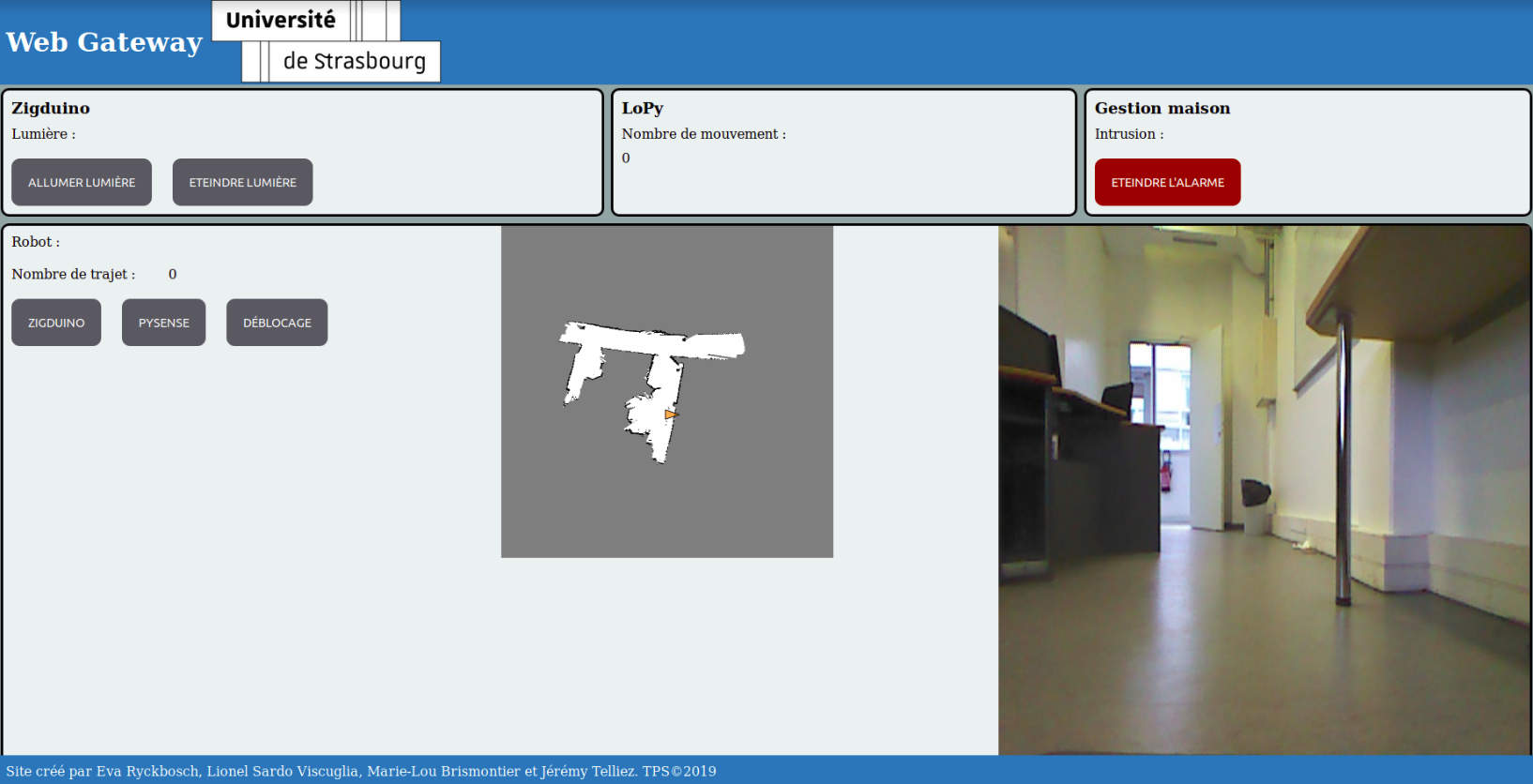
On affiche le stream vidéo de la Kinect du Robot ainsi que la carte interne et la position sur cette carte du robot. Enfin, on affiche le nombre de trajet complété du robot.

Trois fichiers Javascript pilotent la page web.

Le fichier “main.js” contient le code attachant les “handlers” de click des boutons, ainsi que la réception des messages venant du serveur via les websockets.

Le fichier “ros.js” contient tout le code gérant les interactions avec le robot qui utilise aussi les websocket mais facilite leur utilisation a travers les librairies javascript de RobotWebtool.

Le fichier “map.js” gère uniquement l’affichage et le positionnement sur la carte interne du robot.



Document 5: Page Web

Messages échangés :

**Robot - Page web :**

Les messages reçus par la page web le sont via le socket lié à ROS, les actions sont effectuées à la réception des messages, leur contenu n’est pas inspecté.

L’envoi de message au robot se fait via un message de type string, et contient un message permettant de savoir s'il vient du bouton ou du capteur, mais est uniquement utilisé à des fins de d’affichage, la logique étant gérée par les topics dans lesquels ces messages sont postés.

La pose est récupérée sur le topic “/odom” qui nous renvoie un objet Odometry dans lequel on peut récupérer la position.

**Serveur - Page web :**

Même fonctionnement que précédemment où on utilise uniquement la présence d’un message dans le socket pour déclencher une action, ce qui est pratique et rapide a implémenter.

# Conclusion

Bien que nous ayons rempli les objectifs du tp, nous aurions pu augmenter la fiabilité d’une part des informations remontées par les différents capteurs, d’autre part de certaines communications entre le robot et la gateway, telle que le flux vidéo qui était transmis non compressé ou encore la fréquence de rafraîchissement de la carte.

Il aurait également été intéressant de faire tourner rosbridge sur le pc accueillant le serveur web, le netbook étant instable à forte charge. La communication entre Zigduino aurait pu bénéficier d’un système d'acquittement et surtout d’un filtrage de bruit de meilleur qualité.

Ce projet a été très formateur, d’une part en nous faisant interconnecter toutes les briques que nous avions pu voir en TP mais aussi en nous montrant qu’il est très important de tester nos projets en situation réelle car malgré le fait que les composants fonctionne parfaitement séparément, leur assemblage va introduire des potentiels bugs, indétectables sans tester et pouvant mettre en péril l’intégrité du projet tout entier.