Robotique domotique : TP5

Groupe 4 :
HENNACH Loubna
ELAZZAM Abdellah
SCHEER Elisa

Avril 2018

Présentation du projet

Le but du projet de RDC est de rendre le robot communicant avec l'environnement dans lequel il évolue en se basant sur trois technologies de communication :

- IEEE 802.11 (Wi-Fi)
- IEEE 802.15.4 (ZigBee)
- LoRaWAN

Le scénario à implémenter est que le robot, après une phase d'apprentissage de son environnement, devra évoluer dans son habitat soit en mode programmé, soit en mode événementiel.

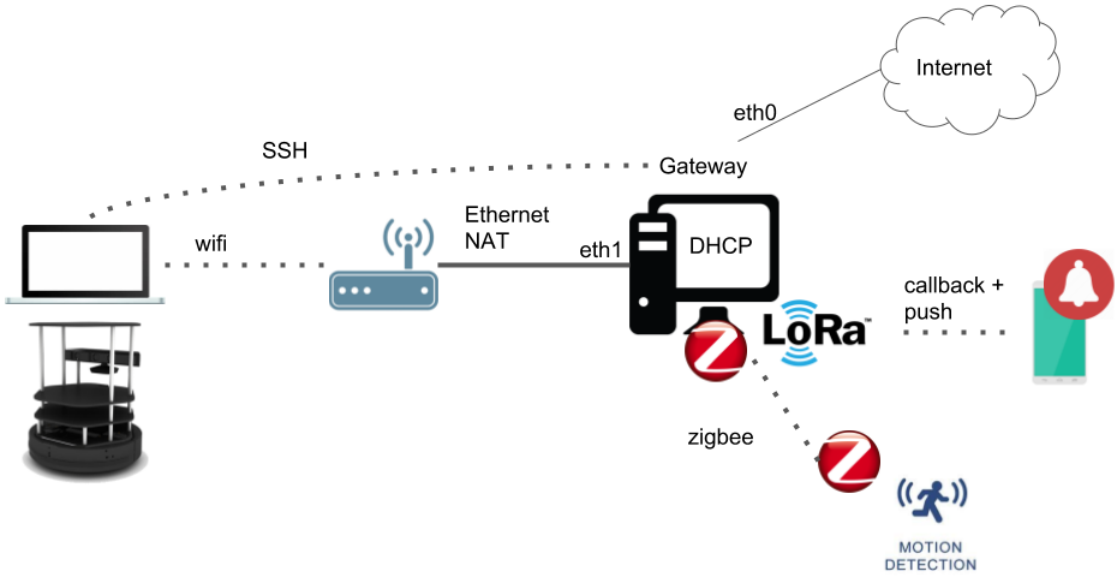
Pour le mode programmé, il s'agit du mode par défaut. On définit 3 goals qui correspondent au différentes étapes du circuit du robot. Le robot tourne en boucle sur ces 3 étapes jusqu'à l'ordre d'arrêter ou si un événement survient.

Pour le mode événementiel, un événement capté par le détecteur de mouvement d'un objet communicant Zigduino est relayé jusqu'à la box et déclenche un ordre pour le robot qui se déplace à l'endroit correspondant à l'alerte.

Une fois sur place, le robot utilise la webcam de la Kinect pour prendre visualiser l'origine du mouvement (prise de vue à 360 en mode panorama) et la renvoi à la box pour la visualiser dans l'interface web.

Et en cas de perte de la connexion à Internet, une connexion de secours LoRaWAN est sera mise en place, la box doit pouvoir relayer les alertes du détecteur de mouvement via l'interface LoRaWAN. Pour simplifier le scénario, on considère que chaque alerte est envoyée via l'interface LoRaWAN et l'utilisateur de réseau de domotique recevra une notification sur son smartphone.

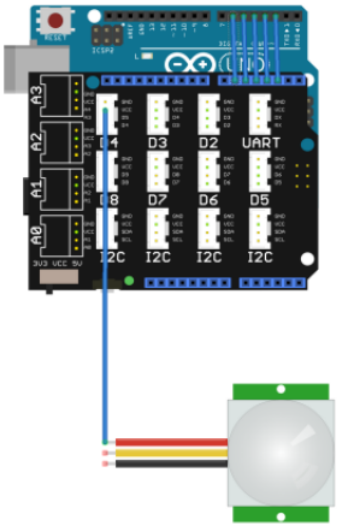
Architecture globale



Architecture matérielle

- 2 objets communicants type Zigduino (Zigduino r2)ayant pour rôle :
  - Détecteur de mouvement à l'aide du capteur de présence (branché sur D4)
  - Passerelle entre le réseau Zigbee, LoRaWAN et internet
- Un module LoRa, pycom pour pouvoir envoyer des notifications sur smartphone
- Un turtlebot + netbook
- Une borne WiFi Cisco pour se connecter au réseau avec le netbook du robot

Schéma de montage électronique Zigduino/capteur



Architecture réseau

Pour que le robot ait un accès à internet, nous avons utilisé une borne WiFi comme point d'accès qui est relié en ethernet à la box (un PC de la I005) sur laquelle nous avons fait un NAT et mis en place un serveur DHCP pour que le netbook du robot puisse obtenir une IP et avoir accès au réseau d'osiris.

Architecture logicielle

Pour notre projet nous avons utilisé ROS (Robot Operating System) qui est une plateforme de développement logicielle pour robot. ROS nous permet d'avoir une abstraction du matériel, un contrôle des périphériques de bas niveau, une mise en œuvre de fonctionnalités couramment utilisées, et une transmission de messages entre les processus et gestions des packages installés.
Le netbook connecté au turtlebot est le ROS master, un package y est crée contenant le code python (navigmotion.py) qui permet au turtlebot de se déplacer selon l'itinéraire du couloir, il permet également au turtlebot d'aller à l'endroit où un mouvement est survenu et de prendre une photo.
Sur l'ordinateur nous avons crée deux packages :

- `gate` qui contient le code `talker.py` qui publie sur le topic `chatter`

```
root@i005pc06:~# rosrun gate talker.py
1
[INFO] [WallTime: 1523779236.011877] 1
1
[INFO] [WallTime: 1523779237.012419] 1
1
[INFO] [WallTime: 1523779254.118428] 1
1
```

– `listen` qui contient le code `listener.py` qui souscrit au topic `chatter`

```
root@i005pc06:~# rosrun listen listener.py
[INFO] [WallTime: 1523779340.149046] /listener_5223_1523779337990recu 1
[INFO] [WallTime: 1523779341.149434] /listener_5223_1523779337990recu 1
[INFO] [WallTime: 1523779344.150305] /listener_5223_1523779337990recu 1
[INFO] [WallTime: 1523779345.150891] /listener_5223_1523779337990recu 1
```

Les commandes pour procéder au mouvement sont les suivantes :
Sur le Turtlebot :

– `roscore`
– `roslaunch turtlebot_bringup minimal.launch`
– `roslaunch turtlebot_navigation amcl_demo.launch map_file:=/home/rdc04/test2_slam.yaml`
– `rosrun web_video_server web_video_server`
– `roslaunch freenect_launch freenect.launch`
– `rosrun robot_pose_publisher robot_pose_publisher`
– `rosrun map_server map_server /home/rdc04/test2_slam.pgm 0.05`
– `roslaunch rosbridge_server rosbridge_websocket.launch`

Sur L'ordinateur :

– `roslaunch turtlebot_rviz_launchers view_navigation.launch`
– `rosrun gate talker.py`
– `rosrun listen listener.py`

Interface web

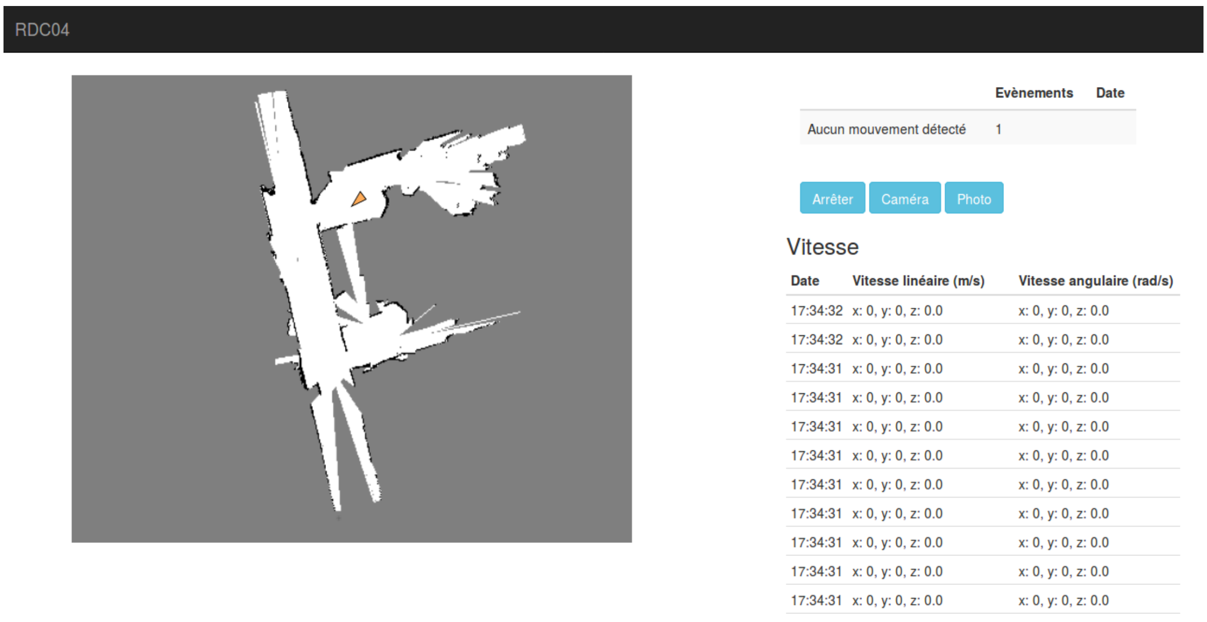
Pour l'interface web, nous avons décidé d'héberger le serveur web sur le robot.
Ce que nous avons installé sur le robot :

– web-video-server pour avoir accès au flux vidéo de la Kinect
– rosbridge-server pour pouvoir se connecter à ROS sur le port 9090
– apache2 pour le serveur web

– frennect-launch pour pouvoir utiliser web-video-server

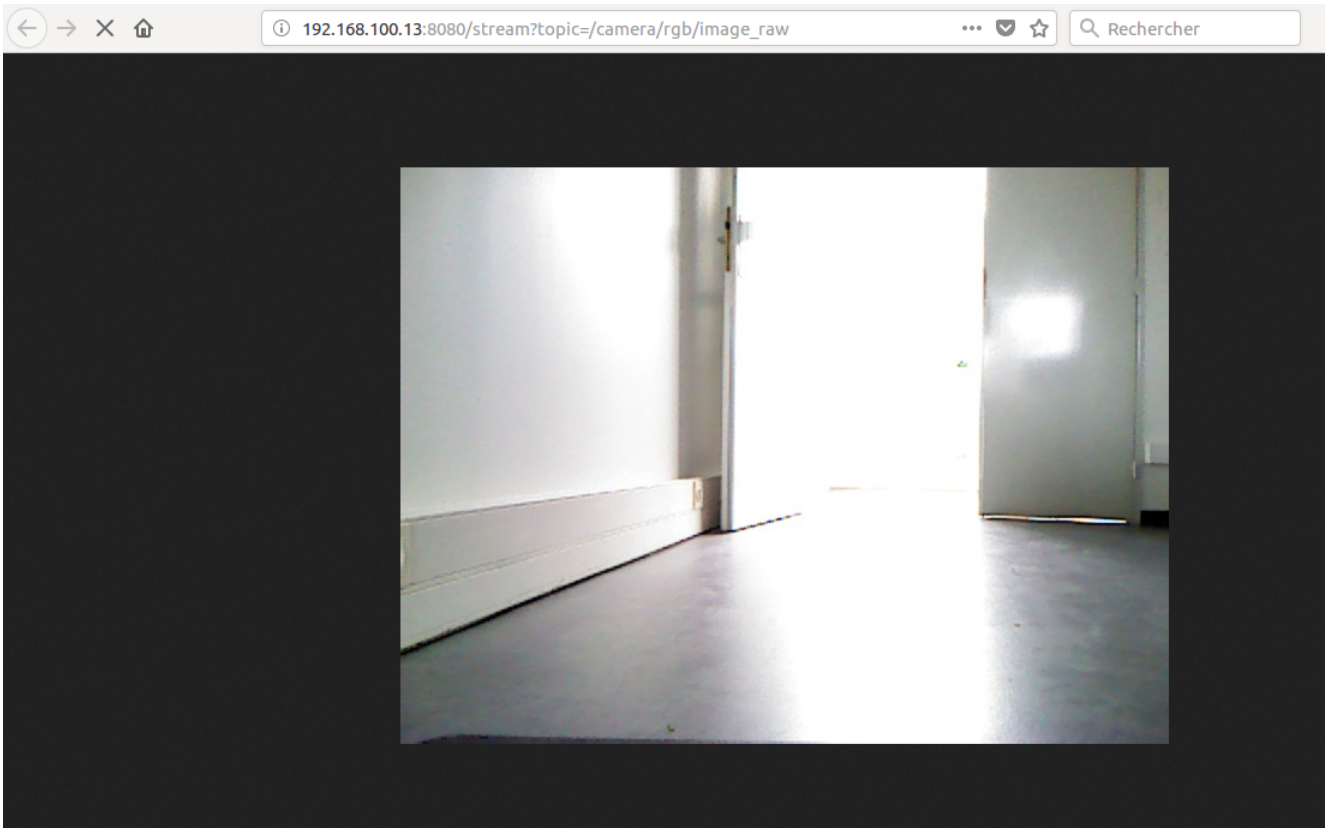
Et nous avons également utilisé les librairies de RobotWebTools suivantes :

– roslibjs pour utiliser les fonctionnalités de ROS
– ros2djs pour visualiser la carte



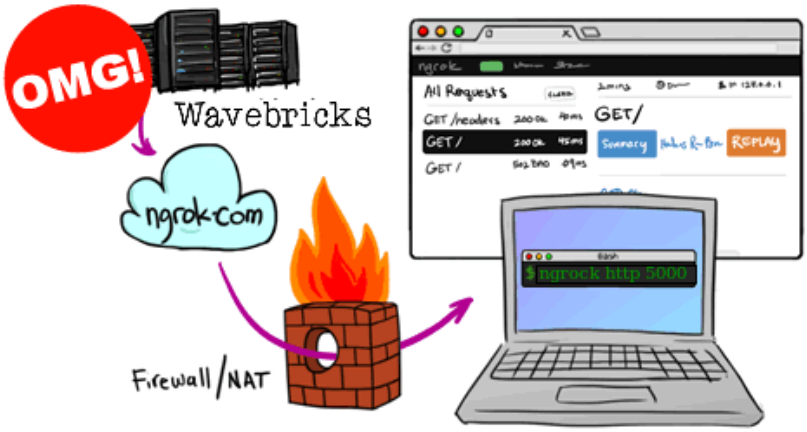
Sur l'interface web, on peut observer la position du robot sur la carte (marqueur jaune) et sur le côté, on peut accéder au flux vidéo du robot pour observer son environnement puis également accéder à la photo prise s'il y a eu mouvement. On peut également stopper le robot depuis cette interface : on publie sur un topic `arret` '1' et le robot, qui souscrit au topic s'arrête lorsqu'il reçoit '1'.

Capture du flux vidéo (bouton 'Caméra') :



Callback LoRaWAN

Sur l'interface LoRaWAN, nous avons configuré un callback qui redirige les trames LoRa vers un serveur via une requête HTTP POST,on utilise Ngrok pour catcher les callbacks sur wavebricks . C'est un outil qui permet de créer un tunnel d'un URL publique vers notre application local.

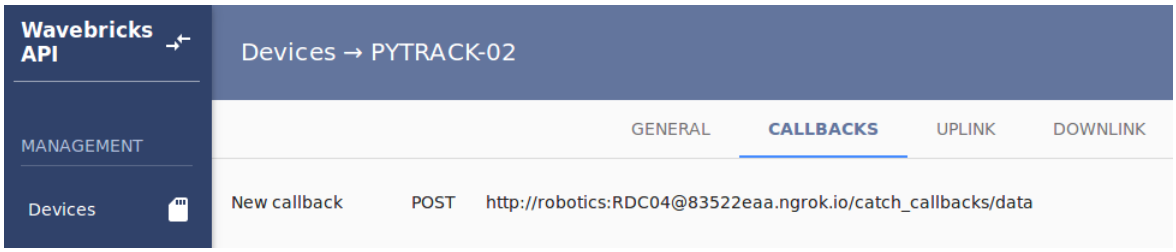


Text

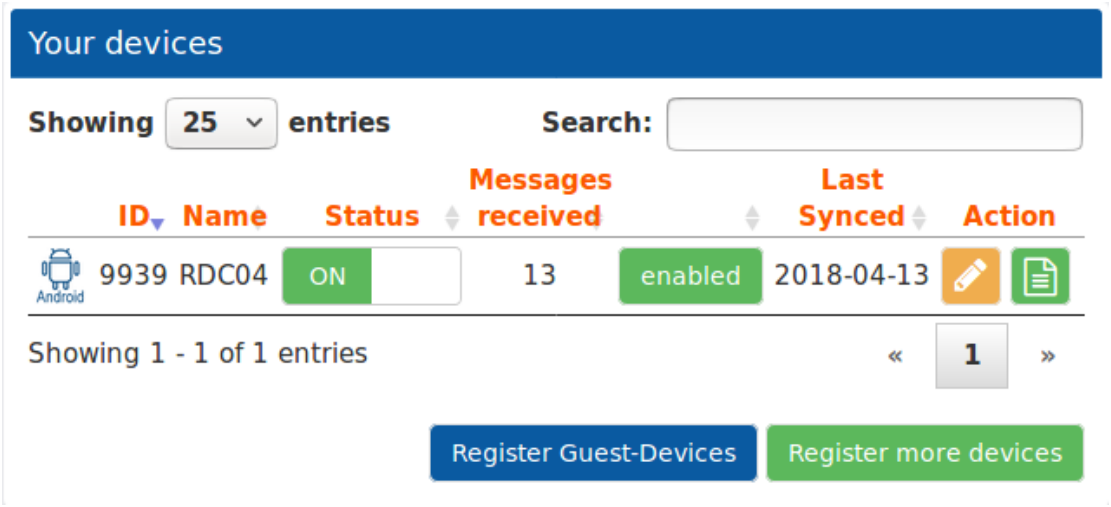
On démarre notre tunnel avec la commande (On utilise un mécanisme d'authentification pour sécuriser notre tunnel contenant l'URL publique.)
`./ngrok -auth user:password http 5000`
Après démarrage, ca s'affiche comme suivant:

```
ngrok by @inconshreveable                                       (Ctrl+C to quit)

Session Status                online
Account                       abdellah.elazzam@gmail.com (Plan: Free)
Version                       2.2.8
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://83522eaa.ngrok.io -> localhost:5000
Forwarding                    https://83522eaa.ngrok.io -> localhost:5000

Connections                   ttl     opn     rt1     rt5     p50     p90
                              9       0       0.06    0.02    0.28    0.58

HTTP Requests
-------------

POST /catch_callbacks/data     200 OK
POST /catch_callbacks/data     200 OK
POST /catch_callbacks/data     200 OK
POST /catch_callbacks/data     200 OK
POST /catch_callbacks/data     200 OK
POST /catch_callbacks/data     200 OK
```

Après, on met le lien publique (http://user:password@subdomain/catch_callbacks/data) fournie par Ngrock dans le callback de l'API wavebricks, de tel manière, on récupère les paquets authentifiés LORAWAN localement

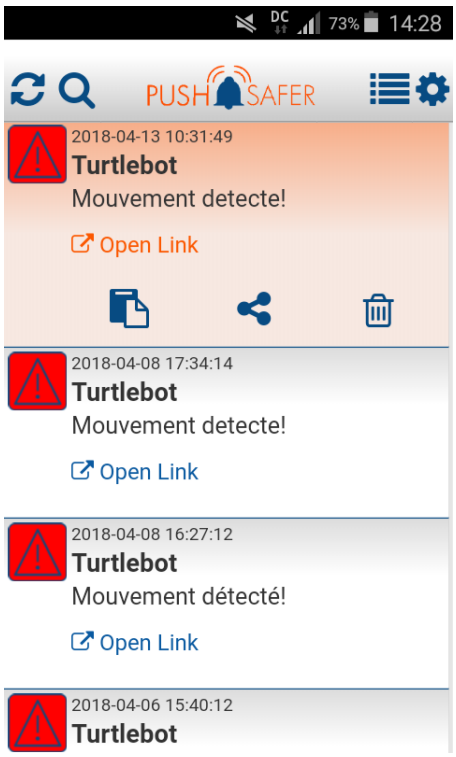


Pour pouvoir recevoir des notifications sur nos smartphones, nous avons utilisé l'application PushSafer que l'on a configuré sur le PC (en se créant un compte pour obtenir une clé privée pour rediriger les notifications). Puis nous avons installé l'application PushSafer sur un de nos smartphones pour le déclarer comme device dans l'API :



Puis enfin, lorsqu'un mouvement est détecté, le module LoRa émet une trame captée localement par ngrok et on peut recevoir une notification sur le smartphone avec PushSafer en utilisant le script `run.py` :

```
elazzam@elazzam-SATELLITE-C55-A-1QG:~/Desktop/Project/ngrock/flask_webserver$ python3 run.py
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 314-952-811
{"status":1,"success":"message transmitted","available":16}
127.0.0.1 - - [15/Apr/2018 17:46:27] "POST /catch_callbacks/data HTTP/1.1" 200 -
{"status":1,"success":"message transmitted","available":15}
127.0.0.1 - - [15/Apr/2018 17:46:37] "POST /catch_callbacks/data HTTP/1.1" 200 -
```



Type de messages échangés

Pour la communication entre les deux zigduinos :

Lorsque le capteur branché sur le premier zigduino détecte un mouvement, il transmet le caractère '1', sinon il transmet '0'. Puis quand le second reçoit '1', il l'affiche dans le moniteur série.

Pour la communication entre le ziguidno-gateway et le PC :

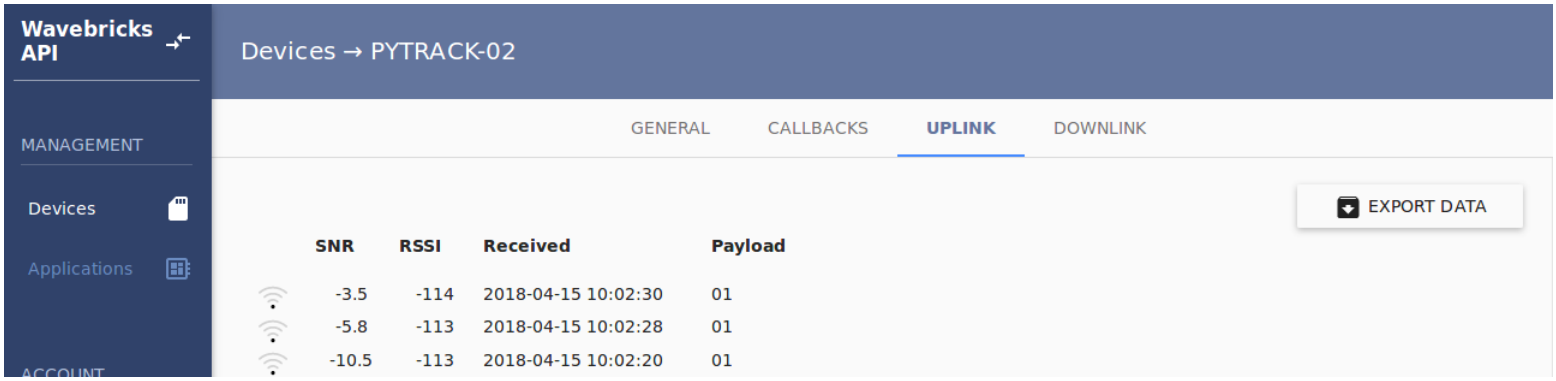
Lorsque le zigduino-gateway affiche le '1' reçu par l'autre module zigduino, le PC lit sur le port série et publie sur un topic `chatter`.

Pour la communication entre le turtlebot+netbook et le PC :

Après avoir déployé l'architecture réseau, on peut se connecter en SSH sur le netbook du robot pour lancer le package `demo_nav` pour qu'il commence sa patrouille, et lorsque le PC publie sur le topic `chatter` le mouvement, le robot ayant souscrit à ce topic se dirige vers l'endroit défini du capteur. Puis si on clique sur le bouton 'Arrêt' dans l'interface web, le robot reçoit un '1' sur le topic `arret` puis s'arrête.

Pour la communication LoRaWAN et le PC :

Package `listen` sur le PC qui souscrit au topic `chatter` puis lorsqu'un message est publié, on écrit sur le port série `ttyACM0` le caractère '1' (s'il y a eu mouvement, sinon on écrit rien) puis quand le module LoRa reçoit '1', il envoi une trame 01. Puis enfin sur l'API wavebricks (en ayant bien configuré notre module avec les bonnes clés et identifiant) on reçoit les trames UPLINK :



Graphe rqt sur le netbook