Documentation technique  
Communication MQTT

La communication entre le superviseur et les 3 automates est un module indispensable afin de pouvoir contrôler et d’accéder aux données des robots à tout moment, principalement pour obtenir l’état du robot ainsi que sa position. Pour effectuer cette communication le plus efficacement possible, nous nous sommes dirigés vers le protocole de messagerie publish-subscribe MQTT (Message Queuing Telemetry Transport) basé sur le protocole TCP/IP.

Le client MQTT est conçu pour les connexions aux sites distants, ce qui nous est grandement utile dans le cadre de la fabrication de notre flotte de robots.

En ce qui concerne le broker, nous nous sommes dirigés vers eclipse-mosquitto qui est un broker open source utilisant le protocole MQTT.

1. **Mise en place de l’environnement MQTT**

Pour commencer, nous devons mettre en place l’environnement MQTT sur chacun des robots. Cela nous permettra ensuite de pouvoir transmettre des données et effectuer une connexion entre les différents opérateurs.

Pour cette partie du projet, nous travaillerons sur linux, plus précisément sur Ubuntu 16.04. Nous allons installer MQTT ainsi que Mosquitto sur les 3 robots, serveur, préparateur et accueil, ainsi que sur le superviseur. Pour ce faire, des simples lignes de commande suffisent1.

$ pip install paho-mqtt

$ sudo apt-get install mosquitto mosquitto-clients python-mosquitto

Une fois le client et le broker installé, nous pouvons commencer directement à envoyer et à réceptionner des messages.

1. **Envoi et réception**

Grâce à la communication effectuée à la différente partie, il ne nous reste plus qu’à créer nos programmes qui nous permettront de remplir cet objectif.

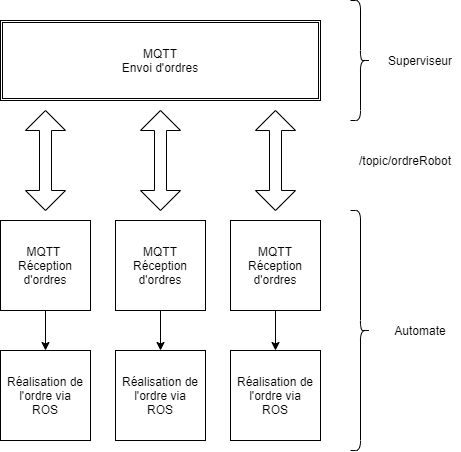
Le broker Mosquitto propose une librairie Python et C qui permettent d’implémenter le client MQTT.

Les programmes peuvent être programmés en Python ou en C, qui sont les langages principaux de Mosquitto, grâce aux librairies proposés par le broker. Nous nous concentrerons donc sur ces 2 langages.

Nous allons effectuer d’une part **la réception des données par le robot de la part du superviseur**, qui se caractérise par la réception et la réalisation d’un ordre par le robot, ainsi que **la réception des données par le superviseur de la part du robot**, qui se caractérise par la réception de l’état et du niveau de batterie du robot.

**A. Réalisation d’un ordre par le robot**

La communication entre le robot et le superviseur s’effectue de la façon suivante :



L’envoi et la réception de l’ordre s’effectue via un topic, ici */topic/OrdreRobot*. Le superviseur est le *publisher*, qui envoie un message sur le topic, tandis que notre robot est un *suscriber* de ce topic, qui lit les informations reçues.

1. **Programme implémenté sur chacun des robots**

#!/usr/bin/env python3

import paho.mqtt.client as mqtt

import paho.mqtt.subscribe as subscribe

import os

def on\_connect(client, userdata, flags, rc):

print("Connected with result code "+str(rc))

if rc==0:

print("Connexion réussie")

else:

print("Connexion impossible")

client.subscribe("topic/OrdreRobot", qos=2)

def on\_disconnect(client, userdata, rc):

print("Client Got Disconnected")

def on\_message(client, userdata, msg):

print("Le message suivant a bien été reçu : " , str(msg.payload))

print("message topic = ",msg.topic)

print("message qos = ",msg.qos)

print("message retain flag = ",msg.retain)

if ((str(msg.payload.decode("utf-8"))) == "1"):

print("Lancement de roslaunch superviseur superviseur.launch")

os.system("roslaunch superviseur superviseur.launch")

if \_\_name\_\_ == '\_\_main\_\_':

client = mqtt.Client()

client.connect("192.168.1.4",1883,60)

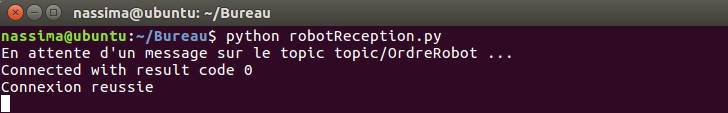
client.on\_connect = on\_connect

print("En attente d'un message sur le topic topic/OrdreRobot ...")

client.on\_message = on\_message

client.loop\_forever()

En lançant le programme Python sur le robot, on obtient un automate qui est en réception continu en attendant de lire un message sur le topic /topic/OrdreRobot.



1. **Programme implémenté sur le superviseur**

#!/usr/bin/env python3

import paho.mqtt.client as mqtt

def on\_connect(client, userdata, flags, rc):

print("Connected with result code "+str(rc))

if rc==0:

print("Connexion réussie")

else:

print("Connexion impossible")

client.publish("topic/OrdreRobot", "1", qos=2)

def on\_disconnect(client, userdata, rc):

print("Client Got Disconnected")

if \_\_name\_\_ == '\_\_main\_\_':

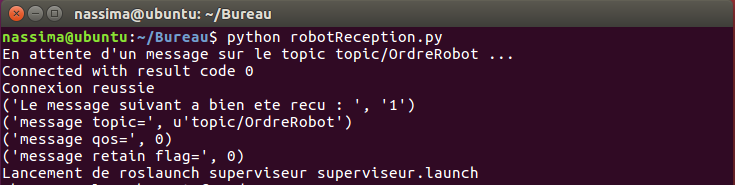
client = mqtt.Client()

client.connect("192.168.1.4",1883,60)

client.on\_connect = on\_connect

client.on\_disconnect = on\_disconnect

Le lancement de ce programme permet la publication *unique* d’un message sur le topic OrdreRobot. Nous avons configuré le programme de réception de tel sorte à ce qu’il réagisse lorsqu’il reçoit le message « 1 » de la part du superviseur. Ainsi, une fois le message lancé, on obtient le résultat suivant dans le programme de la réception situé sur le robot :



1. **Explication détaillée des programmes**

Pour connecter les 2 automates, nous utilisons l’adresse IP 192.168.1.4, qui est l’adresse de notre superviseur. Il est indispensable que tous les robots se connectent à la même adresse IP afin de communiquer entre eux.

Une fois connectés, le robot se met en mode réception en continu grâce à la boucle *client.loop\_forever()*. Le message envoyé, quant à lui, cependant, ne se lance qu’une fois dans le cas de l’envoi d’un ordre, du fait que les robots n’effectuent pas toujours la même action.

Une fois le message lancé par le superviseur, le robot analyse si le message est le chiffre « 1 ». Si c’est le cas, il effectue l’action désirée par le superviseur. Ici, l’action désirée est le lancement du roslaunch *superviseur.launch*, qui est un outil permettant de lancer plusieurs nodes ROS localement.

Le contenu de notre launcher est le suivant :

<launch>

<node name = "drive" pkg = "robotino\_rest\_node" type = "omnidrive.py" output = "screen"/>

<node name = "cmdvel" pkg = "rostopic" type = "rostopic" args = "pub -1 /cmd\_vel geometry\_msgs/Twist ‘{linear : {x : 4.0, y : 0.0, z : 0.0}, angular : {x :0.0,y :0.0,z : 5.0}}’" output = "screen"/>

</launch>

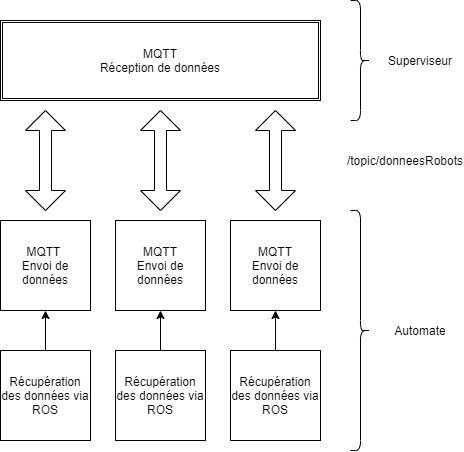
Pour nos tests en cours de réalisation, le launcher permet de lancer le node omnidrive.py, qui lit les vitesses du robot, ainsi que de lancer une publication sur le topic /cmd\_vel afin de faire effectuer au robot un cercle.

Enfin, nous avons décidé d’un QoS (Quality of Service) pour les messages envoyés et reçus via MQTT. Le QoS est un accord entre l’expéditeur et le destinataire sur la garantie de remise d’un message. Il existe 3 niveaux de QoS : 0, 1 et 2. Le niveau 0 correspond à la méthode d’envoi de message la plus simple et la moins lourde : le client publie le message sans accusé de réception. Le niveau 1 garantit que le message sera transféré au moins une fois. Dans le cas où l’accusé de réception est perdu, cela signifie que l’expéditeur renverra le message. Il est donc possible de recevoir plusieurs fois le message. Enfin, le niveau 2 est le niveau de service le plus élevé grâce à une séquence de 4 messages, afin de confirmer que le message a été envoyé, et que l’accusé de réception a bien été reçu. Le message sera donc bien envoyé une seule et unique fois.

Dans notre cas d’envoi d’ordres, nous avons décidé d’utiliser le niveau 2, soit le niveau le plus élevé. En effet, dans le cas où il nous faudrait envoyer un ordre à un robot, il est indispensable que le message soit bien reçu, donc l’accusé de réception est primordial, mais il est également très important à ce que le message ne soit reçu qu’une fois, afin d’éviter que le robot serveur serve plusieurs fois la même commande à la même table, par exemple.

**B. Réception des informations des robots**

Dans ce second cas, la communication s’effectue de la même manière que précédemment, à la différence que l’envoi s’effectue cette fois de la part du robot, qui envoie son état (0 ou 1 en fonction de son état actuel) ainsi que son niveau de batterie.



Ainsi, cela nous permettra de lire l’état et le niveau de charge de la batterie de chaque robot à une certaine fréquence que nous choisirons et de savoir à tout moment si le robot nécessite d’être chargé ou si un robot est libre d’utilisation.

1. **Programme implémenté sur chacun des robots**
2. **Programme implémenté sur le superviseur**
3. **Explication détaillée des programmes**
4. **Intégration de nos programmes sur l’IHM**

Bibliographie

1 Installation MQTT : <https://pypi.org/project/paho-mqtt/>