Communication Logbook

# Initial idea

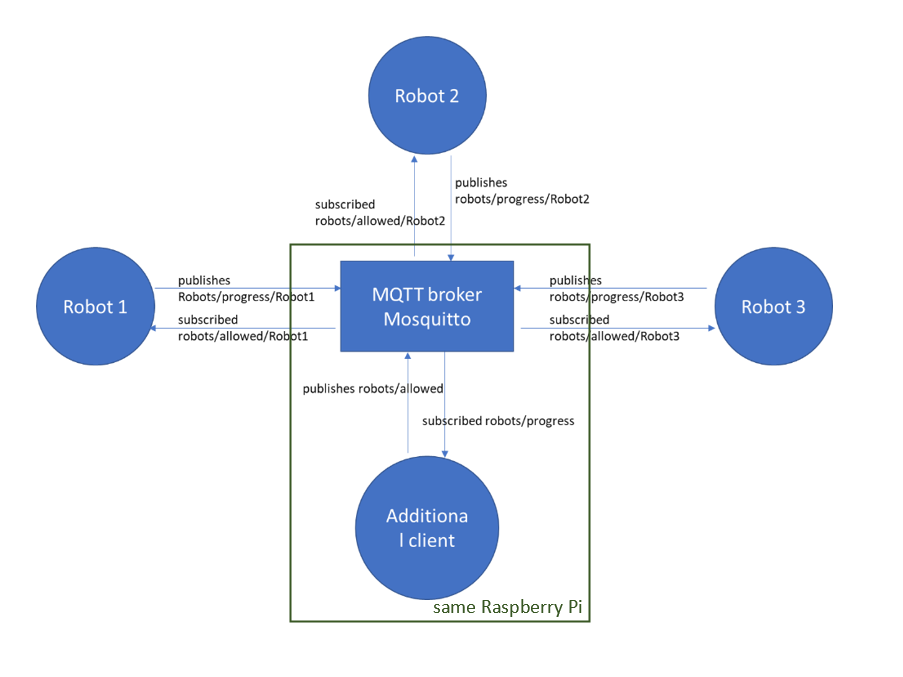
We are going to have an additional Raspberry Pi that will operate as a MQTT broker and it will also host an additional MQTT client. This additional client maintains a list of each client and the number of disks it has processed. It is also the one that computes whether a given robot has done more than (or less than) 2 disks than any other robot. So, instead of the robots keeping track of the work of the others and making the calculation on their own, the calculations are done on the additional raspberry pi.

The first idea (that later turned out to not be really suitable) is the following:

The additional client is subscribed to the topic ‘robots’ and when a robot connects to the broker, it publishes its name in this topic. In this way, when an arbitrary robot (named robot1) connects to the broker, the additional client subscribes to the topics ‘robot1’ and ‘request\_robot1’. When robot1 processes a disk, it publishes 1 in the topic ‘robot1’. When robot1 wants to know if it is allowed to process a disk, it sends 2 to the topic ‘request\_robot1’. As the additional client is subscribed there, on a message that is 2, it computes whether robot1 can do the task and publishes 1 or 0 in the same topic. Thus, robot1 is able to know if it can process the disk. In this way, our robots will do less work.

To test the idea, I installed Mosquitto and the Eclipse Paho MQTT Python client library. I have also implemented code for the additional client using paho-mqtt, as well as a ‘simulation’ of a robot capable of publishing to and subscribing from the topics, emulating disk processing and asking if it can process a disk. Multiple instances of the simulated robots were employed, and it seemed to work as intended.

# Change of topic organization and messages



Imitating requests will not work well. Moreover, in this version, the topics have a hierarchical structure.

Each robot is subscribed to robots/allowed/{the robot’s name}. The additional client is initially subscribed to robots/progress/#. In this way, we do not need the ‘robots’ topic anymore.

When a robot connects, it publishes 0 in its progress topic. The additional client receives this message, and adds the robot to the dictionary of robots with corresponding value 0 (processed disks).

When a robot processes a disk, it publishes 1 in the progress topic and then the additional client publishes in all robots/allowed topics whether the given robot can process the next disk.

When a robot disconnects, it is removed from the dictionary of the robots, and added to the group of disconnected robots. This is done by specifying the last will when connecting to the broker, so that when the robot is disconnected, a message with its name is published to the disconnected topic, and the additional client knows that.

version1.py

## First Testing

Tested with another group. Our groups’ robots were not finished, so we only tested it with the robot.py script on the groups’ Raspberry Pis. To do that, I had to install Mosquitto and configure it so that the robots can access it.

# Handling disconnects on the robot’s side

When connecting to the broker, each robot sets the keepalive to 10s. When a robot disconnects, this is detected by the keepalive. If the disconnect can be fixed (the robot can soon be reconnected), the robot is able to reconnect automatically. This was done by changing the on\_disconnect callback function.

robot.py

# Detecting unrealistic actions

We agreed that no robot is able to process more than 3 disks per 10 seconds, so we have to consider the rate at which robots’ send “I processed a disk” message. I added an additional map with the three robots as keys and the number of processed disks in the last 10 seconds : whenever a robot processes a disk 1 is added to its value and after a timeout of 10 seconds, 1 is removed. Every time 1 is added, it checks whether the value becomes 3 and if so, it sends an error message in the error topic. Whenever the robot receives an error message it disconnects and sets the variable that stores whether it can work to 0.

version2.js

# Having work shifts and breaks

The robots are going to work for 5 minutes and then have a 1 minute break and so on. The first working shift starts when the first robot joins. When the time of operation ends, each robot’s progress goes back to 0 and after the break they start over.

version3.js

## Testing with the robots

Tested it with the robot of another group, the robots were able to connect, receive whether they can process the next disk, signal their actions to the broker, etc.