# Application

This application will connect a simulated robot buggy that will move across different imaginary rooms (due to no access to a physical robot), a System running on a laptop that will record the location and movement of the buggy, and a phone application that is able to control the movement of the robot and set the way of logging. A breakdown of each machine code and functions will be provided later in this document. The simplified application diagram is showed on Figure 1.   
  
For a Video overview of the application and the code see: <https://youtu.be/OJMQ8A6jKzA>  
For the code and other files see: <https://github.com/HefEsti/SOFT564.git>

## MQTT

The application uses MQTT as its main form of wireless communication. MQTT is a lightweight (great solution for limited bandwidth applications) publish/subscribe protocol based on TCP/IP.

The system requires a broker (Server) to run on. Most companies will have their own private servers but for this application a public broker (***HiveMQ***[[1]](#footnote-1)) will be used as a proof of concept. Each machine that we would like to connect in our distributed system will need a client running on it. These connections can be done on *“tcp:// broker.hivemq.com: 1883”* To manage the MQTT connections in this application, ***Eclipse Paho MQTT Python client library***[[2]](#footnote-2) is used.

**HiveMQ broker***address: “broker.hivemq.com” port: 1883*

**PC**

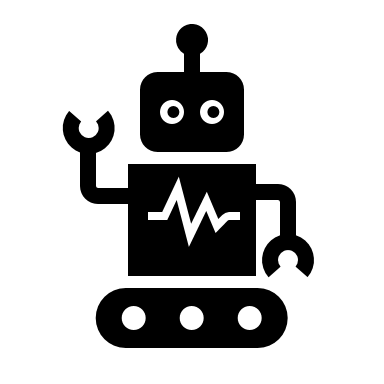
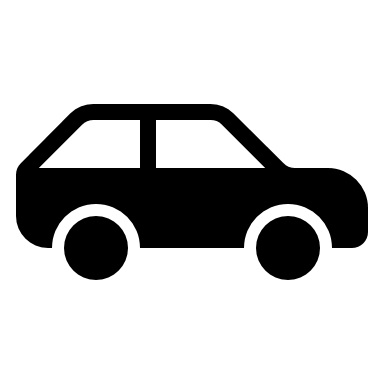
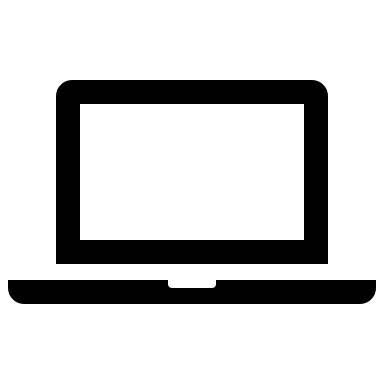
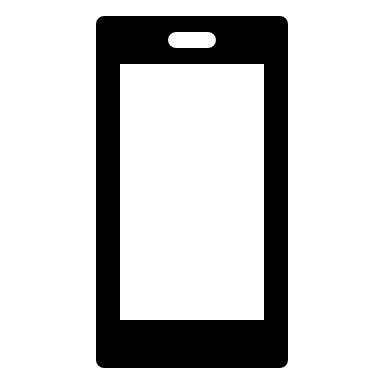
Send Position and Time of message

Set Logging Mode

**Android control App**

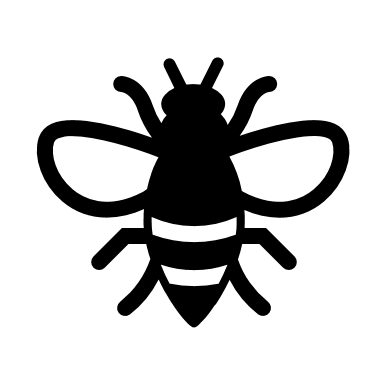
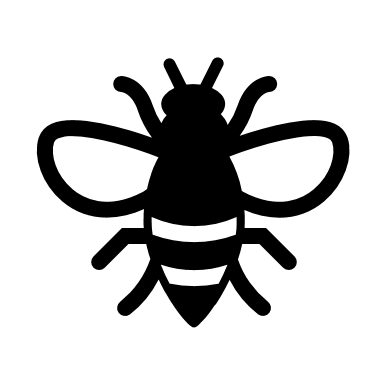
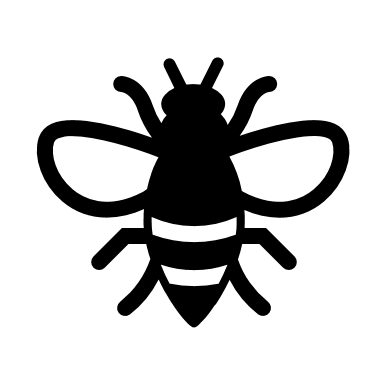
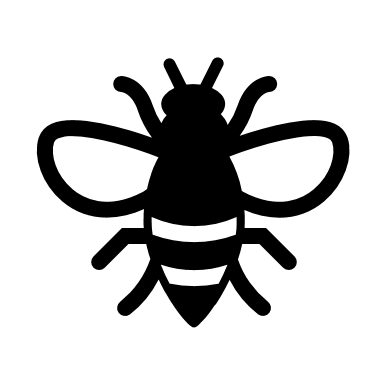
Send Movement Instruction

Send Position and Time of message



**Virtual robot buggy**

**MQTT**



Sends movement info

Figure 1. Simplified application diagram

## Robot buggy

Our imaginary robot buggy will move around the house according to what movement the user chose. The robot can move forward, backwards, left and right. Whenever it enters a new room It will send a signal to the PC, so it can be logged. The message will contain the room and the time it entered.

### Robot.py

Robot is subscribed to the ‘robot/instruction ’topic where it receives the users commands of which direction it shall move. Payload movement commends are ‘left’ ‘right’ ‘forward’ ‘backwards’.

After each of these options the code currently only prints a message with the chosen direction as a proof of concept but later the actual servo movements and additional code can be implemented.   
After the print/move the robot will send a message to the PC of the completed movement on the ‘robot/movement’ so the PC is able to log it. In case of unrecognised command, the robot will not move and send an “unrecognised command” message to be logged.

To simulate the robot entering a new room, a “randomised room generator” is created in the “sending() method”. It will randomly select a room name out of 6 predefined rooms. The rooms are selected every 5-10 seconds. When the buggy ‘enters a new room’, it will create a json file with its location and time it entered. This will is published on the ‘robot/info’ topic so it can be picked up and logged by the PC.

Robot client is subscribed:

* robot/instruction

Robot client publishes on:

* robot/movement – (completed movements)
* robot/info – ( Json file with location and time)

## PC logging base

The PcCode method is responsible for setting the Logging mode from the Android app and logging all incoming robot information accordingly.

### mqtt.json

This file is a configuration file, so in a case where a subscription topic or broker address needs to be changed, it can be simply done in one place.

### ConfigHandler.py

The ConfigHandler module will retrieve all the configurations that were placed in the "mqtt.json" file.

### PcCode.py

The Method will firstly retrieve all the config information and save them into variables for later use. It will also create the logging file if its not already created.   
After this, it will create its own client and connect to the given broker, subscribing all the given topics as well. After this, the code will wait for any incoming message.  
  
On an incoming message (depending on the message topic) it will call either the “mode\_selection” (if the topic is "logging/mode" ) method or the “logging\_robot\_info” (is topic is"robot/info") .   
In mode selection, it will set the Logging mode according to the incoming message’s payload. (“m1”-Location and time, “m2”-Location only, “m3”-Time only).   
In logging\_robot\_info, it will log the incoming robot information according to the set Logging mode.  
Whenever the robot moves, it will send a message on "robot/movement". If the incoming message topic is this, the method will simply log the movement that the robot has just completed.

* Subscribed topics (as set in mqtt.jon):
  + “logging/mode",
  + "robot/info",
  + "robot/movement"
* No published topics

## Android control and monitoring app

For the Android app, the ***“MQTT Dash” app by Routix Software***[[3]](#footnote-3) was used as a basic interface. The UI elements of the app allows to interact with the PC and the buggy over MQTT.

4 observation and control modules have been created:

* Room monitoring module
  + Allows to see which room the robot has been last
* Time monitoring module
  + Allows to see what time the robot communicated last
* Logging mode selection module (multiple choice box)
  + The user can choose which mode the Logging on the PC should be done. This module will publish on the ‘logging/mode’ topic and will have a different payload for each mode. (“m1”-Location and time, “m2”-Location only, “m3”-Time only)
* Buggy movement selection module
  + The user can control the buggy with these movement options. This module will publish on ‘robot/instruction’ and will have a different payload for each movement option (‘‘left’ ‘right’ ‘forward’ ‘backwards’).

To setup the Android app follow the steps in the “AndroidSetupTutorial.doc”

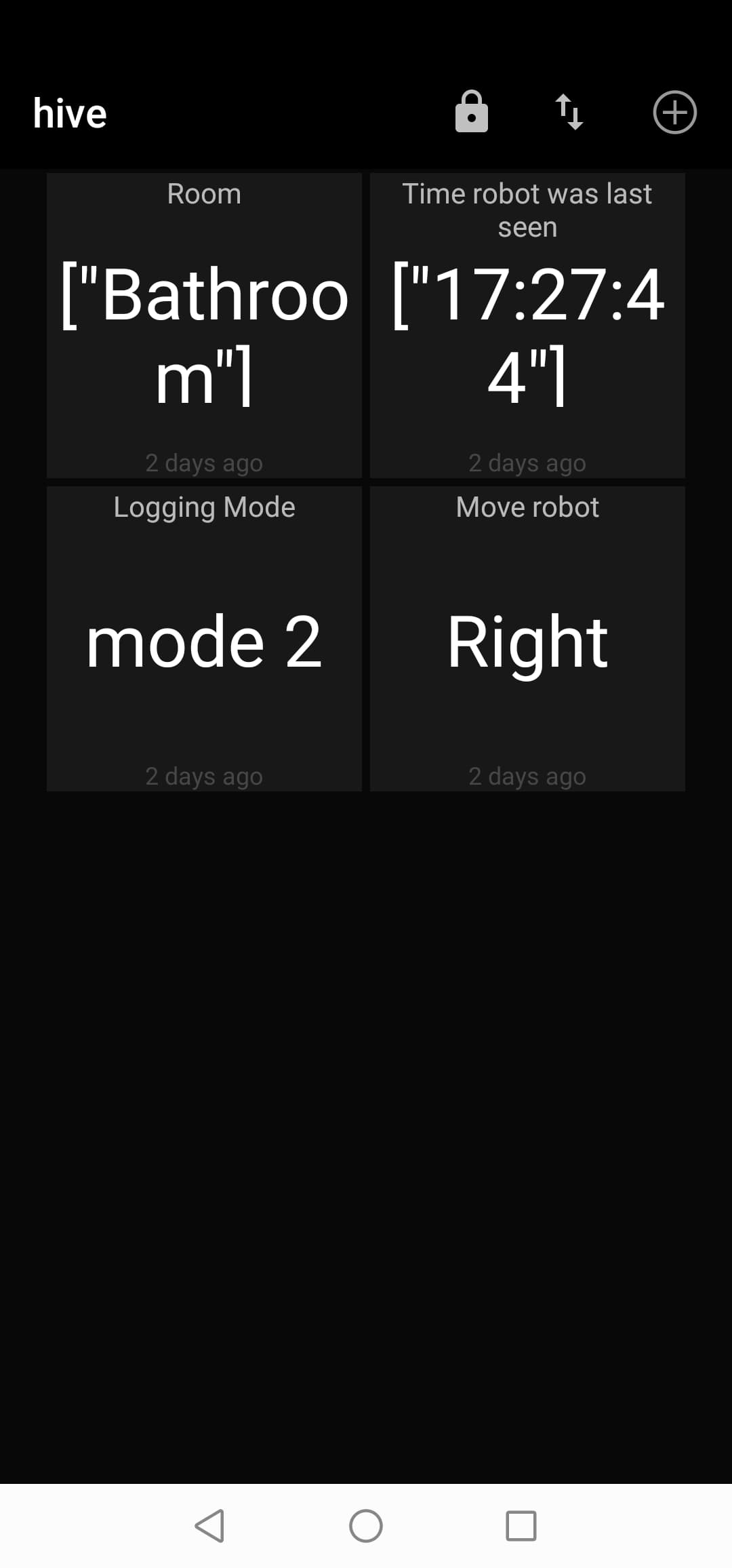
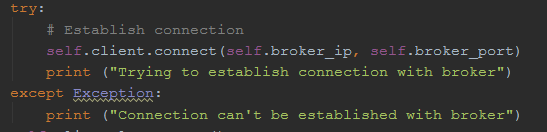


Figure 2 Screenshot of the Android app showcasing the 4 modules

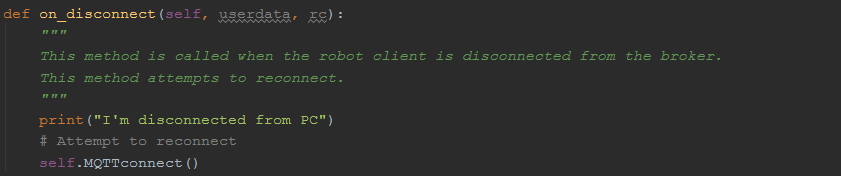
## Testing and Security

The code has been tested during and after the development phase. Tests include connections and message handling. To test the robot code, multiple messages were sent to the PC including messages sent at the same time. All messages has been picked up and logged.

To avoid the application getting stuck or crashing, exceptions have been implemented over the code when trying to establish connection with the broker.



If the client disconnects, it will be forced to try and reconnect again.



Safety measures were also included in case the user sends an unrecognised command to the robot.



1. https://www.hivemq.com/public-mqtt-broker/ [↑](#footnote-ref-1)
2. https://www.eclipse.org/paho/ [↑](#footnote-ref-2)
3. <https://play.google.com/store/apps/details?id=net.routix.mqttdash&hl=en> [↑](#footnote-ref-3)