

Bachelor's thesis

Information and Communications Technology

2022

Samuli Gratscheff

# Integrating Robotics With a Smart Home Environment



Bachelor's Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2022 | 37 pages

Samuli Gratscheff

## Integrating robotics with a smart home environment

This thesis researches the possibility of integrating a Robot Operating System (ROS) with a smart home environment. The smart home environment used in the implementation was Home Assistant because it is the most common open-source smart home platform and is able to use most of the smart home appliances on the market.

The connection between ROS and Home Assistant was made with MQTT (Message Queuing Telemetry Transport) messaging protocol because of its reliability and lightweightness. To simulate a robot an ESP32 DevKit C microcontroller was used. MATLAB was acting as a ROS master, both for its reliability and ability to handle complex calculations if needed in the future.

The implementation was successful and showed that it is possible to connect ROS with Home Assistant through MQTT messages. The implementation was a proof of concept, but if taken further, it would be possible to make a robot that could respond to any event registered by a smart home appliance.

Keywords:

Robot Operating System, smart home, MQTT, microcontroller, Home Assistant

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintäteknikka

2022 | 37 sivua

Samuli Gratscheff

## Robottiikan integrointi älykotiympäristöön

Opinnäytetyössä tutkitaan Robot Operating System (ROS) integroimista älykotiympäristöön. Älykotiympäristönä toimii Home Assistant, joka on yleisin avoimen lähdekoodin älykotialusta ja mahdollistaa suurimman osan markkinoilla olevien älykotilaitteiden käyttämisen.

Yhteys ROSin ja Home Assistantin välillä luotiin käyttämällä MQTT (Message Queuing Telemetry Transport) -viestintäprotokollaa, sen ollessa luotettava ja kevyt. Robotin simuloimiseksi käytettiin ESP32 DevKitC -mikrokontrolleria. MATLAB-ohjelmisto toimii ROS-masterina luotettavuuden sekä edistyneiden toimintojen vuoksi, jos niitä tarvitaan tulevaisuudessa.

Toteutus oli onnistunut ja näytti yhteyden ROSin ja Home Assistantin välillä olevan mahdollinen MQTT-viestejä käyttämällä. Toteutus oli ensisijaisesti todiste konseptin toimivuudesta. Jos toteutuksen veisi pidemmälle, olisi mahdollista tehdä robotti, joka reagoi mihin tahansa älykotilaitteen rekisteröimään tapahtumaan.

Asiasanat:

Robot Operating System, älykoti, MQTT, mikrokontrolleri, Home Assistant

# Contents

<b>List of abbreviations</b>	<b>7</b>
<b>1 Introduction</b>	<b>8</b>
<b>2 Technology background</b>	<b>9</b>
2.1 Robot Operating System (ROS)	9
2.2 MQTT	10
2.3 Home Assistant	10
2.4 Node-RED	11
2.5 MATLAB	11
2.6 ESP32 Microcontroller	11
<b>3 Requirements</b>	<b>12</b>
3.1 Must have requirements	13
3.2 Should have requirements	13
3.3 Could have requirements	13
3.4 Will not have requirements	13
<b>4 Architecture</b>	<b>14</b>
<b>5 Implementation</b>	<b>16</b>
5.1 Necessary software and tools	16
5.1.1 MATLAB Installation	17
5.1.2 ROS Installation	17
5.1.3 Home Assistant installation	18
5.1.4 Node-RED and Mosquitto installation	18
5.2 Starting the ROS master	18
5.3 Connecting to ROS master	19
5.4 Creating a workspace and package	20
5.4.1 Creating a workspace	20
5.4.2 Creating a package	21
5.5 Setting the MQTT broker	21

5.6 Setting the MQTT ROS node	22
5.7 Connecting the ESP32 to ROS	25
5.7.1 Rosserial installation to Arduino IDE	26
5.7.2 Adding ESP32 board to Arduino IDE	26
5.7.3 Communication	29
<b>6 Improvements and future work</b>	<b>33</b>
<b>7 Conclusion</b>	<b>34</b>
<b>References</b>	<b>35</b>

## Figures

Figure 1. ROS publisher and subscriber exchange.	9
Figure 2. Architecture.	14
Figure 3. ROS to Home Assistant message path.	15
Figure 4. Ubuntu Software repositorie permissions.	17
Figure 5. MATLAB ROS master launch command.	19
Figure 6. MATLAB ROS master information.	19
Figure 7. Node-RED flow.	22
Figure 8. Node-RED debug window.	22
Figure 9. MQTT Node running on terminal.	24
Figure 10. Node-RED inject node button.	24
Figure 11. MQTT node receiving message.	25
Figure 12. Node-RED debug receiving and sending a message.	25
Figure 13. Arduino IDE preferences.	27
Figure 14. Arduino IDE Board manager.	28
Figure 15. Selecting ESP32 Dev Module from the boards list in Arduino IDE.	28
Figure 16. LED on.	32
Figure 17. LED off.	32

## **Tables**

Table 1. Requirements for implementation.	12
Table 2. Software and tools for the implementation.	16

## List of abbreviations

IDE	Integrated development environment
IP	Internet Protocol
LCM	Lightweight Communications and Marshalling
LED	Light-emitting diode
MQTT	Message Queuing Telemetry Transport
ROS	Robotic Operating System
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
USB	Universal Serial Bus

# 1 Introduction

Home automation has become more common, and the amount of smart home device manufacturers is growing fast. Manufacturers such as Philips, IKEA and Sonoff have made smart home devices popular with their ease of use. The problem with these systems is that compatibility with other manufacturers devices is not clearly stated and might limit the users to a specific manufacturers devices. For more freedom and still being able to use all these devices, Home Assistant was used.

Robots are also becoming more common in households. Robots like iRobot's Roomba has been around since 2002 [1], but the catalogue has expanded to, for example, window cleaning robots, pool cleaning robots and robot lawn mowers. These robots also use their own respective apps and will not communicate with other manufacturers devices. There are integrations in Home Assistant for some of the most popular robots such as the iRobot devices made by the community. Despite this, in this thesis a new robot will be developed with the Robot Operating System (ROS).

This thesis will research whether it is possible to integrate robotics with a smart home device environment. The thesis will also cover a demonstration, where a simple robot will be integrated with a smart home environment.

This thesis is divided into five main chapters in addition to the introduction. In chapter 2 the backgrounds of the technologies used in the implementation are briefly described. Chapter 3 defines the requirements of the implementation using the MoSCoW-method. Chapter 4 describes the architecture of the implementation. Chapter 5 is the implementation, which describes the process in a step-by-step basis. Chapter 6 describes the future possibilities and possible improvements of the implementation.



## 2 Technology background

In this chapter, the key technologies utilized in this thesis are briefly discussed.

### 2.1 Robot Operating System (ROS)

ROS is an open-source development kit, which includes a vast number of libraries for robotics development. It provides a node-based communication as one of its communication methods, which will be used in the demonstration [2]. In Figure 1 it is demonstrated how ROS nodes communicate with each other. ROS master keeps track of the topics, publishers and subscribers. ROS master in itself is also a node [3].

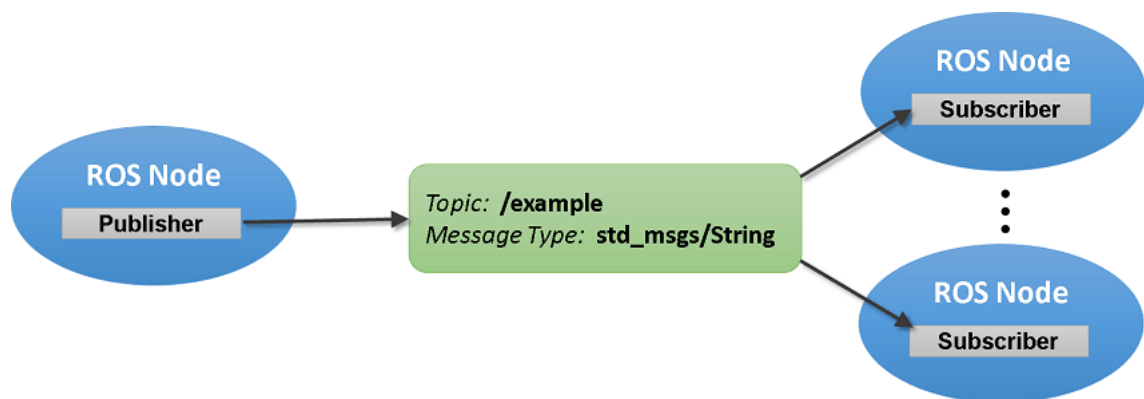


Figure 1. ROS publisher and subscriber exchange.

Source: <https://se.mathworks.com/help/ros/ug/exchange-data-with-ros-publishers-and-subscribers.html>

ROS is a meta-operating system, and runs on UNIX-based operating systems, mainly tested on Ubuntu and Mac OS X, although community has provided support for other operating systems as well [4].

ROS has two major branches, ROS 1 and ROS 2. The branch used in this thesis is ROS 1. ROS 2 does not use the same communication model as ROS 1, as it does no longer have ROS master. It eliminates the problem that ROS 1 has, that

if the ROS master is unavailable, the other nodes cannot communicate with each other. Other changes include for example the use of newer C++ features used in its API [5]. During this thesis, ROS 1 is used.

Other similar systems that utilize the subscriber/publisher model and used in robotics include LCM and ZeroMQ. LCM was developed in MIT in 2006, and has been used in robotics at Ford and Volvo [6]. ZeroMQ has been used at Microsoft and Samsung and has a large number of supported programming languages [7]. ROS was chosen as the platform for clear documentation and active community.

The ROS version used in this thesis is Noetic 1.15.14.

## 2.2 MQTT

MQTT is also a messaging protocol based on publish/subscribe model, much alike with ROS. Being lightweight, both messages and clients, it is a very suitable protocol for Internet of Things and in the case of this thesis, robotics [8].

## 2.3 Home Assistant

Home Assistant is open-source software, designed to control smart home devices. Home Assistant supports integrations for over 1900 smart home devices and services, and it is constantly growing [9]. Home Assistant is recommended to be run on a Raspberry Pi, but since no smart home devices were used in this implementation, Home Assistant in a virtual machine will be run instead. Home Assistant is not bound to certain manufacturers devices, and with the use of a USB stick supporting Zigbee technology, it can connect to for example IKEA or Philips smart home devices. Zigbee is a wireless technology developed by Connectivity Standards Alliance [10]. It's optimized for low power consumption, which makes it suitable for smart home devices. Its mesh topology is scalable to thousands of nodes, so it's possible to build very large networks [11]. Home Assistant supports many add-ons made by the developer or the community. Node-RED and Mosquitto addons are used to control the MQTT messages.

## 2.4 Node-RED

Node-RED provides a node-based platform which reduces the amount of coding needed to perform certain tasks. Node-RED editor works in the browser and is built with node.js. Node-RED is light-weight and is easily run in a Raspberry Pi and is ported to Home Assistant addon by the community. Node-RED has a large palette of nodes and together with community made nodes the palette exceeds 225 000 modules [12].

Node-RED is used to receive and send MQTT messages in and out of Home Assistant.

## 2.5 MATLAB

MATLAB is a programming language and a platform for data analysis, algorithms and creating mathematical models [13]. For its advanced data analysis tools, it will function as ROS master during this implementation. These tools are not used in this thesis, but for future development, its possible uses could be for example image recognition, which is not possible with the robot's own hardware.

## 2.6 ESP32 Microcontroller

ESP32 is a low power, small microcontroller with integrated Wi-Fi and Bluetooth connectivity [14], which makes it a good option to be used in small robots. The specific ESP32 board used in the implementation is ESP32 DevKit-C.

### 3 Requirements

The requirements are presented here with the MoSCoW -method. Special attention was paid to keep the communication between devices lightweight, since IoT devices are usually low powered and communicate over short distances.

Table 1. Requirements for implementation.

Requirement	Importance	Notes
Communication between MATLAB, ROS and Home Assistant	Must have	Different parts of the implementation must be able to communicate with each other.
Message protocols to be lightweight	Must have	Using MQTT messages as the protocol of communication in conjunction with ROS messages.
Functionality to control physical hardware with ROS messages	Should have	Physical hardware consists on LEDs controlled by a microcontroller unit.
Use an interactive way of controlling the robot	Should have	Control the robot with a button in Home Assistant for example, instead of code or terminal commands.
Home automation with IoT devices	Could have	Different sensors added to the Home Assistant ecosystem, like motion sensors and lights.
Own dedicated hardware	Could have	Instead of a virtual machine, real physical hardware could be used.
Advanced robot functionality	Will not have	The robot will not feature any advanced features, like image recognition or voice control.

Some of the most critical requirements are discussed in more detail in the following subchapters.

### 3.1 Must have requirements

Communication between ROS, MATLAB, Home Assistant and ESP32 need to be possible with lightweight messaging protocols. ROS itself uses rosmessages to communicate between nodes, but for Home Assistant the protocol will be MQTT. Communication between ESP32 and ROS must be possible over Wi-Fi.

### 3.2 Should have requirements

There should be functionality to control physical hardware with rosmessages. The physical hardware does not have to be complex, as a proof of concept a simple LED or a motor will suffice. Physical hardware will be connected to the ESP32 microcontroller.

### 3.3 Could have requirements

This implementation, if successful, will have endless number of possibilities for more functionality. The robot can be built to be more complex, for example feature movement, camera or proximity sensors. Implementation could have a Raspberry Pi powered Home Assistant, instead of a Virtual Machine. In case of Home Assistant being on a Raspberry Pi, there is the possibility to add a Zigbee USB stick, and connect different smart home devices to Home Assistant, and control them via rosmessages and MQTT.

### 3.4 Will not have requirements

This implementation will not have a robot including advanced functionality like facial recognition, item recognition or voice control, even though MATLAB has the processing power and tools for them.

## 4 Architecture

In this chapter the architecture of the implementation is described. As seen in Figure 2 below, most of the implementation was made with virtual machines for convenience.

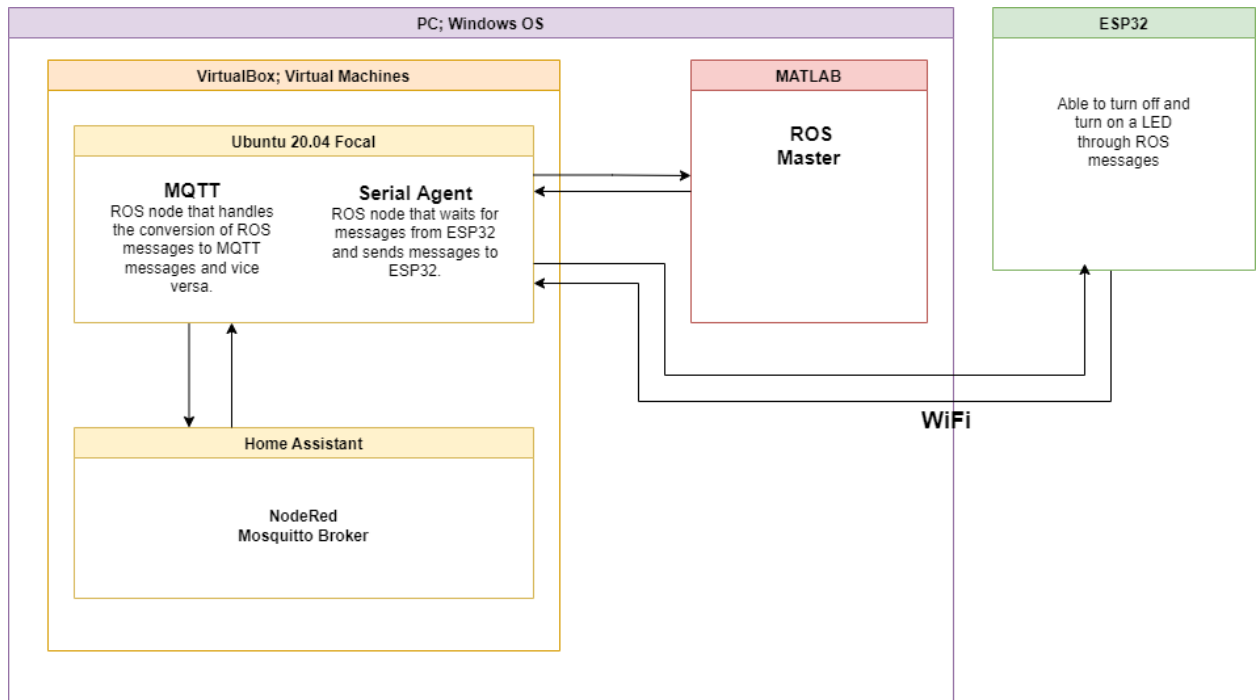


Figure 2. Architecture.

All different components of this demonstration are located inside the same local area network. On this demonstration, Ubuntu and Home Assistant were set up on virtual machines, but often Home Assistant is installed on a Raspberry Pi.

In ROS, all messages are sent to a specified topic. On Figure 3 the path the message will take from a ROS node to Home Assistant is illustrated. ROS to MQTT node contains code, that takes the rosmesssage content, and uses MQTT to send it forward to MQTT Broker, running on Home Assistant.

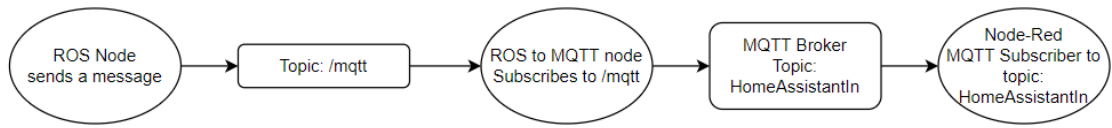


Figure 3. ROS to Home Assistant message path.

## 5 Implementation

The basic idea of the implementation is to research the possibility of integrating robots to the smart home environment. As ROS has become a popular platform for robotics development and is used to teach robotics in universities [15], it was decided to use it instead of a ready robot. There are multiple different smart home solutions on the market, but Home Assistant has the ability to connect to most of the smart home devices on the consumer market.

### 5.1 Necessary software and tools

From the tools needed in the implementation, Ubuntu and Home Assistant could be installed on their own machines. All the devices are connected to the same local area network, and doesn't have the capability to be controlled from the outside.

Table 2. Software and tools for the implementation.

Software / Tool	Role
Windows PC	As a host computer for the virtual machines, as well as running MATLAB instance.
MATLAB	Acts as ROS master.
Oracle VirtualBox	Software for running the virtual machines.
Ubuntu 20.04 Focal VM	Acts as the MQTT to ROS bridge node.
Home Assistant VM	Home automation platform, runs Node-RED and Mosquitto Broker.
ESP32 Microprocessor	Acts as a basic robot. Connection between ROS network and ESP32 is implemented with the rosserial -package.



### 5.1.1 MATLAB Installation

For students of Turku University of Applied Sciences MATLAB is provided for free, with all the official addons. For this implementation, MATLAB needs the ROS Toolbox addon installed. During the MATLAB installation process, the user can choose to install any of the official addons. ROS Toolbox is in that list and makes the installation easy. There was a possibility to use Ubuntu VM as a ROS master, but with MATLAB needing the most processing power, processing the information sent by the other nodes and sending the information nodes needing it, I decided to keep MATLAB as the brains.

### 5.1.2 ROS Installation

ROS is installed with the ROS Toolbox in MATLAB. Before trying to install ROS on ubuntu, its repositories have to be allowed to install software from more sources than the main source.

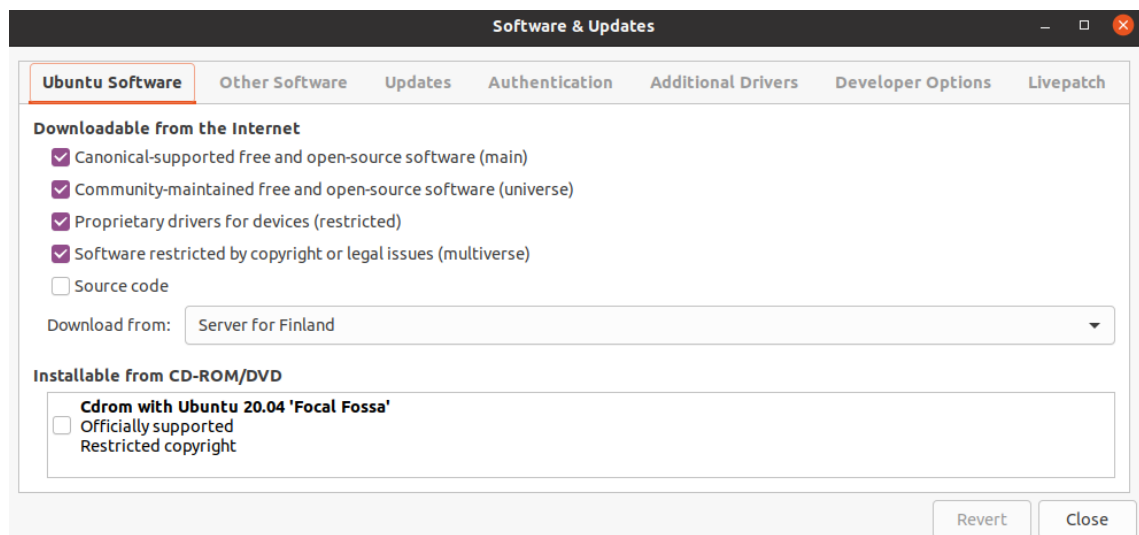


Figure 4. Ubuntu Software repository permissions.

After this change has been made, ROS Noetic can be installed with the following commands [16]:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main"
> /etc/apt/sources.list.d/ros-latest.list'

sudo apt install curl

curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo
apt-key add -

sudo apt update

sudo apt install ros-noetic-desktop-full
```

### 5.1.3 Home Assistant installation

When running Home Assistant on Virtual Box, no actual installation is needed. From the Home Assistant website, there is the possibility to download the .vdi - file containing Home Assistant, and can be opened with Virtual Box [17].

When starting the virtual machine, Home Assistant starts, and the admin account creation process starts. Home Assistant is ready to use.

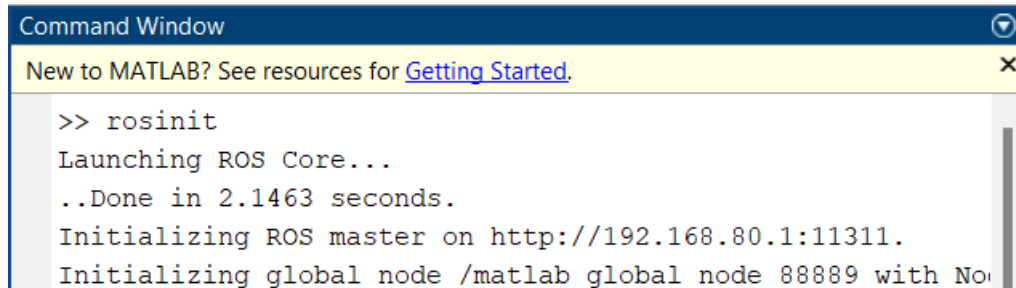
### 5.1.4 Node-RED and Mosquitto installation

Node-RED and Mosquitto Broker are both installed via the Home Assistants own addon store. From the Node-REDs configuration tab, we need to give a credential secret keyword and turn SSL to false. System security is not the focus of this implementation but is an important part of all development. Should the implementation be taken to real use, one should take care of keeping the system secure.

## 5.2 Starting the ROS master

In this thesis MATLAB was chosen to act as the ROS master. The ROS master keeps track of all the topics in use at the ROS environment. MATLAB was chosen as the master, because of all MATLABs possibilities to handle complex algorithms in case for example image processing. This way the robot itself can use less demanding components, and MATLAB can be run on its own server.

Starting the ROS master is initiated by one command: *roscpp*. The command opens a new terminal window, as seen in Figure 6, which shows the details of the ROS master.

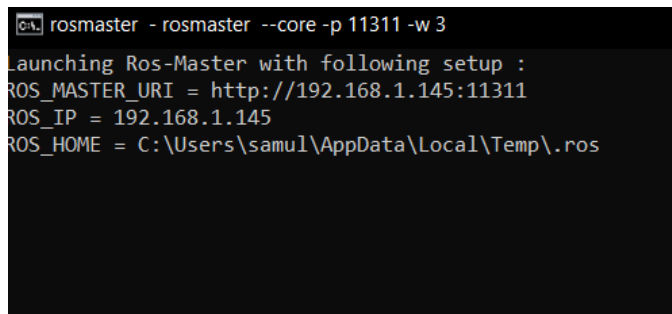


```

Command Window
New to MATLAB? See resources for Getting Started.
>> roscpp
Launching ROS Core...
..Done in 2.1463 seconds.
Initializing ROS master on http://192.168.80.1:11311.
Initializing global node /matlab_global_node_88889 with No

```

Figure 5. MATLAB ROS master launch command.



```

rosmaster - roscpp --core -p 11311 -w 3
Launching Ros-Master with following setup :
ROS_MASTER_URI = http://192.168.1.145:11311
ROS_IP = 192.168.1.145
ROS_HOME = C:\Users\samul\AppData\Local\Temp\.ros

```

Figure 6. MATLAB ROS master information.

The `ROS_MASTER_URI` is the address used by other ROS instances to connect to the master.

### 5.3 Connecting to ROS master

ROS Nodes will be run on an Ubuntu virtual machine. Before ROS can be used in any terminal, the setup script needs to be sourced:

```
source /opt/ros/noetic/setup.bash
```

After sourcing, ROS needs the address of the ROS master. The address can be set by modifying the environment variable `ROS_MASTER_URI`. The current value of the variable can be seen with the `echo` command:

```
echo $ROS_MASTER_URI
```

To set the variable of the ROS master URI, export command is used:

```
export ROS_MASTER_URI=http://192.168.1.145:11311
```

Take note, that the IP address is the IP address of the machine that runs MATLAB. Port 11311 is the default port all ROS masters use.

ROS also uses another environment variable, `ROS_IP`. This environment variable needs to be set for the nodes in the ROS network and is the node machines own IP address.

## 5.4 Creating a workspace and package

Packages are a way to easily organize code. Packages can be reused and shared with others. For example, in this implementation we are creating an MQTT handling node. We could share this node with other ROS users by sharing the package. There are many community-made packages available on the internet.

Packages can be created standalone or organized in a workspace. Workspace is simply a way to organize your ROS environment and makes it faster to get code running if modifications are made to multiple packages.

### 5.4.1 Creating a workspace

The name of the workspace will be 'catkin\_ws'. A workspace can be created with the following commands:

```
mkdir -p ~/catkin_ws/src
```

```
cd ~/catkin_ws/
```

```
catkin_make
```

```
source devel/setup.bash
```

`catkin_make` command makes all the necessary files to your new workspace, and `setup.bash` will overlay the workspace on your environment.

All the commands in the previous steps are found in the installing and configuring instructions from ROS wiki page [18].

### 5.4.2 Creating a package

Before creating the package, one needs to navigate to the workspaces source directory (if workspace have been created, packages can be created without workspaces):

```
cd ~/catkin_ws/src
```

In this implementation, a package will be created with a ready-made script, which comes with the ROS installation. The structure of the package creation command is as follows:

```
catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

On the dependencies, rospy will be needed for us to write the code in python. One can also choose to use C++ or both, by adding roscpp to the dependencies. Std\_msgs is a dependency used to format our ROS messages. MATLAB uses the same message format. Creating the package happens with the following command:

```
catkin_create_pkg mqtt_package rospy roscpp std_msgs
```

After the package has been created, package needs to be built, and added to our ROS environment.

```
cd ~/catkin_ws  
catkin_make  
. ~/catkin_ws/devel/setup.bash
```

### 5.5 Setting the MQTT broker

Previously Mosquitto Broker was installed via the Home Assistants addon store. Node-RED is used to control the MQTT messages and topics. The following flow is created with Node-RED, as seen in Figure 7:

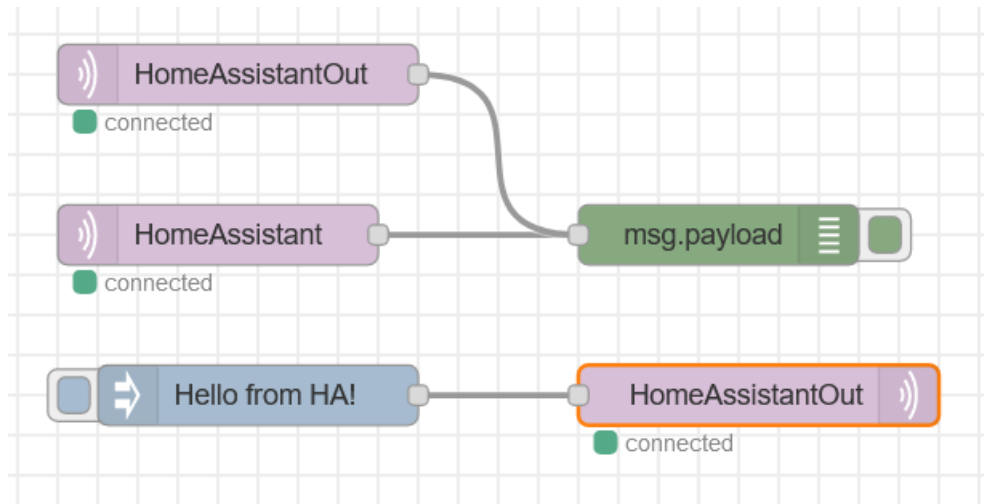


Figure 7. Node-RED flow.

The first HomeAssistantOut node is for debugging purposes. On the bottom, a message can be sent from the inject nodes rectangular button. By sending a message "Hello from HA!" to MQTT topic 'HomeAssistantOut', MQTT broker can be seen working from the debug window, shown in Figure 8.

---

```
4/3/2022, 12:03:46 PM  node: 7482c53555fc449f
HomeAssistantOut : msg.payload : string[14]
"Hello from HA!"
```

Figure 8. Node-RED debug window.

The 'HomeAssistant' node listens to messages in 'HomeAssistant' topic, where the MQTT messages from ROS will be sent. In ROS the mqtt node will subscribe to the 'HomeAssistantOut' topic.

## 5.6 Setting the MQTT ROS node

ROS messages need to be converted to MQTT messages and sent to Home Assistant. The node is created by writing a python program. The code needs to be created inside our package. The code is in the following location on the Ubuntu ROS instance:

catkin\_ws/src/mqtt\_package/scripts/mqtt.py

With the following code the connection between ROS and Home Assistant can be tested:

```
#!/usr/bin/env python
from paho.mqtt import client as mqtt_client
from std_msgs.msg import String
import rospy

# Set the credentials for MQTT connection.
mqtt_username = "username"
mqtt_pw = "password"
mqtt_ip = "192.168.1.33" #location of the MQTT Broker / HomeAssistant
mqtt_port = 1883

#MQTT Callback function, it runs when the MQTT subscriber receives a
message.

def on_message(mqtt_client,userdata,message):
    rospy.loginfo(message.payload.decode())

# ROS Message Callback function, it only runs when the listener() function
notices a new rosmesssage in /mqtt_in topic.
def callback(msg):
    rospy.loginfo("Callback initiated.")
    rospy.loginfo(rospy.get_caller_id() + "I got a message: %s", msg.data)
    rospy.loginfo("Sending info to HomeAssistant...")
    client.publish("HomeAssistant", "Callback function works. Message: " +
msg.data) #This message appears to HomeAssistant, if successful

#Listens for new rosmessages in /mqtt_in topic, aand calls the callback
function if it receives one.
def listener():
    rospy.Subscriber("mqtt_in", String, callback)
    client.on_message = on_message # Prints the ROS message that was sent
to topic /mqtt_in
    rospy.loginfo("listener running")
    client.loop_forever()
    rospy.spin() # Keeps the script running until stopped

#The main function. MQTT is first initialized, and then tries to start the
listener. Sleep used to see the logmessages.
if __name__ == '__main__':
    rospy.init_node("mqtt", anonymous=False)
    pub = rospy.Publisher('mqtt_in', String, queue_size=10)
    Rate = rospy.Rate(1)
```

```

rospy.loginfo("Setting up the MQTT Client...")
client = mqtt_client.Client("ROS_MQTT_CLIENT")
client.username_pw_set(mqtt_username,mqtt_pw)
client.connect(mqtt_ip,mqtt_port)
client.subscribe("HomeAssistantOut")
rospy.loginfo("MQTT Connected")
Rate.sleep()
try:
    listener()
except rospy.ROSInterruptException:
    pass

```

Script can be run with the following command in Ubuntu terminal:

```
roslaunch mqtt_package mqtt.py
```

If there are no errors, the following should appear to the terminal:

```

[INFO] [1648988024.745444]: Setting up the MQTT Client...
[INFO] [1648988024.747447]: MQTT Connected
[INFO] [1648988025.748306]: listener running

```

Figure 9. MQTT Node running on terminal.

In another terminal, a message can be sent to /mqtt\_in topic with the following command:

```
rostopic pub -1 mqtt_in std_msgs/String hello
```

And from Node-RED, an MQTT message can be sent to ROS by clicking the inject nodes button:

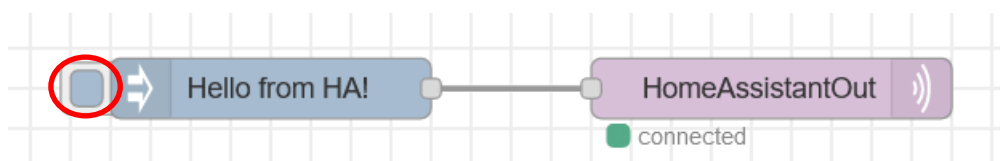


Figure 10. Node-RED inject node button.

If everything is working, the following will appear on the terminal running the listener:



```
[INFO] [1648988122.106009]: Setting up the MQTT Client...
[INFO] [1648988122.107373]: MQTT Connected
[INFO] [1648988123.109372]: listener running
[INFO] [1648988129.922711]: Callback initiated.
[INFO] [1648988129.923777]: /mqttI got a message: hello
[INFO] [1648988129.924789]: Sending info to HomeAssistant...
[INFO] [1648988135.806830]: Hello from HA!
```

Figure 11. MQTT node receiving message.

And in Node-RED's debug window, the message from ROS (hello) can be seen and that the inject node sent the "Hello from HA!" message to ROS:

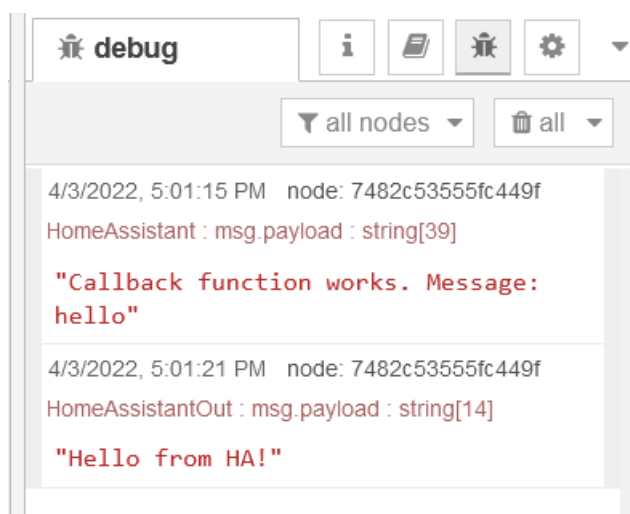


Figure 12. Node-RED debug receiving and sending a message.

Now a functioning MQTT handler node has been setup in ROS.

## 5.7 Connecting the ESP32 to ROS

Connecting microcontrollers to ROS is done by using a ROS package called `rosserial` [19]. As of August 2020, ESP32 was officially announced to be supported by `micro-ROS` and should be used if ROS2 is the platform of choice [20].

### 5.7.1 Rosserial installation to Arduino IDE

ESP32 microcontroller unit will be connected to ROS network with roserial package. Since ESP32 is compatible with Arduino IDE, we can use the roserial\_arduino package. Arduino IDE must be installed on the computer, before installing the package [21].

Rosserial can be installed with the following commands:

```
sudo apt-get install ros-kinetic-roserial-arduino
sudo apt-get install ros-kinetic-roserial
```

After the roserial installation, necessary libraries can be installed with the following commands:

```
cd catkin_ws/src
git clone https://github.com/ros-drivers/roserial.git
cd catkin_ws
catkin_make
catkin_make install
```

Next the ros\_lib folder has to be created, and enable Arduino IDE to interact with ROS:

```
cd <sketchbook>/libraries
rosruntime roserial_arduino make_libraries.py .
```

### 5.7.2 Adding ESP32 board to Arduino IDE

Before a program can be written to connect the ESP32 microcontroller to ROS, Arduino IDE must be installed to use ESP32. For Arduino IDE to be able to install ESP32 board, we need to add the following address to the preferences of the IDE:

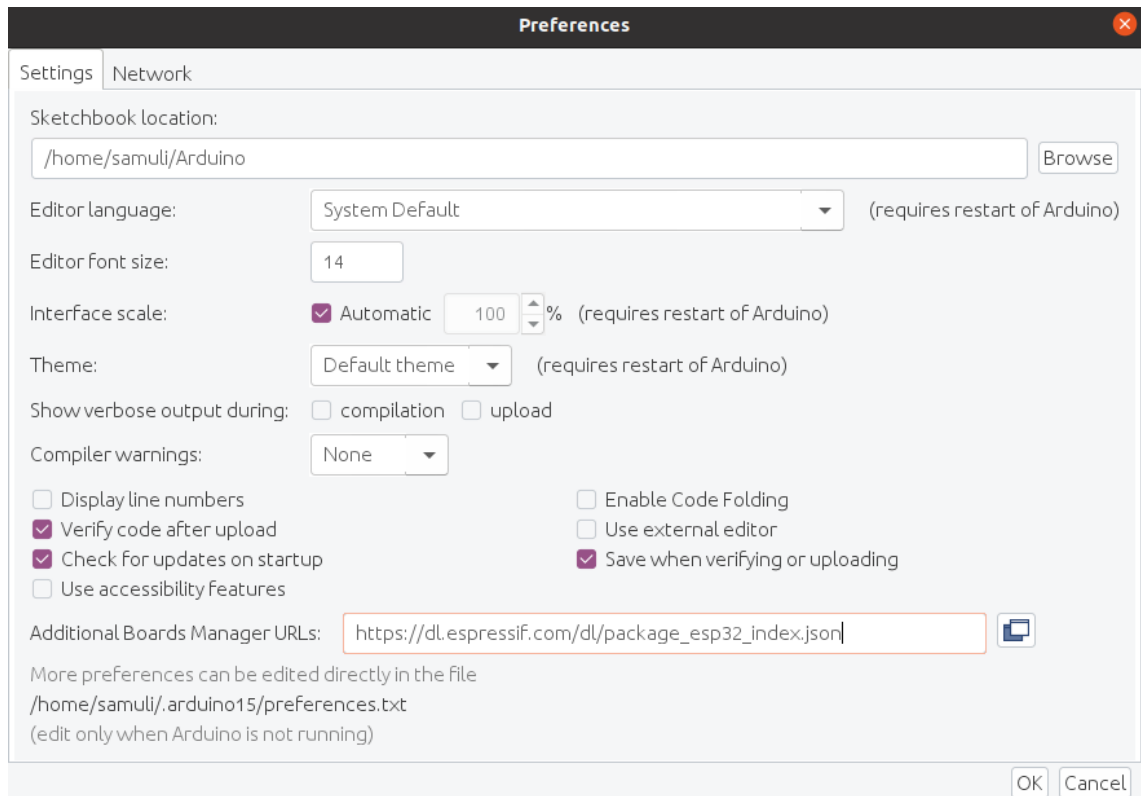


Figure 13. Arduino IDE preferences.

This allows the Arduino IDE board manager to find the ESP32. In the boards manager, we install the ESP32:

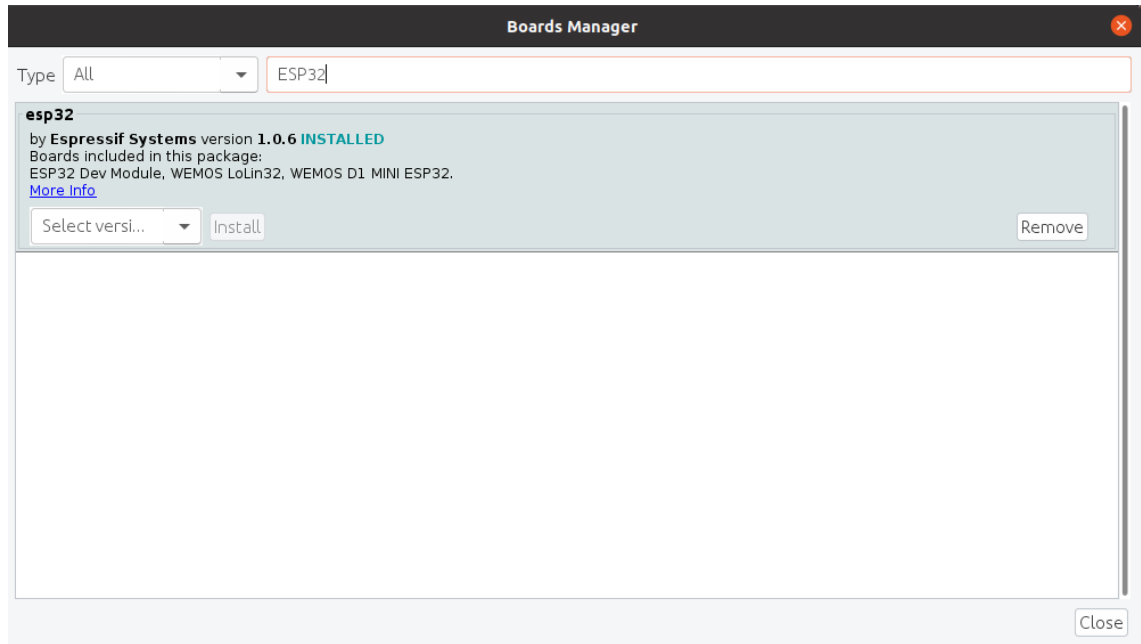


Figure 14. Arduino IDE Board manager.

Now the ESP32 Dev module can be selected from the boards list:

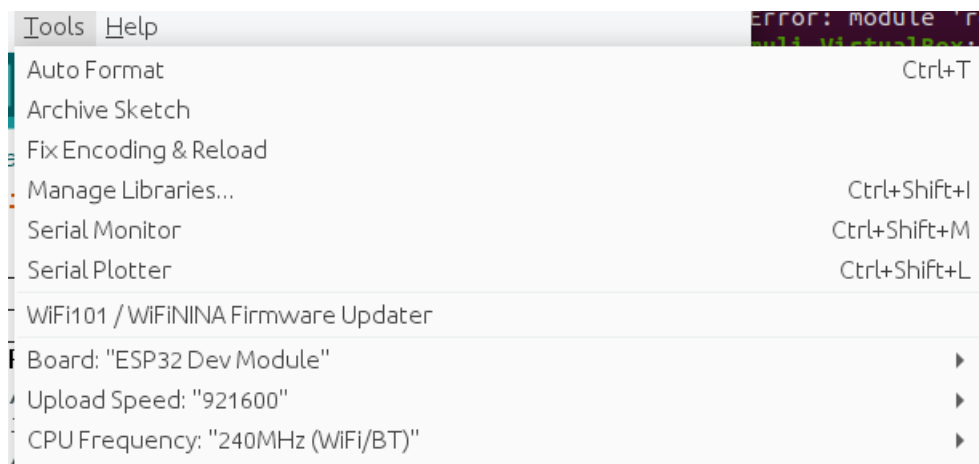


Figure 15. Selecting ESP32 Dev Module from the boards list in Arduino IDE.

ESP32 is ready to be programmed and connected to ROS network.

### 5.7.3 Communication

Rosserial needs a serial node running on ROS. On a new terminal, execute the following command to start the roserial listener to listen TCP messages:

```
roslaunch roserial_python serial_node tcp
```

This node waits for a connection on port 11411. The code which is running on ESP32, will use this port to communicate with the serial node.

With the following code, a led can be turned on or off, by sending a message to /message topic. If the number is 1, the LED is on, and in all other cases it is off.

```
#include <WiFi.h>
#include <ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Int16.h>
#include <std_msgs/Float64.h>
#include <std_msgs/Empty.h>

////////////////////
// WiFi Definitions //
////////////////////
const char* ssid = "WiFi name";
const char* password = "WiFi password";

IPAddress server(192, 168, 1, 240); // ip of your ROS server (Not the master
IP, but the one running serial node)
IPAddress ip_address;
int status = WL_IDLE_STATUS;

WiFiClient client;

class WiFiHardware {
public:
  WiFiHardware() {};

  void init() {
    // do your initialization here. this probably includes TCP server/client
    setup
    client.connect(server, 11411);
  }

  // read a byte from the serial port. -1 = failure
```

```

int read() {
    // implement this method so that it reads a byte from the TCP connection
    and returns it
    // you may return -1 is there is an error; for example if the TCP
    connection is not open
    return client.read();          //will return -1 when it will works
}

// write data to the connection to ROS
void write(uint8_t* data, int length) {
    // implement this so that it takes the arguments and writes or prints
    them to the TCP connection
    for(int i=0; i<length; i++)
        client.write(data[i]);
}

// returns milliseconds since start of program
unsigned long time() {
    return millis(); // easy; did this one for you
}
};

int i;
char hello[13] = "hello world!";
char callback[13] = "callback";

void chatterCallback(const std_msgs::Int16 &msg) {
    Serial.print("Callback function enabled");
    if (msg.data == 1) {
        digitalWrite(18,HIGH);
        Serial.print("Callback up");
    }
    else {
        digitalWrite(18, LOW);
        Serial.print("Callback down");
    }
}

ros::Subscriber<std_msgs::Int16> sub("message", &chatterCallback);
ros::NodeHandle_<WiFiHardware> nh;
std_msgs::String str_msg;
ros::Publisher chatter("chatter", &str_msg);

void setupWiFi()
{
    WiFi.begin(ssid, password);
    Serial.print("\nConnecting to "); Serial.println(ssid);
}

```

```
uint8_t i = 0;
while (WiFi.status() != WL_CONNECTED && i++ < 20) delay(500);
if(i == 21){
  Serial.print("Could not connect to"); Serial.println(ssid);
  while(1) delay(500);
}
Serial.print("Ready! Use ");
Serial.print(WiFi.localIP());
Serial.println(" to access client");
}

void setup() {
  Serial.begin(115200);
  setupWiFi();
  delay(2000);
  nh.initNode();
  nh.subscribe(sub);
  nh.advertise(chatter);
  pinMode (18, OUTPUT);
}

void loop() {
  str_msg.data = hello;
  chatter.publish( &str_msg );
  nh.spinOnce();
  delay(1);
}
```

On a new terminal, a message can be sent to the /message topic with the following command:

```
rostopic pub message std_msgs/Int16 1 --once
```

By changing the number from 1 to any other, we can turn the LED off.

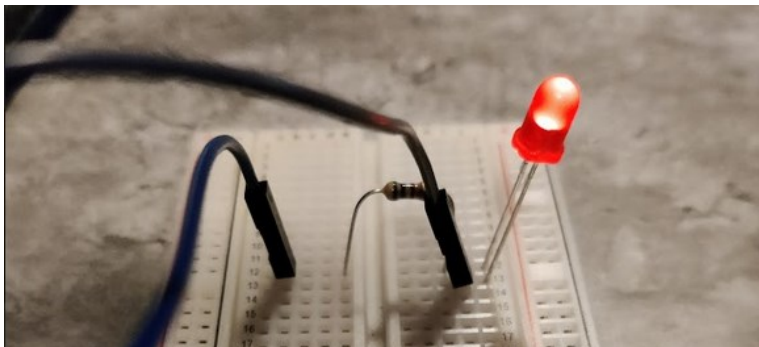


Figure 16. LED on.

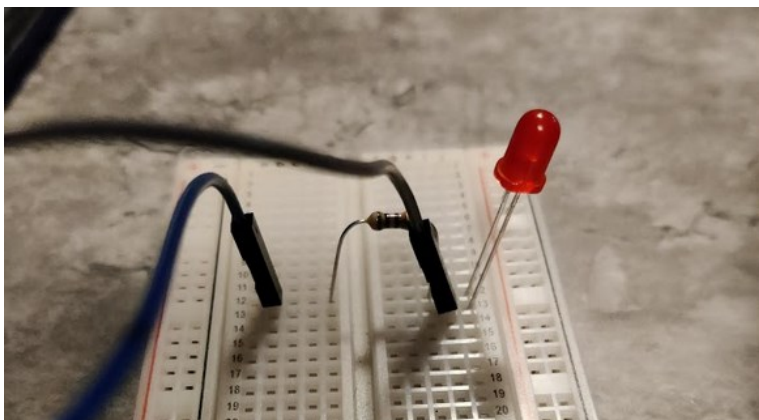


Figure 17. LED off.



## 6 Improvements and future work

Being a proof of concept, the implementation done in this thesis can be improved in multiple ways. The robot itself can be developed to as advanced as the developer sees fit. MATLABs advanced features can be used for image recognition, artificial intelligence or machine learning to further advance the robot.

Home Assistant makes it possible for the robot to interface with smart home devices. Depending on the purpose of the robot, it can react to temperature, lighting or motion with sensors attached to Home Assistant.

ROS makes it possible for multiple robots to communicate with each other, if attached to the same ROS network. This would make for example cleaning robots able to communicate what areas have already been cleaned, or if one robot senses an area it cannot reach, request a different robot to take over with proper equipment.

There is also the possibility of expanding the implementation to utilize ROS 2 and micro-ROS.

## 7 Conclusion

The main goal of this thesis was to research whether it is possible to integrate robotics with a smart home environment, more specifically ROS and Home Assistant. An ESP32 microcontroller was used to act as a proof of concept robot, even though it does not have any robot-like functionalities. The research was carried out through an implementation.

The implementation was a success and created groundwork for future improvement. Communication with ROS network is possible with Home Assistant via MQTT messages and roserial package makes it possible for the ESP32 microcontroller to communicate with ROS network via Wi-Fi.

During the implementation, it became clear that in the future, it would be preferable to use ROS 2 instead of ROS 1, as micro-ROS is the most important advantage of ROS 2 considering this implementation. ROS 2 also does not need a separate ROS master, which makes it more resistant to errors. In ROS 1, if the ROS master is not reachable by other nodes, the whole system is not able to operate.

## References

- [1] iRobot, "iRobot - Roomba," 2022. [Online]. Available: <https://witt.zone/irobot/roomba-fi>. [Accessed 16 April 2022].
- [2] Open Robotics, "ROS Home," 2021. [Online]. Available: [www.ros.org](http://www.ros.org). [Accessed 16 April 2022].
- [3] Open Robotics, "ROS Wiki: Master," 2018. [Online]. Available: [wiki.ros.org/Master](http://wiki.ros.org/Master). [Accessed 16 April 2022].
- [4] Open Robotics, "ROS Wiki: Introduction," 2018. [Online]. Available: <http://wiki.ros.org/ROS/Introduction>. [Accessed 16 April 2022].
- [5] D. Thomas, "Changes between ROS 1 and ROS 2," 2015. [Online]. Available: <http://design.ros2.org/articles/changes.html>. [Accessed 16 April 2022].
- [6] "LCM: Lightweight Communications and Marshalling," [Online]. Available: <https://lcm-proj.github.io/>. [Accessed 16 April 2022].
- [7] T. Z. Authors, "ZeroMQ Home," 2022. [Online]. Available: <https://zeromq.org/>. [Accessed 16 April 2022].
- [8] MQTT.org, "MQTT - The standard of IoT Messaging," 2022. [Online]. Available: <https://mqtt.org>. [Accessed 16 April 2022].
- [9] Home Assistant Inc., "Integrations - Home Assistant," 2022. [Online]. Available: <https://www.home-assistant.io/integrations/#all>. [Accessed 16 April 2022].
- [10] Connectivity Standards Alliance, "CSA - Connectivity Standards Alliance," 2022. [Online]. Available: <https://csa-iot.org/>. [Accessed 24 April 2022].

- [11] Connectivity Standards Alliance, "Zigbee | Complete IOT Solution," 2022. [Online]. Available: <https://csa-iot.org/all-solutions/zigbee/>. [Accessed 16 April 2022].
- [12] O. F. a. N.-R. contributors, "Node-RED," 2022. [Online]. Available: <https://nodered.org>. [Accessed 16 April 2022].
- [13] The MathWorks Inc., "MATLAB - MathWorks," 2022. [Online]. Available: [https://se.mathworks.com/products/matlab.html?s\\_tid=hp\\_ff\\_p\\_matlab](https://se.mathworks.com/products/matlab.html?s_tid=hp_ff_p_matlab). [Accessed 16 April 2022].
- [14] Espressif Systems, "ESP32 Wi-Fi & Bluetooth MCU | Espressif Systems," 2022. [Online]. Available: <https://www.espressif.com/en/products/socs/esp32>. [Accessed 24 April 2022].
- [15] University of Turku, "TIERS | University of Turku | Advanced courses in robotics and autonomous systems," 2022. [Online]. Available: <https://tiers.utu.fi/courses/master>. [Accessed 24 April 2022].
- [16] M. Luqman, "noetic/Installation/Ubuntu - ROS Wiki," 30 September 2021. [Online]. Available: <http://wiki.ros.org/noetic/Installation/Ubuntu>. [Accessed 24 April 2022].
- [17] Home Assistant Inc., "Linux - Home Assistant," 2022. [Online]. Available: <https://www.home-assistant.io/installation/linux>. [Accessed 24 April 2022].
- [18] K. Scott, "Installing and Configuring ROS Environment," 15 June 2020. [Online]. Available: <http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment>. [Accessed 15 April 2022].
- [19] M. P. Paul Bouchier, "roserial - ROS wiki," 2022. [Online]. Available: <http://wiki.ros.org/roserial>. [Accessed 17 April 2022].

- [20] F. Finocchiaro, "micro-ROS porting to ESP32," 2020. [Online]. Available: <https://micro.ros.org/blog/2020/08/27/esp32/>. [Accessed 17 April 2022].
- [21] R. Zickler, "roserial\_arduino/Tutorials/Arduino - ROS Wiki," 18 March 2020. [Online]. Available: [http://wiki.ros.org/roserial\\_arduino/Tutorials/Arduino%20IDE%20Setup](http://wiki.ros.org/roserial_arduino/Tutorials/Arduino%20IDE%20Setup). [Accessed 17 April 2022].