**Microcomputers in Technical Applications *(part 3 – Internet of Things)* – v2.1**

by Chris Hendriks ([hendrikschris@yahoo.com](mailto:hendrikschris@yahoo.com))

1. Pre-install on your laptop: Advanced IP scanner (it is not required but convenient when you are in doubt about an IP address).
2. For the exercises of chapter 8 and 9 it is also convenient if you have installed on your laptop python3, geany and the paho MQTT client. The latest stable version of the paho MQTT client is available in the Python Package Index (PyPi) and can be installed using **pip install paho-mqtt** (you will need pip on your laptop). With these packages you can do the plotting of graphs on one laptop and use the other one as a remote desktop for the RPi. However, the lab can also be done without this facility.
3. For many experiments it works better if both students have a laptop. In that way one can connect to one device, webpage or app and another to a different one. But, also with a single laptop the lab can be done conveniently. Note: only one user can connect to the RPi shell.

The small computer that we use (Raspberry Pi - RPi) operates in the Access Point (AP) mode. That means that it functions as a (basic) router combined with its function as regular computer. So, the device consists of a (basic) router and a computer directly connected to each other. For convenience we will use the name ´RPi´ for the computer function and ´router´ for the router function (we will also use the name ´RPi´ for the device itself, but in those cases there won´t be any reason for confusion). With your laptop you can get access to the router by selecting the proper wifi network. The name of the wifi network is included in the micro SD card (and written on the sticker on the card or RPi). We will use names consisting of two capital letters

Your laptop receives an IP address from the router. However, you don´t have to know what the address is, since the IP address of the RPi is always 10.3.141.1 . You don´t need it, but you could access the router with the address 10.3.141.0 .

The IP range of the AP network is 10.3.141.x; from 10.3.141.50 onwards is used for dynamic addresses; the lease time is 12 hours.

The complete identity of your RPi is contained in the microSD card (SD stands for Secure Digital; the microSD card is a widely used medium for storing digital data). Also, all programs you write are being stored on the microSD card. So, when you put your card in a different RPi, you have an identical environment.

Therefore, **remember the name of your card and always connect your laptop to your own network**.

When logging in:

* Take care that your laptop is in the proper network (the network of your SD card
* Leave the RPi a minute to boot
* For logging on to your RPi: the name is ´pi´ and the password is ´pi´.

When logging out:

* Shutdown the RPi with the command: sudo shutdown now
* When the session has ended (after 10 to 20 seconds) you can switch of power.

1. **Introduction. Internet of Things - the information revolutions in the technical environment**

An enormous information revolution is taking place in the social environment. Less visible but equally important is the information revolution taking place in the technical environment. The essence of this development is that many sensors and actuators in technical systems, like industrial plants or other control systems, are connected to each other and to larger AI[[1]](#footnote-2) systems. All systems exchange data and are operating autonomously or under human control. Three development are relevant in this process:

* the increasing intelligence of sensors and actuators; most of these devices use in some way a local computer or intelligent controller
* the availability of wireless communication virtually everywhere and at low prices
* the use of some form of AI either locally or in a central computer; local AI is often based on some form of image processing, turning a camera into a sensor for detecting specific object or changes in process parameters. More central AI systems support the human controller or are integrated in the control system.

The development described above is usually referred to as the *Internet of Things[[2]](#footnote-3) (IoT)*.

Common to present day IoT systems is the broker architecture.

The name broker comes from the trade business. In that context a broker is an independent party, whose prime responsibility is to bring sellers and buyers together. Thus, a broker is the third-person facilitator between a buyer and a seller. An example would be a real estate broker who facilitates the sale of a property.

In technical systems the broker is the central application that regulates all communication between subsystems: every subsystem communicates with the broker and none of them directly with another subsystem. In that way subsystems become independent from each other and hence they can be purchased from different manufacturers and replaced relatively easy. We will take the broker architecture as starting point for the exercises in this lab.

1. **Imaginary environment for the experiment**

Since the essence of IoT is the interconnection of intelligent devices and their interaction with the physical environment, we will use an imaginary environment in which the lab takes place. We will assume a plant nursery with automatic temperature control, humidity measurement and disease detection.

The temperature control process is simulated in software.

Measurements are being send to the control room and settings are being received from the same control room. The control room consists of a dashboard for the human interface, a computer for running the AI software and a computer for controlling the temperature.

Hence, we have the following more or less independent subsystems:

1. The humidity measurement.
2. The heating subsystem consisting of temperature measurement and heating activation (including in this case the temperature simulation). In practical situations the temperature measurement and heating activation are separate subsystems.
3. The camera that generates images (the camera is considered a sensor that generates images as sensor data).
4. The AI subsystem.
5. The temperature control subsystem based on a PID controller. The human controller can change the temperature setpoint.
6. The human interface based on a process dashboard.

Central in the configuration is the broker that takes care of receiving and distributing data (in terms of the broker: publishing data and subscribing to data).

For convenience of spreading the lab over a number of sessions, every time a part of the system with a specific focus is built and tested.

1. **Physical and logical architecture**

We have seen that there are 6 subsystems. The subsystems have physical relations. Some subsystems are implemented in the same computer. They also have logical relations – a particular subsystem sends data to or receives data from another subsystem. The physical and logical architecture are quite independent from each other. It is essential to understand the difference between the two and to know what architecture a certain structure refers to. First, we will have a look at the logical architecture (see the diagram).

As you can see from the diagram all interactions between subsystems pass through the broker. There is no direct interaction between subsystem. So, if the protocol is fixed and the content of the dataflow is determined, a subsystem can easily be taken off and replaced by another one. Also, the dependency from manufacturers becomes minimal (as long as they stick to the protocol standard at transport level and at data level).

We can have different subsystems in a single computer.

Initially we will implement in the lab

* in an ESP 8266 microcontroller the simulated process plus sensors
* in an RPi the Dashboard, the Controller, the AI system and the Broker.

In the final set-up all data transport is through a wifi network. In the development phase transport between RPi and ESP can be through a USB cable.

For the wifi network a routing function is required. This routing function is integrated in the RPi. Hence, no separate router is required.

You could say that the broker is for the logical system what the wifi router is for the physical system.

For the physical architecture there are various possibilities. We can alter the physical architecture without affecting the logical architecture. Initially the Broker, the AI software, the process control software and the Dashboard run on the RPi. However, for instance the software for process control can move to an ESP controller without affecting the logical architecture.

For the lab we will us the following software:

* 1. For the broker we use Mosquitto that runs on the RPi.
  2. The software for controlling the camera is a (micro)Python program that runs on an ESP or the RPi.
  3. The software for the various sensors runs on one or two ESP controllers; there are two ways to program the controller: with ESPEasy (ESPEasy offers a programming environment based on pre-programmed plug-ins) and with microPython (microPython is a programming language very similar to Python with additional features for sensor and actuator interfacing).
  4. As a dashboard we will use Node-RED (running on the RPi).
  5. The process control application will be programmed in Python (running on the RPi).
  6. The AI software will run on the RPI.

Some of the applications have a user interface through the browser – that can be a browser from any computer. It can be the browser on the RPi or a browser on any of the laptops (as long as the laptop is connected to the RPi subnetwork). The broker and python applications have a user interface through the RPi shell.

In the lab we will go through exercises that focus on specific sections and aspects.





1. **Broker protocols and implementation**

The broker isolates the various subsystems. When the broker protocol has been chosen and the data that is to be exchanged is specified, the subsystems can be designed and built independently.

We will use the MQTT protocol. MQTT[[3]](#footnote-4) is a machine-to-machine (M2M) connectivity protocol[[4]](#footnote-5). It was designed as an extremely lightweight publish/subscribe messaging transport protocol. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium. For example, it has been used in sensors communicating to a broker via satellite link, over occasional dial-up connections with healthcare providers, and in a range of home automation and small device scenarios. It is also ideal for mobile applications because of its small size, low power usage, minimized data packets, and efficient distribution of information to one or many receivers.

Characteristic for the MQTT broker architecture is that all sensors and other devices communicate through the broker by publishing (sending data) and subscribing (receiving data) operations. Both publish and subscribe operation refer to ‘topics’ as the subject of the data they are sending or interested in. The broker takes care of the forwarding of messages. So, there is no direct interaction between the sender (publisher) of data and the receiver (subscriber). For a specific topic there is usually one publisher and there can be one or more subscribers.

There are numerous good tutorials on the MQTT broker that can be found on the internet.

We will use the Mosquitto broker. The broker is downloaded on the Raspberry Pi (Appendix 1).

Start the Mosquitto broker on the RPi with **sudo systemctl start mosquitto**. The status can be checked with: sudo systemctl status mosquitto. Later on, we are going to use the broker in a quite complex setup of different devices and application. So, it is good to test the broker at this stage (we will do that for all components; after using it a few times we can omit the test).

In order to test the broker, we will subscribe to and publish an MQTT topic. We will need two terminal shells for this test – we run the terminal on the RPi. Use one of the terminals to subscribe to a test message topic (you can give it any name e.g. test/message) by using the shell command: **mosquitto\_sub -h localhost -t test/message** (´h´ is short for ´host´ - if the host is the same computer the address is ´localhost´, otherwise it is the IP address; ´t´ is short for ´topic´).

This will send a subscription message to the MQTT broker which is in the example code running on the same system (as specified by the -h localhost option). After the parameter ´-t´ follows the topic.

Use the other terminal shell to send a message to the same topic:

**mosquitto\_pub -h localhost -t test/message -m Hello, world** (again ‘localhost’ should be replaced by the IP address if the broker is running at another machine; ´m´ is short for ´message´). Try some different topics and messages (the message follows after ´-m´; the message can be a text or a numeric value).

You can download a mqtt client on you telephone or laptop (e.g. mymqtt from the Appstore for the telephone and mosquito client for the laptop. But there are many alternatives.

Try out the connection with the broker by publish and subscribe.

Later on, we will see how the subscribe and publish function can be embedded in program code.

1. **Dashboard (Node-RED)**

For the Internet of Things, or any control system, the **dashboard** or **IoT dashboard**is the key HMI (Human-Machine Interface) component that organizes and presents digital information from our physical world into a simply understood display on a computer or mobile device. With the help of IoT Dashboards, users and operators can (remotely) monitor and control specific assets and processes, and depending on safety requirements, access and control an environment from anywhere in the world.

We will use Node-RED as a dashboard application. Node-RED can be started from the programming icon on the RPi main screen.

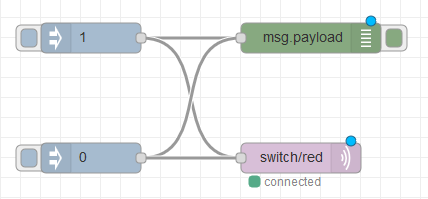
After it has been started you can get access to the app through the browser. This can be from your laptop browser – or from any computer connected to the local (RPi) router – through the wifi network using the IP address of the RPi or from the RPI itself using ‘localhost’. In the browser enter localhost (or the IP address: 10.3.141.1) and port number 1880[[5]](#footnote-6) (use a colon between address and port number: 10.3.141.1:1880).

You can get familiar with the programming of the dashboard by going through the following website pages: <https://nodered.org/docs/tutorials/first-flow>

<https://www.snap4city.org/download/video/Node-RED_dashboard_user_manual.pdf>

for charts:

<https://www.snap4city.org/download/video/Node-RED_dashboard_user_manual.pdf>

Node-RED provides a browser-based flow editor that makes it easy to wire together flows using the wide range of nodes in the palette. Flows can be then deployed to the runtime in a single-click.

Build the configuration shown (the two inject nodes have a number as payload and you can take any name as topic).

Deploy the flow.

Test it on the RPi by opening a subscribe process in a terminal shell at the RPi (this is the same as in exercise 4. since all messages pass through the broker; the topic name should correspond to publish node of the dashboard).

The debug node shows its input in the debug area.

Note: be aware that every time after a modification you have to activate the change by clicking ´deploy´.

1. **ESP microcontroller with ESP-Easy**

In order to control sensors and actuators we use microcontrollers. In this case we use the ESP8266.

We can´t develop software on the ESP itself - the on-board memory is limited: 4 MB (4 MegaByte or 32 Megabit) of flash memory[[6]](#footnote-7) and some RAM. The flash memory (a type of EEPROM) is certainly not enough to load various apps onto the board. That has never been the intention. The idea of a microcontroller is to have a universal and flexible control device that is programmed once – with possibly an odd update for a newer version or to remove a bug.

So, we need a general-purpose computer – like a laptop or the RPi – to develop the software, which is then downloaded (flashed) onto the ESP.

Since the code is stored in a kind of EEPROM, it is not lost when the controller is powered off.

The ESP8266 can be flashed through the USB connection. For the lab all the required software is stored on the microSD card of the RPi so we will flash the ESP from the RPi through a USB connection. The alternative is to flash it from your laptop but then the required software has to be downloaded to the laptop first.

We will discuss and apply two techniques to program the microcontroller:

* Using pre-programmed plug-ins; the software that we will use is ESP-Easy.
* Programming the controller with a high-level language; we will use microPython, a language quite similar to Python with some extra features for controlling sensors and actuators.

In this chapter we will use the first method.

* 1. **Flashing the ESP**

Connect the ESP to the RPi using a USB cable. We now have access to the ESP.

For flashing the ESP we use a program esptool (a Python program). Whenever you flash the ESP you have to press the flash button on the righthand side of the USB connector as shown in appendix 1 (the one on the other side is for reboot).

First, we erase the memory of the ESP with: **python esptool.py erase\_flash[[7]](#footnote-8)**.

Next, we flash the ESP\_Easy software: **python esptool.py write\_flash 0** **ESP\_Easy.bin** (ESP\_Easy.bin is the code to be flashed; 0 refers to the offset in memory where the code is to be stored)

Now that the software is loaded onto the ESP, the next step is to configure the software according to the sensor connected to it. Configuration is done through the browser. You can use any browser (using any laptop or even the RPi; as long as the device is in the same subnetwork). However, what is the IP address of the ESP? And how do we put it in the subnetwork of the RPi so that we can communicate with it? We will find out in the next paragraphs.

One method to give the ESP the proper IP address is the following: the ESP advertises itself as an Access Point (AP) with a fixed name, a fixed password and a fixed IP address; with a laptop you connect to the AP and configure it as a Station in the network you want to work in (usually the regular router network; in our case the RPi router network). Normally, this method works without problems. However, in the lab situation all ESP´s would advertise their AP´s with the same name, which might result in a student connecting to the ESP of his neighbor. Undesirable, so we will use another method.

For setting the proper IP parameters we will use the USB connection. A small program can be used to set up a serial connection over the USB cable (in operation the USB connection will have no other function than supplying the 5 V power). Activate the program minicom in the raspberry by entering the shell command: minicom (alternatively you can start it from ´System tools´ under the raspberry icon).

If you get log info type: debug 1

You can see what IP subnetwork the ESP is connected to by entering the command: settings.

In order to set the minicom to the proper IP network, enter the following commands (the commands are not case sensitive so you don´t have to type capitals):  
WifiSSID <SSID of your network> # SSID stands for ´service set identifier´ and is simply the name of the IP subnet;

# in your case it is the name on the SD card

WifiKey <password> # all RPi router networks have the password ´raspberry´

WifiConnect # with connect to the router (in RPi) the ESP receives an IP address

Check the IP address with ´settings´. It should show an IP address. Use this IP address in the next paragraph.

Connect through the browser to the ESP (use the IP address of the previous paragraph).

Go to the tab ´Config´ and fill in a static IP address under ´Wifi IP setting´: IP 10.3.141.40, Gateway 10.3.141.1, Subnetmask 255.255.255.0 and DNS 10.3.141.1; tick ´submit´ and reboot under tab ´Tools´. The network data are stored now in the EEPROM of the ESP and won´t be lost at power off (of course it will be lost again with ´erase´)[[8]](#footnote-9).

After reboot you have lost the connection to the initial IP address, since the ESP has got a new address. Re-connect through the browser to the ESP using the static IP address you entered (in this case 10.3.141.40)

We won´t need minicom anymore. To exit minicom press cntrl A and then Z and then X (select ´yes´).

* 1. **Reading a DHT sensor**

*Experiment 1: Connecting the DHT*

Connect the DHT sensor to 3.3 V, Ground and a data pin on the ESP 8266. Appendix 1 shows what data pins you can use safely – we will use GPIO-13 (D7) for data. Connect the ESP to any 5V power supply through the usb cable.

Open in the browser the IP address of the ESP.

Program the ESP as follows:

* Under ‘devices’ add a device and select DHT11/12 (do not take the device that communicates with I2C)
* On the configuration page complete: ‘name’ (eg DHT), tick ‘enable’, enter the sensor data (pin is in our setup GPIO-13 (D7) and model is DHT11) and enter for instance 2 as sampling interval. Select ‘submit’ and close the window.
* Check the display and click ´close´ (if necessary, select ‘devices’). By breathing over DHT you can see the temperature and humidity change.

*Experiment 2: Publishing the DHT data*

We will publish the data through a broker on the RPi. The broker itself doesn´t need any configuration after starting. The broker just resends the data it receives. Therefor we open a shell window and subscribe to the broker. For testing purposes, you can open another window and subscribe to the published message

When subscribing to the ESP message we just have to use the topic used by the ESP. E.g. :

mosquitto\_sub -h localhost -t "DHT/Humidity" (or ‘Temperature’ or ‘#’ if want all topics to be displayed; be aware that the topic name has to be exact with capitals where needed).

Back to ESP-Easy:

Configure the interface to the broker for publishing by clicking ‘controllers´. Add a controller by selecting the Home Assistant (openHAB) MQTT.

Enter the IP address of the broker (leave the port number at 1883). Set the publish format to %tskname%/%valname%, which means that you publish topics with only the task name and the value name (alternatives are possible as long as you take the full topic name into account when subscribing to the broker). Tick ‘enabled’ and submit.

Now the DHT has to be told to send the data to the controller – edit the DHT and tick ¨Send to Controller´. In the screen ´devices´ you will see a ´1´ under CTR (controllers) indicating that the output of the device is being sent to controller 1.

Note that with some modifications it is necessary to reboot – select ‘tools’ and reboot.

*Experiment 3:*

Display the Temperature and Humidity in Node-RED with a debug node.

* 1. **Controlling traffic lights**

From the led traffic light connect ground as well as red (take GPIO 14 = D5 as output port).

*Experiment 1: settings at system boot*

In this experiment we are going to work with ‘rules’.

From ‘tools’ select ‘advanced’ and tick ‘rules’ in order to enable the function – don’t forget to submit[[9]](#footnote-10).

Go to the tab ´Rules´.

Switch on the red led at system boot by entering the code in the box (don´t forget to save/submit). The code becomes active after reboot.

on System#Boot do

gpio,14,1

endon

Adjust the code in order to switch off the led at system boot.

Connect the yellow and green led (use for yellow gpio12=D6 and for green gpio13=D7).

Repeat the above experiment for yellow and green (the green light will be less clear).

*Experiment 2: use of timer*

Extend the code as shown in the box.

on System#Boot do

gpio,14,1

timerSet,1,7

endon

on rules#Timer=1 do

gpio,14,0

endon

Explanation of the code:

timerSet,1,7 : timer number 1 is set to 7 seconds

on rules#Timer=1 do : when timer 1 (Timer=1 means timer number 1) reaches zero (the timers always count down to zero after being set) execute the rules between ‘on’ and ‘endon’.

Note: the rules are plain text you can ‘download to file’ or select (a section of) the text and copy it.

*Experiment 3: control output with messages through the broker*

Add a device for subscribing to a topic at the broker (Generic - MQTT Import). You can give the device any name – let´s give it the name ´import´. Enter ´traffic´ as the topic, red as ´value 1´ and tick enable. The topic name determines the name that is sent to the broker to subscribe. The name of the device that connects to the broker (in this case ´import´) plus the name entered as value (in our case ´red´), are being used in the setting of the rules. Why is it set up like that? Could it not be set up simpler? Hint: there can be more that one MQTT Import device and a message can consist of more than one values.

The broker:

If it has not been started yet: start the broker on the RPi with the shell command sudo systemctl start mosquito. If you wish you can check the status with sudo systemctl status mosquito.

on import#red do

gpio,14,1

endon

Send a message to the broker with mosquitto\_pub -h localhost -t "traffic” -m 1 (‘h’ stands for ‘host’ – in this case localhost because the broker runs on the same device; ‘m’ stands for ‘message’ in this case we take ‘1’ for ‘on’). If you want to make sure that the message has been sent, open another shell with mosquitto\_sub -h localhost -t "traffic" (the topic must be the same as the published topic). You can also see from ´Devices´ whether a message has been received. The code in the box switches the led on when a message is received. Test the code in the box.

**Note: the rules-program starts after a reboot; wait till the boot process has been completed before publishing the message to switch off the led (the boot-process takes some 10 seconds – the screen goes to Main).**

Modify the rules in such a way that the red led turns off at boot and turns on when a message is received under topic ´traffic´ by device ´import´ and value ´red´. After 5 seconds the led turns off again.

*Experiment 4: conditional actions depending on content of message*

on import#red do

if [import#red]=1

gpio,14,1

else

gpio,14,0

endif

endon

Modify the rules in such a way that the red led turns on when the message ‘1’ is received under topic ´traffic´ and turns off when a ‘0’ is received. This means that you have to check whether the value ‘1’ or ‘0’ was sent. This can be done by using the ‘if’ construct – see the code in the box.

Note: the conditions specified under ´rules´ remain active (so, it is different from sequential program code). How could the last experiment be implemented with two ´on-rules´ whereby the rule condition is: import#red=1 (or 0).

Extend the code in such a way that when the red led turns off the yellow led turns on for 4 seconds. After that the green led turns on.

Note: if we want to make an action to be dependent on the present state of the led, we have to make use of a virtual switch that is behaving as if it is connected to the led output gpio port. In this experiment we will leave that aspect out. See also <https://www.letscontrolit.com/wiki/index.php/Tutorial_Rules>

*Experiment 5:*

Implement the functionality of experiment 4 using an input from Node-RED.

1. **Microcontroller with microPython**

As was pointed out in the previous chapter we can´t develop software on the ESP itself. Apart from using pre-programmed plug-ins like is done in ESP-Easy, we can develop the firmware in a high-level language on a different computer and flash it onto the microcontroller.

In this chapter we will use microPython as the high-level language. Since microPython – like Python – is using an interpreter during execution we will first load the interpreter on the microcontroller and subsequently develop the software on a different system (in our case the RPi) and load the software onto the ESP as well. In this way we can use an advanced IDE for developing the software. All loading of software to the ESP goes through the USB connection.

We will use Thonny as the IDE for developing the application software. Thonny is running on the RPi.

* 1. **Install microPython** on the RPI

Connect the ESP through the usb cable to the RPi (preferably before the RPi is powered on).

Open a shell in the RPi. We will use esptool for flashing (remember that whenever flashing firmware onto the ESP you have to press the flash button – see appendix 1).

It is good practice to flash the ESP with all blanks before actually flashing any firmware. In order to erase all prior data on the ESP use the command: **python esptool.py erase\_flash[[10]](#footnote-11)**.

With the erase complete, you can **now flash the board with the new firmware by running the command:**

**python esptool.py write\_flash 0 esp8266.bin (0 is the offset in esp memory; esp8266.bin is the file with the microPython interpreter to be loaded onto the esp)**[[11]](#footnote-12)**.**

* 1. Blinking a led through the ESP

We will start getting familiar with the combination of the RPi – where the software is developed – and the esp8266 – where the software runs.

From the Raspberry icon select ´Programming´ and next ´Thonny Python IDE´. In the IDE select ´switch to regular mode´ (top right-hand corner) and restart Thonny (if ´switch to regular mode´ is not shown it is in regular mode already); select tools>options>interpreter. Select MicroPython (ESP8266).

You can open files from the computer you are working on (the RPi) or from the microPython device (the esp). You can try by opening the small boot program on the esp (don´t modify it!).

**Note: whenever the connection between the RPi and the ESP is lost you can reconnect by ´Stop/Restart´ in the Thonny screen.**

The on-board led is connected (on the board) to pin 2. By making pin 2 equal to 0 we can switch the led on (note that whether a ´0´ or a ´1´ makes the led go on depends on whether the other side of the led is connected to Vcc or Ground; what is it in this case?). By making the pin equal to 1 we can switch it off. The module ´Pin´ from the library ´machine´ allows us to access the pins.

from machine import Pin

led = Pin(2, Pin.OUT)

led.value(0)

See the program in the box:

* We import the module Pin
* We create an instance for accessing pin 2 and we specify this pin as output (we call the instance ´led´)
* With the function ´value´ we can switch the led on.

Write the code and run it (use the ´run´-icon or click ´Run´ + ´Run current script´).

Check how and where program files can be stored and how they can be loaded again.

from machine import Pin

from time import sleep

led = Pin(2, Pin.OUT)

for i in range (5):

led.value(0)

sleep(1)

led.value(1)

sleep(2)

The code in the box makes the led at pin 2 go on and off. Try it.

Alter the on-off ratio of the blinking led.

Connect one of the colors of the traffic light to the ESP . Use pin D2 (GPIO4) as the signal pin. Write the program to make the led go on and off.

Connect all three leds and make them go on and off in turn.

* 1. **Connect the ESP to the proper subnetwork**

Because we want the program in the ESP to connect to the broker, we have to configure it into the proper IP subnetwork.

import network  
sta\_if=network.WLAN(network.STA\_IF)  
ap\_if=network.WLAN(network.AP\_IF)  
  
sta\_if.active(True)  
sta\_if.connect('<your\_ssid>','raspberry')  
sta\_if.ifconfig(('10.3.141.41','255.255.255.0','10.3.141.1','10.3.141.1'))  
  
ap\_if.active(False)

For this purpose, we write a small program (the code in the box). Use the name ´net.py´.

Explanation:

* We import the module ´network´ that offers us the possibilities to manipulate the network connection of the ESP.
* The ESP can operate in two modes (at boot it operates in the Access Point mode)
* We create for both methods an object instance
* Make the station active and connect it **to your subnetwork** using name and password (so replace ´your\_ssid´ by the correct subnetwork name)
* Give the ESP a static IP address with station number 41 (in addition to the IP address, we also have to specify the subnet mask, default gateway address and domain name server (DNS) address)
* Deactivate the AP.

Run the program and check in the shell the IP address (use sta\_if.ifconfig() ).

Reboot and check the IP address again (without first running the program. Explain the error message that you get

Now rename the program to main.py, reboot and check the IP address again. What do you notice? The explanation is that the program main.py is always executed automatically after boot.

* 1. **Run a simulation on the ESP**

We are going to write a program that simulates the temperature in the plant nursery. The program will be called ‘sim.py’ and it will be stored and run on the ESP.

We will start with a simple program that shows the connection of the simulation with the broker. The simulation receives the heating setpoint from the broker.

from umqtt.simple import MQTTClient  
global H  
server="10.3.141.1"   # address of MQTT broker  
H=0  
  
def on\_message(topic, msg):  
    global H  
    H=float(msg)

c = MQTTClient("any\_id", server)  
c.set\_callback(on\_message)  
c.connect()  
c.subscribe(´H')  
  
while (H<100):  
    c.wait\_msg()

K=2\*H     
    c.publish("K", str(K))

The code in the box shows the interaction between process and broker:

* The module umqtt is included in micropython. We import the client method;
* A separate process that waits for a message coming in from the broker is going to be started. The variable H is as well being used in the main program as in this process so it is made global
* ´server´ is a constant (the IP address of the server where the broker is running)
* H is given an initial value (since it is used in the while loop later on)
* c is an instance of MQTTClient (it has 2 parameters: an id of the sender and the server address; we don´t use the id of the sender)
* when c receives a message – the function c.wait\_msg() waits until a message comes in – the callback function (on\_message) is activated;
* in the callback function the topic of the received message can be checked but since we have only one message to deal with this is not needed; the message – which is in text code – is converted into a float
* the while loop is left when a value for H larger than 100 has been received
* the program terminates with disconnecting from the broker.

Note: the broker receives and sends just sequences of bytes. The protocol provides the possibility to indicate in the MQTT packet the payload type. However, umqtt doesn´t use this facility and handles all data as a sequence of bytes. By converting the float into a string at publishing and converting the string into a float at reception, this ´problem´ is overcome.

Start the broker and open two shells in the RPi: one for publishing and one for subscribing to the appropriate messages. Test the program.

Modify the program in such a way that it is not terminated by receiving a value for H larger than 100 (see the code in the box). In that case the proper way to terminate the program is by ´pressing´ the stop button:

while (True):

c.wait\_msg()

c.publish("K", str(K))

Extend the program in such a way that the temperature in the nursery is calculated every time a new value for the setpoint H is received.

Parameters for the simulation are:

H – the setpoint for the heating of the nursery

T\_i – the temperature in the nursery

T\_o – the outside temperature.

The following constants are relevant:

C\_i – the rate at which the temperature in the nursery rises because of the heating

C\_o – the rate at which the temperature drops because of heat loss to the outside.

We start with the following values:

T\_i=20.0

T\_o=20.0

C\_i=0.1

C\_o=0.02

The following formula gives the relation between the various parameters:

T\_i=T\_i+C\_i\*H-C\_o\*(T\_i-T\_o)

So, every cycle we update T\_i according to this formula

We choose 1 sec. to correspond to 15 min. in real life (96 sec. corresponds to 1 day). This is going to play a role when T\_o is going to vary in a day-night sequence. We choose delta\_t (the time between two calculations of T\_i to be 2 seconds. This delta\_t is implicitly included in the constants C\_i and C\_o.

All variables have units that suit the given formula.

while (True):

c.check\_msg()

T\_i= --------

Sleep(2)

c.publish("T\_i", str(T\_i))

We assume the outside temperature to be constant at this stage.

Note 1: The formula for H depends on delta\_t, the interval between two calculation cycles. We will assume this interval to be 2 seconds. The timing will be dictated by the while loop where we have included a sleep period of 2 seconds.

In order to generate a value for T\_i every 2 seconds, we use here the simple method of including a sleep-time. This method is of course not precise since the total time in between two values of T\_i, is 2 seconds plus the time it takes to execute the code in the loop.

How could you make the timing more accurate using only functions from the time module? Hint: replace the sleep by a while loop.

Note 2: the multi-threading mechanism is not implemented in microPython. We therefor have to include MQTTClient.check\_msg() which checks if the server has any pending messages. If it has, the message is processed in the same way as wait\_msg(), if not, it returns immediately.

Write the program for the simulation of the temperature in the nursery.

We can check whether the simulation works alright by publishing a heating setpoint and subscribing to the temperature T\_i. For this we open two windows in the RPi shell:

mosquitto\_pub -h localhost -t ´H´ -m 5 which sends the value 5 as message;

mosquitto\_sub -h localhost -t ´T\_i ´ for subscribing to ‘T\_i’.

Run the simulation, the broker and the shell programs together. Use as a setpoint for H the value 2.

Since there is no feedback, the temperature in the nursery continues rising until the heat loss to the outside surroundings per second equals the heat added by the heater per second.

Add the dashboard in such a way that you can publish the values ´0´ and ´2´. Try to keep the internal temperature at 25 degrees. You will see that it is not easy. What you are doing in fact is that you function as the feedback loop. This can be automated – we will do that in chapter 9. First we will plot T\_i in a graph whereby the graph application gets its input from the broker in order to maintain maximum flexibility.

Note: at reset the programs boot.py and main.py run automatically. Other programs you have to start manually. If you want your simulation to start automatically you can include it in main (include a sleep period of 2 seconds between setting up the network connection and actually connecting to the broker): either include the code directly or include ´import sim´ to import the module ´sim.py´. In practical situations it is essential that software in microcontrollers starts automatically at power up. In this exercise however, it can be done manually.

1. **Real-time graphs with input from the broker**

import paho.mqtt.client as mqtt (1)

import matplotlib.pyplot as plt

import matplotlib.animation as animation

broker\_address="localhost"

global x,y1

#global y2,setpoint

step=0.5

x=[]

#setpoint=0

fig, ax = plt.subplots(figsize=(15, 10)) (2)

for i in range (100): (3)

x.append(0.5\*i)

y1 = [0.0]\*100

#y2 = [0.0]\*100

def on\_message(client, userdata, message): (4)

global x,y1

global y2,setpoint

# if (message.topic== 'setpoint'):

# setpoint=float(message.payload)

if (message.topic== 'sine'):

y1.append(float(message.payload))

# y2.append(setpoint)

x.append(x[-1]+step)

x = x[-100:]

y1 = y1[-100:] (5)

# y2 = y2[-100:]

def animate(i): (6)

global x,y1

global y2,setpoint

ax.clear()

ax.set(ylim=(-1.5, 1.5))

ax.plot(x, y1,label="Sine")

# ax.plot(x, y2,label="Setpoint")

plt.ylabel('amplitude')

plt.xlabel('time')

plt.legend()

client = mqtt.Client() (7)

client.connect(broker\_address)

client.loop\_start()

client.subscribe("sine")

#client.subscribe("setpoint")

client.on\_message=on\_message

ani = animation.FuncAnimation(fig, animate) (8)

plt.show() (9)

client.disconnect()

import paho.mqtt.client as mqtt

from time import sleep

import numpy as np

broker\_address="localhost"

client = mqtt.Client()

client.connect(broker\_address)

t=0.

try:

while (True):

y=np.sin(t)

client.publish("sine",y)

sleep(0.5)

t=t+0.2

except:

client.disconnect()

We will use a program that generates and publishes a sine curve. The program is shown in the box. The program is available in your home directory under the name sine.py (if the program is not available copy the code). The subsequent values for the sine function are being published to the broker as a string.

An explanation of the program:

* The mqtt client is imported; be aware that the MQTT client used here is different from the MQTT client used in the ESP8266 running microPython.
* An instance of the client method is made.
* A connection to the broker is established.
* In an endless loop the subsequent values of the sine function are being calculated and published.

Next, we are going to develop a progrom that plots the sine function real-time. The code is given in the box w- the commented lines are for plotting two lines in the same figure.

Stepwise explanation of the plot program (the commented lines are specific for plotting a second line; ignore them for now):

1. The mqtt client for Python (1) is imported, as well as two modules of matplotlib (pyplot is the general plot module and animation is used for plotting graphs);
2. An object ´fig´ is created in which the graph is plotted. Also, an object ´ax´ is created that draws the actual plot (axes and lines) in the figure;
3. Two lists are being defined: x for the x-axis corresponding with time and filled with incremental values, and y1 filled with zeros and corresponding with the amplitude values of the sine function;
4. The on-message function is being defined; it receives the subsequent values of the sine function and appends them to the y1 list; the test for the topic name is harmless but unnecessary here but we need it later;
5. The x and y1 lists are being kept to a length of 100 elements;
6. The animate function does the actual plotting; it generates subsequent plots after deleting the old one (the deletion makes that the labels and legend have to be added again);
7. The connection to the broker is set up and the subscribe loop is started;
8. The animate loop for plotting is started. This loop, as well as the subscribe loop, runs as a separate process in the background;
9. The resulting figure is being shown. Closing the display terminates the program including the animate background processes.

Write the plot program when plotting the sine function that it subscribes to.

Because later on we want to display the setpoint as well as one of the other parameters, we are going to add a second line to the graph. The second line is called ´Setpoint´ and its values are being supplied by the broker.

Since we generate the time axis by increasing the time with 0.5 seconds every time a value for ´sine´ is being received, the is no time reference for the reception of the ´Setpoint´ signal (the ´Setpoint´ signal is independent in time). Therefor we adjust the value for ´Setpoint´ at reception and adjust the line values (y2) at the next time a new value for y1 is being received.

Test the program that displays the ´Sine´ function as well as the ´Setpoint´ function. Generate the ´Setpoint´ message with a mosquito\_pub command in the RPi shell.

Modify the plot program in such a way that it plots the setpoint H as well as the temperature in the nursery t\_i.

From now on the system gets more complicated because various components of the IoT application have to be started.

Take notice of the following:

* Being the core component, it is best to start with the broker. For instance, when starting Node-RED it might have a flow active that has a node that tries to connect to the broker.

You can test the broker with two RPi shells, one for publishing and one for subscribing.

Alternatively, you can test it with a publish and subscribe in Node-RED.

* Now the sequence is less important. For instance, start Node-RED (if you haven´t done so yet). Open or build a flow that generates a setpoint for H (test in the RPi shell whether with a subscribe the message is being received).
* If the simulation program in the ESP is not starting automatically (it only starts automatically if it is included in ´main´), start the program. Again, you can test the proper functioning with a subscribe in the RPi shell.
* Send a message from Node-RED setting the setpoint ´H´ to a value of 2 and check whether you see the value of T\_i rising (with the present initial values it rises slowly from 20.0 to 30.0; why?).

When you are familiar with the system it works best if you include a subscribe in the Node-Red dashboard and connect that to a debug node. In that way you can easily check any signal that the broker receives by subscribing to that particular signal and viewing it in the debug pane.

Another suggestion: use one laptop for the connection to the RPi and a different one for Node-RED.

You can include grid lines in the plot figure with the function: [plt.grid](https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.grid.html#matplotlib.pyplot.grid)(**True**)

Explain the shape of the graph that you get.

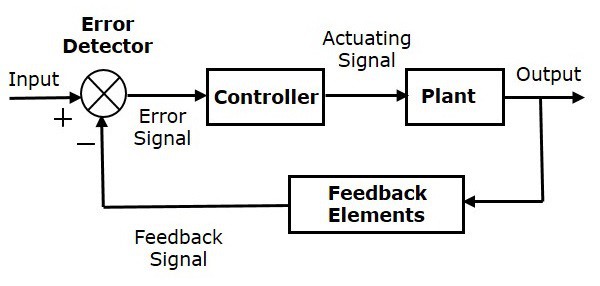
Note:

If you have a spare laptop available you can run the plot on that laptop. Install on your laptop python3, geany and pip. The latest stable version of the paho MQTT client is available in the Python Package Index (PyPi) and can be installed using **pip install paho-mqtt**

Of course, when connecting to the broker from your laptop, you don´t use ´localhost´ as the broker address but you use the actual address of the RPi (that is where the broker is running): 10.3.141.1

Also, your laptop should be in the proper subnetwork. However, don´t initiate a remote session from this extra laptop because the RPi is configured in such a way that it can only accept 1 Remote Desktop session.

1. **The control subsystem**

The control subsystem controls a process by means of a controller (see diagram) that is – in many cases - made in software. The process is usually a physical process in combination with (intelligent) sensors and actuators. The sensors and actuators communicate with the controller through the broker. The advantage of this structure is that the controller can be modified easily and can also make use of additional data coming from - in our case – for instance a weather prediction or AI software that finds patterns of diseases or other specific risks.

For the purpose of this lab, we simulate the process (as we have seen in chapter 7). However, not only in an educational setting a simulated process can be useful. Also, for research purposes or for predicting the behavior of the process under specific circumstances a simulation is essential.

The feedback element is in our case considered to be included in the sensor of the output signal, in other words, it has a value of 1. We also assume the error detector to be included in the controller. So, the controller is fed with the input (S = setpoint of the temperature) and the feedback signal (T\_i = the measured temperature in the plant).

We have the following signals:

S: setpoint for the temperature in the nursery; generated – at this stage manually – at the dashboard and published to the broker

H: setpoint for the heating of the plant (nursery); H is generated by the controller and published to the broker

T\_i: the temperature in the nursery; T\_i results from the simulated process and is published to the broker

T\_o: the outside temperature, which can be considered as an external disturbance.

We have at this stage the following subsystems apart from the broker):

* plant simulation (in the ESP8266); relation to the broker: it subscribes to the setpoint for the heating device (H) and publishes the temperature T\_i in the nursery;
* controller (runs in the RPi); it subscribes to S and T\_i (we assume the ´error detector´ to be included in the controller) and publishes the setpoint H;
* the human interface consisting of the Node-RED dashboard that generates the setpoint plus the display.

Redraw the diagram in such a way that it includes the broker. Draw all data going to and from the broker; in other words, include all the subscribe and publish connections to the broker. Include also the dashboard in the diagram.

The controller is a program that runs autonomously on the RPi. The controller subscribes to the following messages:

import paho.mqtt.client as mqtt      
from time import sleep          
  
global S,T\_i  
broker\_address="localhost"  
S=20.0  
  
def on\_message(client, userdata, message):  
    global S,T\_i  
    if (message.topic== 'S'):  
        S=float(message.payload)  
    if (message.topic== 'T\_i'):  
        T\_i=float(message.payload)  
        H=S-T\_i  
        client.publish("H",H)  
  
client = mqtt.Client()  
client.connect(broker\_address)  
client.loop\_start()  
client.subscribe("S")  
client.subscribe("T\_i")  
client.on\_message=on\_message  
  
while (True):  
    sleep(10)

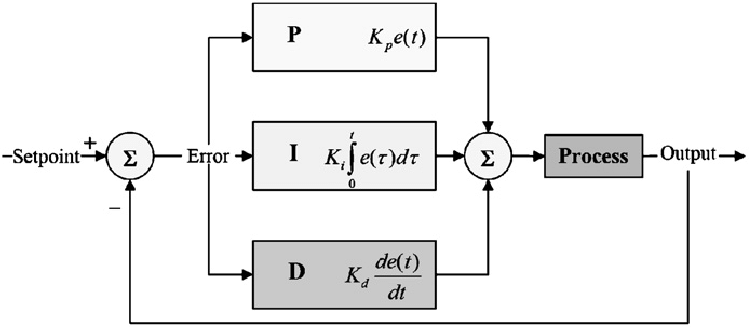
* setpoint (it will receive the setpoint from the dashboard)
* T\_i, the temperature inside the nursery.

Subsequently It calculates the error by subtracting T\_i from the temperature setpoint and applying a formula to calculate the new setting for the heater of the nursery. In chapter 7 we included a controller with a transfer function equal to 1.

Take a look at some program code in the next box:

* Some variables have to be defined global (why?). Some variables have to be given an initial value (why?).
* A message functions is defined in which a check is performed on the source of the message (alternatively two separate on\_message functions could have been defined, each of the messages (variables) it can receive).
* A subscription to the two topics is made.
* The program serves as a proportional controller with a transfer function equal to 1.

Write and test the additional code.

1. Practical implementations of the controller

In the previous chapter we used a controller with a transfer function equal to 1. We can improve the behavior of the system by choosing another value for the proportional component and adding an integral and a derivative component (I and D). In that way we get the basic structure for the controller: the PID controller.

* 1. The PID controller

def on\_message(client, userdata, message):  
    global S,I,previous\_error  
    Kp=1.0  
    Ki=0.1  
    Kd=0.5  
    delta\_t=2  
    if (message.topic== 'S'):  
        S=float(message.payload)  
    if (message.topic== 'T\_i'):  
        T\_i=float(message.payload)  
        error=S-T\_i  
        I=I+delta\_t\*error  
        D=(error-previous\_error)/delta\_t  
        H=Kp\*error + Ki\*I + Kd\*D  
        client.publish("H",H)  
        previous\_error=error

The controller consists of the following components:

* Proportional component

P = error

In which error = Setpoint - T\_i.

* Integral component I

I = I + delta\_t\*error

In which delta\_t is the time interval between two calculations.

* Derivative component D

D = (error - previous\_error)/delta\_t

In which we calculate the difference in two subsequent errors; for this reason, we store the previous\_error at the end of each cycle.

The formula for H becomes: H=Kp\*error + Ki\*I + Kd\*D

The code in the box shows the adjustments to the on\_message function. Include these adjustments as well as other necessary modifications in the code to make the controller operational. Plot in a graph the temperature setpoint S and the temperature in the nursery T\_i. Comment on the shape of the T\_i plot.

* 1. Adjusting the parameters of the PID controller

Optimize the controller in such a way that it has a maximum overshoot of not more than 5% and a minimum time constant (= the **time** needed for the process output variable to reach 63.2% of its total and final change.). The parameters that can be adjusted are: Kp, Ki and Kd. Make screenshots of the various plots for later comparison. You can add extra information to your plot with the function plt.figtext(0.05,0.05,´any text´) whereby the first two parameters are the co-ordinates where to write the text (you might have to choose other values depending on you laptop setting; 0,0 refers to the bottom left side corner).

* 1. **Limited heating capacity**

Extend the plotted graph in such a way that it shows the setpoint for the temperature S, the setpoint for the heater H and the temperature in the nursery T\_i. In the box the adjustments for the on\_message function have been shown. Add yourself the other necessary modifications.

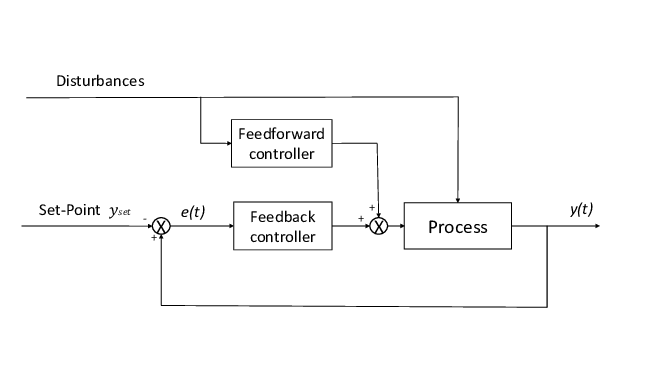
def on\_message(client, userdata, message):          
    global S,T\_i,x,y1,y2,y3  
    if (message.topic== 'S'):  
        S=float(message.payload)  
    if (message.topic== 'T\_i'):  
        T\_i=float(message.payload)  
    if (message.topic== 'H'):  
        y3.append(float(message.payload))  
        y2.append(S)  
        y1.append(T\_i)  
        x.append(x[-1]+step)  
        x = x[-100:]                      
        y1 = y1[-100:]              
        y2 = y2[-100:]  
        y3 = y3[-100:]

In the examples of the previous paragraph, we have assumed a linear heating element that had no upper limit and that was also able to cool. In practical situations that will rarely be the case.

Let’s assume a heating unit that can handle setpoints from 0 to 20.

Investigate the effect of the PID controller on the overall performance of the process control.

Note that limiting the value of H (which means that the calculated value is outside the range but the heater doesn´t react beyond its limits) can have the effect of a time delay. The reason is that the integrating component behaves like a kind of memory function.

* 1. **Feed forward control**

The outside temperature is simulated in the process simulation in the ESP. A temperature sensor is assumed to measure the temperature and send the result to the broker.

Feedforward control can be used very successfully to improve a control loop’s response to disturbances. Feedforward control reacts the moment a disturbance occurs, without having to wait for a deviation in a process variable. If any process control loop is subject to large, measurable disturbances, it can benefit greatly from feedforward control.

A typical and predictable disturbance in the case of the plant nursery is the fluctuation in day and night temperature. This variation can be included in the model by varying T\_o in a sinusoidal 24-hour rhythm. For the sine function and the value of pi we import ´math´ (see the code in the box). Work out the formula for T\_o assuming a sinusoidal pattern and taking 30 minutes to be equal to 2 seconds in our simulation (if we use the real time as reference time for the display, then every next calculation produces the temperature 0.5 hour later – hence, t = t + 0.5). Furthermore, the average outside temperature is 20 degrees while the minimum temperature is 0 degrees.

import math

pi=math.pi

t=0

T\_o = <formula>

t=t+0.5

c.publish(´T\_o´,T\_o)

Investigate the effect of fluctuations in the outside temperature on the temperature in the nursery using the PID controller (so, without the feedforward controller). Use as well the original setting of the PID controller as the setting resulting from section 2b and 2c. Consider the effect with a temperature setpoint of 20 and a setpoint of 40.

# imports and definition global

server="10.3.141.1H=0.  
T\_i=20.0  
T\_o=20.0  
T\_w=20.0  
C\_w=0.5  
C\_i=0.1  
C\_o=0.02  
pi=math.pi  
  
def on\_message(topic, msg):  
    global H  
    H=float(msg)  
      
c = MQTTClient("any\_id", server)  
c.set\_callback(on\_message)  
c.connect()  
print('connected')  
c.subscribe('H')  
  
while (True):  
    c.check\_msg()  
    T\_w=T\_w+C\_w\*H-C\_i\*(T\_w-T\_i)  
    if (T\_w>100): T\_w=100  
    if (T\_w<0): T\_w=0  
    T\_i=T\_i+C\_i\*(T\_w-T\_i)-C\_o\*(T\_i-T\_o)  
    sleep(2)  
    c.publish('T\_i',str(T\_i))

Since the outside temperature can be measured, it is also available to the controller. The controller can be optimized using a feed-forward component in the control algorithm.

In our case we have exact knowledge of the process since it was simulated. So, we know what the impact of the outside temperature is on the process: 0.02\*(T\_i – T\_o). If we add the same amount to the control signal (taking into account the conversion factor of 0.1 from heat setpoint (H) to heat energy in the process) we can compensate the effect of the fluctuating outside temperature completely by adding 0.2\*(T\_i – T\_o) as a feedforward signal (why?). Verify this (the feedforward should compensate the effect of the variations in T\_o completely). Show the effects with screenshots of the chart.

Of course, in practical situations we do not know the effect of disturbances exactly. Suppose that there is a strong wind and the heat loss is not 0.02·(Ti-T\_o) but 0.03·(Ti-T\_o). Investigate the effect ofe deviations between the model used in the feedforward controller and the actual situation. A solution would be to make the feedforward modelling more accurate using more parameters. In this process also Artificial Intelligence comes in.

* 1. **Second order process**

Up to now we have been working with a first order process. The heater heats the air in the nursery directly. In many installations however, the heater heats water that circulates in a water system. The hot water heats the air. The heat exchange between water and air is represented byC\_i\*(T\_w-T\_i).

The system is characterized as a second order process.

Optimize the controller experimentally or by using techniques from control engineering.

1. **Artificial Intelligence**

Artificial Intelligence plays an important role in many IoT projects.

We mentioned already the contribution AI can make to control systems. But even more important is image analyzing. Diseases can be detected in plants by analyzing the images of leaves. But also, other irregularities can be detected by sending images to a properly trained AI system.

The broker architecture facilitates the inclusion of AI systems since it decouples the AI system as a separate unit.

When applying a camera somewhere the image can be sent through the broker to a more powerful computer that runs an AI program. Specially since AI software is quite specific, disconnecting from the other system components can have advantages. For instance, because in that way the AI application can run elsewhere, like somewhere in the cloud.

Before an image is offered to an AI network the image is preprocessed. That means that the size and range of brightness are being adjusted. This happens with the images used for training as well as with the images offered for prediction – both of them go through the same pre-processing procedure. That means that you can´t just ´buy´ a trained network and decide yourself on how to pre-process your images.

The main forms of pre-processing are:

* *Object centralization* (place the object in the center of the image) and *object normalization*. There is usually a range of sizes for the object that give sufficient accuracy (again depending on the training procedure). With object normalization we make the object fit into this range. Apart from that, if the neural network has been trained with the object in a fixed rotational position, we might have to rotate it.
* *Pixel Normalization* (scale pixel values to the range 0.0 to 1.0) and pixel *Standardization* (scale pixel values to have a 0 mean and standard – Gaussian – variance).

Where do we do the pre-processing? In the computer with the camera attached or in the computer that runs the AI application.?

Generally, the computer that runs the AI system is more suitable to run advanced pre-processing software. However, it is desirable to send only images over the broker that are actually suited for prediction by the AI software. Sometimes it is also preferred to resize the image to its proper dimensions before transmission – for instance when bandwidth is critical. Here we touch a more general question: what processing is done at the edge (near the sensor) and what processing is done in the central – usually more powerful – systems? Transmission bandwidth and security ply a major role in answering this question in the design phase. We will come back to this topic later in this chapter.

* 1. **Make a picture with the webcam**

import cv2  
import numpy as np  
  
img\_counter=0  
  
cam = cv2.VideoCapture(0)  
while (True):  
    ret, frame = cam.read()  
    cv2.imshow("image", frame)  
    k=cv2.waitKey(400)  
    if k%256 == 27: # ESC pressed

        print("Escape hit, closing...")  
        break  
    elif k%256 == 32: # SPACE pressed  
        img\_name = "opencv\_frame\_{}.png".format(img\_counter)  
        cv2.imwrite(img\_name, frame)  
        print("{} written!".format(img\_name))  
        cv2.imshow(img\_name,frame)  
        img\_counter += 1  
cam.release()

Take a sheet of white paper and write any digit on it (leave some white space around the digit; use for instance a black felt-tip pen). Take a picture with the digit in upright position. Use either the CSI (Camera Serial Interface) camera connector in combination with Picamera (it is discussed in part 1 – sensors and actuators) or use the webcam in combination with cv2. We will use the webcam in this exercise.

OpenCV (we use the version cv2) is very suitable for making and manipulating images[[12]](#footnote-13). The program in the box shows a sequence of images from the webcam on the screen. The program is terminated by hitting the escape key.

* cv2.VideoCapture(0) captures the video of the first camera (which is shown by the parameter being 0

in the endless loop) a frame is read, converted to grayscale and shown (the parameter ´ret´ tells whether a frame is returned; we don´t use it here)

* every cycle the program waits for a certain period of time (the period of time is passed as a parameter to the waitKey function and measured in milliseconds) to see whether a key has been pressed (the period of time is chosen in such a way that the software can handle the frame without getting clogged)
* when the escape key is pressed the loop is left (with the break statement) and the camera is released
* when the space key is pressed the image is written to your folder. To store multiple images under different name, a name for the file is created with a sequence number.

Run the program. Check the saved image. The digit should be sufficiently large with a white area around it.

import cv2  
  
d1=75  
d2=d2+50  
d3=2  
d4=5  
  
cam = cv2.VideoCapture(0)  
stop=False  
  
while (not stop):  
    ret, im = cam.read()  
    im=cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY) (1)  
    ret,im=cv2.threshold(im,150,255,cv2.THRESH\_BINARY\_INV) (2)  
    img=im.copy() (3)  
    im=cv2.rectangle(im,(d1,d1),(640-d1,480-d1),(255),1) (4)  
    cv2.imshow("image", im)  
    cv2.waitKey(400)  
  
    c1=cv2.countNonZero(img[:,d1:d1+1])< d3 (5)  
    c2=cv2.countNonZero(img[:,640-d1:640-d1+1])< d3  
    c3=cv2.countNonZero(img[d1:d1+1,:])< d3  
    c4=cv2.countNonZero(img[480-d1:480-d1+1,:])< d3  
          
    c5=cv2.countNonZero(img[:,d2:d2+1])> d4 (6)  
    c6=cv2.countNonZero(img[:,640-d2-1:640-d2])> d4  
    c7=cv2.countNonZero(img[d2:d2+1,:])> d4  
    c8=cv2.countNonZero(img[480-d2-1:480-d2,:])> d4  
      
    stop= c1 and c2 and c3 and c4 and ((c5 and c6) or (c7 and c8)) (7)  
      
cv2.imshow("image", img)  
cv2.waitKey()  
  
cam.release() (8)  
cv2.imwrite('digit.png', img) (9)

The size of the image is: width(y)=640, height(x)=480. The origin for measuring a pixel position is the top left corner and the direction to the right and downwards are taken as positive.

These valueas can be obtained by: cam.get(cv2.CAP\_PROP\_FRAME\_WIDTH) and cam.get(cv2.CAP\_PROP\_FRAME\_HEIGHT).

There are two approaches to obtaining the proper picture:

* we can look at the screen and use our judgement to see when to hit the spacebar and save the image, or
* let the software decide when the image is fit to be used for further analysis.

We will now investigate and use the second one (note: you could have a robot moving stepwise over the relevant area but in this case we do it by hand so you still have to position the camera in such a way that the image satisfies the requirements)

Since we have to get an image with the object (in our case a digit written on a sheet of paper[[13]](#footnote-14)) at the proper position and with the proper size, the following two requirements have to be fulfilled for a usable image (usable in the sense that it can be used for prediction using our trained network):

* the object has to be at the proper position (not to close to the edges)
* the object has to have the proper size

To satisfy these requirements we are going to draw a rectangle in the image and only save the image if it fills the rectangle nicely.

The program code in the box shows the subsequent steps (import and initial calculation takes some 30 seconds):

1. the image is converted to grayscale
2. the threshold function converts all light pixels (pixels above the threshold to black (0) – that is where the INVert stands for – and the pixels below the threshold to white (255)); )[[14]](#footnote-15); ret tells you whether the thresholding has been executed successfully
3. we keep a copy of the image for later use
4. a rectangle is being drawn and the new image is being shown (the rectangle has black lines and it outlines an area with a distance of d1 (in the example we took d1=75 pixels) from the edges)
5. a test is performed to see whether the object is within the rectangle – we use the original image to avoid conflict with the pixels that are part of the rectangle
6. a test is performed to check whether the object is large enough (close enough to the upper and lower side or to left and right side – why is the test set up like this?)
7. when all conditions are satisfied, we have obtained a suitable image
8. the cam object is being deleted
9. the image is being stored.

*Why do we check for the picture to fulfill the requirements with stop= c1 and c2 and c3 and c4 and ((c5 and c6) or (c7 and c8)) instead of stop= c1 and c2 and c3 and c4 and c5 and c6 and c7 and c8?*

Note: the values for d1, d2, d3 and d4 have been chosen more or less arbitrarily.

*What is the effect of each of these parameters? Note that an image of a white surface can always contain some black pixels because dirt on the lens or a damaged image-memorycell.*

* 1. **Passing images through the broker**

If you did not manage to make a proper picture, you can use the image that is stored on the SD card.

import paho.mqtt.client as mqtt

import cv2

broker\_address=´localhost´

client = mqtt.Client()

client.connect(broker\_address)

# use the webcam to make an image

# or load load the image from memory

# or use: img=cv2.imread('digit.png',0)

client.publish('image',bytearray(img))

client.disconnect()

Write a program that reads the image and publishes it to the broker.

Also write a program that subscribes to the topic that was used for publishing and shows the image.

Your approach for the publishing program will be more or less like this (see code in the box):

* You establish a connection to the broker:
* You make an image that satisfies the criteria or load the image from the SD card; the image must be within the limits of the broker[[15]](#footnote-16);
* The image is published under the topic ´image´ as a bytearray.

You can test publishing process separately by:

* Start the broker
* Start a mosquito\_sub process (mosquitto\_sub -h localhost -t ´image´)
* Start the program for publishing. The subscribe process will print the data it receives which will just be a single array of 307200 bytes (the number of bytes of the 640x480 greyscale image).

Your approach for the subscribing program will be more or less like this:

# import, global constants

and def on\_message(client, userdata, message):

    global img,received

    img=np.frombuffer(message.payload,dtype=np.uint8)

    received =True

client = mqtt.Client()

client.connect(broker\_address)

client.loop\_start()

client.subscribe("image")

client.on\_message=on\_message

try:

    while (not received):

        time.sleep(1)

    img=np.reshape(img,(480,640))

    cv2.imshow("image", img)

    cv2.waitKey(0)

    print("Key hit, closing...")

except:

pass

client.disconnect()

* You have a function that will be called when a message has been received. In this function a flag is set telling the message has been received;
* In the main program you have got a loop that continues until a message has been received (the flag and the message are global variables because they are used in the message handling thread as well as in the main program);
* The message will be a linear flow of bytes of type uint8. This flow of bytes is converted into a numpy array; in the main program – but it could also have been done in the on\_message function – the 1-dimensional array is converted into a 2-dimensional array which can be shown as an image;
* The program should continue to allow you to see the image. Therefore, when an image has been received, it is shown until you hit a key (like was done in previous exercises).
  1. **Preprocessing and prediction**

Before an image is offered to a neural network for training or when it is offered for prediction to a trained AI network, the image is preprocessed. That means the size and range of brightness are adjusted (these are the basic adjustments; more advanced adjustments are possible). Note that both sets of images (the images for training and the images for prediction) should go through the same pre-processing procedure. That means that you can´t just ´buy´ a trained network and decide yourself on how to pre-process your images – that could reduce the accuracy substantially. In our case the network we use has been trained on a set of pre-processed images from the MNIST database of handwritten digits.

The images in the MNIST database have gone through a series of preprocessing stages. However, if we use only the basic pre-processing for our images, we can come close enough for images of sufficient quality.

The forms of pre-processing that we are going to use are:

* *Object centralization* (place the object in the center of the image) and *object normalization*. There is usually a range of sizes for the object that give sufficient accuracy (again depending on the training procedure). With object normalization we make the object fit into this range. Apart from that, if the network has been trained with the object in a fixed rotational position, we might have to rotate it.
* *Pixel Normalization* (scale pixel values to the range 0.0 to 1.0) and pixel *Standardization* (scale pixel values to have a 0 mean and standard – Gaussian – variance).

In the remaining exercise we will limit ourselves to object centralization and cropping.

There are two main methods for object centralization:

* Draw a rectangle around the object by finding the line closest to the object that consists of only black pixels. Determine the center point of this rectangle. Crop the square that surrounds the object (to crop an area from an image means making a new image containing just that area);
* Determine the center of gravity of the object and take this point as the middle of a square surrounding the object. Again, crop the square around the object in such a way that the cropped image contains just that object plus some surrounding black area.

We will use the second method. This is the method used for composing the mnist database (*how could you verify this?*). The first method is left as an exercise. Compare the two methods with respect to their accuracy in the prediction.

To determine the center of gravity of a ´blob´ or ´region of interest´ (ROI) we go through the following steps:

import cv2  
from tensorflow.keras.models import load\_model

def show(text,image):

cv2.imshow(text,image)  
cv2.waitKey(0)

cv2.destroyAllWindows()  
  
im=cv2.imread("<image name>",0)  
show(ímage´,im)

M=cv2.moments(im)  
y=int(M['m10']/M['m00'])  
x=int(M['m01']/M['m00'])  
cv2.circle(im,(y,x),5,(255),2)  
  
d1=10  
d2=10  
d3=10  
d4=10

d5=10  
while (cv2.countNonZero(im[:,y-d1:y-d1+1])>0): d1=d1+1  
while (cv2.countNonZero(im[:,y+d2:y+d2+1])>0): d2=d2+1  
while (cv2.countNonZero(im[x-d3:x-d3+1,:])>0): d3=d3+1  
while (cv2.countNonZero(im[x+d4:x+d4+1,:])>0): d4=d4+1  
d=max(d1,d2,d3,d4)+d5  
  
img=cv2.rectangle(im,(y-d,x-d),(y+d,x+d),(255),2)  
show(´rectangle´,img)

im=im[x-d:x+d,y-d:y+d]  
show(ímage cropped´,im)

im=cv2.resize(im,(28,28))  
show(ímage resized´,im)

im=255-im  
im=im.astype('float32')  
im/=255  
  
model=load\_model('pretrained\_model1.h5')  
im=im.reshape(1,28,28,1)  
pr=model.predict(im)  
print(pr)

* Read the image from the background memory. In our case the image was sent as a threshold-image with black background. A ´show´ function is defied so that it can be included easily to see the progress of each step. You can include ´show´ function wherever you like to keep track of the modifications done to the image, but you can also leave them out.
* With ´cv2.moments´ the ´moments´ are being calculated whereby a ´moment´ is equal to the distance times the value of the pixel (similar to the moment of a force). The variable ´m10´ represents the sum of the y coordinates of all non-black pixels of the blob (in our case there is only one blob – the digit) multiplied by their pixel value (in our case 255 for white). When dividing this sum by the sum of all pixel values we get the center point (= the point that would give the same moment if all ´light´ would be concentrated there. Notice the similarity with moment of force. Try yourself to make the calculation for the situation that the blob consists of only 2 or 3 pixels. We can do the same for the x-direction.
* With cv2.circle we can draw a circle around the center of gravity.

*There are two issues that need attention here:*

1. *As we have seen in the previous chapter images are generally being represented with the x-axis vertically downwards and the y-axis horizontally to the right. For matrices the first parameter (i or x) represents the column number and the second (j or y) the row number. We follow here the image co-ordinate system. Try yourself to draw some points or lines (img[:,6:8]=255 draws a white vertical line of 2 pixels thickness) in an image and show the image. The formula for moments is based on matrix co-ordinates but converted to image co-ordinates.*
2. *Whenever we manipulate an object – for instance img=255-img – a new object is formed (only the statement im=img will result in a link to the same object – if we need a new object we can use im.copy(). In the program we introduce a new object img since we draw a rectangle in the image and we want to use the ´clean´ image later again.*

* The images in the MNIST database are based on 28x28 matrices. So, we first find the surrounding square for the digit. For this purpose, we determine the distance from the center to the edge of the digit in each of the four directions (the starting value for d1 to d4 have been chosen arbitrarily in such a way that they will definitely be smaller than the largest distance from the center to the edge of the digit). Subsequently we take the largest value and add d5 (in the sample code equal to 10) pixels to avoid the digit touching the edge (the value of 10 is chosen arbitrarily). Subsequently we crop a section from the original image corresponding to the square we calculated. *Determine whether there is a more optimal value for d5.*
* The cropped section is resized to a 28x28 image suitable for the MNIST network.
* We can use one of the pre-trained models to predict the digit that the image shows. By printing model.predict(im) we can see what score is obtained for each of the possible outputs.

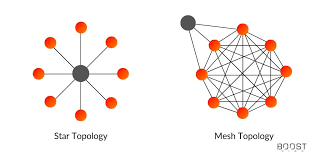
1. **Aspects of security**

Security in ICT systems is very comprehensive. Even when limiting the discussing to IoT systems the variety of size, context and specific requirements is too large to give a useful overview of security measures. We will therefor limit our discussion here to small and medium size IoT projects.

Usually when discussing security, a distinction is made between confidentiality, integrity and availability[[16]](#footnote-17). We will use here a different classification. We will distinguish:

* Access to the system by unauthorized persons. These persons can get access to data or limit the proper functioning of the system.
* Unavailability of the system due to technical failures
  1. System architecture

The architecture of an IoT system has a large impact on (future) measures with respect to the security.

Whenever a number of nodes (computers and sensors or actuators with computer capabilities) exchange data, there are two basic topologies:

* The start topology: every node (computer) communicates only with a central node and through that central node with other nodes.
* The mesh topology: every node communicates directly with other nodes that it needs to exchange data with.

Apart from the fact that you can and will have all kind of combinations of the two, there is an important footnote to be placed: there is a difference between the logical communication level and the physical transport level. In the mesh topology the logical communication is a mesh but the physical topology is still a star since all nodes exchange data through a central router, with each other and possibly with the internet. In the start topology the central node can combine the physical level (router function) and the logical level (broker function) – not necessary in a single device but in two strongly interconnected devices.

The main disadvantage of the start topology is the fact that the central node is a single point of failure as far as the broker function is concerned. In the mesh topology the router is still a single point of failure but there is no central broker. With present-day technology it is possible to make extremely reliable equipment so the ´single point of failure´ aspect plays a less important role. The main advantages of the star topology are the simplicity and flexibility. The effect of the simplicity is that it the number of states (= the number of situations the system can be in) is relatively small and hence, testing is relatively easy. In the mesh topology the number of states soon increases to levels whereby it is virtually impossible to test every possible situation. The flexibility of the star topology is higher because ´rays´ of the star can be replaced without affecting the remainder of the system. Of course, as long as the protocols at the data and transport level remains the same.

* 1. Unauthorized access

Usually, part of the IoT system, typically the equipment that has no direct connection to the physical world, is set up in a safe environment. The central router, the dashboard, AI systems are examples of this category of equipment. These computers typically communicate with the broker through a local wifi system. Sensor and actuator related communication might be over long distances using the public internet of mobile phone net as a communication channel.

Hence, there are two ways an attack can come into the system:

* A hacker can get access to the wifi networks, directly or through a computer that is connected to the wifi network and to internet.
* Through sensors that use public transport channels for transmission.

In both cases a proper level of security can be obtained with the encryption, authentication and authorization techniques available in the broker. Hereby the leading principle should be: never give more access rights than necessary.

* 1. Exercise

< An exercise on unauthorized access will be included. It has not been included yet for the following reasons:

There are various levels of access security that can be implemented in the Mosquitto broker. It requires a certain level of knowledge of access security with the students - <http://www.steves-internet-guide.com/mqtt-security-mechanisms/> ; it also requires adjustments in the mosquito.conf file – in fact adding a .conf file to the /etc/mosquito/conf.d directory; this last addition step requires an automatic removal at boot - <https://www.raspberrypi.org/documentation/linux/usage/systemd.md> ; this requires specific actions when a new SD card has to be configured.>

* 1. Availability

In many industrial IoT systems a high level of availability is essential. Various techniques are available to get the availability at the proper level. Hereby it must be mentioned that availability is not only brought down by hardware failure but also by software bugs and errors made by the operator. In this context the term ´robust´ is also in use, meaning that whatever hardware or software error occurs, the system continues to function. Maybe the system functions at a lower level due to faults, but a disaster is avoided. One way to reach this is by alarming an operator. Another approach can be to bring the system down gracefully.

The advantage of the star-shape broker structure is that it is relatively easy to give each of the ´rays´ a different and appropriate availability level. For instance, if there are 20 temperature sensors and one of them is not functioning, it can be signaled to the operator but the average temperature can still be calculated.

* 1. Exercise

Take the process that we considered in the previous chapters (assume the pictures are being made by an employee; so, they don´t require to be checked).

Write an application that detects unusual situations. Use the traffic light to show an alarm signal (green stands for: system operates correctly; yellow stands for: action to be taken but not urgent; red stands for: immediate action required).

Hint 1: with mosquitto\_sub -h localhost -t # you receive all published messages.

Hint 2: Since the setpoint from the Dashboard is asynchronous, there is no way to tell whether a new value for the setpoint has been missed. Therefor include an automatic resend (msg resend node) in the flow.

Note: in more complex systems an AI network could be trained to detect exceptional situations. This would typically be an example of *unsupervised learning*, since the network would be trained to detect anomalies.

Appendix 1 ESP 8266 pin layout

## Pinout Amica Nodemcu (ESP2866) – EselsbrückeBest Pins to Use – ESP8266[[17]](#footnote-18)

One important thing to notice about ESP8266 is that the GPIO number doesn’t match the label on the board silkscreen. For example, D0 corresponds to GPIO16 and D1 corresponds to GPIO5.

The following table shows the correspondence between the labels on the silkscreen and the GPIO number as well as what pins are the best to use in your projects, and which ones you need to be cautious.

The pins highlighted in green are OK to use. The ones highlighted in yellow are OK to use, but you need to pay attention because they may have unexpected behavior mainly at boot. The pins highlighted in red are not recommended to use as inputs or outputs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Label** | **GPIO** | **Input** | **Output** | **Notes** |
| **D0** | **GPIO16** | no interrupt | no PWM or I2C support | HIGH at boot used to wake up from deep sleep |
| **D1** | **GPIO5** | OK | OK | often used as SCL (I2C) |
| **D2** | **GPIO4** | OK | OK | often used as SDA (I2C) |
| **D3** | **GPIO0** | pulled up | OK | connected to FLASH button, boot fails if pulled LOW |
| **D4** | **GPIO2** | pulled up | OK | HIGH at boot connected to on-board LED, boot fails if pulled LOW |
| **D5** | **GPIO14** | OK | OK | SPI (SCLK) |
| **D6** | **GPIO12** | OK | OK | SPI (MISO) |
| **D7** | **GPIO13** | OK | OK | SPI (MOSI) |
| **D8** | **GPIO15** | pulled to GND | OK | SPI (CS) Boot fails if pulled HIGH |
| **RX** | **GPIO3** | OK | RX pin | HIGH at boot |
| **TX** | **GPIO1** | TX pin | OK | HIGH at boot debug output at boot, boot fails if pulled LOW |
| **A0** | **ADC0** | Analog Input | X |  |

1. AI stands for Artificial Intelligence [↑](#footnote-ref-2)
2. <https://iot-analytics.com/internet-of-things-definition/> gives some historical background. [↑](#footnote-ref-3)
3. MQTT stands for **MQ Telemetry Transport**  [↑](#footnote-ref-4)
4. <http://mqtt.org/> [↑](#footnote-ref-5)
5. A specific application listens (at the specified IP address) to a specific port number. The **port number** is “tacked on” to the end of the **IP address**, for example, “192.168. 1.67:80” shows both the **IP address and port number**. When data arrives at a device, the network software looks at the **port number** and sends it to the right program. The default port number for HTTP traffic is 80. The broker port is 1883. The Node\_RED port is 1880. [↑](#footnote-ref-6)
6. Newer versions have larger volumes of flash memory. [↑](#footnote-ref-7)
7. Instead you can also use the command ./esptool.py erase\_flash (the python command is included in the shebang script; in that case the full path of the program that is to be executed is required – therefore, the dot-slash is included). The **shebang** is a special character sequence in a **script file** (since Python is based on interpreting, every program is in fact a script file) that specifies which program should be called to run the **script**. The **shebang** is always on the first line of the **file**, and is composed of the characters #! followed by the path to the interpreter program. [↑](#footnote-ref-8)
8. Without the static IP address, the ESP might get a different IP address from the router after switching it off and on. This would mean you have to check the IP address every time after rebooting. [↑](#footnote-ref-9)
9. <https://www.letscontrolit.com/wiki/index.php/Tutorial_Rules> [↑](#footnote-ref-10)
10. After downloading esptool it says in the manual to use: esptool/esptool.py --chip esp8266 erase\_flash. However, the –chip parameter is default and for the lab the program was copied to the working directory. Esptool is a python program. [↑](#footnote-ref-11)
11. When the microPython interpreter has been downloaded the manual says to use: esptool/esptool.py --chip esp8266 write\_flash --flash\_mode dio --flash\_size detect 0 /home/pi/Downloads/esp8266-20191220-v1.12.bin (the name might differ depending the file you downloaded and the directory into which it was placed); for the lab the file was renamed to esp8266.bin and moved to the working directory; the other parameters are default; 0 is the starting address for storing the firmware. As was explained before we can also use: ./esptool.py write\_flash 0 esp8266.bin [↑](#footnote-ref-12)
12. A useful tutorial on image processing in as well images as video´s is found here: <https://pythonprogramming.net/loading-video-python-opencv-tutorial/> [↑](#footnote-ref-13)
13. The idea is that an image of a leaf is used to identify the presence of a plant disease. We will use the dataset of images of handwritten digits that was also used in the lab on AI. Reasons are the limited power of the RPi and the availability of a suitable dataset with a public pre-processing scheme. [↑](#footnote-ref-14)
14. Thhere are more threshold criteria. See <https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html> [↑](#footnote-ref-15)
15. According to the mqtt standard the maximum size of messages is 256 MB. However, individual implementation can impose their own limitations. [↑](#footnote-ref-16)
16. <https://www.certmike.com/confidentiality-integrity-and-availability-the-cia-triad/#:~:text=Confidentiality%20means%20that%20data%2C%20objects,and%20the%20resources%20they%20need>. [↑](#footnote-ref-17)
17. <https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/> [↑](#footnote-ref-18)