



FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Information Systems - SINF

2019 / 2020

LAB GUIDE - PL2

Setup Project Application

Gil Gonçalves - gil@fe.up.pt
Luis Neto - lcneto@fe.up.pt
João Reis - jpcreis@fe.up.pt
Vitor Pinto - vitorpinto@fe.up.pt

In this Lab Guide we will focus on the following aspects:

1. Overview of the Application Architecture;
2. How to setup Raspberry Pi environment;
3. Create two bidirectional byte streams and transfer data between them;
4. How to run an altered version of the TelosB CM5000 message simulator (MsgCreator Application);
5. How to run the RGBMatrix application to graphically represent sensors and actuators in the context of your Course Unit Project.
6. Objectives for Sprint #1

1. Application Architecture

The main purpose of the Course Unit application to be developed is to read data from multiple available sensors, interpret those data, implement some rules of actuation and then write a set of commands into an actuation platform. Many applications can range from monitoring humidity and temperature of a swimming pool in order to maintain its thermal conditions into a certain desired value, to monitoring the amount of home appliances energy consumption to turn on or off some actuators that can control windows or blinds in order to have an efficient and sustainable house.

This way, the presented application architecture in Figure 1 is composed by three main components: 1) Sensor data generator to be parsed/interpreted by an application; 2) The application that should read these sensor data, process it and perform the corresponding actuation; 3) The sensor and actuator visualizer where sensor data and resulting actuation commands are displayed.

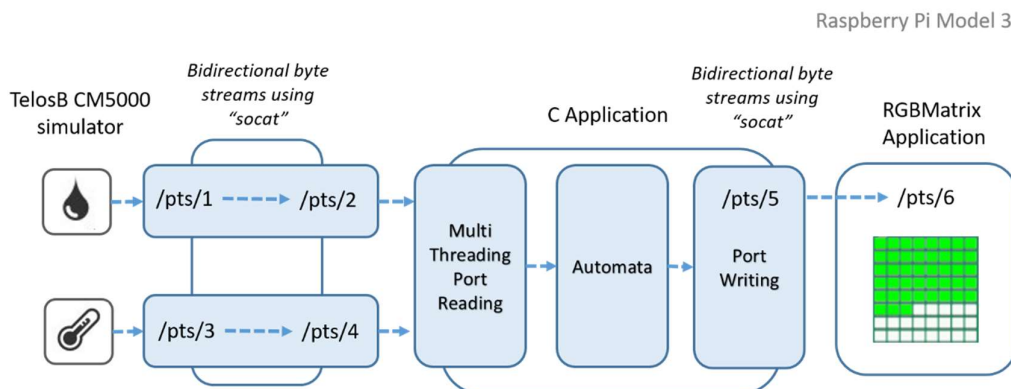


Figure 1 - Overall Application Architecture

In Figure 1 we can see that both humidity and temperature data is being sent from the TelosB CM5000 simulator to the C application that you should implement, where these data are interpreted, processed and finally forwarded to the last component, the RGBMatrix application. In order to establish the communication between these three components, the command line-based utility “*socat*” is used to transfer byte streams. All these applications should be configured and executed in the Raspberry Pi Model 3 that will be provided to you during class, or in the virtual machine environment.

In the remaining sections we will explain how to configure and execute the “*socat*” utility, the TelosB CM5000 simulator and the RGBMatrix application. But first, we need to setup the environment in which all these applications will execute.

2. Setup Raspberry Pi Environment

Raspberry Pi is an embedded system that runs the Raspian OS and is designed to be as simple and small as possible. On one hand, we have an OS ready to use where a large range of applications can be executed, and on the other hand, it is small, portable and light. Hence, this device has a set of interfaces that allow a person to interact with. These interfaces are mainly 4 USB Ports, Ethernet Socket, 3.5mm jack audio interface, HDMI Port and a Micro SD card slot. The full set of interfaces can be seen in Figure 2.

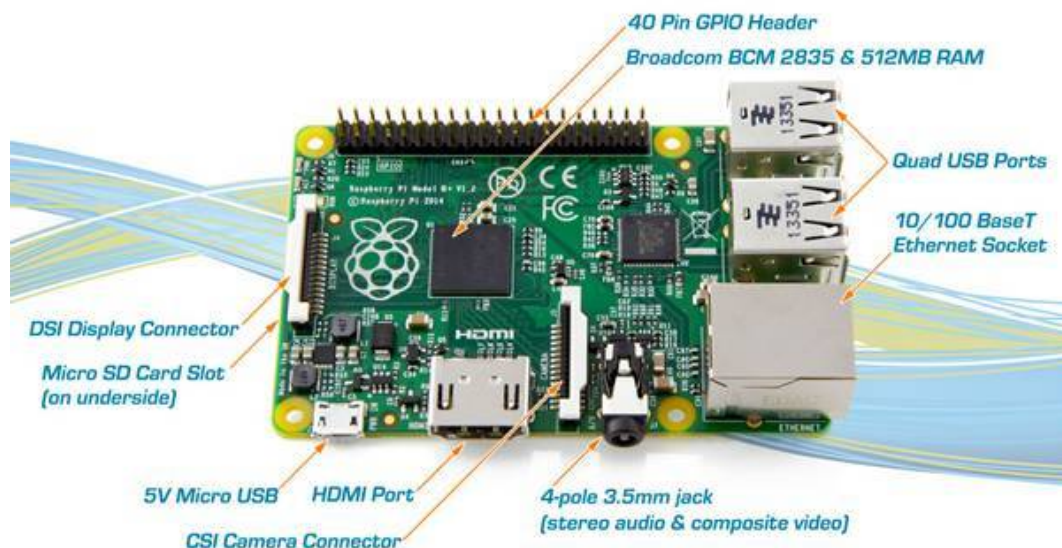


Figure 2 - Raspberry Pi 3 Physical Interfaces

Normally, additional hardware is required to setup the Raspberry Pi such as a keyboard, mouse and a screen with HDMI interface. In the case of the present course, we will be configuring a remote desktop connection to access the Raspberry Pi. The only thing you will need is an Ethernet cable and a Micro SD Card that will be provided to you, at the same time as you will receive the Raspberry Pi.

2.1. Download and Write the Raspbian OS image to the Raspberry Pi

The following two subsections show how you can write the Raspbian OS image to the Raspberry Pi Micro SD Card on both Windows and Ubuntu hosts.

2.1.1. Download and Write the Raspberry Pi OS Image on Windows

In order to run the Raspberry Pi Environment, the first step is to download an *Image* of the Raspbian OS with the SINF course environment already configured and ready to run. This *Image* avoids you to install the Raspbian OS from scratch and configure all the necessary applications and its dependencies to start working on your project. Therefore, you can download the “sinf_env.img” file that contains this environment using the following link:

<https://filesender.up.pt/download.php?vid=1c7895fb-e6dd-4b29-b25f-000037ab14b1>

Next, you will need to copy this *Image* to Micro SD Card that was given to you. For that, you need to download and install on your PC the “Win32 Disk Imager” application that can be accessed using the following link:

<https://sourceforge.net/projects/win32diskimager/>

Afterwards, just run the Win32 Disk Imager application and by clicking the blue folder presented in Figure 3, select the previously downloaded “sinf_env.img” and the Device where you want to copy this *Image*. This Device should match the Micro SD Card Device in your OS (normally, E:).

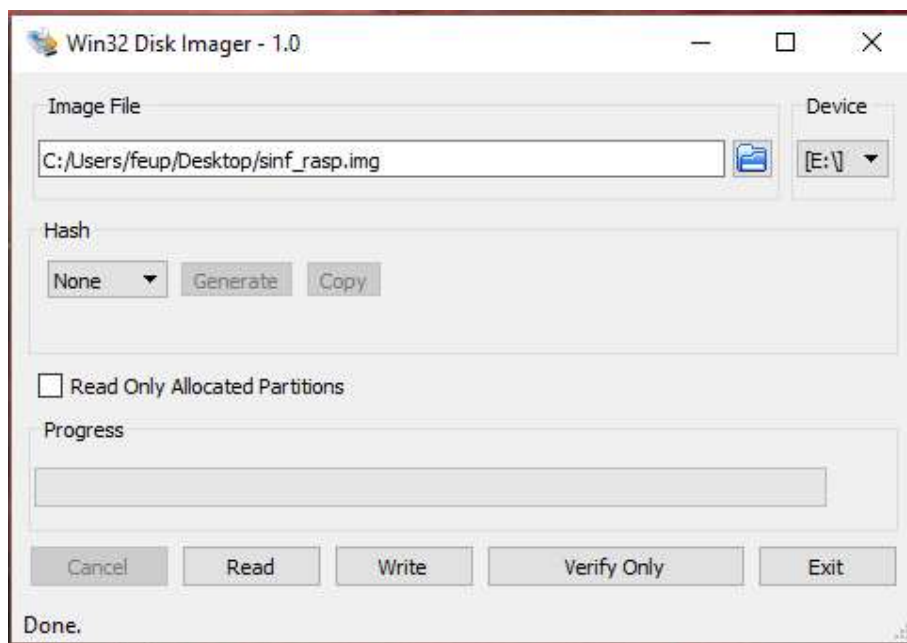


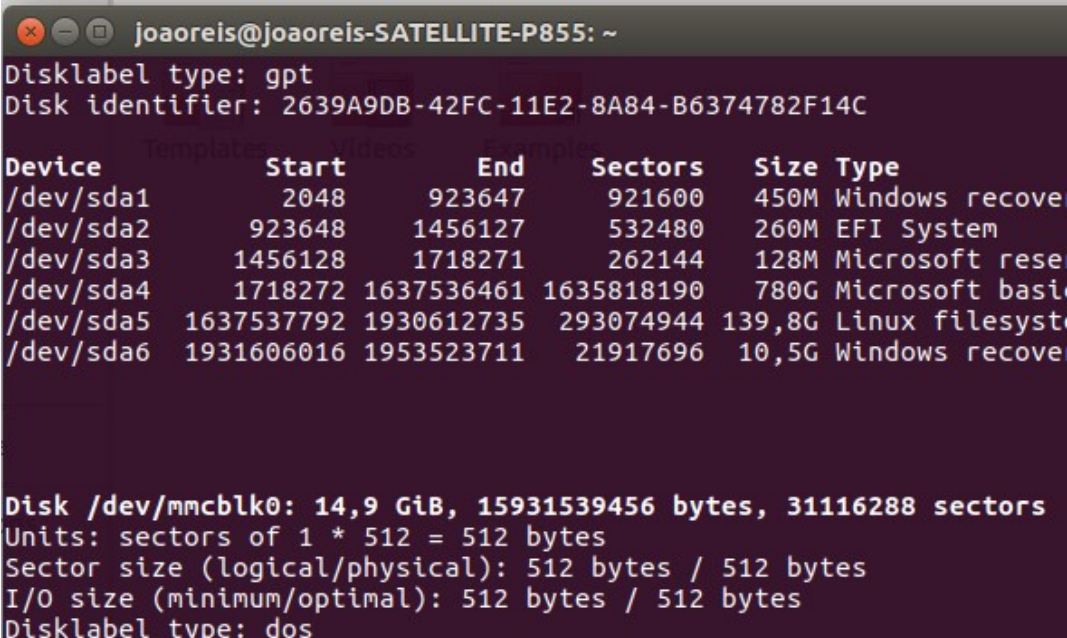
Figure 3 - Write IMG into a formatted SD Card

Then, click “Write” and wait approximately 30 minutes until the process is complete. You might notice that now you have a set of new devices that indicates the writing process to the Micro SD Card was successful. Once the process is complete, safely remove your Micro SD Card (Eject boot (G:)) and insert it on the Raspberry Pi. It should be ready to use.

2.1.2. Download and Write the Raspberry Pi OS Image on Ubuntu

After you download the *Image* of the Raspberry Pi environment, you will need to copy this *Image* to Micro SD Card that was given to you. For that, first you need to introduce your SD Card in the Computer and unmount the created device. The command used to clone the *Image* to the SD Card will not execute if the device is mounted. Then, open the Terminal and check the name of your SD Card device by executing the following command:

```
sudo fdisk -l
```



```
joaoreis@joaoreis-SATELLITE-P855: ~
Disklabel type: gpt
Disk identifier: 2639A9DB-42FC-11E2-8A84-B6374782F14C

Device            Start      End          Sectors      Size Type
/dev/sda1          2048      923647       921600       450M Windows recove
/dev/sda2          923648    1456127      532480       260M EFI System
/dev/sda3          1456128    1718271      262144       128M Microsoft rese
/dev/sda4          1718272   1637536461   1635818190    780G Microsoft basi
/dev/sda5          1637537792 1930612735   293074944    139,8G Linux filesyst
/dev/sda6          1931606016 1953523711    21917696     10,5G Windows recove

Disk /dev/mmcblk0: 14,9 GiB, 15931539456 bytes, 31116288 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
```

Figure 4 - SD Card Device name on Ubuntu

You can see in Figure 4 that there’s a device named `/dev/mmcblk0` with 14,9Gb available to deploy the *Image* corresponding to the Micro SD Card. Afterwards you need to deploy the *Image* into this device. For this case, you need to execute the following command in the Terminal:

```
sudo dd bs=4M if=/path_to_img/sinf_env.img of=/dev/mmcblk0
```

Wait for about 30 minutes and the deployment will be complete. Finally, remove the Micro SD Card from you PC and insert it into the Raspberry Pi. It should be ready to use.

2.2. Connection and Access to the Raspberry Pi

The following subsections detail how you can connect the Raspberry Pi to the Eduroam network and how you can access it from your host PC using VNC.

2.2.1. Connecting the Raspberry Pi to Eduroam

First, insert the Raspberry Pi MicroSD Card in your host machine card reader. Open VirtualBox, go to the SINF Environment virtual machine *Ubuntu_sinf*, and open the VM settings. Navigate to the USB settings and add the USB Device corresponding to the MicroSD Card, as illustrated in Figure 5.

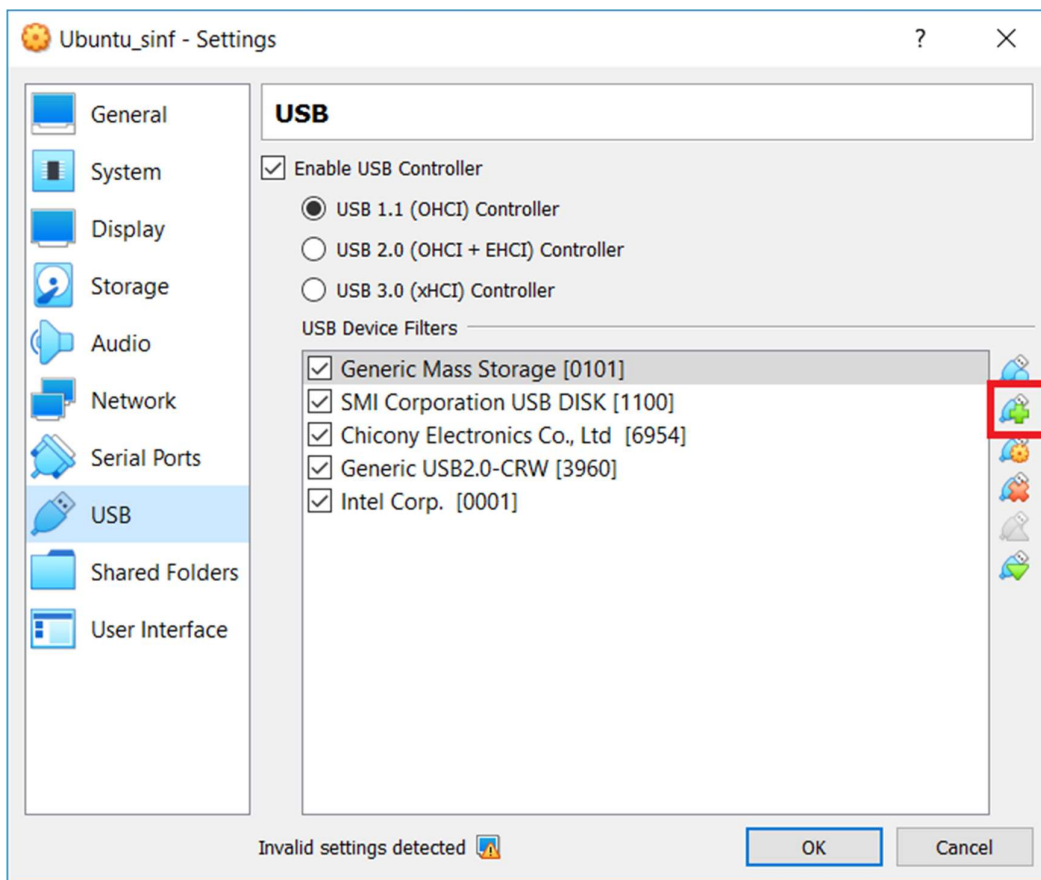


Figure 5 - VirtualBox USB settings.

Start your SINF Environment Virtual Machine (go to *SINF1920 - PL1* if you didn't set it up before). Start a terminal (ctrl + alt + t) and type the following command: *hash eduroam password*. Follow the steps as in Figure 6, you must type your student user and password.

```

pi@SINF: ~
pi@SINF:~$ hash_eduroam_password
Enter your username (example: "user@fe.up.pt") : lcneto@fe.up.pt

Enter Password :
Re-enter Password :
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=PT

network={
    ssid="eduroam"
    scan_ssid=1
    key_mgmt=WPA-EAP
    eap=PEAP
    identity="lcneto@fe.up.pt"
    password=hash:ffb91205a3d288362d86c529728b9dc0"
    phase1="peaplabel=0"
    phase2="auth=MSCHAPV2"
}
pi@SINF:~$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0          11:0    1 1024M  0 rom
sda           8:0     0   25G  0 disk
├─sda2        8:2     0    1K  0 part
├─sda5        8:5     0 1022M  0 part [SWAP]
├─sda1        8:1     0   24G  0 part /
└─mmcblk0    179:0     0 14,9G  0 disk
   ├─mmcblk0p2 179:2     0    1K  0 part
   ├─mmcblk0p7 179:7     0 12,8G  0 part /media/pi/root
   ├─mmcblk0p5 179:5     0   32M  0 part /media/pi/SETTINGS
   ├─mmcblk0p1 179:1     0   1,6G  0 part
   └─mmcblk0p6 179:6     0   69M  0 part /media/pi/boot
pi@SINF:~$ cp wpa_supplicant.conf /media/pi/boot/
pi@SINF:~$ echo "SINF_GROUP_X" > /media/pi/boot/grupo_sinf.txt
pi@SINF:~$ sudo umount /dev/mmcblk0?*
umount: /dev/mmcblk0p1: not mounted
umount: /dev/mmcblk0p2: not mounted
pi@SINF:~$

```

Figure 6 - Eduroam wpa_supplicant.conf file generator.

This command outputs a *wpa_supplicant.conf* file in the same directory you execute it. Copy this file (or the console output) to the *boot* partition of your Micro SD Card, as illustrated in Figure 6.

Create a text file named “*grupo_sinf.txt*”, add a line to this file with your group identification (e.g. “SINF_14”), and finally save this file to the *boot* partition of your Micro SD Card. You can do this by copying the *echo* command in Figure 6.

Unmount the MicroSD Card from your virtual machine and remove it. You can copy the *umount* command in Figure 6. Insert the MicroSD Card in the

Raspberry Pi and turn it on by connecting it to the power. Wait for approximately one minute, go to your host machine and open the following link:

http://systec-fof.fe.up.pt/sinf/sinf_grupo_ip.csv

A .csv file named “sinf_grupo_ip.csv” will be received. Open this file and locate the group identification previously wrote to “grupo_sinf.txt”, the IP address of your Raspberry Pi should be associated with it, as illustrated in Figure 7.

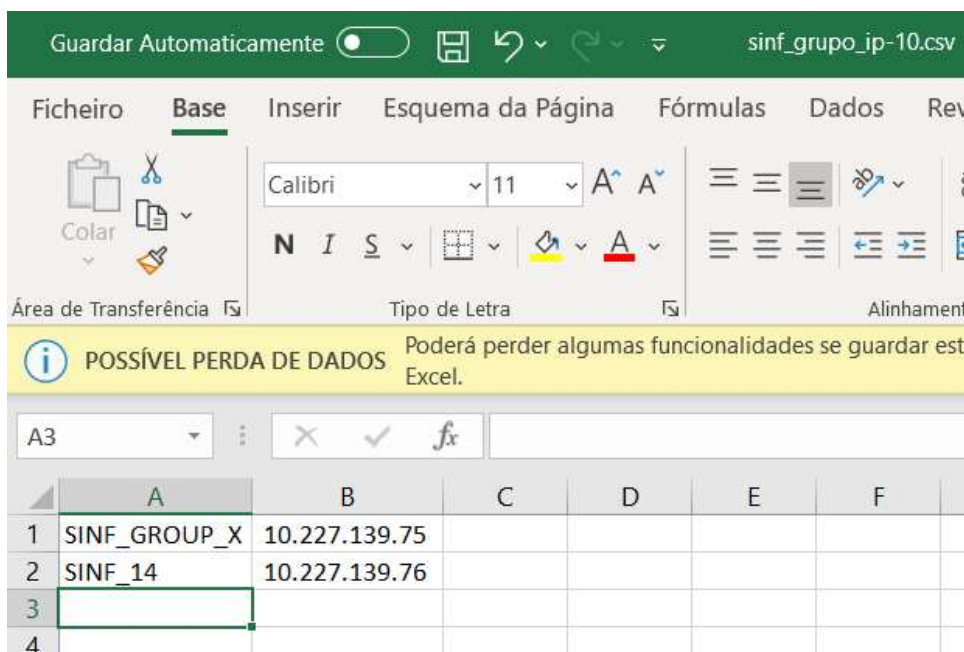


Figure 7 - CSV file mapping groups to IPs.

2.2.2. Configure VNC Viewer on Windows

After the establishing the connection to Eduroam and getting to know the IP address, we need to install the VNC Viewer, which is the application required to remotely access the Raspberry Pi. Hence, please download and install the VNC Viewer program from the following Website:

<https://www.realvnc.com/pt/connect/download/viewer/>

Open the VNC Viewer program and introduce the previously acquired IP as seen in Figure 8, and then click “Enter”.

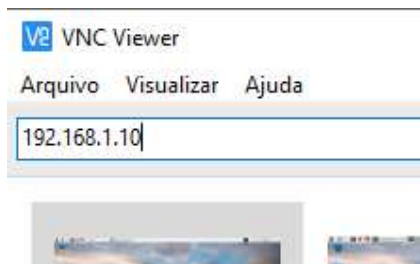


Figure 8 - IP to access Raspberry PI

An authentication form will open, and please insert the user “pi” and password “raspberrypi” (Figure 9).

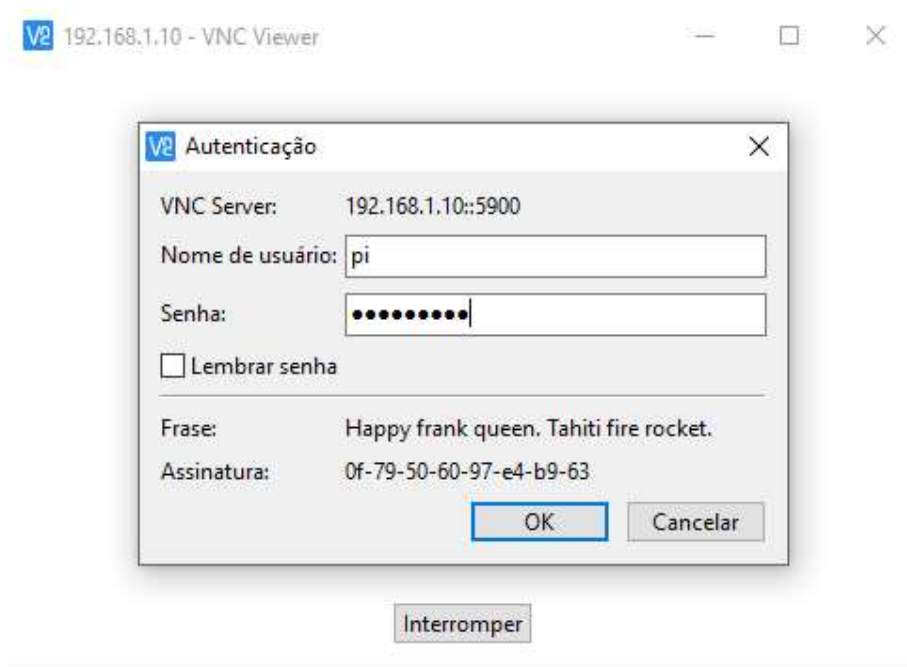


Figure 9 - VNC Authentication Form

Finally, a remote desktop window will open and you can start using your Raspberry Pi.

2.2.3. Configure VNC Viewer on Ubuntu

On Ubuntu OS, you need first to download the installation file from the official VNC website using the following URL:

<https://www.realvnc.com/en/connect/download/viewer/linux/>

On the website, please select the version “Standalone x64” for 64bit versions or “Standalone x86” for 32bit versions according to your personal PC. Then, open the Terminal and execute the following command to make the downloaded file executable:

```
chmod +x VNC-Viewer-6.19.107-Linux-x64
```

And then execute the VNC Viewer using the following command:

```
./VNC-Viewer-6.19.107-Linux-x64
```

Enter the previously acquired IP in the text box. Finally, a remote desktop window will open and you can start using your Raspberry Pi.

2.2.4. Change the default password

After you connect to the Raspberry Pi it is imperative that you change the default password. Use the `passwd` command as instructed in the Raspberry Pi documentation [here](#).

3. Bidirectional Byte Stream

As previously explained, the “*socat*” command utility will be used to transfer byte streams among applications. This way, “*socat*” creates two virtual channels that forward information from each other. This means that everything that will be written in one channel will be available to read the other created channel. Hence, by developing an application that is able to write on a channel, and other that is able to read from the other channel, both can exchange information in a simple way. To create two dedicated channels to transfer byte streams, the following command should be executed in the “Terminal”:

```
socat -d -d pty,raw,echo=0 pty,raw,echo=0
```

Figure 10 shows the messages that will appear in the Terminal after the previous command is executed.

```
pi@raspberrypi:~ $ socat -d -d pty,raw,echo=0 pty,raw,echo=0
2018/02/12 10:39:10 socat[15583] N PTY is /dev/pts/2
2018/02/12 10:39:10 socat[15583] N PTY is /dev/pts/3
2018/02/12 10:39:10 socat[15583] N starting data transfer loop with FDs [5,5] and [7,7]
```

Figure 10 - Execution of the “*socat*” command to create two different channels of communication

In this particular case, two different channels were created, namely “*/dev/pts/2*” and “*/dev/pts/3*”, and both can be used to write and read information. Please note that a different number for each channel might appear in your environment. As an example, if writing in “*/dev/pts/2*” one should be read from “*/dev/pts/3*”, and vice-versa. The “*socat*” command execution can be stopped by closing the Terminal or by pressing “Ctrl+C”.

To test if the channels are created correctly, please open a new “Terminal” and execute the following command:

```
cat < /dev/pts/2
```

This will display on the Terminal everything that is available to read on “*/dev/pts/2*”, which corresponds to everything that is written in “*/dev/pts/3*”. One should notice that this command is waiting for any message being available to read, and then print it on the Terminal. This way, let’s execute a command in a different Terminal to write a “Hello, World” message to “*/dev/pts/3*”:

```
echo “Hello, World” > /dev/pts/3
```

Now you should see a “Hello, World” message in the Terminal that is reading from “*/dev/pts/2*”. Based on this, information can be exchanged between two applications by writing data to “*/dev/pts/3*” and reading from “*/dev/pts/2*”.

NOTE: Please do not close the Terminal in which you’ve executed the “*socat -d -d pty,raw,echo=0 pty,raw,echo=0*”. This will stop the forwarding between the channels and the byte stream transfer will stop.

4. TelosB CM5000 Simulator

The CM5000 TelosB sensors (Figure 11) is IEEE 802.15.4 compliant wireless sensor node based on the original open-source TelosB / Tmote Sky platform design developed and published by the University of California, Berkeley. For this course unit, the sensors used are cable to measure temperature, relative humidity, light and current from a home appliance. For this project we will be using a simulator of such device that will produce the same type of communication this sensor node produces. To a sensor node we usually call *mote* and we will be using this term when referring to a set of temperature, humidity, light and current sensors.



Figure 11 - Telos B CM5000

So, the first step to get familiar with these motes is the type of information they produce and you will need to deal with. They produce a simple and well-defined byte stream that needs to be decoded according to the manufacturer specifications

As an example, please find a message produced by a real Telos B that needs to be interpreted by the C Application that you need to develop:

7E 45 00 FF FF 00 05 0A 3F 0C 00 00 00 E1 00 00 19 8E 7F 0D 48 81 7E

This message is a set of hexadecimal values with a defined structure and each of the hexadecimal plays a specific role in this message. These roles are presented in Figure 12 and based on these, you should build a parser to correctly fetch the required hexadecimal to convert these values into real sensor measurements.

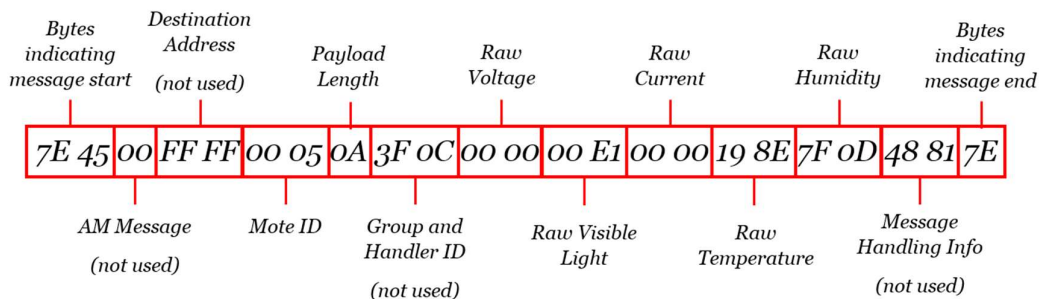


Figure 12 – Altered version of Telos B CM5000 Message Structure

As can be seen, every message starts with a specific identification (*7E 45*) so you can correctly know when to start parsing the message. Afterwards there are the *AM Message* and *Destination Address* bytes that will not be used in the context of this course. Then, the *Mote ID* bytes uniquely identify the mote to be used in your project. For example, if 5 motes are used, they are incrementally uniquely identified from the ID *00 01* to the *00 05*. This allows you to physically distribute the motes and distinguish between them in order to apply the correct actuation instructions in the correct apartment room. After this, the *0A* indicates the length of the payload, which is the section of the message where the sensor measurements are. Then, there's the *Group and Handler ID* that will also not be used in the context of this course. The payload is the set of bytes that come next, and is composed by 5 different sections, with two bytes each: 1) Voltage (battery level); 2) Visible Light; 3) Current; 4) Temperature; 5) Humidity. These are the bytes that should be used to calculate the real sensor measurements to be used in your project. Finally, we have two more bytes for the *Message Handling* which will not be used, and the *Message End* of *7E* that is always fixed.

As previously explained, the bytes from the mote message (payload) are not the final values that represent the real sensors measurements. In order to calculate those, there are a set of equations that should be applied. The next 5 sections will be dedicated to explaining how to calculate these real sensor measurements.

Voltage

Voltage = value/4096 × Vref × 2, where Vref = 1.5V

E.g. If Raw Value = 0x0FCC = 4044 (in decimal)

Voltage = (4044 / 4096) × 1.5 × 2 = **2.96 V**

Temperature

Temperature = -39.6 + 0.01 × SO_T

E.g. If Raw Value SO_T = 0x1D3F = 7487

T = -39.6 + 0.01 × 7487 = **35.27 C**

Relative Humidity

RH_{linear} = -2.0468 + 0.0367 · SO_{RH} + -1.5955 · 10⁻⁶ · SO_{RH}²

E.g. If Raw Value SO_{RH} = 0x0227 = 551

$$RH_{\text{linear}} = -2.0468 + 0.0367 \cdot 551 - 1.5955 \cdot 10^{-6} \cdot 551^2 = \mathbf{17.69\%}$$

Humidity compensated by Temperature

$$RH_{\text{true}} = (T^{\circ}\text{C} - 25) \cdot (0.01 + 0.00008 \cdot SO_{\text{RH}}) + RH_{\text{linear}}$$

E.g. E.g. If Raw Value $SO_{\text{RH}} = 0x0227 = 551$

$$RH_{\text{true}} = (35.27 - 25) \cdot (0.01 + 0.00008 \cdot 551) + 17.69 = \mathbf{18.25\%}$$

Visible Light

$$S_{1087} \text{ lx} = 0.625 \times 10^6 \times (VL/4096) \times 1.5 / 100,000 \times 1000$$

E.g. If Raw Value $VL = 0x00BD = 189$

$$S_{1087} \text{ lx} = 0.625 \times 10^6 \times (189/4096) \times 1.5 / 100,000 \times 1000 = \mathbf{432.59 \text{ lux}}$$

Current

$$S_{1087-01} \text{ lx} = 0.769 \times 10^5 \times ((C/4096) \times 1.5 / 100,000) \times 1000$$

E.g. If Raw Value $C = 0x001E = 30$

$$S_{1087-01} \text{ lx} = 0.769 \times 10^5 \times ((30/4096) \times 1.5 / 100,000) \times 1000 = \mathbf{8.44 \text{ A}}$$

Execute the MsgCreator Application

In order to run the MsgCreator Application, first we need to configure the simulator according to the intended scenario to be explored in the course project. This configuration is present in a file named *MsgCreatorConf.txt* and here is presented an example of the configurations required. **Please note that all the files required to run the MsgCreator Application are in path */home/pi/Desktop/SINF*.**

```
-n 5 -l 100 -f 1 -c 1 -s [1,3,4] -d [['U',0.0,500.0,30.0] , ['C',20.0,35.0,0.5] ,
['L',1.0,5.0,1]] -i 1
```

Please find a description for each of the application parameters:

- -n 5 → number of notes to be simulated, where, in this example, the unique IDs will range from *00 01* to *00 05*;

- -l 100 → number of measurements that each of the motes should produce during time. If 100, the simulator will produce 100 messages to be interpreted for each mote and then stop;
- -f 1 → Frequency in Hz in which the messages should be produced. If 1, a message will be produced every second per mote, if 2, 2 messages will be produced every second. Please note that the minimum value for this parameter is 1;
- -c 1 → Enable the graphical user interface to show the measurements provided by the MsgCreator simulator. If the graphical user interface is not necessary, please use the value 0. Figure 13 depicts the interface for showing the measurements in real time for the presented scenario;

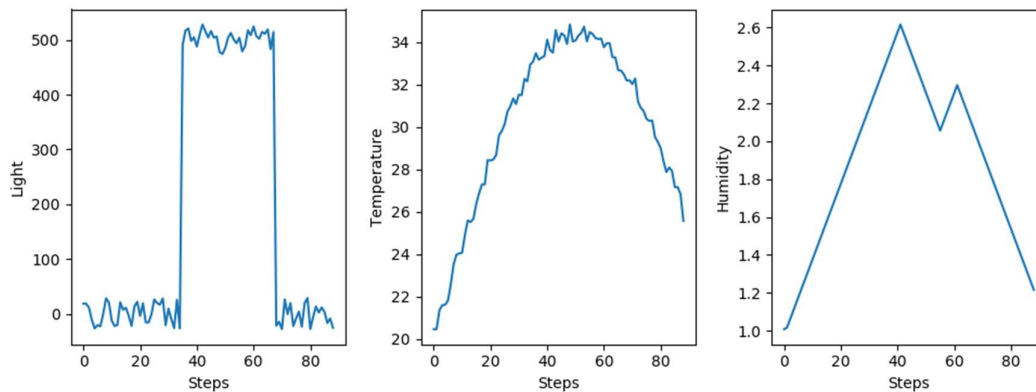


Figure 13 - Graphical User Interface to show the produced measurements in real-time.

- -s [1,3,4] → sensors that should be producing measurements. This is a list of integer values ranging from 0 to 4. Here is the list of sensors available:
 - 0: vref 0..5V;
 - 1: photo 0..3500 lux;
 - 2: current 0..50 A;
 - 3: temperature 0..40 C;
 - 4: humidity 0..100.

Please note that there's a minimum and maximum values for each of the sensors;

- -d [['U',0.0,500.0,30.0],['C',20.0,35.0,0.5],['L',1.0,5.0,1]] → This parameter is a list of lists and indicates the behavior to be used by each of the sensors. There are 3 types of possible behaviors to be used, and the first element of each inner list indicates the behavior to be used. These behaviors are indicated by the letter 'U', 'C' and 'L', meaning Uniform, Curve (or Gaussian) and Linear, correspondingly. For the behaviors 'U' and 'C', the structure of the inner list is as follows: *[type, min, max, var]*. The *type* value indicates which behavior should be adopted by the sensor; the *min* and *max* values are the minimum and maximum values that both behaviors can assume, and *var* is the amount of variation that each measurement have from the true value,

representing some noise. For behavior 'L' the structure of the inner list is $[type, min, max, slope]$. Apart from slope, all other parameters are the same. The slope parameter can be either 1 or -1, depending on the increasing or decreasing linear behaviour. If 1 is specified, the behaviour start at min and finishes at max. If -1 is specified, the behaviour start at max and finishes at min.

Taking the following parameters as an example `'-s [1,3,4] -d [['U',0.0,500.0,30.0],['C',20.0,35.0,0.5],['L',1.0,5.0,1]]'`, for the first list that corresponds to sensor 1 (Photo) (`['U',0.0,500.0,30.0]`), a Uniform behavior will be used starting with mean 0.0 and variation of 30.0, and changing the mean to 500.0 at 1/3 of the specified number of messages, and at 2/3 of the specified messages the mean value goes back to 0.0. For example, if one specifies that the simulator should produce 100 messages, in this case, the first 33 messages will be around 0.0; from 33 and 66 messages the measurement will be around 500.0; and from message 66 to 100 will be again around 0.0. This is intended to drastically vary the measurements produced, simulating turning on and off a room lighting. An example can be seen in the following Figure.

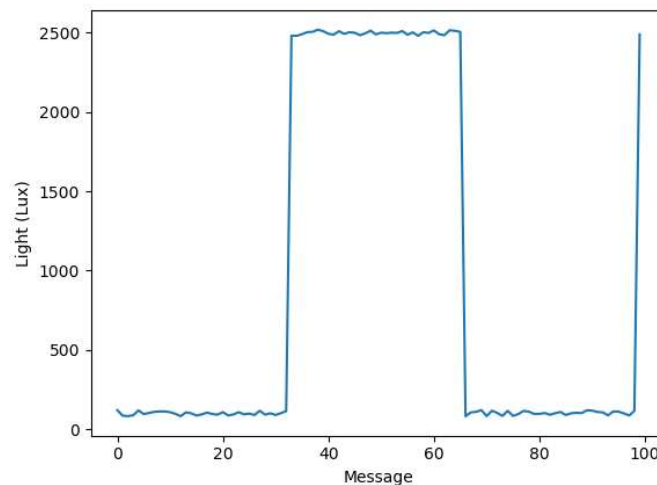


Figure 14 - Uniform behavior

For the second inner list (`['C',20.0,35.0,0.5]`) that corresponds to sensor 3 (Temperature), a curve function is used to produce a smoother variation of sensor values. Figure 15 presents a variation during the specified time, with the minimum value at the beginning and at the end of the behavior, reaching its maximum in the middle of number of samples to be produced.

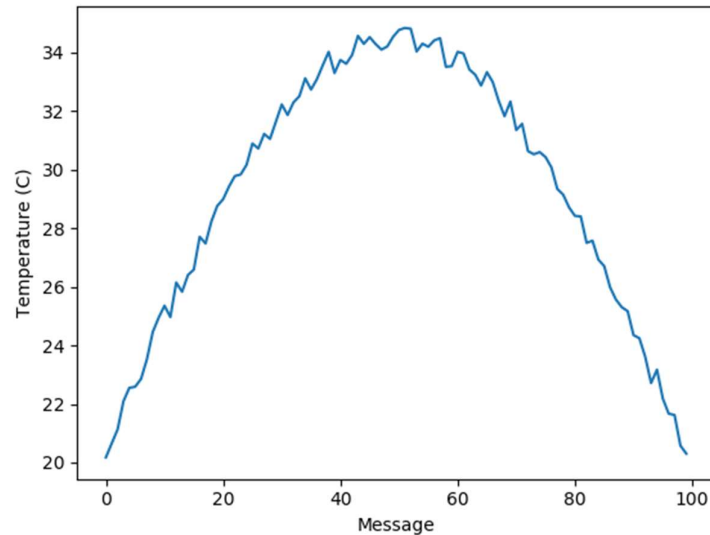


Figure 15 - Smooth variation of sensor measurements

In this case, the minimum value starts with mean 20 with variation of 0.5 and reaches its maximum value of mean 35 at message 50. Then, it drops smoothly again to mean 20 as it approaches message 100. Ultimately, the last inner list (['L',1.0,5.0,1]) corresponds to sensor 4 (humidity) and is the Linear behaviour that varies from 1 to 5 in an increasing way. This is a simple behaviour that uses a linear equation between a min and max where the corresponding number of measurements are produced. In this case 100 messages for humidity sensor that increase from 1 to 5, as can be seen in the following Figure.

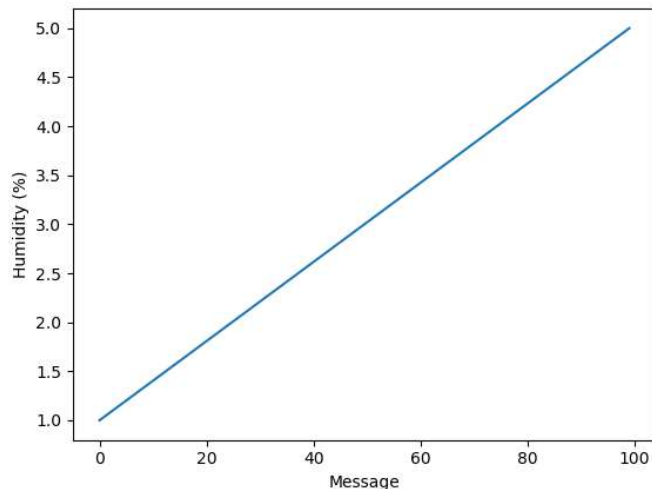


Figure 16 - Linear behaviour - Positive slope

Alternatively, if a decreasing behaviour (negative slope) is preferred, please use -1 in the parameter *slope* of the inner list. The resulting behaviour is presented in the following Figure.

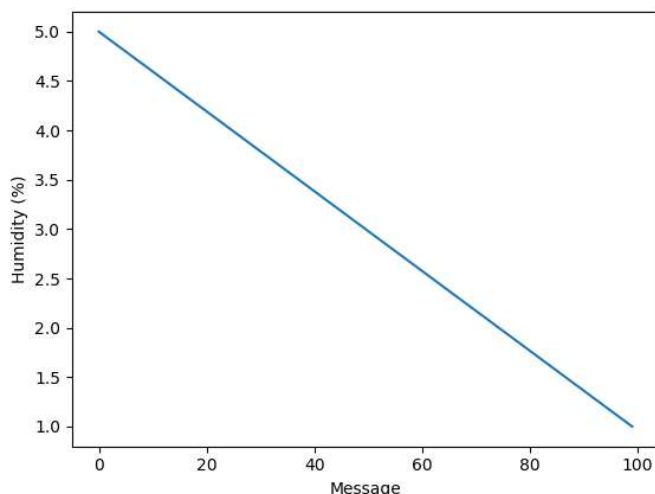


Figure 17 - Linear behaviour - Negative slope

- `-i 3` → This parameter defines the ID of the first mote to use. If 5 different motes are used (`-n 5`) in the simulator, in this case the IDs will be 3, 4, 5, 6 and 7. If only one mote is used, the mote will have the ID 3.

Finally, the MsgCreator application can be executed. For that, please open the Terminal and run the following command:

`./MsgCreator.py`

This will produce a set of messages as presented in Figure 18.

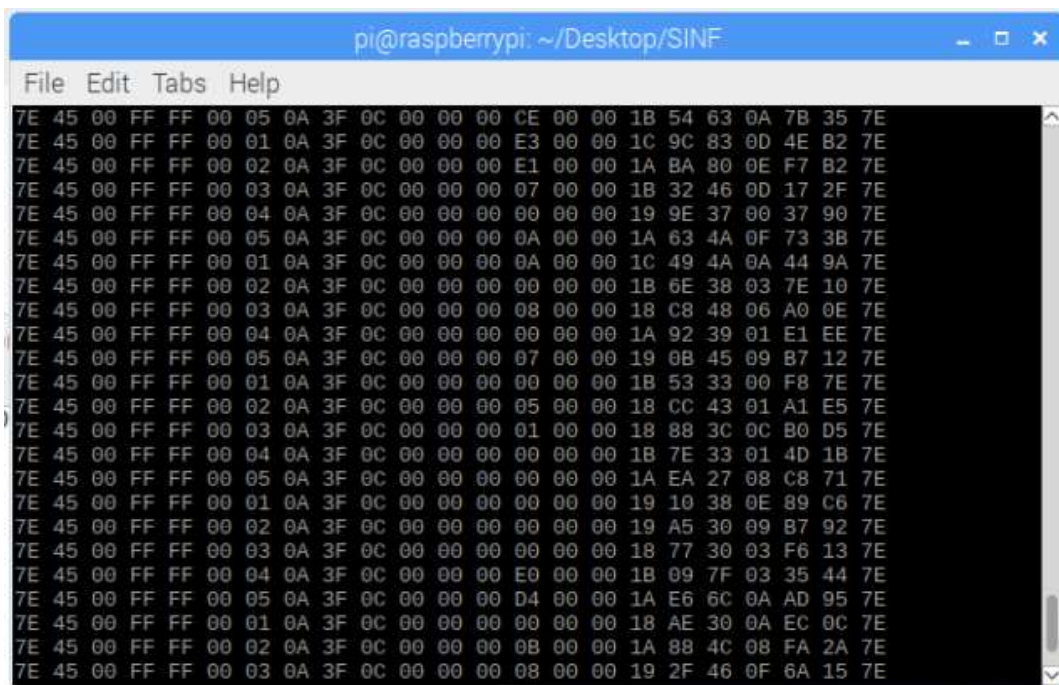


Figure 18 - Messages generated by the MsgCreator Application

In the case of your course project, you should write this information into the created channels by the “*socat*” command. If we have created two channels named “*/dev/pts/1*” and “*/dev/pts/2*” one should write in “*/dev/pts/1*” the message produced as follows:

```
./MsgCreator.py > /dev/pts/1
```

Hence, the C Application that you need to develop should read from channel “*/dev/pts/2*”.

Different behaviours per mote

Although the MsgCreator simulator enables the use of multiple motes, the behaviour of each mote is inherently the same. This means that the simulated conditions in different rooms in your SINF scenario will be the same, and this might not be feasible for your project.

Hence, there is a possible solution to implement a dedicated behaviour per mote. In the previous explanation, you were asked to execute the simulator to emulate the behaviour of each mote. In this proposed setting, you are asked to use multiple simulators, one per dedicated mote behaviour. Based on this, the easiest way to setup your environment to execute multiple simulators is to have multiple folders with the MsgCreator.py and MsgCreatorConf.txt, one per dedicated behaviour. This way, you can set the parameter ‘-n’ to 1 in each simulator and specify different behaviours in the different MsgCreatorConf.txt files (one per folder). After this definition, you can execute the simulators (one per terminal) and send the data to the same buffer pair.

As an example, imagine you have two simulators named MsgCreator.py and MsgCreator2.py in different folders, each with its own dedicated MsgCreatorConf.txt. According to the above explanation, and assuming you have the buffer pairs */dev/pts/1* and */dev/pts2* available, you should execute the simulators as such:

```
python3 MsgCreator.py > /dev/pts/1
```

```
python3 MsgCreator2.py > /dev/pts/1
```

And your C application the following way:

```
./app < /dev/pts/2
```

As you can notice, the way you should read in your C application is the same, either you choose a dedicated mote behaviour or the standard use of only one simulator. However, if you already tried the above solution, you noticed that if you use multiple simulators with only one mote per simulator, the IDs of the motes are 01, which does not allow you to distinguish between motes. To this intent, you should make use of the parameter ‘-i’. This parameter is ‘-i’ and defines the ID of the mote. Hence, the following MsgCreatorConf.txt setting:


```
-n 1 -s [1,3,4] -d [['U',100.0,2500.0,20.0],['C',2.0,20.0,0.5],['C',5.0,30.0,0.5]] -f 1  
-l 50 -c 1 -i 5
```

will produce the corresponding simulator behaviour:

```
7E 45 00 FF FF 00 05 0A 3F 0C 00 00 00 29 00 00 10 67 00 AF 3A 8E 7E  
7E 45 00 FF FF 00 05 0A 3F 0C 00 00 00 32 00 00 10 D6 00 D9 BE 4A 7E  
7E 45 00 FF FF 00 05 0A 3F 0C 00 00 00 31 00 00 11 5F 01 0E 7C 3E 7E  
7E 45 00 FF FF 00 05 0A 3F 0C 00 00 00 2F 00 00 11 E5 01 35 60 F7 7E
```

...

This way, you can specify the ID of your notes in different simulators. Additionally, if you use multiple notes in the same simulator and define the '-i' parameter, the IDs of the notes will start in the specified number, and incrementally attribute the IDs. As an example, the following MsgCreatorConf.txt setting

```
-n 5 -s [1,3,4] -d [['U',0.0,500.0,30.0],['C',20.0,35.0,0.5],['L',1.0,5.0,1]] -f 1 -l 50  
-c 1 -i 5
```

will produce the corresponding simulator behaviour:

```
7E 45 00 FF FF 00 05 0A 3F 0C 00 00 00 28 00 00 10 74 00 A1 7B 34 7E  
7E 45 00 FF FF 00 06 0A 3F 0C 00 00 00 25 00 00 10 B3 00 BA 94 95 7E  
7E 45 00 FF FF 00 07 0A 3F 0C 00 00 00 34 00 00 10 C1 00 BD ED 9F 7E  
7E 45 00 FF FF 00 08 0A 3F 0C 00 00 00 28 00 00 10 92 00 BB 46 AD 7E  
7E 45 00 FF FF 00 09 0A 3F 0C 00 00 00 2F 00 00 10 B2 00 C8 3D B9 7E  
7E 45 00 FF FF 00 05 0A 3F 0C 00 00 00 2F 00 00 10 CB 00 CC 1D DF 7E  
7E 45 00 FF FF 00 06 0A 3F 0C 00 00 00 25 00 00 10 DC 00 F5 84 FC 7E  
7E 45 00 FF FF 00 07 0A 3F 0C 00 00 00 2E 00 00 11 2F 00 ED A6 54 7E  
7E 45 00 FF FF 00 08 0A 3F 0C 00 00 00 28 00 00 11 1D 00 F4 3A 81 7E  
7E 45 00 FF FF 00 09 0A 3F 0C 00 00 00 2F 00 00 11 5C 00 FB 46 40 7E  
7E 45 00 FF FF 00 05 0A 3F 0C 00 00 00 2C 00 00 11 63 00 FF 4C 1C 7E
```

...

Dynamically change the sensor behaviour

One of the interesting ideas of using sensors in home automation scenarios is to have some sort of way to influence the environment using actuators. Thus, the

connection between actuation -> environment -> sensors is key for a quality project. Although the presented behaviours are fixed during time, the Linear behaviour is the only one that allows for this flexibility.

The MsgCreator simulator was initially built to frequently read the MsgCreatorConf.txt file, and not only a single time in the beginning of execution. This way, you can alter this configuration file during time if you are using the Linear behaviour. If initially the behaviour is increasing from 1 to 5, and after a while you want to change the behaviour to decreasing, you can simply change from 1 to -1 in the corresponding parameter. The MsgCreator simulator will read the file and make the necessary adjustments. The resulting behaviour is now decreasing from measurement it was with the same increasing step previously calculated.

Finally, the idea is for your application to also change this file in order to change the sensor behaviour during time and simulating changes in the environment, according to the actuators you'll be using. For example, if a heater is used, then a Linear increasing behaviour can be used in the temperature sensor; If an Air Conditioner is used, then a Linear decreasing behaviour can be used in temperature sensor; If a bathroom window can be controlled, a Linear increasing and decreasing behaviours can be used to simulate the inlet and outlet of fresh air; etc.

5. RGBMatrix Application

In order to display your sensor and actuation information, the RGBMatrix Application should be used. As its name indicates, it is solely an application that graphically shows a RGB Square Matrix with a configurable number of cells and dimension. In the course project, it is advised to have at least one cell per sensor used and also one cell per actuator. If you use 3 sensors per mote, use 2 motes for the project, together with 3 actuators (e.g. Heater, Blinds, Air Conditioner), you will need at least a 3x3 matrix to display all the elements. An example of the graphical representation of the RGBMatrix is depicted in Figure 19.

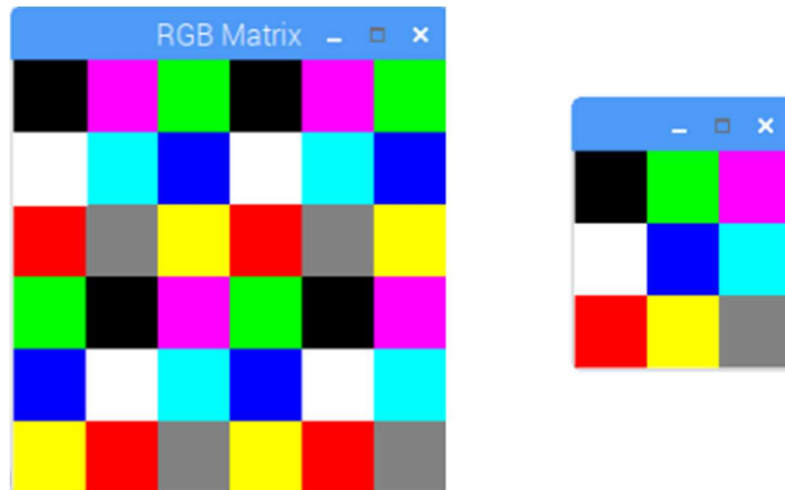


Figure 19 - RGB Matrix Application example with a 6x6 grid (left) and a 3x3 grid (right)

Please note that all the files required to run the RGBMatrix Application are in path `/home/pi/Desktop/SINF`.

In order to display this matrix, the input for the application can be either a text file or a channel created using the “socat” command. The format used as input for the RGBMatrix Application is as follows:

`[[0,0,0],[0,255,255],[255,150,255],[0,0,45]]`

In fact, this format is simply a list of lists, where each list is the regular RGB representation, being the first element Red, the second Green and the third Blue. In the presented example (Figure 20), the first list `[0,0,0]` is the Black color, the list `[0,255,255]` is the Pink color, the list `[255,150,255]` is the Light Blue and last list `[0,0,45]` is the Dark Blue. Hence, the first two lists are the first row of the matrix, and the last two lists are the last row.



Figure 20 - 2x2 matrix representation using the RGBMatrix Application

To configure the number and size of matrix cells, one should use the *RGBMatrixConf.txt* where we can specify these parameters as follows:

`-a 2 -b 50`

where `-a 2` is the number of side cells of the square matrix, and the `-b 50` is the number of pixels per cell. High values of `-a` will produce more cells per matrix, and high values of `-b` will produce larger cells. Before running the RGBMatrix Application, beware to correctly configure this file.

In order to execute the application, as previously explained, you need to use as input the channels created using the “*socat*” command. This means that if we have “*/dev/pts/1*” and “*/dev/pts/2*” channels available for communication, the list of lists should be written to “*/dev/pts/1*” and the RGBMatrix Application should read from “*/dev/pts/2*”, or vice-versa. For this case, the following command should be used:

```
./RGBMatrix.py < /dev/pts/2
```

In order to test this execution, please use the simple application called “*write_matrix.py*” available in the same SINF directory that can be executed as follows:

```
./write_matrix.py > /dev/pts/1
```

This application changes the color of the first cell in a 2x2 matrix every second, during the period of 10 seconds. Obviously, this will only work if the RGBMatrix Application is configured to display a 2x2 matrix.

6. Objectives for Sprint #1

All the previous sections describe how to configure the Raspberry Pi environment so you can develop your project in the proposed embedded system. Hence, we need to define now the specific objectives for your sprint. These will be evaluated at the end of the Sprint and will determine the grade of the group. These objectives are the following:

1. Read the Telos B Simulator messages, and translate them according to the equations defined in Section 4;
2. Implement simple rules that will change the state of your actuators (E.g. Turn heat on, close window, open blinds, etc.);
3. Write all the actuators’ state to a file that the RGBMatrix Application will read and graphically present. Additionally, you can also include the representation of the sensors in this matrix representation.
4. Implement a simple controller for Home Automation.
5. Define a sensor configurations file where the association of sensor and actuator to a certain room is specified;
6. Define a sensor rules file where the rules to change the actuators’ state are defined;
7. Implement the points 1), 2) and 3) using Threads;

Next, we will detail each of the points so each group is aware of how to correctly proceed in the development phase.

1. Read from Telos B simulator

According to the message structure presented in Section 4, you should include into your C Application a parser to correctly interpret those messages. However, this is not enough to handle the real measurements from the sensors. Complementarily, you should use the equations presented in the Section 4 to

translate the byte stream into the real values that will change the states of your actuators. Therefore, there are 5 different equations, one for each measurement made by the Telos B mote. However, one of the measurement is the battery level, which is not an environmental measurement, so it not suitable for your application. Additionally, the infrared sensor is an optional sensor that you can use due to the difficulty in finding simple applications for your project. Hence, the main sensors to deal with are the Temperature, Humidity and Light.

2. Implement simple rules

According to the scenarios defined in your project, you should define simple and clear rules that will change the state of the actuators. As an example, a simple rule can be defined as *“If temperature greater than 25°C turn heat off”*. Therefore, you should implement those rules in your C Application that should already use the translated values from the simulator. For this case, you can use the console to check if these rules are well implemented and have some feedback on how your application is behaving.

3. Write to RGBMatrix Application

The next step after you define some simple and clear rules to change your actuators, you need to show these on a graphical application. For that, you will be using the RGBMatrix Application introduced in Section 5 to display the state of your actuators. In this case, you need to write the states into a .txt file according to the defined structure (an RGB list of lists) so the RGBMatrix Application can display these correctly. Moreover, you should define the dimensions of your matrix according to the complexity of your project. Ultimately, as a way to ease the interpretation of your rules and the behavior of your C Application, you can additionally present the sensor measurements in the same matrix as the actuators. This way will be easier to see the sensor variation and the consequence of your rule definition on the actuators state.

This third point closes the loop of your HAS where you already can read sensor measurements from the simulator, interpret those data, define and implement simple and clear rules for your scenarios, and ultimately write actuator instructions that will influence the environment. Despite being a sufficient implementation for an HAS to work, it is not a flexible and scalable application. This means that if you need to change some rule or add / remove sensors, you need to do it by changing your application's code.

Hence, the following 3 points will focus on improving your application, being closer to what a real HAS should look like.

4. Implement a simple controller for Home Automation

One of the key points of Home Automation is to have a system capable to accommodate all the needs from its users, mainly at making them feel comfortable in their homes or other facilities. This way, the idea is to give the user total control of the environmental conditions. In this case, these conditions are Temperature, Humidity, Light and Energy.

Hence, in this goal, you are asked to include in your application the possibility to directly influence the environment with your actuators. Since there is no real / physical environment, the idea is to change the behaviour of sensors according to the actuators' behaviours. This can be achieved by changing the *MsgCreatorConf.txt* file accordingly, as explained in the previous Sections.

To complete this goal, you must implement a rule of the following kind:

ROOM1: TEMP1=25

For this specific rule, there is no actuator involved, since you should choose the correct actuators that can influence the environment and turning them on and off accordingly. For example, for the previous rule, if temperature is above 25 C°, an AC should be turned on. By turning on the AC you must do two things: 1) Represent this state change in the RGBMatrix and 2) change the behaviour of the temperature sensor accordingly (decreasing). However, by simply turning the AC off will not get you to the target temperature. According to the Linear behaviour, the temperature will continue to drop. Hence, at a certain point, you should turn on the heater to invert this situation. By turning on the heater, you should change the behaviour to increasing. Then, the temperature will continue to increase, and you get back to the point where the AC is required.

Is this simple controller that you should implement, by having a specific implementation to orchestrate these two actuators to maintain the temperature at a certain level. This temperature case is just an example, as different cases might be used for humidity, light and energy efficiency.

5. Sensor Configurations

The first step towards the improvement of your application is related with the association of each sensor and actuator to a certain room. This allows you to have specific rules per room, contrary of having specific rules per sensor as you should have implemented in point 3). Using this approach, you will be able to define more robust and oriented rules such as *"If temperature in the Master Room is greater than 25°C turn heat off"*.

This way, you should write these associations in a *"SensorConfigurations.txt"* file that will be parsed and interpreted by your C Application. For this case, the format that you should use is as follows:

<room_name>:<sensor_name>[,<sensor_name>]:<actuator>[,actuator]

As an example, we associate a set of 10 sensors from 4 different motes to 3 and 5 actuators to different house rooms. In this case, ROOM1 will have on mote and

will be using all 3 sensors together with 2 actuators such as a heater and light, the KITCHEN will also have one mote but only one light sensor and consequently only a light actuator, and finally the LIVING_ROOM that will have 2 motes, with all 3 sensors each with one heater and light as actuators. In this case, each sensor has a number that corresponds to the mote that will be producing data. Hence, TEMP1 is the temperature value from Mote ID 1, and LIGHT4 is the light value from Mote ID 4.

ROOM1:TEMP1,HUM1,LIGHT1:HEAT_ROOM1,LIGHT_ROOM1

KITCHEN:LIGHT2:LIGHT_KITCHEN

LIVING_ROOM:TEMP3,HUM3,LIGHT3,TEMP4,HUM4,LIGHT4:HEAT_LR, LIGHT_LR

This approach allows to easily add more sensors to the rooms that you are already monitoring, if required, and also start monitoring new rooms. If these associations were only defined in your C Application code, you need to change the code whenever any change is required. Thus, this approach increases the flexibility of your application.

6. Sensor Rules

Based on the room-sensor-actuator association made in point 4), we should now be able to define rules per room. This way, you need to create a different file named “*SensorRules.txt*” that will define how the actuators’ state should change by writing explicitly those rules. Hence, your C Application should read this file, interpret these rules and execute them. The format that should be used is as follows:

```
<room_name>: <sensor_name><condition><value> [AND / OR
<sensor_name><condition><value>] <actuator>:<state>[,<actuator>:<state>]
```

Please note that the use of “AND” is not mandatory and should only be used when the change of an actuator’s state depends on more than 1 sensor.

As an example, we can define a rule for ROOM1 as follows:

ROOM1: TEMP1>25 HEAT_ROOM1:OFF

An example of a rule that depends on more than 1 sensor is also presented:

ROOM1: TEMP1>25 AND LIGHT1>3000 HEAT_ROOM1:OFF,LIGHT_ROOM1:OFF

This approach is very advantageous in a real application of a HAS when you should change the rules on demand. Imagine the user wants to change the threshold value that turns on the master room heat (usually is different from winter to summer). If this approach does not exist, one should change the code application, which is not something easy to do and mainly not desired once the application is already in the customers house. Thus, this approach increases drastically the flexibility of your application.

7. Implement 1), 2) and 3) using Threads

This last point is mainly concerned about increasing the scalability of your application. The implementation that you have made until this point is basically a pipeline of instructions, where the parsing and interpretation of sensor measurements, rule execution based on these values, and change the state of actuators is made one after another, and the parsing of new sensor values only starts when the change of actuators' state is complete (full execution of the application). However, if the sensor values are streaming in a high sampling rate, e.g. 5 Hz (5 measurements per second) and if you are using 5 motes, this means that 25 messages will be produced every second, and you have $1000/25 = 40$ milliseconds to finish all the instructions in your pipeline. Based on this, there will be a maximum (that in this case is a very low value) of motes you can add to your HAS before you start losing information. This means that your solution is not scalable. In Figure 21 this approach is presented, where the dashed line represents that only after the Port Writing occurs, the program is ready to parse new data.

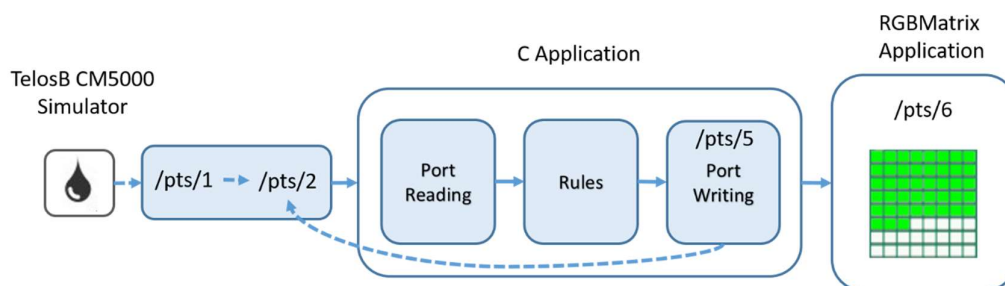


Figure 21 - Pipeline of code execution (sequential)

Hence, you should implement a parallel execution of your program by using Threads. For that, you should implement one Thread for each of the points 1) 2) and 3) (points 4) and 5) implicitly depend on the first 3 points), namely the parsing and interpretation (point 1), rule execution (point 2) and writing the actuators' new state (point 3). This will allow to first have a parallel execution of these blocks, and second, not wait until the whole code executes, but wait only for the block of point 1) executes. This approach is presented in Figure 22, where the dashed lines represent the dependencies of each Thread (Port Reading, Rules, Port Writing) and they can execute concurrently.

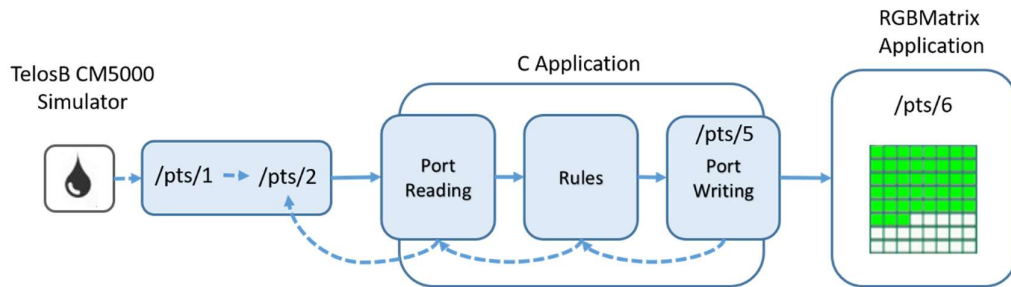


Figure 22 - Thread implementation of your course project

Based on this, the presented approach increases the scalability of your application.