Projects and Laboratory on Communication Systems - Project Report

Paolo Calao (sxxxxx), Samuele Yves Cerini (s256813), Federico Pozzana (sxxxxx)

July 10, 2019

Contents

1 The project

1.1 Introduction

This final project consists of a Smart-Home Thermostat, realized using a given *Raspberry Pi3* together a capacitive touch screen. Additional components were added by ourselves in order to positevely respond to the given and mandatory requirements. While developing the project we have paid particular attention to the organizational phases, trying to think as much as possible as final customers of our product, while delivering something that we hope can be considered something more than a mere *Proof-Of-Concept*.

1.2 General organization

Since the group is composed of 3 people, we divided the project in 3 main components that we will call here, for the sake of simplicity, ***Backend SW Logic***, ***Front-End SW Logic and User Experience*** and finally ***Hardware and Communication Logic***. Finally, these three main components of the project were merged together during the final phases of the development to build and test the final product.

1.3 A Smart-Home Thermostat

In this section, we will discuss both mandatory and non-mandatory requirements we implemented for this project, while illustrating our decisions and approaches. We intended this Smart-Home Thermostat as a product that targets relatively new buildings, that have been designed with both heating and air conditioning systems. Our

system is composed by three main Hardware components: the *Central Unit*, composed by a *Raspberry Pi3* and its related capacitive touch-screen, a multiple number of *sensor modules* to retrieve temperature data from each room (built using *ESP32 Microcontrollers*) and finally multiple *actuator modules* that can control the heating system of the building.

As an example and to keep things in perspective, our home is ideally made of multiple rooms (4 in our case) where each room can be equipped with a stand-alone air conditioner. The heating system for each room is handled by a centralized *actuator module* that groups and drives all the valves required by all the rooms. Each room can have more than one valve serving it, depending on the area to be served by the system, of course. Given the aforementioned scenario, our product has been designed to handle more than one room, where each room can be equipped by one *sensor module* (to measure the ambient temperature of the room),as long as an Infrared system to drive a stand-alone air conditioner, if any.

The *Central Unit* has to be intended as the every-day interface to the system for the customer. It permits to easily add/remove rooms to be controlled, as well as add/remove *sensor modules* and *actuator modules*. We think the system can be easily installed by a normal user with some DIY skills. No big experience is required for using it daily.

2 Hardware and Communication Logic

3 Backend SW Logic

4 Front-End SW Logic and User Experience

4.1 Introduction

In this section we explain what are the solutions implemented in order to keep the system as simple as possible to the end user. At the same time, we will explore the solutions adopted for the Graphical User Interface. In general, we tried to implement a system without any special complications to the final user: we wanted to keep it as simple as possible while at the same time delivering all the functionalities a user is expecting from a similar system. We truly believe that a smart-thermostat should not be overly-complicated or too much fancy: the functionalities have to be the real and most important point. We truly believe that no user will pay for a fancy smart-thermostat only to stare at his screen all the time. Following this idea, the GUI we implemented tries to complete all the functionalities with the minimum number of interactions from the user. As an example, all the functionalities are at maximum of 3 consecutive windows from the principal one. No gestures have been implmented, just single taps to keep the user input as straight as possible.

4.2 GUI framework and languages used

Technically speaking, we decided to implement the GUI using the *QT framework* and libraries since the first moment, but due to the limited time available for development, we chose the *PyQT5* version over the original *C++* version. Although *Python3* is not the best language in terms of speed and overall efficiency (due to its intepreted nature and the presence of a garbage collector), we thought that the tradeoff between the speed and the development time was worthing, especially considering that the *Raspberry Pi 3* processor can perfectly handle this type of load. For some textual inputs a virtual keyboard is invoked (the initial choice initially fell on the *QTvirtualkeboard* module thanks to its high integration with the QTframework. Sadly due to the dated *Raspbian* repositories and dependencies, we were not able to install it and so we were obliged to use the *OnBoard* virtual keyboard. This unexpected issue cost us an entire week of development on itself).

The most of the GUI interfaces have been roughly designed using *QTDesigner* that returns a QML file (`.ui` extension) that we finally convert to a `.py` Python script file. Finally, all the finest modifications have been directly made manipulating the *Python*code. Structurally speaking, each different window corresponds to a different python class and file. Sometimes some additional and separate files were needed to handle persistence of system-related data.

4.3 Main Window interface

4.4 Local Network connection and related interfaces

4.5 Rooms related interfaces

4.6 Sensor module connection and related interfaces