UFSC 2022 - Embedded Systems Programming in C++

Air-Conditioner Controller with Machine Learning

Ramon de Araujo Borba[1]

[1]UFSC -Universidade Federal de Santa Catarina

March 15, 2022

Abstract

Keywords Embedded Systems, C++, Machine Learning

1 Introduction

Air-conditioning equipment is one of the most used tools for ambient temperature control, widely used in residential and corporate environments, for thermal comfort, as well as in laboratories, where a controlled temperature is needed.

Despite technological advances, this type of equipment have an elevated energy consumption, corresponding to a considerable part of electrical energy expenses. In residential and corporate environments, where thermal comfort is desired, the use of air-conditioners is usually left unchecked, resulting in the equipment remaining powered even without necessity.leading to an avoidable environmental impact and unnecessary expenses.

This project consist of the development of an controlling system for air-conditioners, using embedded systems and AI algorithms, aiming to reduce energy wastage and unnecessary expenses, providing thermal comfort in a economic and environmentally friendly manner.

2 Proposed design

The system was implemented with a focus on Object Oriented Programming and C++ language, using the Windows Subsystem for Linux (WSL)[1] as development environment and a GitHub repository[2] for version control, project management and code sharing.

The development of this project is part of the course EEL7323 - Embedded Systems Programming in C++, the complete project requirements provided professor Eduardo Augusto Bezerra can be found in his website[3] and in the project's GitHub repository's[2] Documentation Folder.

The system is designed to control an air-conditioner and use machine learning (abbreviated to ML from here on) algorithms to reduce energy consumption. The system evaluates environmental data and learns user's characteristics to determine the optimal air-conditioner setting at different operating conditions and adjust it automatically.

The software is designed to facilitate portability to different air-conditioner manufacturers and different operating systems through use of polymorphism and abstract classes, although this project being developed for Linux operating system and Samsung ACs.

The system is divided in three main components, the embedded system, the host computer, and the host smartphone.

The embedded system's macro functionalities are:

- Acquire environmental information through sensors and user input;
- Pass this information as input to the ML algorithm;
- Control the air conditioner based on the output of the ML algorithm;
- Generate a log of all events with detailed information;
- Send log information to host computer and smartphone;

The environmental information acquired by the embedded system will consist of room temperature and relative humidity, through a dedicated sensor. The user inputs will consist of two buttons to signal uncomfortable temperature, one for uncomfortably high temperatures and one for uncomfortably low temperatures.

The controlling of the air-conditioner is done exclusively by the embedded system and the user is no supposed to interfere in the AC control.

The host computer and smartphone's macro functionalities are:

- Establish communication with the embedded system;
  - Computer: through serial port;
  - Smartphone: through Wi-Fi/Bluetooth;
- Receive and store the log information;
- Display this information for the user;
- For the smartphone, serve as an alternative way to provide user input to the embedded system;

All of the system's functionalities are going to be implemented in this instance of the project, with exception of Wi-Fi/Bluetooth communication for the smartphone, despite the hardware chosen (described in section 2.2) having built-in Wi-Fi and Bluetooth capabilities.

The following sections present a more detailed description of each system component's software and technical information.

2.1 Host computer software

The host computer software was compiled using g++[4], and uses the GNU Make[5] build system.

The host computer software must accomplish the following tasks:

- Establish serial communication with the embedded system;
- Store the log data received from the embedded system in a queue;
- Display the available log information in two ways:
  - List all entries between two dates;
  - Display total time the AC was powered;

The implementation for the host computer software was heavily based on the RobotLinux[6] example provided in class. Using an abstract base class to provide an interface with the embedded system, and derived classes implementing the methods for different operating conditions, in this case, a derived class that supports serial communication in a Linux environment.

The embedded system interface class implements the main methods necessary for the requested program functionalities, with functions dedicated to opening and monitoring a serial port, methods for listing the log information from the embedded system, and a queue to store the log information.

The embedded system will periodically send the available log information through a serial port. The serial monitoring method is designed to run on a separate thread, and registers any incoming information in the log queue. There is also a method to send a command to the embedded system to trigger the transmission of available log information per user request.

The queue and its node are both implemented as classes, related by aggregation.

The Queue class has methods for enqueueing and dequeueing nodes, searching and listing information, and a two node pointers as attributes, called "head" and "tail", to determine the beginning and end of the queue.

The Node class attributes are a node pointer to indicate the next node, and the data packet with the log information, with basic setters and getters methods.

The data packet itself is also a class, able to store the raw data received from serial communication, and providing methods to translate this raw data into readable information.

2.2 Embedded system software and hardware

The chosen embedded system hardware was an ESP32 based development kit featuring the ESP32-WROOM-32[7] dual-core microcontroller module, developed by Espressif[8], and an DHT11[9] temperature and relative humidity sensor. It's worth noting that this ESP32 module has built-in WI-Fi and Bluetooth capabilities, despite that, as mentioned in section 2, this functionality is not implemented in this instance of the project.

The embedded system software must accomplish the following tasks:

- Establish serial communication with host computer;
- Periodically read data from DHT11 sensor;
- Acquire user inputs through GPIO buttons;
- Periodically run inference in the ML algorithm with acquired data;
- Adjust air-conditioner setting when necessary;
- Log all evens with detailed information;
- Periodically, or when requested, send log to hosts;

The software for the embedded system was implemented using the Espressif's ESP-IDF[10] official development framework and it's available tools. This framework runs a slightly modified version of FreeRTOS[11] natively, and allows the developer to use all of it's features. The FreeRTOS was modified by Espressif in order to allow it perform in ESP32's dual-core architecture, and manage both cores effectively. Details about the modifications can be found in the ESP-IDF Programming Guide[10].

Due to that, the embedded software was designed with the FreeRTOS workflow in mind, mainly through the use of tasks, to accomplish its goals.

Since most of the ESP-IDF libraries are implemented in C, part of the embedded system software development was crating C++ classes to encapsulate this libraries functionalities. Espressif provides a few examples of how this can be done, which can be found in the ESP-IDF GitHub repository[12].

The example classes provided by Espressif were testes and used where applicable in this project. Mainly the GPIO related classes were used from this example classes, and indicated appropriately.

References

[1] *Windows Subsystem for Linux Documentation.* URL: https://docs.microsoft.com/en-us/windows/wsl/.

[2] Ramon de Araujo Borba. *Project's GitHub Repository.* URL: https://github.com/ramonborba/EEL7323.

[3] Eduardo Augusto Bezerra. *Project Specifications.* URL: https://gse.ufsc.br/bezerra/wp-content/uploads/2022/02/Especificacao_2021_2-v2.pdf.

[4] *GCC Online Documentation.* URL: https://gcc.gnu.org/onlinedocs/.

[5] *GNU Make Documentation.* URL: https://www.gnu.org/software/make/manual/html_node/index.html#SEC_Contents.

[6] Eduardo Augusto Bezerra. *RobotLinux Example.* URL: https://gse.ufsc.br/bezerra/disciplinas/cpp/material/Serial_paralela_mysql/Robot_cpp/.

[7] *ESP32-WROOM-32 Datasheet.* URL: https://github.com/ramonborba/EEL7323/blob/finalproj/FinalProject/Documentation/datasheets/esp32-wroom-32_datasheet_en.pdf.

[8] *About Espressif.* URL: https://www.espressif.com/en/company/about-espressif.

[9] *DHT11 Datasheet.* URL: https://github.com/ramonborba/EEL7323/blob/finalproj/FinalProject/Documentation/datasheets/Datasheet_DHT11.pdf.

[10] *ESP-IDF Programming Guide.* URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html.

[11] *FreeRTOS Website.* URL: https://www.freertos.org/.

[12] *ESP-IDF GitHub Repositiry.* URL: https://github.com/espressif/esp-idf.