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 25, 2022

Abstract

This project consist of the implementation of a air-conditioner controller system that uses a machine learning algorithm to determine the appropriate temperature. The main objective is to reduce energy waste and environmental impact in the use of air-conditioners.

The system is divided in three main components. First component is embedded system, which is responsible for data acquisition and processing. Second is a host computer, responsible for storing a event log sent by the embedded system and displaying this information. Third is a host smartphone, serving as a mean to control the air-conditioner and acquire user input, along with the same responsibilities as the host computer.

The hole system is developed using C++ with an object oriented approach, designed to be easily adaptable to different air-conditioner manufacturers and operating systems.

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 source code for the host computer software can be found at the project's GitHub repository[2].

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 software uses a terminal interface to interact with the user, providing a menu with three options, to list all logged events, to display total power on time of the air-conditioner and to send a command to the embedded system requesting data. This menu simply calls methods from the embedded system interface.

The embedded system interface 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 periodically sends the available log information through a serial port, with this in mind, the host software is designed to constantly monitor the serial port waiting for data, through a thread. The serial monitoring method, designed to run on a separate thread, registers any incoming information in the log queue. The thread that monitors the serial port is started as soon as the program starts and notifies if the serial port was opened successfully. There is also a method to send a command to the embedded system, which consists of a pattern of three of the same, specific, predefined character sent consecutively. This command simply triggers the transmission of available log information in the embedded system.

The queue and its node are both implemented as classes, related by aggregation, providing the basic functionality of a queue data structure.

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.

The class diagram for the host computer is seen in Fig.1.

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 source code for the embedded system software can be found at the project's GitHub repository[2].

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].

Although the ESP-IDF framework's features and APIs being implemented in C, it is natively compatible with C++ development, due to the compilers being built upon GCC, it even provides, although very few and with only basic functionality, example components and APIs implemented in C++ which are used in this application where possible.

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.

The ESP-IDF also provides its own serial logging system, with different log levels and tags, through the ESP_LOG* macro. This system was used for logging and debugging during development, but will be deactivated in the release versions of the software.

The embedded software is organized in two sections, components and tasks, along with a main function. The entry point for the user developed code in ESP-IDF framework is the "app_main" function, which is located, in this project, in the "main" folder on the "SmartAC.cpp" file, in the embedded system folder in project's GitHub repository[2].

In this main function, all the the necessary configuration and initialization of the embedded system and it's peripherals is made, along with the creation of the tasks.

There are five main tasks for this application: gpio_event_task, uart_event_task, data_processing_task, ac_comm_task and host_comm_task.

The data_processing_task is responsible for reading and processing data to generate the input vector for the machine learning algorithm, running inference on the algorithm, generating a log entry and signaling the ac_comm_task if necessary. This task is programmed to run periodically, or whenever the user presses one of the buttons.

The gpio_event_task is responsible for handling the response to GPIO interrupts, which occur when the user presses one of the buttons. This task receives the information from the interrupt routine identifying which button was pressed, sends it to the data processing task and triggers its execution.

The host_comm_task is responsible for transmitting log information to the host computer through a UART port. This tasks is programmed to run periodically or whenever the available memory reaches a threshold. The command sent from the host computer can also trigger the execution of this task.

The uart_event_task is responsible for detecting and handling UART events. The main

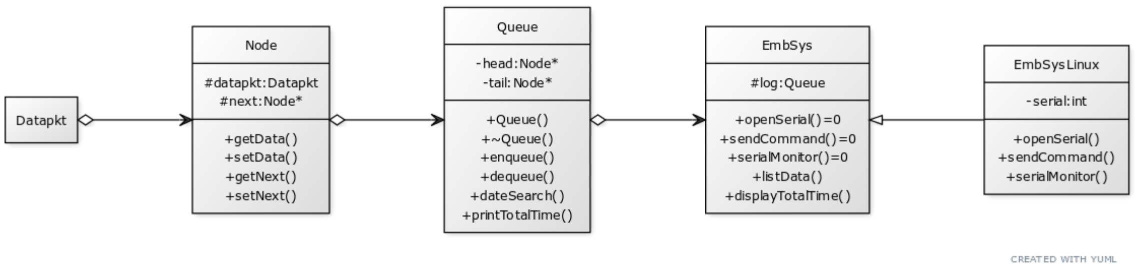

Figure 1: Class diagram for the PC software

event of interest for this application is a pattern detection event, which corresponds to the command send from the host computer. This event occurs when three of the same, specific, predefined character are received through the UART. Upon detection of this event, the execution of the host_comm_task is triggered.

The ac_comm_task is responsible for sending commands to the air-conditioner, either directly, through an IR LED, or indirectly, through the host smartphone. In this application, since communication with the host smartphone is not implemented, the communication is done directly through an IR LED. The implementation of the classes that handle communication with the air-conditioner is based on documentation provided by LISHA[13], found in the project's GitHub repository's[2] Documentation folder.

A diagram showing the relations between the classes is shown in Fig.2

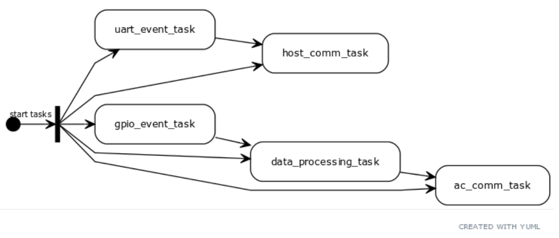

Figure 2: Tasks relation diagram

There are several components in this application, listed below, mainly providing APIs to sensors and peripherals and adding features to the application. The classes developed in previous exercises of the discipline that were used in this project were imported as components.

Embedded system software components:

- esp_exception;
- system_cxx;
- gpio_cxx;
- UART_cxx;
- ClockCalendar;
- Sensor;
- DHT11;
- Queue;
- Node;
- AC_Base
- Samsung AC;
- isr_handlers;

Since there is a decent amount of components and the source codes are fairly documented, only a brief description of each component's features will be presented here.

AS previously mentioned, example components provided in the ESP-IDF framework were used, those example components are esp_exception, system_cxx and gpio_cxx. The most interesting of these components for the application is the gpio_cxx component, which provides a C++ encapsulation of the ESP-IDF gpio driver, which is implemented in C, creating a higher level of abstraction for the GPIO inputs and outputs, each with its own class and methods. The system_cxx and esp_exception components are required by the gpio_cxx component, and provide helper classes and a C++ encapsulation to the native ESP-IDF error system. Since this is an example component, it provides only the most basic functionalities of a GPIO input or output. An API for GPIO interrupts was not implemented in the example initially, so additional classes were implemented in this component, and denoted appropriately in the source code, to provide this functionality.

The UART_cxx component provides a C++ encapsulation of the ESP-IDF uart driver, originally implemented in C. This component was implemented based on the gpio_cxx example component, providing a higer level of abstraction for dealing with UART ports.

The ClockCalendar and Sensor components were adapted and improved from previous exercises of the EEL7323 discipline. The ClockCalendar component provides methods for dealing with date and time, mainly used for recording sensor reading's date and time and log entry's date and time.

The Sensor component is a base abstract class to provide an interface for different sensors, with a virtual method for reading sensor data. In this application, it was used in conjunction with the DHT11 component, an interface for the DHT11[9] temperature and realtive humidity sensor, which provides the DHT11 class that inherits the Sensor class, and provides an implementation for virtual method for reading data.

The Queue and Node components are basically the same ones used im the host computer software, and their purpose is to implement a queue to hold the log entries in the embedded system, befor they are sent to the host.

The AC_Base component is a base abstract class that provides an interface for different air-conditioners, with the common functionalities built in this class.

The Samsung_AC component is a derived class of AC_Base, providing communication methods for the Samsung air-conditioners.

The isr_handlers component provides the interrupt service routines used in the application.

The current class diagrams for the embedded system are shown in Fig.3, Fig.4, Fig.5, Fig.6 and Fig.7, Fig.8.

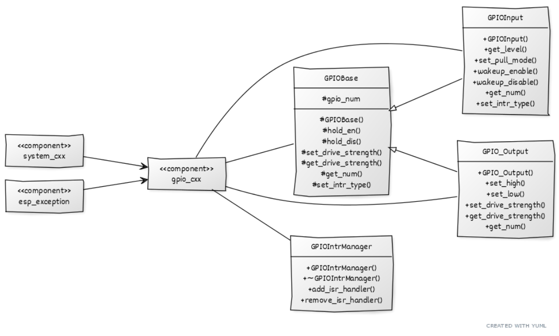

Figure 3: GPIO class diagram

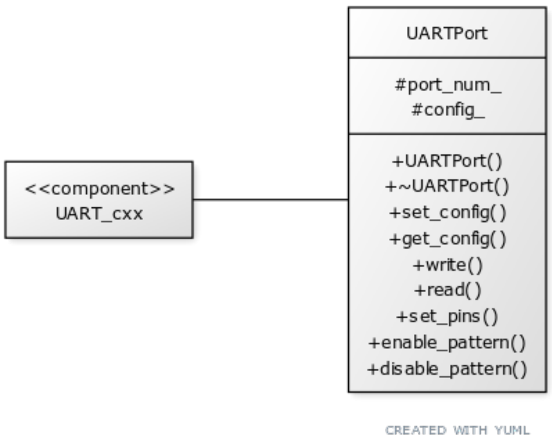

Figure 4: UART class diagram

[tbp]

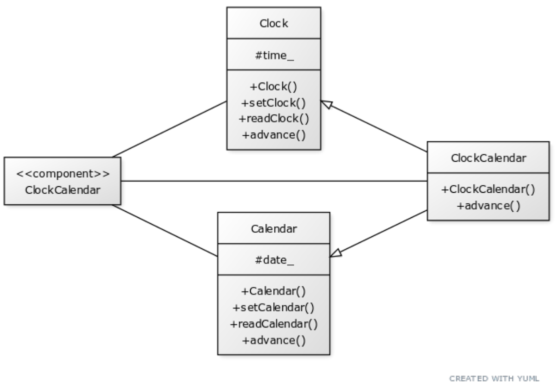

Figure 5: ClockCalendar class diagram

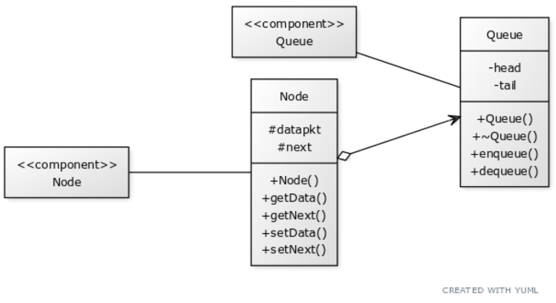

Figure 6: Queue class diagram

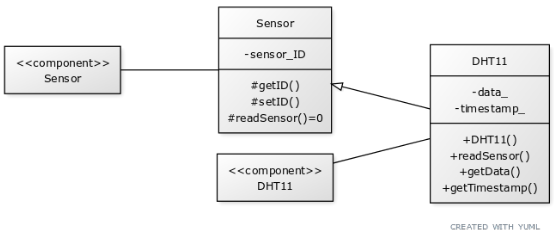

Figure 7: Sensor class diagram

The machine learning algorithm is to be implemented using TensorFlow Lite[14] framework. The embedded system will provide the necessary data, temperature, humidity and state of the buttons, with the required pre-processing, as input to run inference in the machine learning model.

The model should be developed and trained in a computer, and only the final trained version of the model is loaded into the embedded system. Due to time constraints, the machine learning model is not yet developed.

As previously mentioned, the communication between the smartphone and the embedded system is not implemented in this instance of the project, although some suggestions are made here for future implementations.

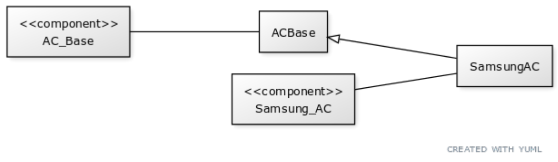

Figure 8: AC class diagram

The ESP32 module[7] has built-in WiFi capabilities, so no additional hardware would be necessary to implement wireless communication between the smartphone and the embedded system, only needing software implementation.

The suggestion for implementing the WiFi functionalities is to follow the same principles applied to the UART and GPIO components, use the C++ to encapsulate the C API in the ESP-IDF framework, to provide a higher level of abstraction and portability.

2.3 Smart phone software

The host smartphone software is to be implemented in C++, using the Android Native Development Kit[15]. The smartphone will function as a host, so the same principles used for the computer software are applied here, with the addition of providing alternate means of communication with the air-conditioner and of acquiring user input.

The software should provide a graphical user interface, with the same options as the computer software's menu, options to list all logged events, to display total power on time of the air-conditioner and to send a command to the embedded system requesting data, along with two additional options for signaling the embedded system, one for uncomfortably high temperatures and one for uncomfortably low temperatures. This serves as an alternative to pressing the physical push-buttons of the embedded system.

The software should also provide an API for the embedded system to send commands to the AC through the smartphone, without needing user interaction for the procedure.

Due to time constraints, this software is not yet implemented.

3 Discussion

As of now, the system is not yet fully implemented, mainly due to time constraints caused by personal matters and technical setbacks.

The most time consuming task in the development of this application was the initial development of the C++ APIs to encapsulate the existing C APIs of the ESP-IDF framework. Although some examples were provided, this implementations requires decent prior understanding of C APIs functionalities on itsef, which required more time than expected to acquire.

Another great deal of time was invested in testing this newly developed APIs, to guarantee that the functionalities were correctly implemented, leading to more delays in development.

This endeavour had its benefits, as discussed in the next section, despite the delays.

Conclusions

Despite the project not being fully implemented yet, its clear how C++ applied to embedded systems can be very beneficial to an application.

The C++ APIs, encapsulating the C drivers functionalities, are much simpler and easier to use, providing a higer level of abstraction. This allows for easier configuration through constructors, safer resource managing through the destructors, and in general a safer approach to using the microcontroller's features.

The Object Oriented aspect of this application allows for easy portability, through use of inheritance and abstract classes providing a common interface to features and devices. As an example, the host computer software could be easily adapted to run in a different operating system, needing only a implementation for the serial communication in the new system, using the abstract base class provided. The same is true for the embedded system software, which easily allows the use of a different temperature sensor or communication with a different air-conditioner, requiring only the implementations of the specific communication parameters for the devices, since the interface is already provided by the abstract base classes.

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.

[13] *LISHA Home Page*. URL: https://lisha.ufsc.br/HomePage.

[14] *TensorFlow Lite website*. URL: https://www.tensorflow.org/lite/.

[15] *Android NDK*. URL: https://developer.android.com/ndk?hl=pt-br.