UFSC 2022 - Embedded Systems Programming in C++

Weather station Embedded System

Gabriel de Oliveira Aguiar[1]

[1]UFSC - Universidade Federal de Santa Catarina

December 1, 2022

Abstract

This project consist of the implementation of a Weather station that uses a machine learning algorithm to determine the next temperature and humidity. The main objective is to reduce energy waste and environmental impact.

The system is divided in three main components. First component is a embedded system inside of a Raspberry Pi, 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, with the same responsibilities as the host computer but using wireless to acquire and send data.

The three of them are developed using C++ with an object-oriented developing, designed to be easily adaptable to different weather station system manufacturers and operating systems.

Keywords
Embedded System, Weather Station, Raspberry, Efficiency, C++, Machine Learning

1 Introduction

A weather station is a facility, either on land or sea, with instruments and equipment for measuring atmospheric conditions to provide information for weather forecasts and to study the weather and climate.

The measurements taken include temperature, atmospheric pressure, humidity, wind speed, wind direction, and precipitation amounts. Wind measurements are taken with as few other obstructions as possible, while temperature and humidity measurements are kept free from direct solar radiation, or insolation. Manual observations are taken at least once daily, while automated measurements are taken at least once an hour.

Weather conditions out at sea are taken by ships and buoys, which measure slightly different meteorological quantities such as sea surface temperature (SST), wave height, and wave period. Drifting weather buoys outnumber their moored versions by a significant amount.

The important thing is that most of the measurements is the same (close) or predictable. We can therefore decrease the number of records using a machine learning model to predict most temperatures and only record outliers. Thus, reducing energy expenditure.

2 Proposed design, Materials & Methods

The system was implemented with a focus on Object-Oriented Programming and C++ language, using the Oracle VM Virtual Box to use Linux (Debian Distro) as development environment and a GitHub repository [1]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 and in the project's GitHub repository's Documentation Folder.

The system is designed to learn how the weather beheaves using machine learning (abbreviated to ML from here on) to reduce energy consumption. The system evaluates environmental data and learns weather characteristics to determine which registerments to do.

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;

- Pass this information as input to the ML algorithm;
- Generate logs registers based on the output of the ML algorithm and sensors;
- Generate a log of all events with detailed information;
- Send log information to host computer and smartphone when requested;

For the hardware will be used:

- Temperature sensor name;
- Humidity sensor name;
- Raspberry Pi 3

2.1 Host computer software

The host computer software was compiled using g++, and uses the GNU Make build system. The source code for the host computer software can be found at the project's GitHub repository[1].

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 events between two dates.
  -Get data, sent or not, between two dates.

The software uses a terminal interface to interact with the user, providing a menu with two options, as mentioned above.

The embedded system interface was heavily based on the RobotLinux 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.

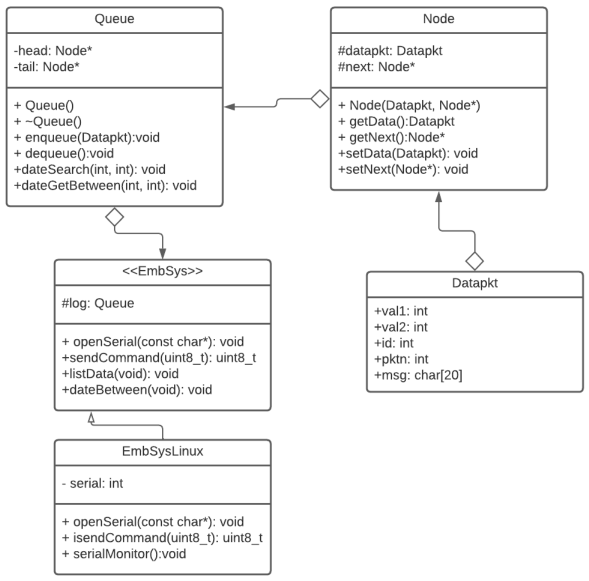

Figure 1: UML Diagram Computer Software

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.

2.2 Embedded software

The chosen hardware to implement the embedded system was an Raspberry Pi 3 model B [2], a single board computer which counts on a Quad Core 1.2GHz Broadcom BCM2837 64bit CPU, plus Wireless LAN, Bluetooth Low-Energy (BLE) and ethernet on board. In addition, the DHT11 sensor was used to measure the local humidity and temperature. The source code for the embedded application can be found at the project's GitHub repository. It worth to remember that despite the Wi-Fi and Bluetooth support that already exist in the platform, the wireless communication will not be implemented in this instance of the project, being purposed as a future work and system upgrade possibility.

The embedded system software must accomplish the following tasks:

- Establish a serial communication with the host computer.
- Periodically read data from DHT11 sensor.
- Periodically run inference in the ML algorithm with acquired data.
- Log all events with detailed information.
- Adjust the data sending periodicity according to the ML results, sending data to the server only when difference between the predicted data and the measured one remain on a pre established error range.
- Send log data to the server when the measuring error is acceptable.
- Send log to host pc when requested.

The Raspberry Pi 3 operating system, called Raspbian OS, natively supports the C++ language, since the C preprocessor and other mandatory packages, as make and libstdc++ are preinstalled on it. To the software implementation the IDE used was Geany, which is preinstalled in the Raspbian OS, in addition to other resources provided by the systems, specially the functionality packages as WiringPi, which were used to control the peripherals (GPIOs, UART bus and memory) [3].

The embedded software is organized in two sections: the main folder, which contains the main function, setup/configuration files and the station controller class; and the components folder, which counts on the files that mainly provide APIs to the sensors and peripherals, adding features to the application. The software is controlled by an FSM implemented in the station controller class,and it's diagram can be seen in Figure 2.

According to this Figure, there are four main states, as follow:

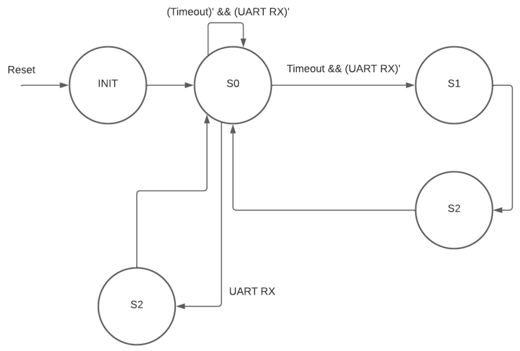

Figure 2: FSM to control the embedded software flow

- **S0:** The first state, which keep in loop waiting for a log sending request delivered by the host PC, meanwhile keep a timer running. Once the timer finish it count, the timeout occurs and the system change it state to S1, in order to perform the sensor reading. Despite that, it only happen if there is no message on the UART buffer, otherwise it means that a log sending request (only command available) has arrived, and the state will change to S2, sending the logs and emptying the queue. This flow is designed in order to attend the user commands as fast as possible, giving it priority over sensor reading. The sensor reading timeout is hardcoded.
- **S1:** Perform the sensor reading and enqueue the obtained data, generating an reading event and changing to state S2.
- **S2:** Compare the read values to the estimated ones by the ML algorithm, verifying if the difference rely in the acceptable error range. The ML algorithm was not implemented in the project, and a function was left as a place holder to future implementation.
- **S3:** Send the enqueued logs when requested, and empty the queue after that.

In order to implement the software control and implement this FSM, a class called StationController has been created, and it's class diagram is in Figure 3.

This project counts on many software components, some of which were built during the course. The components are listed below.

- Clock Calendar
- DHT11

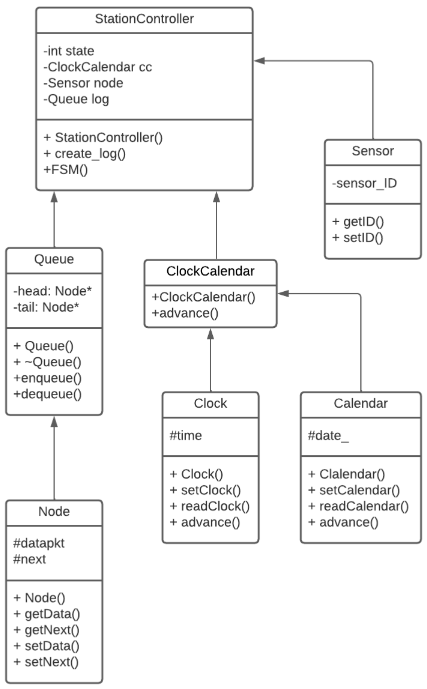

Figure 3: Station Controller class diagram and inheritances

- Node
- Queue
- RaspberryPi
- Sensor
- UART

Some of this components class diagrams are already present in the Figure 3: Queue, Node, Clock Calendar and Sensor. Since there is a significant amount of components and the source code is fairly documented, only a brief description of each component feature will be given here.

The RaspberryPi components are methods for encapsulate the used hardware drivers, and the development were strongly based in the project provided by [4]. The Gpio class create high level methods to handle input and output in RPI3 pins, while the Systemtimer class allows to use the hardware timers to perform the code timing control and the usage in peripherals, like UART. In addition, there is a method to manage the physical memory, which is demanded by the most part of application, since there is a need to set a memory slice for every resource in use. All of this objects are congregated in a higher class called RaspberryPi, wich grant access to this platform features.

The UART component provides a C++ encapsulation of the termios interface, the Linux driver to handle the serial communication, setting up all the configurations and allowing to send/receive data via serial bus. The ClockCalendar provides methods for dealing with date and time, mainly used for recording sensor reading's date and time and log entrys date and time.

The Sensor class is a base abstract class to provide a interface to different sensors, in this application being used in conjunction with the DHT11. The DHT11 component use resources from the RaspberryPi class in order to access the Gpios and perform the data reading, including a checksum to verify the data integrity. The Queue and Node basically implement a queue to hold the log entries in the embedded system before they are send to host.

As said before, the machie learning algorithm was not implemented in this project. It is to be implemented using TinyML framework, and the system will provide the necessary data (humidity and temperature) as input data to run it's inference. The model shall be trained in a external computer, and only the final trained version shall be sent to the embedded hardware. The model can be updated remotely by the server after an evaluation, once the wireless communication is implemented.

2.3 Smart phone software

to do...

2.4 Integration and test

to do...

3 Results

to do...

4 Discussion

to do...

Conclusions

to do...

Acknowledgements

to do...

References

[1] Gabriel de Oliveira Aguiar. Project's GitHub.

[2] Raspberry Pi 3 model B informations.

[3] Raspberry Pi 3 C++ setup.

[4] Raspberry Pi 3 driver methods.