ESIOT 24-25 Assignment #2 - Report

Matteo Bagnolini

December 2024

1 Architecture

After analyzing the behavior of the general system, I decided to tackle the problem using synchronous FSMs.
I identified 4 different tasks that could be implemented as FSMs:

1. Waste Disposal General System: Manages the main state transitions for waste handling.
2. Waste Level Detection: Detects and updates the container's fullness state.
3. Temperature Detection: Monitors and manages the temperature inside the container.
4. User Detection: Handles user presence and sleep mode.

For each of these tasks, I describe the behavior using a state diagram that I will present in the following.

1.1 Waste Detection

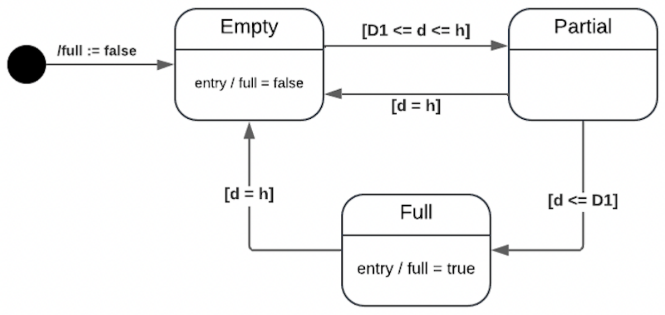


Figure 1: Waste Disposal State diagram

This state diagrams aims to describe the behavior of the task about the detection of the waste level inside the container.
It has 3 different states: Empty, Partial and Full. They represent the state of the container based on the waste level.
A variable named `d` represents the distance measured from some sensor on top of the container. We can compare this variable with `D1` and `h` to determine whether the container is full or not.
D1 is the threshold distance for considering the container full, and **h** is the height of the container. When the sensor measures distance equals to `h`, we can consider the container empty.
A variable named `full` is used to indicate to other FSMs whether the container is full or not.

1.2 Temperature Detection

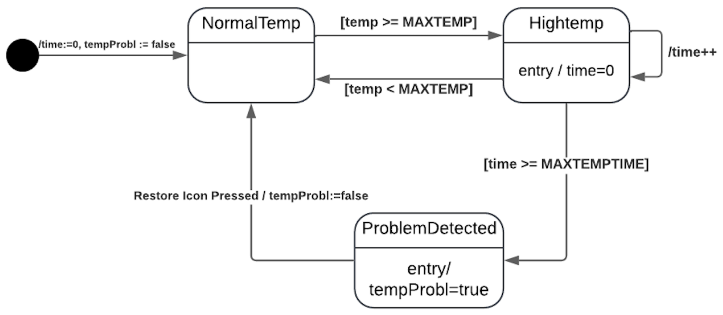


Figure 2: Temperature Detection State diagram

This state diagram aims to describe the behavior of the task about the detection and management of the temperature inside the container.
It has 3 different states: NormalTemp, HighTemp and ProblemDetected.
When the temperature goes above a certain temperature `MAXTEMP` for more than `MAXTEMPTIME` seconds, a problem is detected. This problem can be restored only by an operator acting on the dashboard GUI clicking on the `Restore Icon`.
When a problem is detected, the variable `tempProbl` is set to true to let other FSMs know there is a problem.

1.3 User Detection

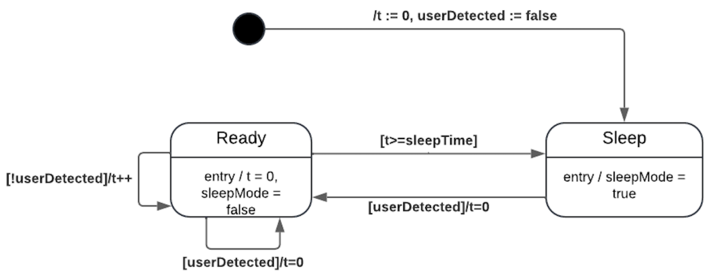


Figure 3: User Detection State diagram

This state diagram aims to describe the behavior of the task about the detection of the user.
It has 2 different states: Ready and Sleep.
If no user is around the container for more than `sleepTime`, the system goes

to sleep, which means the container is not able to collect waste and the LCD screen is turned off to save energy.
When system goes to sleep, the variable `sleepMode` is set to true to let other FSMs know.

1.4 Waste Disposal General System

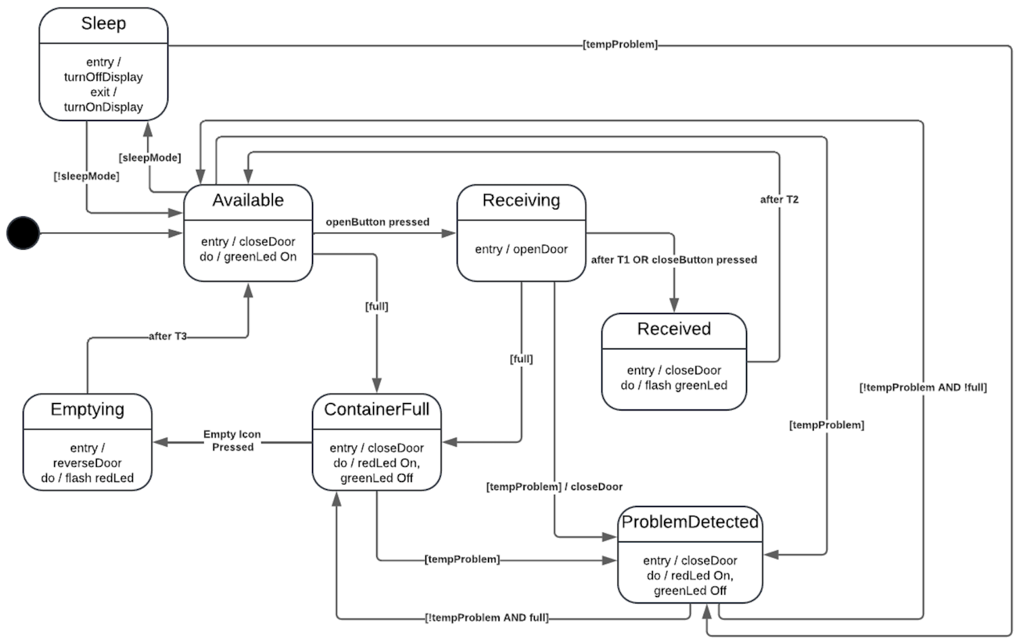


Figure 4: Waste Disposal State diagram

This state diagram aims to describe the core behavior of the system.
It has 7 different states: Available, Receiving, Received, ContainerFull, Emptying, ProblemDetected and Sleep.

1. **Sleep**: The system conserves energy by turning off the display and awakes upon user detection.
2. **Available**: The container is ready to accept waste. Green LED is on, and the LCD prompts the user to press `OPEN`.
3. **Receiving**: The door opens for waste disposal, with a timeout or the user pressing `CLOSE` moving the system to the next state.
4. **Received**: Confirms waste receipt, briefly flashes the green LED, and displays a confirmation message before returning to **Available**.

5. **ContainerFull**: Triggered when the waste level exceeds the threshold. The system stops accepting waste, switches to the red LED, and waits to be emptied.

6. **Emptying**: Activated by the operator. The door opens in reverse to empty the container and then returns to **Available**.

7. **ProblemDetected**: Triggered by high temperature or other issues. The system stops accepting waste and requires operator intervention to restore normal functionality.

Transitions are governed by user interactions with GUI, waste level, temperature, and operator commands. Each state handles specific actions like door movement, LED signals, and LCD messages.
Global variables of FSMs described above are used to determine the state of this FSM.

Notes: the sleep mode I implemented is different from the one described on the specifications. I didn't use arduino's power down mode because I wanted the temperature sensor to continue working even if no operator is detected. Instead I implemented the Sleep state as a "low energy consuming" state, where the lcd backlight is turned off and the system becomes inactive.

2 Implementation

Once the FSMs were defined with state diagrams, the implementation was straightforward. I decided to implement synchronous FSMs, so I needed a scheduler to schedule the execution of each machine.

2.1 Tasks

After creating the Scheduler class (*arduino/include/scheduler/scheduler.h*), I implemented each FSM as a task, that the scheduler can execute (*arduino/include/tasks/*). Every task has its own period, which is defined in the config file (*arduino/include/config.h*). In this file we can also find the mapped pins to connect devices to arduino. Tasks can make known information about the system to other tasks using global variables declared in the globals file (*arduino/include/globals.h*). Some of these variables are used as guards to enable transitions in the FSM. Other variables in this file are useful when communicating with the pc via the serial port.

2.2 Serial Communication

To manage serial communication with the pc I implemented the MsgService class (*arduino/include/communications/msgService.h*). This class can receive and send messages to the serial port.
A task was created to enable the MsgService class to send and receive data periodically. The default period is large enough to not waste resources on arduino. In fact communication consists of periodically sending and receiving the values of the global variables, which represent the state of the overall system.
On the pc side, I implemented a complementary class (*pc/python/src/MsgHandler.py*) that can receive and send messages to arduino.

2.2.1 Packets

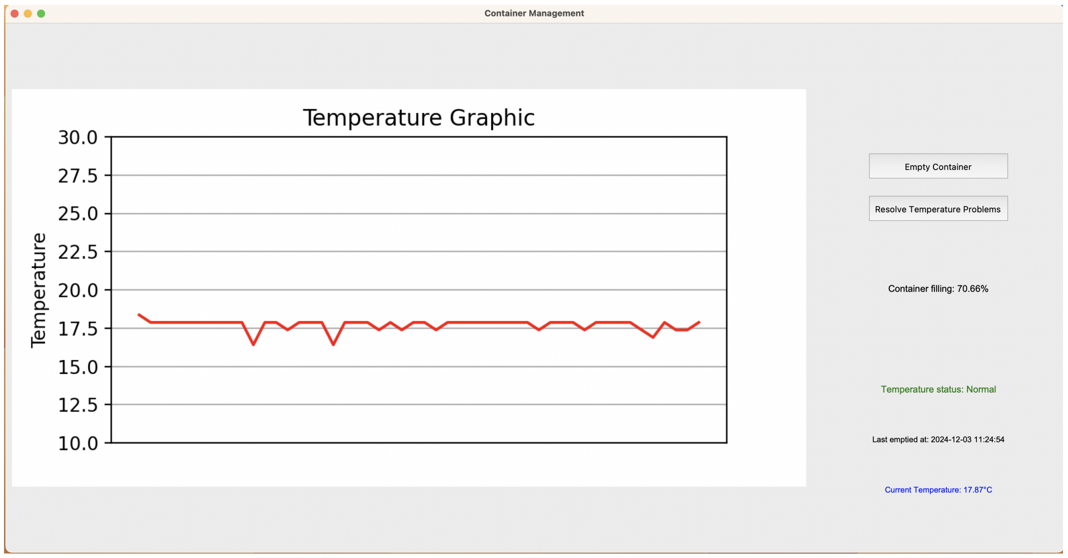
Messages are sent inside of packets. Packets consist of an header and a content. Each global variable to be sent has its corresponding header (a string of 4 characters). The content is the value of the variables to be sent.
Using packets with header and content simplify the process of communication between pc and arduino. In this way each packet is parsed by the receiver using the header, and the corresponding global variable is updated with the content value.
All the different headers can be found in the globals file mentioned above.

2.2.2 GUI

The GUI was build using python and the tkinter library. It can be used by operators to empty the container and resolve temperature problems pressing

the two specific buttons.
The GUI uses serial communication to interact with the Arduino, sending commands (e.g., emptying the container) and receiving real-time updates on system status.

- The graphic on the left shows the temperature evolution inside the container.
- Container filling shows the filling percentage.
- Temperature status shows if the temperature problem is detected.
- Last emptied shows the last time the container has been emptied.
- Current Temperature shows the current temperature inside the container.



Container filling: 70.66%

Temperature status: Normal

Last emptied at: 2024-12-03 11:24:54

Current Temperature: 17.87°C

Figure 5: python GUI

3 Circuit Board

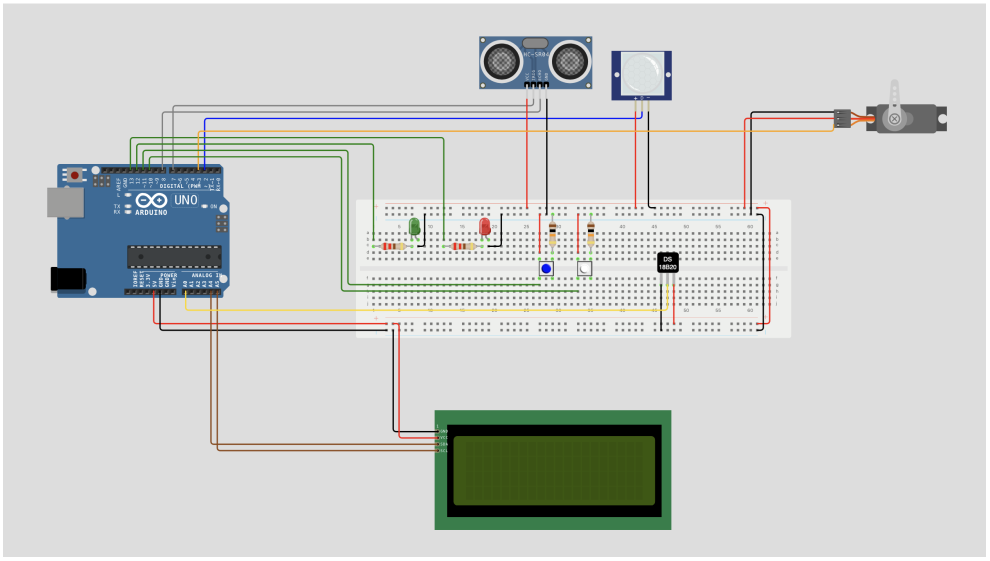


Figure 6: Circuit Board

The circuit includes the following key components:

- **PIR Sensor**: Detects user presence.
- **Sonar Sensor**: Measures the waste level.
- **Servo Motor**: Operates the waste door.
- **Temperature Sensor**: Monitors internal temperature.
- **I2C LCD**: Displays messages to the user.