Alma Mater Studiorum · University of Bologna

Bachelor's Degree in Computer Engineering and Science
A.A. 2025/26

# Smart Tank Monitoring System

**Grazia Bochdanovits de Kavna**
matr. 0001117082

**Alessandro Rebosio**
matr. 0001130557

# Contents

# Chapter 1

# Analysis

This chapter analyses the functional behaviour and hardware structure of the Smart Tank Monitoring System.

## 1.1 Description and Requirements

The Smart Tank Monitoring System is designed to monitor rainwater levels in a tank and automatically control the opening of a water channel based on those levels. The system operates in two modes: `AUTOMATIC` and `MANUAL`, with `AUTOMATIC` as the default startup mode.

**Tank Monitoring Subsystem (TMS).** The TMS is an ESP32-based embedded system responsible for continuously monitoring the rainwater level using sonar sensors. Measurements are sampled at a configurable frequency $F$ and transmitted to the Control Unit Subsystem (CUS) via MQTT. When the system is operating correctly, the green LED is on and the red LED is off. If network connectivity is lost, the red LED turns on and the green LED turns off, signaling a communication failure.

**Water Channel Subsystem (WCS).** The WCS is an Arduino-based embedded system that controls a water channel valve through a servo motor. The opening range spans from 0% (channel closed, 0 degrees) to 100% (channel fully open, 90 degrees). The WCS includes a tactile button to switch between `AUTOMATIC` and `MANUAL` modes. In `MANUAL` mode, operators use a potentiometer to directly control the valve opening level. An LCD display shows the current valve opening percentage and the system mode (`AUTOMATIC`, `MANUAL`, or `UNCONNECTED`).

**Control Unit Subsystem (CUS).** The CUS is the main back-end system running on a PC that orchestrates the Smart Tank Monitoring System. It receives rainwater level data from the TMS via MQTT and implements the control logic. When the level exceeds $L_1$ for longer than $T_1$, the valve opens to 50%; if it reaches $L_2$, the valve opens to 100%. The CUS communicates with the WCS via serial connection and provides HTTP endpoints for the Dashboard Subsystem. If no data is received from the TMS for more than $T_2$ time units, the system enters `UNCONNECTED` mode.

**Dashboard Subsystem (DBS).** The DBS is a web-based front-end accessible from any device connected via HTTP. It displays real-time graphs of the rainwater level over the last $N$ measurements, the current valve opening percentage, and the system state (`MANUAL`, `AUTOMATIC`, `UNCONNECTED`, or `NOT AVAILABLE`). The dashboard includes GUI controls to switch between `MANUAL` and `AUTOMATIC` modes and a widget to manually adjust the valve opening level when in `MANUAL` mode.

**System Architecture.** The four subsystems interact as follows:

- TMS sends rainwater level data to CUS via MQTT.
- CUS processes level data, enforces the control policy, and commands the WCS via serial communication.
- WCS receives commands from CUS and locally manages valve control and operator interaction via the potentiometer and button.
- DBS queries the CUS via HTTP to display real-time system status and provides remote operators with monitoring and control capabilities.

## 1.2 Summary

The Smart Tank Monitoring System is a distributed embedded system that combines automatic and manual control over water channel management. At startup, the system initializes in `AUTOMATIC` mode, with the TMS continuously sampling the rainwater level and transmitting it to the CUS. The CUS evaluates the level against two thresholds: if it exceeds $L_1$ for longer than $T_1$, the valve opens to 50% to begin draining; if it reaches $L_2$, the valve immediately opens to 100%. This dual-threshold approach ensures proportional response to gradually rising levels while providing emergency drainage for critical conditions.

Operators can switch to `MANUAL` mode by pressing the WCS button, allowing direct control of the valve through a potentiometer. The LCD display on the WCS provides real-time feedback of the current opening percentage and mode.The DBS offers remote monitoring and control, displaying historical graphs of rainwater levels and system status.

In case of network failures, the system gracefully degrades to `UNCONNECTED` mode, with the TMS signaling the problem via its red LED and the LCD showing the unavailable state. This design ensures continuous operation of the WCS in `MANUAL` mode, even when the CUS becomes unreachable, maintaining basic functionality during network outages. Once connectivity is restored, the system automatically returns to normal operation.

The system achieves modularity through MQTT publish-subscribe communication, HTTP REST APIs, and serial protocols, allowing each subsystem to operate independently while coordinating through the CUS. This architecture supports scalability, fault tolerance, and ease of maintenance.

## 1.3 Wiring

Table 1.1 lists the essential hardware components required to build and test the Tank Monitoring Subsystem (TMS). Quantities, component types, and their function are summarized for procurement and wiring reference.

| Qty | Component | Notes |
|---|---|---|
| 1 | ESP32 SoC board | Handles sensors, LEDs, and MQTT communication |
| 1 | Sonar sensor | Measures rainwater level in the tank |
| 1 | Green LED | Indicates normal operation and network connectivity |
| 1 | Red LED | Signals network problems or communication failures |
| 2 | Resistors 220 Ω | Current-limiting resistors for LEDs |

Table 1.1: Essential hardware list for Tank Monitoring Subsystem (TMS)

Figure 1.1 shows the wiring diagram of the TMS, highlighting the main connections between the ESP32, sonar sensor, and status LEDs.

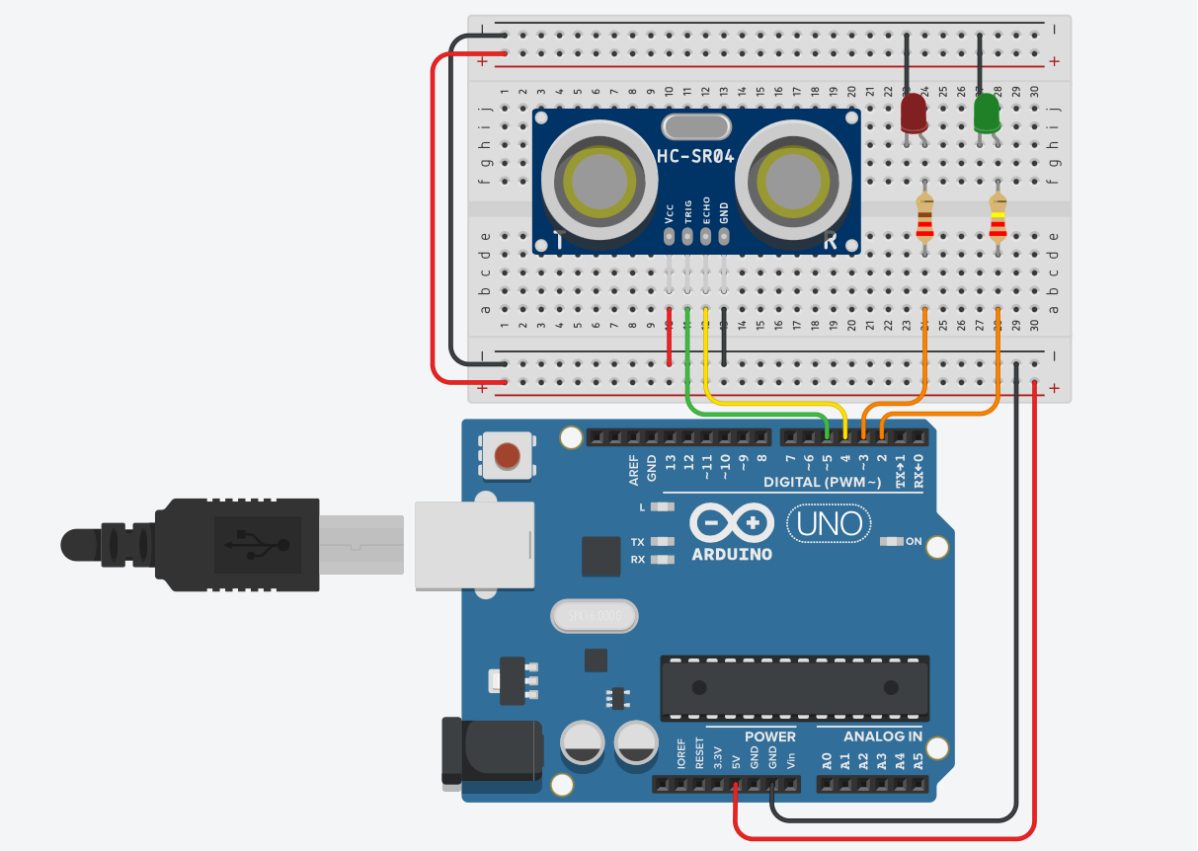


Figure 1.1: Wiring diagram of the Tank Monitoring Subsystem (TMS).

*Note: The wiring diagram shows an Arduino UNO for illustration purposes, but the actual implementation uses an ESP32 SoC board.*

Table 1.2 lists the essential hardware components required to build and test the Water Channel Subsystem (WCS).

| Qty | Component | Notes |
|---|---|---|
| 1 | Arduino UNO board | Handles valve control, sensors, and serial communication |
| 1 | Servo motor | Controls the water channel valve opening (0–90 degrees) |
| 1 | Potentiometer | Allows manual control of valve opening in MANUAL mode |
| 1 | Tactile button | Switches between AUTOMATIC and MANUAL modes |
| 1 | LCD display | Displays valve opening percentage and system mode |
| 1 | Resistor 10 kΩ | Pull-up / pull-down resistor for the button |

Table 1.2: Essential hardware list for Water Channel Subsystem (WCS)

Figure 1.2 shows the wiring diagram of the WCS, highlighting the main connections between the Arduino, servo motor, potentiometer, button, and LCD display.

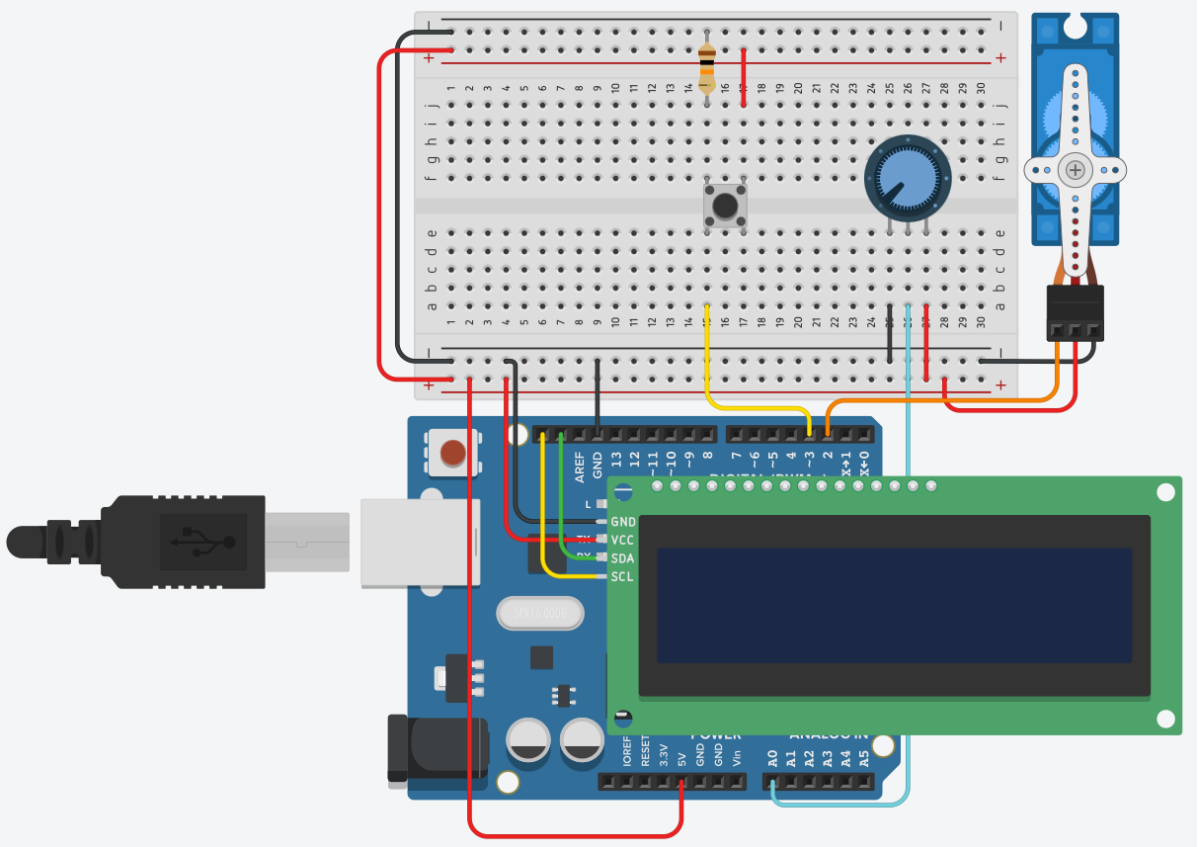


Figure 1.2: Wiring diagram of the Water Channel Subsystem (WCS).

# Chapter 2

# Architecture

## 2.1 Water Channel Subsystem (WCS)

The Water Channel Subsystem is organized according to a task-based architecture, where each task is responsible for a well-defined functionality and is periodically executed by a cooperative scheduler. A shared context is used to store the global system state and to enable indirect communication among tasks.

The following subsections describe the tasks composing the subsystem.

### 2.1.1 System Task

The **SystemTask** manages the operating mode of the subsystem and handles user interaction through the tactile button. At each execution, the task checks the button state and toggles the system mode between `AUTOMATIC`, `MANUAL`, and `UNCONNECTED` when appropriate. The task monitors network connectivity by checking the timestamp of the last received message from the CUS; if no message is received within $T_2$ milliseconds, the system transitions to `UNCONNECTED` mode. The task updates the shared system context to reflect the current mode and to notify other tasks of the state change.

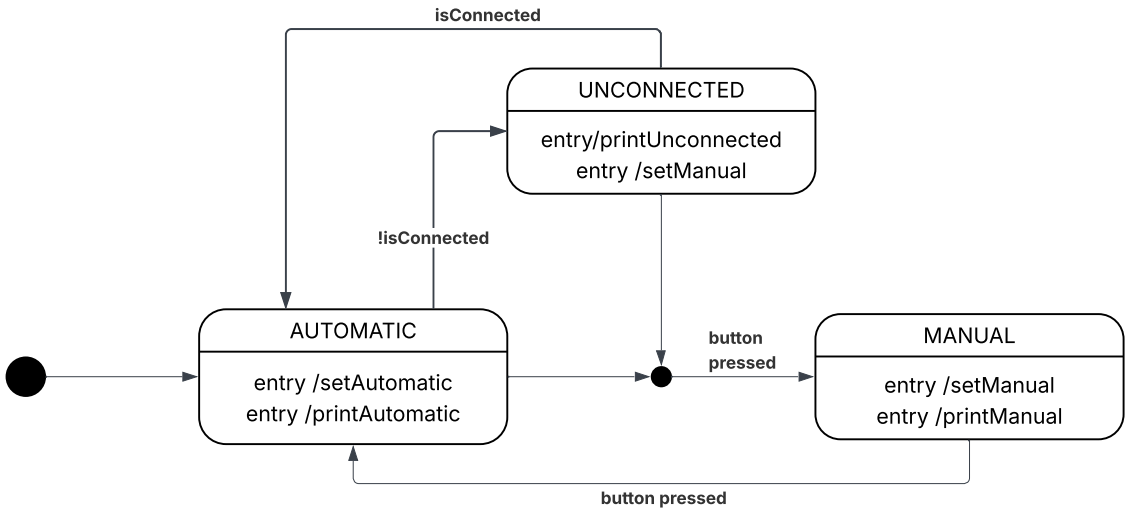


Figure 2.1: System Task state diagram.

### 2.1.2 Valve Task

The **ValveTask** is responsible for controlling the servo motor that operates the water channel valve. At each execution cycle, the task reads the current system mode from the shared context:

- In `AUTOMATIC` mode, the valve position is set according to the target position specified by the CUS through serial communication.
- In `MANUAL` mode, the valve position is directly controlled by the potentiometer analog input, providing operators with immediate feedback and direct control over the valve.

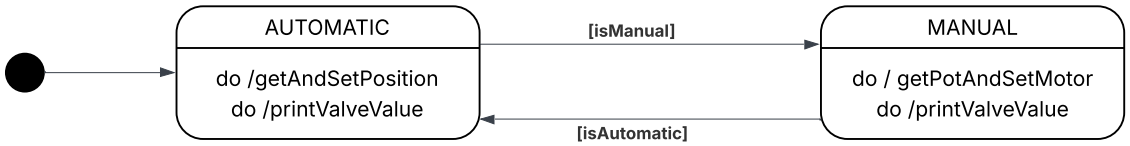


Figure 2.2: Valve Task state diagram.

### 2.1.3 Communication Task

The **CommunicationTask** manages serial communication with the Control Unit Subsystem (CUS). This task periodically checks the serial port for incoming commands and processes them to update the system state. When a new command is received from the CUS, the task updates the context with the target valve position and records the timestamp of the last received message. This timestamp is used by the SystemTask to detect communication losses and trigger the transition to `UNCONNECTED` mode. The task also outputs status information and current sensor readings back to the CUS via the serial interface.

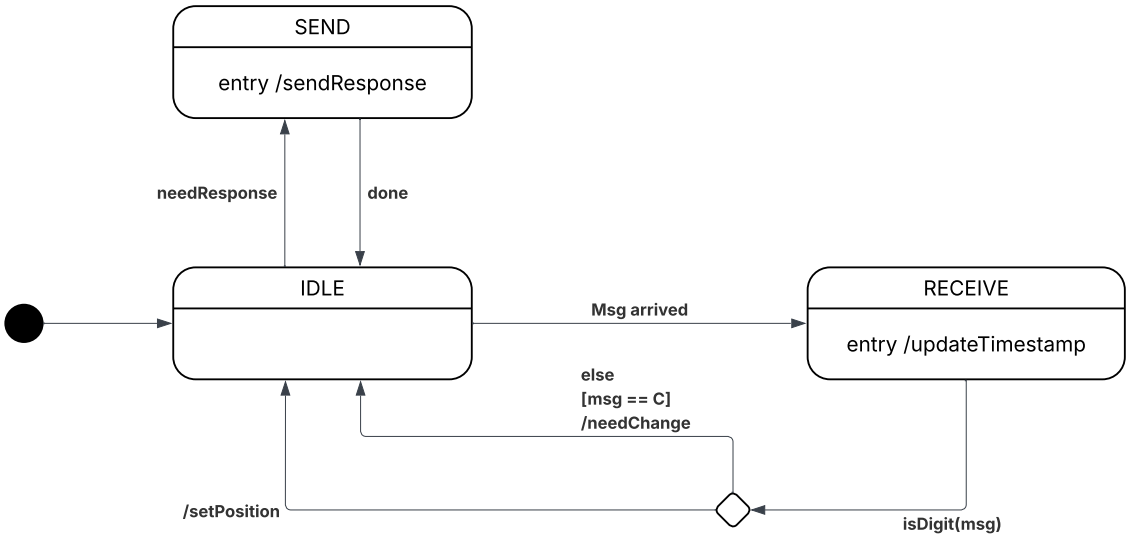


Figure 2.3: Communication Task state diagram.

# Appendix A

# User Interface

## A.1 Overview

The user interface consists of a web-based dashboard that provides real-time monitoring and control of the Smart Tank Monitoring System. Through this interface, you can visualize system data, switch between automatic and manual operation modes, and manually control the valve when needed.

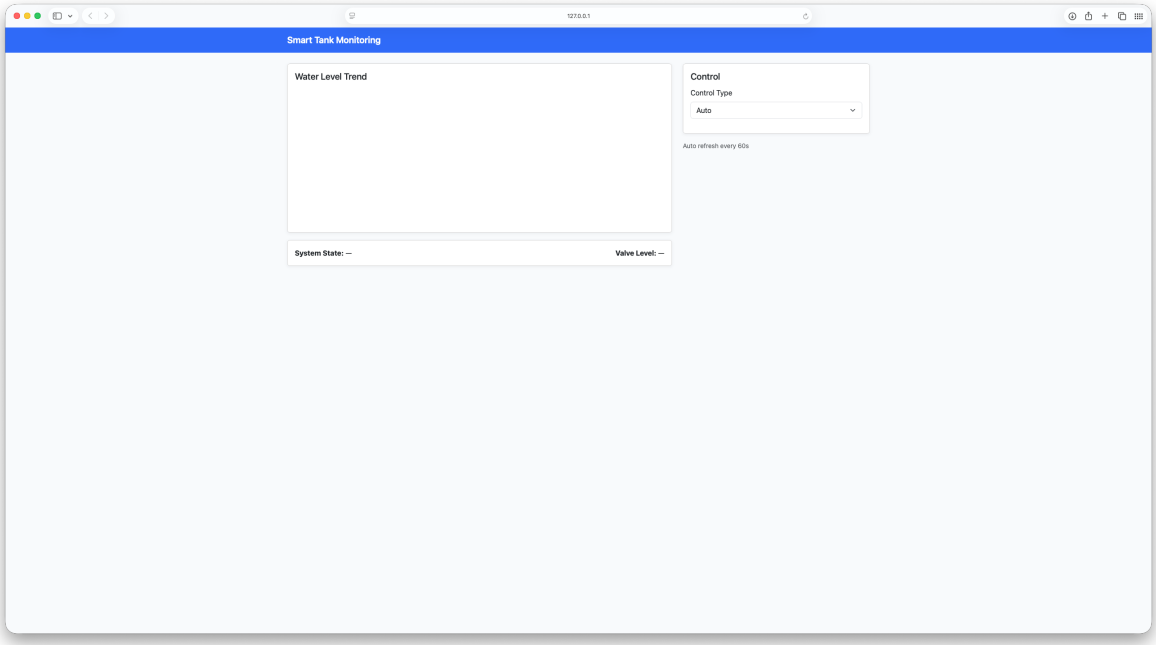


Figure A.1: App view

# Appendix B

# User Guide

## B.1 Setting up the Python environment

Navigate to the control unit subsystem directory and create a Python virtual environment:

```
> cd control-unit-subsystem
> python3 -m venv venv
```

Activate the virtual environment:

- On macOS/Linux:

  ```
  > source venv/bin/activate
  ```

- On Windows:

  ```
  > venv\Scripts\activate
  ```

Install the required dependencies:

```
> pip install -r requirements.txt
```

## B.2 Flashing the embedded devices

The system requires flashing firmware to both the ESP32 and Arduino boards.

### B.2.1 Tank Monitoring Subsystem (ESP32)

Navigate to the tank monitoring subsystem and upload the firmware:

```
> cd ../tank-monitoring-subsystem
> platformio run --target upload
```

### B.2.2 Water Channel Subsystem (Arduino)

Navigate to the water channel subsystem and upload the firmware:

```
> cd ../water-channel-subsystem
> platformio run --target upload
```

## B.3 Running the project

After setting up the Python environment and flashing the embedded devices, you can start the Smart Tank Monitoring System.

### B.3.1 Starting the Control Unit

From the control-unit-subsystem directory (with the virtual environment activated), run:

```
> python control-unit-subsystem/main.py
```

### B.3.2 Starting the Dashboard

Open the dashboard in your web browser:

```
> open ../dashboard-subsystem/index.html
```

Or navigate to the file location and open `index.html` in your preferred web browser.