Smart River Monitoring

1. Design and Architecture

Smart River Monitoring is an IoT system that monitors the water level of a river and controls the related water channel.

The system is divided into four subsystems:

- Water Level Monitoring
- River Monitoring Service
- Water Channel Controller
- River Monitoring Dashboard.

2. Water Level Monitoring

The Water Level Monitoring subsystem runs on ESP32. It is responsible for perpetually monitoring the water level via the sonar.

The water level is sent to the River Monitoring Service via MQTT with a period established by the latter (see *Table 1* on page 2).

This subsystem makes use of a fixed-priority preemptive scheduling policy, and it is composed of the following tasks:

- **Led:** manages the green LED (connected to Wi-Fi) and red LED (no Wi-Fi connection) lighting
- **Keep Connection:** periodically keeps the connection with the MQTT broker alive
- **Reconnection:** in case of connectivity issues with either the Wi-Fi or the MQTT broker, this task periodically tries to reconnect to such services
- **Water Sampling:** periodically samples the water level via the sonar and publishes said value using MQTT.

As an additional functionality, the SPIFFS Filesystem has been used to store the Wi-Fi credentials in the ESP32 flash memory, preventing the display of sensitive information as clear text in the source code.

Said credentials are stored inside a `.env` configuration file with the following structure:

```
WIFI_SSID=my_ssid

WIFI_PASSWORD=my_password
```

This file is subsequently read by ESP at each power up, so that the appropriate variables, containing the actual SSID and password, can be initialised.

3. River Monitoring Service

The River Monitoring Service, the main component of the Smart River Monitoring system, is the backend server and it interacts with all the subsystems via different communication protocols:

- **Serial communication** with the Water Channel Controller
- **MQTT** with the Water Level Monitoring subsystem
- **HTTP** with the River Monitoring Dashboard.

This subsystem decides the current state of the Smart River Monitoring system, depending on the water level sampled by the Water Level Monitoring subsystem.
The table below describes each state:

State	Water level	Sonar sampling period	Valve opening level
NORMAL	$[0.5,\ 1]\ m$	$1000\ ms$	$25\ \%$
ALARM TOO LOW	$<\ 0.5\ m$	$1000\ ms$	$0\ \%$
PRE-ALARM TOO HIGH	$(1,\ 1.5]\ m$	$500\ ms$	$50\ \%$
ALARM TOO HIGH	$(1.5,\ 2]\ m$	$500\ ms$	$50\ \%$
ALARM TOO HIGH CRITIC	$>\ 2\ m$	$500\ ms$	$100\ \%$

[Table 1]

The technologies used to develop the backend server are:

- Node.js
- Express web framework.

4. Water Channel Controller

The *Water Channel Controller* subsystem runs on Arduino and is composed of a valve (modelled with a servo motor), an LCD monitor, a potentiometer, and a button.
This subsystem controls the valve opening level, establishing how much water should flow to the river channel(s).

It is an embedded software comprising a cooperative scheduler that manages three tasks, modelled as synchronous finite state machines (FSMs).
The period of the scheduler is defined by the Greatest Common Divisor (GCD) calculated from all the tasks' periods. This avoids missing relevant events originating from the other tasks.

4.1. Tasks

The tasks' periods have been chosen to avoid missing events, according to the *Minimum Event Separation Time* (MEST). They cooperate via shared variables.
The following table contains the tasks with their related periods:

Task name	Period
System mode	$500\ ms$
Mode switch	$100\ ms$
Message receiver	$100\ ms$

[Table 2]

4.1.1. System mode

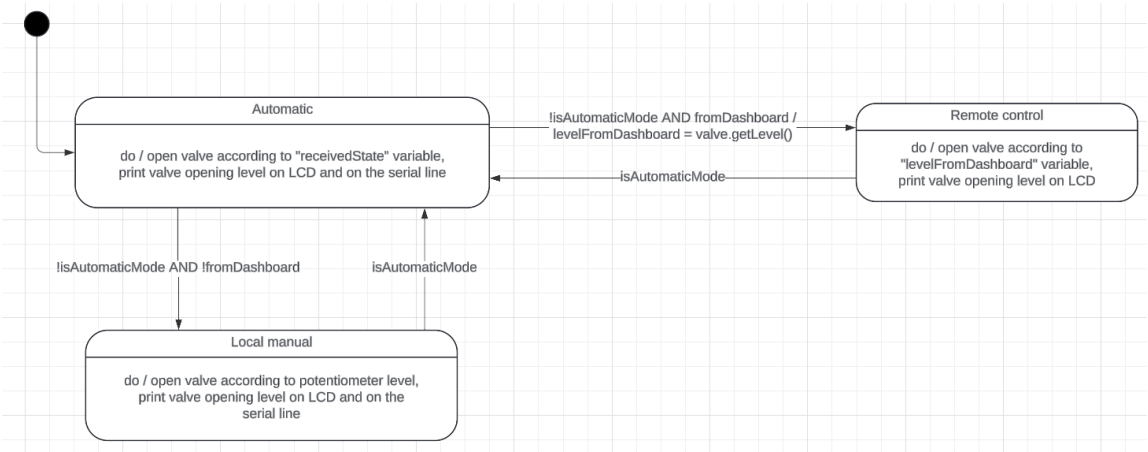
The System Mode Task is the task that manages the valve opening level.
It uses the global variables `isAutomaticMode`, `fromDashboard`, `receivedState`, and `levelFromDashboard`.

It starts in an *Automatic* state, in which the valve opening level is determined by the current system state, which is sent on the serial line by the River Monitoring Service (see *Table 1*).
When the button is pressed (`isAutomaticMode` and `fromDashboard` are both `false`), the FSM goes into the *Local manual* state.
Otherwise, if the switch on the River Monitoring Dashboard is pressed (this event is embodied by the `fromDashboard` variable), the FSM sets the `levelFromDashboard` variable with the current valve opening level and goes into the *Remote control* state.

In the *Local Manual* state, the valve opening level is determined by the potentiometer.
The machine exits from this state when the button is pressed again (`isAutomaticMode` is `true`) and returns to the *Automatic* state.

In the *Remote control* state, the valve opening level is determined by the dashboard. The global variable `levelFromDashboard` contains the value read from the River Monitoring Dashboard.
The machine returns to the *Automatic* state when the dashboard user presses the switch again (the `fromDashboard` variable becomes `false` while `isAutomaticMode` becomes `true`).

In all the states of this finite state machine, the valve opening level is printed on the LCD monitor; while in both the *Automatic* and *Local Manual* states the valve opening level is also sent on the serial line so that the valve opening level on the River Monitoring Dashboard can be updated.



4.1.2. Mode switch

The Mode Switch Task is the FSM appointed to detect the events that trigger a change in the state of the System Mode task machine and to manage the printing of the actual system state on the LCD monitor.
This finite state machine declares and handles the variables `isAutomaticMode` and `fromDashboard`, used by the other FSMs.

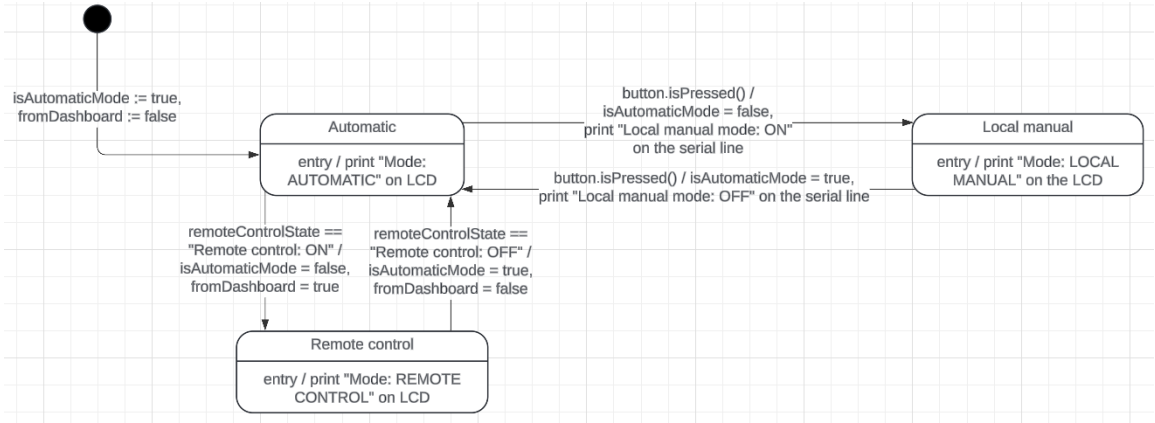
The boolean variable `isAutomaticMode` is `true` when the subsystem must set the valve opening level based on the current system state (the System Mode FSM must be in the *Automatic* state).
When it is `false`, the System Mode machine can be either in the *Local Manual* state or in the *Remote control* state.

The `fromDashboard` variable is a boolean variable that is `true` when the switch on the River Monitoring Dashboard has been pressed, meaning that the System Mode machine has to be in the *Remote control* state.

The FSM uses the global variable `remoteControlState`.
It starts its execution in the *Automatic* state. When the machine enters this state, the string “Mode: AUTOMATIC” is printed on the LCD.
During the *Automatic* state, the machine has to detect the button and the variable `remoteControlState`.
If the button is pressed, the machine goes into the *Local Manual* state, and consequently sets the variable `isAutomaticMode` to `false` and sends the string “Local manual mode: ON” on the serial line to inform the River Monitoring Service that the subsystem has entered the Local Manual mode.
On the other hand, if during the *Automatic* state, the variable `remoteControlState` equals the string “Remote control: ON”, the variable `isAutomaticMode` is set to `false` and `fromDashboard` is set to `true`, and the FSM goes in the *Remote control* state.

When the machine enters the *Local manual* state, the string “Mode: LOCAL MANUAL” is printed on the LCD monitor. In this state the finite state machine has to monitor only the button: if it is pressed again, the FSM sets the variable `isAutomaticMode` to `true`, prints the string “Local manual mode: OFF” on the serial line, and returns in the *Automatic* state.

When the machine enters the *Remote control* state, the string “Mode: REMOTE CONTROL” is printed on the LCD monitor.
During this state, the finite state machine controls only the value of the `remoteControlState` variable: if it equals the string “Remote control: OFF”, the variable `isAutomaticMode` is set to `true`, the variable `fromDashboard` is set to `false` and the machine returns in the *Automatic* state.



4.1.3. Message receiver

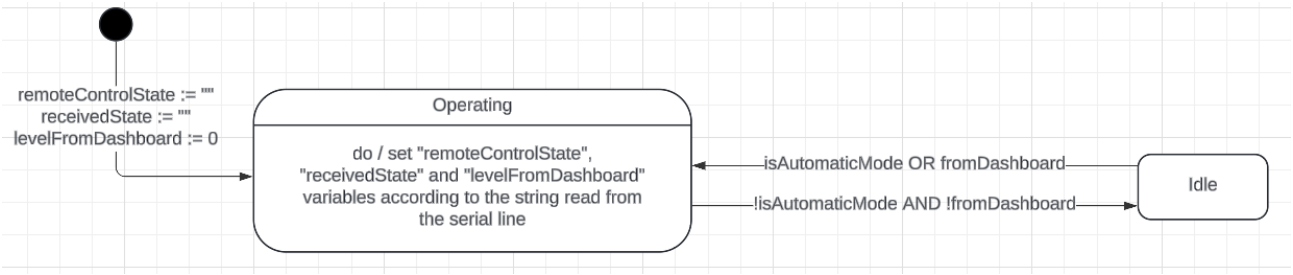
The *Message receiver task* is appointed to read the values from the serial line and set the related global variables, used by the other FSMs.
It declares and handles the variables `remoteControlState`, `receivedState` and `levelFromDashboard`.
The variable `remoteControlState` contains the string that the River Monitoring Service sends when the user on the River Monitoring Dashboard interacts with the switch: its values can be “Remote control: ON”, or “Remote control: OFF”.

The variable `receivedState` contains the current system state (“AUTOMATIC”, “ALARM_TOO_LOW”, etc.).

The variable `levelFromDashboard` contains the valve opening level set from the dashboard.

The FSM uses the global variables `isAutomaticMode` and `fromDashboard`.
It starts its execution in the *Operating* state, where it reads the string from the serial line and, based on its format, sets the right global variable with the value read from the serial line.
For example, if the string starts with “Remote”, then the message arrived on the serial line is related to the switch on the River Monitoring Dashboard, and so the value of the variable `remoteControlState` is set with the string arrived on the serial line.

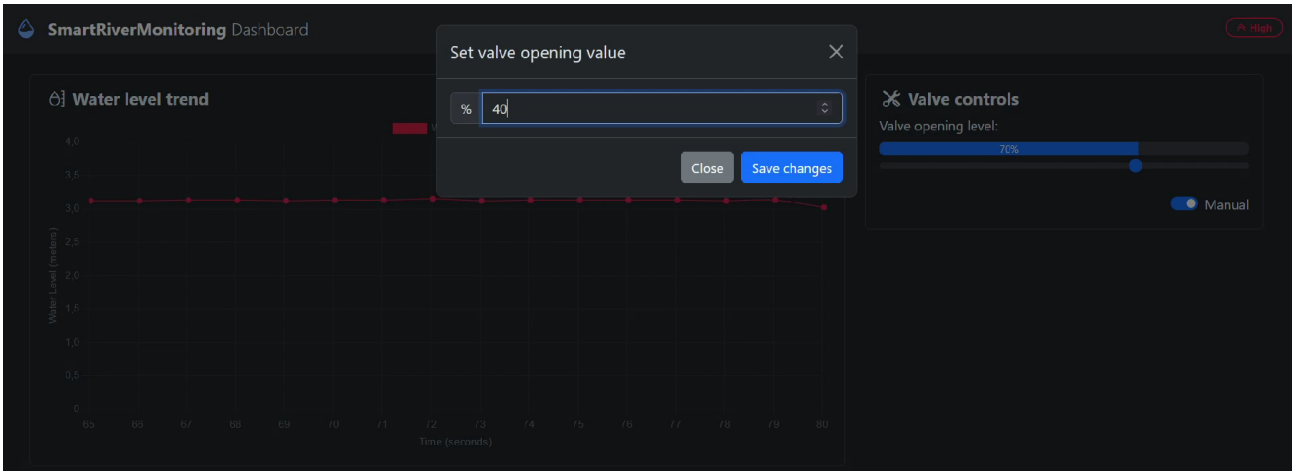
When the System Mode FSM is in the *Local Manual* state (`isAutomaticMode` and `fromDashboard` are both set to `false`) the River Monitoring Service does not write any value on the serial line, so the Message Receiver machine goes into the *Idle* state, where it does nothing. The FSM exits the *Idle* state when the System Mode machine goes either in the *Automatic* state or in the *Remote control* state (either `isAutomaticMode` or `fromDashboard` are `true`).



5. River Monitoring Dashboard

The River Monitoring Dashboard is the front-end web application that visualises the real-time water level on a line graph and provides remote control of the Water Channel Controller through the River Monitoring Service.

A key feature of this dashboard is its responsiveness and synching with the other components. For example, the switch that toggles the manual mode on the dashboard will be disabled when the Water Channel Controller is in *Local Manual* mode and re-enabled when the previously mentioned subsystem exits *Local Manual* mode.



6. Circuit Design