

Smart River-Monitoring System

This embedded project is based on 4 subsystems.

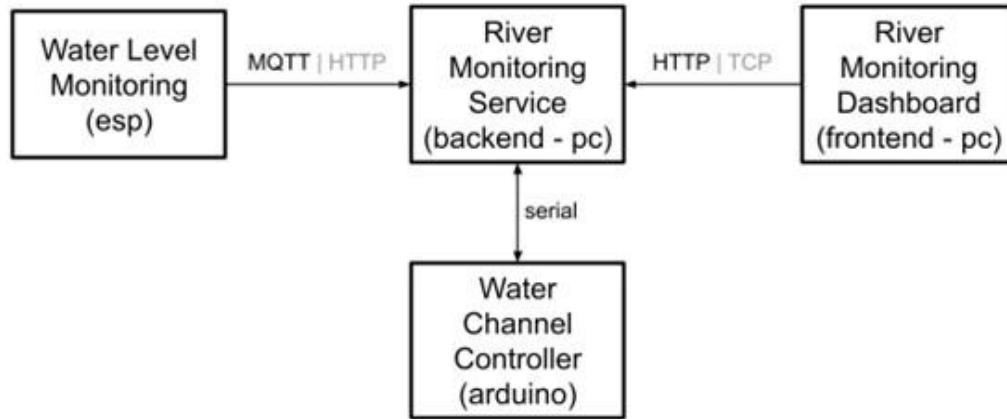


Figure 1 – System outline

The system is composed of 4 subsystems:

- **Water Level Monitoring** subsystem (running on ESP32 SoC):
 - embedded system to monitor the water level of a river
 - it interacts with the *River Monitoring Service* subsystem (via MQTT)
- **Water Channel Controller** subsystem (running on Arduino microcontroller):
 - embedded system controlling the gate/valve of a water channel
 - it interacts via serial line with the *River Monitoring Service* subsystem
- **River Monitoring Service** subsystem (backend - running on a PC server):
 - service functioning as the main unit governing the management of the Smart River-Monitoring System
 - it interacts through the serial line with the *Water Channel Controller* subsystem
 - it interacts via MQTT with the *Water Level Monitoring* subsystem
 - it interacts via TCP with the *River Monitoring Dashboard* subsystem
- **River Monitoring Dashboard** subsystem (frontend/web app - running on the PC):
 - frontend to visualize and track the state of the *River Monitoring Service* subsystem
 - it interacts with the *River Monitoring Service* subsystem via TCP

1. Water Level Monitoring

It runs on ESP32-S3 devkitC-1 n16r8v.

Manages the water level sampling and is coordinated by the River Monitoring Service.

At first, it connects to WIFI through given credentials. Credentials must be correct for the system to work, so they must be set “location dependently”. If WIFI disconnects the SoC tries to reconnect to it. If the connection can’t be established for any reason the ESP32 is unable to communicate with the River Monitoring Service, so the System does not work properly.

If the WIFI connection is established correctly than the SoC can connect with the MQTT broker, particularly broker.emqx.io. The connection is secured by attacks because TLS is used to identify the correct server thanks to a certificate (source <https://github.com/emqx/MQTT-Client-Examples>).

In this case two topics are considered:

- *SmartRiverMonitoringSystem/WaterLevelMonitoring/Period*, where the new frequency (period in implementation) is sent from the River Monitoring Service to this subsystem and the water level sampling changes consequently
- *SmartRiverMonitoringSystem/WaterLevelMonitoring/WaterLevel*, where this subsystem publishes the water levels sampled at the specified frequency
- When the system works correctly, the network is OK, a green led is on, while when there are some errors, a red led is turned on instead.

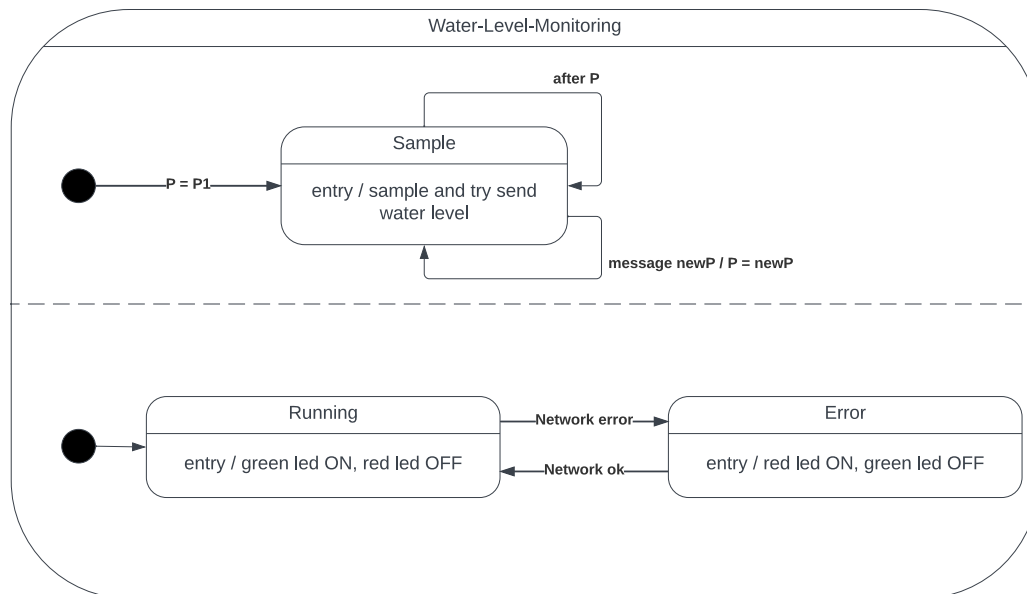


Figure 2 - Water Level Monitoring FSMs

2. Water Channel Controller

It runs on ARDUINO UNO.

Manages the valve opening level in AUTOMATIC or MANUAL modalities.

It does not need the network because it communicates with the River Monitoring Service via serial line.

It manages the modality of the system that can be:

- LOCAL - it is toggled by a button
 - AUTOMATIC – it is coordinated by the River Monitoring Service in whatever modality it wants
 - WHATEVER – the River Monitoring Service decides the valve opening level from remote (via serial communication)
 - MANUAL – it reads the value of a potentiometer and sets the valve opening level according to the read value
- REMOTE - it is toggled by the communication with the River Monitoring Service and is triggered by the River Monitoring Dashboard

Modality and valve opening level are showed in a display.

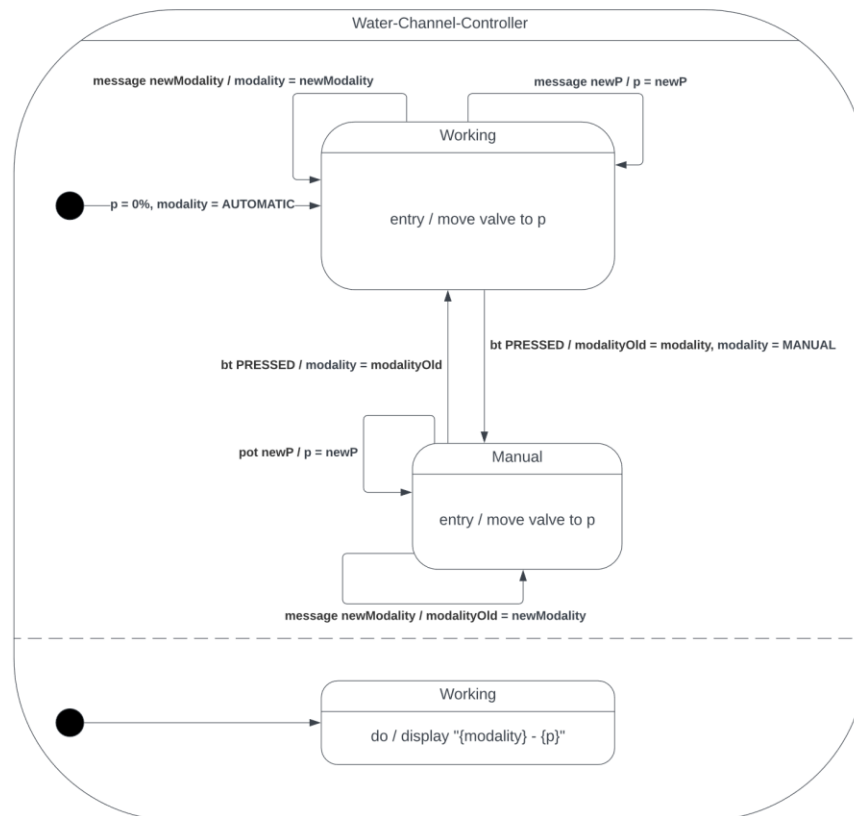


Figure 3 - Water Channel Controller

3. River Monitoring Service

It runs on the PC. In my case I chose a Python application.

Manages the system policy as requested. This service can be considered the core of the system. The different states trigger a change in the behaviors of the two previously described subsystems:

STATE	WATER LEVEL SAMPLING PERIOD (s)	VALVE LEVEL (%)
ALARM-TOO-LOW	10	0
NORMAL	10	25
PRE-ALARM-TOO-HIGH	2	25
ALARM-TOO-HIGH	2	50
ALARM-TOO-HIGH-CRITIC	2	100

The modality of the Water Channel Controller could also be managed by the River Monitoring Dashboard that can set it as MANUAL. In this case if the Water Channel Controller accepts the valve opening level can be set directly by an operator from remote, independently from the described policy of this subsystem. It would be nice to manage a timer to avoid MANUL modality always forget on (this could cause an unexpected behavior).

This service communicates via MQTT with the Water Level Monitoring signing to the described topics in the specular way. Even here the same MQTT broker and TLS system is used (source <https://github.com/emqx/MQTT-Client-Examples>). In particular:

- *SmartRiverMonitoringSystem/WaterLevelMonitoring/Period*, in this case the subsystem sends the period
- *SmartRiverMonitoringSystem/WaterLevelMonitoring/WaterLevel*, in this case the subsystem listens for the water level

Furthermore, if in AUTOMATIC or MANUAL remote mode (from Dashboard) this system sends the valve opening level in percentage to the Water Channel Controller via serial communication.

4. River Monitoring Dashboard

It runs on the PC. In my case I chose a Flask (Python) web application.

Shows System data by communicating with the River Monitoring Service.

Its main components are:

- A web server
- A TCP Client that communicates with the River Monitoring Service to fetch data
- A TCP Client that communicates with the River Monitoring Service TCP Server to toggle the modality from AUTOMATIC to REMOTE MANUAL

It appears as an HTML file. Client-side uses JavaScript with AJAX to fetch data from the web server.

The server uses daemon threads to communicate with the River Monitoring Service via TCP.

Shows if the River Monitoring Service is unreachable, otherwise considers the connection ok. Shows also:

- Current System State
- Current valve level (%)
- Measured water levels in a graph (10 to simplify) with approximative measurements River Monitoring Service times (considering that the Water Level Monitoring subsystem runs on ESP32 that does not have a notion of time synchronized with the external environment this solution was adopted)

The web dashboard enables also to have a MANUAL (remote) control on the Water Channel Controller.

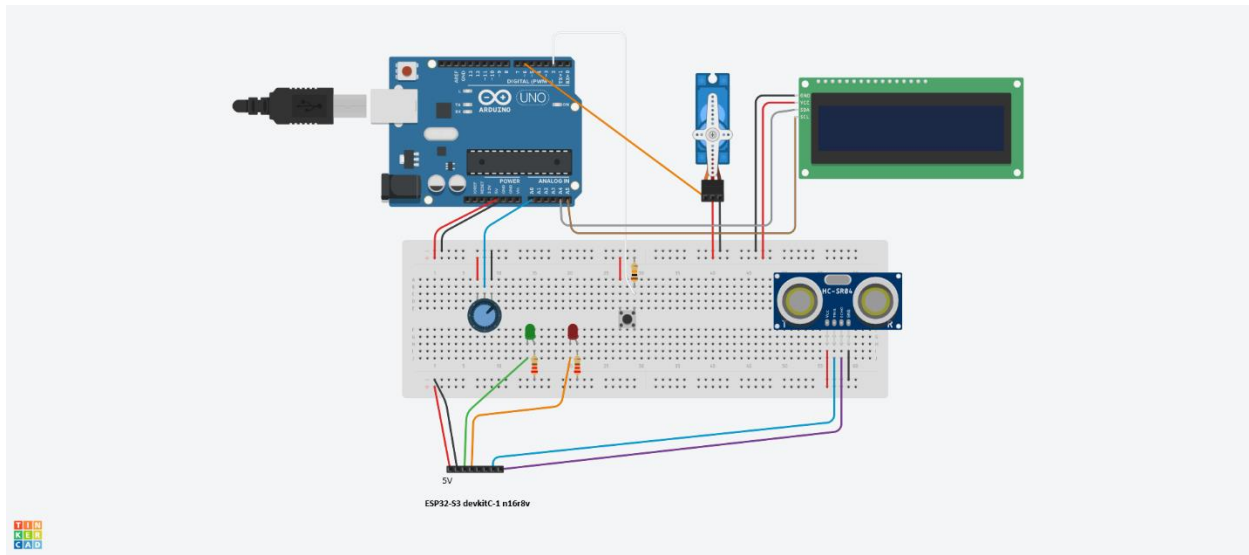


Figure 4 - embedded systems (Arduino + ESP32) components