Water Tank Control and Monitoring System

Control system code documentation

v.1.1

Zahi Abou Chakra – 5468740

Last update 20/07/2025

Index

1. Program Description

The control system program was written in C++ based on the Arduino platform.

The project's layout includes two water tanks. The first accumulates water from a natural water source and the second is directly connected to the structure's plumbing system and is provided with water from the first one. This means that the system that will control the mentioned layouts must support multi-tasking.

Taking into consideration the limits of classic Arduinos' performance (single core) and the necessity to build a program that can control different parts of the system without getting blocked on a single task, the program should be created in a non-sequential manner. This means that the microcontroller (probably with a single core) running the program must perform different actions while the system is running and should not wait until the current task is done to proceed.

2. Code Description

2.1 Code Structure

To make this possible, the code was built with modularity. Starting with single actions, primary functions were created to do small and precise tasks (check section 3. Primary Functions), such as turning on the pump, checking if the tank is being filled, checking the current water level in one of the tanks…

After defining single functions, higher level functions (called logic functions) were created to control the system efficiently using the primary functions. These functions handle different sections of the system and are described in section 4. Logic Functions.

To make the previously mentioned functions work, the system should keep track of its own status. For this reason, a global variable called "status" was created and belongs to an enumeration list containing the different possible statuses of the system.

But how can the status of the system change?

Only logic functions (higher-level functions) discussed in paragraph 4, can alter the status of the system (change the global variable "status").

2.2 Code Files

To respect modularity and make it easier for the reader and debugger to understand the code, it was divided into different files:

2.2.1 control.h

control.h declares the functions that perform input or output actions such as stopping the pump or reading a sensor value. The corresponding control.cpp file defines the functions and their tasks.

The functions declared and defined in these files are discussed in section 3. Primary Functions.

2.2.2 config.h

This file contains all the parameters of the system. It allows the developer to set the parameter from one file instead of changing every value in all the remaining files. It also lets the developer tune the program according to the real-world setup.

The parameters are described in section 4 (System parameters).

config.h belongs to a folder called 'config'. In the config folder, there are config.h, config.cpp and config.ino. The last program is used to calibrate the sensors. In other words, it is a simple program that displays the sensors values, so that the developer can set the max/min parameters in config.h

2.2.3 globalVariables.h

It simply contains the declaration of all the global variables that will be used by different functions and parts of the program, it also contains the declaration of the enumeration list for the system status.

Global variables are 8:

1. status

Defines the global system status with the enumeration format SystemStatus. (system status is described in section 5. **Error! Reference source not found.**)

2. previous_status

This variable is used for the system logic and for the system log via Serial Monitor. It simply memorizes the status before the current one and belongs to the enumeration SystemStatus.

3. source_time

   source_time is used by functions to calculate the elapsed time of filling the accumulator tank. It is useful to check if the tank is being filled after a certain time has passed.

4. pump_time

   Like source_time, pump_time is useful to check the elapsed time of the pump.

5. sup_level_at_start

This global integer variable memorizes the water level of the second tank when the pump has just started. It is used with pump_time to check if the pump is working correctly.

6. inf_level_at_start

   Same as the previous one. Memorizes the water level of the accumulator tank when the natural source was released.

7. sup_current_level

   sup_current_level is used to memorize the water level in the second tank for the current logic loop.

   Measuring different values of the water level in a tank caused by sensor noise or uncertainty can cause errors in the logic loop. This variable solves the problem.

8. inf_current_level

   Same as sup_current_level, used for the accumulator tank. Stores the water level of the accumulator tank for the current loop.

2.2.4 mainLogic.ino

This file contains the main flow of the system. It has a setup() and loop() functions.

On system boot, mainLogic.ino checks for errors before starting the program (this is done inside setup()). The only errors to check in this case are sensor errors.

loop() calls logic functions sequentially and loops indefinitely if there was no errors.

2.2.5 logic.h

This file contains the higher-level function that we called 'logic functions' for managing the two-tank system. There are 4 functions in this file, described in paragraph 4.

3. Primary Functions

There are 7 primary functions. These functions are defined and described in the "control.h" and "control.cpp" files:

3.1 int get_level (int sensor_pin)

The function "get_level" takes in as a parameter the microcontroller's pin number connected to the water level sensor (e.g. ultrasonic sensor) and returns the water level as an integer.

In case of errors (unacceptable values/out of bound), it returns -1.

3.2 void start_pump()

This function simply switches ON the pin connected to the pump.

3.3 void stop_pump()

This function stops the pump by switching the connected pin OFF.

3.4 int check_pumping()

check_pumping is used to check if the pump is working properly. After a certain time has passed it checks if the water level in the second tank has changed (increased). It compares the current water level and the one memorized when the task has started.

It returns “0” if not enough time has passed to check, “1” if the water level has increased (pump working correctly) and “-1” if the water level has not increased (pump not pumping).

The time to check is calculated using the difference between the time set when the function is called for the first time and the current time, using a defined value that is configurable in the config files discussed earlier.

3.5 void start_source()

This function starts filling the accumulator tank releasing the water coming from the natural source.

3.6 void stop_source()

This function stops filling the accumulator tank by switching the connected pin OFF.

3.7 int check_source()

Just like check_pumping, check_source() checks if the accumulator tank is being filled correctly. If not, it throws a SOURCE_ERROR. It compares the current water level and the one memorized when the task has started.

It returns “0” if not enough time has passed to check, “1” if the water level has increased (source available and tank being filled) and “-1” if the water level has not increased (tank not being filled).

The time to check is calculated using the difference between the time set by the first call and the current time, and it is configurable in the config files discussed earlier.

4. Logic functions

There are 4 logic functions:

4.1 void handle_lower_tank()

handle_lower_tank() is responsible for managing the lower tank. If the tank is not full (below a minimum threshold), it starts the water source and updates the system status accordingly. If the source is not filling the tank, it sets the status to SOURCE_ERROR (it uses check_source).

If the tank becomes full, the function stops the source and updates the status to reflect that filling is complete. It also calls again check_source() to reset tracking parameters.

4.2 void handle_upper_tank()

This function is responsible for managing the upper tank. First, it checks if the system is not in a pump error state. If the upper tank is below an acceptable level and the lower tank has enough water, it starts the pump and updates the system status accordingly. It also checks if the pump starts successfully; otherwise, it sets an error status.

If the upper tank is nearly full or the lower tank is nearly empty, it stops the pump and updates the status.

4.3 void handle_error()

The handle_error() function is responsible for diagnosing and responding to different error states in the system, including sensor, pump, and source failures, which are described in section 6.

4.4 void log()

The log() function is responsible for printing status updates to the serial monitor whenever the system status changes. It compares the current status with the previous one, and if a change is detected, it updates previous_status and prints a corresponding message. This includes normal operation states like waiting, pumping, and filling, as well as error conditions such as sensor, pump, or source failures. The function helps provide real-time feedback for monitoring and debugging the system.

5. System parameters

System parameters are declared and defined in the config.h and config.cpp files. There are 3 categories of parameters:

5.1 Connection parameters

- **INF_SENSOR_PIN**: pin number of the microcontroller connected to the water level sensor of the accumulator tank. (input)

- **SUP_SENSOR_PIN**: pin number of the microcontroller connected to the water level sensor of the second tank. (input)
- **SOURCE_PIN**: pin number of the microcontroller connected to the valve that controls the water flow from the natural source. (output)
- **PUMP_PIN**: pin number of the microcontroller connected to the pump (probably via relay. (output)

5.2 Maximum and minimum levels

- **INF_TANK_LO**: Sensor value or minimum value when accumulator tank is Empty (Low)
- **INF_TANK_HI**: Sensor value or maximum value when accumulator tank is Full (High)
- **SUP_TANK_LO**: Sensor value or minimum value when second tank is Empty (Low)
- **SUP_TANK_HI**: Sensor value or maximum value when second tank is Full (High)

5.3 Thresholds

- **MIN_INF_TO_PUMP**: minimum acceptable water level in the accumulator tank to start pumping.
- **SUP_ACC_LEVEL**: second tank threshold for pumping (lower than threshold --> pump)
- **TIME_TO_CHECK**: time in milliseconds to check if pumping or source are working. It defines the time after which the system should check for water level difference.
- **SENSOR_RANGE**: used to avoid sensor errors due to sensor precision. It is used as the divider of the maximum water level of the tank and allows the system to create a threshold for max/min.

6. Error Handling

6.1 Sensor Error

When the system notices an error in the sensors' values (unacceptable values/out of bound) it continuously checks the functionality of the upper and lower tank sensors. After a short delay, it reads each sensor using get_level(). If a sensor still returns an error (value of -1), it logs the malfunction and keeps the system in SENSOR_ERROR. If the sensor reports a valid reading, it logs that the sensor is working and sets the system status back to WAITING, allowing normal operation to resume.

6.2 Pump Error

In PUMP_ERROR the system attempts to verify whether the pump is truly malfunctioning. If the upper tank level is low and the lower tank has enough water, the system tries to restart the pump and monitors the upper tank sensor for a fixed duration (TIME_TO_CHECK). If the water level rises during this period, the error is considered false, and the system resumes normal pumping. If no change is detected, the source is stopped, and the system enters an infinite loop, repeatedly reporting a critical pump failure to prevent further operation.

6.3 Source Error

Source error occurs when the lower tank is not being refilled as expected. It informs via the serial monitor that the source may not be supplying water and keeps the pumping operation available. The system then checks whether the water level in the lower tank increases over time. If it does, the source is assumed to be functioning, and the system resumes normal filling by setting the status to FILLING. If no change is detected, it waits briefly before repeating the check.

7. Useful Links

Github: https://github.com/ZacOS123/Water-Tank-Control-and-Monitoring-System.git