Tank Monitoring and Control System Design Report

Introduction

The following report presents a fluid tank monitoring and control system that can be used as test station in an industrial IOT application design and testing.

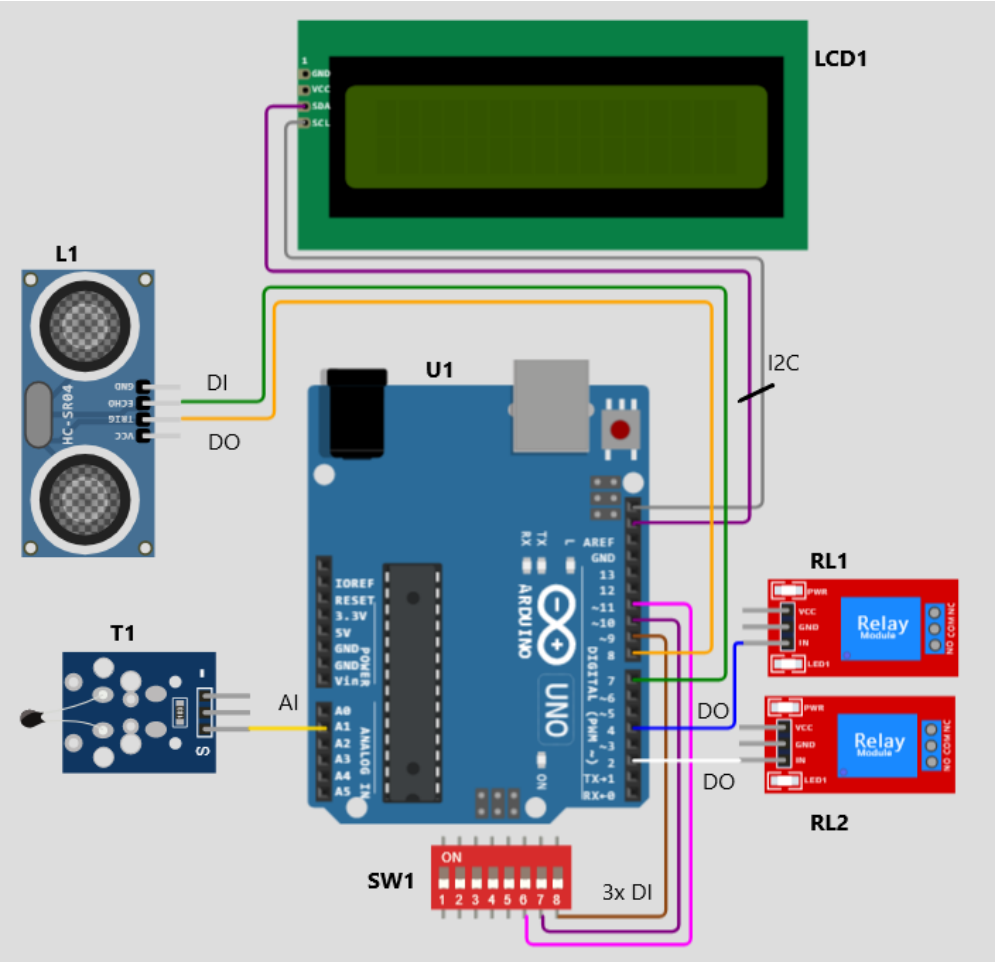
The control algorithm in this report is fictional, but shows a simple operation that can be modified based on the actual application's requirements.

System description and connections

The system consists of two sensors and two relays connected to an Arduino Uno microcontroller board, which transfers the two sensors' data and receives the two actuators' data via serial USB connection from an edge device, which is a Raspberry Pi microprocessor board that stores the sensors data into a database and also contains the logic for the automation and the graphical user visualisation and control. The Arduino system also includes an LCD screen for onsite data visualisation.

As it is a test system, it was built on a test stand for ease of use and fast modification.

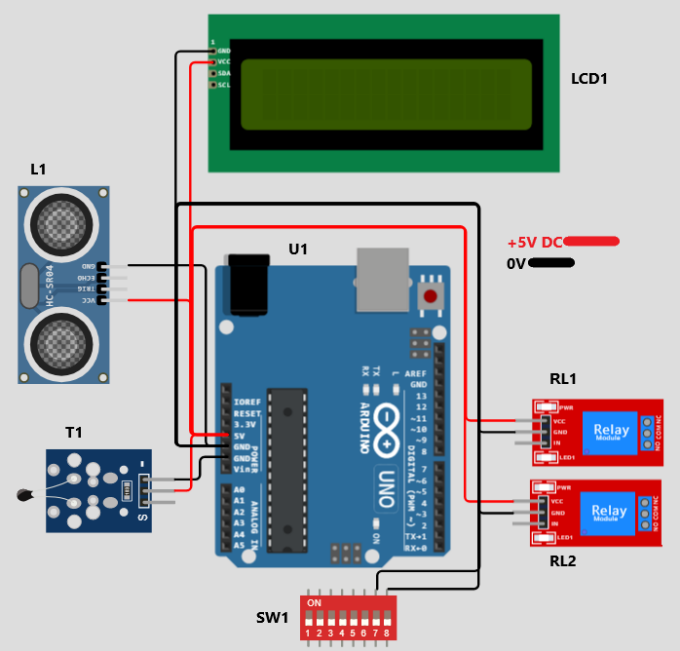
Picture 1 below shows the communication interconnects for the system:



Picture 1. System communication diagram (Created using Wokwi Online Simulator[1])

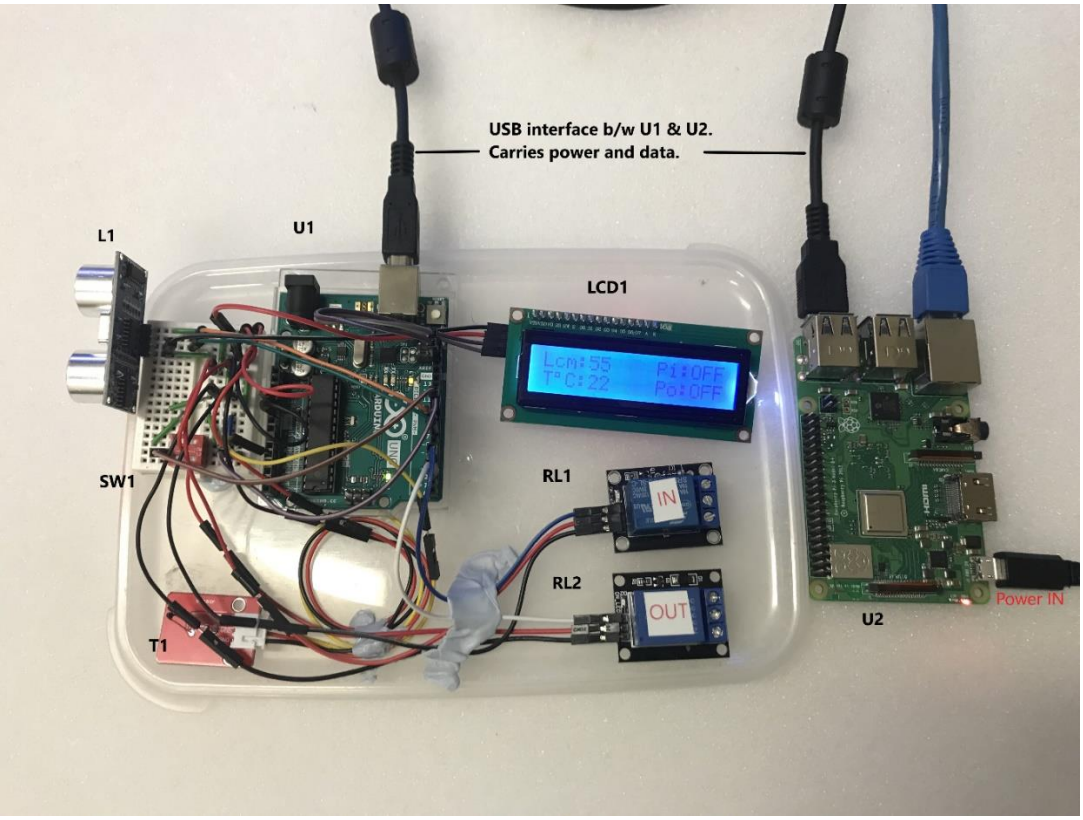
Yeojin Song ID102060145

Picture 2 below shows as simplified power interconnect of the above system:



Picture 2. System power diagram (Created using Wokwi Online Simulator[1])

Picture 3 below shows a photo of the actual system connected and powered from a Raspberry Pi:



Picture 3. Photo of actual system

Table 1 shows the system's bill of material (BOM)

#	Designator	Type	Model	Interface	Power	Quantity
1	U1	Microcontroller Board	Arduino Uno R3	Analog and Digital (5V analog and logic)	5V via USB (from RPi)	1
2	U2	Microprocessor Board	Raspberry Pi 3 B+	Digital (3.3V logic)	5V via wall adapter	1 (physical or virtual)
3	L1	Input device - Ultrasonic Sensor (distance sensing, digital)	HS-SR04	DI & DO (5V logic)	5V from U1	1
4	T1	Input device - Thermistor (temperature sensor, analog)	Jaycar XC4494	AO (0-5V)	5V from U1	1
5	LCD1	Output device -16x2 LCD module	16x2 I2C	I2C serial interface (5V logic)	5V from U1	1
6	RL1, RL2	Output device - Relay module	10A Relay	DI (5V logic)	5V from U1	2
7	SW1	Input device - Dip Switch array for testing	na	DO (5V logic)	na	1
8		USB cable b/w Arduino and RPi	USB B to A	USB	5V	1
9		Jumper wires	na	na	na	as needed
10		Breadboard – power and other interconnects	na	na	na	1
11		Ethernet cable (Rpi to PC, or via WiFi)	na	TCP/IP	na	1

Input devices

L1 and T1 are the two input sensors to the system. SW1 also acts as an input device that was used during development.

The ultrasonic sensor L1 is used to monitor the tank's fluid level. This type of sensor was selected as it is low cost, easy to implement and works well with most fluid mediums. It communicates with the Arduino board via two digital signals - a trigger digital input (digital output in the Arduino) and an echo digital output (digital input in the Arduino). The based code used is from an Arduino PING example code[2] for an ultrasonic sensor that uses single pin that toggles b/w trigger and echo. The code was modified to use two pins as per our sensor. Only the 'cm' conversion was used in this case and the sensor output was capped at

100cm – all received values above 100cm will display 100cm. The range from 0-100cm was also mapped to 0-100% and the % value is what is passed to the edge controller. The LCD screen also receives this value.

The temperature sensor T1 is used to monitor the tank's fluid temperature. It is a thermistor module and was selected due to its ease of use and good accuracy. It communicates with the Arduino board via 0-5V analog output (analog input in the Arduino). A sample conversion code[3] was provided by the manufacturer and modified to suit the system's requirements. The Math.h library that comes preinstalled with Arduino IDE was also required as a number of multiplications, divisions and log calculations are required for the conversion from a raw binary value to a meaningful number in degrees C. The converted temperature value in degrees C is provided in a double format, but was converted to an integer and the number rounded to the nearest high value. This was done so that the temperature value can fit in the limited character space of the LCD. The rounded number is passed to the edge controller and the LCD.

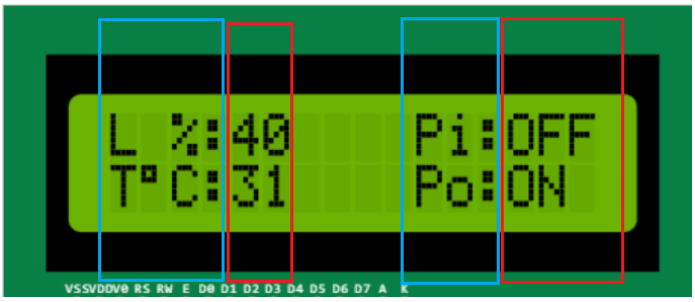
Output devices

The system consists of 3 output devices – 2x relay modules for controlling two valves and one LCD screen for onside data visualisation.

The main output devices are two relay modules RL1 and RL2 that switch ON and OFF the higher-powered valves that stop or allow fluid flow from and out of the tank. The two relay modules are driven directly by two digital outputs at the Arduino side as the modules contain the relay coil driving and protection circuitry. Both relay coils are low power 5V types, which allows power to be supplied to the coils directly from the 5V output from the Arduino board. Initial tests for the relay modules were done with switches (SW1), but later the operation of them was automated based on the input sensor data. The two variables that drive the relays is received from the edge device that checks based on the set logic when to switch ON or OFF each relay. The data is sent via serial UART over USB bus.

The LCD screen is the last output device that is used in the system and its purpose is data visualisation. It is a 16 character by 2 lines (16x2) modules based on the Hitachi HD44780 LCD controller and uses PCF8574AT I2C IO expander IC as its I2C (Arduino side) to parallel (LCD side) interface. Since the communication between the Arduino board and the LCD module is via I2C interface, two libraries are required – first is the preinstalled in the Arduino IDE Wire.h library[4,5] that handles the general I2C initialisation and communication and second is the LiquidCrystal I2C library[6] that handles the LCD control – initialisation, characters display, backlight control etc.

The graphics on the LCD are as per picture 4 below:



The text in the blue areas is static.

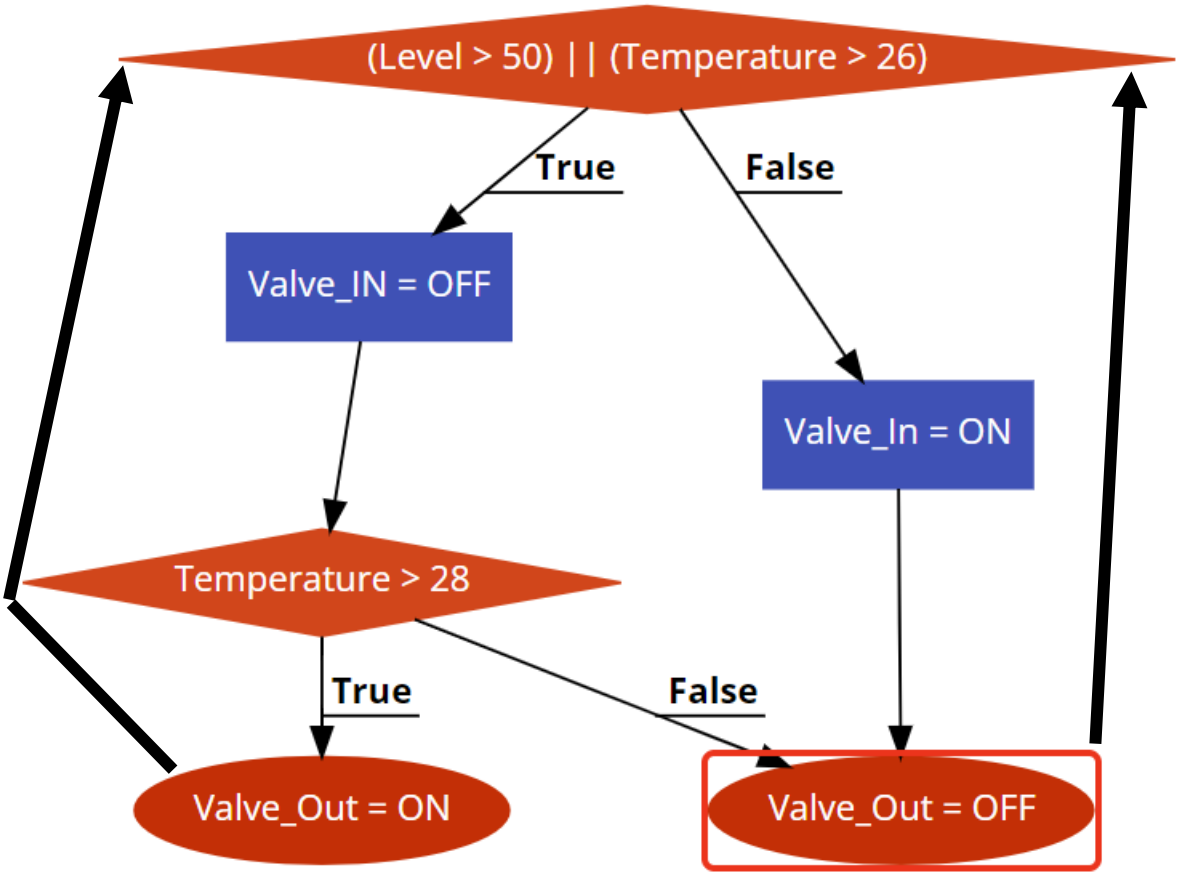
The text in the red areas are the 4 variables, on the left are the input sensor data and on the right are the states of the two relays.

Picture 4. 16x2 I2C LCD module (Created and simulated using Wokwi Online Simulator[1])

Control algorithm

The provided control algorithm is an example of how the system may operate in real situation. The code may be implemented in the Arduino control board for on-site local control, or implemented in the edge device by passing the sensor data, compute and give back the relay states.

A flow chart diagram of the control algorithm is provided below:



Flow Chart 1. Simple Automation Control (Created with code2flow online app[7])

For initial testings the automation logic was ‘placed’ inside the Arduino control board to prove operation. This feature can be later use to develop independent local control if allowed from the edge device, or connection with edge device is lost, or for onsite maintenance purposes. The attached Arduino sketch later in this report shows this local automation control.

The basic operation of the system is described in the following 3 steps:

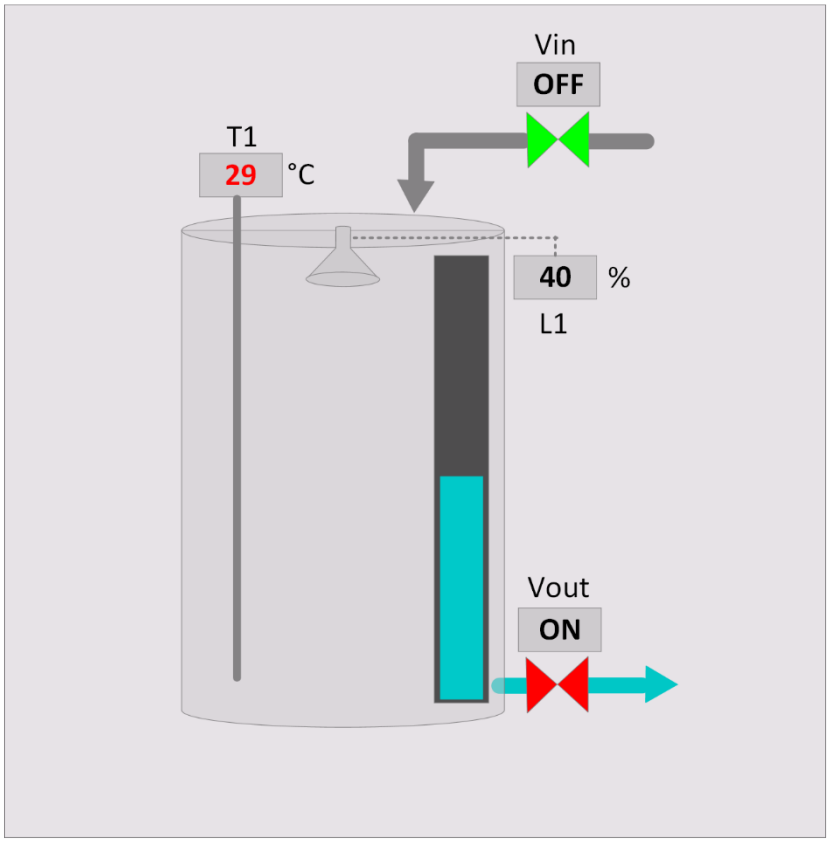
1. Arduino receives sensor data from the two sensors – ultrasonic and temperature;
2. Applying the set logic for the automation and thus controlling the two relays that switch ON, or OFF the valves Vin and Vout. If an edge device is used, the input sensor data will be sent via UART through USB bus to the edge device, information will be stored in database, control applied and the two relay states returned via the same UART through USB bus connection.
3. Displaying temperature, level and the relays status data on the LCD screen. If edge device is not used all this information will also be send via UART through USB bus for visualisation in a e.g. serial monitor in a PC.

Edge system

A Raspberry Pi 3 B+ is chosen (based on availability) for an edge device. The RPi microprocessor board has preinstalled OS on an SD card that handles the firmware and hardware communication tasks. This board also supplies power to the Arduino system via the USB bus.

The incoming sensor information will be stored in a MariaDB database inside RPis OS. Incoming sensor data and outgoing relay status data is handled via a python scrip and passed to the database and the user interface that can be implemented using HTML and CSS.

Picture 5 below shows a proposed HMI (Human Machine Interface), or web-based control panel of the system:



Picture 5 Tank Monitoring and Control UI (Graphics created in MS Visio)

References:

1. Wokwi Online Simulator, https://wokwi.com/
2. Arduino PING Ultrasonic Range Finder, https://docs.arduino.cc/built-in-examples/sensors/Ping
3. Temperature sensor, https://www.jaycar.com.au/medias/sys_master/images/images/9886484398110/XC4494-manualMain.pdf
4. Arduino Wire library, https://www.arduino.cc/reference/en/language/functions/communication/wire/
5. Zambetti, A., A Guide to Arduino & the I2C Protocol (Two Wire), 2023, https://docs.arduino.cc/learn/communication/wire
6. Brabander, F. and Schwartz, M., LiquidCrystal I2C library, Version 1.1.2, https://reference.arduino.cc/reference/en/libraries/liquidcrystal-i2c/
7. Code to flow chart converter, https://code2flow.com/

Appendix

Code 1 Arduino Sketch 1 – local control with print to serial monitor

```
/*
 Author: Yeojin Song
 Date: 28-04-2023
 Revision: 1
 Tank Monitoring and Control System
 Local automation control, no edge device
*/

#include <math.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Constants won't change. They're used here to set pin numbers:
// Set A1 as analog input pin that a thermistor is attached to
const int rawTemp = A1;
// Pin 2 set to control the relay controlling the empty valve
const int RelayOut =  2;
// Pin 4 set to control the relay controlling the fill in valve
const int RelayIn =  4;
// Set pins 7 & 8 as Echo & Triger (DI & DO)
// The two pins are used by the Ultrasonic Sensor
// HC-SR04
const int echoPin = 7;
const int trigPin = 8;

// Variables will change:
// Set variable states
int inState = 0;
int outState = 0;
int manState = 0;
double Thermistor;
double Temp;

// Set the LCD I2C address to 0x3F & config. for 16 chars by 2 line display
LiquidCrystal_I2C lcd(0x3F, 16, 2);

// Initialise setup
void setup() {
  // LCD initialise
  lcd.begin();
  // LCD backlight ON
  lcd.backlight();
  // LCD clear
  lcd.clear();

  // LCD set cursor location and print
  // This data do not change on the LCD
  // Length data on line 1 left side
  lcd.setCursor(0,0);
  lcd.print("L %:");
  // Fill in valve state on line 1 right side
  lcd.setCursor(10,0);
  lcd.print("Pi:");
  // Temperature data on line 2 left side
  lcd.setCursor(0,1);
  lcd.print("T");
  lcd.print((char)223);
```

```
  lcd.print("C:");
  // Empty valve state on line 2 right side
  lcd.setCursor(10,1);
  lcd.print("Po:");


  // Initialize serial communications at 9600 bps:
  Serial.begin(9600);

  // Initialize pins 2, 4 & 7 as outputs:
  pinMode(RelayOut, OUTPUT);
  pinMode(RelayIn, OUTPUT);
  pinMode(trigPin, OUTPUT);

  // Initialize pin 8 as input:
  pinMode(echoPin, INPUT);
}

long microsecondsToCentimeters(long microseconds) {
  // The speed of sound is 340 m/s or 29 microseconds per centimeter.
  // The ping travels out and back, so to find the distance of the object
we
  // take half of the distance travelled.
  return microseconds / 29 / 2;
}

// Main Loop
void loop() {
  // START Temp.
calcs//////////////////////////////////////////////////////////////////////
/
    // Read, assign and convert to K the analog in value:
    Thermistor = analogRead(rawTemp);
    Temp = log(((10240000/Thermistor) - 10000));
    Temp = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * Temp *
Temp ))* Temp );
    // Convert Kelvin to Celcius
    Temp = Temp - 273.15;
    // Round to nearest integer
    int temp_int = round(Temp);
  // END Temp.
calcs//////////////////////////////////////////////////////////////////////
///

  // START Level
calcs//////////////////////////////////////////////////////////////////////
    // Eestablish variables for duration of the sound ping, and the
distance result
    // in inches and centimeters:
    long duration, cm;

    // The sound ping ))) is triggered by a HIGH pulse of 2 or more
microseconds.
    // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    duration = pulseIn(echoPin, HIGH);
```

```
    // Convert the time into a distance in cm
    cm = microsecondsToCentimeters(duration);

    // Cap readings over 100cm to 100cm
    if (cm >100){
      cm = 100;
    }

    // map cm distance from 100 to 0cm to 0-100%
    int Level = map(cm, 0, 100, 100, 0);
  // END Level
calcs//////////////////////////////////////////////////////////////////////
/

   if ((Level > 50) || (temp_int > 26)) {
      // turn Fill In valve OFF due to high temperature (>26) or high level
(>50):
      digitalWrite(RelayIn, LOW);
      Serial.println("Fill In Valve OFF!");
      lcd.setCursor(13,0);
      lcd.print("OFF");
      // Print empty space after the data to ensure all characters clear
      // when relay staus chages from OFF to ON
      lcd.print(" ");

      if (temp_int > 28) {
        // turn Empty valve ON due to high temperature (>28):
        digitalWrite(RelayOut, HIGH);
        Serial.println("Empty Valve ON!");
        lcd.setCursor(13,1);
        lcd.print("ON");
        // Print empty space after the data to ensure all characters clear
        // when relay staus chages from OFF to ON
        lcd.print(" ");
      }
      else{
        // turn Empty valve OFF:
        digitalWrite(RelayOut, LOW);
        Serial.println("Empty Valve OFF!");
        lcd.setCursor(13,1);
        lcd.print("OFF");
        // Print empty space after the data to ensure all characters clear
        // when relay staus chages from OFF to ON
        lcd.print(" ");
      }
    }
    else{
      //turn Fill In valve ON:
      digitalWrite(RelayIn, HIGH);
      Serial.println("Fill In Valve ON!");
      lcd.setCursor(13,0);
      lcd.print("ON");
      // Print empty space after the data to ensure all characters clear
      // when relay staus chages from OFF to ON
      lcd.print(" ");

      // turn Empty valve OFF:
      digitalWrite(RelayOut, LOW);
      Serial.println("Empty Valve OFF!");
      lcd.setCursor(13,1);
```

```
      lcd.print("OFF");
      // Print empty space after the data to ensure all characters clear
      // when relay staus chages from OFF to ON
      lcd.print(" ");
    }

  Serial.println();

  // Print via Serial UART the level in 'cm' to a PC
  Serial.print("Fluid Level = ");
  Serial.print(Level);
  Serial.println("%");
  // Print via Serial I2C the level in 'cm to an LCD
  // Set cursor to charcter # 4 in line 0
  lcd.setCursor(4,0);
  lcd.print(Level);
  // Print empty space after the data to ensure all characters clear
  // when level data chages from xxxx to xxx to xx and x digits
  lcd.print("  ");

  // Print via Serial UART the temp. in '°C' to a PC (format int)
  Serial.print("Fluid Temperature = ");
  Serial.print(temp_int);
  Serial.println("°C");
  // Print via Serial I2C the temp. in '°C' to an LCD
  // Set cursor to charcter # 4 in line 1
  lcd.setCursor(4,1);
  lcd.print(temp_int);
  // Print empty space after the data to ensure all characters clear
  // when level data chages from xxx to xx and x digits
  lcd.print("  ");

  // Set a delay of 1s before the loop executes again
  delay(1000);
}
```

Code 2 Arduino Sketch 2 Edge control – pass input sensor data via serial in the format ‘Level’,’Temp’new line e.g. :

26,21

29,21

and listens to receive via serial the control of the two relay modules:

RelayOut - 1=ON, 2=OFF
RelayIn - 3=ON, 4=OFF

```
/*
 Author: Yeojin Song
 Date: 28-04-2023
 Revision: 1
 Tank Monitoring and Control System
 Remote monitoring and control, edge enabled
*/

#include <math.h>
```

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Constants won't change. They're used here to set pin numbers:
// Set A1 as analog input pin that a thermistor is attached to
const int rawTemp = A1;
// Pin 2 set to control the relay controlling the empty valve
const int RelayOut =  2;
// Pin 4 set to control the relay controlling the fill in valve
const int RelayIn =  4;
// Set pins 7 & 8 as Echo & Triger (DI & DO)
// The two pins are used by the Ultrasonic Sensor
// HC-SR04
const int echoPin = 7;
const int trigPin = 8;

// Variables will change:
// Set variable states
int inState = 0;
int outState = 0;
int manState = 0;
double Thermistor;
double Temp;

// Set the LCD I2C address to 0x3F & config. for 16 chars by 2 line display
LiquidCrystal_I2C lcd(0x3F, 16, 2);

// Initialise setup
void setup() {
  // LCD initialise
  lcd.begin();
  // LCD backlight ON
  lcd.backlight();
  // LCD clear
  lcd.clear();

  // LCD set cursor location and print
  // This data do not change on the LCD
  // Length data on line 1 left side
  lcd.setCursor(0,0);
  lcd.print("L %:");
  // Fill in valve state on line 1 right side
  lcd.setCursor(10,0);
  lcd.print("Pi:");
  // Temperature data on line 2 left side
  lcd.setCursor(0,1);
  lcd.print("T");
  lcd.print((char)223);
  lcd.print("C:");
  // Empty valve state on line 2 right side
  lcd.setCursor(10,1);
  lcd.print("Po:");


  // Initialize serial communications at 9600 bps:
  Serial.begin(9600);

  // Initialize pins 2, 4 & 7 as outputs:
  pinMode(RelayOut, OUTPUT);
  pinMode(RelayIn, OUTPUT);
  pinMode(trigPin, OUTPUT);
```

```
  // Initialize pin 8 as input:
  pinMode(echoPin, INPUT);
}

long microsecondsToCentimeters(long microseconds) {
  // The speed of sound is 340 m/s or 29 microseconds per centimeter.
  // The ping travels out and back, so to find the distance of the object
we
  // take half of the distance travelled.
  return microseconds / 29 / 2;
}

// Main Loop
void loop() {
  // START Temp.
calcs////////////////////////////////////////////////////////////////////
/
    // Read, assign and convert to K the analog in value:
    Thermistor = analogRead(rawTemp);
    Temp = log(((10240000/Thermistor) - 10000));
    Temp = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * Temp *
Temp ))* Temp );
    // Convert Kelvin to Celcius
    Temp = Temp - 273.15;
    // Round to nearest integer
    int temp_int = round(Temp);
  // END Temp.
calcs////////////////////////////////////////////////////////////////////
///

  // START Level
calcs////////////////////////////////////////////////////////////////////
    // Eestablish variables for duration of the sound ping, and the
distance result
    // in inches and centimeters:
    long duration, cm;

    // The sound ping ))) is triggered by a HIGH pulse of 2 or more
microseconds.
    // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    duration = pulseIn(echoPin, HIGH);

    // Convert the time into a distance in cm
    cm = microsecondsToCentimeters(duration);

    // Cap readings over 100cm to 100cm
    if (cm >100){
      cm = 100;
    }

    // map cm distance from 100 to 0cm to 0-100%
    int Level = map(cm, 0, 100, 100, 0);
```

```
  // END Level
calcs//////////////////////////////////////////////////////////////////////
/

  // Following IF statements read the incoming serial data and trigger the
relay modules
  // RelayOut - 1=ON, 2=OFF
  //RelayIn - 3=ON, 4=OFF
  if (Serial.available()>0)
   {
      //Read serial input
      int value = Serial.read();
      if (value == '1')
      {
        digitalWrite(RelayOut,HIGH);
        lcd.setCursor(13,1);
        lcd.print("ON");
        // Print empty space after the data to ensure all characters clear
        // when relay staus chages from OFF to ON
        lcd.print(" ");
      }
      else if (value == '2')
      {
        digitalWrite(RelayOut,LOW);
        lcd.setCursor(13,1);
        lcd.print("OFF");
        // Print empty space after the data to ensure all characters clear
        // when relay staus chages from OFF to ON
        lcd.print(" ");
      }
       else if (value == '3')
      {
        digitalWrite(RelayIn,HIGH);
        lcd.setCursor(13,0);
        lcd.print("ON");
        // Print empty space after the data to ensure all characters clear
        // when relay staus chages from OFF to ON
        lcd.print(" ");
      }
       else if (value == '4')
      {
        digitalWrite(RelayIn,LOW);
        lcd.setCursor(13,0);
        lcd.print("OFF");
        // Print empty space after the data to ensure all characters clear
        // when relay staus chages from OFF to ON
        lcd.print(" ");
      }
   }

  // Print via Serial the level in '%'
  Serial.print(Level);
  // Print via Serial I2C the level in 'cm to an LCD
  // Set cursor to charcter # 4 in line 0
  lcd.setCursor(4,0);
  lcd.print(Level);
  // Print empty space after the data to ensure all characters clear
  // when level data chages from xxxx to xxx to xx and x digits
  lcd.print("  ");

  // Print a coma separator
```

```
  Serial.print(",");

  // Print via Serial the temp. in '°C' (format int)
  Serial.print(temp_int);
  // Print via Serial I2C the temp. in '°C' to an LCD
  // Set cursor to charcter # 4 in line 1
  lcd.setCursor(4,1);
  lcd.print(temp_int);
  // Print empty space after the data to ensure all characters clear
  // when level data chages from xxx to xx and x digits
  lcd.print("  ");

  Serial.println();
  //The Input sensor data is printed in the following form:
  // 'Level data','Temp data'new line   e.g.:
  //28,21
  //30,21

  // Set a delay of 1s before the loop executes again
  delay(1000);
}
```

Code 3 Python code on Raspberry Pi that listens for the 2 incoming sensor values and pass them to an HTML code to be displayed on a web page and also takes the toggled values from the web page via the HTML code and passes it to the Arduino to control the Relays.

```
import serial
import time
from flask import Flask, render_template

app = Flask(_name_)

#Dictionay of pins with name of pin and state ON/OFF

pins={
    2 : {'name' : 'Relay Out' , 'state' : 0},
    4 : {'name' : 'Relay In' , 'state' : 0}
    }
#main function when accessing the website
@app.route("/")
def index():

    #TODO: Read the status of the pins ON/OFF and update dicctionary
    #This data will be sent to index.html(pins dicctionary)
    templateData ={ 'pins' : pins }

    #pass the template data into the template index.html and return it
    return render_template('index.html', **templateData)

#function with buttons that toggle depending on the status
@app.route("/<changePin>/<toggle>")
def toggle_function(changePin, toggle):
    #convert the pin from the URL into an integer:
    changePin = int(changePin)
    #Get the device name for the pin being changed:
    deviceName = pins[changePin]['name']
    #if the action part of the URL is "on," execute the code indented
below:
    if toggle == "on":
```

```
            #set the pin high:
            if chagePin ==2:
                ser.write(b"1")
                pins[changePin]['state'] = 1
            if chagePin ==4:
                ser.write(b"3")
                pins[changePin]['state'] = 1
            #save the status message to be passed into the template:
            message = "Turned " + deviceName + " on."
        if toggle == "off":
            if changePin == 2:
                ser.write(b"2")
                pins[changePin]['state'] = 0
            if changePin == 4:
                ser.write(b"4")
                pins[changePin]['state'] = 0

            #Set the pin low:
            message = "Turned "+deviceName+" off."

        #This data will be sent to index.html (pins diccionary)
        templateData = { 'pins' : pins }
        #pass the template data into the template index.html and return it
        return render_template('index.html', **templateData)
    #Function to send simple commands
    @app.route("/<action>")
    def action(action):
        if action == 'action1':
            ser.write(b"1")
            pins[2]['state'] = 1
        if action == 'action2':
            ser.write(b"2")
            pins[2]['state'] = 0
        if action == 'action3':
            ser.write(b"3")
            pins[4]['state'] = 1
        if action == 'action4':
            ser.write(b"4")
            pins[4]['state'] = 0
        #This data will be sent to index.html(pins deicctionary)
        templateData = { 'pins' : pins }
        #pass the template data into the template index.html and return it
        return render_template('index.html',**templateData)

    #Main function, set up serial bus, indicate port for the webserver,
    #and start the server.
    if _name_ == "_main_":
        ser = serial.Serial('/dev/ttyS0',9600,timeout=1)
        ser.flush()
        app.run(host='0.0.0.0', port=80, debug=True)
```

Code 4 HTML code on Raspberry Pi:

```
<html>
    <head>
    <meta http-equiv="refresh" content="5">
    </head>
    <body>
        <h1>Tank Monitoring and Control</h1>
        <p>Length Value 1: {{value}}</p>
        <p>Temperature Value 2: {{value2}}</p>
    <h2>Toggle Relays</h2>
    {% for pin in pins %}
        <h3>{{pins[pin].name}}</h3>
        {% if pins[pin].state==1 %}
            is currently <strong>on</strong></h2>
            <div class="row">
                <div calss="col-md-2">
                    <a herf="/{{pin}}/off">Turn offf</a>
                </div>
            </div>
        {% else %}
            is currently <strong>off</strong></h2><div class="row"><div
calss="col-md-2">
            <a herf="/{{pin}}/on" class="btn btn-block btn-lg btn-primary"
role="button">Turn on</a></div></div>
        {% endif %}
    {% endfor %}
    <h2> Commands </h2>
    <h3>  Relay Out: <a href="/action1"> TURN ON </a></h3>
    <h3>  Relay Out: <a href="/action2"> TURN OFF </a></h3>
    <h3>  Relay In: <a href="/action3"> TURN ON </a></h3>
    <h3>  Relay In: <a href="/action4"> TURN OFF </a></h3>

</body>
</html>
```

Screen captures of the webpage generated on the Raspberry Pi