Smart Water Bottle Cap for Water Purity Monitoring

Urja Panchal
21101631
u4pancha@uwaterloo.ca

Nidhi Patel
21087749
nv3patel@uwaterloo.ca

Rushi Raval
21086166
r3raval@uwaterloo.ca

Abstract

Water quality is a critical concern for public health and environmental sustainability. Traditional methods of water quality monitoring can be labor-intensive and time-consuming. This project presents a smart water bottle cap designed for real-time water purity monitoring using a network of sensors and Arduino microcontrollers. It is important to note that this project is entirely based on computer simulation. The system measures key water quality parameters, including Total Dissolved Solids (TDS), temperature, turbidity, and water level. The prototype was simulated using Proteus software, integrating multiple sensors connected to Arduino boards. These boards communicate wirelessly, forwarding their readings to a central node. The central node aggregates the data and displays the results on a virtual terminal, providing users with immediate feedback on water quality. This innovative approach offers a low-cost, scalable solution for continuous water monitoring, ensuring safe drinking water and enabling prompt responses to contamination events. The project demonstrates the potential of IoT technologies in enhancing water quality management and public health protection.

1. Introduction

Traditional methods of water quality assessment often involve manual sampling and laboratory analysis, which can be time-consuming and may not provide real-time data. There is a pressing need for a smart, automated system that can continuously monitor and assess water quality parameters such as Total Dissolved Solids (TDS), turbidity, and water level.

The increasing global challenges of water pollution, climate change, and growing population demand more efficient and responsive water quality monitoring systems. Real-time data on water quality can enable quick responses to contamination events, optimize water treatment processes, and provide valuable insights for long-term water resource management.

Our goal is to develop a smart, real-time water quality monitoring system that accurately measures and analyzes key parameters (TDS, turbidity, and water level) to assess water purity continuously.

Our objective is to create an cost-effective sensor system for real-time measurement of Total Dissolved Solids (TDS), turbidity, and water level and also create a wireless network to link several sensor nodes for extensive monitoring of water quality.

The proposed solution involves using Proteus simulation software to create a virtual prototype of the smart water bottle cap. This approach allows for the design and testing of a wireless sensor network comprising multiple Arduino boards connected to various water quality sensors. The system will be configured to capture real-time data on TDS, turbidity, temperature, and water level, transmitting this information to a central node for processing and display.

2. Literature Review

Smart water quality monitoring systems using Internet of Things (IoT) technology have gained significant attention in recent years due to their potential to provide real-time, continuous monitoring of water resources. Several studies have explored various aspects of these systems:

Proteus is a powerful Electronic Design Automation software suite essential for this water quality monitoring project. It enables the virtual creation and testing of electronic circuits and microcontroller systems. In this project, Proteus is used to simulate the Arduino-based sensor network, including TDS, turbidity, and water level sensors, allowing for system design refinement without physical hardware. It also facilitates Arduino code testing and debugging, visualizes sensor outputs, and provides tools for circuit design and PCB layout. By integrating various components within Proteus, we can ensure compatibility and functionality, reducing development time and hardware issues while ensuring

system accuracy and reliability. [1]

Arduino is an open-source electronics platform based on easy-to-use hardware and software. It consists of a microcontroller board and an integrated development environment (IDE) for writing and uploading code to the board. Arduino boards can read inputs—such as light on a sensor, a finger on a button, or a Twitter message—and turn them into outputs, like activating a motor, turning on an LED, or publishing something online. The platform is widely used for creating interactive projects and prototypes due to its simplicity and flexibility. [2] The Arduino IDE, though user-friendly, often presents common issues such as serial port detection problems, programmer communication errors, and compilation failures. These issues typically arise from incorrect board settings, faulty connections, missing drivers, or code errors. Familiarity with troubleshooting techniques like verifying connections, managing libraries, and checking code syntax can help users quickly resolve these problems, ensuring a smoother development experience. [3]

Bhatt and Patoliya (2016) developed an IoT-based water quality monitoring system that uses multiple sensors to measure pH, turbidity, conductivity, dissolved oxygen, and temperature. The system employs a microcontroller to process sensor data and uses Zigbee protocol for wireless communication to a central controller. This setup enables real-time, comprehensive monitoring across multiple locations, offering an efficient method for water quality assessment and management. [4]

In conclusion, while IoT-based smart water quality monitoring systems have shown significant progress, limitations exist in simulating these systems. Notably, Proteus software has constraints in modeling wireless sensor networks due to limited inbuilt modules for wireless communication. This restriction poses challenges in accurately simulating and testing comprehensive IoT-based water monitoring systems. Future research should address these simulation limitations alongside improvements in sensor accuracy, energy efficiency, and data analysis techniques to advance the field of water quality monitoring.

3. Methodology

The design and implementation of the smart water quality monitoring system are centered around creating a wireless sensor network using Arduino boards and various water quality sensors. The system architecture comprises three primary components: sensor nodes, a central node, and a wireless communication network, all

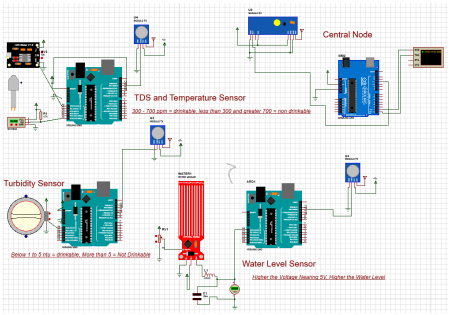

Figure 1. Project Architecture

of which are integrated to provide real-time monitoring of water quality parameters.

The system includes three sensor nodes, each equipped with an Arduino board and a specific water quality sensor. The sensors used are for measuring Total Dissolved Solids (TDS), temperature, turbidity, and water level. Each sensor node is responsible for acquiring data from its respective sensor and transmitting this data wirelessly to the central node.

The central node, which also uses an Arduino board, is equipped with a receiver module that collects data from the three sensor nodes. This node acts as the main controller, aggregating and processing the data received from the sensor nodes. The processed data is then displayed on a virtual terminal, allowing users to monitor water quality parameters in real-time.

The implementation process involves several key steps. First, each Arduino board is programmed to interface with its respective sensor, ensuring accurate data collection. Next, the wireless communication network is established, enabling the sensor nodes to transmit data to the central node. The central node is then programmed to receive, process, and display the data on a virtual terminal. Finally, the entire system is simulated using Proteus software to test its functionality and ensure that all components interact correctly.

Several challenges were addressed during the implementation process. Ensuring sensor accuracy was critical to obtaining reliable data. Power management was another important consideration, as the wireless nodes needed to operate efficiently over extended periods. Additionally, data synchronization across the network was managed to ensure timely and accurate data transmission from the sensor nodes to the central node.

Overall, this methodology outlines a robust approach to developing an effective IoT-based water quality monitoring solution. The attached project architecture diagram visually represents the system's components and their interactions, providing a clear overview of the design and implementation process.

4. Code Snippets:

4.1. Turbidity Sensor Code:

```
#include <VirtualWire.h>

#define sensor_pin A0
#define transmit_pin 12
// Define the pin for RF transmitter

int read_ADC;
int ntu;

void setup() {
  // Initialize serial communication
  // at 9600 baud rate
  Serial.begin(9600);

  // Initialize the RF transmitter
  vw_set_tx_pin(transmit_pin);
  // Bits per second
  vw_setup(2000);
}

void loop() {
  // Read the analog sensor value
  read_ADC = analogRead(sensor_pin);

  // Cap the ADC reading value at 208
  if (read_ADC > 208) read_ADC = 208;

  // Map the ADC value to turbidity NTU
  ntu = map(read_ADC, 0, 208, 300, 0);

  // Prepare the message to send
  char msg[20];
  sprintf(msg, "Turbidity: %d", ntu);

  // Send the message
  vw_send((uint8_t *)msg, strlen(msg));
  vw_wait_tx();
  // Wait until the whole
  // message is gone

  // Delay before the next reading
  delay(1000);
  // Increased delay to reduce
  // transmission frequency
}
```

4.2. TDS Sensor Code:

```
#include <VirtualWire.h>

#define tds_sensor_pin A0
#define temp_sensor_pin A5
#define green_led_pin 8
#define red_led_pin 9
#define buzzer_pin 13
#define rf_transmitter_pin 12

float analog_reference = 4.3;
float ec_calibration = 1;
float electrical_conductivity = 0;
int total_dissolved_solids = 0;
int raw_temperature;
float water_temperature = 0;

void setup() {
  pinMode(tds_sensor_pin, INPUT);
  pinMode(red_led_pin, OUTPUT);
  pinMode(green_led_pin, OUTPUT);
  pinMode(buzzer_pin, OUTPUT);

  Serial.begin(9600);
  Serial.println("Timestamp,
    TDS, EC, Temp");

  // Initialize the RF transmitter
  vw_set_tx_pin(rf_transmitter_pin);
  vw_setup(2000);  // Bits per second
}

void loop() {
  if (ds18b20_read(&raw_temperature))
  {
    water_temperature =
        (float)raw_temperature / 16;
  }
  float raw_ec = analogRead(tds_sensor_pin)
    * analog_reference / 1024.0;
  float temperature_coefficient = 1.0
    + 0.02 * (water_temperature - 25.0);

  electrical_conductivity = (raw_ec
    / temperature_coefficient) * ec_calibration;
  total_dissolved_solids = (133.42 *
  pow(electrical_conductivity, 3) - 255.86 *
  electrical_conductivity *
  electrical_conductivity + 857.39
  * electrical_conductivity) * 0.5;

  if (total_dissolved_solids < 50 ||
    total_dissolved_solids > 700) {
    digitalWrite(buzzer_pin, HIGH);
```

```
    digitalWrite(green_led_pin, LOW);
    digitalWrite(red_led_pin, HIGH);
    delay(300);
  } else {
    digitalWrite(green_led_pin, HIGH);
    digitalWrite(red_led_pin, LOW);
  }

  digitalWrite(buzzer_pin, LOW);

  // Prepare the message to send
  char message[50];
  char ec_string[10];
  char temp_string[10];

  // Convert float to string
  dtostrf(electrical_conductivity, 4,
    2, ec_string);
  dtostrf(water_temperature, 4,
    2, temp_string);

  sprintf(message, "TDS:%d,EC:%s,
    Temp:%s", total_dissolved_solids,
    ec_string, temp_string);

  // Send the message
  vw_send((uint8_t *)message,
    strlen(message));
// Wait until the whole message is gone
  vw_wait_tx();

  // Print data to the serial monitor
  Serial.println(message);

  // Increased delay to
  // reduce transmission frequency
  delay(500);
}

bool ds18b20_start() {
  bool present = 0;
  // Send reset pulse to the DS18B20
  digitalWrite(temp_sensor_pin, LOW);
  pinMode(temp_sensor_pin, OUTPUT);
  delayMicroseconds(500); // Wait 500 us
  pinMode(temp_sensor_pin, INPUT);
  // Wait to read the DS18B20
  delayMicroseconds(100);
  if (!digitalRead(temp_sensor_pin)) {
    // DS18B20 sensor is present
    present = 1;
    // Wait 400 us
    delayMicroseconds(400);
  }
  return present;
}

void ds18b20_write_bit(bool value) {
  digitalWrite(temp_sensor_pin, LOW);
  pinMode(temp_sensor_pin, OUTPUT);
  delayMicroseconds(2);
  digitalWrite(temp_sensor_pin, value);
  delayMicroseconds(80);
  pinMode(temp_sensor_pin, INPUT);
  delayMicroseconds(2);
}

void ds18b20_write_byte(byte value) {
  for (byte i = 0; i < 8; i++)
    ds18b20_write_bit(bitRead(value, i));
}

bool ds18b20_read_bit() {
  bool value;
  digitalWrite(temp_sensor_pin, LOW);
  pinMode(temp_sensor_pin, OUTPUT);
  delayMicroseconds(2);
  pinMode(temp_sensor_pin, INPUT);
  delayMicroseconds(5);
  value = digitalRead(temp_sensor_pin);
  delayMicroseconds(100);
  return value;
}

byte ds18b20_read_byte() {
  byte value = 0;
  for (byte i = 0; i < 8; i++)
    bitWrite(value, i, ds18b20_read_bit());
  return value;
}

bool ds18b20_read(int *raw_temp_value) {
  // Send start pulse
  if (!ds18b20_start())
    return false; // Return false if error
  // Send skip ROM command
  ds18b20_write_byte(0xCC);
  // Send start conversion command
  ds18b20_write_byte(0x44);
  // Wait for conversion complete
  while (ds18b20_read_byte() == 0);
  // Send start pulse
  if (!ds18b20_start())
    return false; // Return false if error
  // Send skip ROM command
  ds18b20_write_byte(0xCC);
```

```
  // Send read command
  ds18b20_write_byte(0xBE);
  // Read temperature LSB byte and
  // store it on raw_temp_value LSB byte
  *raw_temp_value = ds18b20_read_byte();
  // Read temperature MSB byte and
  // store it on raw_temp_value MSB byte
  *raw_temp_value |= (unsigned int)
    (ds18b20_read_byte() << 8);
  return true;
}
```

4.3. Water Level Sensor Code:

```
#include <VirtualWire.h>

#define SensorPin A0
// RF transmitter
#define transmit_pin 12

void setup() {
  // Initialize serial
  // communication at 9600 baud
  Serial.begin(9600);

  // Initialize the RF transmitter
  vw_set_tx_pin(transmit_pin);
  vw_setup(2000);  // Bits per second
}

void loop() {
  // Read the analog sensor value
  int SensorValue =
    analogRead(SensorPin);

  // Calculate the voltage value
  float SensorVolts = SensorValue * 5.0
    / 1023.0;

  // Output the voltage
  // value to the Serial Monitor
  Serial.print("Voltage: ");
  // Print voltage with 3 decimal
  Serial.println(SensorVolts);

  // Prepare the message to send
  char msg[50];
  char voltsStr[10];

  // Convert float to string
  dtostrf(SensorVolts, 4, 3, voltsStr);
  sprintf(msg, "Analog:%d,Volts:%s",
    SensorValue, voltsStr);

  // Send the message
  vw_send((uint8_t *)msg, strlen(msg));
  // Wait until message is gone
  vw_wait_tx();

  // Wait for reading
  delay(1000);
}
```

4.4. Central Node Code:

```
#include <VirtualWire.h>

void setup() {
  // Initialize serial communication
  Serial.begin(9600);
  // Set the receive pin
  vw_set_rx_pin(11);
  // Bits per second
  vw_setup(2000);
  // Start the receiver
  vw_rx_start();
}

void loop() {
  uint8_t buf[VW_MAX_MESSAGE_LEN];
  uint8_t buflen = VW_MAX_MESSAGE_LEN;
  // Check if data is available
  if (vw_get_message(buf, &buflen)) {
    Serial.print("Received: ");
    for (int i = 0; i < buflen; i++) {
    // Print the received message
      Serial.print((char)buf[i]);
    }
    Serial.println();
  }
}
```

5. Conclusion:

Wireless sensor networks and IoT technologies offer promising solutions to many environmental challenges, including water purity monitoring. Our project successfully developed and simulated such a system for real-time water quality assessment. Key achievements include continuous measurement of critical parameters like TDS, temperature, turbidity, and water level using Arduino boards and readily available sensors. This

approach resulted in a cost-effective and scalable solution that allows for easy expansion.

The system's wireless communication setup minimizes maintenance needs, making it suitable for long-term deployment. Overall, our project demonstrates the potential for an efficient and practical IoT-based water quality monitoring system that can significantly enhance water resource management and public health protection.

However, while the technology is resourceful, realizing this system at scale and implementing it with a smart, low-maintenance approach poses significant challenges. The transition from simulation to real-world application requires addressing issues such as sensor durability, power management, data reliability, and network stability in diverse environmental conditions. Despite these challenges, the potential benefits of such systems in improving water quality monitoring and management make further research and development in this area worthwhile.

Future work could focus on miniaturizing the system to integrate it into a water bottle cap. This involves redesigning sensors and circuitry to fit the compact form factor while maintaining functionality, offering consumers a portable, personal water quality monitoring solution. This commercialization effort would require research into miniaturization techniques, power management for extended battery life, and user interface design to ensure effectiveness and user-friendliness. This could potentially open new markets and applications for the technology. This smart water bottle cap can be integerated with a user-friendly mobile app with data visualization, alerts, and GPS-based clean water source locator to enhance usability and provide real-time water quality information to end-users.

References

[1] https://proteustoolkit.org/

[2] https://docs.arduino.cc/software/ide/

[3] https://wiki-content.arduino.cc/en/Guide/Troubleshooting

[4] https://doi.org/10.22214/ijraset.2022.44314