Faculty of Engineering - Cairo University
Credit Hours System –HEM Department
Fall 2023

Final Project
Gastric Lavage Simulator

Team_4_Final_Project

Team Members:

1. Amr Doma ID:1210273
2. Tarek Walid ID: 1210246
3. Youssef Ahmed Afifi ID: 1200883
4. Mahmoud Mohamed Abdelfatah ID: 4220142

Table of contents:

.

1. Introduction

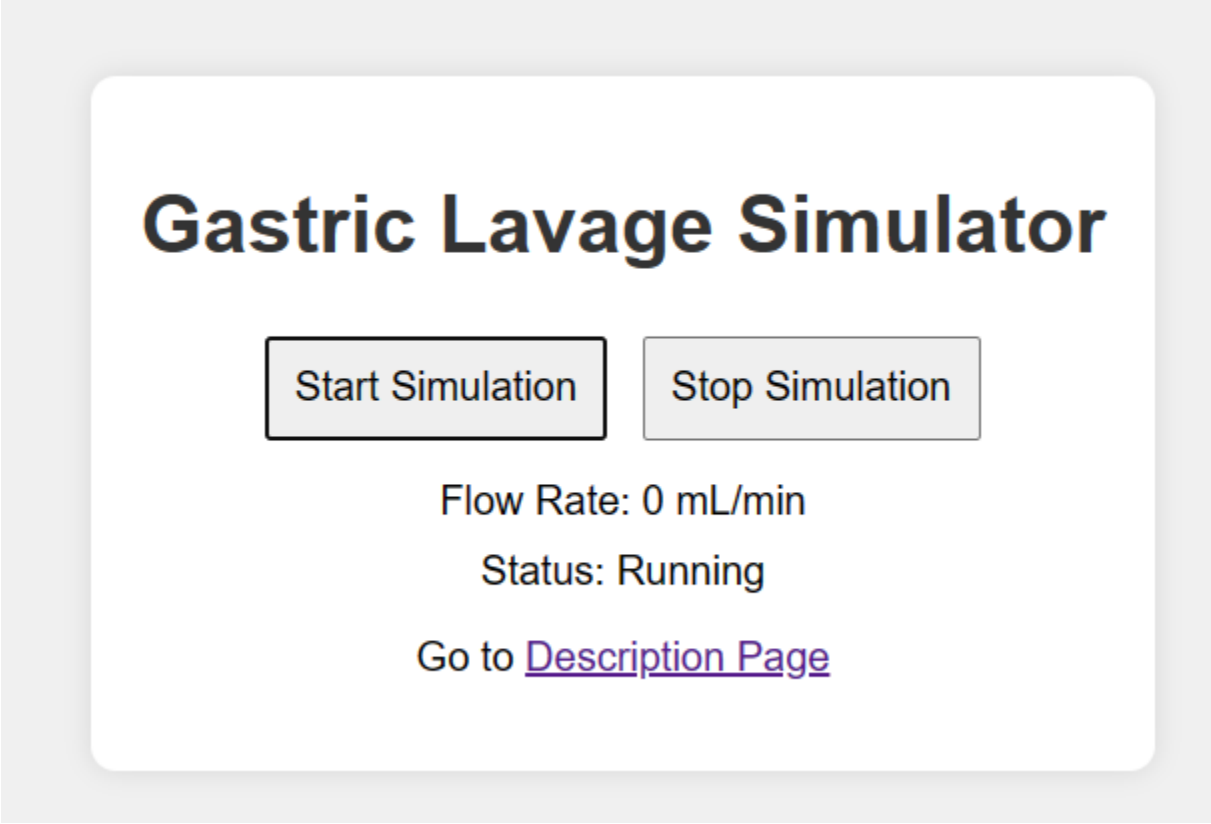
Gastric lavage is a medical procedure used to clean the stomach, typically performed in emergency situations or cases of poisoning. The Gastric Lavage Simulator project aims to create a simulated environment using Arduino to mimic the process for educational purposes. This report provides an overview of the project, including its medical application, hardware prototype screenshots, detailed code explanation, and a discussion of challenges faced and solutions implemented.

2. Medical Application and Significance

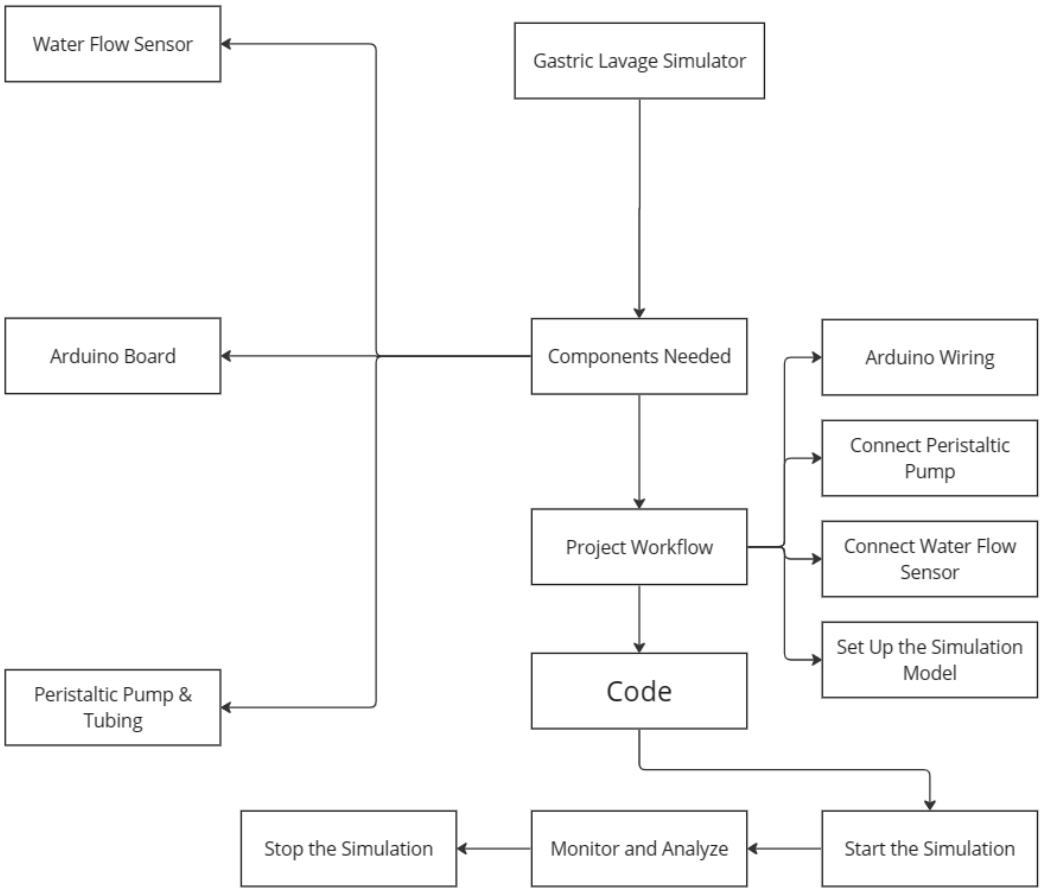
Gastric lavage is a critical medical intervention used to remove harmful substances from the stomach. It is particularly important in cases of ingestion of toxins or poison, helping to prevent the absorption of harmful substances into the bloodstream. The simulator provides a hands-on and visual learning experience for medical students and professionals, allowing them to understand the principles of gastric lavage in a controlled and educational setting.

3. Project Planning :

Web-App

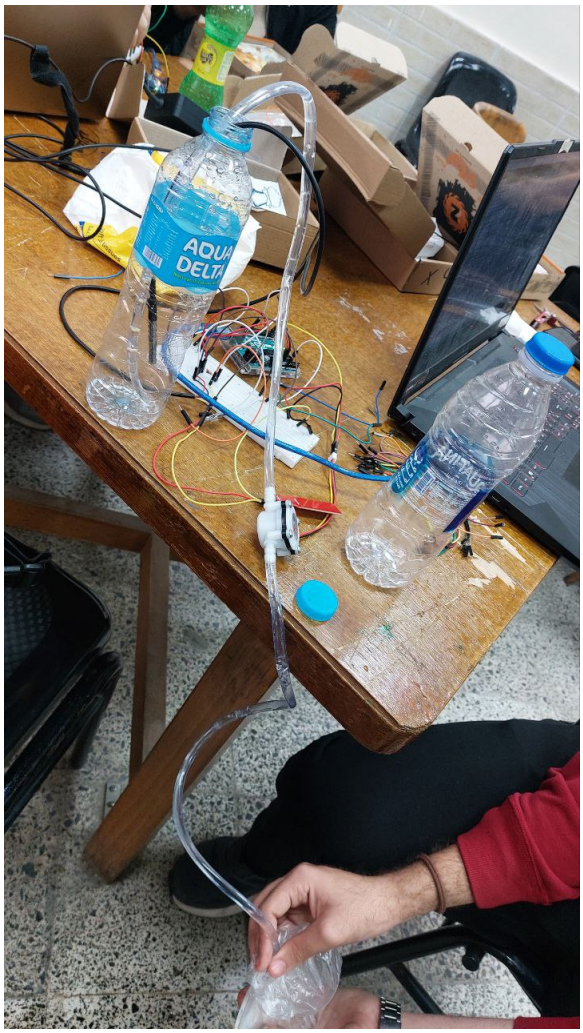


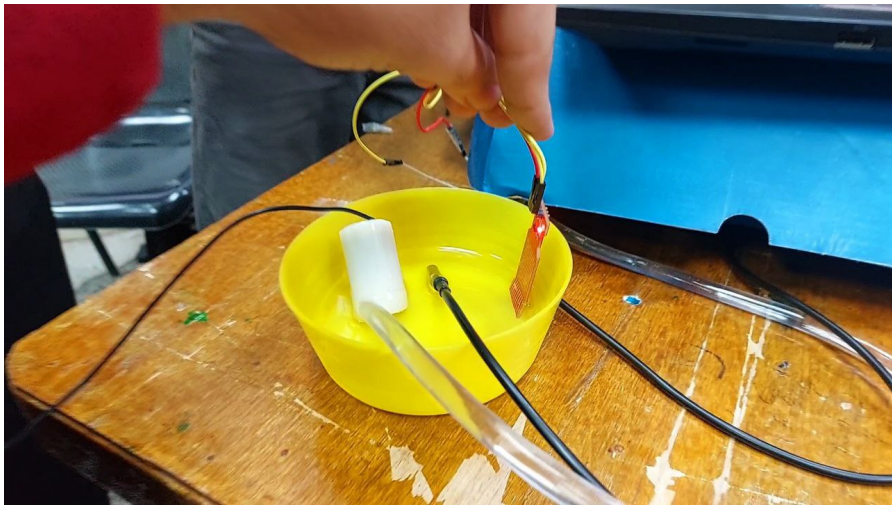
Gastric Lavage Simulator



using [miro]

4. Hardware Prototype Screenshots





5. Detailed Code Explanation

The Arduino code for the Gastric Lavage Simulator is designed to control the water flow, simulate the pumping action, and provide real-time feedback. Below is an overview of key components of the code:

Explanation:

```
// Pin Definitions
#define levelPower 7         // Power pin for water level sensor
#define levelPin A0          // Analog pin to read water level
int val = 0;

// Flow Sensor Pin Definitions
int sensorInterrupt = 0;      // Interrupt 0
int sensorPin = 2;            // Digital Pin 2
const int SENSOR_PIN = 13;    // Arduino pin connected to DS18B20 sensor's DQ pin

// Temperature Sensor Definitions
OneWire oneWire(SENSOR_PIN);         // Setup a OneWire instance
DallasTemperature tempSensor(&oneWire); // Pass OneWire to DallasTemperature library
float tempCelsius;          // Temperature in Celsius
float tempFahrenheit;       // Temperature in Fahrenheit

// Flow Sensor Variables
unsigned int SetPoint = 400;  // 400 milliliters
float calibrationFactor = 90; // Calibration factor for the flow sensor
volatile byte pulseCount = 0; // Pulse count from the flow sensor
float flowRate = 0.0;         // Flow rate in liters/minute
unsigned int flowMilliLitres = 0; // Flow rate in milliliters/second
unsigned long totalMilliLitres = 0; // Total milliliters flowed
unsigned long oldTime = 0;    // Previous time for flow rate calculation
```

1. Pin Definitions:
   - `levelPower`: Power pin for the water level sensor.
   - `levelPin`: Analog pin to read water level.
   - `sensorPin`: Digital pin for the flow sensor.
   - `SENSOR_PIN`: Pin connected to DS18B20 temperature sensor.

2. Temperature Sensor:
   - Uses the DallasTemperature library to interface with a DS18B20 temperature sensor.

3. Flow Sensor Variables:

- `SetPoint`: Desired volume of water for the simulation.
- `calibrationFactor`: Adjusts the flow sensor readings.
- `pulseCount`: Counts pulses from the flow sensor.
- `flowRate`: Calculated flow rate in liters/minute.
- `flowMilliLitres`: Calculated flow rate in milliliters/second.
- `totalMilliLitres`: Cumulative total of milliliters flowed.
- `oldTime`: Records the time for flow rate calculation.

4. Setup Function:
   - Configures pins, initializes sensors, and sets up the interrupt for the flow sensor.

```
void loop() {
  int level = readSensor(); // Read water level
  tempSensor.requestTemperatures(); // Send command to get temperatures
  tempCelsius = tempSensor.getTempCByIndex(0); // Read temperature in Celsius

  // Flow Sensor Processing
  if ((millis() - oldTime) > 1000) {
    detachInterrupt(sensorInterrupt);

    // Flow rate calculation
    flowRate = ((1000.0 / (millis() - oldTime)) * pulseCount) / calibrationFactor;

    // Flow milliliters calculation
    flowMilliLitres = (flowRate / 60) * 1000;

    // Cumulative total milliliters
    totalMilliLitres += flowMilliLitres;

    // Reset pulse counter
    pulseCount = 0;

    // Enable interrupt again
    attachInterrupt(sensorInterrupt, pulseCounter, FALLING);
```

5. Loop Function:
   - Reads water level, temperature, and processes flow sensor data.
   - Prints the values to the serial monitor.

6. Interrupt Service Routine (`pulseCounter`):
   - Increments the pulse count when a falling edge is detected on the flow sensor pin.

```
  // Serial output
  Serial.print("Water level: ");
  Serial.println(level);
  Serial.print("Temperature: ");
  Serial.print(tempCelsius); // Print temperature in Celsius
  Serial.println("°C");
  Serial.print("Flow rate: ");
  Serial.print(flowMilliLitres, DEC);
  Serial.println("mL/Second");
  Serial.println("-------------");
  delay(1000);
}

// Interrupt Service Routine
void pulseCounter() {
  pulseCount++;
}

// Function to read water level sensor
int readSensor() {
  digitalWrite(levelPower, HIGH);
  val = analogRead(levelPin);
  digitalWrite(levelPower, LOW);
  return val;
}
```

7. `readSensor` Function:
   - Reads the water level using an analog sensor.

This code integrates water level sensing, temperature measurement, and flow sensing for a comprehensive simulation setup. Adjustments may be needed based on specific sensor characteristics and simulation requirements.

6. Discussion of Challenges Faced and Solutions Implemented

1. Integration of Multiple Sensors:
   -Challenge: Integrating flow rate, water level, and temperature sensors concurrently on the same Arduino IDE.

- Solution: Ensure that each sensor is properly initialized and connected to its designated pin. Use appropriate libraries for sensor communication. The code provided in the previous response demonstrates the integration of a flow sensor and a temperature sensor. For a water level sensor, consider its specific requirements and adjust the code accordingly.

2. Linking UI with Laptop Port:
- Challenge: Establishing a connection between the user interface (UI) and the laptop port.
- Solution: Ensure that the Arduino board is connected to the laptop via a USB cable. Verify that the correct COM port is selected in the Arduino IDE. Additionally, make sure that the Arduino drivers are installed on the laptop. If the connection issue persists, check the USB cable and try a different one. Ensure that the Arduino board is functioning properly.

3. Collaboration Challenges:
- Challenge: Coordinating the collaboration among team members for efficient project development.
- Solution: Implement version control systems, such as Git, to manage collaborative coding efforts. Use platforms like GitHub for code sharing and version tracking. Clearly define roles and responsibilities within the team. Establish effective communication channels to address challenges promptly. Regularly update team members on progress and issues.

4. Sensor Calibration and Compatibility:
- Challenge: Calibrating sensors to provide accurate and synchronized data. Ensuring compatibility between different sensor outputs.
- Solution: Refer to the datasheets of each sensor for calibration guidelines. Adjust calibration factors in the code to match sensor characteristics. Test sensors individually and collectively to ensure they provide accurate and synchronized data. Fine-tune the code and calibration parameters as needed.

5. Debugging and Serial Monitor:
- Challenge: Identifying and resolving issues using the Serial Monitor for debugging.
- Solution: Use Serial.print() statements strategically in the code to output variable values and debug information. Monitor the Serial Monitor in the Arduino IDE for real-time feedback. Carefully interpret the data to pinpoint issues related to sensor readings, flow rates, or other parameters.

6. User Interface (UI) and Arduino Connection:
- Challenge: Establishing a link between the UI and Arduino to visualize real-time data.
- Solution: Ensure that the UI code communicates effectively with the Arduino code. Verify that the correct COM port and baud rate are set in the UI code. Troubleshoot any issues

with the UI's connection code and serial communication. Test the UI on different browsers if necessary.
Addressing these challenges will contribute to a more seamless integration of sensors and improve the overall functionality of the project.

7. Conclusion

The Gastric Lavage Simulator project successfully creates a simulated environment for educational purposes. By mimicking the gastric lavage process, this simulator provides a valuable tool for medical training. The Arduino-based hardware prototype, coupled with detailed code explanations, enables learners to understand the principles behind gastric lavage in a controlled and safe setting. Challenges faced during the project were addressed with practical solutions, enhancing the overall functionality and educational value of the simulator.

8. References

a. How to insert a nasogastric tube for NG intubation - 3d animation (youtube.com)
b. Cara Mengakses dan Pemrograman Water Flow Sensor YF-S201 Menggunakan Arduino Uno - Arduino Indonesia | Tutorial Lengkap Arduino Bahasa Indonesia
c. Using A Flow Sensor With Arduino - BC Robotics (bc-robotics.com)
d. Gastric lavage • LITFL • CCC Toxicology
e. Using A Flow Sensor With Arduino - BC Robotics (bc-robotics.com)