Mass Flow Meter & Datalogger Scripts

Contents

General Technology

Mass flow meters are widely used instruments in fluid mechanics and engineering applications to measure the mass or volume flow rate of a fluid. The fundamental working principle involves a **low-inertia turbine** placed in the fluid flow path. As the fluid passes through the turbine, it rotates at a speed proportional to the flow rate.

Key Components and Operation

1. **Turbine with Magnetic Ring**: The turbine has a magnetic ring attached to it, with one or more interruptions (gaps) in the magnetic field.
2. **Hall Effect Sensor**: A sensor positioned near the turbine detects the changes in the magnetic field as the turbine rotates.
3. **Pulse Counting**: The sensor generates a series of electrical pulses, where the frequency of the pulses corresponds to the rotational speed of the turbine.
4. **Meter Constant (K-Factor)**: The turbine and sensor assembly has a meter constant

(K-factor), defined as the number of pulses generated per unit of flow. This constant is used to convert the pulse frequency into a mass or volume flow rate.

The output is then processed by a microcontroller or data acquisition system to display the flow rate in appropriate units.

Advantages

- **Accuracy**: Low-inertia turbines minimize lag, allowing precise measurement even at low flow rates.
- **Compact and Scalable**: Mass flow meters are available in a wide range of sizes and capacities, suitable for different applications.
- **Digital Output**: The pulse-based output is easy to integrate with modern control systems and data loggers.

Common Engineering Applications

Mass flow meters are versatile and widely used across industries. Some common applications include:

1. **Water and Wastewater Treatment**

- Monitoring the flow of water, effluent, and chemicals in treatment plants.
- Ensuring compliance with regulatory discharge limits.

2. **Oil and Gas Industry**

- Measuring fuel flow rates in pipelines.
- Monitoring the consumption of lubricants and other fluids in mechanical systems.

3. **HVAC Systems**

- Ensuring accurate flow rates of chilled water, refrigerants, and steam in heating and

cooling systems.
- Optimizing energy efficiency by monitoring flow conditions.

4. **Food and Beverage Industry**

- Measuring the flow of liquids like milk, juice, and syrup in processing plants.
- Monitoring the flow of cleaning agents during sterilization processes.

5. **Automotive and Aerospace Engineering**

- Fuel consumption testing for engines and turbines.
- Flow monitoring in hydraulic and pneumatic systems.

6. **Pharmaceutical and Chemical Manufacturing**

- Precise dosing of chemicals in production lines.
- Monitoring the flow of solvents and active ingredients in formulation processes.

Summary

Mass flow meters offer a reliable and precise solution for measuring fluid flow rates across a broad spectrum of engineering applications. Their ability to provide near real-time digital data and integrate seamlessly with automation systems makes them indispensable in modern industrial processes.

Lab Setup: Raspberry Pi Zero W as Datalogger and Controller

In this lab, we will use a **Raspberry Pi Zero W** as the datalogger and controller, paired with an **inexpensive turbine-type mass flow meter**. The Raspberry Pi Zero W is preconfigured as a WiFi host to simplify connectivity and data acquisition.

Steps to Access the Datalogger

1. **Connect to the Raspberry Pi's WiFi**:
   - Locate the SSID provided in class and connect your device to it.
   - No internet access is required, as this connection is solely for interfacing with the Raspberry Pi.
2. **Access the Raspberry Pi**:
   - Use an SSH client to connect to the Raspberry Pi's IP address (provided in class).
   - Once connected via SSH, start the VNC server using the command:

     ```
     vncserver
     ```

3. **Open the VNC Desktop**:
   - Use a VNC client to connect to the Raspberry Pi's desktop environment.
   - This provides full access to the Raspberry Pi's graphical interface.
4. **Run the Mass Flow Meter Program**:
   - Locate the program file (`/path/filename`, as provided in class) on the Raspberry Pi.
   - Start the program using a terminal command:

     ```
     python3 /path/filename
     ```

   - When prompted, supply:
     - **Meter Constant (K-Factor)**: Provided with the mass flow meter.
     - **Desired Time Interval**: The time in seconds for each reading.

Input Validation

The program is designed with robust input validation using Python's `try-except` blocks to handle invalid or pathological entries, ensuring smooth operation and accurate data logging.

With this setup, the Raspberry Pi Zero W enables efficient and flexible data acquisition from the turbine-type mass flow meter, offering a hands-on experience with both modern

IoT technology and traditional fluid mechanics instrumentation.

- CE-3105-FlowMeter-Datalogger Connection (Component of Laboratory 3) Video showing how to connect to the datalogger.

> **Note**
>
> The **Raspberry Pi Zero W** hosts a short-range WiFi network:
>
> - SSID (network name): **ceceFluids**
> - Passkey (PSK): **OneNekoKat**
> - WAP/Router: **192.168.32.1**
>
> The **Raspberry Pi Zero W** also controls the sensor and records data:
>
> - IP address: **192.168.32.1**
> - Hostname: **pi-lab-1** (connect using IP, DNS is inactive)
> - UID (user): **pi**
> - PWD (passsword): **fluidlab**

Python Scripts for Flow Measurement Lab

Overview

Below are Python scripts provided to document the workflow and enhance the understanding of the lab. These scripts are designed for data logging, processing, and visualization.

Script 1: Datalogger Initialization

Purpose:
This script establishes communication between the Raspberry Pi and the turbine flow meter. It allows users to input key parameters, such as:

- **Meter Constant (K-Factor)**: Provided with the flow meter.

- **Sampling Interval**: The time between successive measurements.

> **Note**
>
> The 2019 script was refactored and reviewed with guidance from OpenAI's ChatGPT. Enhancements included modularization, input validation, time-setting functionality, and general improvements for maintainability and usability. Date of assistance: January 2025.

Once running, the script writes flow rate data to the console.

```
#!/usr/bin/env python
# Flow Meter Recorder
# T.G. Cleveland 2019.
# Refactored and reviewed with guidance from OpenAI's ChatGPT.
# Enhancements included modularization, input validation, time-setting funct

import RPi.GPIO as GPIO
import time
import sys
from datetime import datetime
import subprocess

# Initialize GPIO
def setup_gpio(pin):
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    return pin

# Count pulses
def count_pulse(channel):
    global count
    if start_counter == 1:
        count += 1

# Get validated user input
def get_user_input(prompt, default, value_type, validation=None):
    while True:
        try:
            user_input = input(f"{prompt} (default: {default}): ").strip()
            if not user_input:
                return default
            value = value_type(user_input)
            if validation and not validation(value):
                raise ValueError(f"Input does not meet validation criteria."
            return value
        except (ValueError, TypeError):
            print(f"Invalid input. Using default value of {default}.")
            return default

# Set system time
def set_system_time():
    try:
        new_time = input("Enter new system time (YYYY-MM-DD HH:MM:SS) or pre
        if new_time:
            subprocess.run(["sudo", "date", "-s", new_time], check=True)
            print(f"System time updated to: {new_time}")
        else:
            print("System time unchanged.")
    except subprocess.CalledProcessError:
        print("Failed to update system time. Ensure the script runs with suf

# Write to console and file
```

```
def log(message):
    print(message)
    output_file.write(message + '\n')

# Main Program
print("FlowMeterRecord running")

# User Inputs
output_file_name = input("Enter output file name (default: flowmeter.junk.tx
try:
    output_file = open(output_file_name, 'w')
    log(f"Output will be saved to {output_file_name}")
except Exception as e:
    print(f"Error opening file: {e}")
    sys.exit(1)

stationID = get_user_input("Enter Comment Line 1", "Line 001", str)
sensorID = get_user_input("Enter Comment Line 2", "Line 002", str)
howmany2wait = get_user_input("Enter dwell time", 1.0, float, lambda x: x >
howmany2read = get_user_input("How many intervals", 60, int, lambda x: x >=
meterconstant = get_user_input("Enter the meter constant", 1.0, float, lambd

# Optional: Set system time
set_system_time()

# GPIO Setup
flow_sensor_one = setup_gpio(23)
GPIO.add_event_detect(flow_sensor_one, GPIO.FALLING, callback=count_pulse)

# Initialize counters
count = accCount = howmanyRread = 0

# Print headers
log("# Flowmeter Recording System")
log(f"# {stationID}")
log(f"# {sensorID}")
log(f"# Sampling Interval Duration (seconds): {howmany2wait}")
log(f"# Sampling Interval Count: {howmany2read}")
log("# DateTime, Events, Volume")

# Main loop
try:
    while howmanyRread < howmany2read:
        howmanyRread += 1
        start_counter = 1
        time.sleep(howmany2wait)
        start_counter = 0

        flow = count * meterconstant
        now = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f") if howmany2wai

        log(f"{now}, {count}, {flow}")
        accCount += count
        count = 0
```

```
        log("\nNormal Loop Exit, exiting...")
        log(f"Total Events: {accCount}")
        log(f"Total Volume: {accCount * meterconstant}")

    except KeyboardInterrupt:
        log("\nCaught keyboard interrupt, exiting...")
        log(f"Total Events: {accCount}")
        log(f"Total Volume: {accCount * meterconstant}")
        GPIO.cleanup()
        output_file.close()
        sys.exit()
    finally:
        GPIO.cleanup()
        log("Script execution completed. Closing file.")
        output_file.close()
```

End of Section