Mechatronics System Integration (MCTA3203)

Week **11**: DSP Interfacing with Microcontroller

Signal Filtering and Real-Time Control using Arduino and DSP Principles

Objective: To demonstrate the application of DSP principles in filtering noisy sensor signals and controlling an actuator based on the filtered signal using an Arduino microcontroller.

Concepts Covered:

- Basics of Digital Signal Processing (DSP)
- Signal sampling and quantization
- Noise reduction using filters
- Real-time processing
- Analog-to-digital and digital-to-analog conversion (ADC/DAC)
- Sensor-actuator interfacing

Materials (Hardware) Needed:

- Arduino Board
- TMP36 or LM35 temperature sensor (analog)
- Piezo buzzer or DC motor (actuator)
- Breadboard & jumper wires
- Potentiometer (optional for generating noise)
- Resistors (as needed)

Software Required:

- Arduino IDE
- Serial Plotter (built-in Arduino IDE)
- Python/MATLAB for signal analysis (post-processing)

Background Theory

When using analog sensors, the signal often includes noise. DSP can help clean this signal using algorithms like *moving average* or *IIR filters*. Once a clean signal is obtained, it can be used to trigger an actuator—for example, turning on a fan if the temperature exceeds a threshold.

Experiment Steps:

1. Circuit Setup:
   - Connect TMP36/LM35 analog output to A0 on Arduino.
   - Connect the buzzer/motor to a digital pin (e.g., D9) with a transistor if needed.
   - Power the Arduino and components appropriately.

2. Add Noise:
   - Use a potentiometer or other analog input to simulate noise on the sensor line.
   - Alternatively, shake the sensor or move it near heat/cold sources quickly to simulate variation.

3. Arduino Code: Basic DSP with *Moving Average Filter*:

```
#define SENSOR_PIN A0
#define ACTUATOR_PIN 9
#define NUM_SAMPLES 10
#define TEMP_THRESHOLD 30                    // Celsius

float samples[NUM_SAMPLES];
int index = 0;

void setup() {
  Serial.begin(9600);
  pinMode(ACTUATOR_PIN, OUTPUT);
}

void loop() {
  float raw = analogRead(SENSOR_PIN);
  float voltage = raw * (5.0 / 1023.0);      // Convert to volts
  float temperature = (voltage - 0.5)*100;   // For TMP36

  samples[index] = temperature;
  index = (index + 1) % NUM_SAMPLES;

  // Calculate Moving Average
  float sum = 0;
  for (int i = 0; i < NUM_SAMPLES; i++) {
    sum += samples[i];
  }
  float filteredTemp = sum / NUM_SAMPLES;

  Serial.print("Raw Temp: ");
  Serial.print(temperature);
  Serial.print(" C, Filtered Temp: ");
  Serial.println(filteredTemp);

  // Control actuator
  if (filteredTemp > TEMP_THRESHOLD) {
    digitalWrite(ACTUATOR_PIN, HIGH);
  } else {
    digitalWrite(ACTUATOR_PIN, LOW);
  }

  delay(200);
}
```

Experiment Workflow:

1. Observe noisy vs filtered data using Serial Plotter.
2. Adjust the number of samples in the filter (NUM_SAMPLES) to see responsiveness vs smoothness.
3. Change the *threshold* to see actuator response sensitivity.
4. Optional: Implement a *basic IIR filter* for comparison:

```
float alpha = 0.1;
float filteredTemp = 0;

filteredTemp = alpha*temperature + (1 - alpha)*filteredTemp;
etc..
```

Data Collection and Analysis:

- Understand how DSP principles (filtering) improve signal quality.
- Visualize noise vs. filtered signal in real time.
- Observe reliable actuator response based on filtered data, reducing false triggers.

Further Extensions:

- Use FFT (Fast Fourier Transform) on sampled audio signal (microphone module).
- Send data to Python via serial for advanced DSP.
- Implement PID control using filtered signal.
- Use a digital filter library or design filters in MATLAB and port coefficients to Arduino.

Task

1. Implement a specific type of filter in your signal processing code (e.g., a moving average or low-pass filter). Explain why this type of filter is appropriate for the application, and describe the kind of noise or signal behavior it helps address.

2. Experiment with different filter window sizes in your implementation (e.g., 3-point, 5-point, 9-point). Observe and analyze the effects on the filtered signal. Discuss how the window size impacts the balance between noise reduction and signal delay.

3. Measure and compare the latency of your signal processing system during real-time execution. Identify how real-time processing constraints affect overall system responsiveness, and suggest ways to minimize unwanted delay while maintaining filter performance.

4. Integrate your digital signal processing (DSP) routine into an embedded system (e.g., on an Arduino or similar microcontroller). Reflect on why DSP is critical in such systems, especially in applications that require real-time performance, low power consumption, or limited resources.

Useful Links

[1] https://lastminuteengineers.com/lm35-temperature-sensor-arduino-tutorial/
Interfacing LM35 Temperature Sensor with Arduino

[2] https://www.megunolink.com/articles/coding/3-methods-filter-noisy-arduino-measurements/
Three Methods to Filter Noisy Arduino Measurements

[3] https://github.com/tttapa/Filters; https://github.com/tttapa/Arduino-Filters
Arduino-Filters

[4] https://forum.arduino.cc/t/fir-implementation/465764/18
FIR Implementation

[5] https://ccrma.stanford.edu/~jos/filters/filters.html
INTRODUCTION TO DIGITAL FILTERS WITH AUDIO APPLICATIONS