Project 2: Real-Time Lying Posture Detection

A. System Design

a. Motivation:

Lying posture tracking provides important clinical information about a patient's mobility, risk of developing pressure injuries, and quality of sleep. The goal of this project was to design and implement a simple, real-time embedded system to detect and indicate three distinct lying postures: supine (on back), prone (on stomach), and side (left or right). The system uses the onboard Inertial Measurement Unit (IMU) of an Arduino Nano 33 BLE Sense Rev2 and provides feedback via an LED.

b. High-Level Design:

1. **Sense:** The Bosch BMI270 accelerometer on the Arduino is used to "sense" the board's static orientation. It does this by measuring the constant 1G force of gravity.
2. **Process:** The Arduino's microcontroller runs an "ad-hoc" algorithm developed from experimental data. This algorithm (detailed in Section C) checks the accelerometer's Y and Z axis readings against a threshold to classify the posture.
3. **Act:** An common-anode LED (connected to pin 22) is used to "act." The HIGH state turns the LED OFF, and the LOW state turns it ON. The system indicates the detected posture by blinking the LED in a specific pattern (1 blink for supine, 2 for prone, 3 for side).

c. Observations and Difficulties:

The primary difficulty during design was ensuring consistent sensor orientation. A sensor worn on the chest can be placed in many different ways. To solve this, a key assumption was made: the Arduino is always oriented with the USB port pointing UP towards the user's head. This creates a stable and repeatable frame of reference for the X, Y, and Z axes, making the data predictable.

d. Sampling Frequency:

Two different sampling frequencies were used:

- **Data Collection (data_collection.ino):** A high sampling frequency of **100 Hz** (10ms delay) was used. This was paired with a fast 115200 baud rate to capture high-fidelity, detailed sensor data for analysis.
- **Real-Time Monitoring (real_time_posture_detection.ino):** A lower "meaningful frequency" of **2 Hz** (500ms delay) was chosen for the final application. This rate is fast enough to be responsive to posture changes but slow enough to be power-efficient and allow the LED time to complete its blinking patterns before the next reading.

B. Experiment

a. Experimental Procedure:

To gather "ground truth" data, an experiment was designed to collect sensor readings for four distinct, static postures:

1. **Supine** (On back)
2. **Prone** (On stomach)
3. **Left Side**
4. **Right Side**

The imu_data_collection.ino script was uploaded to the board. For each posture, the board was held steady in the pre-defined orientation (e.g., flat on a desk facing up for "supine") for **one minute**. This process was repeated for **three trials** per posture, resulting in 12 total data files (e.g., supine-trial1.csv, prone-trial1.csv, etc.).

b. Environment and Challenges:

The experiments were conducted on a flat desk to mimic a stable, real-world lying posture. The main challenge was mimicking a "chest band" without one; this was solved by strictly adhering to the "USB port up" orientation assumption. Another challenge was holding the board perfectly still, as even small movements introduced noise, particularly in the gyroscope data.

C. Algorithm

a. Algorithm Design and Observations

The algorithm was designed directly by examining the sensor data collected during the experiment. The 12 .csv files were plotted (see Section D) and analyzed, which revealed a simple, unambiguous pattern.

- **Key Observation (Accelerometer):** The static force of gravity (1G) was isolated to a single axis for each posture:
    - **Supine Data:** The az (Z-axis) column consistently read ~ **+1.0G**.
    - **Prone Data:** The az (Z-axis) column consistently read ~ **-1.0G**.
    - **Left Side Data:** The ay (Y-axis) column consistently read ~ **+1.0G**.
4. **Right Side Data:** The ay (Y-axis) column consistently read ~ **-1.0G**.
- **Key Observation (Gyroscope):** The gyroscope data (gx, gy, gz) was noisy and hovered around zero. Since the gyroscope measures *rate of rotation* and not *static orientation*, it was determined to be irrelevant for this specific problem. The final algorithm **deliberately and exclusively uses the accelerometer**, making it more efficient and stable.

This data-driven observation is the entire basis for the "ad-hoc" algorithm:

```
// 1. Check Z-axis for Supine
if (z > THRESHOLD) {
  blinkLED(1); // Supine
}
// 2. Check Z-axis for Prone
else if (z < -THRESHOLD) {
  blinkLED(2); // Prone
}
// 3. Check Y-axis for Side (Left or Right)
else if (y > THRESHOLD || y < -THRESHOLD) {
  blinkLED(3); // Side
```

```
}
// 4. Otherwise, posture is undefined
else {
  digitalWrite(LED_PIN, LED_OFF); // Undefined
}
```

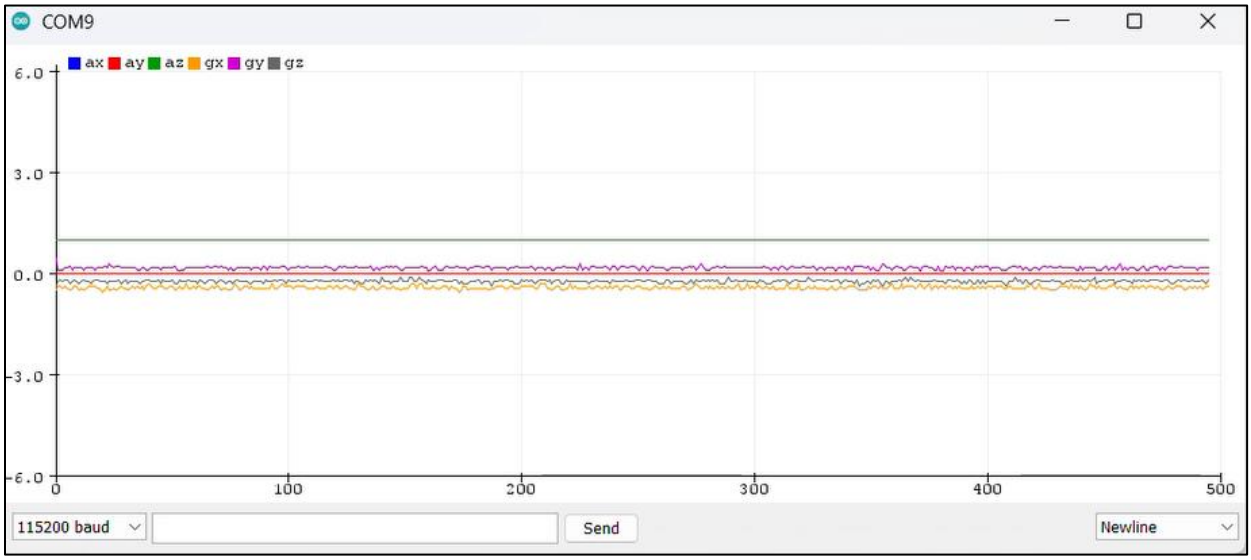
b. Algorithm Success and Robustness:
The approach was highly successful as the postures are clearly separable. The algorithm is made robust by using a THRESHOLD constant of 0.8 instead of 1.0. This allows the system to correctly classify postures even if the sensor is tilted slightly (e.g., propped on a pillow), handling minor real-world variations.

c. Output Frequency
The algorithm generates a new output (or updates the LED state) once every 500ms, or 2 times per second (2 Hz). This is controlled by the LOOP_DELAY constant at the end of the loop() function.

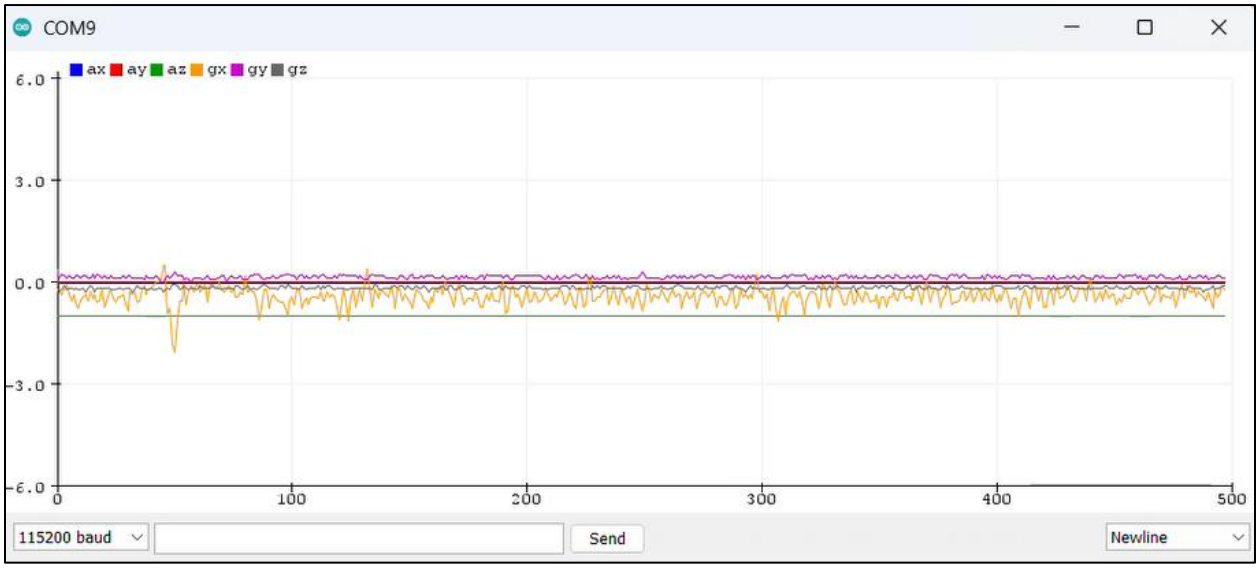
D: Results

a. Sensor Data Plots:
The following plots from the trial1 data files confirm the observations in Section C.

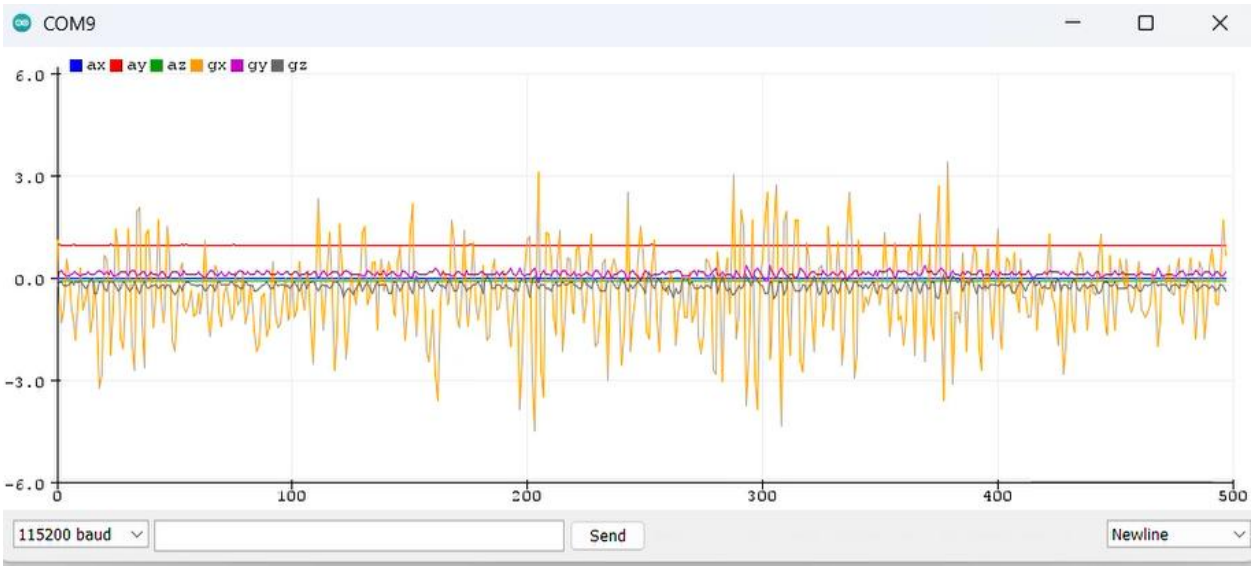
Supine (supine-trial1.csv):
az (green) is stable at +1.0



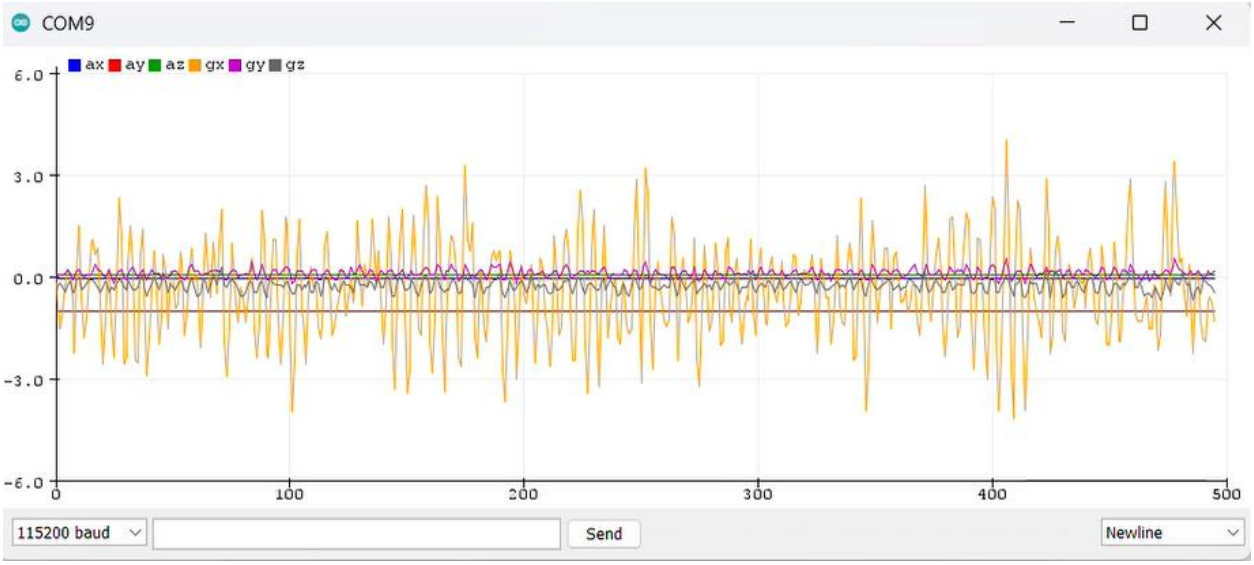
Prone (prone-trial1.csv)
az (green) is stable at -1.0



Left Side (left-side-trial1.csv)
ay (red) is stable at +1.0



Right Side (right-side-trial1.csv)
ay (red) is stable at -1.0



b. System Performance and Accuracy:
The system's performance, as demonstrated in the accompanying demo video, is 100% accurate for the four target postures. The LED blinks the correct number of times for each orientation.

c. Corner Cases:
The system correctly handles its primary corner case: the "undefined" posture.

- **Success:** When the user is sitting up (placing 1G on the X-axis) or at a 45-degree angle, none of the if conditions are met, and the LED correctly stays OFF.
- **Failure:** The system momentarily fails *during* a transition (e.g., while rolling from back to side). The sensor values will pass through the "undefined" region, causing the LED to turn off before blinking the new pattern. This is an expected and acceptable limitation for a static posture detector.

E: Discussions

Summary of Results:
This project was a complete success, culminating in a functional, real-time posture detection system. The final application, running on the Arduino Nano 33 BLE Sense, correctly implements the "sense-process-act" model. It successfully senses orientation using the BMI270 accelerometer, processes this data using a simple, data-driven algorithm, and acts by blinking an external LED in the prescribed patterns. The real-time approach was demonstrably effective, as shown in the demo video, with the system correctly classifying all target postures (supine, prone, and side) as well as the "undefined" corner case.

Difficulties and Challenges:
The most critical and difficult part of the project was Phase 2: Experimentation. Meticulously designing a consistent experiment (the "USB port up" assumption) and carefully logging 12 separate data files was paramount. In contrast, once this high-quality "ground truth" data was

secured and plotted, the algorithm design in Phase 3 became simple and obvious.

Future Improvements:
The system could be extended in two ways:

1. **Transition Detection:** The gyroscope data, which was deliberately ignored, could be used to detect the *act* of rolling over (which would register as a large spike in angular velocity).
2. **Advanced Postures:** To detect more complex postures (e.g., 45-degree recline), a simple machine learning model (like k-Nearest Neighbors) could be trained on the data to classify more than just the four cardinal orientations.

Conclusion

The real-time approach to posture detection proved to be a complete success, with the final system perfectly classifying all target static postures and meeting every project objective. A fundamental takeaway from this project was the critical, task-dependent nature of the sampling frequency. A high rate of **100 Hz** was essential during the data collection phase to capture a high-fidelity signal for analysis, confirming the absence of interfering noise and validating the "ground truth" data. For the final real-time application, a much lower "meaningful frequency" of **2 Hz** (500ms delay) was implemented, which was proven to be more than sufficient for detecting human posture changes, far more efficient, and practically necessary to allow the LED's blinking patterns to complete. This deliberate distinction between sampling for analysis (high detail) versus sampling for deployment (high efficiency) is a core principle of effective embedded design. Ultimately, this project clearly demonstrates that the most robust embedded algorithms are not necessarily the most complex, but are those built from a clear and methodical understanding of sensor data.

References

1. Arduino, "Arduino_BMI270_BMM150 Library," https://www.arduino.cc/reference/en/libraries/arduino_bmi270_bmm150/

Final Source Code

1. imu_data_collection.ino

```
/*
  This program reads accelerometer and gyroscope data from the
  onboard BMI270 IMU and prints it to the Serial Monitor
  in a CSV (Comma-Separated Value) format.
  This data will be used to design our posture detection algorithm.
*/

#include <Arduino_BMI270_BMM150.h> // Include the library for the IMU

// Define the sampling period in milliseconds.
// 1000ms / 10ms = 100 Hz (100 samples per second).
// balance between responsiveness and not flooding the serial port.
const int SAMPLING_PERIOD_MS = 10;

void setup() {
  Serial.begin(115200); // Use a fast baud rate for data transfer
  while (!Serial); // Wait for serial monitor to open

  // Initialize the IMU
  if (!IMU.begin()) {
    Serial.println("Failed to initialize IMU!");
    while (1); // Halt the program
  }

  Serial.println("IMU initialized. Starting data collection...");
  Serial.println("ax,ay,az,gx,gy,gz"); // Print CSV header
}

void loop() {
  float ax, ay, az, gx, gy, gz;

  // Check if both accelerometer and gyroscope data are available
  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()) {
    // Read the sensor data
    IMU.readAcceleration(ax, ay, az);
    IMU.readGyroscope(gx, gy, gz);

    // Print the data as a single CSV line
```

```
    Serial.print(ax);
    Serial.print(",");
    Serial.print(ay);
    Serial.print(",");
    Serial.print(az);
    Serial.print(",");
    Serial.print(gx);
    Serial.print(",");
    Serial.print(gy);
    Serial.print(",");
    Serial.println(gz);
  }

  // Wait for the next sampling interval
  delay(SAMPLING_PERIOD_MS);
}
```

2. Posture_detection.ino

```
/*
* This program uses the onboard BMI270 accelerometer to
 * determine a user's lying posture. It implements an
 * "ad-hoc" algorithm developed by observing collected sensor data.
 * The system indicates the detected posture by blinking an
 * LED connected to pin 22 in specific patterns.
 */

// Include the official Arduino library for the BMI270.
#include "Arduino_BMI270_BMM150.h"

// Define the digital pin number for the LED.
#define LED_PIN 22

// Define the logic levels for our common ANODE LED.
#define LED_ON  LOW
#define LED_OFF HIGH

// Define the posture detection threshold.
// Based on our data, the sensor reads ~1.0G when aligned
// with gravity. We use 0.8 to allow for some tilt/error
// without causing a false negative.
const float THRESHOLD = 0.8;
```

```
// Define the "meaningful frequency" of our algorithm.
// The main loop will pause for this duration.
// This sets our detection rate (1000ms / 500ms = 2 Hz)
// and gives the LED time to blink.
const int LOOP_DELAY = 500;

void setup() {

  Serial.begin(115200);
   while (!Serial);

  // Initialize the IMU sensor.
  if (!IMU.begin()) {
    Serial.println("Failed to initialize IMU!");
    while (1); // Halt program
  }

  // Configure the LED_PIN (Pin 22) as an OUTPUT.
  pinMode(LED_PIN, OUTPUT);

  // Set the initial state of the LED to OFF.
  digitalWrite(LED_PIN, LED_OFF);

  // Print status messages to the Serial Monitor.
  Serial.println("Posture detection system active.");
  Serial.print("Accelerometer sample rate = ");
  Serial.print(IMU.accelerationSampleRate());
  Serial.println(" Hz");
}

void loop() {
  // Declare variables for sensor data.
  // Note: We only need x, y, and z (accelerometer).
  float x, y, z;

  // Check if new acceleration data is available from the IMU.
  if (IMU.accelerationAvailable()) {

    // Read the acceleration data
    IMU.readAcceleration(x, y, z);
```

```
    // This series of if/else if statements classifies the
    // posture based on the data analysis

    if (z > THRESHOLD) {
      // POSTURE: Supine (On Back)
      // Gravity is pulling along the positive Z-axis.
      Serial.println("Supine (Back)");
      blinkLED(1); // Blink once

    } else if (z < -THRESHOLD) {
      // POSTURE: Prone (On Stomach)
      // Gravity is pulling along the negative Z-axis.
      Serial.println("Prone (Stomach)");
      blinkLED(2); // Blink twice

    } else if (y > THRESHOLD || y < -THRESHOLD) {
      // POSTURE: Side (Left or Right)
      // Gravity is pulling along the positive Y-axis (Left)
      // OR negative Y-axis (Right).
      Serial.println("Side");
      blinkLED(3); // Blink three times

    } else {
      // POSTURE: Undefined
      // None of the primary axes are aligned with gravity.
      // (e.g., sitting up, transitioning between postures)
      Serial.println("Undefined");
      digitalWrite(LED_PIN, LED_OFF); // Keep LED off
    }

    // Wait for our defined loop delay. This sets the
    // "meaningful frequency" of our system to 2 Hz.
    delay(LOOP_DELAY);
  }
}

/**
 * Helper function to blink the LED a specific number of times.
 * The number of blinks desired (e.g., 1, 2, or 3).
 */
```

```
void blinkLED(int times) {
  // Loop 'times' number of times
  for (int i = 0; i < times; i++) {
    digitalWrite(LED_PIN, LED_ON);  // Turn LED on
    delay(150); // Keep LED on for 150ms
    digitalWrite(LED_PIN, LED_OFF); // Turn LED off
    delay(150); // Keep LED off for 150ms
  }
}
```