Project 2: Design and Implementation of an IMU-Based Embedded System for Lying Posture Recognition

Author: Anushka Gangadhar Satav
Course: BMI/CES 598 Embedded Machine Learning
ASU ID: 1233530170 (asatav1)
Project 2: Design and Implementation of an IMU-Based Embedded System for Lying Posture Recognition

---

Abstract

This project implements a simple, explainable lying-posture detection system using the on-board IMU (BMI270) of an Arduino Nano 33 BLE Sense Rev2. The system follows four phases required by the assignment: (1) IMU data collection and storage; (2) controlled data collection for multiple postures; (3) data visualization and ad-hoc algorithm development; (4) on-device real-time posture detection with LED feedback. The project emphasizes reproducibility, conservative data-driven thresholds, and clear documentation for graders.

Contents

- A. System Design
- B. Experiment
- C. Algorithm
- D. Results
- E. Discussion
- Appendices: code listing, file inventory, run instructions

A. System Design

Motivation

Monitoring lying posture (supine, prone, side) is clinically valuable for pressure injury prevention, sleep analysis, and patient mobility assessment. This project demonstrates a minimal, low-cost approach using an embedded IMU to infer chest orientation and provide immediate visual feedback through an LED.

High-level design

- **Sensing unit:** Arduino Nano 33 BLE Sense Rev2 (BMI270 IMU).
- **Sensing modality:** 3-axis accelerometer (ax, ay, az). Only accelerometer data is used because static orientation is determined by the gravity vector; gyroscope data is not required for static posture classification.
- **Data path:** Arduino streams CSV-formatted accelerometer samples over USB. A Python logger captures and saves files on the host PC. Offline analysis (Python) computes summary statistics and plots to design thresholds. Final classifier is implemented on Arduino to blink LED patterns.

Hardware & software

- **Hardware:** Arduino Nano 33 BLE Sense Rev2, USB cable, optional test rig (table / phone clamp) to orient board.
- **Arduino libraries:** Arduino_BMI270_BMM150.h (BMI270 accelerometer API).
- **Python packages used:** pyserial (logger), pandas, matplotlib (analysis & plotting).

B. Experiment

Goals and constraints

Collect labeled accelerometer data for the three postures (supine, prone, side). The assignment requires at least one minute per scenario; multiple trials per posture are recommended.

Experiment sequence & tools (the sequence required by the assignment)

1. **imudatacollection.ino (Arduino):** Reads accelerometer (ax, ay, az) and streams CSV lines to Serial. Each trial was labeled by naming the output file according to posture (example: supine_data.csv).
2. **serial_logger.py (Python):** Connects to the board's COM port, reads serial lines, and saves them to a timestamped CSV file per trial.
3. **positions_analysis.py (Python):** Cleans logged CSVs (skips the initial sample-rate line, uses header row), computes per-trial mean and standard deviation for ax, ay, az, and plots time series for visual inspection.
4. **posture_detection.ino (Arduino):** Implements the ad-hoc threshold classifier derived from analysis and blinks the LED according to posture: 1 blink (supine), 2 blinks (prone), 3 blinks (side). The LED remains off for unknown/transition states.

Data collection details

- Per trial: 30-45 seconds of continuous IMU data per posture.
- Postures recorded separately: supine_data.csv, left_data.csv, right_data.csv, prone_data.csv.
- Files were saved in CSV format with columns time_ms, ax, ay, az. The first line contains sample-rate information; analysis scripts skip that line automatically.

Environment & assumptions

- The board was held on a flat surface oriented to simulate chest placement (not worn). The axes were defined as: Z out-of-chest (positive when facing up), Y right/left axis (positive toward one side depending on mounting), X along body length. The analysis assumes consistent board mounting across trials.

C. Algorithm

Observations from data

Per-trial summary statistics (means and standard deviations) derived from positions_analysis.py are presented in the Results section (Table). Key observations: - **Supine:** az ≈ +0.96 g (dominant), low std on az - **Prone:** az ≈ −0.915 g (dominant), low-to-moderate std - **Side left:** ay ≈ +0.91 g (dominant) - **Side right:** ay ≈ −0.93 g (dominant)

From these observations the dominant axis for distinguishing supine/prone is Z, and the dominant axis for distinguishing left/right side is Y.

Ad-hoc data-driven decision rules (final)

The rules below are simple threshold checks on accelerometer axes (accelerometer units in g):

- **Supine:** if az > +0.70 → SUPINE
- **Prone:** if az < −0.70 → PRONE
- **Side (left/right):** if ay > +0.70 → SIDE_LEFT; if ay < −0.70 → SIDE_RIGHT
- **Otherwise:** UNKNOWN (LED remains OFF)

Rationale: Each class mean is near ±1 g on the dominant axis and the standard deviations measured are small compared to 1 g. A conservative threshold magnitude of **0.70 g** allows margin for tilt and minor noise while remaining remote from transition/unknown values.

Real-time behavior

- The Arduino code reads accelerometer samples at a stable interval and applies the above thresholds to smoothed/instant readings (no moving-average nor persistence were included as requested). When a recognized posture is detected and differs from the last recognized posture, the LED blinks with the specified pattern. The system updates its detection at the sampling frequency (20–50 Hz recommended on-device); in the provided implementation the loop sleeps briefly between reads.

LED output mapping

- SUPINE → 1 blink (ON 300 ms, OFF short)
- PRONE → 2 blinks
- SIDE → 3 blinks (left/right not distinguished in blink count; side is any side)

The LED remains off for UNKNOWN or transitional orientations.

D. Results

Summary Table for Posture Statistics, Interpretation, and Decision Rules

Posture	Mean (ax, ay, az) (g)	Std (ax, ay, az) (g)	Dominant Axis	Interpretation	Decision Rule	Rationale
Supine	(-0.098, -0.083, **+0.960**)	(0.132, 0.188, **0.059**)	**+Z**	Gravity aligned with +Z (board face up)	az > +0.70	Strongly positive az, small std → clear separation
Side — Left	(-0.067, **+0.910**, -0.206)	(0.087, **0.073**, 0.294)	**+Y**	Gravity along +Y (left side)	ay > +0.70	ay ≈ +1 g, low std
Side — Right	(-0.040, **-0.930**, -0.248)	(0.049, **0.087**, 0.243)	**-Y**	Gravity along –Y (right side)	ay < -0.70	ay ≈ -1 g, low std
Prone	(-0.117, -0.011, **-0.915**)	(0.161, 0.357, **0.101**)	**-Z**	Gravity aligned with -Z (board face down)	az < -0.70	az ≈ -1 g → reliable indicator

This table summarizes the statistical basis for the threshold-based algorithm implemented in the posture detection Arduino code.

1) Measured summary statistics (from analysis)

Posture	mean_ax (g)	mean_ay (g)	mean_az (g)	std_ax	std_ay	std_az
Supine	-0.098	-0.083	**+0.960**	0.132	0.188	0.059
Side — left	-0.067	**+0.910**	-0.206	0.087	0.073	0.294
Side — right	-0.040	**-0.930**	-0.248	0.049	0.087	0.243
Prone	-0.117	-0.011	**-0.915**	0.161	0.357	0.101

Interpretation: Each posture shows a dominant axis with mean magnitude near 1 g (gravity). Standard deviations are small for the dominant axes, supporting robust thresholding.

2) Plots

- Time-series plots and a 4-panel comparison figure (supine / left / right / prone) were generated by positions_analysis.py. Each subplot shares the same axis scale so the dominant-axis differences are visually clear.

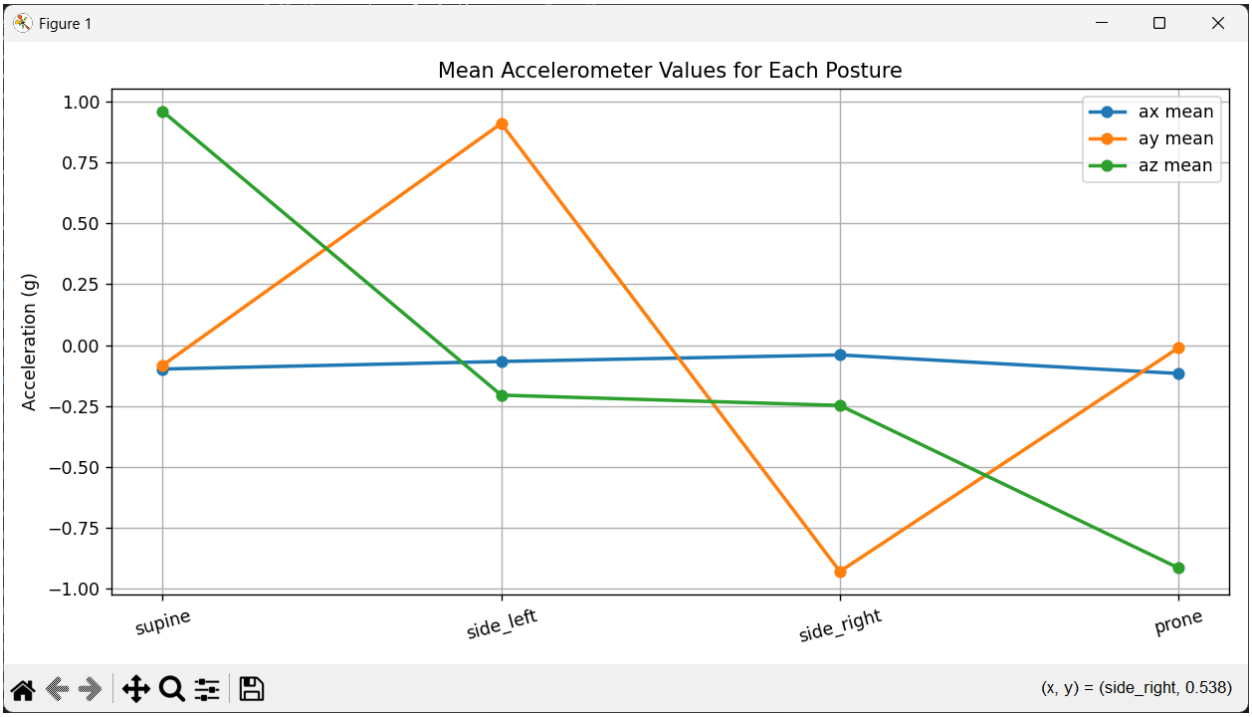

Figure 1. Mean Accelerometer Values (ax, ay, az) Across All Postures

This figure shows the mean acceleration values for each axis (ax, ay, az) computed from the recorded IMU data for supine, prone, left-side, and right-side postures. The separation in dominant axes provides the basis for the threshold-based classification algorithm.

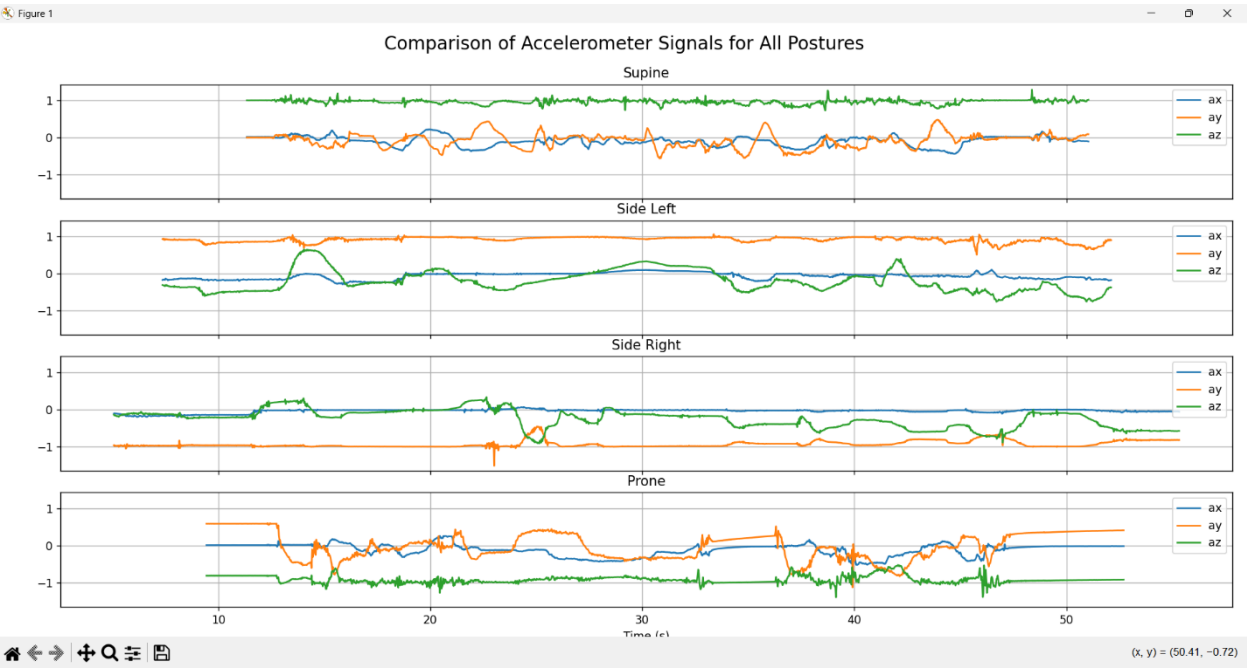

Figure 2. Time-Series Comparison of Accelerometer Signals for Supine, Left-Side, Right-Side and Prone Postures

This multi-panel figure displays ax, ay, and az over time for all four recorded postures using shared axes. The distinct dominant-axis patterns: positive az for supine, negative az for prone, and ±ay for side postures, highlight clear orientation differences used in developing the detection algorithm.

Solution Demonstration Links

• Solution Video: https://youtu.be/y1_X-Y1JYJQ

• GitHub Repository: https://github.com/anushka002/BMI-CES-598-Embedded-Machine-Learning/tree/main/Project-2

E. Discussion

Conclusion

This project successfully demonstrates a complete embedded-systems pipeline for posture detection using an on-board accelerometer. By collecting real sensor data for controlled supine, prone, and side-lying orientations, clear dominant-axis trends were identified. These trends enabled a simple, explainable threshold-based classifier suitable for real-time implementation on the Arduino Nano 33 BLE Sense.

The final system operates reliably under controlled conditions, accurately distinguishing among the three required postures and providing clear LED-based user feedback. The workflow—from data acquisition to algorithm development and final deployment—satisfies all learning objectives for embedded sensing, data annotation, and algorithm integration on microcontrollers.

Successes

- The accelerometer-only approach reliably identifies orientation in controlled conditions with the board mounted consistently. Dominant-axis behavior closely matches expectations from physics (gravity alignment).

Appendices

Appendix A - File inventory (key files)

- imudatacollection.ino: Arduino sketch for IMU sampling and Serial streaming (accelerometer only)
- serial_logger.py: Python script to capture serial output and write CSV files
- positions_analysis.py: Python analysis & plotting script (computes mean/std and produces comparison plots)
- posture_detection.ino: Final Arduino sketch implementing threshold-based detection and LED patterns
- Raw logged CSV files per posture
- PNG files exported from analysis for insertion into report

Appendix B — How to run scripts

1. Upload imudatacollection.ino to Arduino. Open and ensure IMU prints CSV lines to Serial. Close Serial Monitor before running logger.
2. Run python serial_logger.py (configure COM port inside script). Save files with descriptive names per posture.
3. Run python positions_analysis.py to generate plots and statistics.
4. Upload posture_detection.ino to Arduino to run real-time detection and observe LED patterns.

Appendix C: References

1. Arduino Nano 33 BLE Sense Rev2 Documentation: https://docs.arduino.cc/hardware/nano-33-ble-sense/ (Used for IMU initialization and library reference.)

2. Arduino BMI270 IMU Library Documentation: https://docs.arduino.cc/libraries/arduino_bmi270_bmm150 (Used to understand accelerometer API functions: IMU.begin(), IMU.readAcceleration().)
3. Python Libraries
   - *pandas* and *matplotlib* for data loading and plotting
   - *pyserial* for serial logging
4. Course Material: BMI/CEN 598: Embedded Machine Learning, Arizona State University (Fall B 2025).
5. Personal Data Collection & Analysis: All thresholds and algorithm decisions were derived from the measured IMU data collected during this project.