

1. Ubiquitous Computing - Lab 1 : Arduino

Vianney Hervy

This Individual Part details the practical steps taken to implement the two core functionality systems: the posture detection using the IMU and the environmental noise monitoring using the microphone. This work builds directly on the foundational exercises described in the Common Part.

1.1. Posture Detection using the IMU (Practical Task)

The primary goal of the most complex practical task was to accurately detect the board's pitch (forward-backward tilt) to simulate a user's posture.

Steps to Achieve Posture Detection

1. **Sensor Activation and Data Fusion:** We initialized the IMU (LSM6DSOX) to read data from both the accelerometer (which measures the force of gravity and motion) and the gyroscope (which measures rotation speed). To overcome the inherent issues of the accelerometer being noisy and the gyroscope tending to "drift" over time, we used a specific algorithm (the Madgwick filter) to fuse this data. This fusion process is what gives us a stable, reliable angle representing the board's orientation.
2. **Orientation Mapping:** As noted in the Common Part, the biggest challenge was figuring out which physical direction of the board corresponded to the angle we needed. After rigorous testing, we determined that tilting the board across its narrow width provided the most appropriate reading for forward and backward lean.
3. **Threshold Setting:** Once we had a stable angle, we defined the posture zones based on degrees, as measured from the vertical axis:
 - Correct Posture (Green LED): Angles close to 90°.
 - Leaning Forward (Blue LED): Angles dropping below a threshold of 80°.
4. **LED Feedback:** We ensured the calculated angle was continuously checked. Crucially, we implemented non-blocking code (using the `millis()` function) instead of `delay()`. This ensures the posture sensing is always active and doesn't interrupt or freeze other operations, like reading the microphone.

Challenges and Resolution

The main challenge was the Posture detection orientation—matching the physical tilt to the correct sensor axis output. We overcame this by repeatedly testing and logging the raw sensor data until we isolated the specific IMU output axis that registered the pitch movement we wanted to monitor, confirming the accuracy of our angle calculation.

1.2. Environmental Noise Monitoring (Practical Task)

The second critical feature was to detect excessive environmental noise, which acts as a secondary, high-priority warning for the user.

Steps to Implement Noise Warning

1. **Microphone Reading:** The MP34DT06JTR microphone continuously samples the sound intensity around the device. The raw digital readings are very erratic and react to every tiny sound.
2. **Averaging for Stability:** To get a meaningful measure of the background noise, we implemented a process to average the sound readings over a very short time interval. This smoothing process prevented the system from overreacting to short, momentary sounds and provided a stable value for the continuous background noise level.

3. **Noise Threshold and Priority Warning:** Based on calibration in the lab, we defined a numerical noise threshold. If the average sound level consistently exceeded this value, the system immediately triggered the Red LED. This was set as a high-priority warning condition, overriding the posture feedback (Green or Blue) to alert the user to the distracting or excessive noise.

Challenges and Resolution

A notable issue was false alarms caused by brief, loud sounds (like a sneeze, cough, or dropping an item). To solve this, we implemented a “persistence check”: the Red LED would only turn on if the high noise level was detected for a certain number of consecutive readings. This made the system less sensitive to quick spikes and more focused on sustained, disruptive noise levels.

1.3. Theoretical Reflection: Advanced Sensor and Connectivity Features

The final stage of the lab required reflecting on the capabilities of the board beyond the immediate code, specifically the utility of the other main components.

1.3.1. Internal Temperature Sensor

The IMU (LSM6DSOX) also contains a built-in temperature sensor. This sensor was incorporated into an early exercise to provide color-coded feedback: Red for temperatures above 32°C and Green for acceptable temperatures.

- **Calibration Challenge:** The sensor consistently displayed values higher than the ambient room temperature.
- **Resolution:** We applied a fixed offset correction variable in the code to subtract the difference, allowing the LED feedback to accurately represent the actual environment conditions. This step was crucial for ensuring the temperature feature provided meaningful, calibrated data.

1.3.2. Wireless Module and Serial Interface

The Arduino Nano RP2040 Connect features the powerful NINA-W102 module for Wi-Fi and Bluetooth communication, along with the standard Serial interface.

- **NINA-W102 Module (Wi-Fi / Bluetooth):** The module’s primary function in the final product concept is scalability and remote monitoring. While we didn’t implement full network communication, in a complete system, it would be used to:
 - **Data Upload:** Send the posture angle, temperature, and noise status to a remote server or a companion application.
 - **Remote Configuration:** Allow the user to adjust parameters (like the 80° posture threshold) from a phone app.
 - This ensures the core sensor processing remains fast and local, while the results can be shared globally.
- **Serial Monitor:** Throughout the lab, the Serial Monitor was indispensable for debugging and calibration. It allowed us to:
 - **Verify Calculations:** Output the real-time numerical values for the calculated pitch angle and the averaged noise level.
 - **Set Thresholds:** Use the live numerical data to accurately determine the optimal thresholds (e.g., the 80° angle limit or the noise ceiling) needed for the LED feedback logic. The Serial Monitor acts as a necessary visualization tool during development.