Earthquake Detection through High-Pass Filtering of Tectonic Vibrations

Project Report Submitted in Partial Fulfilment of the Requirements for the Degree of

Bachelor of Technology
in
Electronics and Communications Engineering

Submitted by:
Dhruv Prajapat: 23ECE1009
Frederick Barreto: 23ECE1010
Geetanjali Saini: 23ECE1011
Himani: 23ECE1012

Under the Supervision of
Dr. Anirban Chatterjee
Associate Professor

Department of Electronics and Communication Engineering
National Institute of Technology Goa

April, 2025

<u>Done By/Contributions:</u>

1) Dhruv Prajapati: 23ECE1009
    - Python Code For Graphical Visualization of Tectonic Vibrations and conditions required for Earthquake Detection

2) Frederick Barreto: 23ECE1010
    - Arduino Code For Processing Tectonic Vibration and achieving digital High Pass Filtering

3) Geentanjali Saini: 23ECE1011
    - Project Idea, Circuit Design and Analysis of how to use ADXL-345 Accelerometer to obtain Tectonic Vibrations

4) Himani Chauhan: 23ECE1012
    - Connection of Circuit components and calibration of ADXL-345 Accelerometer for suitable output

Earthquake Detection Through High Pass Filtering Of Tectonic Vibrations

Introduction:

Earthquakes are unpredictable natural disasters that can cause widespread damage and loss of life. Traditional seismic detection systems often rely on large-scale infrastructure and may not be responsive enough for real-time local alerts. This project introduces an **Earthquake Detection and Real-Time Visualization System**, which leverages a high-pass filtering approach to detect high-frequency vibrations associated with tectonic activity.

By using the ADXL345 accelerometer, an Arduino Nano, and real-time data plotting through Python, the system identifies rapid changes in acceleration (delta values) to distinguish seismic events from low-frequency background movements. A buzzer and LED alert users locally, while data is simultaneously visualized as both position and delta graphs. This communications-focused project showcases how embedded systems and frequency-domain filtering can work together to enhance early earthquake detection and public safety.

Project Objectives:

- To develop a real-time earthquake detection system using the ADXL345 accelerometer and Arduino Nano by identifying high-frequency tectonic vibrations.
- To apply a software-based high-pass filtering technique that distinguishes seismic activity from slow or irrelevant motion through analysis of delta acceleration values.
- To visualize both positional data and rate-of-change data using Python-based serial communication and live plotting, enabling intuitive monitoring of seismic conditions.
- To enhance local alerting capability by activating a buzzer and LED during potential earthquake events based on dynamic signal analysis.

Components Used:

1. **Arduino Nano Microcontroller**:
   Serves as the brain of the system. It reads acceleration data from the ADXL345 sensor, performs real-time processing (including spike and delta analysis), controls output devices like the buzzer and LED, and sends data via serial communication to a connected computer.

2. **ADXL345 Accelerometer**:
   A three-axis digital accelerometer used to detect movement and vibrations along the X, Y, and Z axes. It communicates with the Arduino via the I2C protocol and enables the system to detect high-frequency motion associated with potential seismic events.

3. **16x2 I2C LCD Display**:
   Used to display real-time acceleration values for all three axes (X, Y, Z). The I2C interface simplifies wiring and reduces the number of pins required, making the system more compact and efficient.

4. **Piezoelectric Buzzer**:
   Provides an audible alert when the system detects an earthquake. The buzzer is triggered by the Arduino based on the frequency of detected vibration spikes.

5. **LED Indicator**:
   A standard LED that lights up alongside the buzzer during earthquake detection. This offers a visual cue in addition to the sound-based alert, making the system more intuitive for users.

6. **Breadboard and Jumper Wires**:
   Used for prototyping and connecting all components in the circuit. These allow quick and flexible assembly of the system without the need for soldering.

7. **USB Cable (for Serial Communication)**:
   Facilitates UART-based serial data transmission between the Arduino and a PC. This cable is also used to power the Arduino and upload the code.

Working Principle:

The system works by using an **ADXL345 accelerometer** to detect vibrations along the X, Y, and Z axes. The accelerometer continuously sends real-time acceleration data to the **Arduino Nano**, which applies a software-based **high-pass filtering technique** by analyzing the **rate of change (delta)** in acceleration values.

This allows the system to distinguish between harmless, slow movements (low-frequency signals) and rapid, high-frequency changes typically associated with seismic events. The system dynamically tracks these changes over short intervals to decide whether an earthquake alert should be triggered.

The earthquake detection logic is based on the following principles:

- **If a high number of large delta spikes are detected in a short time window**, the system classifies this as an earthquake and **activates a buzzer and LED alert**.
- **If only a few or low-magnitude spikes are detected**, the system treats it as minor vibrations and **does not activate any alert**.
- **If no significant vibration is detected**, the system continues to monitor in the background.

The collected data is also transmitted via serial communication to a Python-based plotting system, which displays two types of graphs:

- **Position Plot (X, Y, Z)** showing overall movement.
- **Delta Plot (ΔX, ΔY, ΔZ)** highlighting high-frequency vibration patterns.

This ensures real-time visibility of both general motion and rapid seismic activity, supporting both technical monitoring and effective communication of critical vibration events.

<u>Circuit Design:</u>

The circuit for this earthquake detection and visualization system is composed of the following main components:

- **ADXL345 Accelerometer**: This is connected to the I2C pins of the Arduino Nano (SDA and SCL). It continuously measures acceleration in three axes (X, Y, Z). The sensor outputs digital readings of acceleration, which the Arduino uses to monitor vibration intensity and frequency.
- **Arduino Nano**: Acts as the central processing unit of the system. It collects acceleration data from the ADXL345, applies a high-pass filtering logic by computing delta changes in acceleration, and decides whether to trigger an alert.
- **16x2 I2C LCD Display**: Displays real-time X, Y, and Z positional values. It is connected to the Arduino via I2C (typically A4 for SDA and A5 for SCL), allowing for efficient data display with minimal wiring.
- **Buzzer**: A piezoelectric buzzer is connected to a digital output pin of the Arduino. It is activated when the system detects seismic activity exceeding a predefined threshold.
- **LED**: An LED is used alongside the buzzer to visually indicate when an earthquake event is detected. It is connected to another digital output pin of the Arduino.

- **Connections**: All components are connected on a breadboard. The Arduino Nano provides the necessary power through its 5V and GND pins. Jumper wires are used to connect the accelerometer, LCD, buzzer, and LED to the appropriate pins on the Arduino Nano.

Circuit Diagram

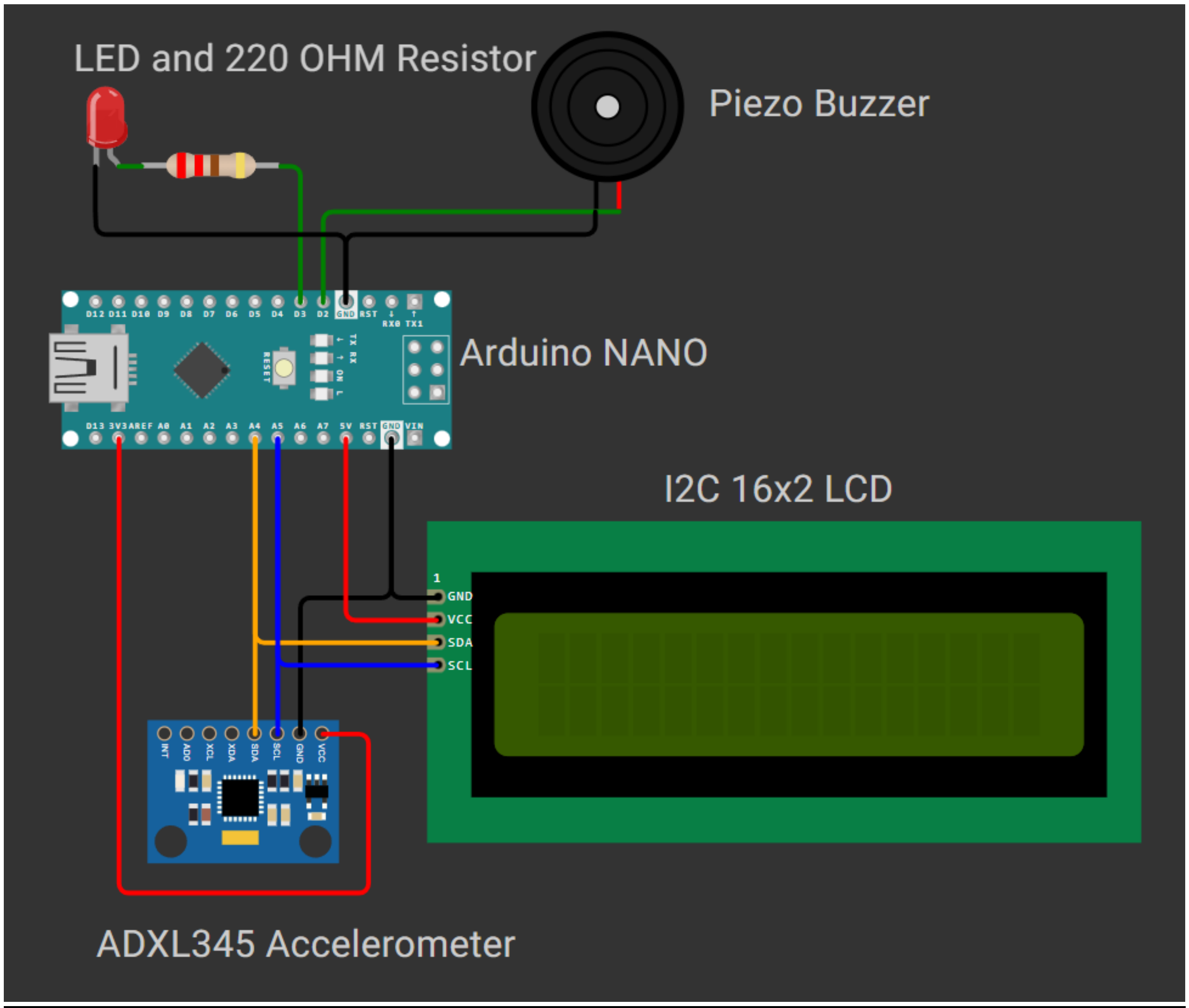


Fig. 1(Circuit Diagram)

System Design:

The design of the system consists of both hardware and software components.

- **Hardware Design**: The main hardware consists of the **Arduino Nano** as the central controller, an **ADXL345 accelerometer** for vibration detection, a **16x2 I2C LCD display** for showing real-time axis readings, a **buzzer** and an **LED** for alerting, and serial communication to a computer for live graph visualization via Python.
- **Software Design**: The Arduino code is responsible for managing the system logic, including:
  - **Reading accelerometer data** via I2C and calculating current acceleration on the X, Y, and Z axes.
  - **Computing delta acceleration** (rate of change) over time to simulate a high-pass filter and detect high-frequency seismic activity.
  - **Triggering alerts** through a buzzer and LED when rapid vibration spikes occur within a defined time window.
  - **Displaying X, Y, Z values** on the LCD screen in real-time.
  - **Transmitting both position and delta values** over serial for live graph plotting using Python.

The **flowchart** for the system is as follows:

1. **Start**: Initialize all components (ADXL345 sensor, LCD, buzzer, LED, serial communication).
2. **Sensor Input**: Continuously read X, Y, Z axis data from the accelerometer.
3. **Delta Calculation**: Compute changes in acceleration (ΔX, ΔY, ΔZ) and track spike frequency.
4. **Trigger Alert**: If the number of spikes exceeds the threshold within a time window, activate the buzzer and LED.
5. **Data Transmission**: Send both position (POS:x,y,z) and delta (DEL:dx,dy,dz) data to the computer for Python-based real-time graph plotting.

Arduino Code Explanation:

The Arduino code implements the system logic described in earlier sections. It starts by setting up the pins for the buzzer, LED, and initializing communication with the ADXL345 accelerometer and the LCD display. The main functions carried out by the code are as follows:

- **Pin Initialization**: The Arduino initializes digital output pins for the buzzer and LED. The accelerometer and LCD display are I2C-based and are initialized through their respective libraries. This allows the sensor to begin transmitting data and the LCD to begin displaying output.
- **Reading Sensor Inputs**: The accelerometer is polled at regular intervals to collect acceleration values along the X, Y, and Z axes. These raw values are adjusted by subtracting baseline readings taken during calibration, resulting in centered motion data. This helps eliminate static bias due to gravity or surface tilt.
- **Calculating Delta Acceleration**: The Arduino calculates the difference between the current and previous values of acceleration on each axis. These changes are referred to as delta X, delta Y, and delta Z. This delta-based calculation mimics the function of a high-pass filter by isolating high-frequency vibrations from slow or gradual movement. If any of these delta values exceed a predefined threshold, a spike is counted.
- **Determining Earthquake Events**: A time window is established (typically one second), during which spikes are counted. If the number of spikes surpasses a specified limit, the system interprets this as an earthquake event and activates an alert. Otherwise, the system continues to monitor passively without interruption.
- **Triggering Alerts**: When an earthquake is detected, the Arduino activates the buzzer using the tone function and illuminates an LED. Both remain active for a fixed duration, after which they are automatically turned off to avoid continuous alerts.
- **Displaying Real-Time Data**: The LCD screen displays the latest positional values (X, Y, Z) so the user can observe motion in real time. This is useful for debugging and real-world demonstrations.
- **Sending Data to Python**: The Arduino transmits both types of data—position and delta acceleration—over serial communication. Each line of data is labeled accordingly. Lines beginning with "POS:" contain the position data (X, Y, Z), and lines beginning with "DEL:" contain the delta values (delta X, delta Y, delta Z). This formatting allows separate Python scripts to parse and visualize the respective data streams in real time.

Code:

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL345_U.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);
Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345);

#define buzzer 2
#define led 3

#define THRESHOLD 15
#define WINDOW_DURATION 1000
#define REQUIRED_SPIKES 8
#define samples 50
#define sampleInterval 100

int xBase = 0, yBase = 0, zBase = 0;
int prevX = 0, prevY = 0, prevZ = 0;
int spikeCount = 0;
unsigned long windowStartTime = 0;
unsigned long lastSampleTime = 0;
bool alertActive = false;
unsigned long alertStart = 0;

void setup() {
  Serial.begin(9600);
  lcd.init();
  lcd.backlight();

  pinMode(buzzer, OUTPUT);
  pinMode(led, OUTPUT);
  digitalWrite(led, LOW);
  noTone(buzzer);

  lcd.print("Initializing...");
  if (!accel.begin()) {
    lcd.clear();
    lcd.print("Sensor Error!");
    while (1);
  }
  delay(500);
  lcd.clear();
  lcd.print("Calibrating...");
  lcd.setCursor(0, 1);
  lcd.print("Please wait...");

  int sumX = 0, sumY = 0, sumZ = 0;
  sensors_event_t event;

  for (int i = 0; i < samples; i++) {
    accel.getEvent(&event);
    sumX += event.acceleration.x * 10;
    sumY += event.acceleration.y * 10;
    sumZ += event.acceleration.z * 10;
    delay(20);
  }

  xBase = sumX / samples;
  yBase = sumY / samples;
  zBase = sumZ / samples;

  prevX = xBase;
  prevY = yBase;
  prevZ = zBase;

  lcd.clear();
  lcd.print("Calibrated");
  delay(1000);
  lcd.clear();
  lcd.print("Monitoring");

  windowStartTime = millis();
}

void loop() {
  if (millis() - lastSampleTime >= sampleInterval) {
    lastSampleTime = millis();

    sensors_event_t event;
    accel.getEvent(&event);

    int rawX = event.acceleration.x * 10;
    int rawY = event.acceleration.y * 10;
    int rawZ = event.acceleration.z * 10;
    int currX = rawX - xBase;
    int currY = rawY - yBase;
    int currZ = rawZ - zBase;

    int deltaX = abs(currX - prevX);
    int deltaY = abs(currY - prevY);
    int deltaZ = abs(currZ - prevZ);

    prevX = currX;
    prevY = currY;
    prevZ = currZ;

    if (deltaX > THRESHOLD || deltaY > THRESHOLD || deltaZ > THRESHOLD) {
      spikeCount++;
    }

    lcd.setCursor(0, 1);
    lcd.print(padValue(currX));
    lcd.setCursor(6, 1);
    lcd.print(padValue(currY));
    lcd.setCursor(11, 1);
    lcd.print(padValue(currZ));

    Serial.print("POS:");
    Serial.print(currX); Serial.print(",");
    Serial.print(currY); Serial.print(",");
    Serial.println(currZ);

    Serial.print("DEL:");
    Serial.print(deltaX); Serial.print(",");
    Serial.print(deltaY); Serial.print(",");
    Serial.println(deltaZ);
  }

  if (millis() - windowStartTime >= WINDOW_DURATION) {
    if (!alertActive && spikeCount >= REQUIRED_SPIKES) {
      lcd.clear();
      lcd.print("Earthquake !!!!");
      digitalWrite(led, HIGH);
      tone(buzzer, 1000);
      alertStart = millis();
      alertActive = true;
    }
```

```
    spikeCount = 0;
    windowStartTime = millis();
  }

  if (alertActive && millis() - alertStart >= 3000) {
    digitalWrite(led, LOW);
    noTone(buzzer);
    lcd.clear();
    lcd.print("Monitoring");
    alertActive = false;
  }
}

String padValue(int value) {
  String strValue = String(value);
  while (strValue.length() < 4) {
    strValue += " ";
  }
  return strValue.substring(0, 4);
}
```

Working of the System:

The Earthquake Detection and Real-Time Visualization System is designed to identify seismic activity by analyzing the frequency of vibrations occurring in the environment. At the core of the system is the ADXL345 accelerometer, which continuously measures acceleration along the X, Y, and Z axes. These measurements are captured in real time by the Arduino Nano, which serves as the central processing unit of the system. Rather than simply looking at absolute acceleration values, the system calculates how these values change over time — focusing on the **rate of change**, or **delta**, to emphasize rapid movement patterns.

To mimic the effect of a high-pass filter, the system computes the difference between each new accelerometer reading and its preceding value on each axis. This technique effectively filters out low-frequency noise and minor vibrations, such as subtle hand movements or gentle table shifts. Instead, it highlights high-frequency motion patterns, which are more likely to resemble the sharp, sudden shocks that occur during an earthquake.

When the delta acceleration values surpass a specific threshold, the system interprets that as a "spike." These spikes are counted within a short, fixed time window (typically one second). If the number of spikes in that time exceeds a predefined limit, it is considered an earthquake event. At that point, the Arduino activates a local alert mechanism: a buzzer sounds and an LED lights up to warn users of the detected vibration surge.

To help visualize the logic, the system also transmits position and delta data to a Python script that plots the results in real time. The graph below shows a typical reading during low-frequency movement. As seen here, the acceleration along all three axes forms a smooth, wave-like pattern. This behavior is characteristic of natural or slow-paced human movement, and it does not cross the threshold required to trigger an alert.

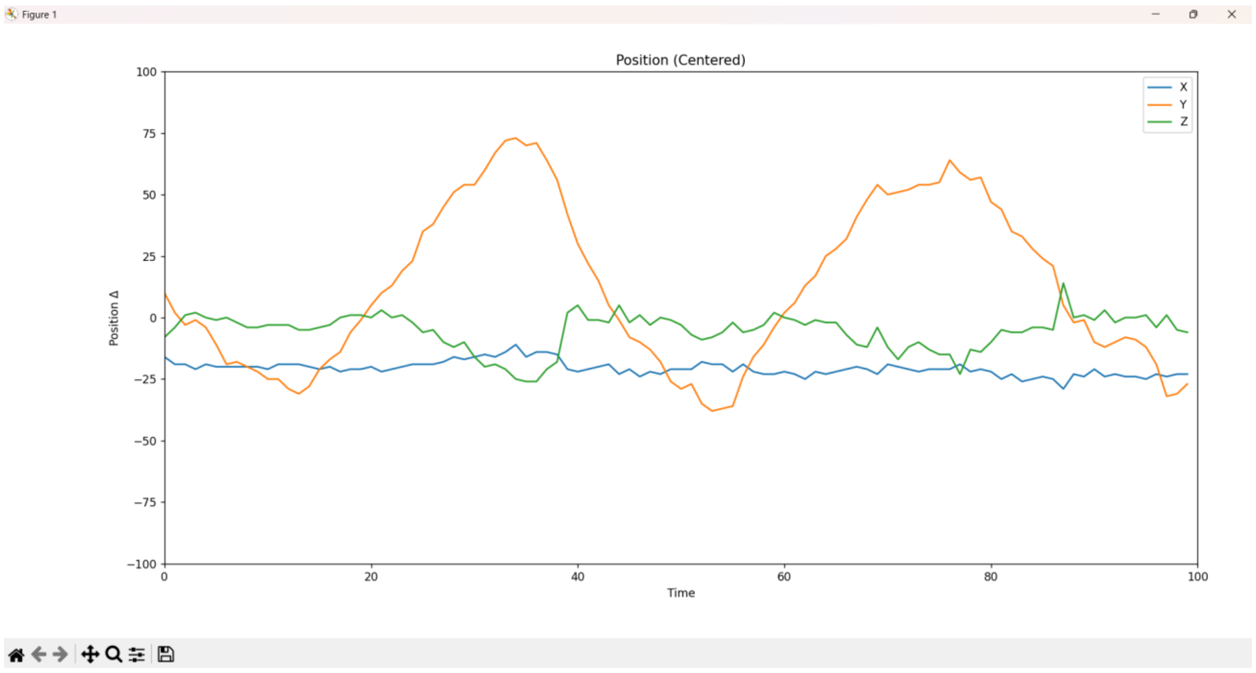
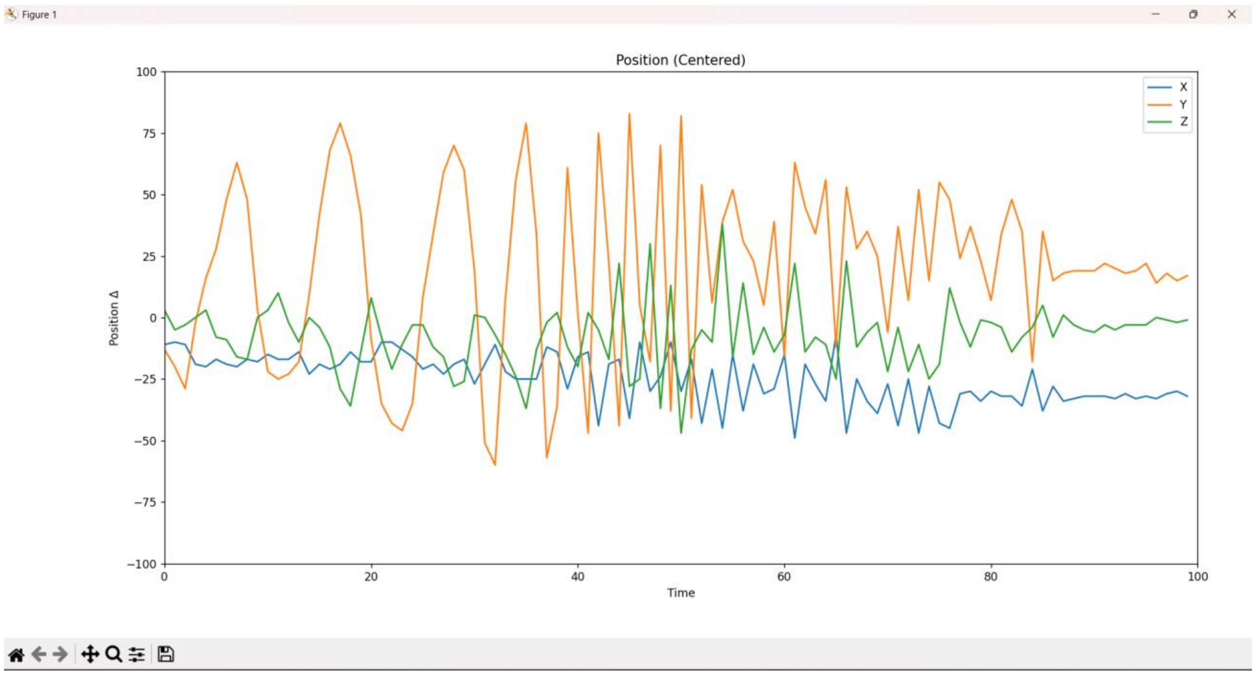


Fig. 2(No Earthquake)

In contrast, the following graph presents a case of high-frequency vibration. The waveform becomes sharper and more erratic, with frequent fluctuations that exceed the delta threshold. This is the exact kind of pattern the system is designed to detect and respond to. In such a case, the spike counter rapidly increases, leading to the activation of the earthquake alert through the buzzer and LED.



<u>Fig. 3(Earthquake Detected)</u>

Throughout this process, a 16x2 I2C LCD display is used to show the current X, Y, and Z values, allowing the user to observe motion directly on the device. Combined with the Python-based plots, the user receives both visual and auditory cues of potential seismic activity. This approach effectively combines real-time sensor monitoring, software-based signal filtering, and intuitive communication methods to detect and visualize earthquake-like conditions.

Concepts Of Communication Systems Used:

1) High-Pass Filtering via Software-Based Delta Analysis:
Traditional communication systems often rely on filtering techniques to isolate meaningful signals from background noise. In this project, a software-implemented high-pass filter is realized through the calculation of delta values — that is, the rate of change in acceleration across the X, Y, and Z axes. Instead of analyzing raw acceleration data, the Arduino compares each new value with the previous one to compute how quickly the signal is changing. Low-frequency, gradual motion (like slow tilts or surface vibration) results in small delta values and is ignored. High-frequency movement, however, produces large spikes in delta values, which the system recognizes as potential indicators of an earthquake. This approach effectively filters out irrelevant data and only communicates critical high-frequency vibration information, enhancing the system's sensitivity and accuracy in detecting seismic activity.

2) Serial Communication for Real-Time Data Visualization:
The system utilizes UART-based serial communication to transfer processed motion data from the Arduino Nano to a computer running a Python visualization script. This is a foundational communication system concept where one device (transmitter) sends structured, real-time data to another device (receiver) over a serial line. The Arduino formats outgoing data using specific tags like "POS" for position and "DEL" for delta, ensuring clarity in data framing — a technique common in digital communication protocols. The receiving Python program interprets these tags and displays live graphs, providing the user with a continuous visual understanding of the motion profile. This not only represents machine-to-machine communication but also supports human-readable communication, bridging hardware sensing with intuitive, graphical outputs.

Future Scope and Enhancements:

- **Integration with IoT Platforms:**
  The system can be extended to transmit seismic activity data to an online server or cloud dashboard. This would enable remote real-time monitoring, long-term storage, and centralized alert management, enhancing the system's scalability for large-area deployment.
- **GPS and Location Tagging:**
  Incorporating GPS modules can allow the system to geo-tag seismic data. This could help build distributed earthquake monitoring networks where each node contributes localized data, increasing the accuracy of event localization.
- **Wireless Communication Protocols:**
  Future versions can replace serial USB communication with wireless options like Wi-Fi or LoRa. This would allow the sensor modules to operate in the field without needing a computer, supporting independent and low-power deployment in remote areas.

- **Adaptive Thresholding using Machine Learning:**
  By integrating simple machine learning algorithms, the system could learn from past vibrations and self-adjust its spike threshold over time. This would reduce false positives and increase the reliability of earthquake detection under varying environmental conditions.
- **Mobile App Notification System:**
  A companion mobile app could be developed to receive alerts whenever an earthquake is detected. This enhances the communication aspect by allowing notifications to reach users even if they are not near the device physically.

Conclusion:

The **Earthquake Detection and Real-Time Visualization System** presents an efficient and accessible method of identifying high-frequency seismic activity using an embedded communication-based approach. By utilizing the ADXL345 accelerometer and Arduino Nano, the system captures motion data in real time and applies a high-pass filtering logic to detect sudden vibration spikes. This enables accurate detection of earthquake-like events while filtering out irrelevant motion. Through UART serial communication, the system provides real-time feedback in the form of dynamic graph plots and visual LCD readings. Additionally, local alerts using a buzzer and LED offer immediate tactile and auditory signals to users.

This project highlights how core communication system principles — including digital sensor interfacing, threshold-based decision logic, serial data transmission, and visual signaling — can come together to solve a real-world problem. With further integration into IoT platforms, wireless communication, and adaptive algorithms, this system has the potential to evolve into a smart, distributed, and fully autonomous earthquake monitoring solution.