Technica®

Technica Corporation

Vibrational Anomaly Detection Using Tri-Axial Accelerometer Data

Version 1.0

July 30, 2018

Technica Corporation
22970 Indian Creek Drive, Suite 500
Dulles, VA 20166
703.662.2000 phone
703.662.2001 fax

technicacorp.com

Table of Contents

Technica Corporation Contact Information

Author: Heather Whittaker
Address: 22970 Indian Creek Drive, Suite 500
City, State, Zip: Dulles, VA 20166
Phone: 703-622-6072

1 Introduction

From bridges to cranes to servers to trains, the modern world is filled with essential equipment and infrastructure. Despite society's reliance, breakages endanger safety, halt operations, and incur high financial costs. Preventative maintenance involves detecting degradation or faults *before* disaster strikes.

One of the best approaches to preventative maintenance is monitoring vibrational patterns during usage. In fact, factories actually hire employees to evaluate assembly lines though feel and sound to detect changes in vibrations. However, machine learning combined with Internet of Things (IoT) technology could allow for sensitive analytics on easily deployable, inexpensive, small form-factor devices. One notable application would be military aircraft – sensors on the engine could diagnose potential engine failure, while sensors on the hull could indicate a breach.

This Vibrational Anomaly Detection project relies on extreme memory efficiency to fit onto highly constrained devices. Many other machine learning algorithms are too resource intensive to fit inference on severely resource-constrained devices. However, this project fits both the training and inference phases on a true edge device while running analytics in real time. This code is original and not an adaptation of an open-source project. The development and testing was run on the BBC Micro:bit. This device runs at only 16MHz with only 16kb of RAM, classifying the device as a Class I IoT device.

2 Code Overview

An accelerometer captures tri-axial accelerometer data accounting for the x, y, and z dimensions in a series of time windows. These data points within each window are manipulated into a collection of features used to detect anomalies. Features include averages, mins, maxes, and frequencies for x, y, and z data as well as magnitude, which takes each dimension into consideration.

To fit within the tight memory constraints, training is broken into two phases. Phase I collects data for a configurable number of training windows and computes the averages. Phase II uses these averages and, using incoming data, computes the standard deviations for each feature. Again, the number of training windows is configurable. Finally, an infinite inference loop analyzes new time windows, computes averages, and determines how many standard deviations the new value is from the trained average. If this z-score value is greater than a threshold, that feature is marked as suspicious. If the number of suspicious features exceeds a threshold, an anomaly is reported.

In this project, an anomalous event triggers a radio transmission. This message is received by another Micro:bit, which flashes its LED grid to notify a human.

3 Features

During runtime, the accelerometer captures raw acceleration data in the x, y, and z dimensions. The algorithm computes features for each so as to detect potential changes in movement in any dimension. The average, minimum, maximum, and frequency are calculated for each axis independently.

These features were selected to encompass any type of change within vibrational waves. Computing averages as features reveals general trends within the data. There are two ways in which alterations to the energy within the system can manifest: magnitude and frequency. The maximum and minimum features detect alterations to magnitude while the frequency detects changes in the number of times the value crosses the average.

4 Training and Inference

One challenge of vibrational anomaly detection is that vibrations by nature cause data values to fluctuate over time. These fluctuations may be high or low in magnitude depending on the application, and the

machine learning algorithm must successfully adapt. In this project, "normal" or acceptable values are determined in a two-step training process. First, the code runs a configurable number of loops collecting the averages for each feature. Then, the code runs a second configurable set of loops that computes the standard deviation from the distribution of each feature.

The inference phase involves an infinite loop that computes each feature and performs subsequent anomaly detection. Prior-determined means and standard deviations are used to perform z-score analysis. If the average of a feature within the time window is more than a configurable number of standard deviations from the mean, the feature is marked as suspicious. If a window has over a configurable number of suspicious features, the window is marked as anomalous.

Table 1. Equations

Function	Description	Variables
Magnitude	$mag = \sqrt{x^2 + y^2 + z^2}$	x, y, and z represent axis data
Standard Deviation	$\sigma = \sqrt{\frac{\sum(x - \bar{x})^2}{n - 1}}$	x represents new data, $\bar{x}$ the sample mean, and n the number of data samples
Z-Score	$z = \frac{x - \mu}{\sigma}$	x represents new data, μ the population mean, and σ the prior-computed standard deviation

5 Anomalous Alert

Upon detecting an anomaly, the sensing Micro:bit transmits an anomalous message over radio. Another Micro:bit listens for the message and flashes its LED grid. Thus, a human can be alerted in real time.

The BBC Micro:bit also offers Bluetooth capability, which in theory could be linked with a cell phone for alerts. However, Bluetooth is only supported when developing in C++, since the Bluetooth Low Energy (BLE) stack is too large to fit on the Micro:bit device alongside micro-python.

6 Data Structures

Extreme memory constraints necessitate strategic data structure usage. Storing all of the features intuitively risks overwhelming the available memory; hence variables are stored in arrays which are repurposed through the code. Another concern was the actual file length, which can cause memory errors. As such, one-dimensional lists were frequently selected over 2D arrays to facilitate looping, thus reducing number of lines and the file size to allow the code to run on the Micro:bit.

Since arrays are reused, names are short and unintuitive, and the looping structures required to shorten the code are difficult to follow, use the table below as a guide to tracking computation through the code.

Table 2. Feature Storage

Feature	Current Values	Means	Stdevs	z-scores
X max	n_v[0]	n_avgs[0]	stdevs[0]	z_score[0]
X min	n_v[1]	n_avgs[1]	stdevs[1]	z_score[1]
X avg	n_v[2]	n_avgs[2]	stdevs[2]	z_score[2]
Y max	n_v[3]	n_avgs[3]	stdevs[3]	z_score[3]

Feature	Current Values	Means	Stdevs	z-scores
Y min	n_v[4]	n_avgs[4]	stdevs[4]	z_score[4]
Y avg	n_v[5]	n_avgs[5]	stdevs[5]	z_score[5]
Z max	n_v[6]	n_avgs[6]	stdevs[6]	z_score[6]
Z min	n_v[7]	n_avgs[7]	stdevs[7]	z_score[7]
Z avg	n_v[8]	n_avgs[8]	stdevs[8]	z_score[8]
X freq	n_f[0][0]	n_x_freq	stdevs[9]	z_score[9]
Y freq	n_f[0][1]	n_y_freq	stdevs[10]	z_score[10]
Z freq	n_f[0][2]	n_z_freq	stdevs[11]	z_score[11]
Mag	n_mag_sum	n_mag	stdevs[12]	z_score[12]

7 Functions

The following functions enable real time analysis, memory control to address resource constraints, and radio transmission for human notification.

Table 2. Functions

Function	Description
get_stdev()	Computes the standard deviation in the training phase
z_score()	Computes the z-scores in the inference phase
get_n_values()	Finds the maxs, mins, avgs, and magnitudes for train phase I. Also finds frequencies for train phase II and the inference loop
gc.collect()	Runs garbage collector to clear memory
gc.mem_free()	Returns the amount free memory; helpful for code development
radio.transmit()/radio.recieve()	Enables communication between Micro:bits
display.show()	Displays a configurable image on the Micro:bit LED grid

8 Parameters

One strength of this project is the ability to be placed on a device and adapt to vibrations in virtually any environment, from aircraft to lab server. The following parameters allow the user to fine-tune the algorithm for a given use case.

Table 3. Parameters

Parameter	Description
train_runs	Number of training windows to compute averages for train phase I
train_runs_std	Number of training windows to compute standard deviations for train phase II
wndw_sz	Window size, specifying the number of data points collected per time window.
z_thresh	Threshold for z-score values. Any feature exceeding the threshold is marked suspicious
n_thresh	Threshold for the number of suspicious features. Exceeding this value results in an alert transmitted over radio.

9 Code Minification

The Pyminifier library is recommended to minify the code to reduce file size, thus alleviating memory costs. As such, two versions of the code are available: long (more readable) and minified (more efficient for runtime).

10 Demo Conditions

This project was tested and demonstrated on a BBC Micro:bit with 16kb RAM, 256 flash storage, and 16MHz clock speed.

The device was attached to a fan with two modes: blade on/motor on and blade off/motor on. While running without the blades turning, power can be adjusted form low to medium to high. (Testing without the blades turning made the vibrational changes more subtle, thus challenging the algorithm). The algorithm was trained on medium level, and detected anomalies in low and high power levels. These tests simulated environments such as engines or assembly lines.

Parameters were set as follows:

- train_runs=5
- train_runs_std=5
- wndw_sz=80
- z_thresh=3
- n_thresh=3

11 Conclusion

This project offers real-time, sensitive, and highly efficient machine learning for vibrational anomaly detection using a statistics-based time-series approach. Unlike other machine learning algorithms, this algorithm performs both training and inference on a true edge device. Specifically, this algorithm was developed and evaluated on a BBC Micro:bit, an IoT class I device. With complete on-edge processing, the algorithm minimized bandwidth costs to only transmitted alerts and removed reliance on the cloud or fog.

Possible extensions to this project include 1) translating code to C++ to enable Bluetooth communication, which could alert a humans cell-phone 2) implementing on a non-Micro:bit IoT device 3) experimenting with novel use cases, ranging from detecting medical tremors to detecting dangerously loud sounds including explosions or unauthorized large vehicles entering a secured base.