System of Networked Sensors for Detection and Characterization of Unauthorized Underground Activity Individual Final Product Report

Tyler McKean - May 15, 2021

Abstract: For this project, my team was tasked with designing a ground vibration-based security system that can capture and classify vibration signals and display information to a GUI for user operation. A subsystem for vibration data acquisition was designed using SM-24 geophone sensors, a DC Offset PCB, and Arduino Due microcontroller. The ground vibrations would be captured and converted to digital values via the ADC of the Arduino and then serially imported into the Raspberry Pi subsystem. The Raspberry Pi handled data preprocessing of the digital signal information, feature extraction using statistical methods and functions on the data and compiled the features into a table that was fed into a Kth-Nearest Neighbor machine learning classification algorithm. The entire process was designed to operate with a real-time functionality with the GUI on the Raspberry Pi 4" display updating predicted signals and keeping a history tab of all signals classified along with the date and time of incidence. The goal of the project was to have a future team continue our progress, after my initial team had graduated, and to optimize and/or increase the system's functionality. The Technical Manager for this project was Dr. Tomas Materdey and the Customer Mentor for this project was Captain William Shepherd from the Steven's Institute for Technology.

Introduction & Background:

For my team's problem definition, our Customer Mentor – Cpt. William Shepherd provided us with requirements from a Capstone Marketplace document that included the following details: design a system of networked sensors for the detection and characterization of unauthorized underground activity to provide security for government property such as military bases, government facilities, and National Border security. Some of the features this document asked us to implement were to make the system portable, durable, and unobtrusive when setup. The system needed to consist of unexpensive components, have self-contained power source and be able to process the vibration data such that a user can interpret the system's predictions. We were initially asked to strive for a 100-meter radius, which our team was unable to meet this design requirement.

Our team's final design includes 4 subsystems consisting of the Geophones & PCB, Arduino Due microcontroller, Raspberry Pi with 3.5" display, and the 12V 6000mAh Power Brick Battery. The Geophones would pick up the vibrations in the ground with the PCB providing a DC Offset of 1.65V. The Arduino Due would perform a 12-bit A/D conversion with a sample rate of 500Hz since the bandwidth of the geophones was 250Hz. The Arduino was tied to the Raspberry Pi via a USB-connection, and the Raspberry Pi would serially import values being captured by the Arduino and store this data to a .csv file on the Pi's SD card. With these values saved, Python scripts were written to preprocess the data, extract statistical and probabilistic features, input these features into a KNN machine learning model, and finally

update the predictions of the model to the Graphic-User Interface displayed on the 3.5" touchscreen. The Power Brick Battery would provide about 10 hours of usability for the system and powered all the subsystems via a USB-connection to the Raspberry Pi.

In terms of Work Distribution, I primarily handled all the MATLAB scripts that were written for the project (1-2), worked with the Arduino Due, helped collect and build our signal library, designed the PCB in Altium Designer, and assisted in the Python scripting on the Raspberry Pi. I was also in charge of the time management for the project utilizing the Gantt Charts in the Microsoft Project software. Our team fell behind the timeline at the beginning of the Spring semester but having the chance to work together in the lab on campus at UMass allowed our team to catch up and meet all the set deadlines. Given more time, our team member Yohannes and I were working towards implementing a Filter and Amplifier circuit for the geophones. We unfortunately ran out of time, but I think my team member Yohannes will have the chance to incorporate the design next semester as this project is continued next semester. In terms of budget, our team was provided with both a UMass spending limit and an additional amount provided by the Stevens Institute thanks to our CM. Our team's Technical Manager for the project was Dr. Tomas Materdey, and our team scheduled weekly meetings with our CM/TM every Monday or Friday depending on availability of our advisors.

The important materials I learned for this project involved topics previously covered in my Signals & Systems class and new materials that were not covered in my courses at UMass Boston. The relationship between an input signal, output signal, and the system transfer function in both the time and frequency domain was a critical concept to this Senior Design Project. Finding various research papers [1] helped refresh my knowledge and improve it in the context of underground vibrations. The figure below shows a relationship of an induce vibration, the site transfer function, and the resulted signal.

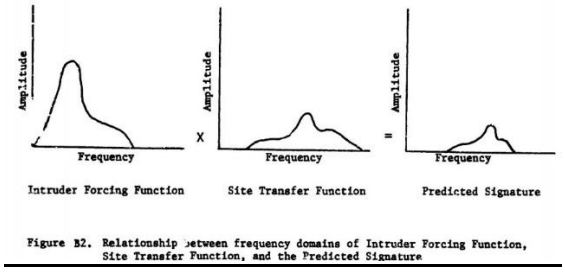

Figure B2. Relationship between frequency domains of Intruder Forcing Function, Site Transfer Function, and the Predicted Signature

Figure 1 – Site Transfer Function Input/Output Relationship

In addition to understanding this relationship, my team had to quickly learn the concept of Machine Learning since this was used to create predictions from ground vibrations. My team used resources provided to us by the MATLAB online learning onramps [10] that were available to us students for topics such as Machine Learning and Digital Signal Processing. These onramps gave me a great understanding and foundation for writing the MATLAB code for the project. This MATLAB code was essentially translated to Python by me, Augustus, and Vulindsky when implementing the code on the Raspberry Pi.

Main Section:

The final design of our project can observe real-time vibrations from the geophone sensors and have the Raspberry Pi serially import the values from the Arduino into a csv file and within 15 seconds, the Raspberry Pi displays a vibration prediction for each geophone sensor on the GUI. Our team prioritized real-time functionality over accuracy, so the system still needs another phase of validation tests to determine its real-time accuracy, though my teammates and I are pleased with where the project ended. Below is a flowchart and 3D graphic of the final design.

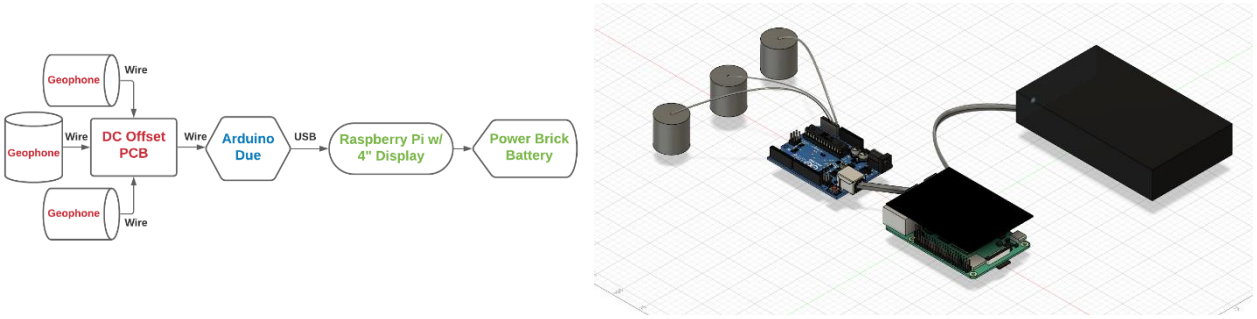

Figure 2 - Flowchart and 3D graphic of my team's design

Since our team was new to the area of Machine Learning, the MATLAB online onramps introduced our team to the available downloadable toolboxes and built-in applications of MATLAB. The most important tool we used for verification tests was the MATLAB Classification Learner. We were able to use it for a verification test to check what type of Machine Learning model would best fit the type or data and number of classes we were trying to predict for our project. Figure 3 below shows the results of our MATLAB validation test.

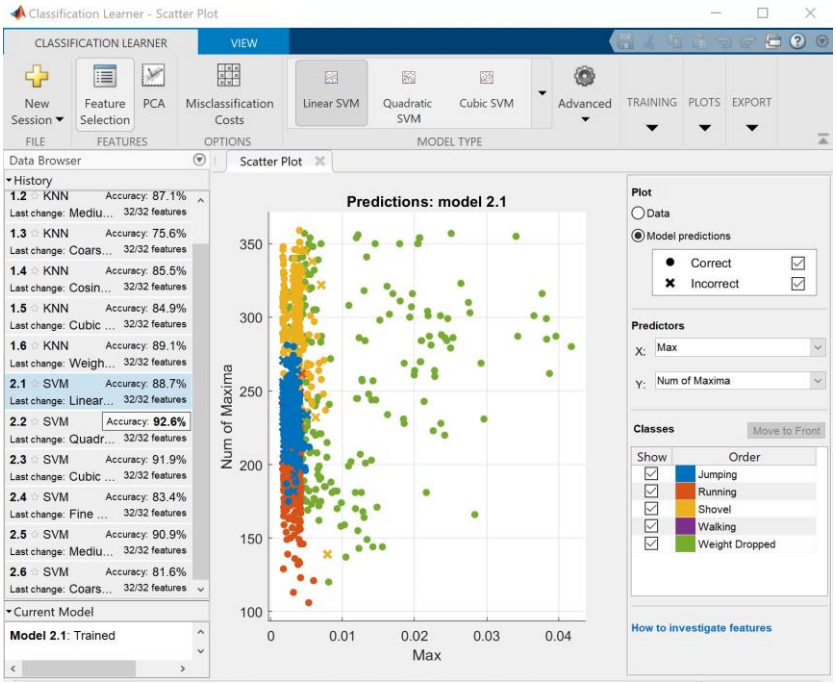

Figure 3 – MATLAB Classification Learner Verification Results

Our team used this Classification Learner after preprocessing and extracting all the features from the MATLAB code in the Appendix (1-2). After feeding the extracted feature table into the Classification Learner, we were expecting our accuracy for the Machine Learning model to be around 70-80%. My teammates Augustus and Vulindsky beforehand had experimented with a Kth Nearest Neighbor model to get familiar with Machine Learning in Python, so our team was hoping for a 70-80% accuracy of a KNN model when performing this verification test. All the models we trained fell between the range of 75-93%, with the KNN model ranging between 85-90%. We performed outdoor tests to collect 1215 samples between 5 different classes as seen in the Fig 3 class window in the bottom right of the image.

Figure 4 – Python KNN Model results

Figure 4 above shows the results of our KNN model when the same feature table is implemented into Python. The implementation in Python resulted in an accuracy of 89.3% which exceeded our expected values. One weakness I would point out is that the model only contains a little over 1200 samples. Typical of Machine Learning applications, the sample sets can well into the thousands or tens of thousands when used for striving for image recognition. In retrospect, if our team had managed to stumble upon the MATLAB online onramps the project would have taken off faster and the team would have been able to collect a lot more samples for the project. On the other hand, I would say that one strength is that our team managed to bring the project into a real-time functionality when initially the team was developing the project in a static manner. From the little amount of real-time verification we performed, we were able to see the system provide a semi-accurate classification with the geophones setup outside and myself walking around. The GUI was able to display either ‘Walking’ or ‘Running’ with multiple geophones predicting the same class. At the start of the semester, I had no idea how much progress my team and I were going to be able to achieve, so to bring the system to real-time functionality for the future teams to start off with was a proud moment of the project. I would add that my team’s productivity improved the more we were working together on the project in the lab.

Some of the greatest lessons I learned from this project involved the design process and project management. Previously, no other coursework classes had given me the opportunity to work on a

single project for a full academic year. I learned a valuable lesson in time management when setting up my team's Gantt chart and having to communicate with my team members about certain deadlines that needed to be addressed. For example, the Filter and Amplifier PCB was initially going to be made to meet the PCB requirement, but our junior member Yohannes was busy with other classes, as was I, and the team needed to shift the PCB design because we did not have enough time in the design process to complete it. Rather than trying to force this circuit into our project, our team used our time to incorporate a DC Offset PCB since it would take less time and we could meet our deadlines as well. Another lesson I learned involved the design process under advisory of our Customer Mentor – William Shepherd. Since the project was made to be designed by his needs, it was great experience to conduct a project that involved a weekly feedback on where the team should focus its energy and research. There were multiple times throughout the last two semesters where Mr. Shepherd urged the team to slow down and focus on the quantitative analysis and reassure ourselves with the math and physics of our project. The last lesson that was most crucial to the project was managing the budget. The team's budget ended up being around $815, although we were provided with extra budget money from the Stevens Institute. Previous budget calculations were estimated to be over $1000 due to switch from the Analog Discovery 2 to the Arduino-Raspberry Pi combination. We also included the price for including the MATLAB onramp and toolboxes, until we realized that our student accounts can give us free access to these resources while still in school. The biggest take-away from managing the budget was to be concise and certain about hardware or software before just ordering it spontaneously. It would not be a difficult predicament to end up wasting all the team's budget money on equipment we don't need or couldn't use. Thankfully, our team was selective and informative when selecting and ordering equipment for the project. We were sure to conduct enough research when selecting the Arduino Due and Raspberry Pi units and SM-24 geophones as the right sensors for this project. Our team worked with Dr Tomas Materdey for the first month of the project to just pick the right vibration sensors. Overall, I think the team performed a great job given the circumstances of the remote learning and balancing the time and budget management for the project.

Conclusion:

My team and I were able to successfully build a real-time vibration detection system meeting all the design requirements and deadlines throughout the last academic year. Using the helpful MATLAB onramps, I was able to obtain a solid foundation of how to classify vibrations using MATLAB and wrote the preprocessing and feature extraction scripts that were later translated into Python to achieve our team's success. I gain some valuable experience in time management, setting up deadlines on our team's Gantt chart and communicating with the team the efficiently meet short-term and long-term goals. Through proper research of viable equipment, our team was able to lower the budget spending by striving for quality equipment that met the needs of the design process. I think the team did a superb job for having to start from an initial R&D phase, to designing the system, and fully developing it through validation tests. We were able to take a problem definition and see it through to a working, real-time system that the future team will have a great head start on. We've made sure to provide all the necessary

guidance for the next team through documentation of our whole process included as a deliverable.

Acknowledgements:

I'd like to personally thank Dr. Tomas Materdey and Cpt William Shepherd for providing my team with an invaluable learning experience that taught me more about what an engineer can expect when entering a design product team. I'd like to thank Andrew Davis for being a reliable source of contact when ordering components or getting access to the lab on campus. Lastly, I'd like to thank the Engineering Department at UMass Boston for providing an opportunity to grow within their curriculum and start me down a path towards a success career in the field of engineering.

Appendix

1.
```
clear all
close all

a = 27; % # of observations
b = 25; % # of observations
Fs = 500; % Sample Rate of 500Hz
[WALK1table,time] = xlsx2tableV4('WALK1.xlsx',a); %Preprocess WALK1 xlsx file, input a
specifies 27 rows
WALK2table = xlsx2tableV4('WALK2.xlsx',a); %Preprocess WALK2 xlsx file, input a specifies
27 rows
WALK3table = xlsx2tableV4('WALK3.xlsx',a); %Preprocess WALK3 xlsx file, input a specifies
27 rows

RUN1table = xlsx2tableV4('RUN1.xlsx',a); %Preprocess RUN1 xlsx file, input a specifies 27
rows
RUN2table = xlsx2tableV4('RUN2.xlsx',a); %Preprocess RUN2 xlsx file, input a specifies 27
rows
RUN3table = xlsx2tableV4('RUN3.xlsx',a); %Preprocess RUN3 xlsx file, input a specifies 27
rows

JUMP1table = xlsx2tableV4('JUMP1.xlsx',a); %Preprocess JUMP1 xlsx file, input a specifies
27 rows
JUMP2table = xlsx2tableV4('JUMP2.xlsx',a); %Preprocess JUMP2 xlsx file, input a specifies
27 rows
JUMP3table = xlsx2tableV4('JUMP3.xlsx',a); %Preprocess JUMP3 xlsx file, input a specifies
27 rows

SHOVEL1table = xlsx2tableV4('SHOVEL1.xlsx',a); %Preprocess SHOVEL1 xlsx file, input a
specifies 27 rows
SHOVEL2table = xlsx2tableV4('SHOVEL2.xlsx',a); %Preprocess SHOVEL2 xlsx file, input a
specifies 27 rows
SHOVEL3table = xlsx2tableV4('SHOVEL3.xlsx',a); %Preprocess SHOVEL3 xlsx file, input a
specifies 27 rows

WEIGHT1table = xlsx2tableV4('WEIGHT1.xlsx',a); %Preprocess WEIGHT1 xlsx file, input a
specifies 27 rows
WEIGHT2table = xlsx2tableV4('WEIGHT2.xlsx',a); %Preprocess WEIGHT2 xlsx file, input a
specifies 27 rows
WEIGHT3table = xlsx2tableV4('WEIGHT3.xlsx',a); %Preprocess WEIGHT3 xlsx file, input a
specifies 27 rows

NOISEtable = xlsx2tableV4('NOISE75sec.xlsx',b); %Preprocess NOISE75sec xlsx file, input b
specifies 25 rows

featureTable(1:81,:) = buildFeatureTable(WALK1table,81); %Extract features from
WALK1table and store in rows 1-81
```

```
featureTable(82:162,:) = buildFeatureTable(WALK2table,81); %Extract features from
WALK2table and store in rows 82-162
featureTable(163:243,:) = buildFeatureTable(WALK3table,81); %Extract features from
WALK3table and store in rows 163:243

featureTable(244:324,:) = buildFeatureTable(RUN1table,81); %Extract features from
RUN1table and store in rows 244-324
featureTable(325:405,:) = buildFeatureTable(RUN2table,81); %Extract features from
RUN2table and store in rows 325-405
featureTable(406:486,:) = buildFeatureTable(RUN3table,81); %Extract features from
RUN3table and store in rows 406-486

featureTable(487:567,:) = buildFeatureTable(JUMP1table,81); %Extract features from
JUMP1table and store in rows 487-567
featureTable(568:648,:) = buildFeatureTable(JUMP2table,81); %Extract features from
JUMP2table and store in rows 568-648
featureTable(649:729,:) = buildFeatureTable(JUMP3table,81); %Extract features from
JUMP3table and store in rows 649-729

featureTable(730:810,:) = buildFeatureTable(SHOVEL1table,81); %Extract features from
SHOVEL1table and store in rows 730-810
featureTable(811:891,:) = buildFeatureTable(SHOVEL2table,81); %Extract features from
SHOVEL2table and store in rows 811-891
featureTable(892:972,:) = buildFeatureTable(SHOVEL3table,81); %Extract features from
SHOVEL3table and store in rows 892-972

featureTable(973:1053,:) = buildFeatureTable(WEIGHT1table,81); %Extract features from
WEIGHT1table and store in rows 973-1053
featureTable(1054:1134,:) = buildFeatureTable(WEIGHT2table,81); %Extract features from
WEIGHT2table and store in rows 1054-1134
featureTable(1135:1215,:) = buildFeatureTable(WEIGHT3table,81); %Extract features from
WEIGHT3table and store in rows 1135-1215

one = ones(243,1); %Create column vector of all 1's
two = ones(243,1); %Create column vector of all 2's
three = ones(243,1); %Create column vector of all 3's
four = ones(243,1); %Create column vector of all 4's
five = ones(243,1); %Create column vector of all 5's

actId = [one;two;three;four;five]; %Create column vector of all previous Classification
Index numbers

featureTable = [featureTable actId]; %Append Classification Index column array to the
feature table
%Creates feature table with known classifications in the last column

featureTable = array2table(featureTable); %Converts feature array to a formated table
%Create Header labels for each feature corresponding to the same column
featureTable.Properties.VariableNames = {'Mean' 'Std' 'Max' 'Variance' 'Skewness'
'Kurtosis' 'RMS' 'Mean Freq' 'Median Freq' 'Peak to Peak' 'Peak to RMS' 'Num of Maxima'
'SpecPeak1' 'SpecPeak2' 'SpecPeak3' 'SpecPeak4' 'SpecPeak5' 'SpecPeak6' 'SpecPeak7'
'SpecPeak8' 'SpecPeak9' 'SpecPeak10' 'SpecPeak11' 'SpecPeak12' 'AutoCorrMainPeak'
'AutoCorr2ndHeight' 'AutoCorr2ndPeak' 'SpecPwr1' 'SpecPwr2' 'SpecPwr3' 'SpecPwr4'
'SpecPwr5' 'VibrationClass'};
featureTable.VibrationClass = cell(1215,1); %Create Cell array to add in Classification
strings to feature table
featureTable.VibrationClass(1:243) = {'Walking'}; %Create cell array for Walking
classification
featureTable.VibrationClass(244:486) = {'Running'}; %Create cell array for Running
classification
featureTable.VibrationClass(487:729) = {'Jumping'}; %Create cell array for Jumping
classification
featureTable.VibrationClass(730:972) = {'Shovel'}; %Create cell array for Shovel
classification
featureTable.VibrationClass(973:1215) = {'Weight Dropped'}; %Create cell array for Weight
Dropped classification
featureTable.VibrationClass = categorical(featureTable.VibrationClass); %Make Vibration
class a categorical type
%When opening the MATLAB Classifcation Learner as part of the Deep Learning
%Toolbox, choose the featureTable as an input and the categorical column of
%classifications will be the predictor types. Classification Learner will
```

```
    %then uses the categorical vibration names as the labels for the difference
    %machine learning models.

    writetable(featureTable, '3.27.21.BackyardTests.csv') % write featureTable to a csv file
```

2.
```
function [feat] = featureExtraction(data)
%FEATUREEXTRACTION performs statistical analysis to extract features from
%Vibration data
fs = 500;
T = 1/fs;
L = length(data);
t = (0:L-1)*T;
Mag = FourierTransform(data,fs,t);
energy = topEnergy(Mag);
% Initialize feature vector
feat = zeros(1,32);
% Average value of top 10% of energy from vibration data
feat(1) = mean(energy);

% Standard deviation of top 10% of energy from vibration data
feat(2) = std(data);

% Max of Frequency Domain of vibration data
feat(3) = max(energy);

% Variance of top 10% of energy from vibration data
feat(4) = var(data);

% Skewness is the third central moment of X, divided by the cube of its standard deviation.
feat(5) = skewness(Mag);

% Kurtosis is the fourth central moment of X, divided by fourth power of its standard
deviation.
feat(6) = kurtosis(Mag);

% RMS value of vibration data
feat(7) = rms(data);

% Mean Freq of Vibration data
feat(8) = meanfreq(data,fs);

% Median Freq of Vibration data
feat(9) = medfreq(data,fs);

% Peak to Peak is the difference between the largest and smallest element in X.
feat(10) = peak2peak(data);

% Peak to RMS is the ratio betMaween the largest absolute value and root mean squared value
in X.
feat(11) = peak2rms(data);

% # of maximums in Frequency data of vibration data
feat(12) = numel(findpeaks(Mag));

% Spectral peak features (12 each): height and position of first 6 peaks
feat(13:24) = spectralPeaksFeatures(data, fs);

% Autocorrelation features for vibrtaion data
% height of main peak; height and position of second peak
feat(25:27) = autocorrFeatures(data, fs);

% Spectral power features (5 each): total power in 5 adjacent
% and pre-defined frequency bands
feat(28:32) = spectralPowerFeatures(data, fs);

% --- Helper functions
function feats = spectralPeaksFeatures(x, fs)

feats = zeros(1,12);
N = 4096;
```

```
mindist_units = 0.3;
minpkdistance = floor(mindist_units/(fs/N));

% Cycle through number of channels
nfinalpeaks = 6;
i=1;
    [psd, f] = pwelch(x,rectwin(length(x)),[],N,fs);
    [pks,locs] = findpeaks(psd,'npeaks',20,'minpeakdistance',minpkdistance);
    opks = zeros(nfinalpeaks,1);
    if(~isempty(pks))
        mx = min(6,length(pks));
        [spks, idx] = sort(pks,'descend');
        slocs = locs(idx);

        pkssel = spks(1:mx);
        locssel = slocs(1:mx);

        [olocs, idx] = sort(locssel,'ascend');
        opks = pkssel(idx);
    end
    ofpk = f(olocs);

    % Features 1-6 positions of highest 6 peaks
    feats(12*(i-1)+(1:length(opks))) = ofpk;
    % Features 7-12 power levels of highest 6 peaks
    feats(12*(i-1)+(7:7+length(opks)-1)) = opks;
end

function feats = autocorrFeatures(x, fs)

feats = zeros(1,1);
i=1;
minprom = 0.0005;
mindist_xunits = 0.3;
minpkdistance = floor(mindist_xunits/(1/fs));

% Separate peak analysis for 3 different channels

    [c, lags] = xcorr(x);

    [pks,locs] = findpeaks(c,...
        'minpeakprominence',minprom,...
        'minpeakdistance',minpkdistance);

    tc = (1/fs)*lags;
    tcl = tc(locs);

    % Feature 1 - peak height at 0
    if(~isempty(tcl))   % else f1 already 0
        feats((i-1)+1) = pks((end+1)/2);
    end
    % Features 2 and 3 - position and height of first peak
    if(length(tcl) >= 3)   % else f2,f3 already 0
        feats((i-1)+2) = tcl((end+1)/2+1);
        feats((i-1)+3) = pks((end+1)/2+1);
    end
end

function feats = spectralPowerFeatures(x, fs)

edges = [0.5, 1.5, 5, 10, 15, 20];

[psd, f] = periodogram(x,[],4096,fs);

featstmp = zeros(5,1);

for kband = 1:length(edges)-1
    featstmp(kband,:) = sum(psd( (f>=edges(kband)) & (f<edges(kband+1)), :),1);
end
feats = featstmp(:);
end
```

```
function [Mag,f] = FourierTransform(X,Fs,t)
%FOURIERTRANSFORM
L = length(t);
f = Fs*(0:(L/2))/L;
Y = fft(X);
P2 = abs(Y/L);
Mag = P2(1:L/2+1);
Mag(2:end-1) = 2*Mag(2:end-1);
end

function [Y] = topEnergy(X)
%TOPENERGY uses logical indexing to find frequencies where the top
%magnitudes are between 60-100% of max peak
a = max(X);
b = 0.60*a;
c = X <= a & X >= b;
Y = X(c);
end
end
```

References

[1] D. H. Cross, "Terrain Considerations And DataBase Development For The Design And Testing Of Devices To Detect Intruder-Induced Ground Motion," May-1978. [Online]. Available: https://apps.dtic.mil/sti/pdfs/ADA055602.pdf.

[2] Rubin, Marc & Camp, Tracy & van Herwijnen, Alec. (2012). Automatically Finding Avalanches in Geophone DataL A Pattern Recognition Workflow. International Snow Science Workshop (ISSW).

[3] "Signal Processing and Machine Learning Techniques for Sensor Data Analytics - Video," *Video - MATLAB*. [Online]. Available: https://www.mathworks.com/videos/signal-processing-and-machine-learning-techniques-for-sensor-data-analytics-107549.html. [Accessed: 19-Apr-2021].

[4] "Designing and Implementing Real-Time Signal Processing Systems Video," *Video - MATLAB*. [Online]. Available: https://www.mathworks.com/videos/designing-and-implementing-real-time-signal-processing-systems-1502972724722.html. [Accessed: 19-Apr-2021].

[5] KAFADAR, O. A geophone-based and low-cost data acquisition and analysis system designed for microtremor measurements In-text: (Kafadar, 2020) Your Bibliography: Kafadar, O., 2020. A geophone-based and low-cost data acquisition and analysis system designed for microtremor measurements. *Geoscientific Instrumentation, Methods and Data Systems*, 9(2), pp.365-373.

[6] N. Evans and D. Chesmore, "Automated Identification of Vehicles using Acoustic Signal Processing," *The Journal of the Acoustical Society of America*, vol. 123, no. 5, pp. 3342–3342, 2008.

[7] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

[8] *MATLAB and Simulink Training*. [Online]. Available: https://matlabacademy.mathworks.com/. [Accessed: 16-May-2021].