Trained Classification Of Surfaces Via Imu-Driven Tactile Sensation

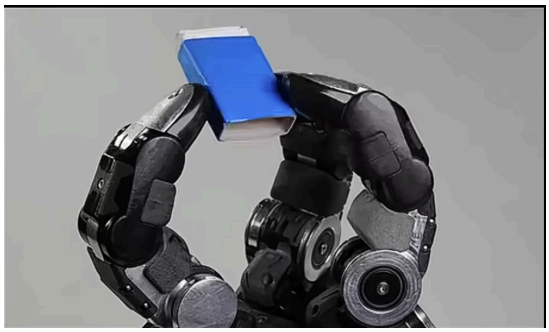
Calvin Kai-wen Smith, Nathan Cinco, Dr. James S. Kang

Department of Electrical and Computer Engineering

California State Polytechnic University, Pomona

Introduction

Dexterity is a major challenge for modern robotics. Current robot hands can pick up a cup, but none of them can button a shirt. Some robotic systems rely on computer vision only, but viable robotic hands will most likely need tactile sensors to help them touch and understand the detailed geometries of their manipulated objects. There is no current consensus on the ideal sensation approach. This project is to explore the usefulness of inertial measurement units (IMUs) (Also known as accelerometers) to probe and correctly identify various surfaces.



A demo of the DEX-EE robotic hand by Shadow Robot. It uses many high-quality optical sensors to understand object geometry.

Methods

Data Collection

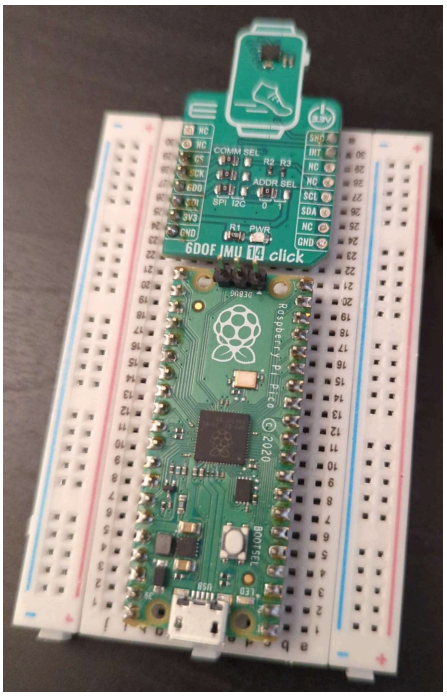
A device was created to observe the IMU signal in response to being dragged across a surface.

Prototype 1 Design

With our first prototype, we aim to record six channels of IMU data (acceleration x, y, z & angular velocity x, y, z) at a rate of 1kHz. Each data point contains 2 bytes of timestamp data and 12 bytes (6 * 2 Bytes) of information. After acquiring data we would transmit it in real time to a PC. Assuming an estimate of 10bits per byte (to account for communication protocol overhead) We calculated the required data rate.

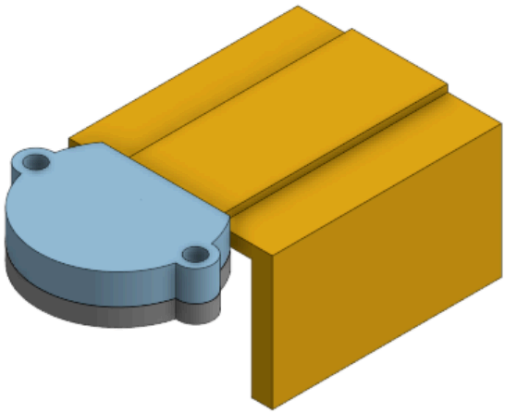
*10bits/Byte * 14Bytes/Sample * 1000Samples/Second = 140,000baud.*

We chose the RP2040 microcontroller (MCU) to acquire and transfer our data. The RP2040 is capable of both 1MHz (1,000,000 baud) SPI and UART communication, which we will use for IMU and PC communications respectively.



A recording setup (w/ wires removed) that consists of a RP2040 microcontroller and a ICM-42688-P IMU.

Additionally, a semi-circular chassis was created and rigidly clamped onto the sensor to facilitate the collection of consistent IMU signal waveforms. Consistency will help the training process as extraneous variation in our training data is a major concern for successful classification.



A 3D model of the two-part sensor chassis (blue & grey) clamped onto the IMU board (yellow).

Prototype 1 Flaws

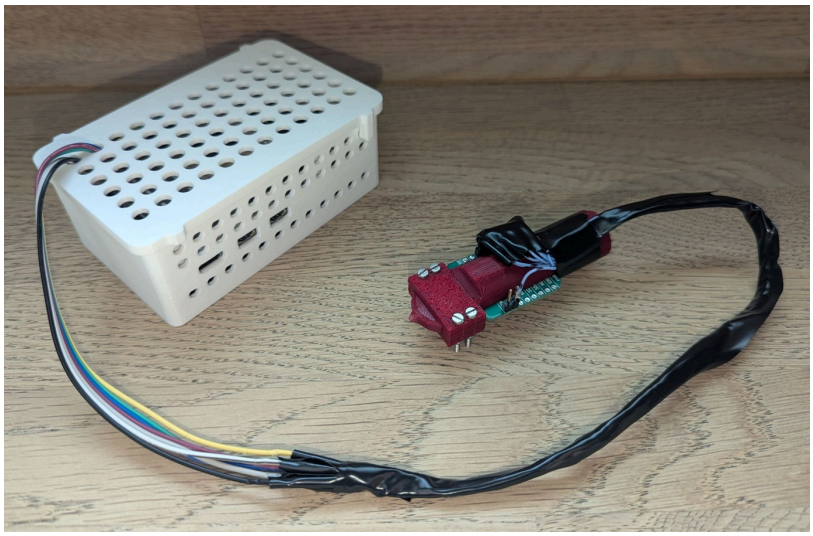
Prototype 1 had issues as data was being corrupted as it was received by the PC. This was most likely because even though 1MHz UART communication should be achievable by the RP2040, it is near the limit of the MCU and has varying success.

We switched to the on-board USB1.0 serial communication, which with its “low speed” of 1.5MHz should’ve been suitable for the throughput. However, the additional communication protocol overhead was too much for our required data rate.

We initially used a semi-circular sensor shell to enclose the IMU sensor. During signal analysis of the waveforms (discussed in the next section), we discovered that this shape was not able to fall into the smaller grooves of rougher surfaces unless dragged on the sharp edge of the chassis geometry. This effect created smoothness where there was none and effectively acted as a mechanical low-pass filter, obscuring higher frequency data.

Prototype 2 Design

We switched to a Raspberry Pi 5 as our MCU to acquire the data.

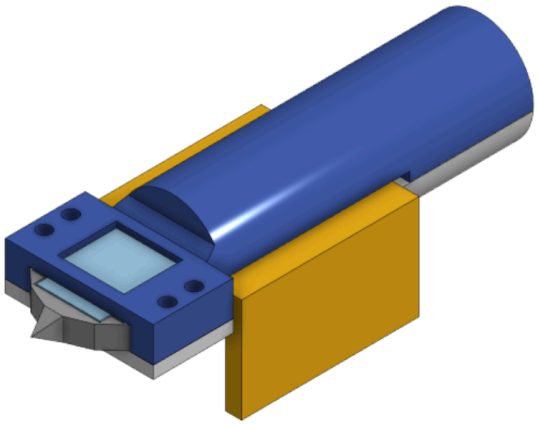


A recording setup that consists of a Raspberry Pi 5 connected to the IMU enclosed in a chassis.

This approach not only solved the MCU-PC communication issues by allowing communication through ssh, it also allowed us to simply store the recording files on the device which would greatly simplify the classification phase.

This switch, however, required us to reprogram the data acquisition software to function inside RasPi’s Debian 13 Linux distribution. Since Linux is not a real-time operating system, we had to make accommodations for the time-sensitive aspects of our setup (notably time synchronization between the IMU and the MCU). Accommodations included setting Linux “niceness numbers” to prioritize the program’s scheduling and using GPIO libraries to store the IMU “interrupt signals” for later accessing.

We also modified the chassis for prototype 2, adding longer wires, a better handle, and replaceable sensor tips. The shape of the sensor tip was also fundamentally changed. We switched to a more pointy tip for prototype 2.



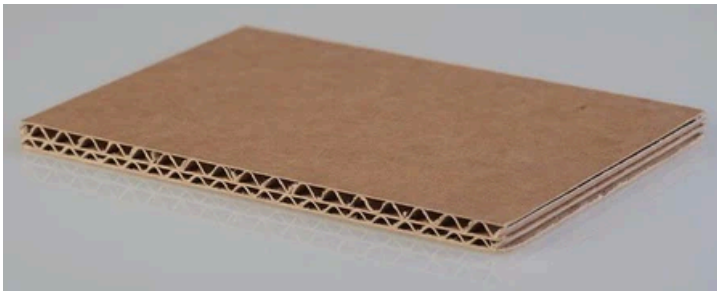
A 3D model of prototype 2's sensor chassis. Pointy & replaceable tips (grey, light blue), clamping handle (white, navy blue), and IMU board (yellow) are shown.

Signal Analysis

After data collection was set up, we decided to collect basic IMU signals and manually analyze them in MATLAB. In order to classify the signals, we must first gain an understanding of what features of the signal are important for differentiating surface types.

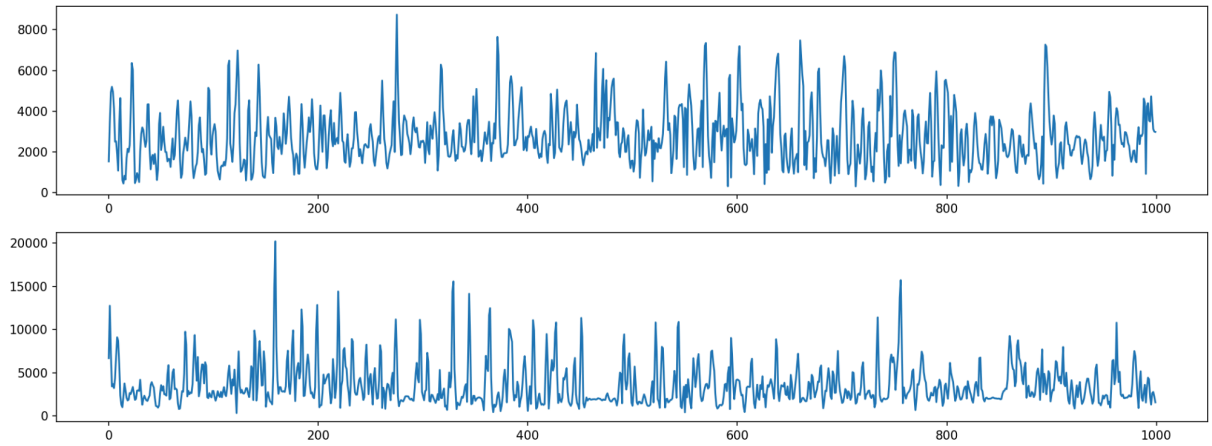
We analyzed the signals of two surfaces: rough wood and smooth cardboard.

The surface of a wooden chair.



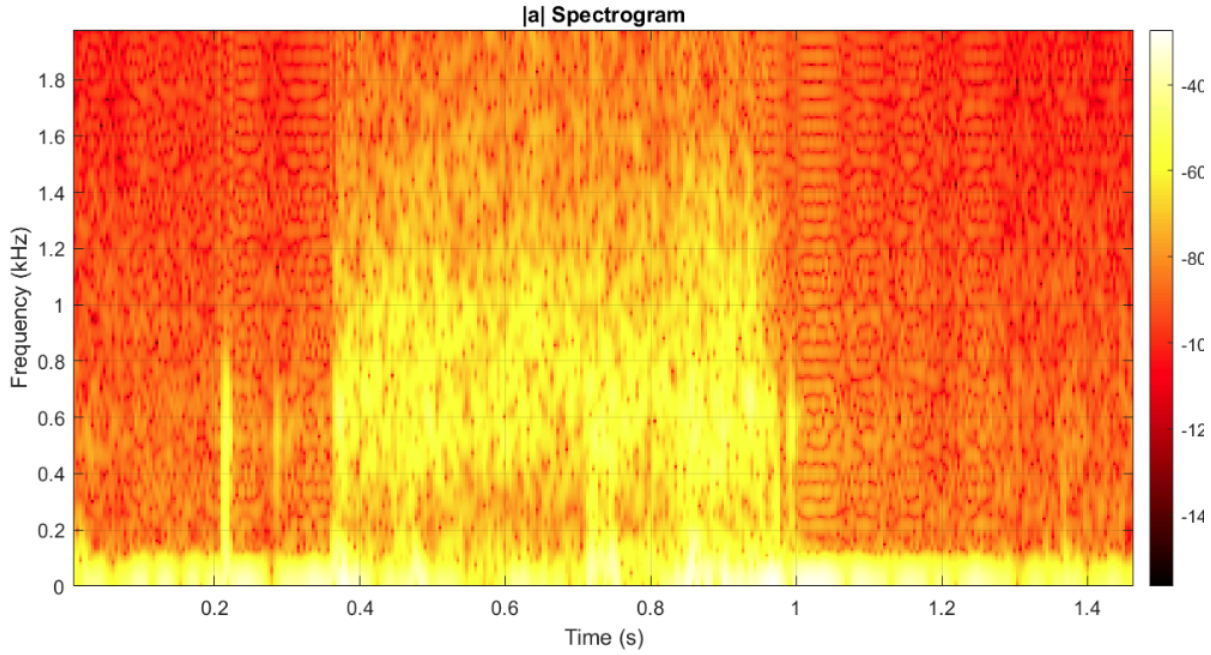
Smooth cardboard surface.

The two signals were plot over time in MATLAB. No initial qualitative difference was observed between the waveforms.



The magnitude of the acceleration vector (ax, ay, az) of the wooden chair (top) and cardboard (bottom).

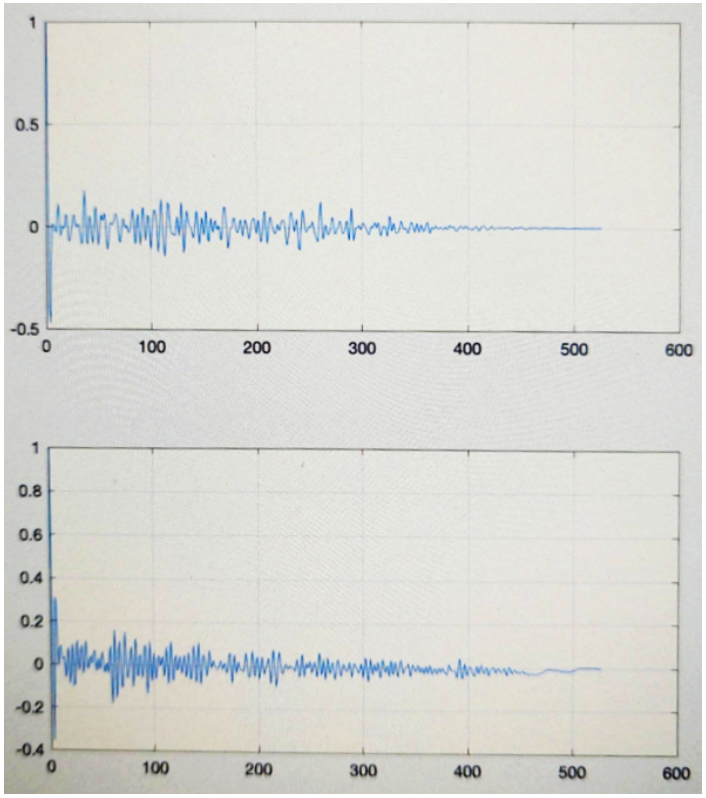
We then played the waveforms through a speaker at different audio sample rates. There was a noticeable difference between the two; the wood sounded more “rumbly” while the cardboard sounded more “hissy”, implying a difference in the frequencies that the IMU captured. These results let us know that there was indeed a qualitative difference between the two, but that a quantifiable difference still needed to be found.



Spectrogram of an IMU signal.

Autocorrelation Plots

Since the waveforms sound like noise (e.g. white noise or brown noise) we were inspired to view them as a stochastic process and take the autocorrelation of the two signals. There was a clear difference in how the autocorrelation changes between surfaces.



The autocorrelation of wood (top) and cardboard (bottom). Y-axis is average statistical correlation between two chosen samples and X-axis is number of samples between those two chosen samples.

Processed Waveforms

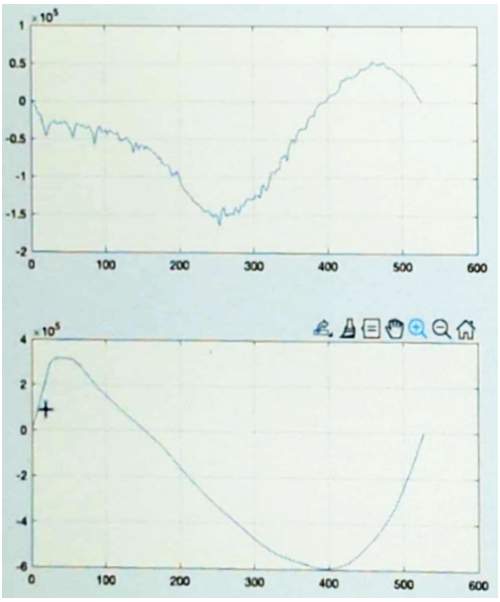
Additionally, we also processed the acceleration signal in order to plot the position of the sensor over time. The Laplace-domain transfer function of this post-processing is derived by removing the DC component (with a low-cutoff highpass filter) to get $s/(s+\omega)$, integrating the signal to then get $1/(s+\omega)$, and then repeating those two steps again.

$$H(s) = (s+\omega)^{-2}$$

Which is equivalent to double integration combined with a second-order highpass filter,

$$H(s) = (s+\omega)^{-2} = s^{-2} * s^2/(s+\omega)^2$$

This allows us to effectively get a magnified view of the sensor position over time.



The processed signal of wood (top) and cardboard (bottom)

There is noticeable difference between the processed signals of rough wood and smooth cardboard.

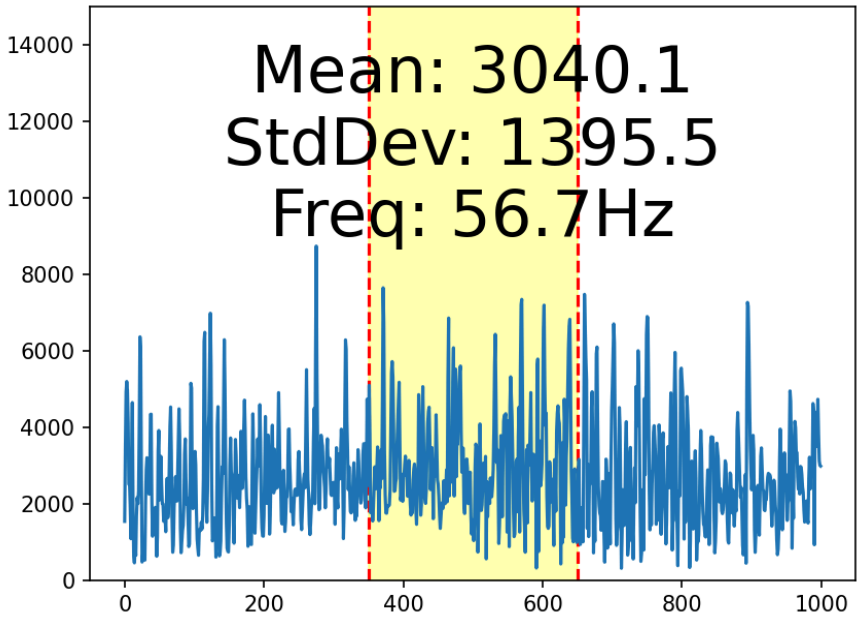
We also plotted the Fourier transform and power spectral density of the waveform but found no significant difference. The autocorrelation plot and processed waveforms proved to be the most effective at qualitatively distinguishing the surfaces and helps us derive quantifiable features to extract for model training.

Prototype 1 Chassis Flaw

As previously mentioned, we initially used a semi-circular sensor shell to enclose the IMU sensor. However, during processing of the waveforms, we discovered that this shape was not able to fall into the smaller grooves of rougher surfaces unless dragged on the sharp edge of the semi-circular geometry. This effect created smoothness where there was none and effectively acted as a mechanical low-pass filter, obscuring higher frequency data. We switched to a more pointy sensor chassis for prototype 2.

Feature Extraction

Data fed into a classifier comes in the form of statistical features. Raw data over time is divided into small windows and then statistical features are calculated for each one. Examples of features include the mean, variance, peak frequency, etc.



Example of a window and its features.

Time Domain Features

Features were extracted from the time domain representation of the data.

Range: the highest value subtracted by the lowest value in the window.

Mean: the arithmetic average of the data within the window.

Variance: the spread of the signal amplitudes around the mean.

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Mean.

$$\sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2$$

Variance.

Frequency Domain Features

The time domain representation of the data was converted to the frequency domain via FFT. Features were extracted from the frequency domain to train the classifier.

Peak frequency: the frequency with the highest spectral energy

Peak spectral power: the absolute magnitude of the energy at the peak frequency

Mean spectral power: the average power across the frequency spectrum

Spectral entropy: a measure of the data spectral complexity. Calculated by normalizing the power spectrum into a probability distribution and then applying the Shannon Entropy formula.

$$P_{mean} = \frac{1}{K} \sum_{k=1}^{K} P_k$$

Mean spectral power.

$$p_k = P_k / \sum P$$

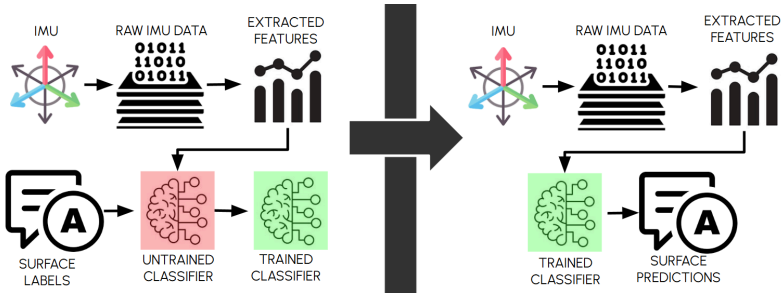
Normalizing the power spectrum (used for Spectral Entropy).

$$H = -\sum_{k=1}^{K} p_k \log_2(p_k)$$

Shannon Entropy Formula (used for Spectral Entropy).

Classification Model Training

Each window is assigned a label of which surface it came from so the model can learn the classification. The label can be a material type or a physical property.



Data paired with labels can train a classifier (Left). A trained classifier paired with unlabeled data can produce a prediction (Right).

Classification models are prone to overfitting, a problem where trained models have learnt by rote memorization rather than pattern comprehension and cannot accurately make predictions on new data. To protect against this, cross-validation and testing set separation are used.

Cross-validation splits the training process into multiple sessions, evaluating the progress of each against its peers rather than against itself.

Testing set separation moves a percentage of the dataset into a testing set (we use 20%). The testing set is used to evaluate the accuracy of the final classification model. Since the final classification model has never seen the testing data before, the testing set provides a more unbiased insight into a model's performance.

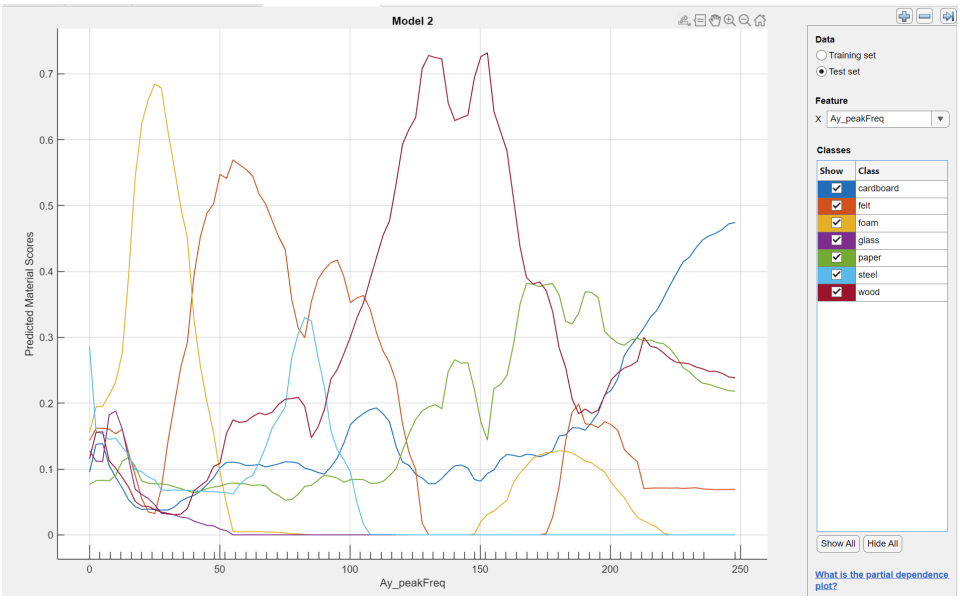
Model Types

We used MATLAB's Classification Learner App to train multiple different classifiers on basic statistical features.

Using the Bagged Trees model trained on MATLAB, we generated C code using MATLAB's C coder. The compiled executable was able to load test data and make predictions.

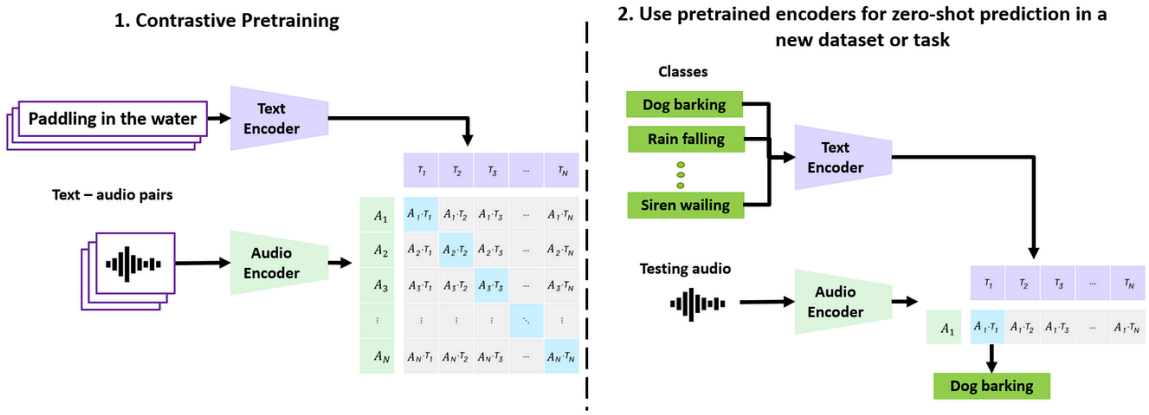
```
PS C:\Users\natha\SDP\Sandpaper\C_Code\BaggedTrees> ./runModel.exe
Attempting to open test_data.csv...
Successfully opened test_data.csv.

--- Processing Line 1 ---
Feeding IMU features into the Bagged Trees model...
Model successfully ran the prediction!
Line 1 - Predicted Class ID: 2

--- Processing Line 2 ---
Feeding IMU features into the Bagged Trees model...
Model successfully ran the prediction!
Line 2 - Predicted Class ID: 1

--- Processing Line 3 ---
Feeding IMU features into the Bagged Trees model...
Model successfully ran the prediction!
Line 3 - Predicted Class ID: 1

--- Processing Line 4 ---
Feeding IMU features into the Bagged Trees model...
Model successfully ran the prediction!
Line 4 - Predicted Class ID: 1

--- Processing Line 5 ---
Feeding IMU features into the Bagged Trees model...
Model successfully ran the prediction!
Line 5 - Predicted Class ID: 1
```

Terminal output of exported C code.



A plot in MATLAB's "Classification Learner App" that shows the influence of a certain data feature in making predictions for each class.

We also used PyTorch (an AI training library) to embed the waveforms into a high-dimensional space designed for audio files. The embedded data is then run through a simple neural network (NN) to produce classifications. This approach uses Contrastive Language-Audio Pretraining (CLAP) models, and effectively allows us to classify the waveform as it relates to a human understanding of everyday sounds. We used Hugging Face to download a CLAP model.



A diagram of how a CLAP model is trained and used.

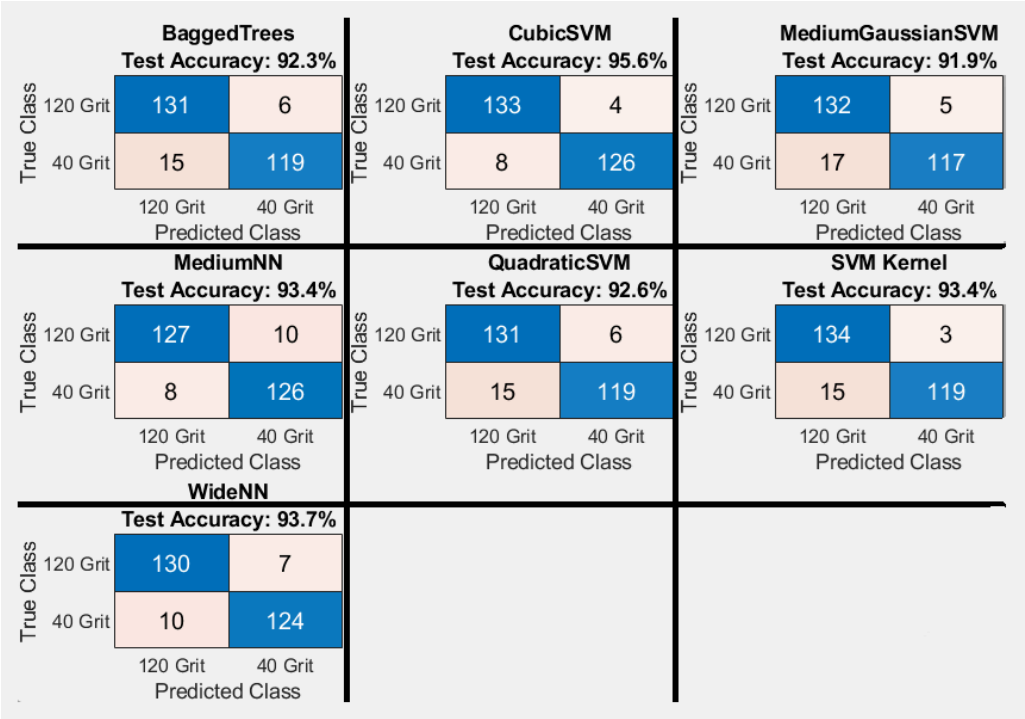
Results

MATLAB Model

Data was collected on 120-grit and 40-grit sandpaper. 7 models were trained. Cross-validation and testing set accuracies were recorded.

Model Name	Cross-validation Accuracy	Testing Set Accuracy
CubicSVM	92.25%	95.572%
WideNN	90.50%	93.727%
MediumNN	89.67%	93.358%
SVM Kernel	89.76%	93.358%
BaggedTrees	88.93%	92.251%
QuadraticSVM	91.70%	92.251%
MediumGaussianSVM	88.75%	91.882%

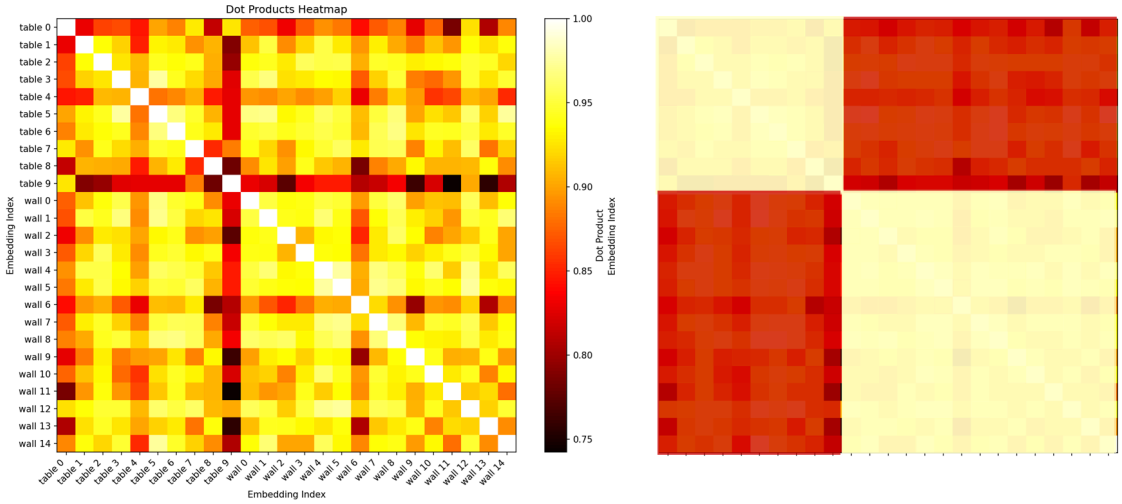
Table of MATLAB models' cross-validation and test-set accuracies



A grid of MATLAB Classification Learner App testing confusion matrices and testing accuracies. Confusion matrices show how many tests fall into each quadrant (prediction, truth). Blue quadrants are correct classifications while the other quadrants are misclassifications.

PyTorch CLAP Embedding Classification

The PyTorch CLAP embedder + neural network was unfortunately never trained or tested due to preliminary findings. Analysis of the CLAP embedder's output showed no discernable deviation between the dot products of the 40 grit and 120 grit vectors. This implies that no classification would be able to be made.



A heatmap (left) of two different surfaces (smooth table, rough wall) and the dot product of their high-dimensional vectors. A visualization (right) shown above is what a very ideal heatmap should look like for classification purposes. Notice how there is no pattern to the dot products other than outliers. Classification is unlikely to succeed.

Discussion

Cross-validation accuracy correlates with how distinct the data of the different surfaces are from each other. A low cross-validation accuracy implies a dataset with unclassifiable data. However, this number can be artificially inflated by choosing more complex models (i.e. models that are better at rote memorization).

Testing set accuracy correlates with how well the trained model performs on new data. Our results show that an IMU is capable of distinguishing between basic roughness properties of surfaces. Interestingly, our testing set accuracies were higher than our cross-validation accuracies, but this can be explained by the cross-validation procedure reducing training set accuracy to promote better learning. MATLAB currently does not have a way to benchmark the model on non cross-validated training set data.

The difference in data quality between pointy and semi-circular sensor tips is possibly analogous to the difference between animals without and with fingerprints. There is speculation that the fingerprints of primate fingers were used for the tactile sensation of different surfaces. The impact of sensor tip geometry supports this theory.

It is still unclear whether treating the IMU waveform as statistical noise or as audio is the better approach. On one hand, IMUs record data in a very similar way to standard audio microphones (by measuring the displacement of a physical object). But on the other hand, IMUs typically cannot reach the higher sample

rates (>40kHz) that human hearing is meant for. In this way, the IMU waveform might be better modeled with stochastic methods more in line with the analysis of earthquakes and ocean tides.

Future Work

Future work could entail switching from IMUs to microscopic “MEMS” microphones. This will increase the sample rate of the waveform and push the research more into the CLAP, audio-embedding approach, although with a more appropriate embedder.

Future work could also move more towards statistical modeling. By adding more IMUs to the device, each located and actuated in a different way, A more comprehensive IMU profile of the surface can be generated, possibly allowing for better statistical classification.

References

[1] C. K. Smith and N. Cinco, “Hylthek/IMU-robot-finger,” Imu-Robot-Finger, https://github.com/Hylthek/Imu-Robot-Finger (accessed May 15, 2026).

[2] Y. Wu et al., “Large-scale contrastive language-audio pretraining with feature fusion and keyword-to-caption augmentation,” arXiv.org, https://arxiv.org/abs/2211.06687 (accessed May 15, 2026).