# Comparing neuromorphic encoding and analysis methods for texture classification using touch sensors

Robin Kock
*University of Groningen*

Examiners: Elisabetta Chicca & Bart Besselink
Daily supervisor: Alejandro Pequeno-Zurro

February 2025

**Abstract**

Touch enables humans to effortlessly assess object properties, but robotic systems lack this ability. While advanced tactile sensors provide high-resolution pressure signals, traditional machine learning methods are computationally expensive. Neuromorphic computing, integrating memory and computation akin to the brain, presents a promising alternative. This study explores neuromorphic methods for classifying surface textures from pressure data. Sampled pressure data is converted into spike trains using biologically inspired encoders. These spike trains are analyzed using an array of spiking phase-locked loops (sPLLs) to extract frequency components, which are then classified using a linear regression model. The objective of our research is to investigate which neuromorphic encoding method yields the highest classification accuracy and whether a network of sPLLs can extract frequency components sufficiently to enable accurate classification. To this end, we optimize the sPLL network's parameters for different encoders. The best network of sPLLs attained the highest accuracy of 95.4% using an encoder based on a leaky integrate-and-fire neuron. Generally, the sPLL network exhibited a performance comparable to that of a spiking fast Fourier transform, suggesting the potential of neuromorphic approaches in this domain. Furthermore, the findings of this study underscore the promise of neuromorphic methodologies in the analysis of touch sensor signals. Consequently, this contributes to the advancement of neuromorphic tactile sensing systems and to the enhanced adaptability of robots in real world environments.

## Contents

## 1 Introduction

The sense of touch is not one that is typically considered consciously; however, it becomes a focal point when an individual encounters stimuli such as spiky, hot, or cold surfaces. In the development of autonomous systems, the significance of tactile perception becomes evident, as it plays a critical role in enabling effective interaction with the physical environment. Human fingers possess the ability to rapidly assess the weight, fragility, and surface material of an object through brief contact. In contrast, a basic robotic actuator lacks these capabilities, which often results in difficulties when performing tasks involving objects for which it was not explicitly designed. For example, a robotic system trained to grasp paper cups may face challenges when attempting to manipulate a heavy-steel bottle or a soft towel [1].

One approach to improve the ability of robots to interact with a wide variety of objects, is through improving their touch sensing abilities. Akin to how humans have touch receptors a robotic end-effector may have sensors embedded in their artificial fingers or skin. These generate signals representing the interaction between the finger and the material of the object, e.g. the preassure felt by the finger, or vibrations produced by the touch interaction. However, these signals do not improve the grasping ability of the robot by themselves. First, they must be analyzed to determine which material the actuator is in contact with. Then, different controll algorithms are needed to control the actuator based on the determined material. In this study we will be focusing on the first task, determining the material based on the interactions between a sensor and the material.

More specifically, we use data gathered by sensors moving along a material sample. This movement induces vibrations in a probe and is recorded using pressure transducers. These recorded signals are then analyzed using our methods. We use two different datasets, that each record the aforementioned interactions in a different manner. Then, three different encoders are used to turn the pressure signals into spike trains. Finally, we use a network of spiking phase-locked loops (sPLLs) to turn the spikes into feature vectors, which are analyzed using traditional regression analysis.

The methods we explore in this study are of a neuromorphic nature, i.e. they are inspired by brain like structures and convey information using spikes [2]. This will be further motivated in Section 1.1, along with more details about the encoders and neuromorphic circuits used in this research.

The overall goal of our reasearch is to work on the problem of texture classification, for the reasons given above. Additionally, the efficacy of using neuromorphic means to perform this texture classification is investigated. Through this research, we aim to advance the fields of neuromorphic computing and robotics by demonstrating a practical implementation of a detection system and contributing valuable data for future robotic system development.

### 1.1 Theoretical Background

The advent of commercially available touch sensors, capable of generating high-resolution pressure signals, has led to a proliferation of research opportunities in the field of machine learning. These sensors, when coupled with conventional machine learning-based time series classification algorithms, such as time series random forests [3], hierarchical vote collective of transformation-based ensembles (HIVE-COTE) [4, 5], or other feature extractors paired with standard classification algorithms [6, 7], have opened new avenues for analysis. However, this technological advancement is accompanied by a significant limitation inherent to modern autonomous systems: power consumption. Humans possess approximately 230,000 touch-sensitive afferents within their skin [8], enabling continuous tactile input processing, albeit largely subconsciously. The nervous system adeptly prioritizes salient stimuli, facilitating instantaneous detection and response to unexpected contact. The replication of this phenomenon in an autonomous system using conventional machine learning, is impractical due to the substantial processing demands. The analysis of even a limited number of signals necessitates a significant computational capacity [9], a requirement that becomes impractical when the number of signals is increased to the thousands observed in the human body.

The issue of power consumption and parallel processing is an enduring problem, with origins that extend to the fundamental design principles of modern computing. The vast majority of computers are constructed based on the Von Neumann architecture, which necessitates the retrieval of data from memory, the execution of computations, and the subsequent storage of results back into memory. This process is inherently constrained by the maximum attainable data transfer rate between the processor and memory, thereby establishing an upper limit on scalability [10]. One potential solution to this problem is to combine the functions of computing and memory into a unified entity, akin to the integrated nature of the brain. This approach, known as neuromorphic computing, aims to emulate the way in which the human brain processes information [2].

As mentioned above, this paper explores various neuromorphic methods for classifying surface textures from pressure data. The process is generally comprised of two steps. First, the sampled pressure data is converted into spikes using an implementation of a touch receptor. Then, these spikes are analyzed using a neuromorphic classifier. The subsequent sections will provide a concise introduction to both steps.

First, the rationale behind the utilization of pressure encoders is elucidated from a biological standpoint. Human skin contains three primary categories of touch receptors: slowly adapting afferents, rapidly adapting afferents, and Pacinian afferents. Where an afferent is the nerve connecting the receptor cell to the rest of the human nervous system, while processing the signal. These receptors are sensitive to different stimuli, with the slowly and rapidly adapting afferents encoding the onset and cessation of pressure applied to the skin, as well as the intensity of the pressure. In contrast, the Pacinian afferent responds to diverse vibration frequencies [11]. Conventionally, the slowly adapting afferent is regarded as the primary classifier of textures [12]. However, our approach does not entail the classification of textures through direct contact with a surface; rather, it involves the gliding motion of the sensor across the surface. In this context, the vibrations in the finger are more pertinent, thus emphasizing the role of the Pacinian afferent. It is noteworthy that contemporary research suggests a lack of definitive responsibility for different afferents, suggesting instead a collaborative nature in performing sensory tasks [12]. To investigate the phenomenon of texture sensing, a bottom-up approach is employed. In this study, we opted to utilize single-channel data, concentrating our attention on a solitary afferent. This methodological decision serves as a solid foundation for subsequent investigations involving multichannel data.

In this research, an implementation of the dynamics of the Pacinian afferent [13] will be utilized, as well as, a simplified version of the encoder that bypasses some of the processing and is based on a simple leaky integrate-and-fire neuron. Additionally, a baseline encoder that detects the peaks in the frequency spectrum, derived using the fast Fourier transform [14], and produces constant spikes with the frequencies of the detected peaks will be used. Subsequently, the spike trains generated by these three encoders must undergo processing to extract the frequency-related information.

Our approach to analyzing the frequency components of the spike trains involves the utilization of an array of sPLLs. A phase-locked loop is a traditional electronic circuit comprising a phase detector, a low-pass filter and a voltage-controlled oscillator. During typical operation, the phase detector will detect differences between the phase of the input signal and the signal generated by the oscillator. The difference signal is then filtered by the low pass filter, to ensure signal smoothness, before being fed back to the voltage-controlled oscillator. This iterative process leads to a situation where the oscillator frequency is perfectly aligned with the input signal's frequency.

The neuromorphic version of a phase-locked loop, sPLL, operates according to the same principles as it's traditional counterpart. However, a notable distinction emerges in the composition of its constituent components. Specifically, the oscillator and the low-pass filter are neurons, while the phase detector is a special synapse called the temporal difference encoder (TDE). The fundamental disparity between the sPLL and its traditional counterpart lies in the former's utilization of a spike train as an input, along with an oscillator that generates spikes. The inherent instantaneous nature of spikes poses a challenge in measuring phase differences reliably [15]. However, empirical evidence has demonstrated the sPLL's capacity to lock onto the frequency of spike trains [16]. It has even been demonstrated that certain neurons in the brain appear to be functioning like phase-locked loops [17].

We employ an array of 50 sPLLs, each tuned to a distinct frequency, to receive a common spike train encoded by one of the three encoders. Subsequently, we measure the average spike rate of each sPLL and use the 50 different values to construct a feature vector. These feature vectors are collected for all the samples in a dataset and are then used to train a linear regression classifier. The performance of this classifier is then utilized to assess the overall effectiveness of the system. Through this process, we optimize the parameters of the sPLL network and identify the encoder that demonstrates the greatest efficacy.

## 1.2 Research Questions

The simulation described above was developed to address the following research questions. Firstly, we ask *which neuromorphic method of converting pressure vibrations into spikes results in the highest material classification accuracy by a network of sPLLs?* Additionally, we are also interested in validating the architecture for classifying surface textures and related time series information. Thus, we pose the following additional question: *Is single-channel non-spatial pressure data enough to reliably classify textures using neuromorphic methods?*

The remainder of this thesis is structured as follows. In the methods section, we provide a comprehensive description of the aforementioned architecture, accompanied by a thorough exposition of the theoretical underpinnings. We also describe the preliminary analysis done on the data and encoders. Subsequently, in the Results section, we present the findings from these preliminary analyses and assess the performance of our models. Then, we conclude with a summary of our key findings and an articulation of the research question that has been addressed. Finally, in the Discussion section, we will attempt to explain the results obtained, as well as reflect on the research in a broader context.

## 2 Methods

In this section, we outline the methodologies employed in our study, starting with an initial analysis of the training data. We subsequently detail the various encoding methods utilized to process the training data with the objective of isolating frequency components and converting them to spiking signals. Then these methods are compared to biological afferent nerve fibers. Thereafter, we introduce the spiking phase-locked loop technique employed in our analysis. Next, we outline the method for extracting information content through the application of linear regression techniques. Each of these steps is crucial for understanding the computational approaches and the efficacy of the data processing methods used in our research. Finally, the algorithms used for the simulations and calculations are presented. For a graphical representation of the final model and the steps we take to get there, see Figure 1.

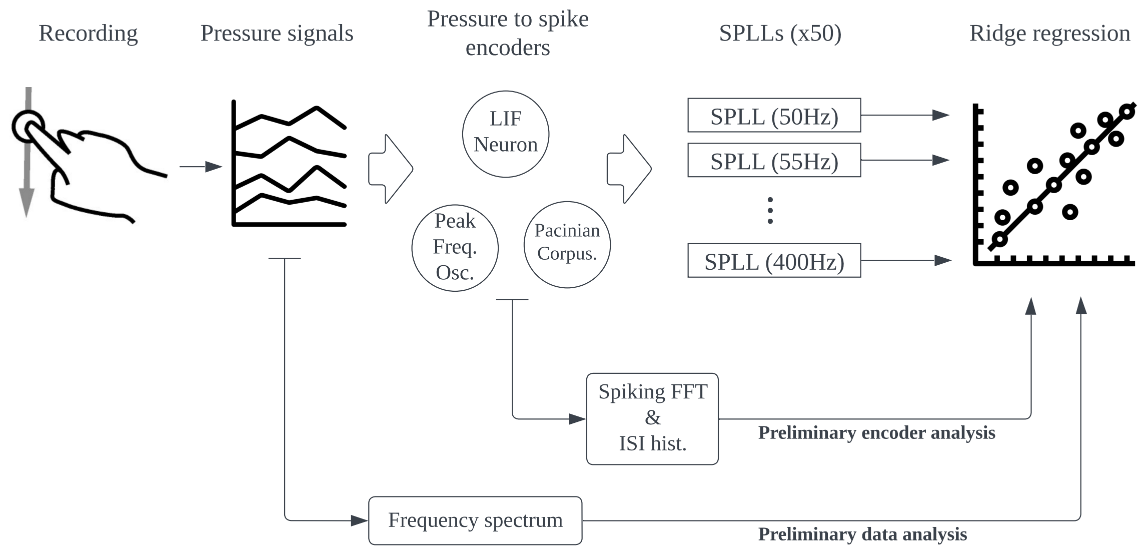


Figure 1: Overview of the architecture used to analyze the performance of different pressure encoders. **Top**: The pressure encoders integrated with the spiking phase locked loop. *Recording*: Two datasets are looked at, one recorded vibrations using a microphone attached to a stylus, while the other used a pressure transducer connected to a liquid filled synthetic finger. *Pressure signals*: the recorded signals are split into two second long single channel samples. *Pressure to spike encoders*: we test three different encoders, which turn the pressure signals into spike trains. *sPLLs*: each spike train is fed into a network of 50 sPLLs, where the spike rate of the time difference encoder of each sPLL is measured. *Ridge regression*: The 50 spike rates are collected into a feature vector for each sample and are used as the input data to train a regression classifier, the accuracy of this classifier is used to determine the performance of the whole system. **Bottom**: preliminary analyses, without the sPLL network. *Data analysis*: Using the frequency spectrum of the data to extract features, the same classifier is trained and the accuracy measured. *Encoder analysis*: Using a spiking fast Fourier transform (sFFT) or an inter-spike interval histogram for feature extraction, the same classifier is trained once again and the performance analyzed.

### 2.1 Data Collection

To assess the efficacy of diverse pressure encoding methodologies, a selection of two datasets was utilized, each comprising vibration data that was obtained through distinct methodologies. For the first dataset, Kursun et al. [18] collected data using a rotating drum with a fixed stylus. The material was glued on to the drum and was rotated beneath the sensing aperatus, then, the vibrations were recorded using a condenser microphone attached to the stylus. The materials utilized in this study encompass a diverse range of properties, including four distinct fabrics, aluminum film, two maquette fabrics, two sticky fabrics, two sparkle papers, and toy tire rubber. The data collection process involved a sampling duration of 20 seconds at a frequency of 8 kHz for each material. A comprehensive overview of the collected data is provided in Figure 3, located in the Results Section. Throughout the course of this research, the dataset will be referred to as VibTac.

The second dataset is more akin to a real-world scenario, involving a robot arm maneuvering a synthetic finger across stationary material specimens. Gao et al. [19] emplyed a SynTouch BioTac sensor affixed to a ”KUKA LBR iiwa 14” robotic arm. The BioTac sensor collects low-resolution localized pressure data through electrodes, along with a high-resolution global pressure value. The artificial finger is filled with a fluid, that transmits the vibrations from the skin to the global pressure transducer. We are only using the high resolution pressure data collected by the transducer in the finger and ignore the localized information. Overall, they collected 50 samples of 20 different materials. Where each sample is 6 to 8 seconds long, sampled at 2.2khz. This data was collected for a carpet net, cotton, a bath towel, fake leather, three different polypropylene materials, a felt cloth, two different papers, two different soft materials, cork, ethylene-vinyl acetate, a fiber board, wood, styrofoam, a sponge, foam and a metal plate. This dataset will be referred to as BioTac.

The two datasets are subdivided into samples with a duration of two seconds each, resulting in 10 samples per class for the VibTac dataset and 195 to 200 samples per class for BioTac.

During the preliminary analysis of the BioTac dataset, it was observed that certain classes exhibited highly similar frequency distributions. Consequently, the decision was made to eliminate several classes from the experimental scope. The following classes were removed: cork, felt, fiber board, fake leather, both different papers, the smooth polypropylene material, the sponge, and finally the wood. Leaving a total of 11 out of the 20 original classes. The selection of these particular classes was guided by a meticulous analysis of the data, as outlined in Section 3.1, with the objective of identifying and eliminating classes that exhibited significant confusion with the regression model. For a comprehensive regression analysis of the entire dataset, refer to Figure 13 in Appendix A.

### 2.2 Pressure encoders

To classify textures using a spiking neuromorphic architecture, continuous or sampled pressure signals must first be converted into spikes. In biology, this is done by afferent neurons that respond to different pressure levels, vibrations, and other mechanical stimuli. Previous research has shown that the pacinian afferent plays the most important role in, sliding texture classification [11]. This afferent is most sensitive to vibrations between 250 and 550 Hz [20]. For this reason, we focus on artificial afferents, or rather pressure encoders, which are most sensitive in this frequency range. In this paper, three different encoders are tested: An implementation of the pacinian corpuscle afferent, a frequency peak oscillator, and a simple leaky integrate and fire neuron. These three encoders are benchmarked individually and tested in combination with the spiking phase-locked loop. In the following paragraphs, each encoder is explained in more detail.

**Pacinian corpuscle (TouchSim)** In humans, the Pacinian corpuscle afferent responds to vibrations and other rapid changes in pressure. This behavior can be approximated by a fairly simple algorithm [13]. The pressure signal is first passed through a low-pass filter to remove frequencies above 550 Hz. The resulting signal is then differentiated. Both the derivative and the regular signal are split into positive contributions (signal is above zero) and negative contributions (signal is below zero). The negative contributions are inverted, leaving four different positive signals. Next, a new signal is formed by a weighted sum of the four signals and passed through a saturation filter. The saturation filter ensures that the corpuscle does not overreact to strong signals. Finally, some noise is added to the signal and used as the input current to a leaky integrate and fire neuron, which converts the signal into spikes.

The weights of the weighted sum as well as the other parameters of the process described above were determined experimentally by Saal et al. [13] to correspond to measured spikes in humans. They implemented this afferent in a Python library[1] that is used to convert pressure signals into spikes for this research.

**Frequency peak oscillator** This pressure encoder does not exist in biology because it is an idealized transducer to compare with others. It works by detecting the peak frequencies in a given pressure signal and then combining the

[1]TouchSim: `https://github.com/hsaal/touchsim`

spikes from different oscillators that peak at that frequency. Similar to the artificial pacinian, the pressure signal is first low-pass filtered at a cutoff frequency. A Fourier transform is then used to compute the frequency components of the signal. The phase information is discarded by taking the absolute value of the complex output, and a peak detector is used to find three frequency peaks. The peak detector maintains a minimum spacing of 16 Hz between peaks to spread them more evenly across the spectrum.

After the peaks are found, they are converted to spikes in the following manner. For each peak, spikes are periodically generated at the frequency of the peak. Then all these spikes are taken together as the output of the encoder. A great advantage of this method is that the spike train is stationary and can be made as long as necessary, so even if the pressure sample is only one second long, the frequency peak oscillator can produce spikes for as long as necessary.

**Leaky integrate and fire** Finally, the last pressure encoder uses a standard leaky integrate and fire neuron, where the synapse is replaced by a current source that mirrors the pressure signal. This encoder is a simpler version of the artificial pacinian presented above; we skip the preprocessing steps and pass the signal directly to a leaky integrate and fire neuron. Although we do not expect to improve performance in this way, this encoder is very easy to implement in neuromorphic hardware, as it requires only a single neuron and basically no additional hardware.

To measure the performance of the encoders, they are evaluated on the datasets, producing spikes for each trial of each texture. The spikes are then converted into a feature vector using one of the frequency analysis methods described below. Finally, the classification accuracy of a simple regression model based on this feature vector is used as a proxy for the performance of the encoders. This method will be used throughout this article and is explained in more detail in Section 2.6.

## 2.3 Frequency analysis

Achieving the goal of classifying surface textures using a network of sPLLs is not trivial, especially since the performance of individual encoders are not always consistent. In other words, it is not clear whether the spike trains produced by different encoders are distinct enough to distinguish the different input classes. Thus, to understand the performance of the sPLL network, we must first understand the pressure signals themselves and then understand the spike trains produced by the encoders. The comparison of spike trains directly is not feasible, as they might differ significantly within classes. Hence, we convert the spike trains into fixed size feature vectors and compare these.

Several assumptions are made to facilitate this conversion. First, we assume that the surface textures are uniform over the duration of the sample, then we assume that the spike train is stationary, i.e., the distribution of spikes should be the same over any sufficiently large subsection of the spike train. Of course, this assumption only holds if the pressure signals are also stationary, but since the textures are uniform, the assumption should hold. Finally, we assume that most of the information in the spike train is carried by the frequency components, and it is valid to ignore the larger time dependencies in the signal.

To further illustrate this, consider an example that does not fulfill this condition. A surface that is first smooth, and then rough. The signal would change drastically halfway through. To effectively analyze such a signal, our method would need to recognize when the change occurs and classify the signal as two different materials in different sections. This is out of scope for the model utilized for this research, but might be implemented in a later stage of a neuromorphic system, where our model is used as the first stage. For a more detailed discussion of these assumptions, see Section 4.

Given these assumptions, two methods for converting spike trains to feature vectors were conceived. The first method is a more traditional analysis of the inter-spike interval (ISI). All ISIs of a spike train are computed by taking the time difference between all consecutive spikes. Then a histogram of all the ISIs in the spike train is computed. This histogram has 50 bins, where the count in each bin corresponds to a feature in the feature vector. The bin boundaries are chosen, so that the smallest bin is on par with the smallest refractory period, while the largest bin was empirically chosen to correspond to a large ISI, that rarely occurs in the spike trains. This method exploits the time information of the trains of spikes, which is a highly relevant coding method used in the biological brain [21].

The second method uses a spiking fast Fourier transform (sFFT) analysis. Which is performed through the following steps: First, the spikes are discretized into a time series signal with a sampling rate significantly higher than the highest spike rate. When a spike occurs, the signal is one, otherwise it is zero. Next, this signal is analyzed using a Fourier transform, then the phase information is discarded by taking the absolute value of the complex output. Finally, the resulting spectrum is converted to a feature vector of size 50 through downsampling. The downsampling is performed by reducing the fidelity of the spectrum using an appropriate low-pass impulse response filter and then decimating the signal to 50 values. To decimate a signal, all samples except every *n*th are dropped, where *n* is the length of the signal divided by 50. The same downsampling procedure is used to turn all frequency spectrums and sFFTs into feature vectors.

Both the ISI and sFFT analyses are global transformations, i.e. they do not preserve any local information. In addition, neither transform preserves the phase of the peaks. However, given our previous assumptions, neither of these features should be detrimental to the classification performance.

rm

## 2.4 Spiking phase-locked loop

Phase-locked loops are ubiquitous in digital electronics, where they are mainly used to lock onto repeating signals and synchronize clocks. Research suggests, that our brains use similar structures to process spikes [17]. Thus, we investigate the sPLL for the analysis of spikes generated by our encoders. In this section, the model of the sPLL used for this research is presented, and the equations are solved to derive some properties that are used to aid the optimization of the fully integrated classification problem.

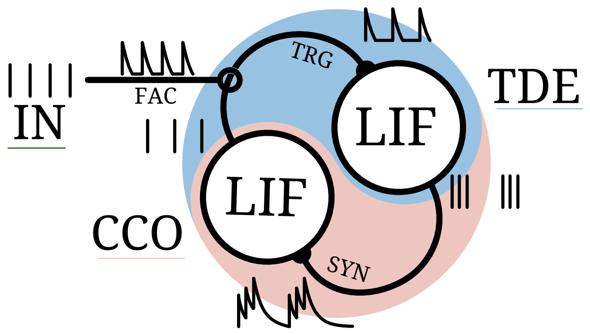


Figure 2: Visualization of the sPLL, reproduced with the permission of Mastella et al. [16]. The top section, with the blue background, represents the temporal difference encoder (TDE) synapse and neuron. The spiking input (IN) induces a current in the facilitatory synapse (FAC), which modulates the current produced by the trigger synapse (TRG). The trigger synapse then activates the current based leaky integrate and fire neuron (LIF), which produces spikes. The bottom section, with the red background, represents the current-controlled oscillator (CCO) neuron, which spikes continuously, with the spike rate increasing as the input spike rate rises. The CCO's input synapse is connected to the TDE neuron and the output is the trigger input of the TDE. Analogous to a conventional PLL, the TDE acts as a phase detector and the CCO as the oscillator.

The sPLL consists of a variable frequency oscillator and a phase detector. The variable frequency oscillator is a current based leaky integrate and fire neuron (LIF) neuron with a constant offset current, which causes the neuron to spike periodically, the frequency of the spikes varies based on the current of the synapse, thus the neuron is a current-controlled oscillator (CCO). The phase detector is implemented using a temporal difference encoder (TDE) which consists out of another LIF neuron, albeit with two synapses, where one acts as the facilitatory synapse and one as the trigger synapse. Only the trigger synapse is directly connected to the neuron, Thus, the membrane voltage of the neuron only increases when a trigger spike arrives shortly after a facilitatory spike. In practice, this works by modulating the gain of the trigger synapse using the output current of the facilitatory synapse. When there is no activity at the facilitatory synapse, the output current is zero and the trigger synapse does not react to any spikes. Only after the facilitatory synapse has received a spike and is producing an output current, is the trigger synapse able to produce a current, which in turn causes the membrane potential of the LIF neuron to rise.

To create the sPLL circuit, the input spikes are connected to the facilitator synapse, the output of the CCO is fed back to the trigger synapse. Finally, the output of the TDE is passed to the CCO, as well as, being used as the output of the sPLL, i.e. to detect whether the sPLL has locked onto the input signal. For a graphical representation of this arrangement, see Figure 2. Thus, whenever an input spike arrives shortly before the CCO spikes, the TDE produces an output and the frequency of the CCO is momentarily increased. Next, assuming the input is periodic, the next time the CCO will spike right before the next input arrives. Which, in turn, will not cause the TDE to spike again, so the CCO will slow down until the input spike arrives before the CCO spikes and the TDE triggers again. When this cycle repeats indefinitely, the CCO and input have the same spike rate and the sPLL is locked-in. However, the input data produced by the encoders, does not just contain a constant frequency spike train, it contains multiple frequencies of spikes along with some phase noise. Hence, the behavior outlined above does not happen as cleanly, but the sPLL still locks-in to these more noisy signals. Furthermore, there is a simple solution to detecting multiple frequencies in the signal: Using more sPLLs, or rather a network of sPLLs, all connected to the same input.

When there are no input spikes being received by the sPLL, the TDE will never trigger, as the facilitatory current is zero. However, the CCO neuron still spikes periodically, as the offset current causes the membrane potential to rise even without input spikes. The spiking frequency in this condition is called the free-running frequency, and it is important for tuning the sPLL as an input spike train with this frequency, is the lowest frequency spike train at which the sPLL will start locking in. Thankfully, deriving a closed form solution for this frequency is fairly trivial, and it will be presented below.

### 2.4.1 Differential Equations

Next, the differential equations governing the two neurons are presented, along with an explanation of all the parameters. Throughout this report, the following notation is used: Variables of the CCO have the $\tilde{o}$ subscript, while variables belonging to the TDE have the $\Delta$ subscript. Furthermore, ”syn” is short for synapse, ”fac” is a shorthand for facilitatory, and ”trg” is short for trigger.

**Synapse Equations** The current based synapse used to implement the LIF neuron integrates the voltage spikes arriving at the synapse, producing an output current. The current decays with a constant time constant. First, the synapse equation for the current controlled oscillator neuron is presented.

$$\frac{dI_{\tilde{o}}}{dt} = \frac{-I_{\tilde{o}} + g_{\tilde{o}}\ \mathrm{spk}_{\tilde{o}}(t)}{\tau_{\tilde{o},\mathrm{syn}}} \tag{1}$$

where:

- $I_{\tilde{o}}$ is the internal current state of the synapse, as well as the input to the neuron.
- $g_{\tilde{o}}$ is the voltage to current conversion factor, or gain, of the synapse.
- $\tau_{\tilde{o},\mathrm{syn}}$ is the time constant of the synapse.
- $\mathrm{spk}_{\tilde{o}}(t)$ is the sum of Dirac delta functions for every spike, i.e. $\mathrm{spk}_{\tilde{o}}(t) = \sum_i \delta(t - t^i_{\mathrm{spk}})$, where $t^i_{\mathrm{spk}}$ is the arrival time of spike $i$. I.e. it is positive when there is a spike and zero otherwise.

The equations for the synapse of the temporal difference encoder are similar, with the exception that the gain of the trigger synapse depends on the current of the facilitatory synapse.

$$\frac{dI_{\Delta,\mathrm{fac}}}{dt} = \frac{-I_{\Delta,\mathrm{fac}} + g_{\Delta,\mathrm{fac}}\ \mathrm{spk}_{\Delta,\mathrm{fac}}(t)}{\tau_{\Delta,\mathrm{fac}}} \tag{2}$$

$$\frac{dI_{\Delta,\mathrm{trg}}}{dt} = \frac{-I_{\Delta,\mathrm{trg}} + I_{\Delta,\mathrm{fac}}\ g_{\Delta,\mathrm{trg}}\ \mathrm{spk}_{\Delta,\mathrm{trg}}(t)}{\tau_{\Delta,\mathrm{trg}}} \tag{3}$$

where:

- $I_{\Delta,\mathrm{fac}}$ and $I_{\Delta,\mathrm{trg}}$ are the internal current state of the synapses.
- $g_{\Delta,\mathrm{fac}}$ and $g_{\Delta,\mathrm{trg}}$ are the voltage to current conversion factors, or gains, of the synapses.
- $\tau_{\Delta,\mathrm{fac}}$ and $\tau_{\Delta,\mathrm{trg}}$ are the time constant of the synapses.
- $\mathrm{spk}_{\Delta,\mathrm{fac}}(t)$ and $\mathrm{spk}_{\Delta,\mathrm{trg}}(t)$ are the instantaneous input spike voltages as described above.

**Neuron Equation** The current output of the synapse is integrated again by the LIF neuron. Until a threshold voltage is reached and the neuron fires and resets its internal potential. The equations below do not describe the firing of the neuron, as it would make the equations much harder to solve and can be added after the equations are solved. The equations for both the CCO neuron and the TDE are basically the same, except for the offset current being added to the input current of the CCO.

$$\frac{dV_{\tilde{o}}}{dt} = \frac{I_{\tilde{o}} + I_{\tilde{o},\text{offset}}}{C_{\tilde{o}}} - \frac{V_{\tilde{o}}}{\tau_{\tilde{o}}} \tag{4}$$

$$\frac{dV_{\Delta}}{dt} = \frac{I_{\Delta,\text{trg}}}{C_{\Delta}} - \frac{V_{\Delta}}{\tau_{\Delta}} \tag{5}$$

where:

- $V_{\tilde{o}}$ and $V_{\Delta}$ are the membrane potentials of the two neurons. When these values surpass the threshold voltage, the neuron will fire.
- $C_{\tilde{o}}$ and $C_{\Delta}$ are the capacitances of the neurons.
- $\tau_{\tilde{o}}$ and $\tau_{\Delta}$ are the time constants of the neurons.
- $I_{\tilde{o},\text{offset}}$ is a constant current continually fed to the CCO neuron, to cause it to periodically fire even without input spikes.

These equations as they are presented here are missing one important component, the refractory period. This is implemented as a fixed delay after each neuron fires, during this time the membrane potential of the neuron ($V_{\tilde{o}}$, $V_{\Delta}$) is fixed to zero until the refractory period is over. The duration of the delay is $t_{\tilde{o},\text{refr}}$ and $t_{\Delta,\text{refr}}$ for the CCO and TDE, respectively.

Given these equations, we can now derive the free-running frequency of the CCO neuron. Which is used to find the correct offset current to induce the sPLL to be sensitive to a specific frequency.

$$f_{\tilde{o},\text{free}} = \frac{1}{t_{\tilde{o},\text{refr}} + \tau_{\tilde{o}} \ln\left(\frac{I_{\tilde{o},\text{offset}}\tau_{\tilde{o}}}{I_{\tilde{o},\text{offset}}\tau_{\tilde{o}} - V_{\tilde{o},\text{thr}}C_{\tilde{o}}}\right)} \tag{6}$$

### 2.4.2 Tunable parameters

Given the equations above, we have a total of 11 tunable parameters. For the CCO neuron, there are: $\tau_{\tilde{o}}$, $\tau_{\tilde{o},\text{syn}}$, $g_{\tilde{o}}$, $C_{\tilde{o}}$ and $I_{\tilde{o},\text{offset}}$. For the TDE there are: $\tau_{\Delta}$, $\tau_{\Delta,\text{fac}}$, $\tau_{\Delta,\text{trg}}$, $g_{\Delta,\text{trg}}$, $g_{\Delta,\text{fac}}$ and $C_{\Delta}$. Additionally, there are four more parameters, these are the two refractory periods $t_{\tilde{o},\text{refr}}$ and $t_{\Delta,\text{refr}}$, as well as, the two voltage thresholds for the neuron firing $V_{\tilde{o},\text{thr}}$ and $V_{\Delta,\text{thr}}$. Giving us a total of 15 parameters.

Not all the aforementioned parameters are varied during our hyperparameter tuning. The refractory periods are set to $t_{\tilde{o},\text{refr}} = t_{\Delta,\text{refr}} = 10^{-3}$ to simplify the tuning, as these are realistic values. Furthermore, to remove redundancies in the model, the capacities and membrane potential spike thresholds are set to $C_{\tilde{o}} = C_{\Delta} = V_{\tilde{o},\text{thr}} = V_{\Delta,\text{thr}} = 1$. Changing these values is not necessary, as an sPLL with identical behavior can be created by altering the gains and neuron time constants. Finally, we do not tune the CCO offset current $I_{\tilde{o},\text{offset}}$, because this value will be computed to set the free-running frequency of the CCO to a specific value.

To ensure proper operation of the sPLL there are a few constraints on these parameters. The TDE should have lower time constants for the synapse, than for the neuron. Such that, the two synapses filter for spikes arriving shortly after another, while the neuron itself may integrate multiple events, or produces multiple spikes for spikes arriving very shortly after one another. Additionally, the two synapses should fully reset by the time the next input spikes arrives. Thus, the two time constants should be small enough to satisfy the following inequalities: $f_{in} < \frac{1}{\tau_{\Delta,\text{fac}}}$ and $f_{in} < \frac{1}{\tau_{\Delta,\text{trg}}}$, where $f_{in}$ is the highest expected input frequency.

The parameters for the CCO only have one important constraint, the offset current needs to be high enough to cause the neuron to spike periodically. Hence, the following inequality must be satisfied: $V_{\tilde{o},\text{thr}}\, C_{\tilde{o}} < I_{\tilde{o},\text{offset}}\, \tau_{\tilde{o}}$. In practice, we adjust $I_{\tilde{o},\text{offset}}$ to achieve a specific spiking frequency, which will always satisfy the inequality given above.

### 2.4.3 Scaling the sPLL

In this study, the sPLL is not used by itself, but rather in a network of many sPLLs, each being tuned to a different frequency. Theoretically, it would be possible to tune all the parameters of each sPLL in the network. However, this would be computationally infeasible and also exceptionally difficult to implement in hardware. Thus, only a single sPLL is tuned, and is subsequently scaled up and down to respond to different frequencies. We conceived two methods, by which to accomplish this scaling: Varying the offset current ($I_{\tilde{o},\text{offset}}$) and time-scaling all parameters of the sPLL.

Varying the offset current parameter of the sPLL allow us to tune it to any free-running frequency. To achieve this, Equation 6 is solved for $I_{\tilde{o},\text{offset}}$, and the correct offset current is computed. The benefit of this scaling method, is that it is straight forward to implement in hardware, with the same circuit being used for all sPLLs and only one constant current being varied between them. Nevertheless, while we can tune the sPLL to almost any free-running frequency, it might not be very well-behaved. An sPLL which is sensitive in the 50 Hz regime will struggle around 500 Hz, even if the free-running frequency is increased, as all the other parameters are tuned for lower spike rates and the synapse currents and neuron membrane potentials will likely saturate due to the large number of spikes. Thus, we have conceived a second way to scale sPLLs.

Given a set of parameters, it is possible to find a new set of parameters, which produce a faster or slower sPLL which otherwise behaves identical. This is useful, as it allows us to find a single set of parameters, which work well for detecting a specific frequency and use a scaled version of these parameters to detect different frequencies. Suppose, $s_t$ is the factor by which we want to slow down or speed up the sPLL. To derive a new set of parameters, divide all the capacitances, time constants and refractory periods by $s_t$. The gains and offset current stay the same. This can be derived by substituting $t$ by a scaled $t' = t \cdot s_t$ and then solving the system equations for the new parameters. Throughout this research, we refer to this type of scaling as the time-scaled sPLL.

## 2.5 sPLL Network

The network of sPLLs analyzed in this research consists out of multiple sPLLs run in parallel, each with the same input spike train. Two different methods for choosing sPLL Network parameters are analyzed, resulting in two different networks. Each network consists out of 50 different sPLLs, whose output spike counts are used to create a feature vector. The first one being a network of time-scaled sPLLs and the second one only varying the offset current between sPLLs, see Section 2.4.3. Both methods only need a single set of parameters, along with some parameters describing to which frequencies the sPLLs are tuned.

The architecture used in this research is based on the sPLL network described by Mastella et al. [22], they also vary the offset current of the CCO, but proceed to pass the output of the sPLL through a LIF neuron based high pass filter and then into a winner take-all network. Due to the multi frequency nature of the texture classification task, a simple winner take-all network would not suffice. Thus, the decision was made to convert the output of the sPLLs to spike counts and use these directly to create a feature vector, which is further analyzed using non neuromorphic means. In practice, classification might be implemented using a spiking neural network layer, between the high pass filter and the winner take-all layer.

Each sPLL in the network is tuned to respond to a different frequency. How this is done is explained above. However, we still need to choose which frequencies are used for the different sPLLs. As this also depends on the dataset, we introduce three more hyperparameters. The first two are quite logical, they are the lowest and highest frequency respectively. The lowest frequency is between 30 and 100 Hz, while the highest is between 300 and 700 Hz. These ranges were chosen based on the frequency response range of the pacinian afferent in humans, about 20 to 450 Hz at room temperature [23], as well as some preliminary data analysis, there are peaks at around 600 Hz in both datasets.

The last sPLL network hyperparameter decides how we distribute the 48 other frequencies, between the lowest and highest frequency. The simplest approach is a linear distribution. The drawback of distributing the frequencies linearly is, that the sPLLs are not just sensitive to their base frequency, but also all the harmonics, i.e. integer multiples of the base frequency. Thus, there can be a lot of overlap between the frequency ranges each sPLL is sensitive to. To mitigate this, the frequencies might also be distributed exponentially, the parameter β controls the transition between a linear and an exponential distribution. If β is 0, the numbers are distributed exponentially, and as β approaches ∞ the frequencies are linearly distributed. The frequencies are calculated according to the following formula:

$$f_i = (f_{\text{end}} + \beta)^{\frac{i}{n}} (f_{\text{start}} + \beta)^{1-\frac{i}{n}} - \beta$$

where:

- $f_i$ is the frequency of the $i$th sPLL node,
- $f_{\text{start}}$ and $f_{\text{end}}$ are the start and end frequency as discussed above.
- $n = 50$ is the number of sPLL nodes in the network.

## 2.6 Regression analysis

Above, three different methods to convert spike trains into feature vectors are given, the spiking fast Fourier transform (sFFT), the ISI histogram and the sPLL network TDE output rate. Each method takes as input the spikes produced by an encoder and produces a 50 dimensional vector. These feature vectors in themselves are not enough to classify different surface textures. A classification architecture is needed. There are many such methods, including unsupervised methods such as k-means clustering and supervised methods including deep neural networks. As we have labeled training data, a supervised approach was selected, more specifically a ridge regression was chosen. A ridge regression consists out of a single linear layer taking the feature vector as an input and producing an output value for every class, the highest output value is chosen as the predicted class. Furthermore, large weights of the linear layer are penalized by an L2 normalization factor $\alpha$. This penalty increases the classifier's robustness to collinearity of the feature vectors, which is important as frequency harmonics and overlapping lock-in ranges produce feature vectors with potentially highly correlated features.

Our goal is to tune encoders and the sPLL network, so that the regression classifier can be trained to a high accuracy. It is standard practice to not test the classifier with the same data it was trained with, to mitigate overfitting. One way to achieve this, is by keeping a separate set of training and test data. However, since the VibTac dataset only has 20 seconds of data per class, it would be great to use all of it for training. Thus, the decision was made to use stratified K-fold cross validation, where K is 5. The set of feature vectors is split into 5 sets, and then 5 different classifiers are trained, always using one of the 5 sets for testing the accuracy and the rest for training. Additionally, when splitting the data, we ensure that all the 5 sets have a similar number of elements from each of the classes.

## 2.7 Numerical Simulation

The sPLL as it is described above can be sensitive to the precise timing of input spikes, as small changes in spike times can make the difference between the TDE triggering or not. Thankfully, this does not impact the lock-in range significantly, but the duration until the sPLL locks-in can deviate a bit. Both biological systems and neuromorphic implementations are not perfect, and the precise timing of spikes may vary. Thus, to approximate this and to get less varied results for optimizing hyperparameters, the simulation is repeated multiple times with slightly different conditions, see Section 2.9. Repeating the simulation multiple times, for each input sample, and then for many different hyperparameters results in a high number of simulations which need to be run. To allow for quick simulation, an efficient method for simulating the system was devised. This method steps through time from one spike to the next, after each spike we set up equations for each neuron and solve them for the next spike time. Then we skip to that spike time and update the equations to account for the spike. This is repeated until the end of the simulation. Below, the solutions to the equations are presented, along with how they are updated when spikes arrive or when the neuron triggers.

### 2.7.1 System solutions

To solve this system of differential equations, we ignore incoming and outgoing spikes, along with the refractory period. Afterward, the behavior of the system is rectified, by updating variables when specific conditions are met to account for input spikes and send output spikes. Below are the solutions to Equations (1) to (5).

$$I_{\tilde{o}} = \exp(-\frac{t}{\tau_{\tilde{o},\text{syn}}} + c_{\tilde{o},\text{syn}}) \tag{7}$$

$$V_{\tilde{o}} = \frac{\tau_{\tilde{o},\text{syn}}\tau_{\tilde{o}}}{C_{\tilde{o}}(\tau_{\tilde{o},\text{syn}} - \tau_{\tilde{o}})} I_{\tilde{o}} + \frac{I_{\tilde{o},\text{offset}}\tau_{\tilde{o}}}{C_{\tilde{o}}} - \exp(-\frac{t}{\tau_{\tilde{o}}} + c_{\tilde{o}}) \tag{8}$$

$$I_{\Delta,\text{fac}} = \exp(-\frac{t}{\tau_{\Delta,\text{fac}}} + c_{\Delta,\text{fac}}) \tag{9}$$

$$I_{\Delta,\text{trg}} = \exp(-\frac{t}{\tau_{\Delta,\text{trg}}} + c_{\Delta,\text{trg}}) \tag{10}$$

$$V_{\Delta} = \frac{\tau_{\Delta,\text{trg}}\tau_{\Delta}}{C_{\Delta}(\tau_{\Delta,\text{trg}} - \tau_{\Delta})} I_{\Delta,\text{trg}} - \exp(-\frac{t}{\tau_{\Delta}} + c_{\Delta}) \tag{11}$$

where $c_{\tilde{o},\text{syn}}$, $c_{\tilde{o}}$, $c_{\Delta,\text{fac}}$, $c_{\Delta,\text{trg}}$ and $c_{\Delta}$ are the integration constants which need to be updated whenever the spikes occur. Note, there is a singularity when $\tau_{\tilde{o}} = \tau_{\tilde{o},\text{syn}}$ or $\tau_{\Delta} = \tau_{\Delta,\text{trg}}$, as we divide by zero. There is a different solution to the equations, when the time constants are equal, but it is only valid in that case. For the purpose of this research, we simply always kept a slight difference between the time constants, to avoid the singularity.

### 2.7.2 Simulation procedure

To simulate the sPLL, a general purpose algorithm was devised. This algorithm works with blocks. Each block can have multiple synapses, which receive input spikes, as well as, being able to produce output spikes. Both the CCO neuron and the TDE are each implemented as a block, and so is the input. The algorithm interacts with these blocks using three methods: `next_spike_time`, `incoming_spike` and `advance`. These three methods are described in more detail below. Each block is implemented as a stateful trait object and keeps track of internal variables. Since the input is just another block, it does not need to be handled specially. The algorithm used to simulate the sPLL system is presented in Algorithm 1.

**Algorithm 1** Spiking simulation procedure

```
1: $t \leftarrow 0$
2: $B \leftarrow \{$TDE, CCO, input$\}$                          ▷ Initialize set with statefull block objects.
3: while $t \leq$ simulation duration do
4:     $t_n \leftarrow \infty$
5:     for $b$ in $B$ do                                        ▷ Find the next spike time in all blocks.
6:         $t_n \leftarrow \min(t_n, \texttt{next\_spike\_time}(b,t))$
7:     end for
8:     if $t_n = \infty$ then return                             ▷ Return, if there are no more spikes.
9:     $t \leftarrow t_n$
10:    $S \leftarrow \{\}$
11:    for $b$ in $B$ do                                        ▷ Advance all blocks, collecting the spikes.
12:        $s \leftarrow \texttt{advance}(b,t)$          ▷ advance returns the index of the target synapse, if a spike occurs.
13:        if $s \neq -1$ then
14:            $S \leftarrow S \cup \{s\}$                      ▷ Add the synapse to the set of spiking synapses.
15:        end if
16:    end for
17:    for $s$ in $S$ do                                        ▷ Propagate all spike to the respective synapses.
18:        $b \leftarrow B[\text{syn\_to\_block\_idx}(s)]$                                  ▷ Look up target block.
19:        $\texttt{incoming\_spike}(b,s)$
20:    end for
21: end while
```

**`next_spike_time`** This method returns the next time point when the given block will produce a spike. For the input block, this is trivial, as all spikes are known ahead of time. For the neuron blocks, we need to solve $V_{\tilde{o},\text{thr}} = V_{\tilde{o}}$ and

$V_{\Delta,\mathrm{thr}} = V_\Delta$ for $t$, using Equations (8) and (11). There are no closed form solutions to these equations, so a numerical approach was chosen. First, the interval of possible solutions is found by finding the roots of the derivative and then using the Brent method [24] to solve for the exact value of $t$. Should no solution be found, no spike will occur under the current conditions and the method returns $\infty$.

**`advance`** The advance method advances the block to the given time. If the block spikes at that time, the integration constants are updated to reset the membrane potential, and the index of the synapse that receives the spike is returned. Otherwise, -1 is returned. Note that the advance method also handles the refractory period by updating an internal timer of the block and keeping the membrane potential at zero until the refractory period is over.

**`incoming_spike`** The incoming spike method updates the given block so that the correct synapse current is increased. This is done by evaluating the synapse equation to get the momentary current, then the gain is added to the current, and once again solve for the integration constants to update the equations.

## 2.8 Hyperparameter optimization

The goal of hyperparameter optimization is to find the best set of parameters for classifying the textures in a dataset. This is done using the Optuna optimization framework [25]. A range was selected for each hyperparameter based on the constraints given above, and the full experiment was repeated for 500 runs, with parameters chosen by Optuna. Each run simulates the 50 sPLLs for every data sample three times with different phase noise, see Section 2.9. The best set of parameters were used to generate the plots and are presented below. For a table with the hyperparameter ranges, see Table 5.

Internally, Optuna uses a tree-structured parzen estimator (TPE) to choose a set of hyperparameters for each run. This estimator uses Bayesian learning to update an estimation of the function $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is the configuration space, and $f$ returns the estimated accuracy of the sPLL network. Unlike traditional Gaussian processes, which model the expected performance directly, TPE divides hyperparameter samples into two groups: "good" and "bad," based on a threshold of performance. It then models each group separately, constructing two probability density functions. $l(x)$ for the good samples and $g(x)$ for the worse ones. Using these distributions, TPE calculates the likelihood of a parameter value being in each group, then it samples from $l(x)$ to focus on promising regions, while keeping some exploration by occasionally considering $g(x)$. This results in an algorithm that can quickly find a good set of hyperparameters. One limitation of the TPE, is that it does not consider correlations of hyperparameters. Thus, if the "good" value of one parameter depends on the value of another parameter, TPE might struggle to converge on a good value quickly.

## 2.9 Phase noise

The neuromorphic circuits, as they are simulated in this work, are perfectly predictable. Thus, repeating an experiment with the same inputs and configuration, results in the same output. However, this is not the case for real world systems. In the brain, noise is introduced by the inherent fluctuations of the ion distributions, as well as at the synapses [26]. Even in non-biological circuits, noise is still omnipresent. The power supply is generally shared by multiple transistors, so when one transistor switches, it reduces the voltage supplied to the rest. With good circuit design these effects can be minimized, but they can not be entirely eliminated. Additionally, memristors often behave stochastically, so they don't always produce the exact same voltage, even with the same input conditions [27, 28]. It has even been shown, that this noise can be exploited to improve learning performance in spiking systems [29].

Phase noise is introduced in our models, to augment the data and improve the resilience of our tuned hyperparameters. As opposed to frequency noise, phase noise shifts the time of every spike slightly, but does not alter them further. As the total number of spikes in each time period stays constant, the frequency of the spike train is not affected, as long as the standard deviation of the noise is small enough. To apply noise to a spike train, the timing of each spike is altered by a value picked from a normal distribution: $t'_{\mathrm{spk}} = t_{\mathrm{spk}} + \mathcal{N}(0, \sigma^2)$. The standard deviation is chosen to be on the order of the smallest inter-spike interval. A small amount of noise also allows us to repeat experiments for the hyperparameter tuning and get a better estimation of the accuracy of a set of hyperparameters. For the tuning, we used a noise value of $5 \cdot 10^{-4}$ and repeated the experiment three times. Note, when determining the accuracy using the regression classifier, we only use each sample once, i.e. we do not use a sample multiple times with different noise values added. Thus, we train three different classifiers and average the accuracies.

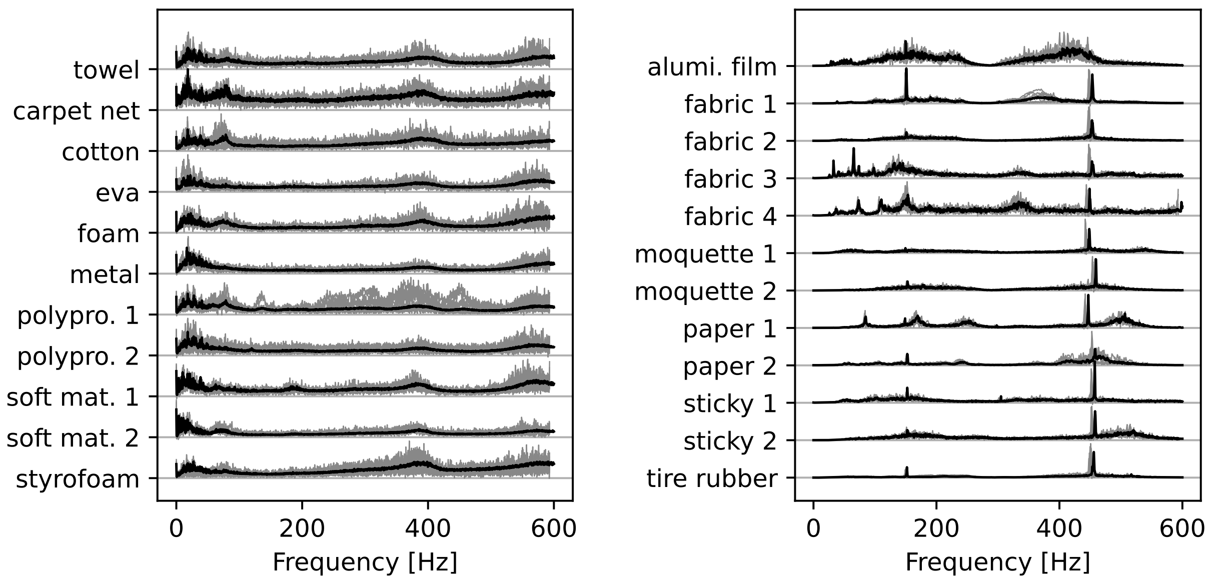
## 3 Results

In the following, the results of the experiments described above are presented, along with some explanations and analysis. First, the input data is investigated. Then, the outputs produced by the pressure encoders are presented. Next, some analysis of a single sPLL is shown. And finally, the results of the whole sPLL network with the classifier are used to answer the research questions.

### 3.1 Data Collection

As described in Section 2.1, we use two different datasets, both recording the interaction between a surface and a moving sensor. Below we present an analysis of both datasets and outline some of the differences between them. This section concerns the bottom most preliminary analysis in Figure 1.

Here, we first present the average frequency spectrums of each of the classes in the dataset, then we do some preliminary analysis, to check if the frequency information in the data is enough to distinguish the classes with high accuracy. The spectrums are computed up to 600Hz, and are plotted in Figure 3. It is evident, that the two datasets are inherently different. The spectrums of BioTac are generally more varied within classes, which is compounded by the fact, that the BioTac dataset also contains more samples per class. Additionally, VibTac appears to have more contrast in the spectrums, with peaks rising far above the average spectral power. Both datasets contain classes which have spectrums that look rather similar to each other. Thus, we check if the spectrums are enough to train a classifier to differentiate between all classes.



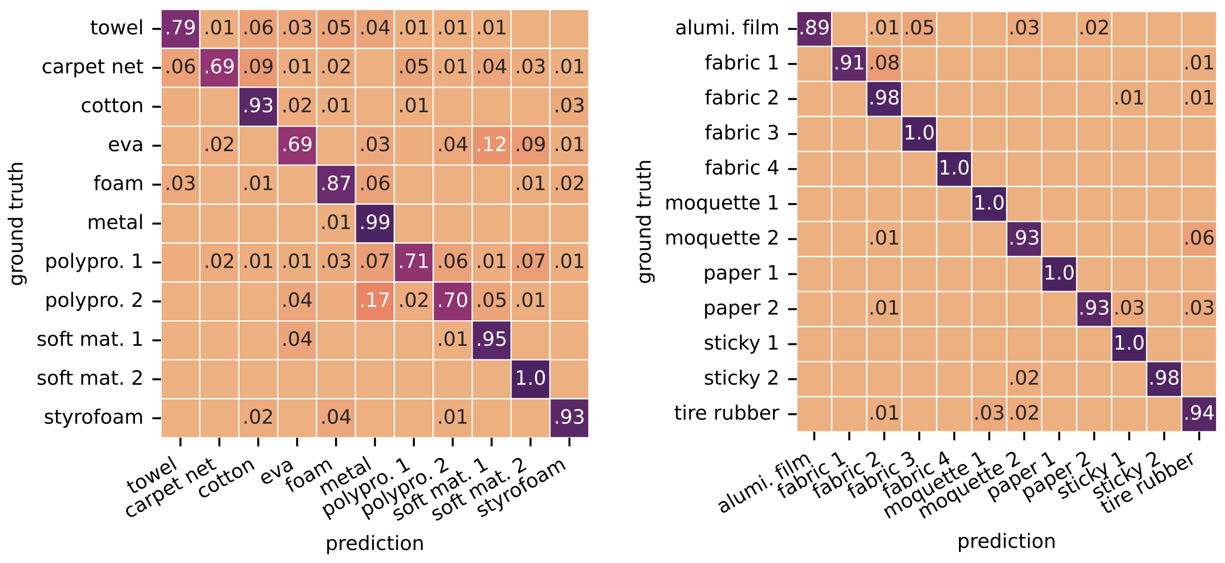
(a) BioTac dataset

(b) VibTac dataset

Figure 3: Average frequency spectrums for all the classes of the datasets. The light gray area is the extent of the spectrum of all samples for each class. The spectrums are computed using the absolute value of an FFT of the sampled pressure values. To aid classification, small in class variability with clear differences between classes are ideal.

First, we convert the frequency spectrums to feature vectors. This is done by decimating the spectrums from about 1200 values to 50, see Section 2.3. Every value in the feature vector corresponds to the spectral power in a band of about 12Hz. The reason for this decimation, is that it makes the accuracies computed here comparable with the ones computed in later analyses. Afterward, two classifiers are trained, the confusion matrices along with the accuracies of these classifiers are presented in Figure 4. The classifier, trying to predict the VibTac classes, outperforms the BioTac classifier. This is likely due to the increased variability within BioTac classes, along with the higher number of samples.

Note, these confusion matricies were also used to remove some of the classes from the BioTac dataset, as explained in Section 2.1.



(a) BioTac dataset confusion matrix. Accuracy: 84.4%.

(b) VibTac dataset confusion matrix. Accuracy: 96.5%.

Figure 4: Confusion matrix of classifiers trained on features extracted straight from the data using a downsampled frequency spectrum. Each sample in each dataset was converted to a feature vector by the following steps: compute FFT spectrum, take absolute value, downsampled to 50 values. These feature vectors were then used to train a regression model, which resulted in the presented confusion matrices.

We theorize, that the higher variability in the BioTac dataset might stem from multiple sources. First, the method of moving the sensor along the material is less rigid and likely introduces its own vibrations, due to the size and degrees of freedom of the robotic arm. Even if the arm performs very well, the vibrations we are measuring are caused by displacements smaller than 0.1 mm, so even a little bit of imprecision or vibrations in the robotic arm can be picked up by the sensor. Another factor at play might be the much larger contact area between the stylus/finger and the material. For VibTac, the stylus only touches the material at a point, while BioTac drags a patch of synthetic skin across the material. Thus, the many contact points might average out and produce a more ambiguous signal. This would explain the much wider peaks in the BioTac spectrum as compared to the VibTac spectrum.

## 3.2 Pressure encoders

This section presents the spike trains produced by the encoders and the preliminary analysis done on those spike trains. As explained above, we convert the spike trains to feature vectors using two different methods, and then check how well a linear regression classifier can tell apart the different classes. This section mainly presents the preliminary analysis presented in the middle of Figure 1. For some example spike trains, see Figure 5. We carefully selected a representative sample of classes, to present different behaviors of the encoders.

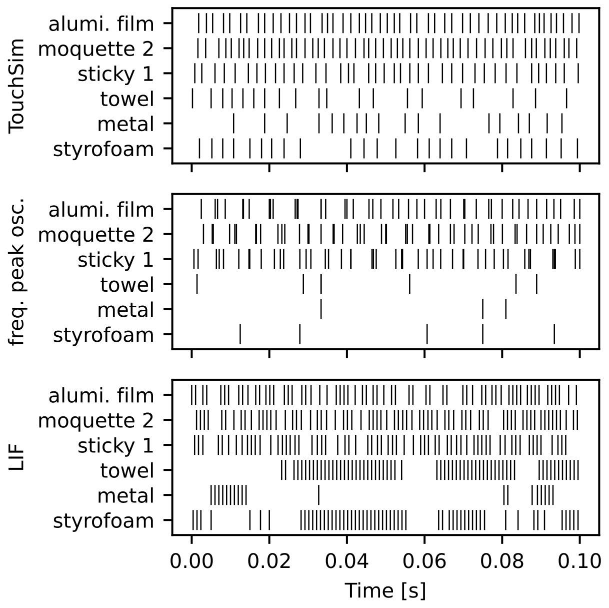


Figure 5: Example of spikes produced by each of the three encoders. The encoders convert the input pressure values to spikes, see Section 2.2. Here, a 100ms window is presented for carefully selected samples from both datasets. The classes aluminum film, moquette 2, and sticky 1 are from the VibTac dataset, while the classes towel, metal, and styrofoam are from the BioTac dataset.

Due to the structural differences between the different encoders, the spike trains they produce vary in their nature. The frequency peak oscillator has the largest difference in spike frequencies, with some classes producing very low-frequency spike trains. Both the LIF encoder and the TouchSim encoder ultimately use a LIF neuron to produce spikes. However, the pre-processing done to the pressure signal in the TouchSim afferent ensures that the spike trains are more consistent, without large gaps in the spike train. In the pure LIF encoder, these gaps are produced when the pressure signal is negative for extended periods of time, so only when low-frequency components are present. We conjecture that these gaps are beneficial for classification performance, as the LIF encoder generally outperforms the TouchSim encoder, as presented in Table 1.

| Configuration | | Encoder | | |
|---|---|---|---|---|
| Dataset | Feature extraction method | LIF | TouchSim | f. peak osc. |
| VibTac | sFFT | 98.9% | 96.1% | 61.5% |
| BioTac | sFFT | 63.4% | 53.9% | 35.3% |
| VibTac | ISI hist. | 87.6% | 72.1% | 31.2% |
| BioTac | ISI hist. | 38.3% | 39.7% | 21.5% |

Table 1: Classification accuracies of the preliminary encoder analysis. The values are the test accuracies of the linear regression classifier trained on the feature vector produced by one of the two feature extraction methods, for each of the three encoders analyzed in this paper. The sFFT method computes an FFT on the discretized spike train signal and decimates the resulting spectrum to 50 values, while the ISI histogram method computes a histogram of the inter spike intervals with 50 bins.

Of the two methods for converting spike trains to feature vectors, as presented in Section 2.3, the sFFT generally outperformed the ISI histogram method. Hence, we include some examples of these frequency spectrums below in Figure 6. Note, before these spectrums are used for classification, they are first decimated from 2000 data points to only 50 data points, see Section 2.3. After downsamepling, each feature in the feature vector corresponds to a frequency band with a width of 20 Hz.

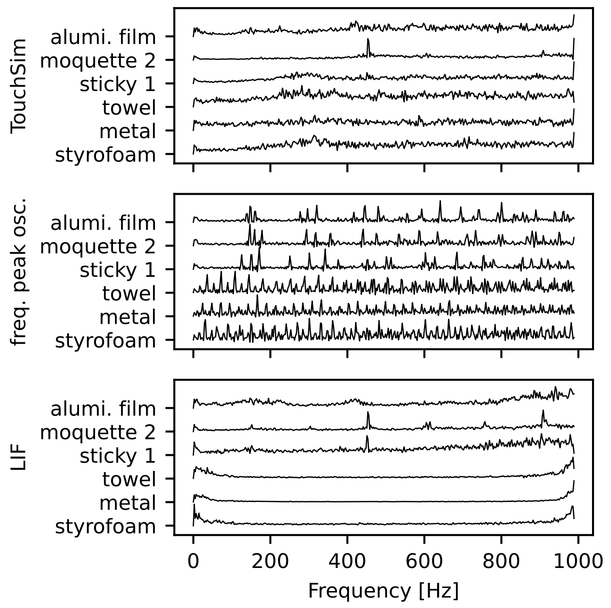


Figure 6: Example of the frequency spectrums of the spike trains produced by each of the three encoders. The encoders convert the pressure signals from the datasets into spikes. For a subsection of the spike trains, see Figure 5. These spikes are then converted to a binary sampled signal, which is converted to a frequency spectrum by an FFT. The samples aluminum film, moquette 2, and sticky 1 are selected from the VibTac dataset, while the samples towel, metal, and styrofoam are from the BioTac dataset.

When comparing the example spikes of the frequency peak oscillator with the resulting spectrums, an inherent limitation of the sFFT becomes visible. Take, for example, the metal surface, in Figure 6. The peak oscillator spikes at a very low frequency, compared to the other encoders/classes. Yet, there are lots of high frequency components in the spectrum. We have two explanations for this behavior. First of all, it might be harmonics of the square waveforms of the spikes. However, we would expect the same harmonics to show up for other encoder/class pairs. An alternative explanation is the three spike trains which are added together produce lots of varied interspike times. Which in turn get detected by the sFFT as many more frequencies, besides the original three. It is possible, that this would improve if a longer time sample than two seconds is used for the sFFT.

Finally, we computed the feature vectors, from the spectrums or directly from the distribution of ISI, for each dataset, and trained a classifier using the spike trains. For the test accuracies of the trained classifier, see Table 1. There are a couple observations. Foremost, as in the preliminary analysis of the data, it appears that VibTac classes are more readily distinguishable from one another. Second, using the sFFT for feature extraction, works better than the ISI histogram, this was expected, as a Fourier takes dependencies between distant spikes into account, while the ISI only considers consecutive spikes. Generally, the LIF encoder performs the best, closely followed by the TouchSim encoder. Surprisingly, the LIF encoder even outperforms the classifier trained on the VibTac pressure data directly, see Figure 4b. Finally, the frequency peak oscillator encoder performs significantly worse than the other two encoders. To inspect the confusion matrices of the encoder classifiers, see Figure 14 in the Appendix.

## 3.3 Single sPLL

In this section, the responses of individual sPLLs to the input spikes produced by the encoders is presented. The sPLLs in this section have been tuned in a network of sPLLs to enable high classification accuracies, when their TDE spike rates are passed to a classifier as feature vectors, see Section 2.8. We found that there are multiple different mechanisms, by which an sPLL selects for frequency ranges. It can lock-in to a specific frequency, as we explained above, and additionally, by changing the parameters, the sPLL can produce different non-linear relationships between the input frequency and the spike rate of the TDE. For an example response of an sPLL to the encoded inputs, see Figure 7. This plot demonstrates the sPLL working as described in Section 2.4, although the frequency of the oscillator only changes marginally. This is due to the small CCO synapse gain parameter for the sPLL, which leads to a small lock-in range. We conjecture that this is optimal, due to the exact spike frequencies of the frequency peak encoder. The relationship between the aforementioned parameter and the lock-in range will be explained further below.

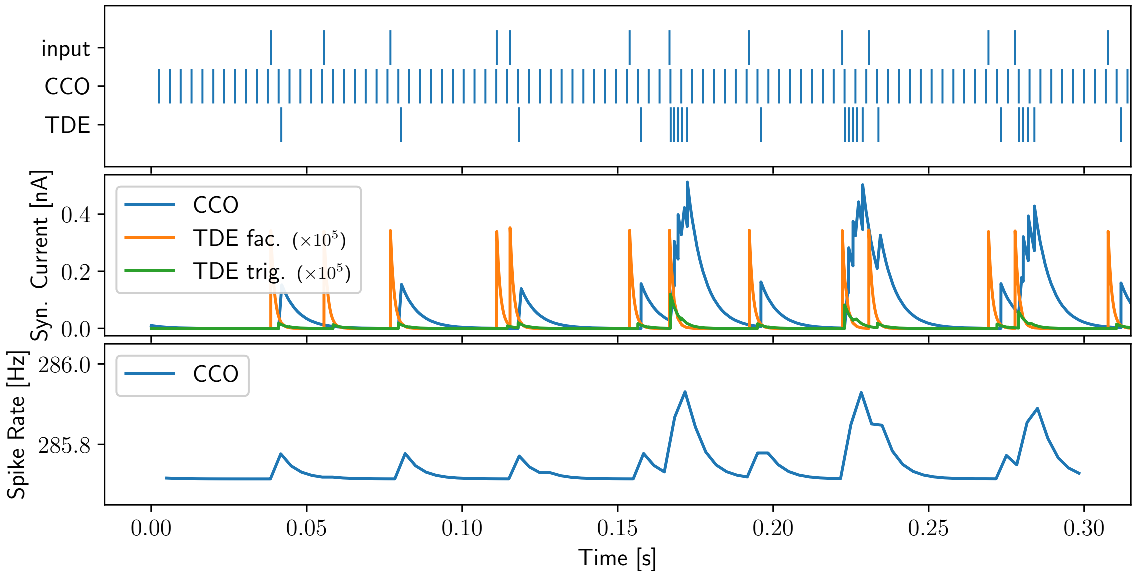
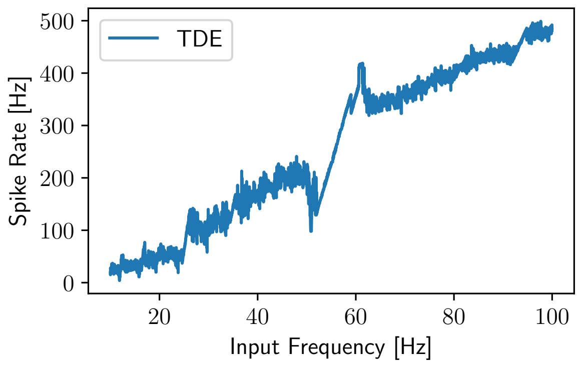


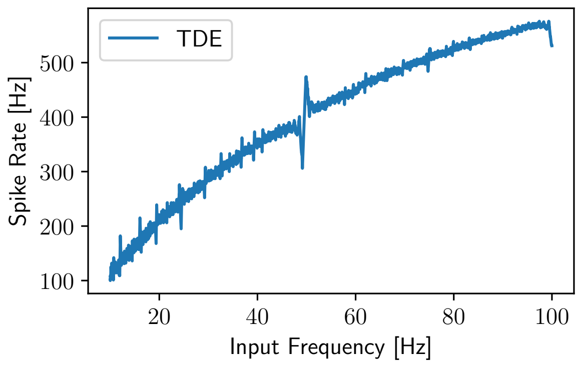
Figure 7: Here the spiking behavior of an sPLL circuit is presented. The input spikes are derived from the pressure signal from the BioTac carpet net texture, encoded using the frequency peak oscillator. The spikes get converted to currents in the synapses, which are presented in the middle plot, for the connection of the synapses and neurons, see Figure 2. At the bottom, the spike rate of the CCO neuron is plotted. The sPLL used in this figure was tuned to produce distinct feature vectors in a network of sPLLs, according to our method.

It is hard to make direct inferences from spike plots of the sPLL, thus, in further figures, we also plot the frequency response of the sPLL. More specifically, the TDE spike counts of many sPLLs reacting to a different constant input frequency each. For these plots, we are using the optimized sPLLs and tune them so the CCO spikes with a frequency of 50Hz without any input spikes. Hence, it will be most sensitive to input spike trains at roughly 50Hz.

First, we examine an sPLL, which is showing a clear spike in the TDE count. Remember, when the sPLL is locked-in, the TDE will spike periodically, to keep the CCO at the same frequency and phase as the input. In the frequency response plot we can see this as a fast constant rise in the spike count, over a small frequency range. For two examples of this, see Figure 8. Note, how at an input frequency of 50 Hz (the frequency the sPLL is tuned to), there is a local minimum in the spike rate of the TDE. As the CCO will spike at the same rate as the input without any input from the TDE. Afterward, there is a linear rise in spike rate as the TDE spikes more to increase the frequency of the CCO. The sPLL is locked in, from the minimum at 50 Hz until the consistent linear rise stops.



(a) sPLL tuned for VibTac encoded using the frequency peak oscillator, in a time-scaled sPLL network ($g_{\tilde{o}} = 5.77$).



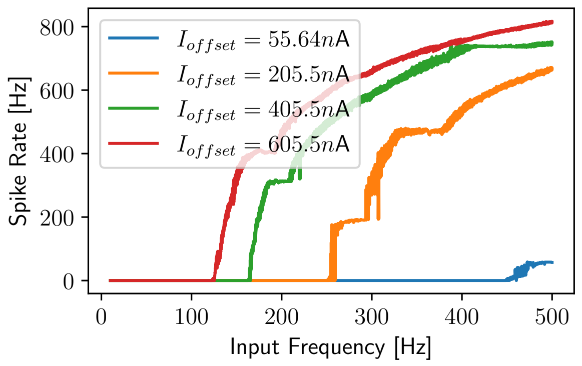
(b) sPLL tuned for VibTac encoded using the frequency peak oscillator, while only varying the offset current throughout the sPLL network ($g_{\tilde{o}} = 0.568$).

Figure 8: Frequency response of two different sPLLs. Both sPLLs were optimized according to our method, to produce high accuracies when they are used in a network of sPLLs fed to a regression classifier. Thus, they represent our best set of parameters given the dataset and encoder. To compute the frequency response, the sPLLs are set to have a free-running frequency of 50 Hz, and then they are run with constant frequency inputs. This is repeated for 2000 different input frequencies between 0 and 100 Hz. The spike rate of the TDE neuron is plotted against the input frequency. Finally, the CCO synapse gain $g_{\tilde{o}}$ is stated, as this parameter has a significant influence on the response.

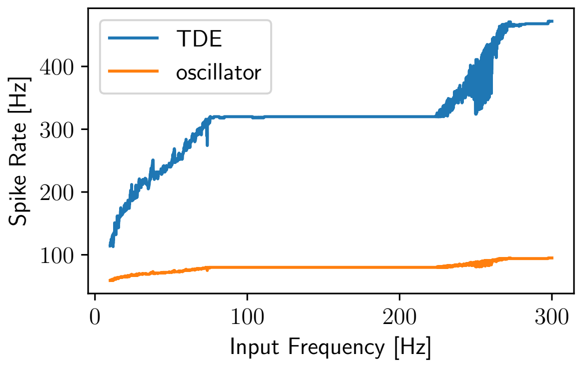
An important parameter for tuning the width of the lock-in range is the CCO synapse gain $g_{\tilde{o}}$. This parameter controls how much the TDE spikes influence the CCO. When the gain is small, the lock-in range is narrow, as the TDE can not change the frequency of the CCO much. Thus, it only spikes with the same frequency of the input, when this frequency is close to the free-running frequency of the CCO. Increasing the gain has the opposite effect of making the lock-in range wider, as the CCO adjusts more to different frequencies. Besides adjusting the width of the lock-in range, the gain has another effect, it controls the amount of TDE spikes while being locked in. With small gains, the TDE has to spike more to adjust the CCO, as a single spike might not suffice to increase the frequency past the input frequency. Once again, the opposite is true with larger gains. Thus, there is a tradeoff between being sensitive to more frequencies and having a larger response of the TDE to the specific frequencies. This response is important, as we also use the TDE spikes as the output of the sPLL. An example of this tradeoff can be seen in Figure 8. On the left, the peak is wider, but not very discernible from the general trend of the TDE spike rate, while on the right the peak is very clear, but quite narrow.

The increased TDE spike count, as described above, is present in almost all tuned sPLL models produced by our hyperparameter optimization technique. Nonetheless, there is likely more going on, as the spikes produced by our encoders rarely output a constant frequency spike train, as demonstrated by the width of the peaks in Figure 3. Consequently, the sPLL can not really lock in to a spike train with variable frequency. Another way information is encoded by the sPLL is as a non-linear high pass filter. The TDE does not spike until a certain input frequency is reached, then the TDE spike count quickly rises and slowly levels off, see Figure 9a. However, as shown in the same figure, depending on the hyperparameters, there might be plateau regions, where the spike count does not increase for a large range of input frequencies, only to continue increasing for higher frequencies. These plateaus are unrelated to the saturating of the refractory period, which also result in plateaus and will be explained below.

We conjecture these plateaus are relevant, as they represent a nonlinear function applied to the frequency detected by an individual sPLL. One of the fundamental theorems of AI is the Universal Approximation Theorem, it states that a neural network with a single hidden layer can approximate any function, as long as enough neurons and a non-linear activation function is used [30, 31]. Interestingly, the frequency response in Figure 9b roughly resembles a sigmoid function, which was used extensively in early neural networks.



(a) An sPLL, tuned in an sPLL network, where only the offset current is varied, using the BioTac dataset encoded by the LIF encoder as input. The plot contains four different instances of this sPLL, each with a different offset current passed to the CCO, the TDE spike rate is plotted for each.



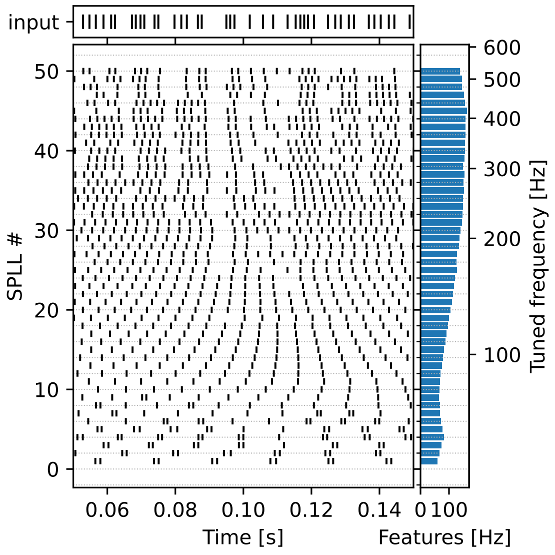
(b) A single sPLL, tuned in a time-scaled network of sPLLs using the BioTac dataset encoded by the TouchSim encoder.

Figure 9: Frequency response of different sPLLs. The sPLLs were optimized according to our method, to produce high accuracies when they are used in a network of sPLLs fed to a regression classifier. Thus, they represent our best set of parameters given the dataset and encoder. To compute the frequency response, they are run with constant frequency inputs, which is repeated for 2000 different input frequencies between 0 and 100 Hz. Finally, the spike rate of the neurons is plotted against the input frequency. Note, all frequency responses containing plateau regions, where the spike rate is the same for different input frequencies.

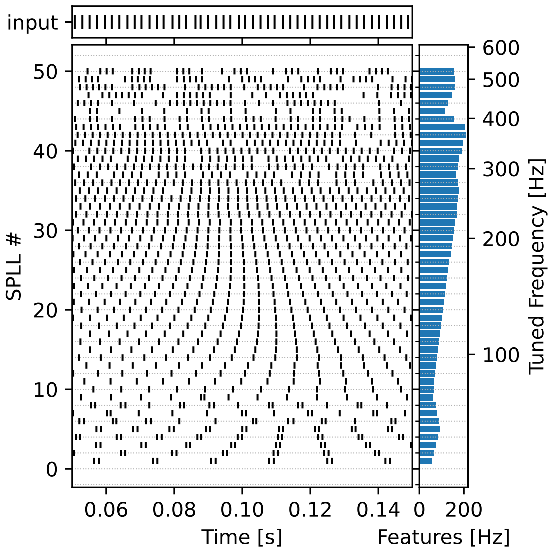
These plateaus form if the TDE time constants are configured in a specific way. $\tau_{\Delta}$ and $\tau_{\Delta,\text{trg}}$ are configured to be fairly small, so that the neuron voltage and trigger current return close to zero between each spike of the CCO. However, $\tau_{\Delta,\text{fac}}$ is significantly larger, thus the facilitatory current does not return to zero and oscillates around a value determined by the frequency of input spikes. Due to the exponential decay of the current value, the facilitatory current does not change much with changes in the input frequency. For example, take the sPLL in the right plot of Figure 9, at an input frequency of 100 Hz the facilitatory current has an average value of about 180$\mu$A, and at 200 Hz the current only increased to 320$\mu$A. Now, whenever the CCO spikes, the TDE will spike a constant amount and then return to it's resting state. When the input frequency is too low or too high, this behavior breaks down and the TDE spike count continues to increase with the input frequency. More specifically, when the input frequency is too low, the facilitatory current varies too much and the time difference between facilitatory and trigger spikes matters. While if the input frequency is too high, the TDE will not come back to it's resting state and once again, the difference between spike times starts to matter.

## 3.4 sPLL Network

In this section, we give examples of how the network of sPLLs operates, and present the final accuracies of our classifiers. The analysis, presented here, follows after all the steps in the top section of Figure 1. A linear ridge regression classifier is used, which predicts the type of texture, based on the spike counts of the TDE for each sPLL in the network, as described in Section 2.5 & 2.6. Next, we present example output spikes of the TDEs in Figure 10 and give notes for each of the presented plots. Finally, we present some distributions of the feature vectors produced by the sPLL networks, which will also be discussed.



(a) Input texture: sparkle paper 1

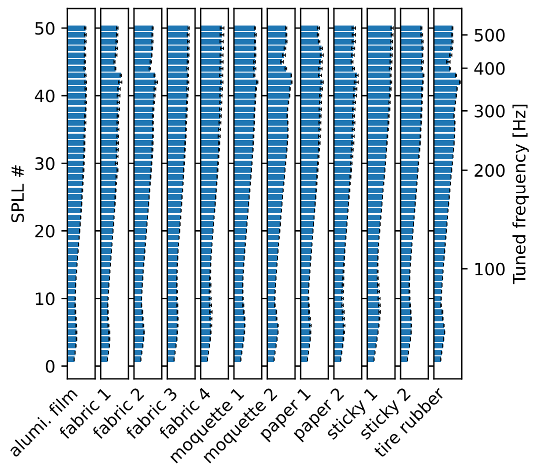


(b) Input texture: moquette 2

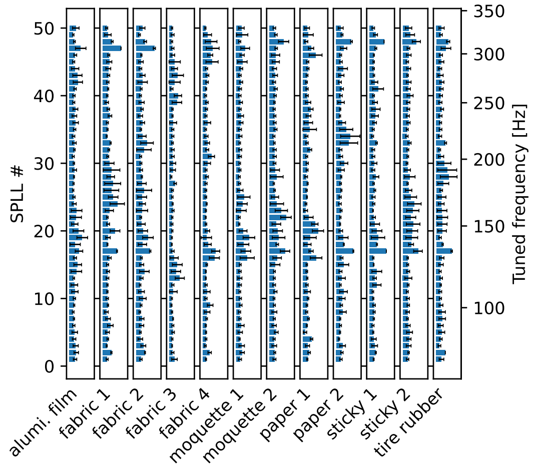
Figure 10: Example spike trains produced by each TDE in the best performing sPLL network. The sPLL network was tuned to generate feature vectors, which lead to high classification accuracy when they are used in a regression model. The network was optimized using the VibTac dataset encoded by the LIF encoder and each sPLL in the network is time-scaled, according to the tuned frequency displayed on the right of each plot. The average spike rate of each TDE is combined to create the feature vector used to classify the sample. These are displayed as horizontal bars to the right of the spikes.

First, Figure 10a shows how different sPLLs react to the input in distinct ways. The sPLLs numbered from 1 to 24 react fairly similar and do not vary much as the input varies, they are tuned from about 50 to 180Hz. However, above, the sPLLs output reflects changes in the input. At around 0.09 seconds into the simulation, there is a large gap between spikes in the input, this results in sPLLs numbered 25 to 50 each having a gap in their output. The same effect can be observed to a lesser extent with smaller gaps in the input spike train. When the gaps are smaller, only higher frequency sPLLs stop spiking. sPLLs, numbered 45 to 50, are tuned from about 400 to 500Hz, these generally only spike if the input spikes are in close succession, i.e. an inter spike interval of less than 2ms. Note, there are still exceptions, as sometimes, an input spike will randomly line up with a spike from the CCO and generate a spurious spike, but on average they only spike for high frequency inputs above 500 Hz.

Next, looking at Figure 10b, we can observe, what happens if the input spike train varies less in frequency: It becomes possible for a few sPLLs to lock on to the input. The sPLLs numbered 45 and 46 spikes significantly less than the sPLLs around them, which is visible in the output spikes, but also in the feature vector presented to the right of the spikes. Similarly, sPLLs numbered 42 & 43 spike more than the sPLLs surrounding it. This behavior is very similar to what we expect according to the frequency response presented in Figure 8, with a dip in spike count, and then an increase. Note, here the reduction in spike rate occurs for higher frequencies, then the increase, which is reversed from how it is in Figure 8. This is because for the frequency response we vary the frequency of the input, and in the sPLL network we vary the frequency the sPLLs are sensitive to. From the frequency response plots, Figure 8 in Section 3.3, we saw that the point where the sPLL spikes least is where the input frequency matches the tuned frequency of the sPLL. Thus, it is no surprise, that the tuned frequency of sPLL number 45 and a large spike in the input spike train frequency are both at 450Hz. For the input frequency distribution, see Figure 6. Next, we compare the feature vectors produced by the spike rates of the TDEs in the network.



(a) Encoder: LIF (best network)



(b) Encoder: frequency peak oscillator

Figure 11: Distribution of feature vectors produced by the time-scaled sPLL network tuned on VibTac data for different encoders. The networks are tuned to generate feature vectors which result in high accuracies when they are used to classify the textures using a regression model. The black lines represent the standard deviation of features (spike rate) between samples of the same texture. If they are not visible, the standard deviation is very close to zero.

In Figure 11 we present distributions of the feature vectors, each feature is just the average spike rate of each TDE in the network. The feature vectors of the best performing network are presented in Figure 11a. Overall, the vectors look fairly similar between classes. However, as is visible in Table 2, the differences are large enough to classify the classes with high accuracy. It is noteworthy, that the difference within classes are very small, with the error bars barely being visible. The other feature vectors presented here use the frequency peak oscillator encoder, see Figure 11b. The performance of this encoder is quite bad, but it does illustrate that if the input spikes to the sPLL network are very consistent, it is possible to detect individual frequency peaks. The problem with the encoder is that it dismisses most information and only transmits the frequency of three peaks in the frequency spectrum of the input data. Since these peaks can vary between samples, there are large differences within classes, which is visible by the large error bars in the plot. Nonetheless, there are individual peaks in the feature vectors where the frequency peak oscillator consistently spikes with the right frequency. To see the distribution of all other tuned sPLL networks, see Figure 16 in the Appendix.

Last, but not least, concider the performance of the sPLL for classifying textures. The test accuracies are presented in Table 2. Once again, the results are generally better for the VibTac dataset. Interestingly, the time-scaled sPLL only works slightly better than just varying the offset current between sPLLs, this is a promising result, as only varying the offset current simplifies the implementation in hardware. For the confusion matrices of the classifiers, see Figure 15 in the Appendix.

| Configuration | | Encoder | | |
|---|---|---|---|---|
| Dataset | Type | LIF | TouchSim | f. peak osc. |
| VibTac | time-scaled | 95.4% | 68.0% | 57.9% |
| VibTac | vary offset | 71.3% | 54.2% | 61.6% |
| BioTac | time-scaled | 47.4% | 50.7% | 37.3% |
| BioTac | vary offset | 37.1% | 53.4% | 37.2% |

Table 2: Test classification accuracies of the different sPLL networks and encoder pairs looked at in this study. The networks generate feature vectors, which are used to create a regression model. The 5-fold cross-validation test accuracies of these models are presented here.

The hyperparameters for the tuned models are presented in the Appendix, see Table 6 & 7. In the next section we will broadly asses some of the parameters along with their performance.

## 3.5 Tuned parameter analysis

To investigate the importance of the parameters in our model, an algorithm based on a functional ANOVA model was used [32]. The results of this analysis across all tuned models are presented in Figure 12. The computed importances are normalized to add up to 100%. In this case, the more important a parameter is, the more variance in the performance it explains, for the exact calculations, see Hutter et al. [32].

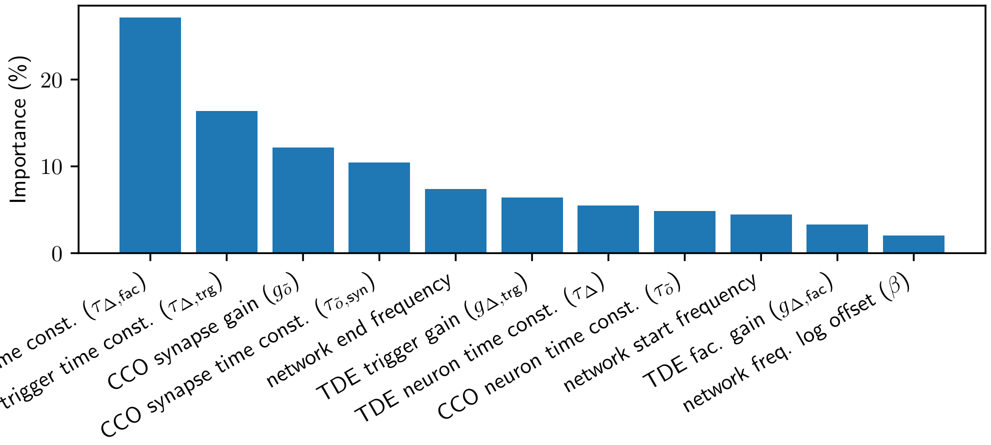


Figure 12: Computed importance of the different parameters tuned for the sPLL networks. The importances were calculated according to Hutter et al. [32] and are normalized to add up to 100%. Our networks were tuned to encode spike trains into feature vectors, such that these feature vectors result in the highest accuracy when they are used to train a regression classifier. The optimizations were performed using the Optuna framework [25], which uses tree-structured parzen estimators internally.

The most important parameter when tuning our models is the facilitatory synapse time constant of the TDE, followed by the time constant of the trigger synapse. These parameters determine the behavior of the sPLL in regard to input spikes, thus their high importance is justified. The third most important parameter is the CCO synapse gain, we have explained above, how this parameter is directly linked to the width of the lock-in range of the sPLL. The relatively high importance of this parameter is a confirmation of our postulation, that this parameter is relevant to the tuning of the sPLL.

Finally, the least important parameter is the network frequency log offset, this suggests, that the aliasing issue we foresaw, is less of an issue in practice. We predicted that due to the linear arrangement of the free-running frequencies of the sPLLs, sPLLs with integer multiple frequencies of one another would be sensitive to the same spike patterns. This was observed, when running the sPLLs with constant frequency inputs. As a mitigation, we parameterized the distribution of the sPLL frequencies, between exponential and linear, using the $\beta$ parameter, called the log offset. However, it appears that this effect is not relevant, when the model is run with real world data, as the log offset is not important to the network performance.

## 3.6 Summary of Results

In this section, the results presented above are summarized and put into context with one another. First, we present the accuracies of the best performing system, compared to the respective accuracies computed in the preliminary analyses. Then, we present average accuracies, computed for different configuration options of our final model, across all models we trained.

| Configuration | | | Analysis stage | | |
|---|---|---|---|---|---|
| Dataset | Encoder | sPLL type | Dataset | Encoder | sPLL |
| VibTac | LIF | time-scaled | 96.5% | 98.9% | 95.4% |
| VibTac | LIF | vary offset | 96.5% | 98.9% | 71.3% |
| BioTac | TouchSim | time-scaled | 84.4% | 53.9% | 50.7% |
| BioTac | TouchSim | vary offset | 84.4% | 53.9% | 53.4% |

Table 3: Accuracies of the classifier from the preliminary data analysis, over the encoder analysis, until the integrated system with the sPLL network. Presented for the best encoder, for each dataset and sPLL type. At each step, a regression model was created based on extracted features, and the 5-fold cross-validation accuracies are presented here. The feature vectors for the dataset are created using an FFT spectrum of the pressure signals, which is decimated to 50 values. The encoder feature vectors are created similarly, but the signal passed to the FFT is a discretized signal of the spike trains, see Section 2.3. Finally, the sPLL features are generated by taking the spike rate of each TDE neuron in the network. For an overview of the three steps, see Figure 1

In Table 3, we present the accuracies computed at different steps in our classification pipeline, this is done for the best encoder for both types of sPLL network and both datasets. For an overview of our pipeline and the preliminary analyses done, refer back to Figure 1. It is interesting to note, that accuracy drop between the encoder and the sPLL network is less than 4% in three out of four cases. This suggests, that the sPLL is generally capable of extracting a similar amount of information as a spiking Fourier transform. The BioTac dataset proved more challenging for classifying textures, but even here, the sPLL network performed very similar to the preliminary analysis, thus, most of the information present is extracted from the encoded signal. Next, the performance differences between different configuration options are compared.

| Configuration | | Accuracy average |
|---|---|---|
| Dataset | VibTac | 68.1% |
| | BioTac | 43.8% |
| Encoder | LIF | 62.8% |
| | TouchSim | 56.6% |
| | f. peak osc. | 48.5% |
| sPLL type | time-scaled | 59.4% |
| | vary offset | 52.5% |

Table 4: Average accuracies computed for different configuration options, the averages are calculated from Table 2, thus they only concern the accuracies of the fully integrated model with the sPLL network. This table serves as a comparison between the different configuration options.

Table 4 presents the averaged accuracies for different configuration options. The largest difference in performance is between the two datasets, with VibTac performing 24% better on average. We have outlined potential reasons for this in the data analysis section (3.1). It is clear that one of the most important steps when building a real world system is ensuring the sensors produce high fidelity data. While the LIF encoder worked great for the VibTac dataset, it did not perform great with the BioTac dataset, thus, the average accuracy is not much higher than the average accuracies of the other two encoders. Finally, the time-scaled sPLL network performs better than the sPLL where only the offset current is varied, this was expected, as the time-scaled sPLL can be tuned better to lock-in to many frequencies. It is encouraging to observe that the difference between the two is not large (7%), as this validates the sPLL network as it is implemented in hardware today.

## 4 Discussion

In this section, we consider the explanation underlying our findings and provide a more detailed context for our results. In addition, we talk about the assumptions mentioned in the methods section and explore how further research could build on this work. First, we take another look at our main findings and answer the research questions.

With the results, as they are presented above, we are confident in answering the research question posed in the introduction. The primary research question was: *Which neuromorphic method of converting pressure vibrations into spikes results in the highest material classification accuracy by a network of sPLLs?* Which encoder worked best depended on the dataset used. However, a simple LIF encoder provided the best result for the VibTac dataset, which has the clearest distinction between classes, and also provided the best average result. For the BioTac dataset, the TouchSim encoder worked best. Although, the classification accuracy of the whole system is significantly worse than when using the VibTac dataset with any of the three encoders. Leading to the conclusion, that besides the encoder used, the method of gathering the data is likely just as important, if not more. However, to prove this conclusively, we would need one dataset containing both methods, with one set of textures. I.e. the same material being sensed with a stylus and a robotically actuated finger.

As a secondary research question, we asked, *whether the single-channel non-spatial pressure data is enough to reliably classify textures using neuromorphic methods?* The answer to this question is less conclusive. We achieved an accuracy of 95.4% with our best classification architecture. However, this architecture used the time-scaled sPLL, which has not yet been implemented in neuromorphic hardware. Furthermore, we used the VibTac dataset, which was recorded in a lab setting, and it is not sure if the recordings could be reproduced reliably in the real world, the BioTac dataset is closer to a real-world scenario, but had worse performance. Hence, while we have shown that high classification accuracies are achievable. In practice, it might pose a challenge to achieve these high accuracies in a real-world system, due to the higher variability we observed in the BioTac dataset. We conjecture that using multiple channels can significantly improve the reliability, and suggest that this should be looked into in further research.

Next, we interpret the results as to understand why the networks performed as they did and how they might be improved. Starting with the feature networks produced by the sPLL network. As illustrated in Figure 11, there is an obvious contrast between the two subfigures. The figures show the mean spike rates produced by each TDE in the sPLL network for each texture class. Figure 11b exhibits clear distinctions between the various classes, while the different classes in Figure 11a appear quite similar. Despite the suboptimal performance of the frequency peak oscillator, the feature vectors generated by the respective networks have one quality the other feature vectors lack: There are large differences in the spike rate of neighboring sPLLs in the network, as depicted in Figure 11b. This indicates that the sPLLs properly lock in to the input signal, resulting in substantial disparities between classes, which are ideal for reliable classification of the feature vectors. As previously discussed, the substandard performance of the frequency peak oscillator encoder can be attributed to inherent limitations of the encoder itself. The encoder demonstrated an average accuracy of 48.5% when utilizing the sPLL network for feature extraction, while the sFFT analysis yielded an average accuracy of 48.4%. This suggests that the sPLL network operates at a level comparable to that of a Fourier transform in its capacity to differentiate between classes.

This behavior exemplifies a notable strength of the sPLL, namely its capacity to analyze overlaid constant frequency spike trains with high precision, as evidenced by the network's ability to accurately recover the input frequencies. This finding is in alignment with the observations reported by Mastella et al. [16]. However, it is noteworthy that the other encoders analyzed in this study do not produce spike trains of this nature. There are multiple frequencies present, but they might phase-shift throughout the sample, and are generally less consistent. We conjecture, that this reduces the ability of the sPLL network to discern these frequencies. Leading to feature vectors with striking similarity between classes, as in Figure 11a and even more obviously in some of the other configurations presented in Figure 16. Our findings indicate that the sPLLs are capable of generating feature vectors that exhibit sufficient variability to differentiate between classes. However, it may be advantageous to explore methods to overcome this limitation.

Two potential solutions are conceivable. The first solution would be to use an encoder that is more consistent, which conceptually would be situated between the LIF encoder and the frequency peak oscillator. However, even if an algorithm is found that satisfies these requirements, it will likely be impossible to implement it in neuromorphic hardware, due to the global nature of such a solution. The second solution involves the manner by which feature vectors are created. Instead of the average spike rate over the entire sample, a method could be conceived to build the features with a classification algorithm that also takes into account the difference in spike rates of the sPLL output neuron over time. The subsequent paragraph re-examines the fundamental assumptions underlying our model, providing additional rationale that further supports the efficacy of this approach.

As described in the methods section, two underlying assumptions of our classification architecture are the stationary and uniformity of the input signals. We have given reasons above why these assumptions should generally hold. However, when adapting our methods to the real world, it is likely that these assumptions will be broken. For example, when sensing for a sticker on a wall, not only are we classifying the texture of the sticker, but also where it begins and ends.

Thus, we explicitly want to detect the change in texture, rather than just the texture. With the method presented in this research, we would need to know when we are feeling the sticker, before we can run the analysis.

Given these reasons, consider replacing the linear regression classifier with a spiking classifier. To illustrate this proposition, consider the implementation of a spiking layer in conjunction with a winner-take-all network, a configuration capable of modulating its prediction in accordance with the variations in input data. In principle, this architectural design possesses the capacity to discern textures as they are being perceived, in addition to discerning changes between disparate textures. Furthermore, the spiking layer has the potential to take advantage of the spike timing of the TDE output, as opposed to solely the spike rate averaged over the entire sample duration. Furthermore, this enhancement is expected to result in more distinct feature vectors, as previously outlined.

The primary disadvantage associated with this system is that its calibration is predicted to be considerably harder, as the addition of layers with numerous parameters necessitate more calibration. Additionally, it has been observed that it can take up to 250 ms for an sPLL to lock on to a spiking signal. Consequently, the development of a system that incorporates a spiking layer and a winner-takes-all network, and that is sufficiently responsive to accurately detect changes in textures, is anticipated to be challenging.

Therefore, it is reasonable to conclude that before extending the network with a neuromorphic classifier and significantly increasing its complexity, the reliability of the sPLL network should be tested using real hardware. In the event that the performance proves to be inadequate, a subsequent course of action exists that entails the transition from single-channel data to multichannel data, a process that does not introduce a significant increase in complexity. The artificial finger utilized in the BioTac dataset has more lower-resolution channels available, which are collected from electrodes in the artificial skin. Each of these channels can be fed through its own encoder and sPLL network, with the features being constructed from a combination of all of them. This approach results in an increase in the number of neurons; however, the complexity does not increase significantly, as the neurons do not interact. This approach would also bring us closer to biology, as it has been found that vibration waves in the skin play a significant role in touch reception, and these waves are picked up by many afferents, as well as, the general interdependence of afferents [33–35].

Another reason for multichannel analysis is the within class variability of the BioTac dataset, as can be seen in Figure 3. With the different samples recorded having such different frequency spectrums, it is close to impossible to tell them apart reliably. We conjecture, that adding more channels might allow the classifier to rely on more than just the single spectrum, for example, differences between the channels. Or alternatively, the average of the different channels may have fewer differences between samples, than the single channel pressure signal. Next, some thoughts on the implementation in hardware are given.

To augment our data, a small amount of phase noise was introduced after the encoders. Besides, this phase noise, It would be intriguing to also model frequency noise, given that frequencies may vary over time in biological systems and neuromorphic hardware. For instance, an increase in circuit temperature could result in neurons firing more rapidly or more slowly. Despite the absence of model-based simulations of these shifts in frequency, observations revealed variability in spike frequency among the LIF and TouchSim encoders. Nonetheless, the current model demonstrates the capacity to effectively classify textures encoded by these encoders. Thus, we expect the model to be resilient to low amounts of frequency noise. Next, we propose a method of overcoming the limitation of implementing the time-scaled sPLL in hardware.

We found that the time-scaled network performed better than the one where only the CCO offset current is varied. Unfortunately, implementing this in hardware is challenging, as every sPLL circuit would need to be unique. A potential solution could be, multiple groups of sPLLs on the chip. Each group would repeat the same sPLL multiple times, but the different groups would have different time-scaled sPLLs. The exact free-running frequencies of the sPLLs could then be adjusted by varying the offset current, but the main drawback, of the sPLLs running far outside their ideal frequency range, would be addressed, by the different groups being scaled differently. How to best split up the sPLL blocks should be looked into in a further study.

## 5 Conclusion

In conclusion, the present study demonstrates the impact of neuromorphic methodologies for converting pressure vibrations into spikes on texture classification accuracy. The findings underscore the significance of selecting an appropriate encoder and sensor. The enhancement of sensors is poised to exert a substantial influence, as it leads to the refinement of features that can be extracted from the generated data. The performance of the LIF encoder was optimal with the VibTac dataset, yet its performance declined with the BioTac dataset, underscoring the significance of utilizing high-quality.

indicating that employing our methods facilitates the extraction of data recorded using a microphone, which has been demonstrated to exhibit a more diverse frequency spectrum, see Figure 3. The sPLL network, when operated in time-scaled mode, yielded superior outcomes in comparison to its offset current varied alternative, thereby substantiating its capacity to classify textures with greater precision.

Overall, this research provides a comprehensive response to the primary research question by demonstrating the optimal conditions under which neuromorphic methods are most effective. The VibTac dataset and the LIF encoder emerged as the most efficient combination, though the challenges encountered with the BioTac dataset underscore the complexity of implementing these methods in real-world scenarios. Addressing the secondary research question, the study demonstrates that single-channel pressure data can achieve high accuracy in texture classification, albeit exclusively for specific data collection methodologies. Further research is necessary to test these methods in real-world settings and to develop hardware for advanced architectures, such as the time-scaled sPLL. Future studies should prioritize the combination of improved datasets and encoders to enhance the reliability and practicality of these systems.

## References

[1] Z. Huang, Y. Hu, T. Du, S. Zhou, H. Su, J. B. Tenenbaum, and C. Gan, "Plasticinelab: A soft-body manipulation benchmark with differentiable physics," in *Proceedings of the International Conference on Learning Representations*, 2021, spotlight Presentation.

[2] E. Chicca, F. Stefanini, C. Bartolozzi, and G. Indiveri, "Neuromorphic electronic circuits for building autonomous cognitive systems," *Proceedings of the IEEE*, vol. 102, no. 9, pp. 1367–1388, 2014.

[3] H. Deng, G. Runger, E. Tuv, and M. Vladimir, "A time series forest for classification and feature extraction," *Information Sciences*, vol. 239, pp. 142–153, 2013.

[4] J. Lines, S. Taylor, and A. Bagnall, "HIVE-COTE: The hierarchical vote collective of transformation-based ensembles for time series classification," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, 2016, pp. 1041–1046.

[5] M. Middlehurst, J. Large, M. Flynn, J. Lines, A. Bostrom, and A. Bagnall, "HIVE-COTE 2.0: a new meta ensemble for time series classification," *Machine Learning*, vol. 110, no. 11, pp. 3211–3243, 2021.

[6] J. Faouzi, "Time Series Classification: A review of Algorithms and Implementations," in *Machine Learning (Emerging Trends and Applications)*, K. Kotecha, Ed. Proud Pen, 2022.

[7] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data mining and knowledge discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[8] G. Corniani and H. P. Saal, "Tactile innervation densities across the whole body," *Journal of Neurophysiology*, vol. 124, no. 4, pp. 1229–1240, 2020.

[9] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, "Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning," *Sustainable Computing: Informatics and Systems*, vol. 38, p. 100857, 2023.

[10] G. Indiveri and S.-C. Liu, "Memory and information processing in neuromorphic systems," *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1379–1397, 2015.

[11] E. L. Mackevicius, M. D. Best, H. P. Saal, and S. J. Bensmaia, "Millisecond precision spike timing shapes tactile perception," *Journal of Neuroscience*, vol. 32, no. 44, pp. 15 309–15 317, 2012.

[12] H. P. Saal and S. J. Bensmaia, "Touch is a team effort: interplay of submodalities in cutaneous sensibility," *Trends in Neurosciences*, vol. 37, no. 12, pp. 689–697, 2014.

[13] H. P. Saal, B. P. Delhaye, B. C. Rayhaun, and S. J. Bensmaia, "Simulating tactile signals from the whole hand with millisecond precision," *Proceedings of the National Academy of Sciences*, vol. 114, no. 28, pp. E5693–E5702, 2017.

[14] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[15] T. Timens, "Neuronal phase locked loops: models and circuits," 2023, unpublished bachelor thesis.

[16] M. Mastella, T. Tiemens, and E. Chicca, "Texture recognition using a biologically plausible spiking phase-locked loop model for spike train frequency decomposition," *arXiv preprint arXiv:2403.09723*, 2024.

[17] E. Ahissar, S. Haidarliu, and M. Zacksenhouse, "Decoding temporally encoded sensory input by cortical oscillations and thalamic phase comparators," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 94, p. 11633–11638, 1997.

[18] O. Kursun and A. Patooghy, "VibTac-12: Texture dataset collected by tactile sensors," 2020.

[19] R. Gao, T. Taunyazov, Z. Lin, and Y. Wu, "Supervised autoencoder joint learning on heterogeneous tactile sensory data: Improving material classification performance," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 10 907–10 913.

[20] J. Bell, S. Bolanowski, and M. H. Holmes, "The structure and function of pacinian corpuscles: A review," *Progress in Neurobiology*, vol. 42, no. 1, pp. 79–128, 1994.

[21] S. S. Kim, A. P. Sripati, and S. J. Bensmaia, "Predicting the timing of spikes evoked by tactile stimulation of the hand," *J Neurophysiol*, vol. 104, no. 3, pp. 1484–1496, Jul. 2010.

[22] M. Mastella and E. Chicca, "A hardware-friendly neuromorphic spiking neural network for frequency detection and fine texture decoding," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[23] M. Sato, "Response of pacinian corpuscles to sinusoidal vibration," *The Journal of Physiology*, vol. 159, pp. 391–409, 1961.

[24] R. P. Brent, "An algorithm with guaranteed convergence for finding a zero of a function," *Comput. J.*, vol. 14, pp. 422–425, 1971.

[25] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[26] A. Destexhe and M. Lilith, *Neuronal Noise*. Springer New York, 02 2012.

[27] S. Yu, Y. Wu, and H.-S. P. Wong, "Investigating the switching dynamics and multilevel capability of bipolar metal oxide resistive switching memory," *Applied Physics Letters*, vol. 98, pp. 103 514–103 514, 03 2011.

[28] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino, "Impact of memristor switching noise in a neuromorphic crossbar," in *2015 National Aerospace and Electronics Conference (NAECON)*, 2015, pp. 320–326.

[29] G. Ma, R. Yan, and H. Tang, "Exploiting noise as a resource for computation and learning in spiking neural networks," *Patterns*, vol. 4, no. 10, 2023.

[30] G. V. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, 1989.

[31] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[32] F. Hutter, H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," in *Proceedings of the 31st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, E. P. Xing and T. Jebara, Eds., vol. 32, no. 1. Bejing, China: PMLR, 22–24 Jun 2014, pp. 754–762.

[33] L. R. Manfredi, A. T. Baker, D. O. Elias, J. F. Dammann, M. C. Zielinski, V. S. Polashock, and S. J. Bensmaia, "The effect of surface wave propagation on neural responses to vibration in primate glabrous skin." *PLoS ONE*, vol. 7, no. 2, pp. 1 – 10, 2012.

[34] K. O. Johnson, "The roles and functions of cutaneous mechanoreceptors," *Current Opinion in Neurobiology*, vol. 11, no. 4, pp. 455–461, 2001.

[35] Q. Ouyang, J. Wu, Z. Shao, D. Chen, and J. W. Bisley, "A simplified model for simulating population responses of tactile afferents and receptors in the skin," *IEEE Transactions on Biomedical Engineering*, vol. 68, no. 2, pp. 556–567, 2021.

## A Supplementary Figures and Tables

This section contains extra figures omitted in the main body of work.

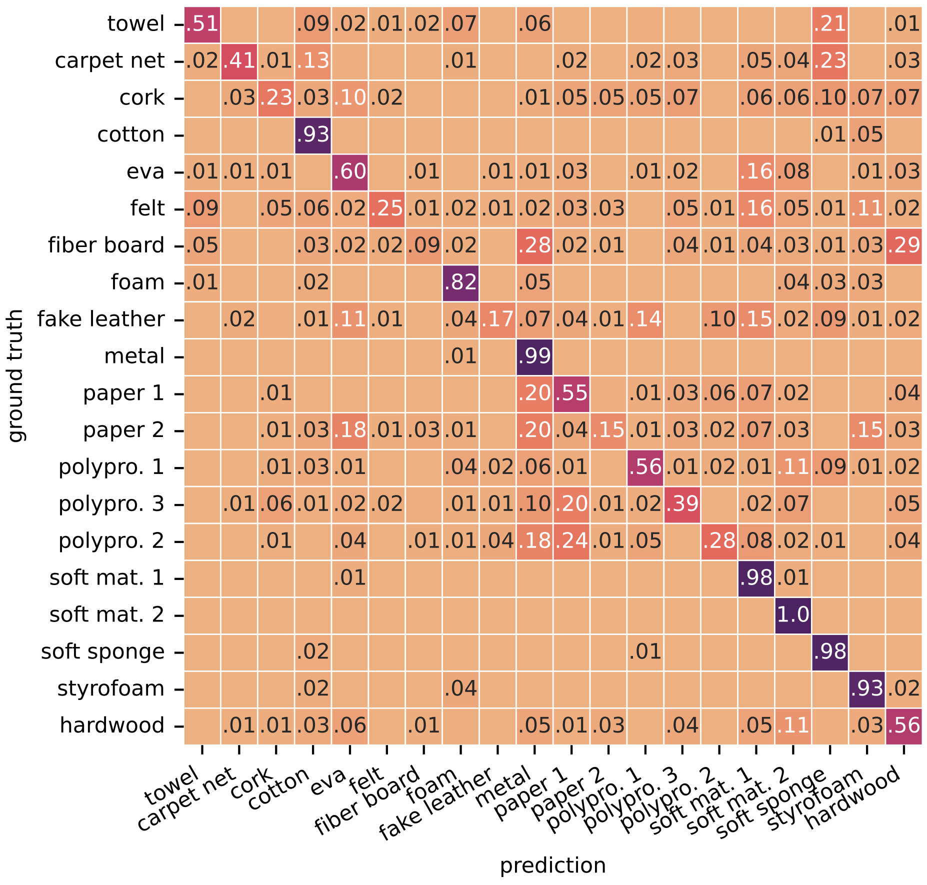
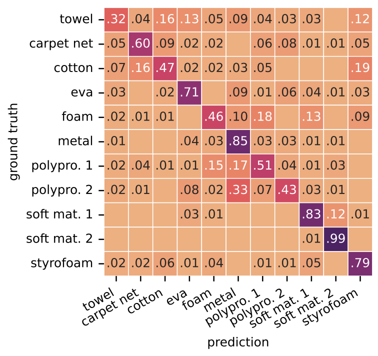
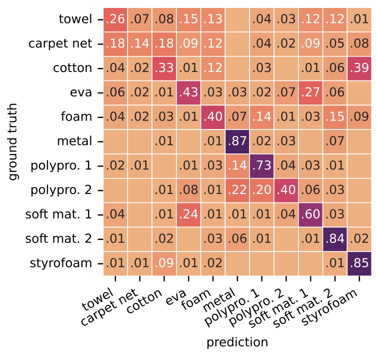


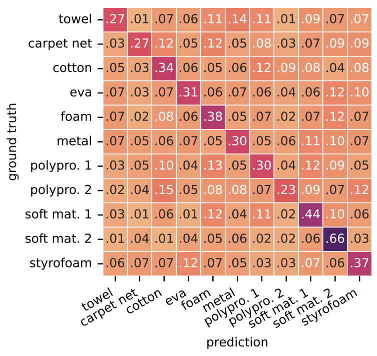
Figure 13: Confusion matrix of classifiers trained on features extracted straight from the BioTac data using a downsampled frequency spectrum. This figure contains the classes which were omitted for this research. Each sample in each dataset was converted to a feature vector by the following steps: compute FFT spectrum, take absolute value, downsampled to 50 values. These feature vectors were then used to train a regression model, which resulted in the presented confusion matrix.



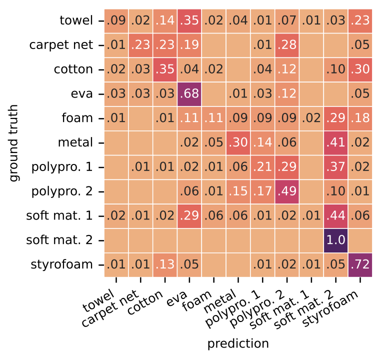
(a) BioTac, LIF, sFFT



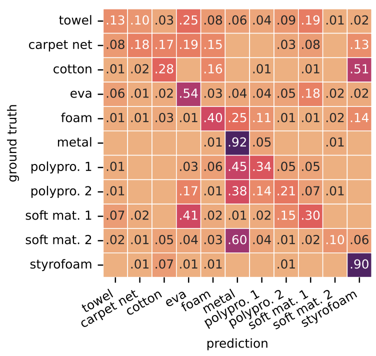
(b) BioTac, TouchSim, sFFT



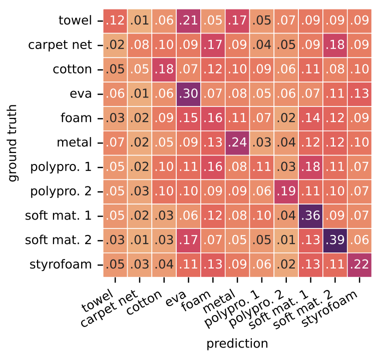
(c) BioTac, f. peak osc., sFFT



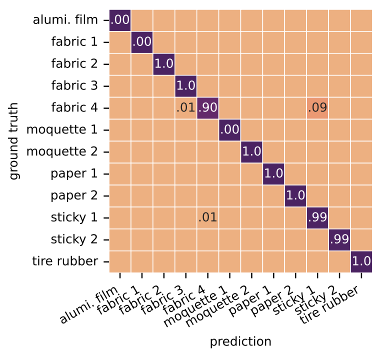
(d) BioTac, LIF, ISI hist.



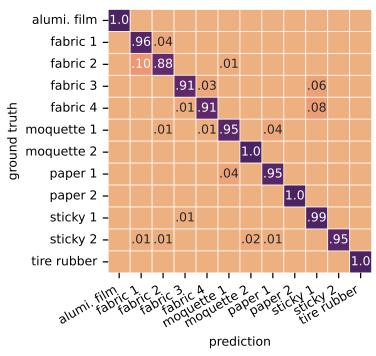
(e) BioTac, TouchSim, ISI hist.



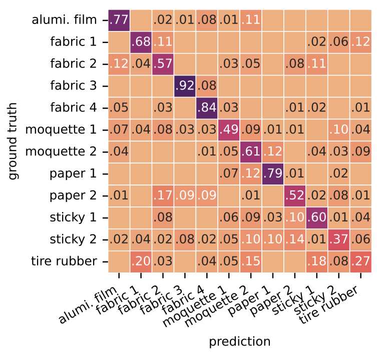
(f) BioTac, f. peak osc., ISI hist.



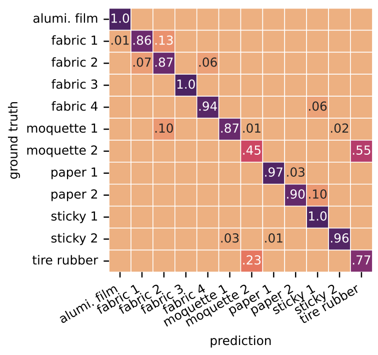
(g) VibTac, LIF, sFFT



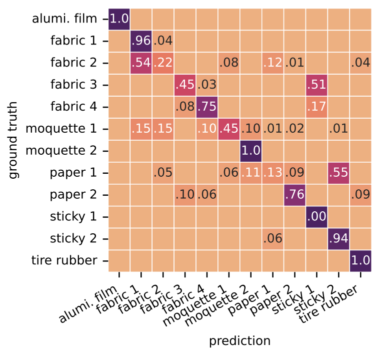
(h) VibTac, TouchSim, sFFT



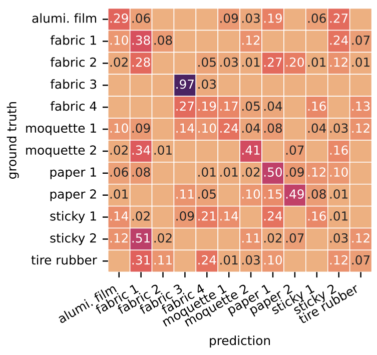
(i) VibTac, f. peak osc., sFFT



(j) VibTac, LIF, ISI hist.

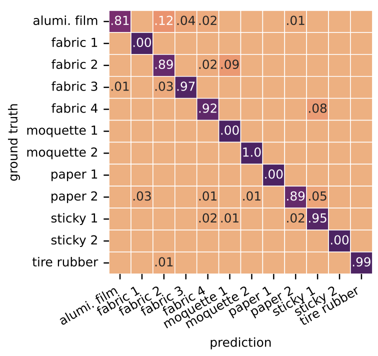


(k) VibTac, TouchSim, ISI hist.

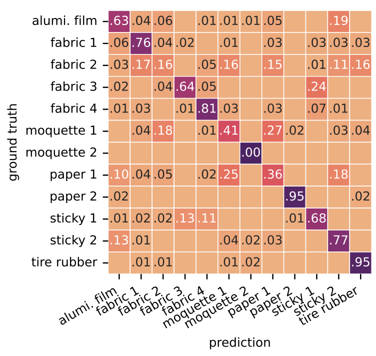


(l) VibTac, f. peak osc., ISI hist.

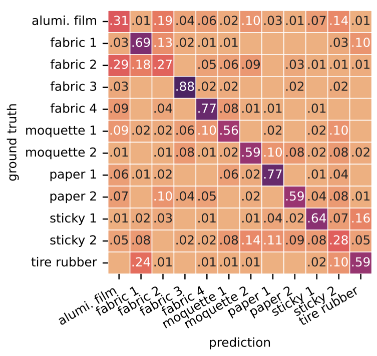
Figure 14: Confusion matrices for a regression classifier based on the data processed by each encoder and one of two feature extraction methods. The encoders turn the pressure signals into spikes, which are then turned into features using an sFFT approach or by computing a histogram over the inter-spike intervals. Feature vectors are always of length 50.



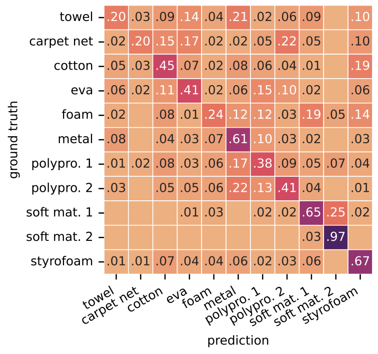
(a) time-scaled, VibTac, LIF, A: 95.4%



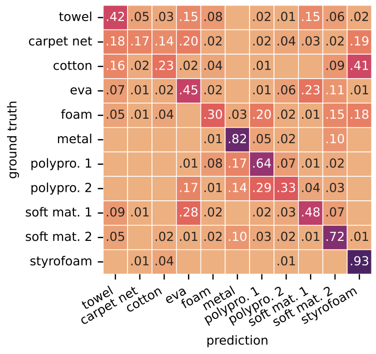
(b) time-scaled, VibTac, TouchSim, A: 68.0%



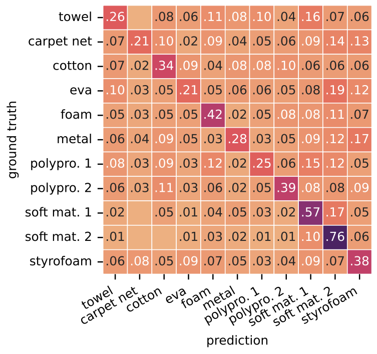
(c) time-scaled, VibTac, f. peak osc., A: 57.9%



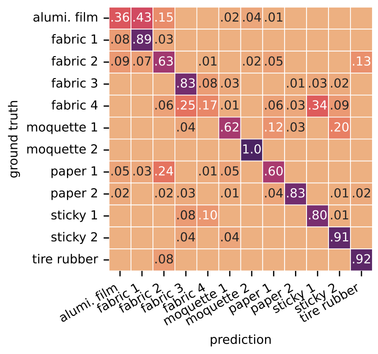
(d) time-scaled, BioTac, LIF, A: 47.4%



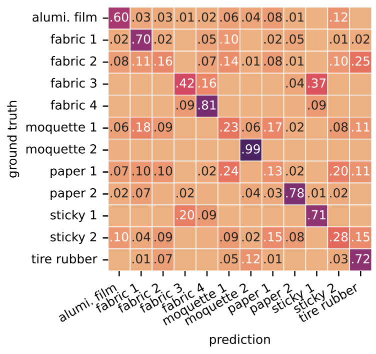
(e) time-scaled, BioTac, TouchSim, A: 50.7%



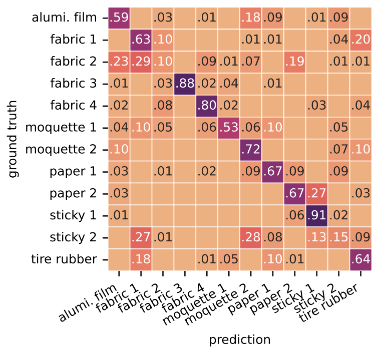
(f) time-scaled, BioTac, f. peak osc., A: 37.3%



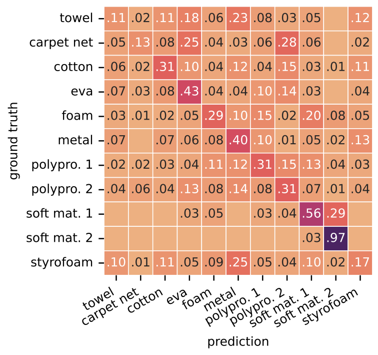
(g) vary offset, VibTac, LIF, A: 71.3%



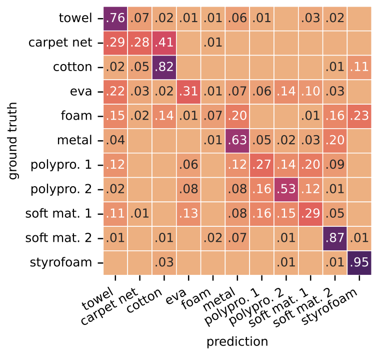
(h) vary offset, VibTac, TouchSim, A: 54.2%



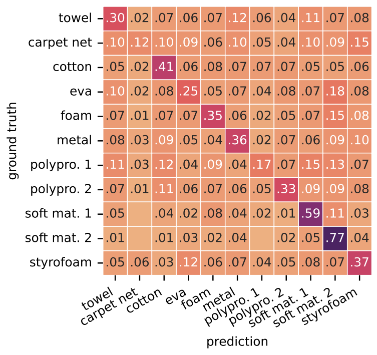
(i) vary offset, VibTac, f. peak osc., A: 61.6%



(j) vary offset, BioTac, LIF, A: 37.1%

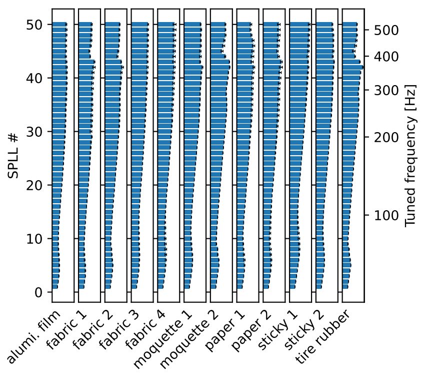


(k) vary offset, BioTac, TouchSim, A: 53.4%

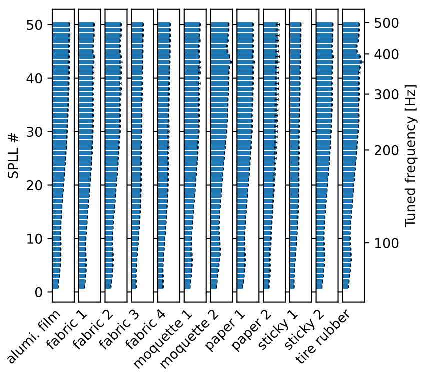


(l) vary offset, BioTac, f. peak osc., A: 37.2%

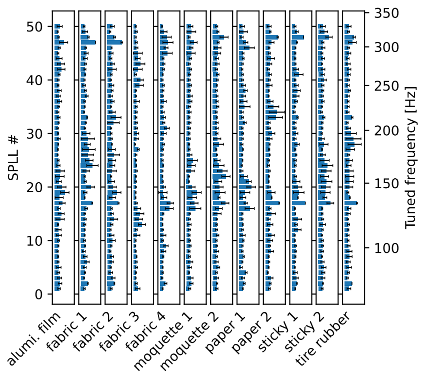
Figure 15: Confusion matrices for regression classifiers based on the data processed by each encoder and turned into feature vectors using sPLL networks. The networks were optimized to maximize the accuracy. Each subfigure represents one sPLL network. The accuracies are noted as "A: xx.x%" in the caption of each subfigure.



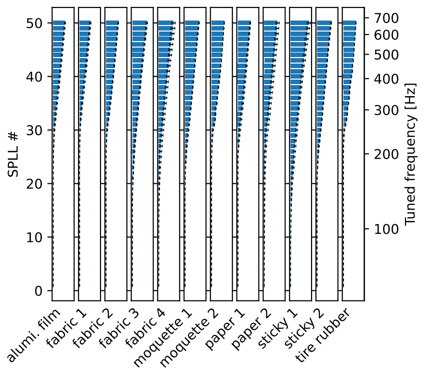
(a) VibTac, LIF, time-scaled



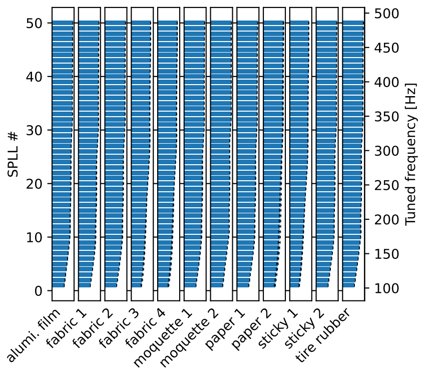
(b) VibTac, TouchSim, time-scaled



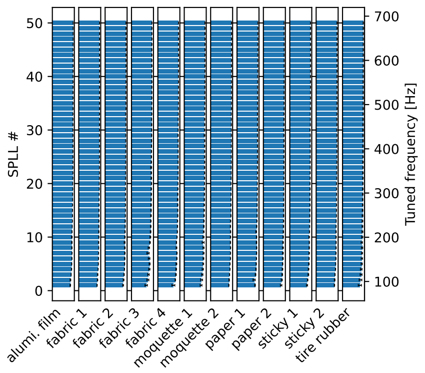
(c) VibTac, f. peak osc., time-scaled



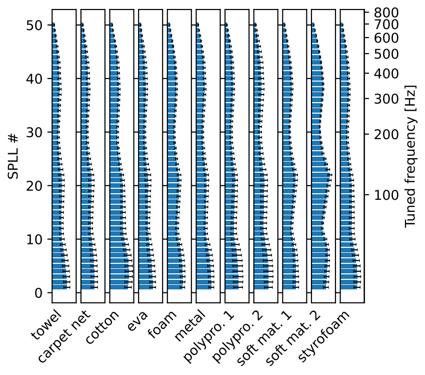
(d) VibTac, LIF, vary offset



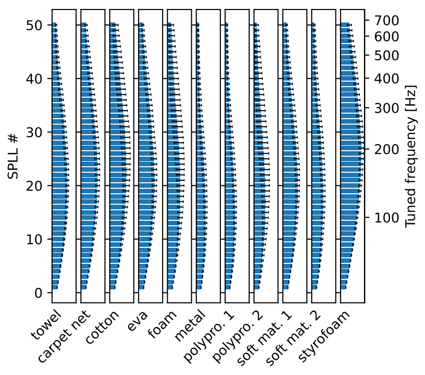
(e) VibTac, TouchSim, vary offset



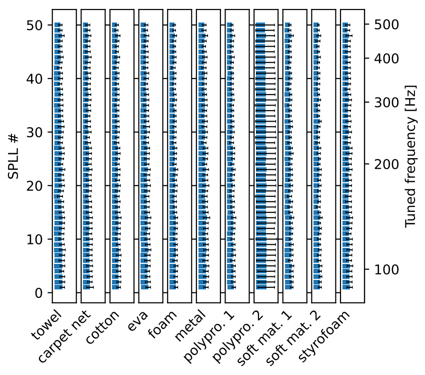
(f) VibTac, f. peak osc., vary offset



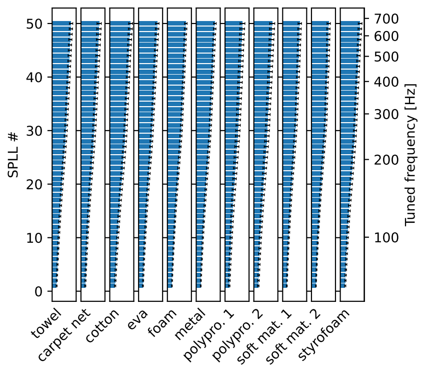
(g) BioTac, LIF, time-scaled



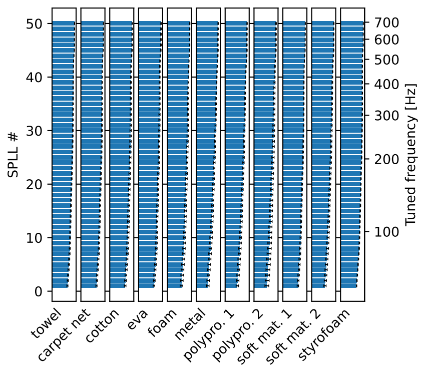
(h) BioTac, TouchSim, time-scaled



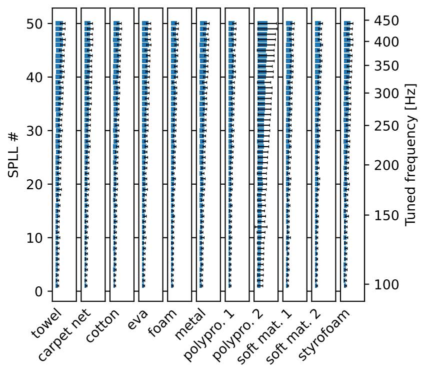
(i) BioTac, f. peak osc., time-scaled



(j) BioTac, LIF, vary offset



(k) BioTac, TouchSim, vary offset



(l) BioTac, f. peak osc., vary offset

Figure 16: Feature vector distribution of all tuned sPLL networks. Each feature in the feature vector is the spike rate of the TDE neuron of the respective sPLL in the network. The networks were tuned to achieve a high accuracy when a regression classifier classifies the texture, based on the provided feature vectors. On the axis on the right of each plot is the free-running frequency of the sPLL that produced the respective feature.

| Parameter name | Range start | Range end | Distribution |
|---|---|---|---|
| start freq | 30 | 100 | linear |
| end freq | 300 | 700 | linear |
| log offset | - | - | [1, 10, 100, ∞] |
| $\tau_{\tilde{o},\mathrm{syn}}$ | 0.005 | 0.2 | logarithmic |
| $\tau_{\tilde{o}}$ | 0.05 | 0.5 | logarithmic |
| $\tau_{\Delta,\mathrm{trg}}$ | 0.0005 | 0.02 | logarithmic |
| $\tau_{\Delta,\mathrm{fac}}$ | 0.0005 | 0.02 | logarithmic |
| $\tau_{\Delta}$ | 0.001 | 0.1 | logarithmic |
| $g_{\tilde{o}}$ | 0.1 | 10 | logarithmic |
| $g_{\Delta,\mathrm{trg}}$ | 0.1 | 5 | logarithmic |
| $g_{\Delta,\mathrm{fac}}$ | 500 | 100000 | logarithmic |

Table 5: Initial ranges of parameters for hyperparameter optimization.

| Configuration | | | Hyperparameters | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | Type | Encoder | $\tau_{\tilde{o},\mathrm{syn}}$ | $\tau_{\tilde{o}}$ | $\tau_{\Delta,\mathrm{trg}}$ | $\tau_{\Delta,\mathrm{fac}}$ | $\tau_{\Delta}$ | $g_{\tilde{o}}$ | $g_{\Delta,\mathrm{trg}}$ | $g_{\Delta,\mathrm{fac}}$ |
| VibTac | time-scaled | LIF | $1.38\times10^{-2}$ | $6.53\times10^{-2}$ | $5.70\times10^{-4}$ | $1.46\times10^{-2}$ | $2.29\times10^{-2}$ | 8.92 | 0.466 | $7.5\times10^{3}$ |
| | | TouchSim | $3.23\times10^{-2}$ | 0.287 | $8.61\times10^{-4}$ | $1.89\times10^{-2}$ | $4.73\times10^{-2}$ | 2.61 | 2.68 | $1.32\times10^{3}$ |
| | | f. peak osc. | $1.41\times10^{-2}$ | 0.11 | $5.57\times10^{-4}$ | $7.19\times10^{-4}$ | $2.47\times10^{-3}$ | 5.77 | 2.6 | $4.45\times10^{3}$ |
| BioTac | time-scaled | LIF | $5.39\times10^{-2}$ | $6.36\times10^{-2}$ | $1.15\times10^{-3}$ | $1.99\times10^{-2}$ | $2.09\times10^{-2}$ | 9.83 | 0.348 | $3.45\times10^{3}$ |
| | | TouchSim | 0.178 | 0.463 | $2.36\times10^{-3}$ | $7.47\times10^{-3}$ | $2.89\times10^{-3}$ | 7.28 | 0.227 | $3.78\times10^{4}$ |
| | | f. peak osc. | $5.48\times10^{-3}$ | 0.483 | $1.48\times10^{-2}$ | $3.76\times10^{-3}$ | $6.60\times10^{-2}$ | 0.108 | 0.201 | $1.98\times10^{3}$ |
| VibTac | vary offset | LIF | $7.50\times10^{-3}$ | 0.314 | $1.25\times10^{-3}$ | $9.59\times10^{-3}$ | $4.52\times10^{-3}$ | 9.84 | 0.271 | $6.15\times10^{2}$ |
| | | TouchSim | $1.37\times10^{-2}$ | 0.159 | $1.20\times10^{-3}$ | $1.06\times10^{-2}$ | $5.07\times10^{-2}$ | 8.04 | 3.57 | $1.83\times10^{3}$ |
| | | f. peak osc. | $7.69\times10^{-3}$ | $6.01\times10^{-2}$ | $1.59\times10^{-2}$ | $9.41\times10^{-4}$ | $9.99\times10^{-2}$ | 0.117 | 0.568 | $1.72\times10^{4}$ |
| BioTac | vary offset | LIF | $4.70\times10^{-2}$ | $7.68\times10^{-2}$ | $9.66\times10^{-4}$ | $1.25\times10^{-2}$ | $7.31\times10^{-3}$ | 3.96 | 2.42 | $1.23\times10^{3}$ |
| | | TouchSim | $6.49\times10^{-2}$ | $8.10\times10^{-2}$ | $6.68\times10^{-4}$ | $1.49\times10^{-2}$ | $2.37\times10^{-3}$ | 6.1 | 2.39 | $1.43\times10^{4}$ |
| | | f. peak osc. | $7.09\times10^{-3}$ | $5.63\times10^{-2}$ | $1.57\times10^{-3}$ | $5.32\times10^{-4}$ | $3.63\times10^{-2}$ | 0.1 | 1.7 | $4.3\times10^{3}$ |

Table 6: Best hyperparameters found by our hyperparameter search for each configuration of the sPLL network. These are the parameters which configure the base sPLL, which is then scaled based on the type of the sPLL network.

| Configuration | | | Hyperparameters | | |
|---|---|---|---|---|---|
| Dataset | Type | Encoder | start-freq | end-freq | log-offset |
| VibTac | time-scaled | LIF | 48.1 | $5.23\times10^{2}$ | 10 |
| | | TouchSim | 68.1 | $4.93\times10^{2}$ | 10 |
| | | f. peak osc. | 70.5 | $3.29\times10^{2}$ | 100 |
| BioTac | time-scaled | LIF | 32.2 | $6.92\times10^{2}$ | 1 |
| | | TouchSim | 44.0 | $6.68\times10^{2}$ | 10 |
| | | f. peak osc. | 85.7 | $4.97\times10^{2}$ | 1 |
| VibTac | vary offset | LIF | 56.2 | $6.68\times10^{2}$ | 1 |
| | | TouchSim | 96.2 | $4.86\times10^{2}$ | linear |
| | | f. peak osc. | 79.3 | $6.85\times10^{2}$ | linear |
| BioTac | vary offset | LIF | 61.6 | $6.69\times10^{2}$ | 1 |
| | | TouchSim | 54.6 | $6.92\times10^{2}$ | 10 |
| | | f. peak osc. | 95.9 | $4.42\times10^{2}$ | 10 |

Table 7: Best hyperparameters found by our hyperparameter search for each configuration of the sPLL network. These parameters configure the sPLL network, more specifically, the free-running frequencies of the CCO neuron for each sPLL.