University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

A deep learning approach to acoustic surface detection and odometry

Petr Sramek

March 27, 2019

Abstract

Precisely estimating speed and surface type using acoustic information could improve current interaction devices and introduce new techniques for acoustic interaction. The goal of this project is to explore whether a state-of-the-art end-to-end learning system is capable of predicting speed and surface type from acoustic signal; such a signal is produced while two objects are in contact and one is moving relative to the other. Accurately predicting speed and surface type with a single system that uses raw data to make these predictions has not been previously explored. It was found that estimating speed from acoustic signal is a rather straightforward problem and for slower speeds the system does not require large dataset to be accurate. However, detecting surface type in the same context proved to be a much more difficult problem as the system was unable to correctly classify surface types.

Education Use Consent

I hereby grant my permission for this project to be stored, distributed and shown to other University of Glasgow students and staff for educational purposes. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Signature: Petr Sramek Date: 27 March 2019

Contents

1 Introduction

1.1 Motivation

To set the scene for this dissertation, we begin with an example. Imagine a finger running over a textured surface such as the back of a book or a computer mouse moving across an office desk. This movement produces a sound of varying apparent pitch related to the finger or mouse velocity and the texture of the surface. In fact, reverberations from friction are produced every time an object is moved across a surface when in contact. If we could use this sound of movement to accurately estimate speed, it would provide us with a cheap way of improving current optical devices - such as gaming mice - which might under-perform on certain surfaces. Adding this new speed information might increase their precision and ensure robust performance across surfaces. Moreover, if we could reliably detect the surface on which the interaction is happening while, at the same time, estimate the speed of motion, it would give us an entirely new way of interacting with different systems using only acoustic information to force action.

Furthermore, we found that acoustic speed estimation is actively developed in industry. Hypersurfaces[1] is a company that uses vibration sensors to make ordinary surfaces interactive. Their system can be embedded into any surface and used to detect gestures, patterns, speed and position of interaction, thus giving any surface interactive properties.

1.2 General problem and our idea

Consider when two objects are in contact and one is being moved relative to the other. Contacts are made and broken which causes vibrations that depend upon the surface types of the objects and the speed of interaction. These vibrations can be captured in the form of an acoustic signal and analysed in order to determine certain physical properties of the interaction.

The general problem that we will attempt to solve is whether we can take this raw acoustic signal, process it, and transform it into precise predictions of the speed of the interaction that produced the signal. Moreover, whether using the same acoustic signal we can also detect the type of surface on which the interaction took place.

We propose a single end-to-end learning model which will take in raw acoustic data and which will learn how to precisely predict both speed of motion and surface type at the same time.

1.3 Aim

The aim of this project is to explore how well an end-to-end learning neural network can estimate speed of motion and detect surface type using only acoustic information. In order to achieve this, we will:

- describe the problems of acoustic surface type detection and acoustic speed estimation (Section 3) and show that these problems have not yet been approached using end-to-end learning (Section 2)
- create a dataset of continuous interactions consisting of recordings of acoustic signal paired with speed reference information for each interaction (Section 5)

[1] https://www.hypersurfaces.com/

- design and implement a state-of-the-art end-to-end learning neural network which will accept a block of acoustic signal and output corresponding block of speed estimations along with predictions about the type of surface on which the interaction is happening (Section 5)
- evaluate the performance of the implemented neural network using standard metrics for regression and classification tasks and explore how the performance changes for different hyperparameters (Section 6)

2 Background

2.1 Acoustic speed estimation

Automotive applications

One application of acoustic speed estimation is vehicle sensing. It has been shown that acoustic information can be used for estimating the speed of vehicles with either an external microphone (Cevher et al. 2009) or with an on-board microphone (Göksu 2018). Cevher et al. (2009) showed that one external omnidirectional microphone on the side of a road is not only able to predict the speed of a vehicle with a standard deviation of 0.8246 m/s from the ground truth, but is also able to classify the type of vehicle (e.g. a small personal vehicle or a truck). Moreover, using features of the signals obtained from Wavelet packet analysis Göksu (2018) achieved precision of up to 97.89% in speed estimation with an on-board microphone placed on the front passenger's seat.

While Göksu (2018) and Cevher et al. (2009) were concerned with specific problems to automotive applications, we find both papers relevant because of noise handling. There are many sources of noise in vehicles (e.g. the engine, the air-condition, the exhaust, etc.) and even though there will not be an engine noise present in the context of acoustic interaction, vibrations produced by laptop fans can have similar characteristic to the vibrations produced by a car engine. The fact that both approaches mentioned above were remarkably successful in very noisy environments makes it likely that, in a simpler context of acoustic interaction with considerably less noise, acoustic speed estimation will be possible as well.

Acoustic interaction

Other applications of acoustic speed estimation can be found in acoustic interaction where, in terms of gesture speed, Goel et al. (2014) can recognise two modes of speed, fast and slow, and while Rolshofen et al. (2005) shows potential to be able to estimate the speed of interaction given that it is able to track the movement, this angle is not explored. Harrison and Hudson (2008) allows for variable rate control by inferring velocity of movement from amplitude, however this feature was not evaluated in the experiment and similarly to Goel et al. (2014) only two modes of speeds are available. This limits capabilities of the systems to either binary option (fast or slow) or a number of discrete values for speed.

2.2 Acoustic surface type detection

2.2.1 Everyday surfaces

Detecting everyday surface (e.g. wall, table, etc.) type based on its acoustic properties can be used for object localisation as was demonstrated in Kunze and Lukowicz (2007) where vibrations and sounds were used to identify surface types to recognise where the object is located. Kunze and Lukowicz (2007) achieve high accuracy in recognising everyday surface types, but while the overall accuracy of the system is high, there are certain pairs of surfaces which the system is more likely to confuse, and the system has difficulties recognising two locations that are near each other. Although it was not the original goal of the research, Harrison et al. (2010) provides a supplemental experiment using the developed system to identify surfaces with which a hand

is interacting and demonstrates the possibility of this approach. However, the experiment was performed only on a small number of surfaces and more extensive evaluation is required to prove feasibility of the system for surface detection. Everyday surface type detection has been used in industry for Phillips[1] vacuum cleaners which can recognise between different types of hard, smooth floor and between different types of carpet (Schallig et al. 2000).

2.2.2 Manufactured surfaces

There has also been research done in detecting and recognising manufactured surface types (Harrison et al. 2012; Murray-Smith et al. 2008; Savage et al. 2015). Manufactured surfaces are fabricated surfaces that can be incorporated with existing surfaces or used on its own and possess specific acoustic or haptic properties. Manufactured surfaces can be designed to produce a very particular acoustic profile. Moreover, the recognition does not have to be dependent on the material from which the device is manufactured (Murray-Smith et al. 2008). The advantage of detecting everyday surface types compared to manufactured surfaces is the occurrence of these surfaces (e.g. wooden table, wall, etc.) in areas where we usually perform our daily activities. However, manufactured surfaces can allow for eyes-free interaction which may be harder to achieve with everyday surfaces. We could argue that using manufactured surfaces can reduce the classification task complexity as those surfaces have distinctive texture (and can therefore be more easily classified), however some commonly occurring surfaces such as carpet can be highly textured as well, and classifying them may not be as straightforward as one might think. Although Schallig et al. (2000) does not provide accuracy for the system, it mentions that previous attempts were only able to recognise two surfaces, carpet and hard floor, despite the fact that various types of carpets have very different and distinctive textures.

2.2.3 Object type detection

Acoustic profiles can be used for detecting which object is interacting with the system as well. Harrison et al. (2011) was extended with the ability to recognise six different objects for the purpose of multi-touch interaction with an interactive display. Acoustic object detection could be used to allow for different behaviour based on the input object, and different approaches have been used to address acoustic object recognition. Antonacci et al. (2007) exploits different acoustic profiles of objects to classify seven objects on two surfaces, and demonstrates that fingerprinting approach provides an accurate method for acoustic object detection. Antonacci et al. (2009) decided to use a set of features extracted from the sound of interaction as an input to a SVM classifier to categorise objects rather than to use fingerprinting approach, and in the worst case achieved 73% accuracy. Approaches described in both Antonacci et al. (2007) and Antonacci et al. (2009) have the advantage that the computational complexity is low and therefore can run on a small inexpensive processor.

2.3 Surface types of manufactured surfaces

Manufactured surfaces can be homogeneous in nature, which limits the number of possible interactions to the number of surfaces. While the surfaces used in Harrison et al. (2012) do not have to be 3D printed and can be incorporated into existing materials such as wood or acrylic, the system only allows for two intentions to be encoded as one can only swipe left to right and vice versa. Murray-Smith et al. (2008) uses acoustic profiles of homogeneous manufactured surfaces to detect with what surface the user is interacting. However, this limits the interaction to stroking and scratching. System described in Savage et al. (2015) uses acoustically unique tines with a slider to determine the position of the slider by mathematically predicting what the frequency profile of a single tine is, and then searching for it in the input signal. This allows

[1] Dutch electronics company

Savage et al. (2015) to increase the number of distinct interactive intentions and to encode the location of the slider.

2.4 Microphone types

With respect to acoustic interaction both generic microphones and contact microphones have been used in research projects.

2.4.1 Vibration sensor

One way to sense vibrations is to use generic microphones which are designed to capture mostly vibrations in the air, similarly as human ear does. They typically have a membrane (i.e. diaphragm) attached to a coil, or use capacitors to sense vibrations. Airborne vibrations then move the membrane and cause a change in the voltage. On the other hand, piezoelectric microphones such as contact microphones use the piezoelectric effect to capture reverberation of a surface (Bhattacharyya 2014). Contact microphones create change in voltage as a consequence of capturing waves propagated through a surface which means that they do not pick up any airborne vibrations.

2.4.2 Generic microphones

Generic microphones, either coupled with amplifier such as a stethoscope (Harrison et al. 2011; Harrison and Hudson 2008) or on their own (Goel et al. 2014; Robinson et al. 2011), provide simple way of picking up vibrations of a surface. These can be purchased at a very low price (from 50 pence per piece) offering a chance to cheaply capture broad range of frequencies. Furthermore, all mobile phones are already equipped with a generic microphone allowing for purely software solutions without the need to purchase any additional hardware. However, the main purpose of generic microphones, especially in mobile phones, is to capture speech. These signals along with other air-borne surrounding sounds represent noise when one is trying to capture vibrations of a surface.

Dealing with background noise

In order to increase the signal to noise ratio, Harrison and Hudson (2008) chose to use a generic microphone with a stethoscope and to consider frequencies strictly above 3kHz. This approach does filter out most of the environmental noise such as speech or mechanical vibrations and serves well for the purpose of gesture recognition with the given set of gestures, but frequencies below 3kHz can carry important information about the interaction. Harrison et al. (2011) used a generic microphone with a stethoscope as well, but rather than discarding lower frequencies to suppress background noise, Harrison et al. (2011) argues that the stethoscope on its own provides a sufficient shielding from environmental noise and that for louder noise an amplitude threshold is sufficient to distinguish between noise and impact. Harrison et al. (2011) then uses these lower frequencies to categorise impacts based on by which part of a finger they were performed. This could not be done if frequencies below 3kHz were filtered out as Harrison et al. (2011) identified that the key information is in frequencies between 0Hz and 1000Hz. Although using solely a mobile phone microphone without a stethoscope, Robinson et al. (2011) uses similar approach to Harrison and Hudson (2008) where frequencies below 3kHz are discarded in order to eliminate background noise.

2.4.3 Contact microphones

Rather than removing information from the signal by using arbitrary thresholds, one can use a piezo contact microphone instead. Contact microphones have been used in many projects (Amento et al. 2002; Murray-Smith et al. 2008; Savage et al. 2015; Ishii et al. 1999; Rolshofen et al. 2005; Harrison et al. 2012; Paradiso and Leo 2005; Harrison et al. 2010) and unlike a generic microphone it does not suffer from environmental noise because it is insensitive to sounds in the air.

Frequency response

While contact microphones pickup signals conducted through materials well, they are not particularly sensitive to very low frequencies; depending on the exact contact microphone its lower limiting frequency can be from around 50Hz and lower (Paradiso and Leo 2005). One possible way, presented in Paradiso and Leo (2005),to pick up even these low frequencies is to use transducer which is configured specifically for these frequencies. While Paradiso and Leo (2005) admits that the information from contact microphone can be sufficient for capturing low frequency signals, choosing a hardware solution over a software one seemed like an easier and more robust option for their project.

2.5 One microphone vs multiple

2.5.1 Single microphone

Using only single microphone, whether it is a generic or a contact one, comes inevitably with a major drawback: one microphone cannot be used to precisely infer the position of an object. Saxena and Ng (2009) attempted to create a monaural system capable of distinguishing between different angles from which the sound was coming, but the average error was 13.5 degrees and the system was unable to infer distance to the source. However, this disadvantage does not mean that one microphone is useless when it comes to interaction. A single microphone, among other things, is easier to install and setup compared to having to place number of microphones on a surface, and was used in multiple projects (Amento et al. 2002; Murray-Smith et al. 2008; Harrison et al. 2012; Harrison and Hudson 2008; Robinson et al. 2011). Recognising even simple gestures can be an unobtrusive way of interacting with a device as was shown in Robinson et al. (2011), where taps on the back of the phone could control the playback speed of a call without the need to look at the screen. Similarly, Murray-Smith et al. (2008) allowed for more complex eyes-free interaction with a media player by using a special 3D printed object with differently textured surfaces. In addition, Harrison and Hudson (2008) was able to achieve high accuracy of increasingly complex gestures on large passive surface with just one microphone. While these approaches allow for interaction they can at most recognise on what surface the interaction is happening or what gesture is being performed, but not the exact position.

2.5.2 Microphone arrays

On the other hand, the benefit of using multiple microphones is that it allows tracking the movement of an object on a surface. This is especially useful when the interaction includes a visual display like in Rolshofen et al. (2005), where continuous movements can be tracked and displayed to the user or like in Ishii et al. (1999) where the impact of a ping pong ball is visually accommodated by projecting animations onto a ping pong table. Even when a visual display is not present or is not important for the interaction, the knowledge of direction of movement and/or position of an object/impact can be essential for the desired interaction (Goel et al. 2014;

Harrison et al. 2010). Using an array of contact microphones Harrison et al. (2010) estimated position of interaction on human skin on pre-learned locations.

2.6 Optical flow sensor

In order to perform any sort of calibration or evaluation we need a reference of the true speed of motion. Since we are focusing on interaction, we will be estimating speed at a close distance. There are two most notable approaches for measuring speed in close proximity to a surface. One is using a mechanical sensor - such as the one found in mechanical ball mice - and the other is using an optical flow sensor, which can be found in almost every optical mouse. We will focus on the optical flow sensor due to the lower precision of the mechanical sensor, noise it can produce, and unreliability when the mechanisms becomes clogged with lint.

Optical flow sensors work by capturing images at time t and time $t+1$ and then calculating the displacement or shift between the two images - the previous and current position of the sensor.

2.7 End-to-end learning

The traditional approach to signal modelling would be to process the raw audio signal before feeding it to the neural network by obtaining a feature vector using feature extraction techniques such as Mel-Frequency Cepstral Coefficients (MFCC), Fourier transformation or Wavelet transformation. This reduces the size of the input and consequently the computational complexity. On the other hand, in end-to-end learning the neural network is given the raw input, be it pixels or signal samples, and the features necessary to produce certain output are learned (acquired) from the input itself as part of the learning process, which reduces the possibility of providing unsuitable feature vectors. In addition, we do not have to make choices which might limit what the system can learn. For example, suppose we choose to represent the signal as MFCC then the system can only use what the MFCC can capture. However, if we provide the raw signal then the system can theoretically learn any information in the signal, assuming it is sufficiently complex to be able to learn it (Glasmachers 2017). Moreover, end-to-end learning offers greater flexibility in terms of output complexity as it is easier to output (produce) more complex data types. However, at the same time end-to-end learning increases the computational cost because raw input is usually many times larger than feature vector. In addition, it usually requires more labelled data because it has to learn the mapping from one type of raw data to a different type of raw data. The latter problem of dataset size requirements can be partially addressed by dataset augmentation techniques and cross-fold validation.

2.8 Neural network architecture

WaveNet is a state-of-the-art autoregressive feedforward convolutional neural network architecture for sequence modelling widely used by Google[2]. While WaveNet was designed for end-to-end learning it could be appropriated to use MFCC or similar feature vectors. Dilated causal convolutions are what discriminates WaveNet architecture from other autoregressive convolutional neural networks and these are described in Subsection 2.8.1 and 2.8.2.

While recurrent neural networks used to be synonymous with sequence modelling tasks, a recent research by Bai et al. (2018) has shown that neural networks with dilated causal convolutions outperform recurrent neural networks in sequence modelling tasks on standard datasets. Examples of such tasks where convolutional neural networks outperform typical recurrent approaches are character-level language modelling (Kalchbrenner et al. 2016), machine

[2]US-based technological company

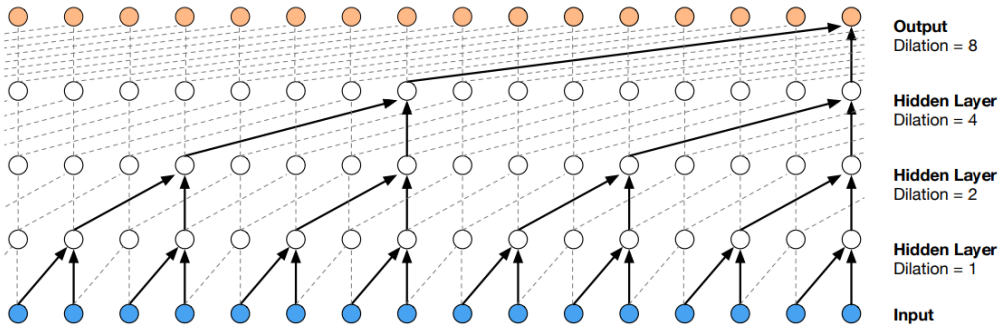


Figure 2.1: *Figure from Oord et al. (2016a) showing dilated causal convolutions. Dilation here is by a factor of 2.*

translation (Gehring et al. 2016), and sequence to sequence learning (Gehring et al. 2017). In addition, the temporal neural network that was implemented in Bai et al. (2018) was a greatly simplified version of WaveNet architecture, which hints that even better performance could be achieved with more complete implementation. Moreover, given that feedforward networks do not have any cycles they are usually faster to train than recurrent neural networks and can allow for longer context windows (Oord et al. 2016a).

2.8.1 Causal Convolutions

Described in Oord et al. (2016a), causal convolutions can be used to ensure that given an input $X_1..X_t,X_{t+1},..X_T$ a model can predict output y at timestep t using only input values X_1 to X_t but not input X_{t+1} or any values following it. This is sensible to do as predictions at certain timestep t do not depend on any future values in signal modelling. A similar idea to causal convolutions was explored in Oord et al. (2016b) where masked convolutions were used in the context of image generation.

2.8.2 Dilated Convolutions

However, causal convolutions can become computationally expensive with larger receptive field. In order to address this issue and thus help with scalability, the idea of dilated convolutions was introduced into the model. In dilated convolution, each next layer uses only part of the input from the previous layer as shown in Figure 2.1. This reduces the computational cost significantly.

2.9 Summary

We have discussed techniques from literature and reported their accuracy for acoustic interaction such as gesture recognition, surface type detection or speed estimation involving one or more generic or contact microphone(s) on everyday or manufactured surfaces. We have also introduced a state-of-the-art WaveNet architecture for neural networks and explored what the benefits and drawbacks of end-to-end learning are. After this review, we noticed that there is a niche to be filled, which is using end-to-end learning to precisely estimate speed using acoustic information from the interaction of an object moving across a surface with an optical reference to provide sufficient training data.

3 Analysis

3.1 General problem

3.1.1 Acoustic speed estimation

To define the problem of acoustic speed estimation, we consider an object moving across an arbitrary surface. There are small irregularities on any surface and if we increase the number of contact events between those irregularities – or in other terms increase the speed of movement across this surface – the frequency increases as well. We can see this effect in Figure 3.1 where there is a frequency shift at the time of the movement. This tells us that there is a correlation between the speed of the movement and the frequency the interaction produces. However, different surfaces have different types of irregularities, for example glass will be less textured compared to a different material such as wood. These differences in texture suggest there will be variation in the frequency shift across different materials due to the different number of contact events occurring per unit of time. Therefore, more precise estimates would be possible if the system could recognise the surface with which it is interacting.

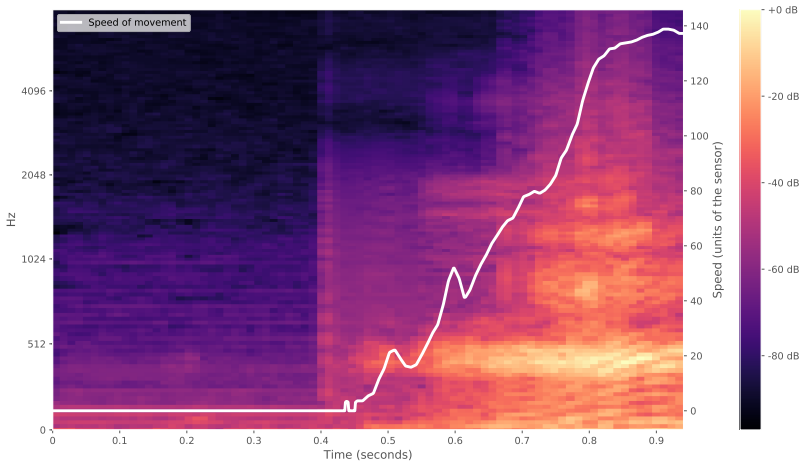


Figure 3.1: *Mel-scaled spectrogram overlaid with speed signal showing the frequency shift with increasing speed.*

3.1.2 Surface type detection

As we discussed in Section 2.2, using the differences in irregularities of surfaces was successfully used to distinguish between different types of manufactured surfaces. Moreover, by using acoustic patterns and properties of different surfaces a system was developed that can recognise between everyday surfaces on which the interaction is happening. This suggest that acoustic surface type detection is a problem that is not constrained by the approach used and high accuracy can be achieved.

3.2 Microphone

3.2.1 Single microphone

Using multiple microphones (i.e. a microphone array) would allow us to use techniques such as beamforming or Time Delay of Arrival (Rolshofen et al. 2005), however using more than one microphone complicates the setup as multiple microphones have to be placed onto a surface. In addition, we decided that the microphone will be physically attached to the object that is performing the interaction. Therefore, multiple microphones on the object would not provide us with much more useful information.

3.2.2 Contact microphone

One can argue that – since moving across a surface while in contact (e.g. scratching and similar moves such as swiping) produce a lot of high frequency components – a generic microphone could perform better in the context of speed estimation, however given the premise established in Section 2.4 that lower frequencies carry more discriminative information about an interaction, contact microphones pose as a better candidate given that they can pick up lower range of frequencies without suffering much from environmental noise. The relevance of using a contact microphone for continuous movements has been shown in Amento et al. (2002) where a contact microphone on a wrist was used to classify gestures such as tapping or flicking and although the paper does not mention any formal evaluation or precise numbers, the authors were optimistic about the results from the prototype.

3.2.3 Position of microphone

There are two general positions the microphone could be placed in – either on the surface of interaction or on the object with which the interaction is performed, be it a finger or some object such as a pen or a pointer. We chose to place the microphone on the interacting object as that makes the approach more transferable between different surfaces and usable under various circumstances. Fixing one out of the two interacting surfaces means that we are constraining the problem to one particular surface interacting with many others rather than having many surfaces interacting with many other surfaces, which would require a much more extensive study. In order to reduce the number of sources of noise, we decided to place the microphone on an object rather than a human body due to the noise a human body can introduce to a system. By doing so, we simultaneously heightened the signal because a homogeneous rigid object has better acoustic conduction properties than a heterogeneous human finger. It is important to note that there are different positions on the object itself. The microphone can be placed further from the interaction contact, but that would significantly both decrease the strength of the signal and increase the time of travel for the signal as it would have to be conducted throughout the whole object. Therefore, a position closer to the interaction is more suitable.

3.3 Processing

3.3.1 Sequence modelling

The problem mentioned in previous section can be reduced down to using a sequence of samples of an audio signal to model a sequence of samples of speed. In Section 2 we have discussed approaches used in the literature where other researchers used feature vectors to represent the audio signal and then used those to infer the type of interaction or surface. In Figure 3.1 we can see that doing the acoustic speed estimation by explicitly providing a set of feature values would be possible as there is a clear frequency shift in the signal; however rather then providing

a feature representation of the input signal, we want to learn a mapping from raw samples of acoustic signal to raw speed estimates using end-to-end learning.

End-to-end learning

After reviewing the literature in Section 2, there seems to be no research attempting to use end-to-end learning for speed estimation using acoustic signal. While we realise the drawbacks of end-to-end learning discussed in Section 2.7, we wanted to examine this option because it can improve the performance of a system on sequence modelling tasks. End-to-end learning has been very successfully used many times for speech recognition, which bears similar characteristics to our, much simpler problem. Therefore, we have a reason to expect that this approach will be successful.

More complex output data types such as simultaneous regression and classification is another benefit of end-to-end learning which would be harder to achieve using other machine learning methods. In addition, end-to-end learning can be very flexible in terms of timing which means that a different sample rate can be used for input and output.

Ground truth

In order to use any machine learning technique, we will need a set of labelled data which will be used for training. This labelled dataset will require to have some reference values for speed and surface type and while information about the surface can be added manually, it would be extremely time intensive and imprecise to estimate the speed of movement in the same way. Therefore, a velocity estimating device will be used to provide this kind of information. Mechanical speed sensors could introduce unnecessary noise to the system but there are optical sensing devices that compute the optical flow and can be very accurate in certain speed ranges. It is possible to use such an optical flow device because the interaction we are examining is happening in close contact with the surface and thus it provides a robust ground truth. Moreover, optical flow technology is widely available in optical mice and the current sensing technology can provide very accurate measurements.

3.4 Summary

In this Section we have introduced the general problem of acoustic speed estimation and acoustic surface type detection, we have discussed the reasons for choosing to use one contact microphone placed on an interacting object which provides input for end-to-end learning to predict the surface type and speed.

4 Design

4.1 Hardware design setup

There will be one specific object equipped with a contact microphone and an optical flow sensor, both physically mounted on the said object. This arrangement will be used to create a dataset for the end-to-end learning algorithm by moving the object across a range of surfaces while recording the acoustic signal and reference speed using the contact microphone and the optical flow sensor, respectively.

4.1.1 Hardware - optical flow sensor

The optical sensor will not be providing the speed information directly - instead it will output the displacements in the x and y axis which can later be converted into speed using a standard formula for calculating vector magnitude, defined as follows:

$$speed(t) = \sqrt{(dx(t)^2 + dy(t)^2)}, \tag{4.1}$$

where $dx(t)$ and $dy(t)$ are integer values of displacement at time t for x and y axis respectively and the resulting *speed* will be given in default units of the sensor. We will not be storing the speed information calculated using Equation 4.1 but rather want to get as low level access as possible to the least distorted values of displacement which will form our data store. These displacements will be in a form of a sequence of integer values given at a particular rate and in the units of the sensor.

In order to achieve the best resolution and most precise values the optical flow sensor must be in close proximity to the surface (approximately 1 to 5 mm from the surface). A range of standard optical devices exists and some of these will be investigated later in the project, examples include optical mice or small optical cameras.

4.1.2 Hardware - contact microphone

For the contact microphone to work properly it needs to be in contact with a surface from which we want to capture the vibrations. The contact can be direct or through a medium which needs to have a good acoustic propagation properties in order to conduct vibrations reliably (i.e. in a way that some frequencies will not be lost or amplified and frequencies that were not present in the source signal will not be introduced). Moreover, the way we mount the microphone needs to be robust to avoid any secondary vibrations which would introduce unwanted noise into the system. After the microphone is mounted correctly we want to capture a single channel of audio signal, but to do that we might need to use a special amplifier to interface with the contact microphone. This can make the sensor more sensitive to certain frequencies which it would not normally be able to record.

4.2 System design overview

4.2.1 Description of the system

The system will take sequences of audio signal samples and produce estimates of speed and surface type. These estimates will be computed by the neural network, which learned what representation of the input signal yields the most optimal results during previous training. In Figure 4.1 we can see that during training, outputs of the neural network are compared with the ground truth. To compare the output with the ground truth we calculate the error between two identically formatted blocks of speed values and two predictions of the surface type. This error is then passed back to the machine learning algorithm to adjust the values of the weights accordingly in order to make more precise predictions.

It is important to note that the whole system will not be using batches of data but rather it will be stream-based. Therefore blocks of audio samples can be passed to the neural network continuously and it will produce the estimates of speed and surface type at the corresponding rate in an online fashion.

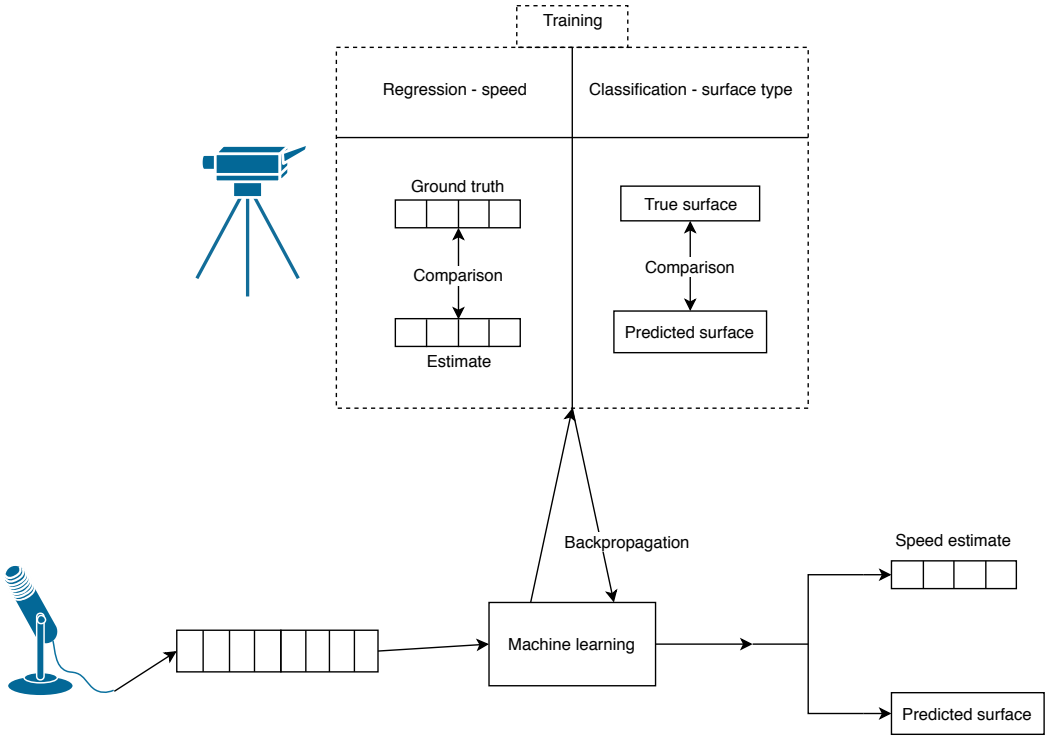


Figure 4.1: *Flow chart of the system design. Starting with the audio signal captured by the contact microphone, there are two modes of operation. Training, in the upper part of the figure, will be used to initialise the weights in the neural network. During the training output from the network will be compared against the ground truth obtained by the optical flow sensor (speed - left) and hand crafted (surface type - right). When the system is not in training mode of operation it will produce the speed and surface type estimate given a block of acoustic signal.*

Latency

When creating a stream-based system we need to take into account the round trip latency which will be present. The delay between the time of contact of the object with the surface and the estimates will depend on a number of factors, such as the hardware the system is running

on or the size of the windows passed to the machine learning algorithm. The larger the size of the windows of the audio signal, the longer the machine learning will have to wait before producing the estimates. Moreover, there is a linear relationship between the complexity of the neural network and the computational cost because more calculations will result in longer runtime.

4.2.2 Speed estimation - regression

Speed ranges to look at

In order to define what speed ranges we will be looking at we need to identify the limiting factors of the system. The two main limitations will originate from human and sensor capabilities. We expect that the speed of human interaction will not exceed several meters per second due to the physical limitations of the human body and the spatial limitations of an interactive surface. The sensor will have a maximal saturation after which it will not be able to reliably capture displacements. The limit of the maximal saturation will depend on the resolution of the optical sensor and the surface on which it is moving. We can define the bounds as follows:

$$upper\_bound = min(sensor\_max, human\_max), \tag{4.2}$$

where *sensor_max* is the upper bound of sensor capabilities and *human_max* is the upper limit of the speed of human movement, and similarly

$$lower\_bound = max(sensor\_min, human\_min), \tag{4.3}$$

where *sensor_min* is the lower limit of sensor capabilities and *human_min* is the lower bound of speed of human hand.

Units of output - conversion

Depending on the chosen sensor we might lack the knowledge of the units of the sensor or might be unable to convert those to real-life units. In that case the displacement will be an integer value of unknown units and we will need to post-process the speed estimation of the network to convert the units of the sensor to standard units such as m/s.

Types of movement

Movement can be characterised by the acceleration curves it produces. For example, an acceleration equal to zero will mean a constant speed and slowly varying acceleration will signify slow changes in the speed. However, rapid fluctuations in the acceleration curve will mean large differences in speed. If there are rapid changes in speed, the network will have to estimate the speed quickly in order to avoid aliasing.

Usefulness of the output

The output of the network will be a series of independent predictions of speed on a window basis. However, this representation does not reflect reality because in real life speeds are highly correlated from one point in time to the next. In order to capture this dependence relationship, we can use time based filters in a post-hoc fashion. We can apply simple linear filters such as moving average or we can use probabilistic filters which will use previous states to give more reliable predictions of the future state.

4.2.3 Surface type classification

Types of surfaces

Since we aim to use the system on variety of surfaces, we need to consider how to capture irregularities for as many different scenarios as we can. Let us consider surfaces with isotropic properties – we can expect these to produce the same audio signal irrespective of the direction of movement of a sensor. Next, we can have surfaces that have similar textures based on the direction of movement – for example a surface with straight, raised lines; we can expect these to produce similar acoustic patterns in certain directions. Finally, there are surfaces which are irregular in every direction and these are not expected to produce consistent results due to the unpredictable nature of the material, despite maintaining constant speed and direction of movement of a sensor. Moreover, irregularities of the same category will produce different acoustic patterns because of the profile height of the irregularities.

4.3 Summary

We have examined what information will be provided by the optical flow sensor and the contact microphone. To demonstrate how the signals from the two sensors will be used, we introduced a high-level abstract diagram displaying the data flow in the system. We have discussed the two tasks that we will attempt to solve and their different aspects, namely the regression task for speed estimation and the classification task for surface type detection.

5 Implementation

5.1 System overview

We have implemented a system which uses the information from the contact microphone to make continuous estimations about the speed of motion and about the surface type on which the interaction is taking place. We train this system using ground truth provided by an optical flow sensor which is not used during testing. This gives us two main modes of function: training and live predictions, as we can see in Figure 5.1.

5.1.1 Training overview

In order to train the network, we had to create a dataset which contains raw data from the optical flow sensors paired with raw data from the contact microphone along with associated metadata such as length of each recording, sample rates, or offset between each pair of signals.

Before using this data for training the neural network, we need to pre-process the data to a format which the neural network can process, for example this includes converting displacements into speed using Equation 4.1 or synchronising the audio and speed signal based on the offset. Hence, we designed and implemented a Data Generator pipeline which allows us to flexibly manipulate the data and generate the data in any format we need based on the parameters we specify. The key steps involved in pre-processing - described in greater detail in Section 5.3.2 - are:

- resampling of the audio signal
- conversion of displacements into speed signal and interpolation of this signal in order to have a regular fixed sample rate
- splitting the signal into windows of appropriate size
- obtaining windows from speed signal which correspond in time to the windows of audio signal based on the offset between the two signals
- adding a small artificial noise to the signal in order to extend the effective size of the dataset

After performing these steps the output of the Data Generator is passed to the model for training.

5.1.2 Live predictions overview

As we discussed in Section 4.2.1 we want to develop a stream-based system which can make predictions in an online fashion. Therefore, we want to pass fixed sized blocks of audio signal from the contact microphone into the model and generate predictions for each corresponding block of signal. However, this implies that there will be a latency between the start time of the motion and the time the model produces corresponding output of at least the size of the window because we have to record the whole window before passing it to the model. Moreover, there will be a computational cost to processing the block of audio signal in the model, which is dependent on the hardware on which the model is running and will further increase the latency.

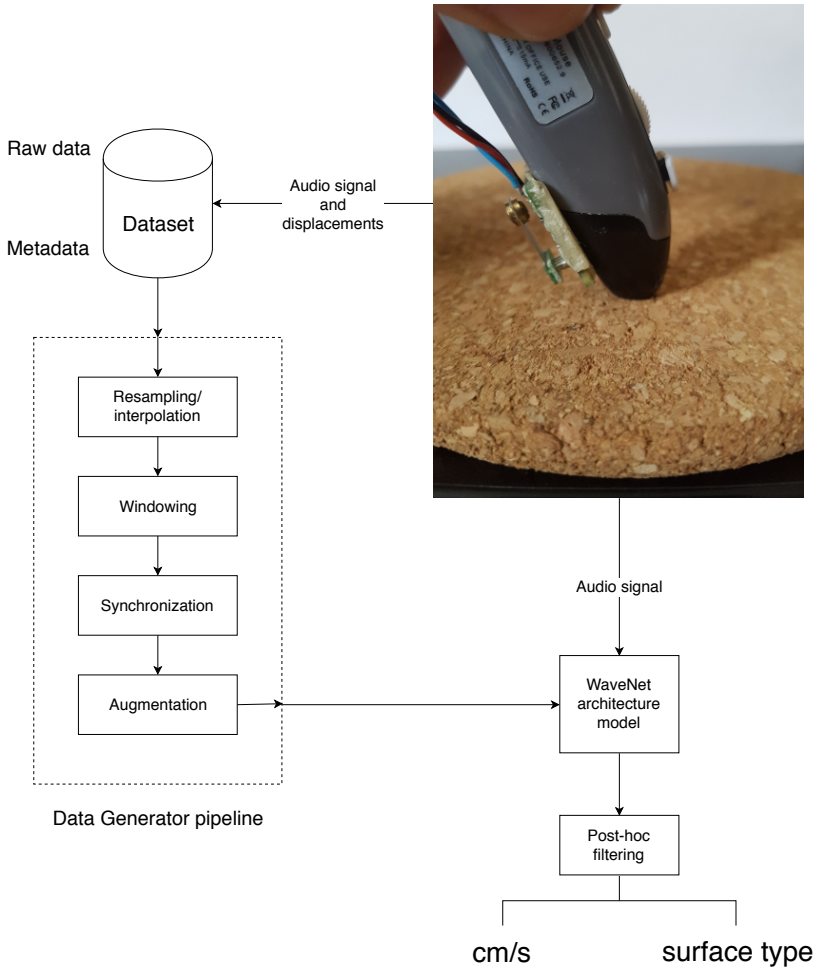


Figure 5.1: *Overview of the implemented system. Information from the optical flow sensor and contact microphone are stored into a dataset which is then used for training the model. Alternatively, the audio recorded with the contact microphone is passed in a fixed sized block directly to the WaveNet architecture model which allows for live predictions. A Data Generator pipeline was implemented which pre-processes the data from the dataset before using this data for training of the WaveNet model. The main pre-processing steps are converting the raw displacement values into the speed signal and interpolating the signal so that it is regularly sampled, resampling the audio signal, splitting the signal into separate windows of desired size, synchronising the audio and speed signal given the offset between them, and introducing a small amount of noise in order to augment the data. The output of the WaveNet model (i.e. block of speed estimations in the units of the sensor and one-hot encoding of surface type) is then post-hoc processed by converting the units of the sensor into real-world units (cm/s) and decoding the one-hot representation of the surface type back to a string representation. Post-hoc processing can include application of exponential or probabilistic filters which smooth the signal and reduce the noise.*

5.1.3 Output and post-hoc processing

For each block of audio signal passed to the network it will produce two outputs. Firstly, the model will produce a block of speed predictions in the units of the sensor and secondly it will estimate the surface type in one-hot encoding.

Although in the speed estimation we can see the change in speed, it does not tell us what this change means in real-world units. Hence, we will post-process the speed output by converting it from units of the sensor into cm/s using a conversion factor we calculated during the process of sensor calibration (discussed later). We will also decode the one-hot representation of the surface type into a string. Furthermore, we can use signal filtering techniques such as exponential or probabilistic filters on the speed signal in order to reduce the noise and capture the dependence between separate windows.

5.2 Hardware

5.2.1 Optical flow sensor

We experimented with different optical flow sensors and various configurations. At the start of the project we tried optical flow sensor ADNS-5050 connected to a USB-based microcontroller system Teensy 3.2 but the development board did not support the protocol which was necessary to establish communication with the sensor. In addition, we also realised that attaching the sensor to an object would not provide a stable fixed distance between the surface and the sensor, which is needed for optimal performance. Therefore, we have decided to use an off-the-shelf optical device such as an optical mouse. However, mounting the contact microphone onto an optical mouse (in our case Logitech MX510) proved difficult because we could not find a suitable place to minimise the noise introduced by the USB cable and to achieve the best signal capture. Hence, we used the wireless Pen Mouse (Fairy) Series which offers multiple positions for attaching the contact microphone and lacks the USB cable because the connection is wireless.

We wanted to get low latency access to the raw sensor values which were in the least distorted form before any acceleration curves are applied. While there are high-level libraries such as *pygame* which have the ability to get the displacement from an optical flow sensor, they introduce a high latency component which is sub-optimal for our use case. Therefore, in order to get the displacements measured by the optical flow sensor in the Pen Mouse with the lowest latency in Python we have used *evdev* library which allows to read in HID[1] values directly from the USB stream. This library reads in events from the */dev/input* on Linux OS on which the whole system was developed.

Maximal saturation

As mentioned in Section 4.1.1 the optical flow sensor will provide us with displacements in the x and y axis, however while using the Pen Mouse we noticed that the displacement values are limited to the range of -128 and 127 for each axis. Through empirical experimentation we have found that the implication of this is that the human hand can move faster with the sensor than the sensor can capture. We can see the saturation in Figure 5.2 where the sensor stops capturing speed above 178 units. The theoretical upper limit is 181 units calculated using the Equation 4.1, however that is assuming a perfect diagonal motion which is difficult to achieve with a human hand.

[1] Human Interface Device

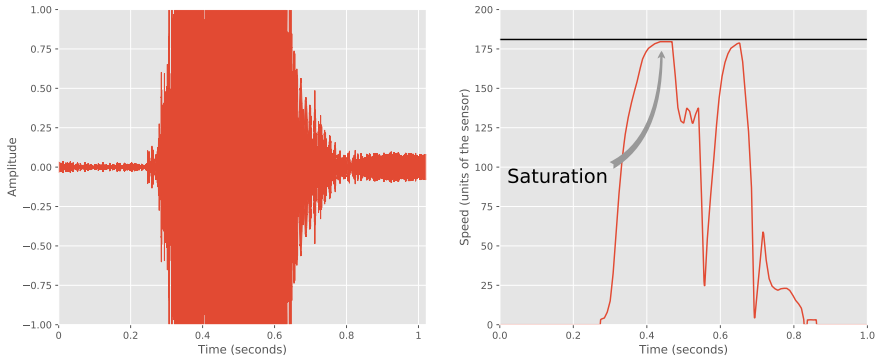


Figure 5.2: *On the left is speed calculated from the displacements obtained from the Pen Mouse showing the saturation when the device is moving faster than the sensor can capture. Horizontal line is at the upper limit of 181 units of the sensor calculated using Equation 4.1. On the right is the sound signal captured by the contact microphone (The MiniSense 100NM) during recording of the speed. We can see the saturation of the audio signal at the same time interval as the saturation of speed signal.*

Irregular sample rate

Moreover, there are no incoming events from the Pen Mouse when it is not moving. On one hand, this is understandable because the sensor does not want to clog the system with events that do not provide any new information about the position of the sensor. However, on the other hand, the implication is that the raw displacement values for x and y are irregularly sampled. Therefore, after recording the raw values we will need to apply an interpolation method to have a fixed sample rate before using the speed signal.

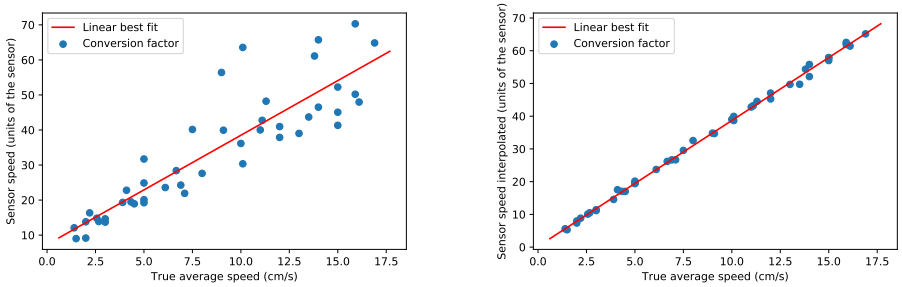
Calibration and real-world units

The Pen Mouse does not provide units in which the displacement is specified thus we lacked the knowledge of the units of speed in which we are measuring. Hence, we wanted to approximate the conversion factor between the units of the sensor and real-world units. Moreover, we wanted to be certain that there are no acceleration curves applied even despite choosing a very low level access strategy for reading in the raw values. Because of this, we have recorded a separate calibration dataset where for each recording we have also recorded the average speed by measuring the distance travelled in a certain amount of time. The average speed was calculated using the standard equation for average speed:

$$avg\_speed = \frac{\Delta d}{\Delta t}, \tag{5.1}$$

where avg_speed is the average speed expressed as change in distance (Δd) over some time interval (Δt).

In Figure 5.3a we can see the true average speed measured against the average speed calculated from the displacement measurements using Equation 4.1. Although the measurements are noisy, there is a linear relationship between them, which is shown using the best fit line in the same figure. Moreover, the measurements become even more stable after interpolation which can be seen in Figure 5.3b where the linearity of the relationship between measured values and true speed is clear. From this calibration we have calculated that the conversion factor between the units of the sensor and cm/s is 0.258 with a standard deviation of 0.008.



(a) Calibration using unaltered speed measurements *(b) Calibration using interpolated speed signal*

Figure 5.3: *Calibration for the values from the Pen Mouse showing that there are no acceleration curves applied and the results are stable. Using this calibration we have calculated the conversion factor between the units from Pen Mouse and cm/s which is 0.258 with a standard deviation of 0.008.*

5.2.2 Contact microphone

To capture the vibrations produced by the interaction we have used a high sensitivity vibration sensor The MiniSense 100NM (see Figure 5.4) with an amplifier which helped us interface with the sensor and allowed for capturing single channel of audio. This sensor is mounted onto the bottom part of the Pen Mouse using beeswax which has excellent acoustic propagation properties while simultaneously providing a robust connection without secondary vibrations. Although we did not expect to capture very high frequencies, we recorded the audio with a sample rate of 44.1kHz which means that there should be no issues with aliasing. Similarly to the Pen Mouse, our contact microphone can also be saturated during fast movements as can be seen in Figure 5.2.

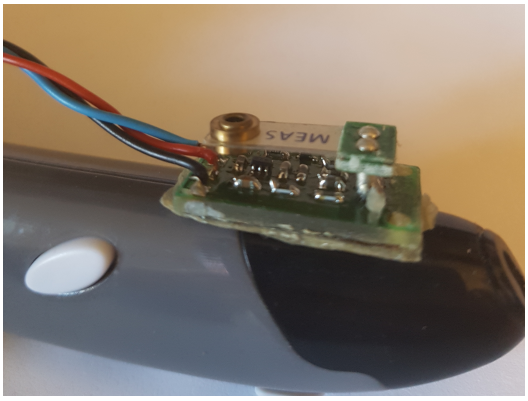


Figure 5.4: *Picture of the MiniSense 100NM mounted on the Pen Mouse. This contact microphone was used in this project.*

Frequency response

We have also approximated the frequency response of the contact microphone by placing the MiniSense 100NM onto a speaker and recording white noise. This helped us find out to what frequencies the microphone is most sensitive. In the documentation of the MiniSense 100NM we found that the microphone is most sensitive between 10Hz and 100Hz (TE-Connectivity 2018). In Figure 5.5 we can see that our calibration of the contact microphone supports this and in fact the microphone is most sensitive at 85Hz. However, the microphone is also considerably sensitive at 1000Hz where according to the documentation it should have very low response. This is most likely caused by the amplifier used to interface with the MiniSense 100NM.

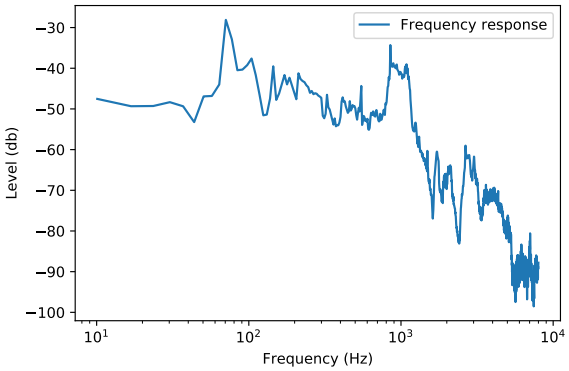


Figure 5.5: *Frequency response of the MiniSense 100NM approximated by placing the microphone onto a speaker and recording white noise. There are two considerable peaks - first is at 85Hz which corresponds with the documentation, however the second at around 1000Hz is most likely caused by the amplifier used to interface with the microphone.*

5.3 Machine learning

5.3.1 Dataset

In order to use machine learning for our problem, we needed to create a dataset with audio recordings coupled with reference displacements. As we mentioned in Section 4.1.1 we are saving the raw displacement values that we get directly from the optical flow sensor and then in our pre-processing pipeline we calculate the speed using Equation 4.1 (see Section 5.3.2).

We have recorded 10 minutes of audio with displacement reference on each of the four surfaces (surfaces detailed later) and stored those recordings in a Hierarchical Data Format version 5 (HDF5). This format provides a convenient way of large structured data storage which can be queried in a memory-efficient way by directly accessing specific recordings or parts of specific recordings. HDF5 format is similar in structure to a filesystem on a POSIX[2] operating system, where group represents a folder and dataset represents a file. Both group and dataset can have number of metadata attributes associated with them.

In Listing 5.1 we can see the schema of our HDF5 file where the *swipe* group contains all of our recordings. The reason for not having all the recordings in the *root* group is because all our recordings have the characteristic of a continuous motion on a surface. However, if we wanted to extend the dataset with a different interaction such as tapping in order to do a gesture recognition as well, we could simply add a new group on the same level as *swipe* and have a clear division between recordings for different interactions.

Inside each recording group, whose name is created using the date and time of the recording to achieve uniqueness, there is a dataset for the displacements and dataset for the associated audio signal. The dataset for displacements contains displacements in the x and y axis along with time stamps, which will allow us to interpolate the signal to have a fixed sample rate in the pre-processing pipeline later on. Each recording group has a number of metadata attributes which specify the length of the recording, sample rates of the two signals, offset in seconds between the two signals, and surface type on which the recording was made.

[2]Portable Operating System Interface

```
# root group
/
   # swipe group
   swipe/
      # recording group
      yyyy-mm-dd hh:mm:ss.xxxxxx/
         movement
         sound
         .metadata/
            length
            offset
            sound_sr
            speed_sr
            surface
      yyyy-mm-dd hh:mm:ss.xxxxxx/
      yyyy-mm-dd hh:mm:ss.xxxxxx/
      ...
```

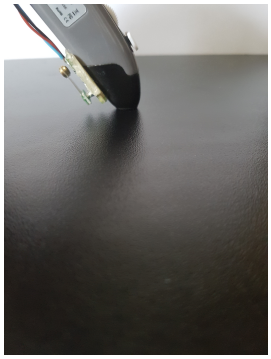
Listing 5.1: *Schema of the HDF5 file containing all the recordings. Each recording group is comprised of a movement dataset - displacements in x and y axis along with time stamp for each sample, and a sound dataset - audio signal recorded with the contact microphone. In addition, each recording group has a metadata attributes which give us more detail about a recording, such as the length of the recording in seconds (*length*), offset between the two dataset signals in seconds (*offset*), sample rate of the sound and desired sample rate of the speed (*sound_sr *and* speed_sr*, respectively), and string detailing type of the surface on which the interaction happened (*surface*).*

Surfaces

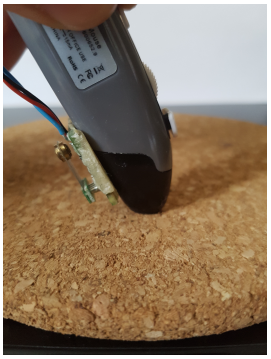
While we cannot record data on every possible surface due to time constraints we wanted to have surfaces where each captures a different texture characteristic. Therefore, we chose materials which we classify as *wood* (Figure 5.6a), *cork* (Figure 5.6b), *paper* (Figure 5.6c), and *plastic* (Figure 5.6d). Each of these materials has different properties. Figure 5.6a shows a common surface for office tables making it an example of an everyday surface. Similarly, material in Figure 5.6b is an example of everyday surface and we can see it is highly textured. While we did not have manufactured surface at our disposal we approximate one using the cover of a plastic document folder shown in Figure 5.6d which has highly textured surface with two uniform areas. Depending on the direction of motion, these have a different acoustic profile. Therefore, *plastic* has similar characteristic to the surfaces discussed in Section 2.3.

Signal synchronisation

When we are recording any two signals there will always be a problem with synchronisation because we can never start the recording at the exact same time. We use the time of the start of recording to calculate the offset between the two signals, which we then use to align the two signals. An auto-correlation method could be used to calculate this offset, however we found that calculating the offset from the start times of the recordings is sufficient as we will show in Section 5.3.2.

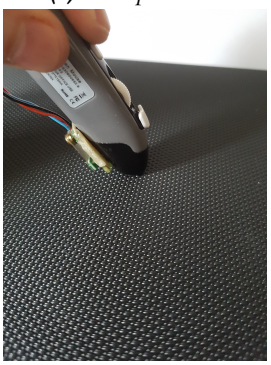


(a) *Particle board covered with ABS plastic*



(b) *Cork pot stand*

(c) *Sheet of paper*



(d) *Plastic folder*

Figure 5.6: *Four chosen surfaces on which our dataset has been recorded. Each of the surfaces has different texture properties. All surfaces are shown with the optical pen for reference. In (a) we can see a particle board covered with ABS plastic; this surface represents the most commonly found surface in everyday life where the surface is not completely smooth but has some small texture which is noticeable by touch alone while being firm. In (b) we can see cork which has a highly textured surface but the texture is not uniform in any direction of movement. (c) shows a multi-purpose paper with thickness of 80 gsm (grams per square meter), which is an example of a very smooth surface. The last chosen surface is (d), which is highly textured with two different texture types which are orthogonal to each other.*

5.3.2 Data Generator

In accordance with good scientific practice after storing data in the rawest form obtainable, we created a generator pipeline which fetches the data from the storage, performs several pre-processing steps and then passes them to the machine learning algorithm in the correct format. The main pre-processing steps it performs are resampling, windowing and augmentation.

We can specify multiple parameters in the Data Generator and by changing these we can flexibly manipulate the data and generate them in any format we want in order to train our neural network. We define these parameters in a separate file (*config.json*) which makes it easier to change them. Examples of the most important parameters are *batch_size* (which sets the number of batches produced by the Data Generator), *window_len* (which specifies how large the time window passed to the network should be in seconds), *sample_rate* through which we can change the sample rate of the audio signal, and *overlap* allows us to specify what overlap should be between windows, which increases the size of the dataset but also introduces more correlation between windows.

Through other parameters we can also specify if we want to take windows randomly and not sequentially, if we want to add noise to the windows, if we want to output only the speed window corresponding to the audio window or also the surface type in one-hot encoding, what mode should be used to do the resampling, or what size the kernel should be for the median filter used to smooth the speed signal after interpolation.

The key operations the Data Generator performs are as follows:

- **Fetching data from dataset** – Our dataset is a structured collection of raw data with associated metadata and using HDF5 format allows the Data Generator to fetch the exact slices of data that we want without the need to load the whole file into memory which makes it very memory efficient. Another advantage of using HDF5 format is that it allows to associate user defined metadata with each group and/or dataset, which means that we can specify the sample rate in which the recordings were made or the units of the stored values. In addition, it allows to store flexibly-sized datasets and therefore our recordings can be of variable length.
- **Resampling** – When loading the audio signal from our data storage it has a sample rate in which it was recorded (i.e. 44.1kHz), but that might not be the sample rate we want to use for the machine learning and hence the Data Generator allows to resample the audio signal to the desired sample rate. However, resampling the audio signal can be computationally expensive and can therefore increase the training time.
- **Interpolation** – As we discussed in Section 5.2.1 the stored displacements are irregularly sampled and to do any machine learning we need to use an interpolation technique to get a regular sample rate. We use linear interpolation on the displacements converted into speed using Equation 4.1 and we use the time stamps of when each sample was recorded to help the interpolation function to better approximate the speed signal. We chose the sample rate for speed to be 1000Hz, which gives us enough values for a window length of 0.01 seconds but minimises the number of values that had to be interpolated.
- **Alignment** – As we mentioned in Section 5.3.1, the two recorded signals are not perfectly aligned and we need to be certain that window from the audio signal will be paired with the corresponding window from the interpolated speed signal. We use the offset calculated from the difference between the start times of the two recordings. In Figure 5.7 we can see how the Data Generator aligns the two signals using the offset calculated in this way and moreover we can see that this is a sufficient synchronisation technique.
- **Windowing** – We can only pass a fixed number of samples to the machine learning and therefore we must divide the signal into equally-sized windows, where the size of each window is defined by *window_len* parameter. For example, if the *sample_rate* is 44,100Hz and the *window_len* is 0.1 seconds, the size of the input window will be 4410 samples. Having *window_len* and *sample_rate* as parameters of the Data Generator we can flexibly change the input size. For every window from the audio signal we find the corresponding

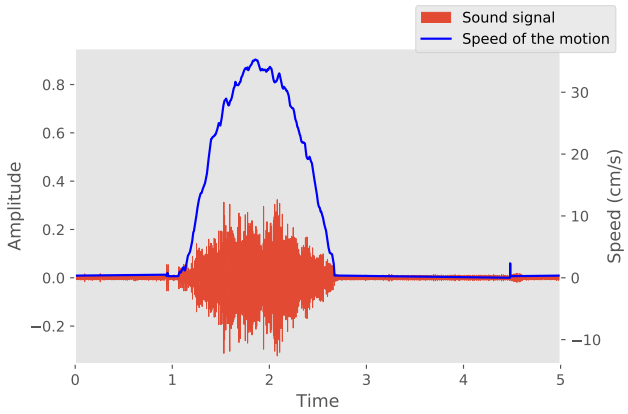


Figure 5.7: *Sound signal aligned with the speed signal as the output of the Data Generator using the offset calculated from the difference between start times of recording of the two signals*

window from the speed signal given the offset of the two signals. Moreover, we can also specify *overlap* parameter which increases the number of windows by producing windows that are partially overlapping.

- **Augmentation** - The last step in the pre-processing is to add noise to the audio signal, which is a data augmentation technique that helps us expand the effective dataset size. Adding Gaussian noise to the signal and amplitude modulation are two common techniques used for dataset augmentation[3], which we implemented in the Data Generator.

5.3.3 Deep learning architecture

We have tried using the standard approach to sequence modelling by implementing recurrent neural networks such as Long Short-Term Memory architecture, however as we have discussed in Section 2.8 autoregressive feedforward architectures have shown improved performance over recurrent neural network models in sequence modelling tasks. Hence, we have implemented a state-of-the-art WaveNet architecture with dilated causal convolutions, gated activation units, skip and residual connections as described in Oord et al. (2016a). Implementation was done using *Keras*[4] library for Python.

In Figure 5.8 we can see the architecture of our deep learning model. We set the input window size to 0.1 second (with sample rate of 44.1kHz this is equivalent to 4410 samples per window) which gives us the lower bound of latency of 0.1 seconds. Next, we pass the input to 4 WaveNet blocks where each block accepts increasingly dilated input (the increase is by a factor of 2) and consists of gated activation unit as described in Oord et al. (2016a). Each WaveNet block produces two outputs, one for the next WaveNet block with added residual connection and one without the residual added, called skip-connection. These skip-connections are added together after the last WaveNet block and passed further into the network. Following *ReLU* activation and two convolutional layers both with single activation unit, we split the network into two branches (see Listing A.1), one for regression and the other for classification.

Regression branch

In the regression branch of the network we have tried using *tanh* as an activation function for the convolutional layers but that caused the network not to converge during training. Similarly, the network did not converge during training when we added a Dense layer with

[3]https://www.kaggle.com/CVxTz/audio-data-augmentation
[4]https://keras.io/

tanh activation after the last convolutional layer or when we experimented with the advanced activation function *LeakyReLU*. Moreover, we used the *softplus* activation function for the last Dense layer because when we used *linear* activation, we noticed that the network produced negative values for speed, which is physically impossible. Therefore, using *softplus* activation function forces all the output values to be positive (see Listing A.2). The output of this branch is a window capturing the same time interval as the input window, only with a different sample rate of 1000Hz.

Classification branch

Similarly to regression branch, we have also tried to use the *tanh* activation function in the Dense layers of the classification branch (see Listing A.3), however this resulted in a bias towards one particular surface type which changed depending on the number of units in each layer. In order to convert the string which represents surface type (i.e. *wood*, *cork*, *plastic*, *paper* as described in Section 5.3.1) into numerical representation, we used standard one-hot encoding. Hence, the output of this branch is a binary array of size 4 (i.e. one value for each surface) where all values are zero except the predicted surface type, which has a non-zero positive value.

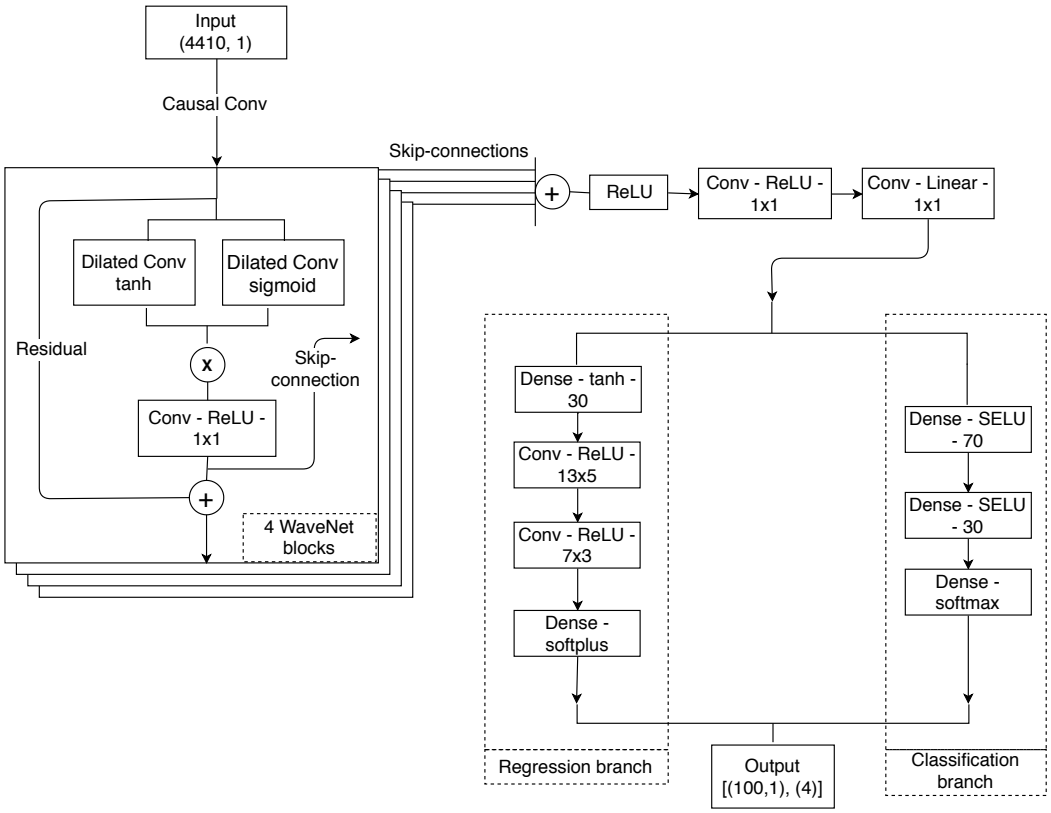


Figure 5.8: *Architecture of our implemented model starts with 4 WaveNet blocks (i.e. layers of dilated - by a factor of 2 - causal convolutions with gated activation units and residual and skip-connections (Oord et al. 2016a)) which along with* ReLU *activation and two convolutional layers both with 1 unit and kernel size set to 1 form the common part of the network before we split the network into two branches. One branch is for the regression task of speed estimation and another is for the classification task of surface type detection. In the regression branch we use Dense layer with 30 units and* tanh *activation function along with two convolutional layers both with* ReLU *activation function, where the first layer has 13 units and a kernel size of 5 and the second has 7 units and a kernel size of 3. The regression branch ends with a Dense layer with* softplus *activation function, which forces the network to only output positive speed estimations. In the classification branch we use two Dense layers with* SELU *activation function, where the first layer has 70 units and the second has 30 units. The classification branch ends with a Dense layer with* softmax *activation function, which is an appropriate function for single-label multi-class classification problems such as surface type detection. The network outputs a list of two elements, where one element is the sequence of speed predictions and the other is surface type in one-hot encoding.*

5.3.4 Training

We have split the whole dataset into test and training data: 25% of the dataset, unseen by the model, was used for testing purposes and evaluation (see Section 6). Moreover, we used 5-fold cross validation to minimise the effects of over-fitting with 20 epochs per single fold.

Two loss functions had to be used in order to train the network as we had two different tasks. We used *mean squared error* for the regression task and *categorical crossentropy* for the classification task. Using *categorical crossentropy* as a loss function for the classification task was appropriate because we have a single-label multi-class classification problem. In addition, *mean squared error* is a standard loss function used in regression tasks. We have used the Adadelta optimiser for our training with the default parameters given in *Keras*.

In order to decrease the training time we used early stopping which allowed us to skip the

remaining epochs if the loss function did not decrease. In our case, if the value of the validation loss does not decrease by 5 units in 5 epochs we will stop training and move on to the next fold.

Another advantage of the Data Generator apart from those described in Section 5.3.2 is that we can pass the Data Generator object to the training function and it will internally call the implemented functions to get batches of data at a time.

Training time

All models evaluated in Section 6 were trained on a system with GPU acceleration using a Nvidia GeForce GTX 1060 graphics card. The exact times for training are detailed in Table A.1 but they ranged from 2.5 hours to 11 hours depending on the model's complexity and window size, because a smaller window size means more batches per epoch, which results in longer training time.

Hyperparameter optimisation

Many of the options for activation functions, number of WaveNet blocks, window sizes, optimisers or number of units in some layers were explored automatically using the hyperparameter optimisation tool Talos[5]. However, the number of hyperparameters specified caused a 'combinatorial explosion', meaning that the optimisation did not finish and therefore we used the validation loss of the finished part of the optimisation to determine the best parameters from the combinations of parameters that Talos was able to evaluate. The resulting architecture is described in Section 5.3.3.

5.4 Post processing

5.4.1 Live predictions

To show that the model can be used in real-time we have implemented a simple visualisation program using *pyqtgraph* library which displays the predictions from the network along with the reference speed from the sensor and the audio signal passed to the model as input. We have used *pyqtgraph* library because doing real-time visualisation can be slow with standard high-level Python libraries such as *matplotlib*.

5.4.2 Signal filtering

As we mentioned in Section 4.2.2 the output of the network does not reflect reality due to the independence of the window-based predictions. In order to capture the correlation between windows we can use a time-based filter.

Holt-Winters filter

One type of time-based filter is exponential smoothing. We implemented a second-order exponential smoothing using the Holt-Winters method. In Figure 5.9 we can see the difference between the unfiltered predictions which are noisy and oscillate rapidly, and the predictions which have been smoothed using Holt-Winters method of second-order exponential smoothing (Kalekar 2004). We can specify parameters that determine how much the signal will be smoothed but these need to be chosen carefully depending on the type of signal, the purpose of the application and the amount of noise. We have experimentally found the parameters that smooth

[5] https://github.com/autonomio/talos

the speed signal (so it does not oscillate as much) but also so that it preserves the important changes in the signal.

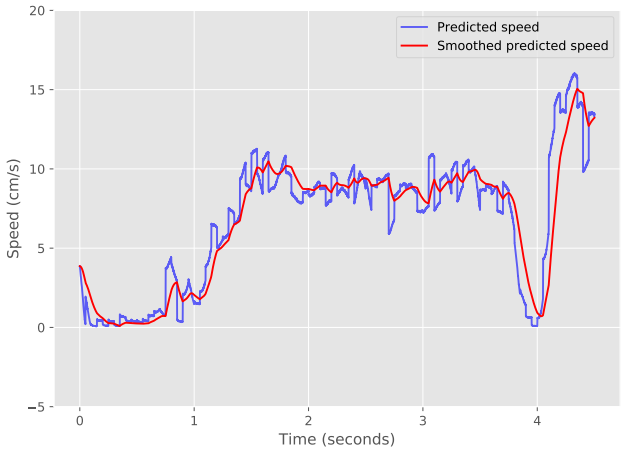


Figure 5.9: *Speed predictions for the same acoustic signal. In blue the raw predictions from the model and in red the smoothed predictions using Holt-Winters second-order smoothing.*

Kalman filter

Another type of time-based filter is a probabilistic Kalman filter which is, for example, widely used filter in satellite navigation systems. It makes a prediction about the next state using the previous state and corrects future predictions based on the observations from sensors. Therefore, we wanted to use a Kalman filter with the optical flow sensor and predictions from the machine learning to track the movement of the optical pen. We were successful in implementing such a filter using *pykalman* library and were able to use a Kalman filter for post-hoc processing of the speed signal and tracking of the movement of the mouse. However, our hope was that since a Kalman filter is probabilistic in nature, it can still function even when some of the sensors fail. We wanted to simulate the failure of the optical flow sensor and show that using just predictions from the machine learning, the Kalman filter is still able to predict the movement of the optical pen with a degree of error. However, due to the limitations of the *pykalman* library used this was not possible.

6 Evaluation

6.1 Regression task

In the regression task we use acoustic information to predict the current speed of the interacting object.

6.1.1 Regression metrics

In order to evaluate the performance of the model on the regression task, we chose to use root mean squared error (RMSE) as our main metric with standard deviation (SD). RMSE is a standard metric for measuring the difference between two time series of particular data and is frequently used for regression tasks where we are estimating a value of variable that changes with time. The advantage of RMSE is that it has the same units as the dependent variable. For example, RMSE between two speed signals in cm/s will have the same units, i.e. cm/s. We report the mean of RMSE values for each predicted window and SD then gives us the variation of the RMSE value over all windows. We calculate the RMSE using Equation 6.1 for each predicted window from speed signals in cm/s, which means that RMSE can be interpreted in the same units. The formula for RMSE is:

$$RMSE = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(x_{1,n} - x_{2,n})^2}, \tag{6.1}$$

where N is the number of samples in a window, x_1 is the predicted speed window and x_2 is the reference speed window. Moreover, SD can be calculated using the formula:

$$SD = \sqrt{\frac{1}{N-1}\sum_{n=1}^{N}(x_n - \hat{x})^2}, \tag{6.2}$$

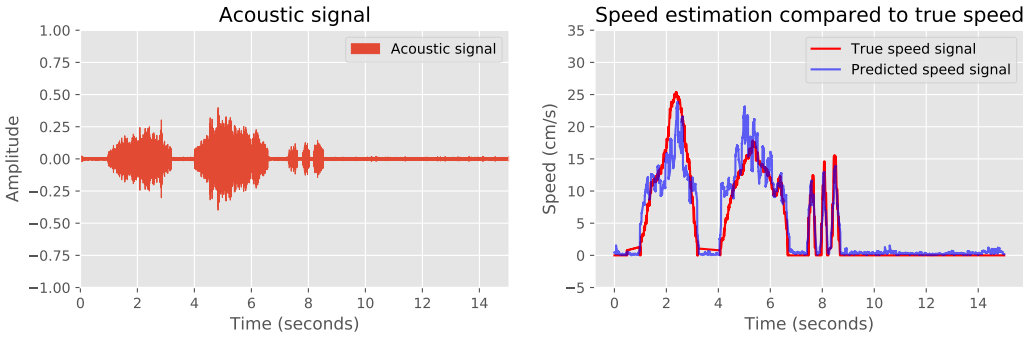
where N is the number of values and $\hat{x}$ is the mean of all the values.

6.1.2 How well can we predict the speed of movement across surfaces from acoustic information?

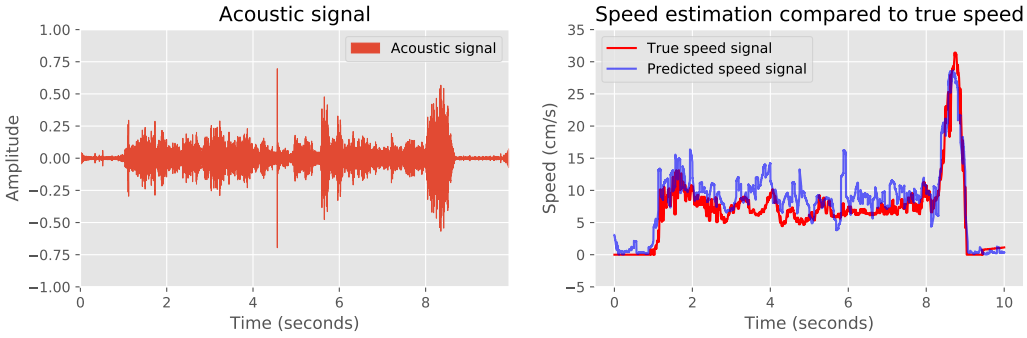
We chose the best architecture based on the validation loss (i.e. mean squared error) of the finished part of hyperparameter optimisation described in Section 5.3.4. Therefore, the best model's architecture accepts windows of size 0.1 second, with 4 WaveNet blocks and with other layers in the two branches for regression and classification as described in Section 5.3.3. This architecture reports overall mean RMSE ± SD equal to 2.610 ± 2.526 cm/s on the unseen test data. This means that on average the difference between the true speed and the predicted speed is 2.610 cm/s ± 2.526 cm/s. Therefore, for some windows the predictions can be very precise where the RMSE is almost zero and in other cases the error is much larger. The cases where performance is poorer are discussed later in this Section.

Figure 6.1 shows how the predicted speed signal looks compared to the true speed signal for different surfaces and also the amplitude envelope of a waveform of the acoustic signal which was used to produce the estimations. We can see that the raw predicted signal is very similar to the true signal in both cases, however we will discuss later in this section that the model

performs worse for higher speeds. Poorer performance for higher speeds might be the cause for relatively high average RMSE of 2.610 cm/s, despite the predictions being very similar to the true speed as we can see in Figure 6.1.



(a) Predictions for acoustic signal recorded on wood



(b) Predictions for acoustic signal recorded on cork

Figure 6.1: *Speed prediction produced by the best model overlaid over the true speed as recorded by the optical flow sensor for reference along with the amplitude envelope of a waveform of the acoustic signal that was recorded during the interaction. (a) shows predictions and recording made on* wood *and (b) displays predictions and audio recording on* cork.

What is the influence of the surface on the ability to predict?

In order to answer this question we plotted a box plot of RMSE for all the predicted windows on each of the surface and overall. We can see in Figure 6.2 that there are numerous outliers for each surface, and while the predictions on *paper* have the lowest RMSE median of all the surfaces, there are also by far the most outliers. Predictions on *cork* and *plastic* show similar distribution of RMSE which is also the lowest. This suggests that performance is better on highly textured surfaces no matter the kind of texture of the surface (i.e. whether the texture is regular as is the case of *plastic* or highly irregular in the case of *cork*). Predictions on surfaces which are smoother (i.e. *wood* and *paper*) have larger distribution and, in the case of *paper*, also the highest number of outliers. The overall distribution of RMSE on all four surfaces lies between the distribution on *paper* and *plastic*, which is logical given that there are two surfaces which are smooth on which the performance is lower and two surfaces which are highly textured on which the performance is better.

What is the performance on unseen surfaces?

We wanted to evaluate how well our model generalises to other unseen surfaces. Hence, we have recorded a small test dataset (1 minute of recording for each surface) on *wall* and *denim*. The

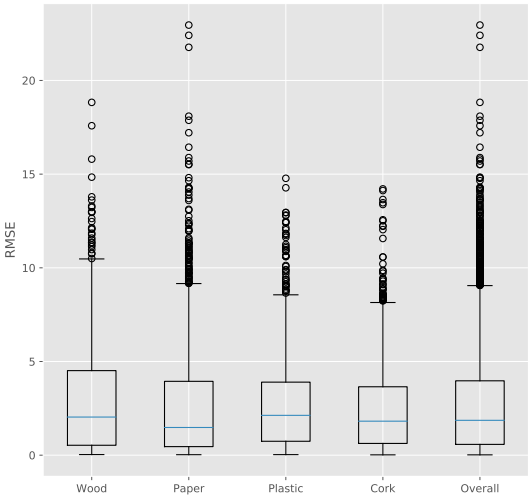


Figure 6.2: *Boxplot showing the distributions of RMSE of windows for each of the 4 surfaces and overall. Distribution on* cork *and* plastic *is similar and lowest across the 4 surfaces which suggests better performance for speed prediction on highly textured surfaces.*

average RMSE with SD is reported in Table 6.1 from which we can see that results on *denim* are close to results on the training surfaces. Moreover, we can seen in Figure 6.3 that the predictions are similar to the reference speed signal. On the other hand, the average performance on *wall* is significantly worse than performance on other surfaces (i.e. the average RMSE is 1.9 times worse than performance on *denim* and 2.9 times worse than the average overall performance on all surfaces). However, it is important to note that the optical flow sensor did not work well on *wall* and therefore we suspect that while the RMSE might be high, it can actually be closer to the true speed than the reference which is used to calculate the RMSE.

Table 6.1: *Average RMSE and SD for unseen surfaces showing that the model can be used on certain new surfaces. In this case predictions on* denim *are close to the average RMSE on all training surfaces being worse by 1.327 cm/s on* denim*. However, performance on* wall *is 2.9 times worse than the average overall performance.*

Surface	RMSE (cm/s)	+/- SD (cm/s)
Denim	3.937	2.510
Wall	7.606	4.885

What speed profiles/ranges are effectively predicted?

There is a clearly visible trend shown in Figure 6.4 where the performance decreases with increasing speed on all surfaces, although there are surfaces on which we can predict higher speeds more reliably than on others. Given the 3rd order polynomial fit in Figure 6.4 to highlight trends we can see that predictions on *cork* and *wood* have RMSE below 5 cm/s up to speed of 25 cm/s, while predictions for *plastic* and *paper* start to have RMSE above 5 cm/s, and therefore be less precise, in slower speeds (i.e. 18 cm/s and 15 cm/s respectively). It is important to note that for predictions on *wood* we can see a visible increase in RMSE for speeds of around 4 cm/s, which suggests that those speeds are harder to predict. We can see similar but not as significant increases for predictions on *plastic* and *paper* around the same speed. This indicates

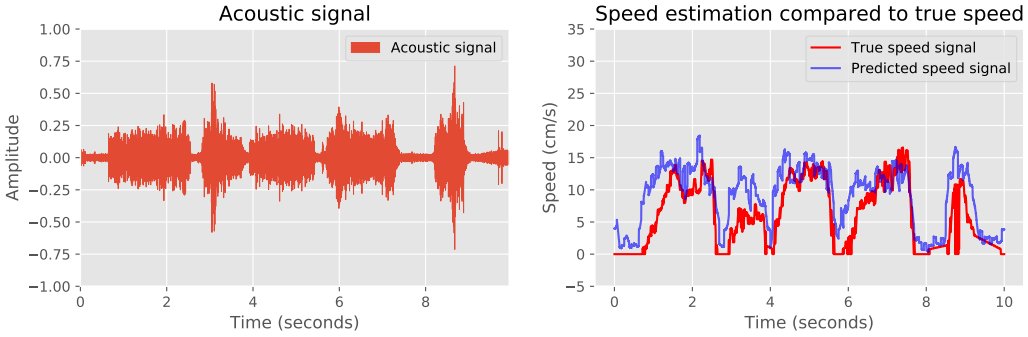


Figure 6.3: *Speed prediction on unseen surface* - denim *overlaid over the reference speed that was recorded along with the acoustic signal which was used to generate the prediction.*

that we can reliably predict very slow speeds between 0 and 2 cm/s, after which there is a drop in performance for speeds between 3 and 5 cm/s and again for the range between 6 and 15 cm/s the performance increases. This phenomena is not visibly present in predictions on *cork*, suggesting that on *cork* we can reliably predict speeds in a continuous range from 0 to 20 cm/s. This could be caused by the highly textured surface of *cork* which produces a distinctive acoustic signal even at slower speeds.

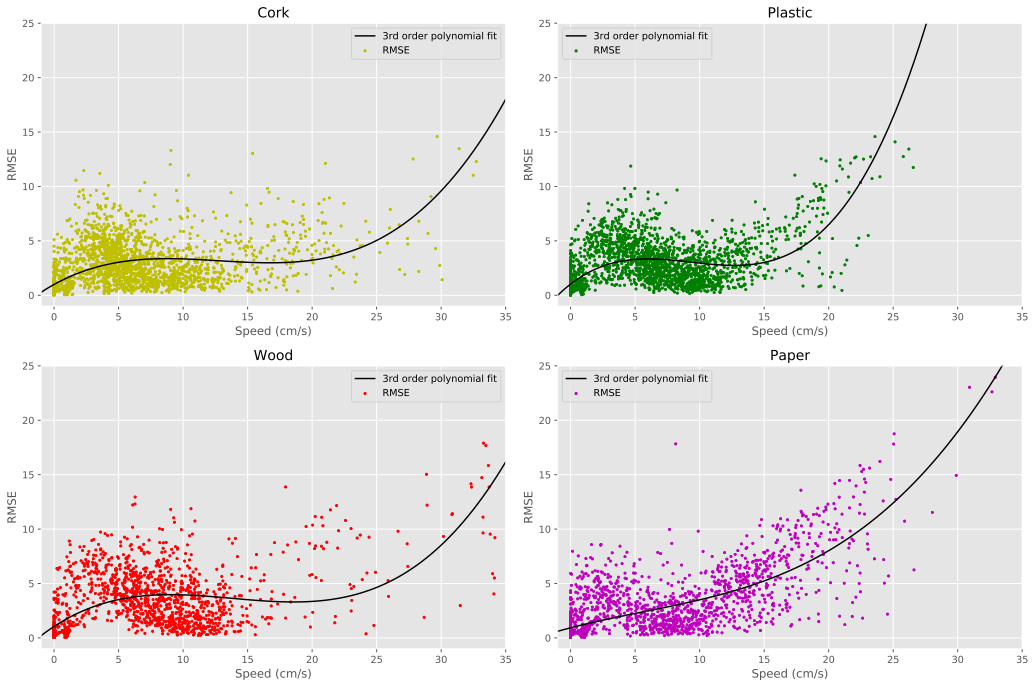


Figure 6.4: *We show RMSE as a function of speed for four different surfaces fitted with 3rd polynomial line to highlight the trends in the graphs.*

How well calibrated are those speeds?

In Section 5.2.1 we discussed how well the optical flow sensor in Pen Mouse is calibrated with respect to real world units and whether there are any acceleration curves applied on the recorded data. Through empirical experimentation we found that there are no acceleration curves applied and that after interpolating the speed signal we obtain a very stable readings. Similarly, we

can ask whether the implemented WaveNet architecture model does not introduce any sort of scaling or whether it is producing stable predictions. Using the same calibration dataset as in Section 5.2.1, we found that the predictions are not as stable as the interpolated speed signal which was almost perfectly linear. However, it also does not introduce any scaling or acceleration curves as we can see in Figure 6.5. This indicates that there is also a stable linear relationship between the predictions and real world units.

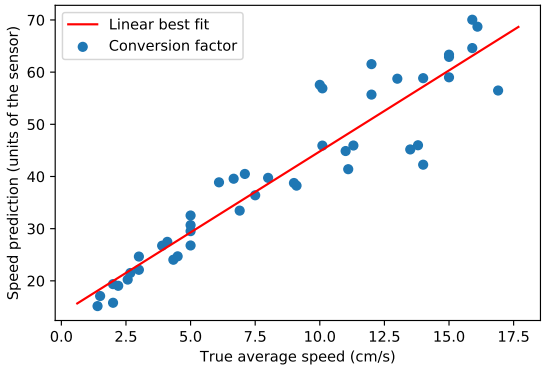


Figure 6.5: *Calibration between predictions and true speed shows that estimations are stable and there are no acceleration curves introduced by the model.*

How much data is required to accurately estimate speeds?

We created a dataset with 10 minutes of recordings for each of the chosen surfaces. We used 25% of the whole dataset for testing and the rest (i.e. 75% of the dataset) constituted 100% of our training data. Percentages in Table 6.2 are subsets of the training dataset (i.e. 100% is 75% of the whole dataset). We can see in Table 6.2 the effects of the training dataset size on the average RMSE across surfaces and overall. There is a slight increase in the overall performance with 75% of the training data, which can be caused by bias in that portion of the dataset which is also present in the test data. Hence, the model can be over-fitted on particular features which are partially present in the test dataset. Following this, the overall performance decreases with the decreasing size of the dataset either as the average RMSE score increases or as the standard deviation from the score increases. There is no clear trend between the performances on specific surfaces with decreasing dataset size, however performance on *cork* is consistently best across all surface regardless of the training dataset size. Furthermore, we can see that although the performance is slightly worse with decreasing dataset size, the difference between performances is not as significant as we expected and even with just 5% of the training data, the performance is comparable to the one of the model trained with 100% of the data. This is directly in contrast with what we established about end-to-end learning, i.e. that it requires a large labelled dataset in order to perform well.

Although not shown here, we have evaluated how well the model trained on 5% of the training data predicts higher speeds. The upper limit of reliable predictions (predictions with RMSE of most windows under 5 cm/s) is much lower for all surfaces. This indicates that using more training data could improve the model's ability to predict a wider range of speeds, in particular predict even higher speeds (15 cm/s and above) reliably.

Which elements of the network architecture contribute to the accurate estimates of speed?

The results from the hyperparameter optimisation indicated that the best architecture is one with 4 WaveNet blocks, however due to the aforementioned 'combinatiorial explosion' of the

Table 6.2: *Average RMSE results with SD reported as 'RMSE (cm/s) ± SD (cm/s)' for each surface and overall for models trained on different portion of the training dataset. We highlight the main model that was identified as best using hyperparameter optimisation output.*

Dataset size	Cork	Wood	Plastic	Paper	Overall
100%	2.399 ± 2.155	2.856 ± 2.732	2.603 ± 2.237	2.653 ± 3.013	2.610 ± 2.526
75%	2.257 ± 2.122	2.411 ± 2.389	2.403 ± 2.242	2.841 ± 3.129	2.463 ± 2.485
30%	2.110 ± 2.264	2.118 ± 2.644	2.285 ± 2.577	3.287 ± 3.925	2.431 ± 2.919
5%	2.462 ± 2.444	2.767 ± 3.083	2.501 ± 2.192	2.775 ± 3.212	2.609 ± 2.721

hyperparameters, the optimisation did not finish in its entirety and we chose the best architecture based on the validation loss of the finished part as we discussed in Section 6.1.2.

For this reason we have performed an ablation study on the WaveNet blocks of our model because we expect that they constitute the main part of our architecture. However, having more WaveNet blocks in the architecture could result in a model which is more difficult to train with the amount of data we created.

From Table 6.3 we can see that model with 8 WaveNet blocks has better overall RMSE score than our main model with 4 WaveNet blocks leading us to the conclusion that such a combination of parameters was in the unfinished part of the hyperparameter optimisation. In addition, model with 2 WaveNet blocks has a seemingly lower RMSE score, but it also has more variation of the RMSE given the larger SD, therefore having more oscillation than a model with 4 WaveNet blocks. Overall, average scores for the models with different numbers of WaveNet blocks are very similar to each other, indicating that number of WaveNet blocks does not affect the predictions as much as we expected.

Table 6.3: *Results from an ablation study performed on the number of WaveNet blocks of the network reported as average RMSE ± SD. Using 8 WaveNet blocks produces the best average RMSE overall. Note that performance on* cork *is consistently the best across the four materials. This could be due to the highly textured surface of* cork*. We highlight the main model that was identified as best using hyperparameter optimisation output.*

# WaveNet blocks	Cork	Wood	Plastic	Paper	Overall
2 blocks	2.123 ± 2.104	2.584 ± 2.942	2.341 ± 2.268	2.813 ± 3.349	2.437 ± 2.676
4 blocks	2.399 ± 2.155	2.856 ± 2.732	2.603 ± 2.237	2.653 ± 3.013	2.610 ± 2.526
8 blocks	2.103 ± 2.096	2.164 ± 2.419	2.341 ± 2.382	2.771 ± 3.218	2.337 ± 2.556

What is the effect of window size on the performance?

Changing the window size of the input effectively changes the lower bound of round trip latency as discussed in Section 4.2.1, which means that increasing the size of the input window will increase the latency. However, we wanted to know whether a longer window size gives us any increase in performance. From Table 6.4 we can see that increasing the size of window above the size set in our main model causes a decrease in average RMSE performance overall. While there is a slight increase in average RMSE score for a model with a 0.01 second window size, there is also an increase in SD for the same model, which means that the predictions vary more than for our main model (i.e. model highlighted in Table 6.4). Moreover, we noticed once more that the average performance on *cork* is consistently better than the average RMSE performance on other surfaces. Overall, increasing the size of the input window above 0.1

second causes decrease in average performance and decreasing the window size introduces more variation to the predictions.

Table 6.4: *Average RMSE performance based on the context window size. The overall trend is that with longer window length the performances decreases. We highlight the main model that was identified as best using hyperparameter optimisation output.*

Window size	Cork	Wood	Plastic	Paper	Overall
0.01	2.027 ± 2.423	2.052 ± 2.666	2.236 ± 2.544	2.611 ± 3.480	2.223 ± 2.788
0.1	2.399 ± 2.155	2.856 ± 2.732	2.603 ± 2.237	2.653 ± 3.013	2.610 ± 2.526
0.3	2.557 ± 1.872	2.900 ± 2.548	2.676 ± 2.024	3.140 ± 3.045	2.806 ± 2.372
0.5	2.519 ± 1.708	2.817 ± 2.743	2.610 ± 2.004	3.844 ± 3.404	2.915 ± 2.534

6.2 Classification task

In the classification task we use acoustic information to determine the surface type on which the interaction is happening.

6.2.1 Classification metrics

In order to evaluate the performance of the model on the classification task, we chose to use accuracy, precision, recall and F1-score which are all standard metrics for single-label multi-class classification problems. Precision, recall and F1-score are calculated using 'micro' average which means that in order to calculate the score we count the global total number of true positives, false negatives and false positives. The equations used to calculate each of the four metrics are as follows:

$$\begin{aligned} Accuracy &= \frac{TP + TN}{TP + TN + FN + FP}, \\ Precision &= \frac{TP}{TP + FP}, \\ Recall &= \frac{TP}{TP + FN}, \\ \text{F-1} &= 2 * \frac{Precision * Recall}{Precision + Recall}, \end{aligned} \tag{6.3}$$

where TP is the total number of true positives, TN is the total number of true negatives, FN is total number of false negatives and FP is total number of false positives.

6.2.2 How well can we predict the surface type from acoustic information?

Using the same architecture as in Section 6.1.2, which was chosen based on the results of the hyperparameter optimisation we achieved very low performance overall. The confusion matrix is shown in Table 6.6 where we can see that only two surfaces are effectively predicted, i.e. paper and wood. This is reflected in the scores for our chosen metrics shown in Table 6.5 which are all very low and it might be possible to achieve better precision with a model that chooses the surface type at random.

Table 6.5: *Scores for the best model chosen using the result of the hyperparameter optimisation.*

Accuracy	Precision	Recall	F-1 score
0.36	0.16	0.37	0.22

Table 6.6: *Confusion matrix for the best model base on the results of the hyperparameter optimisation.*

	cork	paper	plastic	wood
cork	0	7	0	3
paper	0	10	0	0
plastic	0	6	0	5
wood	0	3	0	4

How much data is required to accurately predict surface type?

Similarly to the regression task, we can see in Table 6.7 that the performance starts to drop with less training data. However, there is an improvement in performance across all four scores for the model trained on 30% of the training dataset size. This is higher than the performance of the model trained on 100% of the training data. It is likely to be caused by the presence of bias in the training dataset which overlaps with a portion of the test dataset. Therefore, the model is likely over-fitted on specific features which are present in the test dataset explaining the better performance. Overall, apart from the over-fitted model on 30% of the training data, the performance decreases with a decreasing size of training dataset.

Table 6.7: *Scores for models trained on different sizes of dataset. We highlight the main model that was identified as best using hyperparameter optimisation output.*

Dataset size	Accuracy	Precision	Recall	F-1 score
100%	0.36	0.16	0.37	0.22
75%	0.31	0.14	0.32	0.18
30%	0.42	0.45	0.42	0.35
5%	0.28	0.08	0.29	0.13

Which elements of the network architecture contribute to the accurate surface type detection?

Although WaveNet architecture reports improved performance for sequence modelling tasks which are in nature regression tasks, we wanted to evaluate what the effect of WaveNet blocks in the architecture is on the classification task. In Table 6.8 we can see that our main model with 4 WaveNet blocks has in fact worse performance compared to models with 8 and 2 WaveNet blocks. It is interesting to see that both adding and removing WaveNet blocks improves performance across all four score metrics. Since the model with 2 WaveNet blocks has slightly higher performances than the model with 8 WaveNet blocks, it could indicate that the classification branch in our architecture performs better when the input is pre-processed less by previous layers (i.e. less modification of the original input). Overall, using 2 WaveNet blocks in the architecture means that the classification branch obtains the data in a less modified form and can therefore learn more about the surface types. Moreover, the improved performance with 8 WaveNet blocks could be caused by the combination of bias and overfitting or the performance would continue to increase with more WaveNet blocks which was, however, not evaluated.

Table 6.8: *Results from ablation study performed on the number of WaveNet blocks of the network. Using other than 4 WaveNet blocks seems to improve the performance on classification task, however the overall performance is poor for all 3 models. We highlight the main model that was identified as best using hyperparameter optimisation output.*

# WaveNet blocks	Accuracy	Precision	Recall	F-1 score
2	0.50	0.42	0.50	0.43
4	0.36	0.16	0.37	0.22
8	0.47	0.36	0.47	0.39

What is the effect of window size on the surface type detection?

While increasing the window size increases the latency, we wanted to evaluate whether longer window sizes could give the architecture more information about the surface and thus improve the performance. We can see in Table 6.9 that by changing the window size from 0.1 second the accuracy only decreases. Similarly, the recall score decreases whenever the window size is different from 0.1 second. This suggests that the best results can be achieved with window size of 0.1 second and therefore increasing or decreasing the window size does not improve the performance for the classification task.

Table 6.9: *Performance of the architecture based on the context window size. Increasing the window size decreases the performance overall. We highlight the main model that was identified as best using hyperparameter optimisation output.*

Window size	Accuracy	Precision	Recall	F-1 score
0.01	0.31	0.31	0.32	0.24
0.1	0.36	0.16	0.37	0.22
0.3	0.18	0.03	0.18	0.06
0.5	0.23	0.43	0.24	0.15

6.3 Summary

Using our main model we were able to predict speed of movement with average RMSE of 2.61 cm/s and SD of 2.52 cm/s. This performance is similar across all surfaces but there is a slight increase in performance on *cork*, which is consistent across all evaluated models. This suggests that the architecture can predict speeds better on highly textured surfaces.

We showed that the architecture can be generalised to some unseen surfaces and while the performance decreases, the produced predictions are still visually very similar to the reference signal.

We investigated what the effects of the context window size, number of layers in the architecture and training dataset size are on the average performance. Our findings suggest that the training dataset size has almost no effect on the performance because even models trained on a small amount of data can make reasonably accurate predictions similar to the ones made by the model trained on the full training dataset. However, models trained on the full training dataset can make predictions for wider range of speeds compared to models trained on less data.

Furthermore, the results indicate that 0.1 seconds is the best context window size for our architecture because the performance decreases when we increase the window size above 0.1 seconds. In addition, the predictions become more varied when we decrease the window size below 0.1 seconds.

Changing the architecture to contain 8 WaveNet blocks showed a slight improvement in performance compared to using smaller architecture, however the improvement was not as

significant as we expected.

In terms of surface type predictions we found that our model performs very poorly and effectively predicts only two surfaces. The performance of our model is essentially random. Changing the architecture, decreasing the training dataset size or changing the context window size has minimal to no effect on the performance which throughout the evaluation did not surpass 0.5 for any of the four chosen metrics (i.e. accuracy, precision, recall, F-1 score). However, changing the architecture resulted in the biggest improvement in performance. More specifically, using only 2 WaveNet blocks improved the accuracy of our main model from 0.36 to 0.5 and nearly doubled the F-1 score (0.22 and 0.43 for the main model and the model with 2 WaveNet blocks, respectively).

6.4 Discussion

The limitation of our approach to the problem of acoustic speed estimation is that our model does not estimate higher speeds (above 15 cm/s) reliably. This limitation could have been caused by the optical sensor saturation, thus using sensor which can capture higher speeds might improve the predictions of the model. Given that we showed that models trained on less data are more imprecise in higher speeds than models trained on more data, we could argue that by adding more data we could achieve good performance in higher speed ranges as well. Moreover, we have showed that if predicting slower speeds is desired, the end-to-end learning does not require a huge labelled dataset as even a model trained on 5% of the data could produce good predictions for slower speeds. This further supports the statement that WaveNet architecture is very successful in sequence modelling tasks.

Surface type detection proved to be a much more difficult problem and there could be several reasons for the low metrics of our model on the classification task. Firstly, our architecture could be incompatible with the classification task and as such was not able to support it. This would mean that WaveNet architecture makes it significantly harder to detect surface type using acoustic information. Secondly, we could argue that we have not provided a large enough dataset for the classification task when using end-to-end learning and that more labelled data would be needed to achieve better performance. Lastly, attaching the contact microphone on the object which is performing the interaction could make the problem of surface type detection significantly more difficult to solve, because the acoustic waves are always propagated through this object which can mask the frequencies needed to correctly detect the surface.

Overall, the effect of acoustic surface type detection on acoustic speed estimation was not as crucial as we expected although there is a small correlation between improving the acoustic surface type detection and slight improvement of the acoustic speed estimation. We can see this correlation when investigating architecture contributions to the performance in both tasks. Using 2 or 8 WaveNet blocks improves surface detection and marginally improves speed estimation as well.

7 Conclusion

We have explored whether end-to-end learning is a viable approach to acoustic surface detection and acoustic speed estimation in the context of two objects in contact when one is moving relative to the other. In order to assess this we have designed a system, collected the total of 40 minutes of recording on 4 surfaces into a dataset, implemented and trained a state-of-the-art WaveNet end-to-end neural network and evaluated several combinations of models to examine how certain changes to the architecture or dataset size affect the results.

We found that the problem of acoustic speed odometry can be solved reasonably precisely for motions of the human hand with our architecture. Our model struggled to predict speeds above 15 cm/s accurately, however this could be due to insufficient training dataset size. There was no significant effect on the overall performance after changing the architecture, reducing the training dataset size or changing the context window size. Moreover, we found that not being able to precisely detect the type of surface does not affect the speed estimations in a significant way and the model is still capable of making accurate speed predictions even when surface predictions are incorrect.

On the other hand, we found that solving the problem of acoustic surface type detection in the same context proved to be much more difficult with our architecture and dataset. Our model performed very poorly regardless of the changes to the architecture, training dataset size or window size. This could be due to incompatibility of the WaveNet architecture with the classification task such as acoustic surface detection or due to an insufficiently large training dataset, which is usually required for end-to-end learning architectures.

To put our work in the context of other research, we have not achieved the same precision of speed estimation as Göksu (2018) in automotive applications, however we have showed that precise acoustic speed estimation in the context of acoustic interaction is possible. In addition, surface type detection in motion has been mostly successful on manufactured surfaces, however detecting even highly textured everyday surfaces proved to be much harder in motion and with our architecture than in a static context.

7.1 Future work

We have already attempted to show that our model can be used with a Kalman filter to improve the precision of an optical pen mouse when the optical flow sensor malfunctions or fails completely. However, due to the limitations of the library used, we have been unable to examine whether the Kalman filter could track the movement of the optical pen mouse solely using predictions from our model. Therefore, changing the implementation of the Kalman filter so that it can work temporarily without the observations from the optical flow sensor could be explored in the future.

Another possible angle we could explore is whether more complex architecture would improve the performance of the system on both tasks. Moreover, we could research whether collecting considerably more data across multiple surfaces helps to generalise the network to all surfaces and increases the range of speed that can be reliably predicted.

Lastly, if we make predictions accurate enough, we could couple the system with existing projects in tangible interaction to offer more granular control based on the speed of the interaction. In addition, we could use the speed and surface type predictions to implement a new musical instrument which would synthesise sound based on the surface texture and the speed of movement on the surface.

A Appendices

A.1 Implemented architecture code

A.2 Training times

Table A.1: *Training times for all evaluated models. The first model (i.e. 100% training data model) was trained on 100% of training data with 4 WaveNet blocks and window size of 0.1. The following three models were trained on portion of the training data. For the next two models (i.e. 2 and 8 WaveNet blocks) we kept all parameters the same except for the number of WaveNet blocks. In addition, last three models have 4 WaveNet blocks and we changed the size of the context window.*

Model	Training time (seconds)
100% training data	10834
75% training data	8427
30% training data	3191
5% training data	766
2 WaveNet blocks	5609
8 WaveNet blocks	20284
window of 0.01	39243
window of 0.3	7773
window of 0.5	5871

```
def wavenet_block(units, kernel_size, padding, dilation_rate):
   def f(residual):
      tanh_out = Conv1D(units, kernel_size=kernel_size, padding=padding,
                  activation="tanh", dilation_rate=dilation_rate,
                  kernel_initializer=glorot_uniform(seed=1))(residual)

      sigmoid_out = Conv1D(units, kernel_size=kernel_size, padding=padding,
                  activation="sigmoid", dilation_rate=dilation_rate,
                   kernel_initializer=glorot_uniform(seed=1))(residual)

      mul_out = Multiply()([tanh_out, sigmoid_out])

      skip_out = Conv1D(1,1,activation="relu", padding=padding,
                  kernel_initializer=glorot_uniform(seed=1))(mul_out)

      out = Add()([skip_out,residual])
      return out, skip_out
   return f

input = Input(shape=(int(params["sample_rate"]*params["window_len"]),1))
l1a, l1b = wavenet_block(20,5,"causal", 2)(input)
l2a, l2b = wavenet_block(20,5,"causal", 4)(l1a)
l3a, l3b = wavenet_block(20,5,"causal", 8)(l2a)
l4a, l4b = wavenet_block(20,5,"causal", 16)(l3a)
l5 = Add()([l1b,l2b,l3b,l4b])
l6 = Lambda(keras.activations.relu)(l5)
l7 = Conv1D(1,1, activation="relu",
    kernel_initializer=glorot_uniform(seed=1))(l6)
l8 = Conv1D(1,1, kernel_initializer=glorot_uniform(seed=1))(l7)
l9 = Flatten()(l8)
```

Listing A.1: *Common part of the network with WaveNet blocks implemented in Keras*

```
l10 = Dense(30, activation="tanh", kernel_initializer=glorot_uniform(seed=1))(l9)
l11 = Reshape((30,1))(l10)
l12 = Conv1D(13,5, activation="relu",
    kernel_initializer=glorot_uniform(seed=1))(l11)
l13 = Conv1D(7,3, activation="relu",
    kernel_initializer=glorot_uniform(seed=1))(l12)
l14 = Flatten()(l13)
l15 = Dense(int(1000*params["window_len"]), activation="softplus",
    kernel_initializer=glorot_uniform(seed=1))(l14)
l16 = Reshape((int(1000*params["window_len"]),1), name="speed_output")(l15)
```

Listing A.2: *Code for regression branch implemented in Keras*

```
l17 = Dense(70, activation="selu", kernel_initializer=glorot_uniform(seed=1))(l9)
l18 = Dense(30, activation="selu",
    kernel_initializer=glorot_uniform(seed=1))(l17)
l19 = Dense(4, activation="softmax", kernel_initializer=glorot_uniform(seed=1),
    name="surface_output")(l18)
```

Listing A.3: *Code for classification branch implemented in Keras*

7 Bibliography

B. Amento, W. Hill, and L. Terveen. The Sound of One Hand: A Wrist-mounted Bio-acoustic Fingertip Gesture Interface. *CHI 2002 Ext. Abstracts*, 2002.

F. Antonacci, L. Gerosa, A. Sarti, S. Tubaro, and G. Valenzise. Sound-based classification of objects using a robust fingerprinting approach. In *2007 15th European Signal Processing Conference*, pages 2321–2325, Sept. 2007.

F. Antonacci, G. Prandi, G. Bernasconi, R. Galli, and A. Sarti. Audio-based object recognition system for tangible acoustic interfaces. In *2009 IEEE International Workshop on Haptic Audio visual Environments and Games*. IEEE, nov 2009. doi: 10.1109/have.2009.5356138. URL https://doi.org/10.1109/have.2009.5356138.

S. Bai, J. Z. Kolter, and V. Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *arXiv:1803.01271 [cs]*, Mar. 2018. URL http://arxiv.org/abs/1803.01271. arXiv: 1803.01271.

I. Bhattacharyya. Piezoelectric and piezo sensors, 2014. URL https://www.slideshare.net/IndranilBhattacharyya/piezoelectric-and-piezo-sensors.

V. Cevher, R. Chellappa, and J. McClellan. Vehicle Speed Estimation Using Acoustic Wave Patterns. *IEEE Transactions on Signal Processing*, 57(1):30–47, Jan. 2009. ISSN 1053-587X, 1941-0476. doi: 10.1109/TSP.2008.2005750. URL http://ieeexplore.ieee.org/document/4625947/.

J. Gehring, M. Auli, D. Grangier, and Y. N. Dauphin. A Convolutional Encoder Model for Neural Machine Translation. *arXiv:1611.02344 [cs]*, Nov. 2016. URL http://arxiv.org/abs/1611.02344. arXiv: 1611.02344.

J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional Sequence to Sequence Learning. *arXiv:1705.03122 [cs]*, May 2017. URL http://arxiv.org/abs/1705.03122. arXiv: 1705.03122.

T. Glasmachers. Limits of End-to-End Learning. *Proceedings of Machine Learning Research 77:17âĂŞ32, 2017*, page 16, 2017.

M. Goel, B. Lee, M. T. Islam Aumi, S. Patel, G. Borriello, S. Hibino, and B. Begole. SurfaceLink: using inertial and acoustic sensing to enable multi-device interaction on a surface. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI '14*, pages 1387–1396, Toronto, Ontario, Canada, 2014. ACM Press. ISBN 978-1-4503-2473-1. doi: 10.1145/2556288.2557120. URL http://dl.acm.org/citation.cfm?doid=2556288.2557120.

H. Göksu. Vehicle speed measurement by on-board acoustic signal processing. *Measurement and Control*, 51(5-6):138–149, June 2018. ISSN 0020-2940. doi: 10.1177/0020294018773777. URL https://doi.org/10.1177/0020294018773777.

C. Harrison and S. E. Hudson. Scratch input: creating large, inexpensive, unpowered and mobile finger input surfaces. In *Proceedings of the 21st annual ACM symposium on User interface software and technology - UIST '08*, page 205, Monterey, CA, USA, 2008. ACM Press. ISBN 978-1-59593-975-3. doi: 10.1145/1449715.1449747. URL http://portal.acm.org/citation.cfm?doid=1449715.1449747.

C. Harrison, D. Tan, and D. Morris. Skinput: Appropriating the body as an input surface. In *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*. ACM Press, 2010. doi: 10.1145/1753326.1753394. URL https://doi.org/10.1145/1753326.1753394.

C. Harrison, J. Schwarz, and S. E. Hudson. TapSense: enhancing finger interaction on touch surfaces. In *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*, page 627, Santa Barbara, California, USA, 2011. ACM Press. ISBN 978-1-4503-0716-1. doi: 10.1145/2047196.2047279. URL http://dl.acm.org/citation.cfm?doid=2047196.2047279.

C. Harrison, R. Xiao, and S. Hudson. Acoustic barcodes: passive, durable and inexpensive notched identification tags. In *Proceedings of the 25th annual ACM symposium on User interface software and technology - UIST '12*, page 563, Cambridge, Massachusetts, USA, 2012. ACM Press. ISBN 978-1-4503-1580-7. doi: 10.1145/2380116.2380187. URL http://dl.acm.org/citation.cfm?doid=2380116.2380187.

H. Ishii, C. Wisneski, J. Orbanes, B. Chun, and J. Paradiso. Pingpongplus: Design of an athletic-tangible interface for computer-supported cooperative play. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, pages 394–401, New York, NY, USA, 1999. ACM. ISBN 0-201-48559-1. doi: 10.1145/302979.303115. URL http://doi.acm.org/10.1145/302979.303115.

N. Kalchbrenner, L. Espeholt, K. Simonyan, A. v. d. Oord, A. Graves, and K. Kavukcuoglu. Neural Machine Translation in Linear Time. *arXiv:1610.10099 [cs]*, Oct. 2016. URL http://arxiv.org/abs/1610.10099. arXiv: 1610.10099.

P. S. Kalekar. Time series Forecasting using Holt-Winters Exponential Smoothing. *Time Series Forecasting Using Holt-Winters Exponential Smoothing*, page 13, Jan. 2004.

K. Kunze and P. Lukowicz. Symbolic object localization through active sampling of acceleration and sound signatures. In *UbiComp 2007: Ubiquitous Computing*, pages 163–180. Springer Berlin Heidelberg, Sept. 2007. doi: 10.1007/978-3-540-74853-3_10. URL https://doi.org/10.1007/978-3-540-74853-3_10.

R. Murray-Smith, J. Williamson, S. Hughes, and T. Quaade. Stane: synthesized surfaces for tactile input. In *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*, page 1299, Florence, Italy, 2008. ACM Press. ISBN 978-1-60558-011-1. doi: 10.1145/1357054.1357257. URL http://portal.acm.org/citation.cfm?doid=1357054.1357257.

A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. *arXiv:1609.03499 [cs]*, Sept. 2016a. URL http://arxiv.org/abs/1609.03499. arXiv: 1609.03499.

A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel Recurrent Neural Networks. *arXiv:1601.06759 [cs]*, Jan. 2016b. URL http://arxiv.org/abs/1601.06759. arXiv: 1601.06759.

J. A. Paradiso and C. K. Leo. Tracking and characterizing knocks atop large interactive displays. *Sensor Review*, 25(2):134–143, jun 2005. doi: 10.1108/02602280510585727. URL https://doi.org/10.1108/02602280510585727.

S. Robinson, N. Rajput, M. Jones, A. Jain, S. Sahay, and A. Nanavati. TapBack: Towards richer mobile interfaces in impoverished contexts. In *Proceedings of the 2011 annual conference on Human factors in computing systems - CHI '11*. ACM Press, 2011. doi: 10.1145/1978942.1979345. URL https://doi.org/10.1145/1978942.1979345.

W. Rolshofen, P. Dietz, G. SchÃďfer, and R. Kruk. TAI-CHI: Tangible Acoustic Interfaces for Computer-Human Interaction. *FORTSCHRITTE DER AKUSTIK 31.2*, page 773, 2005.

V. Savage, A. Head, B. Hartmann, D. B. Goldman, G. Mysore, and W. Li. Lamello: Passive Acoustic Sensing for Tangible Input Components. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*, pages 1277–1280, Seoul, Republic of Korea, 2015. ACM Press. ISBN 978-1-4503-3145-6. doi: 10.1145/2702123.2702207. URL http://dl.acm.org/citation.cfm?doid=2702123.2702207.

A. Saxena and A. Ng. Learning sound location from a single microphone. In *2009 IEEE International Conference on Robotics and Automation*, pages 1737–1742, Kobe, May 2009. IEEE. ISBN 978-1-4244-2788-8. doi: 10.1109/ROBOT.2009.5152861. URL http://ieeexplore.ieee.org/document/5152861/.

M. A. Schallig, A. J. Meijer, P. S. Viet, and J. Tiesinga. Electrical surface treatment device with an acoustic surface type detector, June 2000. URL https://patents.google.com/patent/US6076227A/en.

TE-Connectivity. MINISENSE 100 NM (No Mass) Vibration Sensor, 2018. URL https://www.te.com/commerce/DocumentDelivery/DDEController?Action=showdoc&DocId=Data+Sheet%7FPiezo_Minisense_100-NM%7FA1%7Fpdf%7FEnglish%7FENG_DS_Piezo_Minisense_100-NM_A1.pdf%7FCAT-PFS0011.