Universiteit Leiden

Master Computer Science

Enginetron: realtime car exhaust note synthesis using on-board diagnostics through Text-to-Speech networks

Name:	Ing. T.P. Gmelig Meyling
Student ID:	s2017881
Date:	05/12/2022
Specialisation:	Advanced Data Analytics
Supervisor:	Dr. E.M. Bakker
Supervisor:	Prof.dr. M.S.K. Lew

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Acknowledgements

First of all, I would like to thank my supervisors Erwin and Michael for their feedback and support during the project. Being able to have consistent meetings even during the pandemic has greatly benefited this work. I would also like to thank Erwin for being able to use the LML GPU training server. I want to thank my friends for hearing me out countless times on developments on this project and help me get useful insights. I would like to thank my wife, Cio, for her patience and support and keeping me motivated to finish the master's. Finally I would like to thank my parents for supporting me and for always encouraging me to follow my dreams.

Abstract

Electric vehicles lack an auditory footprint as opposed to their Internal Combustion Engine counterparts. Even though in urban environments, engines are typically considered an annoyance and their sounds should typically be minimal, having some auditory footprint has its safety benefits. Research shows that the sound of a combustion engine is easily detected as a moving object. Hence the synthesis of a realistic engine sound can be an important safety feature.

This work focusses on synthesizing realtime car exhaust notes using an on-board diagnostics (OBD)-based language through Text-to-Speech (TTS) networks. A method is presented to synthesize OBD transcribed data through FMOD with AudioMotors. A large dataset of synthetic exhaust notes with corresponding OBD data is constructed by using synthetic OBD data to synthesize exhausts notes using FMOD with AudioMotors (using our FMOD-Enginesynth). The resulting dataset is named the FMOD-OBD-Dataset.

An Extraction, Transformation and Load (ETL) script is introduced to transcribe the OBD data of engine expressions such that the dataset can be used as training set for a novel modified TTS engine. This ETL and modified TTS engine form Enginetron. The performance of Enginetron is evaluated using a Fréchet Audio Distance (FAD) score.

Contents

1. Introduction

Even though the first car ever produced was in fact an electric car, most production cars at the time of writing utilize an Internal Combustion Engine. With the exhaustion of fossil fuels we are transitioning towards alternative energy sources to power our cars. Many car manufacturers start to produce electrical vehicles. The problem however is that electrical cars lack the sound that Internal Combustion Engines make. This is not only important for product experience, it also plays an important factor in safety.

Synthesis of exhaust noise is an active field both in research and in a lot of commercial companies. It has applications in many areas. Some research focusses on adding audio to cars with little to no auditory footprint. There are several motivations to do this such as product perception (sportiness, perceived product quality, etc) [1] as well as safety (detectability of a vehicle, perception of speed of a vehicle etc) [2, 3, 4]. Other applications are in multimedia contexts such as cinematography or the gaming industry. Another application would be product exploration. Using a Generative Adversarial Network (GAN) network on a dataset comprising car exhaust notes could be used to explore a compact latent space of exhaust notes [5], which allows a sound artist to make adjustments to latent variables to explore the kind of sound they are looking for. Finally exhaust note metrics, which are also researched in this work, also have applications in detecting engine malfunctions of cars that do utilize an Internal Combustion Engine [6].

This work focusses on synthesizing exhaust notes using end-to-end Text-to-Speech (TTS) networks, such as Tacotron. End-to-end training requires good evaluation metrics. Preliminary research [5] showed that good evaluation metrics for exhaust notes synthesis are lacking. Spectrograms do reflect if harmonics are preserved, but this is a manual observation and hard to quantify. An important part of this work is the search and selection of a better evaluation metric, that could be used for end-to-end training neural network (NN) exhaust synthesizers. Similarly these metrics could be used to train an exhaust note synthesizer in an adversarial setting, which are known to be highly effective in synthesizing realistic sounds with low resources.

This work will cover the following important aspects:

- Selection of a metric. train a network to evaluate synthesized exhaust notes quality.

  Initially considered to be part of this work, creating and training a network to deduce revolutions per minute (RPM), gear or potentially even speed (more difficult) from a sound, is a fruitful avenue for future research.

- Improve existing attention mechanisms in TTS NNs, i.e. improve the attention and decoder part of the network such that it will work better on exhaust notes sound.

- Change the Encoder such that it can take raw diagnostic data such as RPM through preprocessing with a custom language. This will allow for better learning of the semantics by the network.

- Address the problem for car exhaust note synthesis of limited data. There is even less data available that will be annotated with on-board diagnostics data. This work will cover the creation of a large exhaust note dataset with matching On-Board Diagnostics (OBD).

  Initially considered to be part of this work, voice cloning. It would be interesting to apply voice cloning techniques that are presented in text-to-speech papers to the car exhaust note synthesis problem. Ideally one could train on a big data set with a lot of exhaust notes (such as the one introduced in this paper) and then would need a much smaller exhaust note dataset with a new car for training. It is still considered a promising direction for future research.

Contributions

The main contributions of this work are the following:

- **Metrics.** A study of quantitative and qualitative evaluation metrics that can be used to evaluate the quality of synthesized exhaust notes is presented. Qualitative evaluation metrics that are covered are MOS scores, and preserved harmonics. Quantitative evaluation metrics that are covered are F_0 evaluation, MOSNet, Frequency Analysis using GoldWave®, and Fréchet Audio Distance.
- **FMOD-OBD Dataset.** This work provides the largest OBD with matching audio dataset named *"FMOD-OBD Dataset"* available to train data-hungry neural network based networks to synthesize exhaust notes. A script is also provided to generate such datasets as part of this contribution.
- **FMOD-Enginesynth.** A script that can synthesize OBD data using the FMOD and AudioMotors tool.
- **Enginetron.** This work provides a network named *"Enginetron"* that can be trained to to synthesize exhaust notes. A pre-trained network, trained on the aforementioned dataset, and hyperparameters such as learning rates are also provided.

Structure

The remainder of this thesis is structured as follows. Related work is covered in Chapter 2. The fundamentals are described in Chapter 3. The methodology for this work is given in Chapter 4. The dataset and creation presented in Chapter 5. The experiments setup is given in Chapter 6. Results are presented in Chapter 7. Finally this thesis is concluded in Chapter 8.

2. Related work

In this section we discuss the related research on exhaust note synthesis and on speech synthesis technologies. In this thesis we adapt the latter for exhaust note synthesis.

2.1 Exhaust notes synthesis

Most works that deal with exhaust note synthesis work with sample based methods. Typically a few samples based on RPM and load are provided from which, to obtain different samples such as low rpm low load, low rpm high load etc. The provided samples are stitched while using simple parameters and have their pitch shifted to match the RPM of the car.

State of the art car exhaust notes synthesizers [7] typically use a method called granular synthesis. Granular synthesis [8] is a form of (additive) synthesis which involves generating thousands of very short sonic grains to form larger acoustic events.

Lesound AudioMotors[1] is a proprietary exhaust note synthesizer that is used in the gaming industry. AudioMotors also uses this so-called granular synthesis method. There is a free trial available on their website. They also provide a link to a databank on their website having short recordings of cars cycling accelerating. They integrate with a tool called FMOD[2], which is a tool for making adaptive sound for games. In this work FMOD with AudioMotors is used. A custom script is made such that FMOD with AudioMotors can synthesize on-board diagnostics data.

2.2 Speech synthesis

In this section the different speech synthesis methods and current research directions that are currently active are discussed. MOS-scores are shown for US English that are presented in the original papers. They might use the exact same dataset, have varying preprocessing or different benchmarking methods and therefore those MOS-scores are not suitable for side-by-side comparison. However it should still be able to give a broader overview of the progression of text-to-speech research and different methods.

Concatenative methods

A well-known speech synthesis methods is concatenative speech synthesis [9]. In this method you have a large dataset of segments and their recorded speech. Segments could be sentences, words, syllables etc. To form sentences different segment recordings would be concatenated. The disadvantage of this method is that to synthesize the audio you will need a dataset of all the segments possible audio. Recent concatenation methods [10] achieve a Mean Opinion Score (MOS) score of about 3.82 ± 0.08 for real speech, which is behind the current state of the art methods that typically have a MOS higher than 4.5.

[1] https://lesound.io/audiomotors-wwise/

[2] https://www.fmod.com/

Statistical parametric methods

Another approach is statistical parametric speech synthesis [11]. In statistical parametric speech synthesis parameters that characterize the audio sample are extracted such as the frequency spectrum, fundamental frequency and duration. The parameters are estimated using a trained statistical model.

An advantage of statistical parametric speech synthesis is that there is no need for a big database with all speech samples for synthesis. Also parametric speech synthesis is more flexible in (voice) characteristics compared to concatenation methods. Recent statistical parametric methods [12] achieve MOS scores of 3.69 ± 0.119 on US English, which is worse than the concatenated speech synthesis methods.

Deep learning methods

During the last decade deep learning has been applied to the text-to-speech problem. With deep learning, one can effectively capture the hidden internal structures of data and use more powerful modeling capabilities to characterize the data. In this section we will dive deeper in several different deep learning text-to-speech methods.

Autoregressive models

WaveNet [13] is a model that synthesizes raw waveform of the audio signal instead of acoustic features. WaveNet is a so-called autoregressive model in which each sample statistically depends on the previous ones. To build the autoregressive model a fully convolutional neural network was used inspired by PixelCNN where PixelCNN is a network which is used to generate natural images. PixelCNN is a network that auto-completes an occluded image according to its content by generating pixel predictions from a pixel's nearest neighbors. WaveNet operates on one-dimensional time-series as opposed to PixelCNN two-dimensional RGB pictures. WaveNet achieves a MOS score of 4.21 on US English.

Statistical parametric methods

Many diffferent Deep Neural Network (DNN) architectures have been studied for TTS. Early models that try to achieve end-to-end training started with a so-called encoding, decoder architecture. Typically this architecture had an encoder that would convert characters, words, phonemes, etc, into an intermediate representation. Further successes were obtained by adding attention mechanisms to the network. Finally, for decoding the network would output (mel)-spectrograms or wav waveform audio.

Tacotron [14] is an end-to-end system that incorporates many of these techniques. It is a sequence to sequence model that has an encoder-decoder architecture with an attention mechanism. The encoder extracts sequential representations of text. Attention is used to pass the representation to the encoder. The decoder module generates a linear-scale spectrogram after which a Griffin-Lin vodoer is used to get the raw waveform. Tacotron achieved a MOS of 3.82 on an US English evaluation set.

Tacotron has been evolved into Tacotron2 [15]. Tacotron 2 is an improved version with a simplified architecture compared to the original Tacotron. It uses local sensitive attention instead of the original additive attention. The decoder is now an autoregressive Recurrent Neural Network (RNN). Instead of Griffin-Lin it uses WaveNet to convert the spectrograms into raw waveform. Tacotron 2 received a MOS score of 4.53 which is close to the ground truth for recorded speech which is 4.58. Note that for evaluation Tacotron 1 & 2 both are trained and evaluated on the a

internal normalized dataset. As part of the normalization numbers are written out (i.e. “16” is “sixteen”).

Generative Adversarial Networks

Generative adversarial networks (GANs) provide a way to learn deep representations without extensively annotated training data. They achieve this through deriving backpropagation signals through a competitive process involving a pair of networks the generator and the discriminator. The representations that can be learned by GANs may be used in a variety of applications, including image synthesis, semantic image editing, style transfer, super-resolution and classification.

GAN Singing Networks

An interesting model is the Korean Singing Voice Synthesis System (KSVSS) [16]. This network takes three inputs. The text, audio and midi files. The midi files contain the fundamental frequency for the singing and how long it should hold. This model has three different encoders. A text encoder, a mel encoder and a pitch encoder. The network has a so-called super-resolution network that upsamples the generated mel-spectrogram into a linear spectrogram. It receives a MOS-score of 3.07 for singing. It was better at maintaining the pitch compared to other methods. Obviously the singing task is more difficult than standard text-to-speech, and also typically there is less training data available.

In this work we will use Tacotron for our implementation of Enginetron.

2.3 Exhaust notes datasets

In preliminary research [5] no large public available dataset was found comprising car exhaust notes audio annotated with raw OBD-data. It is known that for training text-to-speech networks large amounts of data are required to synthesize comprehensible speech. In this earlier work audio was recorded using a microphone and an OBD dongle. The car was driven for about 3.5 hours and an ETL script was used to convert this to training data. When this was used to train Text-to-Speech networks it was found that it was insufficient data to synthesize good exhaust notes. The main problem was that because the car was driving stationary at approximately the speed limit for long times, the dataset showed little variation. This was later tackled by modifying the ETL script to only select samples with variations (such as when the OBD data showed accelerating or deceleration). Using this ETL script did result in a training dataset with more variations, but also in an even smaller training dataset. To tackle the problem of small and little variation datasets, this work presents a novel “FMOD-OBD Dataset”, a large synthesized training dataset that is synthesized using a custom script through FMOD and AudioMotors. A drawback of this method when training on the FMOD-OBD Dataset is that Enginetron’s performance is bound to the performance of FMOD. Any flaws that will be in FMOD will likely also be learnt in the Enginetron model. Enginetron will logically not be able to outperform the FMOD synthesizer. In this work this is justified because there is no better data available. Recording a sports car with very high variation in speed and RPMs for tens of hours was not realistically possible within the context of this thesis.

The hypothesis of this work is that training Enginetron on the FMOD-OBD Dataset can synthesize realistic engine sounds. It can also prove previous hypotheses (such as the hypothesis

that with more and more varying data Enginetron will perform better) right. If Enginetron can be used to synthesize realistic exhaust notes with the FMOD-OBD Dataset, we believe exhaust note synthesis through TTS networks is an interesting and fruitful avenue for future research.

3. Fundamentals

This chapter will cover the fundamentals for this work. This chapter covers on-board diagnostics, and how they are captured. Next "mel"-spectrograms are covered which are often used as intermediate inputs and outputs in neural networks. Raw audio waveforms are covered which typically are part of the training input and the output of a Text-to-Speech network. Finally, qualitative and quantitative evaluation metrics are described. These metrics are used in training and to evaluate the quality of synthesizers in this work.

3.1 On-board diagnostics

All production cars since 2001 (2004 for diesels) in Europe are equipped with on-board diagnostics through an OBD-II port. With an OBD dongle diagnostic data such as fault codes can be read from the car. Although it depends on the vehicle manufacturer and make, most cars allow reading the car speed, engine RPM, absolute throttle position, and engine load in realtime through this OBD port.

There are many consumer based products that show readings via an OBD-dongle connected via either bluetooth or cellular networks such as the TomTom Curfer or Carly Connected Car, which have a focus on reading OBD-II fault codes and coding certain features, such as folding mirror on lock, like OBDEleven.

Preliminary research [5] presented a script that can be used for recording OBD data and exhaust audio in realtime to record OBD with audio datasets. In that work OBD is stored in CSV format with the following headers: RPM, Speed, Load, Throttle and Time. In Table 1 an example of the raw OBD data is listed. OBD data is typically stored in intervals

RPM [0, 8000]	**Speed (km/h)**	**Load ([0,100])**	**Throttle ([0,100])**	**Time (ms)**
1631	20	40	35	0
1782	22	38	30	100
1835	24	37	28	200
1798	26	37	27	300

Table 1: Example of raw recorded OBD data (before any preprocessing).

This work does not use the script to record audio from a real car. Instead OBD-data is generated and synthesized in this work. Generated OBD data is stored in a similar fashion as the preliminary research [5].

3.2 Mel spectrograms

A spectrogram is a visual representation of the specrum of frequencies of a signal as it varies over time. In Figure 1 you see a spectrogram of an original engine sample. A spectrogram has

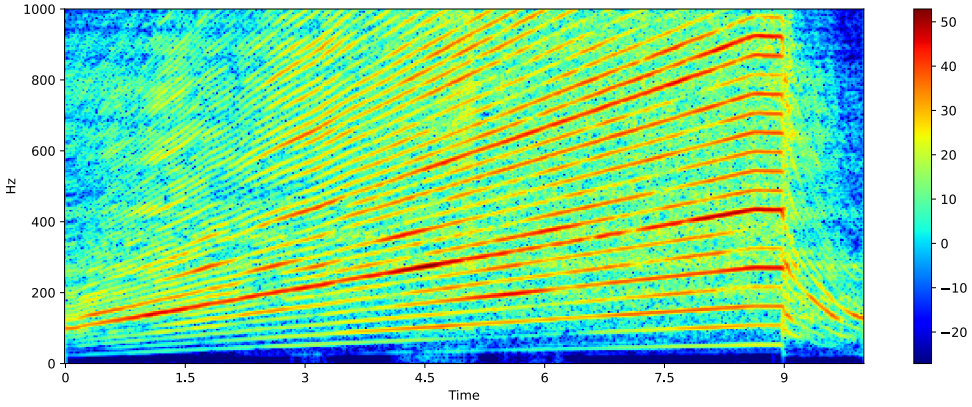

Figure 1: Mel spectrogram of the original engine sample. Frequencies are limited from 0 to 1000 Hz.

frequencies (in Hz) on the y-axis. On the x-axis the spectrogram has time. The color axis is the amplitude in dB. A mel-spectrogram is a special kind of spectrogram where the frequencies are on the so-called mel-scale. The mel-scale is not a linear nor a logarithmic scale, but a scale that is inspired by the humar perception of speech and sound. On the mel scale is a perceptual scale of pitches on which each pitch is determined to be equally 'far' from each other judged by human listeners. Those mel-spectrograms are often an intermediate form in neural networks that do audio, or in particular, speech synthesis. We use librosa [17] to implement the function to calculate the mel-spectrograms (see subsection 3.4).

3.3 Waveform

A waveform sample is a a sequence of samples, where each sample is a real number denoting the amplitude of the signal. The sample rate of 48 kHz means that the raw audio has been sampled with 48.000 samples per second. The output of a text-to-speech network consists of such waveform samples.

Our dataset is sampled with a sample rate of 22050 Hz at 16-bits sample depth.

3.4 Librosa

Librosa[1] [17] is a python package for music and audio analysis. Librosa is used in this work in the ETL scripts to generate shorter samples with matching OBD from the longer version. Librosa also has tools to generate mel-spectrograms and comes with many analysis tools. For instance in preliminary experiments the fundamental frequency estimation algorithms *yin* [18] and *pyin* [19] from librosa were used. Librosa is also used in the implementation of Tacotron to generate samples from spectrograms.

[1] `https://librosa.org/doc/main/index.html`

3.5 Qualitative evaluation metrics

A common metric for speech synthesis is the so-called Mean Opinion Score (MOS). The International Telecommunication Union (ITU) has defined the opinion score as the "value on a predefined scale that a subject assigns to his opinion of the performance of a system" [20]. The mean opinion score (MOS) is the average of these scores across subjects. For a MOS subjects are asked to rate the naturalness of the stimuli in a 5-point Likert [21] scale score where 1 is bad and 5 is excellent. This method is used in many Text-to-Speech papers as an evaluation metric for the perceived quality of synthesized speech.

Another qualitative evaluation metric is the preservation of *harmonics*. The harmonics are integer multiples of the fundamental frequency F_0. In the case of an engine you have three different frequencies which are integral multiples. You have the explosion frequency F_{ex}, the engine revolution frequency F_{en} and the individual cylinder firing frequency F_c. From the formulas in Equation 3.1, 3.2, and 3.3 it follows that those frequencies are integral multiples. In a four stroke engine the piston of each cylinder completes four strokes within a single thermodynamic cycle comprising of intake, compression, power, and exhaust. During a complete cycle of a four-stroke engine, the crank shaft turns two times. Independently from the number of cylinders and their disposition, an engine noise is essentially harmonic due to the sequential explosions occurring in the combustion chamber of each cylinder. The explosion frequency F_{ex} for a four-stroke engine is given by

$$F_{ex} = \frac{N_c}{2}\frac{\text{rpm}}{60} \tag{3.1}$$

where N_c is the number of cylinders in the engine and rpm is the number of revolutions per minute. Two successive explosions do not occur in the same cylinder and thus are not identical. Furthermore, two successive engine revolutions do not contain the explosions of the same cylinders. On that account, two other frequencies F_{en} and F_c and their harmonics enrich the spectrum of an engine sound:

$$F_{en} = \frac{\text{rpm}}{60} \tag{3.2}$$

$$F_c = \frac{1}{2}\frac{\text{rpm}}{60} \tag{3.3}$$

where F_{en} is the frequency corresponding to the engine revolution and F_c is the fundamental frequency corresponding to a complete cycle of the engine (two revolutions). The harmonics are clearly visible in mel-spectrogram see Figure 1. A good engine synthesizer would preserve the harmonics, (i.e. the F_{ex}, F_{en} and F_c are still multiples of each other and do not deviate from each other. However, manual observation of the fundamental frequency and harmonics in the mel-spectrogram is needed to confirm if the harmonics are preserved in synthesized audio. Manual observation is needed because in exhaust notes there is a lot of noise, and there are more harmonics which are integral multiplications of the three frequencies earlier mentioned. You can see in Figure 1 there are a lot more frequencies. Because the engine noise has a wide range of frequencies, it is very hard to consistently find the fundamental frequency F_{ex} compared to human voice where the range of frequencies is much smaller.

An attempt was made to systematically use the F_0 of the exhaust note. Extracting it could be used as an evaluation metric as well as be used during training as a separate encoder. Unfortunately all the F_0 estimation algorithms that were considered (e.g. RMSE of F_0) did not give a meaningful F_0 estimation (even on the original signal). Since the nature of the engine you have to be able to find the fundamental frequency of the whole range of RPMs the engine offers. The challenge is that within those ranges there is already some harmonics causing the F_0 estimation algorithms to become instable. Since the F_0 could not reliable be obtained, the quantitative measures like root mean square of the fundamental frequency could also not reliably be obtained. Therefore other F_0 based metrics are not considered in this section.

3.6 Quantitative evaluation metrics

There are several different quantitative evaluation metrics that can be used for exhaust note synthesis. This paper covers F_0 evaluation, MOSNet, Frequency Analysis using GoldWave® (alpha function as shown in Section 3.7), and Fréchet Audio Distance score. For this work MOSNet was attempted but it did not yield meaningful results. In the end only F_0, Frequency Analysis using GoldWave®, and Fréchet Audio Distance scores were used. The scores for these metrics are covered in the Chapter 7 Results.

F_0 evaluation

In Korean Voice Singing [16] a quantitative absolute metric is presented that can be used to determine the quality of singing. The metric they use is extracting the F_0 sequence. They then converted it into a pitch sequence, and compared it to the input pitch sequence. The higher the similarity between the two sequences, the more the network generates a singing that reflects the input condition. The precision, recall and f-score is then calculated of the generated sequence by frame-wise comparison. This network has not been used directly and none experiments were made with this network as no code or pre-trained network was available. The paper is still included in this chapter as it has relevant insights as it is a melody based network and inevitably this work taken inspiration from it in some shape or form. In this work the F_0 sequence part of the WORLD vocoder [22] is used for extraction. There is a Python version of WORLD available.

For F_0 extraction the Root Mean Square Error between the predicted and expected outcome can be used to evaluate the performance of F_0 extraction. The RMSE_{F_0} is computed as follows where $\hat{Y}_i$ is the predicted F_0 and Y_i is the expected F_0 at i:

$$\text{RMSE}_{F_0} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2} \tag{3.4}$$

MOSNet

MOSNet [23] is a quantitative absolute metric for end-to-end speech naturalness[2]. That means it does not need a reference signal. It is based on deep learning. MOSNet is trained on speech and when and for an input it will give a MOSScore is (0-5). The advantage of MOSNet as opposed to real qualitative MOS-scores is that it can be fully automated and used as a quick feedback loop in experiments, or even as a metric for training in an *adversarial setting* [24]. In this work MOSNet is not used. There was done some small experiments using MOSNet with a pre-trained model. The pre-trained model did not perform well in early experiments. This because MOSNet is pre-trained on a large set of speech and MOS-scores. As part of future work a set of exhaust notes and their respective MOS-scores could be created to train MOSNet for exhaust notes.

[2] `https://github.com/aliutkus/speechmetrics` is a wrapper around many objective speech metrics including MOSNet.

3.7 Frequency Analysis using GoldWave®

GoldWave can be used to do frequency analysis on audio. What particularly proved useful for this work is the GoldWave Control Window, which can be configured to show a spectogram and the spectrum. In Figure 2 you see the GoldWave control user interface. Spectrum typically shows the spectrum. It also shows a red peak line which displays the highest magnitude since playback started. Finally a third yellow line is shown which is the so-called alpha line. It seems that the average line (also called Alpha line in GoldWave) is a reliable method to find the F_0 of an exhaust note sample. The alpha line function is shown in Equation 3.5. Even though GoldWave is scriptable it is not easy to automate the frequency analysis through GoldWave. The alpha function could easily be implemented in Python however. In this work GoldWave is used as an inspection tool to verify metrics. Also the alpha function implementation was used in python for F_0 experiments.

$$M = alpha * M_{previous} + (1 - alpha) * M_{new} \tag{3.5}$$

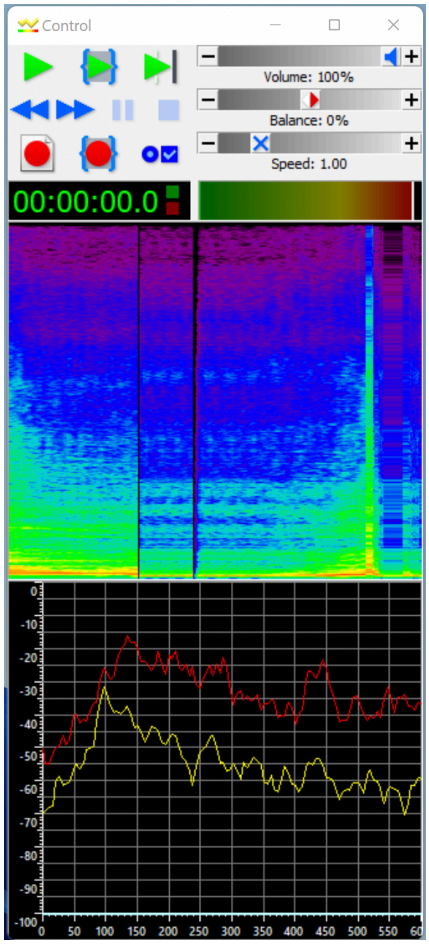

Figure 2: GoldWave control window as you would see after playing back an exhaust note audio file.

3.8 Fréchet Audio Distance (FAD)

Fréchet Audio Distance [25] is another measure that we can use to objectively evaluate quality of synthesized audio. FAD is used in recent papers that deal with evaluating quality of synthesized audio [26]. The idea of FAD is to measure the closeness of the data distribution of the real data versus the data distribution of the generated data in a certain embedded space.

FAD compares embedding statistics generated on the whole evaluation set with embedding statistics generated on a large set of training data. FAD uses embeddings generated by the VGGish model. It then computes multivariate Gaussians on both the evaluation set embeddings $\mathcal{N}_e(\mu_e, \Sigma_e)$ and the background embeddings $\mathcal{N}_b(\mu_b, \Sigma_b)$ as shown in Equation 3.6 where tr is the trace of a matrix. The background embeddings are then referred to as a model's FAD score

The lower a FAD score the better. In this project the FAD distance is used. One advantage of this metric is that it is a relative metric. This means that it compares the data distributions in the synthesized files to those in the original audio files. That means synthesized exhaust notes will be compared to original exhaust notes. In early experiments it has shown that this metric works better than absolute metrics (such as MOSNet) which is an absolute metric where lower is better, but optimized for speech. The FAD classifier is publicly available[3] through Google Research GitHub project, which is used for this project.

$$\mathbf{F}(\mathcal{N}_b, \mathcal{N}_e) = \|\mu_b - \mu_e\|^2 + tr(\Sigma_b + \Sigma_e - 2\sqrt{\Sigma_b \Sigma_e}) \tag{3.6}$$

[3] github.com/google-research/google-research/tree/master/frechet_audio_distance

4. Methodology

This section presents the method for this work. First it presents the input representation in Section 4.2 as was used in preliminary research. At the end a simplified input representation is presented, that will be used in this work when synthetic OBD is generated. Since the Tacotron input space was limited to 256 characters, a choice had to be made with variables to include in the language. The simplified language only considers revolutions per minute such that a higher resolution is available.

The overview of the Enginetron architecture during training is depicted in Figure 3. The Enginetron network is presented in Section 4.1 as well as the changes that are made in the network for exhaust note synthesis. Section 4.2 covers the custom language which is used in the Enginetron ETL unit.

Finally, the Enginetron synthetic OBD generator, FMOD-OBD-Dataset and FMOD-Enginesynth units are covered in the next chapter (Chapter 5 FMOD-OBD Dataset).

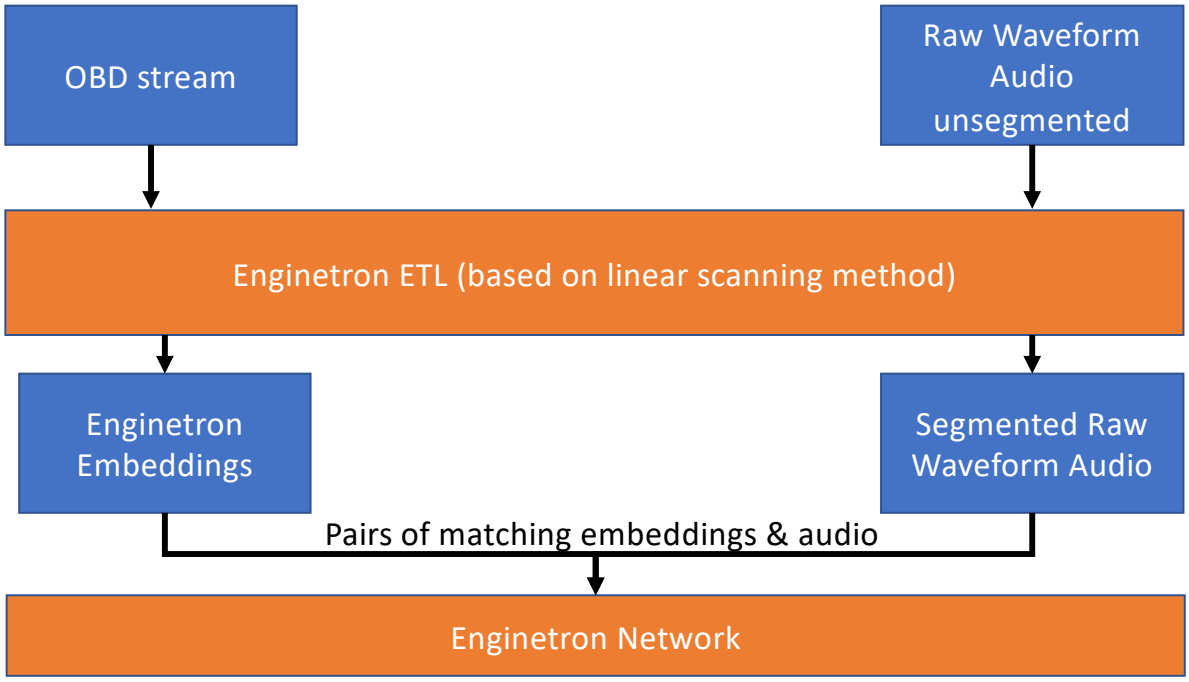

Figure 3: Overview of the Enginetron in training mode. Blue squares represent data, orange squares represent units of this work.

4.1 Enginetron network

This section will describe the architecture of Enginetron which is a slightly modified network based on Tacotron. Therefore this section will first cover the architecture of Tacotron, and then introduces the modifications made to train on exhaust note data.

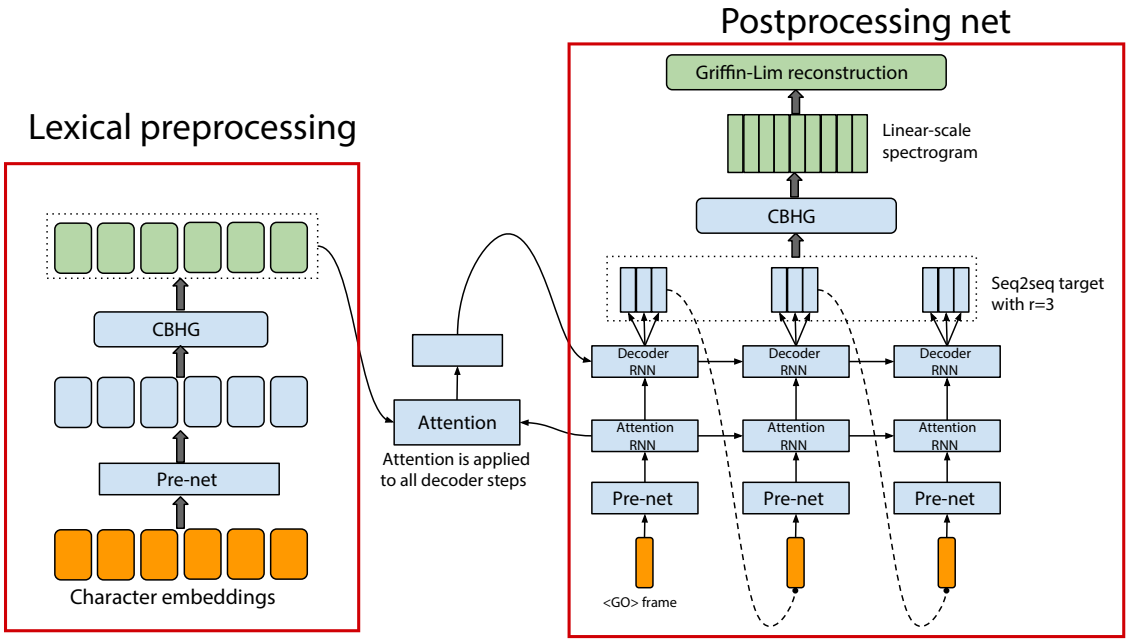

Figure 4: The tacotron network as described in [14]. The part before attention we annotated and will refer to as "lexical preprocessing". The attention we will refer to as attention. The part of the net after the attention we will refer to as postprocessing net.

Tacotron [14] (See Figure 4) is an end-to-end TTS network and trains on Text-Audio pairs and require minimal annotation. That is why for this work Tacotron is deemed the most promising. At a high-level, Tacotron takes characters as input and produces spectrogram frames, which are then converted to waveforms. Tacotron exists of the following models: CBHG module, Encoder, Decoder, Post processing and waveform synthesis. The CBHG module stands for 1-D Convolutional Bank, Highway Networks, and a Bidirectional Gated Recurrent Unit.

The CBHG is the combination of 1-D convolution banks with a layer of highway network, and after that a layer of bidirectional Gated Recurrent Unit (GRU). CBHG is a powerful module for extracting representations from sequences. The bank of 1-D convolutional filters is used to extract local information. This is passed through a highway network to extract higher-level features. Finally there is the Bidirectional GRU, which is used to learn long-term dependencies in the forward and backward directions.

The encoder consists of the CBHG module with a bottleneck layer with dropout to improve generalization. The CBHG module transforms the prenet outputs into the final encoder representation. The CBHG-based encoder reduces overfitting and mispronunciations.

The decoder targets are highly compressed 80-band mel spectrograms. The post-processing net

transforms the highly compressed representation into waveform. This post-processing net also uses a CBHG module, to learn to predict spectral magnitudes on a linear frequency scale. Multiple non-overlapping output frames are predicted at each decoder step to reduce the model size, training time and inference time. It also improves the convergence speed.

Modifications in network

This section covers the Enginetron architecture.

Most of the changes done in the network will therefore be in the Lexical preprocessing part of the network (depicted in Figure 4). The attention and postprocessing net are not changed in this work, although this could be promising for future work. In, and before the lexical preprocessing there are some changes made such that it works on datasets with exhaust notes and OBD data.

First of all, because there is a custom language that does not use standard latin alphabet, the alphabet is changed in "`basic_cleaners`". Since the Tacotron [14] pre-net has a 256-D character a language was build with no more than 256 unique characters. When creating a custom language for Tacotron that does not use a standard latin alphabet it is important to change the cleaners in Tacotron's hyperparameters. Tacotron's implementation comes with cleaners that convert numbers to text, remove whitespace and puncutation and convert common symbols or abbreviations to their full text. The remaining text is than filtered for the programmed alphabet and strips any characters outside of the alphabet. Also to use Tacotron with non-standard latin alphabet the `_characters` variable in `text/symbols.py` should be updated accordingly. The characters used are 256 different characters that are used to encode RPM. It is 256 consecutive characters in UTF-8 starting with $R_0 ... R_{256}$. Finally the hyperparameters that had that yielded the best results in preliminary research were used in this work too.

Enginetron uses Tacotron Text-to-Speech network with the similar changes as proposed in preliminary research [5]. The implementation by Keith Ito on GitHub is used as a starting point[1]. The modified network is made available here[2].

4.2 Input representation

In preliminary research [5] a language is presented that is used to represent the exhaust notes. In this section the most basic and important takeaways from this language will briefly be summarized. In Figure 5 you can see the Extended Backus-Naur form for the language. Speed is referred to as $S_0 \dots S_{127}$ The second 128 characters are used to represent RP. Those are referred to as $R_0 \dots R_{127}$. Analogously to real world corpus one could say that speed are the vowels, and RPM are the consonants of the "language".

There is also a slight extension of the language that will also be used in Figures or text in this thesis. The extension allows to add repeating words (i.e. for idling). This makes it easier to represent long repeating sequences in the paper. You can see the extension in Figure 6.

[1] `https://github.com/keithito/tacotron`

[2] `https://github.com/tpgmeligmeyling/tacotron-exhaustnotes`

$$\langle \textit{exhaust-sentence} \rangle ::= \langle \textit{exhaust-word} \rangle$$

$$\langle \textit{exhaust-word} \rangle ::= \{ \langle \textit{exhaustmorpheme} \rangle \}, \langle \textit{EOL} \rangle$$

$$\langle \textit{exhaustmorpheme} \rangle ::= \langle \textit{speed-symbol} \rangle, \langle \textit{rpm-symbol} \rangle$$

$$\langle \textit{speed-symbol} \rangle ::= S_0 \mid S_1 \mid S_2 \mid \ldots \mid S_{128}$$

$$\langle \textit{rpm-symbol} \rangle ::= R_0 \mid R_1 \mid R_2 \mid \ldots \mid R_{128}$$

Figure 5: EBNF for the exhaust note language as used in the descriptions. This is the purest form of the language as used in the datasets.

$$\langle \textit{exhaust-word} \rangle ::= \{ \langle \textit{exhaustmorpheme} \rangle \}, \langle \textit{EOL} \rangle \mid$$
$$\langle \textit{repeatN} \rangle \langle \textit{exhaustmorpheme} \rangle \}, \langle \textit{EOL} \rangle$$

$$\langle \textit{repeatN} \rangle ::= \{ \langle \textit{number} \rangle \} \times$$

$$\langle \textit{number} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Figure 6: EBNF extensions for the exhaust note language. Those extensions are only used for intelligibility of this thesis.

Simplified input representations

An engine under load sounds different than an engine that is not under load. Ideally an engine sound would be generated from the combination of the rpm, load, throttle and speed values. A lot of those values strongly correlate (or can be used to obtain new values). For instance the gear for a certain engine is a product of its speed and RPM. Because it is a difficult task to generate realistic OBD data with all these parameters this work opted to only use RPM for synthesis. This choice is justified cause a much larger dataset has precedence over having a detailed dataset. In this section a slight modification is presented of the input representation shown in Section 4.2. How the input representations are used as character embeddings in the TTS network is explained in the next section. The input representations are also used in the dataset creation in Chapter 5.

$$\langle \textit{exhaust-sentence} \rangle ::= \langle \textit{exhaust-word} \rangle$$

$$\langle \textit{exhaust-word} \rangle ::= \{ \langle \textit{exhaustmorpheme} \rangle \}, \langle \textit{EOL} \rangle$$

$$\langle \textit{rpm-symbol} \rangle ::= R_0 \mid R_1 \mid R_2 \mid \ldots \mid R_{128}$$

$$\langle \textit{exhaustmorpheme} \rangle ::= \langle \textit{rpm-symbol} \rangle$$

Figure 7: The modified EBNF language used for the dataset.

In Table 2 you see examples of the simplified language

Transcription	**Description**
$10 \times R_0$	Engine off
$R_1 1 R_1 2 R_1 2 R_1 2 R_1 3 \ldots$	Accelerating
$R_{17}(3 \times R_{16}) R_{15} \ldots$	Decelerating
$(100 \times R_{14})$	Cruising at constant speed
...	...
...	...

Table 2: Example of OBD data in simplified language, where R_i denotes steps of RPM where R_0 is the lowest RPM (engine off) and R_{128} is the max amount of revolutions per second for an engine.

For this work an ETL script is made available that converts raw OBD data and unsegmented raw waveform audio into shorter audio with those custom input representations.

Enginetron in synthesis mode

When Enginetron is trained, the custom embeddings can be used to synthesize raw audio. Figure 8 depicts a scheme of Enginetron's functionality when used for synthesis.

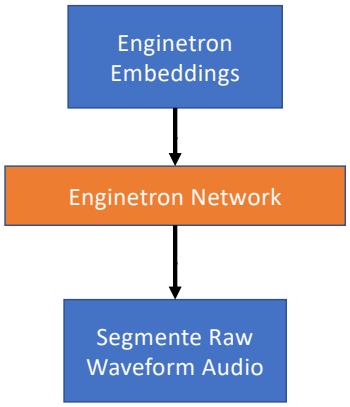

Figure 8: Overview of the Enginetron in synthesis mode. Blue squares represent data, orange squares represent units of this work.

5. FMOD-OBD Dataset

One of the first steps is the creation of a large dataset to train Enginetron. The creation of a large dataset can be divided in a few tasks. First the synthetic OBD data should be generated. Then a tool such as AudioMotors should be used to synthesize the sound for the synthetic OBD data. Then the dataset should be cut into ten seconds segments with matching audio and OBD annotations.

To create the dataset FMOD is used in combination with the AudioMotors plugin by LeSound. Those tools are useful because parameters can be set manually for RPM and load. In previous work it was found that an even distribution of those parameters in the training dataset is desirable. With those tools it was possible to generate a large dataset. Because it is synthesized from parameters a good transcription will be available to use for training with a Text-to-Speech network. To generate a larger and diverse dataset, one can also generate synthetic OBD, and then use it to synthesize a large dataset using FMOD with AudioMotors. Figure 9 schematically shows the process of dataset generation. The FMOD-Enginesynth is covered in Section 5.1. Synthetic data is generated using a Enginetron synthetic OBD generator presented in Section 5.2.

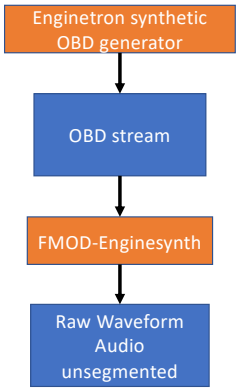

Figure 9: Overview of the data generation. Blue squares represent data, orange squares represent units of this work.

After generating synthetic OBD and synthesizing the synthetic OBD to raw waveform audio, the fake OBD and raw waveform collectively form FMOD-OBD Dataset as shown in Figure 10.

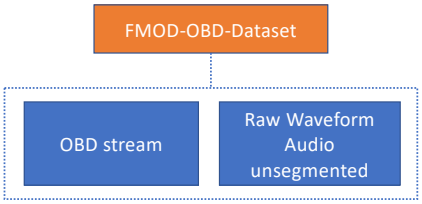

Figure 10: Overview of the dataset. Blue squares represent data, orange squares represent units of this work.

5.1 Synthesizing with FMOD Studio and AudioMotors

The first experiment to synthesize a dataset with FMOD with the AudioMotors plugin using OBD data. This was trained on the dataset from previous works in which OBD was recorded from the car. We used a script to synthesize the audio from FMOD. FMOD Studio however is a product that is specialized in making dynamic audio content for games. Bouncing out audio files produces linear audio content, and is therefore beyond the scope of what FMOD Studio is designed to do.

Since FMOD Studio does not come with an export audio option, a script was made to play back OBD CSV files in realtime. This was used to modify the RPM parameter of the AudioMotors plugin which was played back and recorded in realtime. Appendix A presents the BASH script that was used to record the audio. This method of synthesis is considered as a baseline method in this work. It will also be used for other experiments in this work. The code to synthesize exhaust note data is shown in Appendix A. Because the dataset of real OBD from the preliminary research was too short, synthetic OBD needs to be generated. The process of synthesizing raw OBD data will be covered in the next section.

5.2 Generating synthetic OBD

The previous section shown how to synthesize OBD using FMOD and LeSound AudioMotors plugin. This is an essential step in synthesizing a large dataset. Another step in generating a large dataset is generating synthetic OBD data. Also in preliminary research it was concluded that better results might be achieved in future work if the dataset was not only larger but also would contain many scenarios with a wide range of different RPMs and accelerations. The problem with a real dataset was that because most of it was recorded on a public road, the car was often at cruising speed, and not switching gears, accelerating or decelerating for that matter. In this section we will focus on generating synthetic OBD. Since we have simplified our language, we will only need the RPM and time labels. We still want to use the same CSV format as introduced in the Fundamentals in Section 3.1. The RPM, load and throttle will just be zero-padded (See Table 3). Using the same format makes it easier to extend the code in the future to also synthesize load, throttle and speed, as well as being able to reuse some parts of the ETL from preliminary research.

RPM [0, 8000]	**Time (ms)**
1631	0
1782	100
1835	200
1892	300

Table 3: Example of synthetic simplified OBD data.

The remainder of this section will go into the pseudocode used to generate the synthetic OBD. For the code the packages `numpy`, `bezier`, `csv` and `random` were used.

The code is divided in the following two algorithms:

1. `write_pull(csvwriter, is_accelerating, from_rpm, from_time)`

   This algorithm (as shown in Algorithm 1) is responsible for writing a pull. A pull is defined as an acceleration or deceleration from a certain RPM to a certain RPM that takes a certain time. A bezier path is computed with a random bezier point that defines a curve from the RPM to the RPM over the time. The function returns a tuple with a boolean whether or not the next pull should be accelerating or decelerating, the final RPM, and the final time. Those values can be used for the consecutive pull. Pulls are always altering between accelerating and decelerating.

2. `make_dataset(csvwriter)`

   This algorithm (as shown in Algorithm 2) generates one hour of synthetic OBD.

Result: A pull written to CSV
Function `write_pull`(*csvwriter, is_accelerating, from_rpm, from_time*)**:**
    time = from_time
    pull_duration = random(4, 10)
    if *is_accelerating* **then**
        to_rpm = random(3000, 6000)
    else
        to_rpm = random(1000, 2500)
    end
    pull_bezier_point = random(1, pull_time - 1)
    if *is_accelerating* **then**
        pull_rpm_point = random(from_rpm + 100, to_rpm - 100)
    else
        pull_rpm_point = random(from_rpm - 100, to_rpm + 100)
    end
    $\text{curve\_matrix} = \begin{bmatrix} 0, & \text{pull\_bezier\_point}, & \text{pull\_time} \\ \text{from\_rpm}, & \text{pull\_rpm\_point}, & \text{to\_rpm} \end{bmatrix}$
    curve = create_bezier(curve_matrix)
    for *from progress_time to pull_time in steps of 0.1 seconds* **do**
        progress = progress_time / pull_time
        rpm = curve.evaluate(progress)
        write_rpm_and_time(csv, rpm, time)
        time = time + 100
    end
    return (not is_accelerating, to_rpm, time)

Algorithm 1: Function to write a pull. A pull is either one accelerating with a from and to rpm, or a deceleration with a from and to rpm.

Result: Writes an one hour dataset of generated OBD data.
Function `make_dataset` (*csvwriter*)**:**
    is_accelerating = True
    rpm = 1000
    time = 0
    while *time < 3600000* **do**
        is_accelerating, rpm, time = write_pull(csv, is_accelerating, rpm, time)
    end

Algorithm 2: Function that loops until one hour of OBD data is generated. The code is alternating between accelerating and decelerating.

5.3 OBD and raw audio waveform ETL to Enginetron script

Since the input language has changed a modification of the ETL script is also used. The full transformation script can be found in Appendix B. This script converts the generated OBD data and the raw audio waveform that is one hour long into 10 second audio files and their transcription in the custom input representation. In the preliminary research [5], there were two different methods for the ETL. One was called "linear scanning" which just started at the beginning and would cut every ten seconds (and convert the matching OBD into our custom character embeddings). The other method would find a (remaining) place in the OBD data with a lot of variation, and cut the matching ten seconds of audio (and convert the matching OBD into our custom character embeddings). In this work we can rely on the linear scanning, since our synthesized source OBD has enough variation.

5.4 Transformed dataset

The dataset is processed with the script presented in Section 5.3. The dataset contains transcriptions and a location a reference to the stripped waveform audio file path. The individual files are all just under 10 seconds. The transformed dataset is divided in three parts. There is a 12 hour training dataset. There is an 1 hour test dataset and finally there is an 1 hour validation dataset.

Session	**Length**	**Percentage**
Train	12 hours and 0 minutes	~86%
Validation	1 hour and 0 minutes	~7%
Test	1 hour and 0 minutes	~7%
Total	14 hours and 0 minutes	100%

Table 4: The composition of the dataset.

5.5 Test trained network script

After training the network all the samples in the test dataset are synthesized such that they can be compared with the original signal. To generate all the samples in the test dataset a script was made.

6. Experimental Setup

This section presents the experiments in this thesis. Most experiments work on one the FMOD-OBD Dataset presented in Chapter 5. In Table 5 an overview is given of the experiments. This table shows the experiment number, name, section that described it in greater detail, goal and the metric that is used. The results are presented in the next chapter.

No.	Name	Section	Goal	Metric
1	Novel metrics	6.1	Comparison of several F_0-based metrics on exhaust notes. Useful metrics are used in following experiments	Various
2	FMOD-Enginesynth	6.2	Baseline method for exhaust note synthesis using FMOD-OBD Dataset. This baseline can be used to determine performance of other methods	FAD Score
3	Enginetron	6.3	Proposed method for exhaust note synthesis using FMOD-OBD Dataset to evaluate effectivenes of using TTS networks for exhaust note synthesis	FAD Score

Table 5: Overview of experiments

6.1 Exploring novel metrics for exhaust notes synthesis

This experiment will evaluate different metrics from Section 3.5 and 3.6. It will cover F_0 evaluation.

Networks for which melody is important typically have a pitch encoder in the text-to-speech network. In this experiment we will use common pitch extraction methods on exhaust note sounds and evaluate their precision. We use the world vocoder to extract the F_0 sequence. It will compare different F_0 estimators and their performance on the FMOD-OBD Dataset. The performance is measured using the RMSE_{F_0} by comparing the estimations with expected values.

A Python implementation of the WORLD vocoder was used [27]. Code[1] is available. The python package librosa[2] [17] also comes with two different F_0 estimation algorithms *yin* [18] and *pyin* [19]. Those were also experimented with.

Besides the F_0 comparison this experiment contains a quantitative experiment showing the presence of harmonics in FMOD exhaust notes on a 2000 RPM constant exhaust note.

[1] https://github.com/tuanad121/Python-WORLD

[2] https://librosa.org/doc/main/index.html

6.2 Synthesizing real OBD through FMOD-Enginesynth

In Section 5.1 a method is presented to synthesize OBD data with FMOD AudioMotors. This script plays the OBD-data in real time and sets the RPM value on the AudioMotors plugin to the corresponding value in the OBD dataset every tick. Although the main objective for this script was to generate synthetic OBD to create a large dataset through AudioMotors, this experiment focusses on synthesizing real OBD data through this method. The OBD data that was used for this experiment was made available in preliminary research [5].

As mentioned in the fundamentals (Chapter 3) using granular synthesis is the current best and most commonly used way to synthesize exhaust notes. Since AudioMotors is an established name in the industry, and available to us, this method can be used as a baseline to compare novel exhaust note synthesis methods with. Finally on the resulting exhaust notes novel metrics (as presented in Section 6.1) can be applied.

For metrics a FAD score is computed. The background embeddings for the FAD-score are the ones from the original 10 second sample that FMOD uses internally to synthesize any engine sound.

6.3 Synthesis of exhaust notes through Enginetron

In the FMOD-Enginesynth experiment (Section 6.2) we synthesized real OBD through FMOD & AudioMotors. This experiment focusses on synthesizing OBD through Enginetron. In Chapter 4 Enginetron is presented. In this experiment Enginetron will be trained and evaluated. For training the the FMOD-OBD Dataset with synthesized audio trough FMODAudiosynth will be used. A custom ETL script as presented in Section 5.3 will transform the dataset with our custom embeddings such that Enginetron can be trained. The network will be trained until the loss function flattens typically at 100-150K steps. A FAD score will then be computed with the same background statistics as used in the FMOD-Enginesynth experiment. Since the same benchmark is used, those two scores should be comparable.

7. Results

Within this chapter the results of the experiments are presented. The first experiment describes how different metrics were experimented with. At last the metric is decided that is used in the other experiments for evaluation. From this experiment it was decided that the FAD score will be applied as a metric for the FMOD-Enginesynth and Enginetron experiments to evaluate the performance.

7.1 Novel metrics for exhaust notes

This experiment's goal is comparison of several F_0-metrics on exhaust notes. The goal of this experiment was to use automated F_0 extraction using librosa, Python WORLD. Experiments were done with fundamental frequency estimation algorithms *yin* [18] and *pyin* [19] from librosa and pyworld dio and harvest. See Chapter 8 for more details on proposed future work for F_0 extraction on exhaust notes.

Method	**RMSE**$_{F_0}$ (lower is better)
YIN	51.121 Hz
PYIN	35.605 Hz
PyWorld dio	68.708 Hz
PyWorld harvest	33.717 Hz

Table 6: Comparison of different F_0 estimators and their RMSE to expected fundamental frequency. on the twelve hours of training FMOD-OBD Dataset.

Qualitative evaluation of a F_0 on fixed 2000 RPM
To explore metrics for exhaust notes an analysis of the fundamental frequencies at The FMOD synthesis was first tested by synthesizing a constant 2000 RPM exhaust note. We can apply the formula from Section 3.5. Peaks in amplitude are clearly visible at, F_{en} and F_c are shown in Equation 7.2 and 7.1 respectively. You can see that the highest in amplitude is F_{ex} as shown in Equation 7.3. For F_0 extraction we consider that the explosion frequency is the fundamental frequency: $F_0 = F_{ex}$.

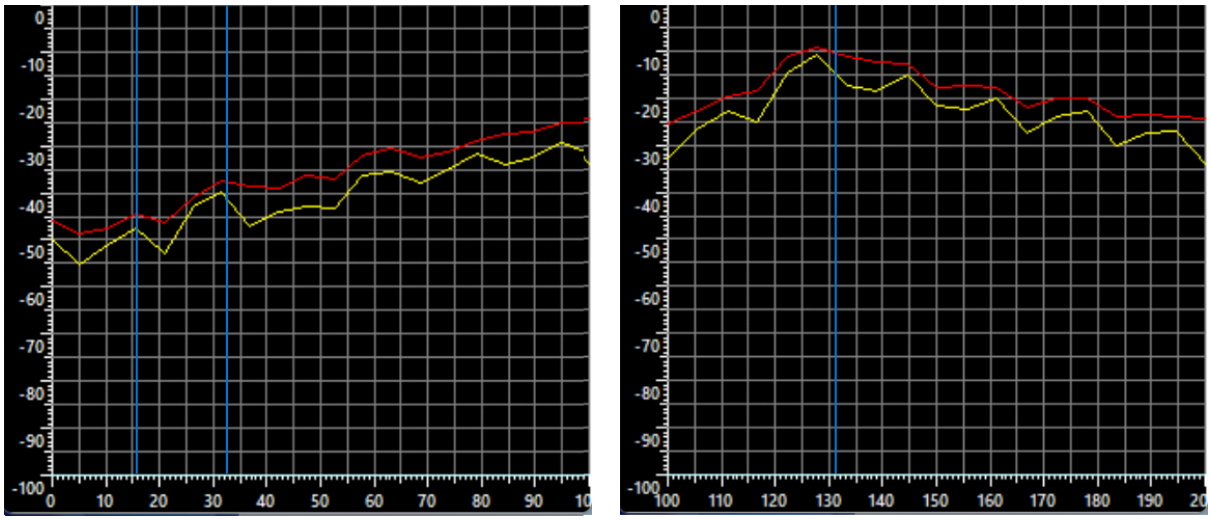

Figure 11: Both figures show the 2000 rpm charger being played back. Left shows the frequencies 0 to 100 Hz you can see F_c, F_{en} and right shows frequencies 100 till 200 Hz in which F_{ex} is clearly visible. On the x-axis it is frequencies. On the y-axis it is dB.

$$F_c = \frac{1}{2}\frac{2000}{60} = 16\frac{2}{3} \tag{7.1}$$

$$F_{en} = \frac{2000}{60} = 33\frac{1}{3} \tag{7.2}$$

$$F_{ex} = \frac{8}{2}\frac{2000}{60} = 133\frac{1}{3} \approx 133.33 \tag{7.3}$$

7.2 Synthesis of real OBD through FMOD-Enginesynth

First we synthesized real OBD data through FMOD with AudioMotors plugin. This subjectively sounded more realistic than the results achieved in preliminary research. This synthesis method however currently did not have gear shifting sounds. Also because this method only synthesizes engine noise it misses background noises such as wind noise, tire noise etc. The problem with this training approach is that if the car is accelerating too fast, you hear that it is skipping from one RPM to one much further instead of smoothly transitioning. This is because the OBD dataset has about 100 ms between each datapoint. In Figure 12 you see a spectrogram of an synthesized exhaust note. As part of this work the audio is provided for listening. FMOD-Enginesynth obtained a FAD score of 1.73.

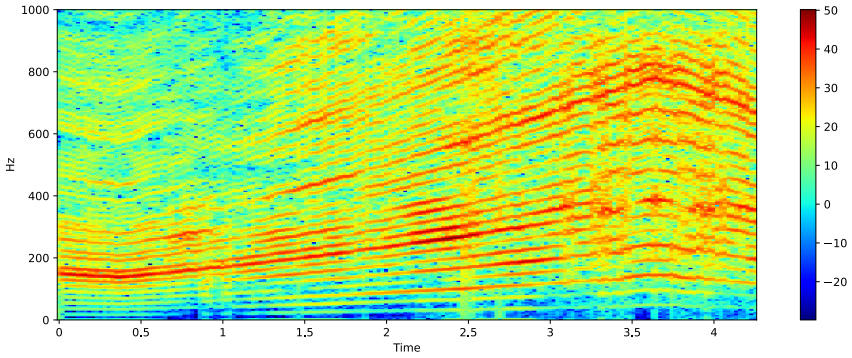

Figure 12: Mel spectrogram of a sample generated with FMOD-Enginesynth. Frequencies are limited from 0 to 1000 Hz.

7.3 Synthesis of exhaust notes through Enginetron

For this work Enginetron was trained on the generated FMOD-OBD Dataset, a much larger dataset than that was used in preliminary research. This dataset also has the benefits that there is no artifacts in the audio such as wind noise, tire noise, passing cars etc. The synthesized samples are made available as part of this work. In Figure 13 you see the loss function during training. The y-axis is logarithmic.

To evaluate the quality of the trained network all the samples in the test dataset where synthesized and observed. Interestingly some of the samples are 12.5 (max_iters · outputs_per_step · frame_shift_ms) seconds long instead of the 10 seconds that the training samples are. Typically the end of the sample will sound bad. This is a sign that the decoder does not know when to stop. This can be solved (in future work) using a special stop token and stop token projection layer in the network.

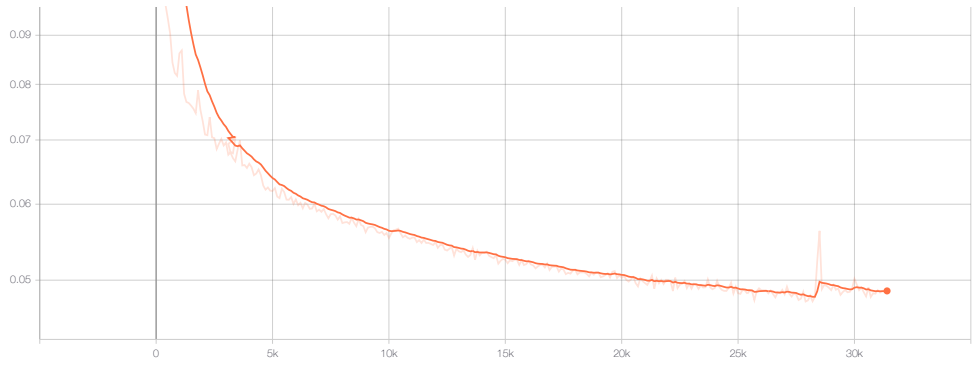

Figure 13: Loss function during training after 35000 steps. On the x-axis you see the steps the network was trained. On the y-axis you see the loss.

As you can see in Figure 14 and Figure 15 the signals after training have maintained harmonic characteristics of exhaust notes. You can see the harmonics are preserved after training Enginetron.

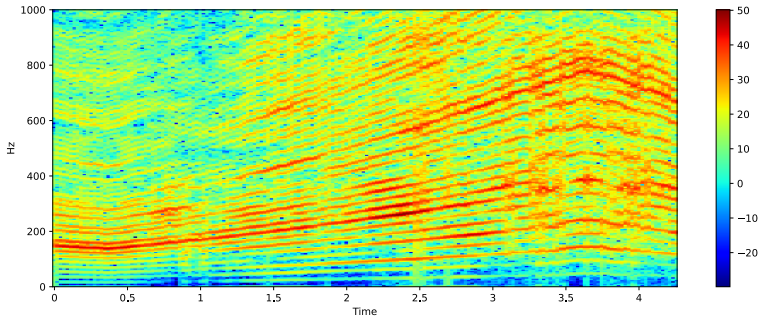

Figure 14: Mel spectrogram of a training dataset sample. Frequencies are limited from 0 to 1000 Hz.

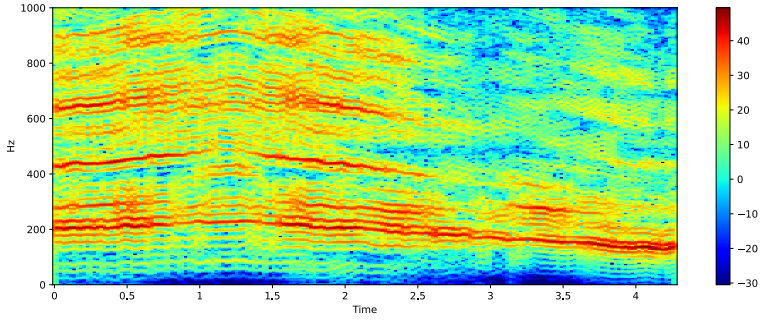

Figure 15: Mel spectrogram of a Enginetron exhaust note. Frequencies are limited from 0 to 1000 Hz.

The FAD score as presented in Section 3.8 was computed. To do so the original training dataset and a synthesized test dataset was put in place. For the original dataset statistics were exported using the network. The same was done for the synthesized dataset. With the `frechet_audio_distance.compute_fad` the final FAD score was computed for this experiment. Enginetron achieved a FAD score of 7.88. In Table 7 you see the FMOD-Enginesynth and Enginetron's performance compared.

Method	**FAD** (lower is better)
FMOD-Enginesynth	1.73
Enginetron	7.88

Table 7: FMOD-Enginesynth and Enginetron performance using FAD metric.

8. Conclusions and Future Work

A large dataset of OBD data with matching sound was generated that can be used to train large networks. Finally a network was presented that can be used to synthesize realistic exhaust notes when trained on the large dataset of OBD data and raw audio. A pre-trained network is made available as part of this thesis.

Training on this more diverse dataset that was generated has proven that a bigger more diverse dataset, with not as much background noise gives a better result.

Finally a study of different metrics was presented. It was observed that available F_0 extraction algorithms do not perform well on exhaust notes even with the F_0 floor and F_0 ceil parameters adjusted.

Training a neural network metric to derive RPM from an audio signal is something that was not attempted due to time constraints but likely a better approach to get the RPM, and therefor the fundamental frequency of the car.

Key Contributions
This work presented FMOD-OBD Dataset, which is over twelve hours worth of OBD data and matching exhaust notes audio. It presents the FMOD-Enginesynth method which can be used to synthesize exhaust notes and scores a FAD score of 1.73. Lastly this work presents a novel method Enginetron which is a modified text-to-speech network with custom embeddings trained on the FMOD-OBD Dataset. This method is Enginetron which scores a FAD score of 7.88 (lower is better).

Although there is quite a difference in FAD score between Enginetron and FMOD-Enginesynth, we think that the qualitive characteristics of Enginetron are very promising. As a qualitative evaluation by the author, sound is considered significantly improved compared to preliminary research [5]. The conclusion is that with sufficient annotated data a modified text-to-speech network can be trained to synthesize other sounds than speech, in this case exhaust notes.

Future work
Since synthesizing exhaust notes with text-to-speech networks is a new research direction there is a lot of future work. First this work presents a generated dataset, but training on a real dataset that is professionally recorded would be beneficial. The AudioMotors plugin used to create the large dataset uses granular synthesis and therefore using it to train a neural network inadvertently brings any imperfections in this method into the trained network. Finally the current dataset did not contain any gear shiftings. Also since the OBD data was generated too and only contained the speed parameter now. Better results could be achieved with a dataset that has more parameters such as load, throttle, and speed.

The network could also be improved to synthesize a better exhaust note. There are better networks with higher fidelity available such as Tacotron2, which were not used for this work cause they require a lot of time to train. As shown in the results the network could benefit from a stop token. There is a Tacotron fork available that adds Tacotron2's local-sensitive attention and stop token available here[1].

Also encoding pitch into the network would be beneficial. For this it is best if a method is found to reliable calculate the F_0 for an exhaust note signal. Future work could develop a neural based metric to get the F_0 from an exhaust note signal.

[1] https://github.com/begeekmyfriend/tacotron

Bibliography

[1] Richard Lyon, *Designing for product sound quality*, 2000.

[2] M Ercan Altinsoy, Jürgen Landgraf, Margitta Lachmann, Matthias Esser, and Dirk Volkenborn, "Investigations on the detectability of synthesized electric vehicle sounds-vehicle operation: Approaching at 10 km/h," in *Forum Acusticum, September*, 2014.

[3] Ichiro Sakamoto, Michiaki Sekine, Kazumoto Morita, and Hiroyuki Houzu, "Research for standardization of measures against quiet hv/ev in japan," in *Forum Acusticum, September*, 2014.

[4] M Ercan Altinsoy, Jürgen Landgraf, Matthias Esser, and Dirk Volkenborn, "Electric vehicle alert sound design–comparison of commencing motion sound and sound at idle," in *Forum Acusticum*, 2014.

[5] Thomas Gmelig Meyling, "Realtime car exhaust note synthesis using on-board diagnostics through text-to-speech networks," MSc Research Report, 2021.

[6] Aaron Lee Hastings, *Sound quality of diesel engines*, Ph.D. thesis, Purdue University, 2004.

[7] Jan Jagla, Julien Maillard, and Nadine Martin, "Sample-based engine noise synthesis using an enhanced pitch-synchronous overlap-and-add method," in *The Journal of the Acoustical Society of America*, 2012, vol. 132, pp. 3098–3108.

[8] Curtis Roads, "Introduction to granular synthesis," in *Computer Music Journal*, 1988, vol. 12, pp. 11–13.

[9] Andrew J Hunt and Alan W Black, "Unit selection in a concatenative speech synthesis system using a large speech database," in *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*. IEEE, 1996, vol. 1, pp. 373–376.

[10] Xavi Gonzalvo, Siamak Tazari, Chun an Chan, Markus Becker, Alexander Gutkin, and Hanna Silen, "Recent Advances in Google Real-Time HMM-Driven Unit Selection Synthesizer," in *Proc. Interspeech 2016*, 2016, pp. 2238–2242.

[11] Heiga Zen, Keiichi Tokuda, and Alan W Black, "Statistical parametric speech synthesis," .

[12] Heiga Zen, Yannis Agiomyrgiannakis, Niels Egberts, Fergus Henderson, and Przemysław Szczepaniak, "Fast, Compact, and High Quality LSTM-RNN Based Statistical Parametric Speech Synthesizers for Mobile Devices," in *Proc. Interspeech 2016*, 2016, pp. 2273–2277.

[13] En Li, Zhi Zhou, and Xu Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proceedings of the 2018 Workshop on Mobile Edge Communications*, 2018, pp. 31–36.

[14] Yuxuan Wang, R.J. Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc Le, Yannis Agiomyrgiannakis, Rob Clark, and Rif A. Saurous, "Tacotron: Towards End-to-End Speech Synthesis," in *Proc. Interspeech 2017*, 2017, pp. 4006–4010.

[15] Jonathan Shen, Ruoming Pang, Ron J Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, et al., "Natural tts synthesis by conditioning wavenet on mel spectrogram predictions," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 4779–4783.

[16] Juheon Lee, Hyeong-Seok Choi, Chang-Bin Jeon, Junghyun Koo, and Kyogu Lee, "Adversarially Trained End-to-End Korean Singing Voice Synthesis System," in *Proc. Interspeech 2019*, 2019, pp. 2588–2592.

[17] Brian McFee, Colin Raffel, Dawen Liang, Daniel P Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*. Citeseer, 2015, vol. 8, pp. 18–25.

[18] Alain De Cheveigné and Hideki Kawahara, "Yin, a fundamental frequency estimator for speech and music," in *The Journal of the Acoustical Society of America*, 2002, vol. 111, pp. 1917–1930.

[19] Matthias Mauch and Simon Dixon, "pyin: A fundamental frequency estimator using probabilistic threshold distributions," in *2014 ieee international conference on acoustics, speech and signal processing (icassp)*. IEEE, 2014, pp. 659–663.

[20] ITUT Recommendation, "Vocabulary for performance and quality of service," *International Telecommunications Union—Radiocommunication (ITU-T), RITP: Geneva, Switzerland*, 2006.

[21] Rensis Likert, "The likert-type scale," *Archives of Psychology*, vol. 140, no. 55, pp. 1–55, 1932.

[22] Masanori Morise, Fumiya Yokomori, and Kenji Ozawa, "World: a vocoder-based high-quality speech synthesis system for real-time applications," *IEICE TRANSACTIONS on Information and Systems*, vol. 99, no. 7, pp. 1877–1884, 2016.

[23] Chen-Chou Lo, Szu-Wei Fu, Wen-Chin Huang, Xin Wang, Junichi Yamagishi, Yu Tsao, and Hsin-Min Wang, "Mosnet: Deep learning-based objective assessment for voice conversion," *Proc. Interspeech 2019*, pp. 1541–1545, 2019.

[24] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[25] Kevin Kilgour, Mauricio Zuluaga, Dominik Roblek, and Matthew Sharifi, "Fréchet audio distance: A reference-free metric for evaluating music enhancement algorithms," in *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*, Gernot Kubin and Zdravko Kacic, Eds. 2019, pp. 2350–2354, ISCA.

[26] Tun-Min Hung, Bo-Yu Chen, Yen-Tung Yeh, and Yi-Hsuan Yang, "A benchmarking initiative for audio-domain music generation using the freesound loop dataset," in *Proceedings of the 22nd International Society for Music Information Retrieval Conference, ISMIR 2021,*

Online, November 7-12, 2021, Jin Ha Lee, Alexander Lerch, Zhiyao Duan, Juhan Nam, Preeti Rao, Peter van Kranenburg, and Ajay Srinivasamurthy, Eds., 2021, pp. 310–317.

[27] Tuan Dinh, Alexander Kain, and Kris Tjaden, "Using a manifold vocoder for spectral voice and style conversion," *Proc. Interspeech 2019*, pp. 1388–1392, 2019.

A. FMOD Synthesis script

```
#!/bin/bash

write_commands() {

  sleep 5
  initial=1607170739727

  echo 'pluginSound.volume = 0'
  echo 'var event = studio.project.lookup("event:/Engine");'

  while IFS=, read -r rpm load unused1 unused2 epoch
  do
      epoch=$(echo $epoch| sed -e 's/\r$//')
      timelast=$(echo $epoch - $initial | bc)
      wait=$(echo ${timelast}.0 / 1000.0 | bc -l)
      initial=$epoch
      echo "event.timeline.automationCurves[0]
          .automationPoints[0].value = $rpm;"
      sleep $wait
          done < <(tail -n +2 record1607170737164217000.csv)

  echo 'pluginSound.volume = -100'
}

main() {
  if ! command -v telnet &> /dev/null
  then
    echo "telnet could not be found"
    exit
  fi

  write_commands | telnet localhost 3663
}

main
```

Listing A.1: Script used to synthesize audio from CSV with OBD-data using FMOD Studio with AudioMotors plugin.

B. Dataset ETL script

```
import csv
from pydub import AudioSegment
import math

dataset = "dataset-12"
stoken = "1"
firstTime = 0
currentSegmentTime = 0
iterations = 0
audio = AudioSegment.from_wav(dataset + ".wav")
currentString = ''

def processSegments(csvReader, csvwriter):
  global firstTime
  global currentSegmentTime
  global currentString
  global iterations
  global audio
  for row in csvReader:
      iterations = iterations + 1
      rowTime = int(row[4])
      if row[0] == 'None' or row[1] == 'None' or row[2] == 'None'
        or row[3] == 'None':
        continue
      if firstTime == 0:
        firstTime = rowTime
      if currentSegmentTime == 0:
        currentSegmentTime = rowTime

      rpm = math.floor(float(row[0]))
      rpmEncode = math.floor(rpm / 25)

      currentString = currentString + str(chr(rpmEncode+192))
      if ((rowTime - currentSegmentTime) / 1000) >= 10:
        audioStart = round(currentSegmentTime - firstTime, -3)
        audioEnd = round(rowTime - firstTime, -3)
        if audioEnd > len(audio):
          break

        segmentAudio = audio[audioStart:audioEnd]
        segmentAudio.export('wavs/' + stoken +str(iterations)+'.wav',
                            format='wav')
```

```
            csvwriter.writerow([
              stoken + str(iterations), currentString, currentString
            ])

            currentSegmentTime = 0
            currentString = ''

with open(dataset + '.csv', newline='') as csvfile,
     open('metadata.csv', 'w', newline='') as csvoutfile:
  spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
  csvwriter = csv.writer(csvoutfile, delimiter='|', quotechar='"',
                         quoting=csv.QUOTE_MINIMAL)
  processSegments(spamreader, csvwriter)
```

Listing B.1: Code to transform the waveform and corresponding raw OBD data to a common text-to-speech format similar to LJSpeech.