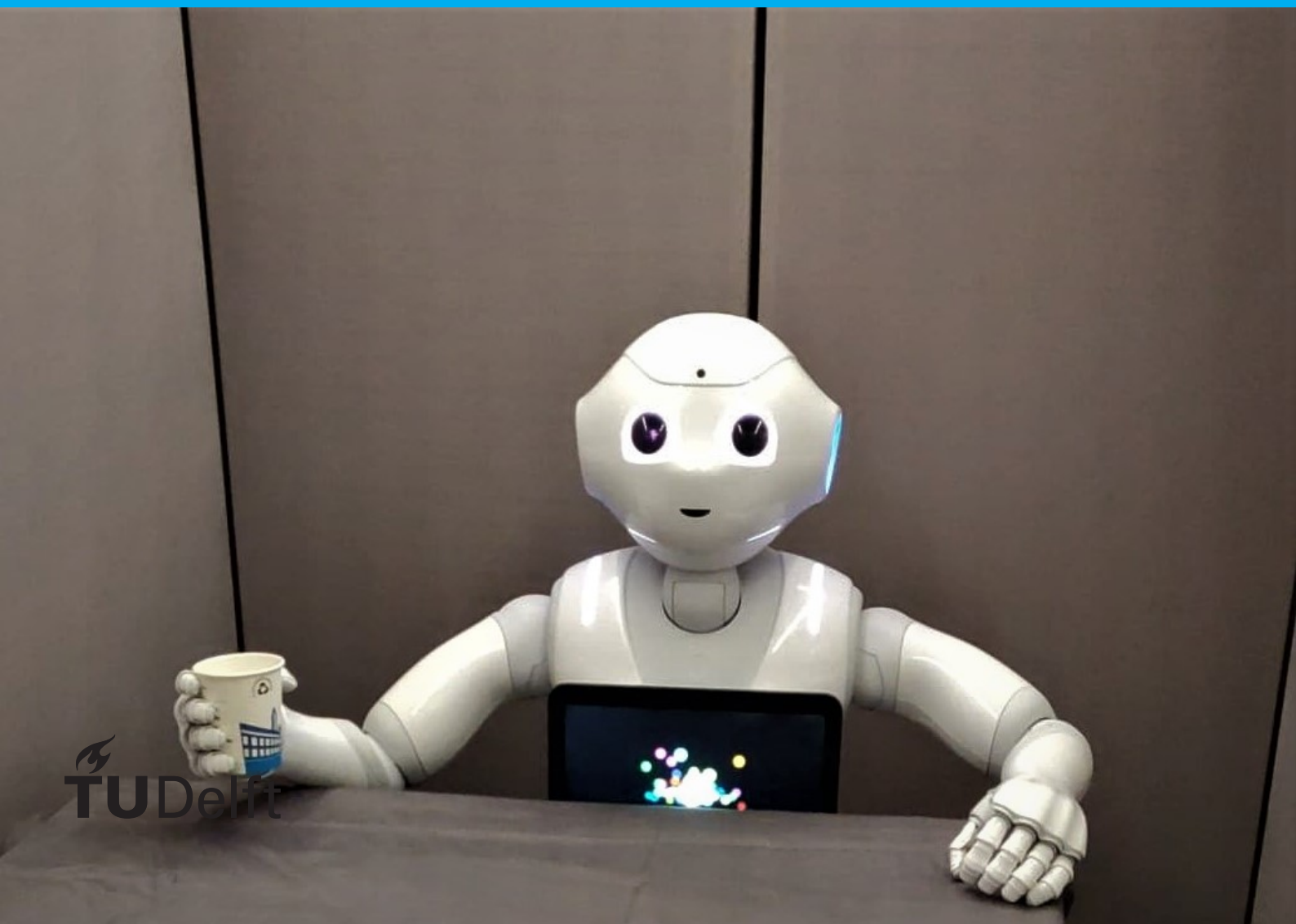
Evaluating an ASR Pipeline for a Social Robot

J. Sparreboom



Evaluating an ASR Pipeline for a Social Robot

by

J. Sparreboom

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday August 28, 2019 at 12:15PM

Student number:	4204751	
Thesis committee:	O. Scharenborg,	TU Delft, supervisor
	K.V. Hindriks,	TU Delft
	C. Oertel,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Abstract

There has been a big increase in the use of social robots, such as Pepper, which use verbal communication as the main method of interacting with a human. Verbal communication with a robot is performed using Automatic Speech Recognition (ASR) to recognize words from an audio stream containing speech. These social robots are being more frequently used in noisy environments. As such, this thesis investigates 1) whether Pepper's built-in keyword spotter can be replaced by an ASR system able to recognize continuous speech in Dutch; 2) whether Pepper's ASR pipeline can be made more robust against noise, without changing its hardware. To that end, Pepper's built-in keyword spotter and Sound Source Localization (SSL) algorithm are evaluated against an ASR pipeline based on a Delay-and-Sum beamformer, MUSIC Sound Source Localization, and Google Cloud Speech-to-Text.

The proposed pipeline showed a significant decrease in Keyword Error Rate of 6.2% compared to Pepper's built-in keyword spotter, and a significant decrease of Word Error Rate (WER) of 21.4% on Dutch continuous speech in clean listening conditions. A decrease in WER of 13.3% was observed in an SNR of 8 dB, and a decrease in WER persisted throughout lower Signal-to-Noise ratios (SNR).

As such, it has been shown that Pepper's speech recognition can be improved and made more robust against noise by preprocessing the audio using MUSIC SSL and a Delay-and-Sum beamformer, and transcribing the speech (in Dutch) using Google Cloud Speech-to-Text.

Preface

In this thesis, I evaluated the ASR pipeline of the Pepper robot compared to a proposed pipeline based on MUSIC Sound Source Localization and Delay-and-Sum beamforming. This research is done at the Multimedia Computing Group of the Faculty of Electrical Engineering, Mathematics, and Computer Science of the Delft University of Technology.

I started this research with only a small amount of background knowledge in the field of Signal and Audio Processing, ASR and beamformers, though I have gained much knowledge throughout the creation of this thesis. During the creation of this thesis, I have also submitted a paper (with help from Odette Scharenborg and Koen Hindriks) for the IEEE Automatic Speech Recognition and Understanding Workshop 2019 (ASRU) with preliminary results achieved during this research, the result of which can be found in Appendix B.

I would not have been able to create this thesis without the help of my supervisors. I would like to thank my main supervisor Odette Scharenborg, for her quick and helpful feedback throughout this research (even outside of office hours), and for helping me push through the stressful phases. I would also like to thank Koen Hindriks for his help, both while being my supervisor during the first months of my thesis, and continually afterwards. Of course I also want to thank Catharine Oertel for joining my committee.

Furthermore I would also like to thank Marnix de Graaf, Zilla Garama and Lars van der Zwan for ensuring there were copious amounts of noise and laughter while working in the Insyght Lab.

J. Sparreboom
Delft, August 2019

Contents

1 Introduction

In the last years, the number of advancements in robotic solutions has grown quickly. This increased research in robotics has brought forward many different types of robots. These robots are being used in industrial applications in the form of, for example, robotic arms for sorting or picking up items, autonomous buggies to do pick-ups in warehouses and social robots.

Social robots could be used to perform many different scenarios, e.g., to welcome guests in a hotel [1], to provide information in a shopping mall [2], to interview patients to collect patient data [3], or to educate children [4]. These scenarios assume some form of communication between a human and the social robot. The most intuitive communication method when conversing with a (humanoid) social robot is using voice interaction. For this type of interaction, Automatic Speech Recognition (ASR) is required.

ASR is the process of recognizing words from an audio stream containing speech. This can be used for transcription and also as input for a dialogue system. To use ASR in any scenario, it is vital that the quality of the recognized transcription is as good as possible. Wrong transcriptions can negatively influence the user experience, especially in human-robot dialogues. While ASR systems perform relatively well in situations without noise, with Word Error Rates dropping as low as 5% [5], its performance can drastically decrease in more adverse environments [6].

This thesis aims at evaluating the ASR pipeline in a Pepper[1] social robot. In recent years, the Pepper robot is increasingly more often used at events and in companies. To the best of my knowledge, the research in this thesis is the first to evaluate and implement a Dutch ASR pipeline, including a beamformer, specifically for Pepper; although similar work has been carried out on other social robots [7, 8], or on Pepper without additional preprocessing [9, 10]. Section 2 gives an overview of these related works.

Social robots often need to deal with noisy environments in the scenarios in which they are used. Unfortunately, typically, the performance of the ASR systems of these robots deteriorates in noisy conditions [8], limiting the locations and situations in which such a robot can be effectively used. Given the social setting Pepper is used in most commonly, the solutions should also only make use of the hardware that Pepper contains by default. Adding additional hardware, such as more microphones, would require these to be attached on the

[1] `https://www.softbankrobotics.com/us/pepper`

exterior of the Pepper or make significant changes to its design. Attaching hardware externally could negatively influence the Human-Robot interaction, as Pepper would get a less humanoid expression. The limitation of not adjusting Pepper's hardware does not include the usage of external processing units, such as a laptop connected to Pepper by WiFi.

There are different ways to make an ASR system more robust to noise, e.g., using speech enhancement [11, 12] or by training the ASR system on noisified speech [13]. However, Pepper's speech processing system is not trainable, nor can it deal with preprocessed data. Consequently, to deal with background noise, these methods cannot be used directly. The main aim of my research is to try to make Pepper more robust against the presence of background noise without changing its hardware, by using noise reduction based on a beamformer.

1.1. Research Questions

This thesis aims to answer two questions:

1. *Can Pepper's built-in keyword spotter be replaced by an ASR system able to recognise continuous speech in Dutch?*

2. *Can Pepper's ASR pipeline be made more robust against noise, without changing its hardware?*

Pepper's built-in ASR pipeline forces some limitations upon the user, e.g., recordings cannot be used as input, preprocessing on live audio data is not possible, and the number of supported languages is limited. For example, while Pepper is capable of recognizing continuous speech for multiple languages, including English and Japanese[2], Dutch is one of the languages for which only recognition of pre-set utterances, i.e., a keyword spotter, is supported[3]. The inability of Pepper's built-in ASR system to transcribe continuous Dutch speech, or other less supported languages, severely limits its functioning as a flexible social robot world-wide.

Replacing Pepper's ASR system could allow the circumvention of these limitations. To that end, we propose to replace Pepper's built-in keyword spotter with an ASR service in the cloud. Only a limited number of cloud ASR systems for Dutch were available; we chose Google Cloud Speech-to-Text (GC-STT)[4].

Using GC-STT also allows us to preprocess the speech signal to remove the background noise. Specifically, I propose to replace Pepper's built-in Sound Source Localization (SSL) algorithm with a pipeline using SSL based on Multiple Signal Classification (MUSIC) [14, 15], and to add a Delay-and-Sum beamformer [16, 17] to preprocess the data to reduce noise and enhance the speech signal before performing the ASR [5]. Importantly, because Pepper is typically used in social settings, all solutions should work in (pseudo-)real-time,

[2] Languages supported by Pepper's ASR: `http://doc.aldebaran.com/2-5/family/pepper_technical/languages_pep.html`

[3] Pepper does not allow for continuous speech recognition as mentioned on `http://doc.aldebaran.com/2-5/naoqi/interaction/dialog/dialog-syntax_full.html?highlight=qichatinputstoring#input-storing`

[4] More information about Google Cloud Speech-to-Text can be found here: `https://cloud.google.com/speech-to-text/`

[5] A shorter version of this research has been submitted as workshop paper for IEEE ASRU 2019, and can be found in Appendix B

which is defined as a period of time short enough for the user to not notice a significant delay before receiving a transcript, after finishing an utterance. This delay is set to approximately 500ms from the end of the utterance. This amount has been decided on, as half a second is short enough to not notice any disruptive delays, even in human-human interaction.

The robustness against noise of the new ASR pipeline will be tested and compared to Pepper's baseline ASR system. Because Pepper cannot deal with audio processed outside of Pepper, a series of three experiments was designed to tease apart the influence of changing the SSL algorithm (see 3.4.1), changing the ASR engine (see Section 3.4.2) and the addition of a beamformer (see Section 3.4.3) on the performance of the proposed pipeline. Especially the addition of a beamformer should allow for significant performance increases compared to the built-in ASR system without beamformer or single microphone solutions [18, 11]. Furthermore, an additional experiment has been performed to test the performance and usability of the system in a real, noisy environment (see Section 3.4.4).

2

Related Work

This chapter first explains the concept of automatic speech recognition. In the subsequent three sections, related research on ASR in social robots, ASR in noise and beamformers is presented.

The section on ASR in social robots relates to the first research question: *Can Pepper's built-in keyword spotter be replaced by an ASR system able to recognise continuous speech in Dutch?* The last three sections help provide the relevant background to the second research question: *"Can Pepper's ASR pipeline be made more robust against noise, without changing its hardware?"*

2.1. ASR

Automatic Speech Recognition (ASR) is the process of recognizing words from an audio stream containing speech. Research into ASR goes back many years, with one of the first ASR systems being HARPY [19]. This ASR system from 1976 could approximately understand 1000 isolated words. Since then the performance of ASR has dramatically improved, both in usability and performance, even leading to a Word Error Rate as low as 5.1% on continuous speech [5].

An ASR process generally works in several consecutive steps [20]. These steps are shown in Figure 2.1. The input of the process is an acoustic signal, digitally represented in sam-

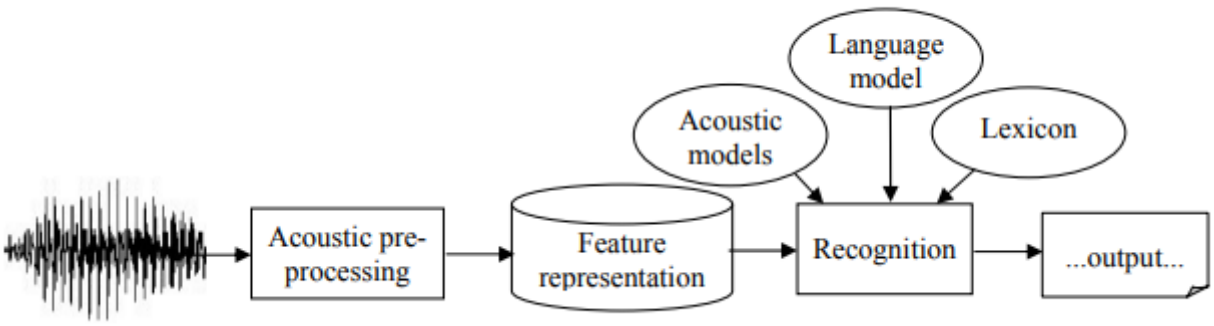


Figure 2.1: Visual representation of the steps performed in an ASR system, as described by O. Scharenborg [20].

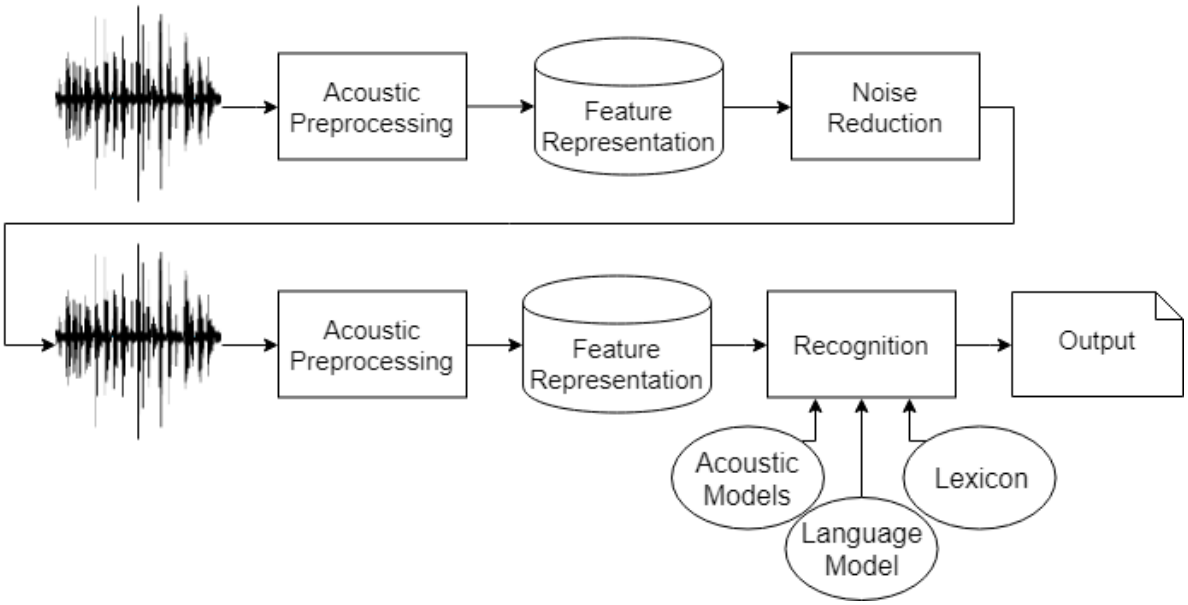


Figure 2.2: The proposed extention to an ASR pipeline, based on the visual representation by O. Scharenborg [20].

ples. Acoustic features are created from this acoustic signal by the acoustic preprocessing step. The recognition is performed using acoustic models, a language model and a lexicon. During the recognition, the created acoustic features are compared to sequences of acoustic models. These sequences of acoustic models represent words in the lexicon. The word, represented by the sequence of acoustic models which matches best with the created acoustic features, is hypothesised. Acoustic features is a broad definition, defining any representation of information contained in an acoustic signal. Often used features are Mel-Frequency Cepstral Coefficients (MFCC) [21] and Perceptual Linear Predictive (PLP) cepstral coefficients [22]. A more basic acoustic feature is the Power Spectrum which is created by the Fourier Transform. The power spectrum, which comprises of a frequency-domain representation of the signal, is used in my research to preprocess the data before performing the ASR.

The acoustic models and lexicon are crucial in the recognition process. The lexicon contains all words, constructed from smaller units, such as phonemes or syllables, which can be recognized by the system. The acoustic models are a representation of these smaller units, based on the acoustic features used by the system.

The language model, on the other hand, contains the linguistic information used to represent the probability of words succeeding each other, based on the language of the speech. These probabilities are often represented by bi- or N-grams, where a bigram, for example, depicts the following probability: $P(w_2|w_1) = \frac{P(w_1,w_2)}{P(w_1)}$.

Up until about a decade ago, many of the state-of-the-art ASR systems were mainly based on Hidden Markov Models (HMM) [23–25] in combination with Gaussian Mixture Models (GMM) to represent the acoustic models. Nowadays the use of GMMs to represent the acoustic models is surpassed by methods using Deep Neural Networks (DNN) [26, 27]. In 2012, Google already reported a decrease of Word Error Rate (WER) of approximately 5% when comparing a DNN acoustic model compared to their best HMM-GMM hybrid method [28].

Google Cloud Speech-to-Text (GC-STT) is used in this thesis (see section 3.1.4), which is nowadays based on a DNN-based acoustic model. Since GC-STT performs end-to-end transcription, it is not possible to change the acoustic models, or even which acoustic features are used. Therefore, the research in this thesis focuses on the steps performed to preprocess the audio before the acoustic features are created by Google Cloud Speech-to-Text. In Figure 2.2, three steps between the signal and acoustic preprocessing are added to the start of the pipeline from Figure 2.1. These three steps, shown at the top of Figure 2.2, would be the addition of another acoustic preprocessing step, followed by noise reduction or signal enhancement based on the acoustic features, followed by the transformation of the acoustic features to a signal once again as input for the Google Cloud Speech-to-Text.

2.2. ASR for social robots

Human-Robot interaction is an essential research area for social robots, given their goal to interact with humans as naturally as possible. The introduction already mentioned several possible scenarios in which a social robot can be used [1–4]. In some of these interactions, verbal communication with the robot is important. Roberto Pinillos, et al. [1] already mention the difficulty in using voice interaction a noisy hotel environment, recommending multi-modal input (i.e., using the robot's tablet and speech recognition) as solution. The speech recognition in the research performed by Masahiro Shiomi, et al. [2, 29] even made use of an operator to listen along to the interlocutor's input, to select the correct response of the robot. This was done due to "the difficulty of speech recognition in real environments" [2]. The abovementioned research shows the need for a noise-robust ASR system for social robots.

The need for noise-robustness already prompted Michiel de Jong, et al. [10] to use Google Cloud Speech-to-Text in their efforts to improve Pepper's perception. They propose using both Pepper's built-in ASR system, and GC-STT to improve the robustness of the speech recognition system. The measure used for their evaluation of the speech recognition systems is the amount of correctly recognized commands (in the form of sentences), instead of the transcription quality for continuous speech. They show a 4% increase in correctly recognized commands by using GC-STT instead of Pepper's built-in keyword spotter, and a 16% increase when using both systems. Their experiments are performed without noise, and furthermore, use the English language. The results for Dutch continuous speech, in noisy environments, is not evaluated as of yet.

Human-Robot interaction in which noise-robust speech recognition is used to decode the speech of possibly multiple speakers in real-time is called *Robot Audition*. For example, HARK [7] is an open-source Robot Audition framework which can be used for multiple different types of robots, and is specifically tested on the ASIMO, SIG2 and Robovie robots. While this could include Pepper, the system is evaluated using isolated words and to the best of my knowledge does not support Dutch. As such, it does not provide an environment which can be used for the research in this thesis. Its system consists of a full ASR pipeline from recording to transcription. The HARK software system contains several modules, which includes modules for sound localization using the MUSIC algorithm, and a Delay-and-Sum beamformer or Geometric Source Separation (GSS). These can be used on robots which have multiple microphones to perform sound separation and noise reduction (Sound source localization and beamforming are further explained in section 2.4). Furthermore, it contains modules for the extraction of acoustic features, creation of Missing

Feature Masks (MFM) and an ASR interface. The use of Missing Feature Masks is a method to create noise-robust ASR by identifying and removing noisy acoustic spectro-temporal regions (i.e., the power of a frequency on a specific time) [30].

The evaluation of HARK makes use of a pipeline consisting of the MUSIC sound source localization, GSS, and MFM only. Delay-and-Sum beamforming has not been evaluated. The evaluation of HARK was performed on the ASIMO robot in a room with artificial noise sources, i.e., music and speech from a 60-degree angle from the front of the robot. Isolated Word Recognition was performed while focusing the speech enhancement forwards, i.e., its Sound Source Localization turned off and steered forward statically. In my research, continuous speech will be used, which is more challenging to recognize than isolated words. Furthermore, the noise data used in the research in this thesis contains noise from more than a single direction, as is done in the evaluation of HARK. The measure of the amount of noise used in the evaluation of HARK is the "Target-to-Robot-Noise ratio", which is the Signal-to-Noise ratio (SNR) using the amount of noise, caused by, and recorded by the robot as reference. The pipeline used to evaluate the HARK system was shown to achieve a Word Error Rate (WER) of 4% to 11% without added speech or music noise, and 10% to 40% with added noise. When the raw, noisy speech signal from a single microphone was used as input, instead of the audio processed by the pipeline, a WER of 100% was found.

Furthermore, an experiment was performed to test HARK's performance in separating and recognizing isolated words from three speakers simultaneously. The experiment showed that the use of Sound Source Separation results in a WER of 25% to 50% for all speakers, depending on the separation between the speakers. The use of MFM further increased its performance, dropping the WER to between 20% to 35%.

Toshinori Ishi [8] created a noise-robust ASR system for a Robovie robot which also takes the age of speakers into account. To achieve this noise-robust system, a 12-microphone array is used with a Generalized Sidelobe Canceller (i.e., a type of beamformer) to perform the noise suppression in conjunction with a Minimum Mean Squared Error (MMSE) feature-space noise suppression, based on the optimal single-channel noise suppression, the Wiener Filter. Furthermore, separately trained acoustic models are used to process the speech of adults and children separately.

The system created by Ishi is firstly evaluated using recorded, short Japanese sentences with cafeteria noise mixed in. When looking at the noise-robustness of the system, each of its proposed components added to the robustness, where the MMSE noise suppression increased the performance in noise the most. Furthermore, the system is also evaluated in a real noisy environment (i.e., a cafeteria during lunchtime), and showed an average of 27% WER.

2.3. ASR in noise

Multiple methods exist to improve the performance of ASR in noisy situations. The main methods include training or designing an ASR for noisy situations, and preprocessing the audio before performing the ASR on it.

The first method, training or designing an ASR on noisy situations, is mostly based on training the acoustic model of the ASR on noisy speech. This method allows the ASR to recognize the utterances in noise, as the acoustic models are trained with noisy speech; therefore being able to match its acoustic model to the features created from the noisy input speech. For example, Seltzer et al. [24] investigated replacing acoustic models, based

on Guassian Mixture Models, with Deep Neural Networks. Seltzer et al. also shows multiple approaches to training this DNN-based acoustic model which help contribute to the noise robustness. This includes training with speech in several (noisy) conditions, using enhanced features, and performing noise-aware training. These methods all add to the noise robustness in their own way. Training with speech in noisy conditions makes it possible to find features that are invariant to noise. Performing feature enhancement can lower the variability of the features by enhancing distorted features. Noise-aware training makes use of adding an estimation of the noise signal to the acoustic models, allowing the DNN to learn about the relationship between clean speech and noise.

The second method involves the preprocessing of audio before sending it to the recognition engine. This preprocessing often entails performing noise reduction or speech enhancement on the input signal to create an as clean as possible speech signal. The noise reduction can be done in many ways. One way of categorizing noise reduction methods is by single- and multiple microphone methods.

Single microphone methods to noise reduction and speech enhancement are used frequently and are often based on Discrete Fourier Transforms [11]. These methods generally consider additive signal models. In additive signal models, noise and speech are assumed to be statistically independent. Since these models are assumed to be independent, both the noise and speech signals can be estimated, which in turn can be used to extract an estimated clean speech signal. One of these methods is the Wiener filter, which is the optimal filter for single-channel noise reduction [11, 31]. It is often found though, that single-channel noise reduction techniques, including the Wiener filter, can add much distortion and artifacts to the speech signal. These distortions and artifacts can negatively influence ASR systems which are not explicitly trained or designed to handle distorted audio.

Multiple microphone methods for noise reduction are often (based on) beamformers. Beamformers make use of spatial data which can be acquired by using the delays between the reception of sound at each microphone. These methods can be combined with single-microphone methods for additional noise reduction, as a beamformer creates single-channel speech signal from the multi-channel input speech [11]. More information on beamformers is given in the next section, 2.4.

2.4. Beamformers

Beamformers are used for the improvement of many different signals, from radar [32, 33] to ultrasound imaging [34, 35]. Beamformers make use of arrays of receivers, in this case, microphones, to attenuate noise. Spatial data can be extracted using the Time Delay of Arrival (TDOA), i.e., the delay between the reception of the audio on the different microphones. In general, beamformers perform noise reduction by changing the phase of audio received on each microphone with respect to the TDOA, such that audio from a specified direction is in phase for all microphones.

The TDOA is often represented by the Direction of Arrival (DOA). This representation uses the assumption that the sound waves arrive in planar waves from a specific direction.

To find the TDOA's, a Sound Source Localization (SSL) algorithm is frequently used [36]. These algorithms make use of the received signals and knowledge of the microphone array layout to give an estimation of the location or DOA of the signal. Several SSL methods include the Steered-Response-Power Phase-Transform (SRP-PHAT) [37], Generalized Cross-Correlation (GCC-PHAT) [37] and subspace methods such as Multiple Signal Classification

(MUSIC) [14, 15, 36]. Especially the latter of these is interesting, given that it allows for the extraction of multiple sound sources simultaneously.

Using the TDOA's retrieved by an SSL algorithm, a beamformer can automatically steer in the estimated direction of the sound source. The steering of a beamformer is done with so-called *steering vectors*. These steering vectors contain the phase shift that has to be performed by the beamformer to steer in the estimated direction, attenuating sound from other directions than the one steered towards. Since the delays can rarely be represented by the number of samples, the phase shift is most often performed in the frequency domain, which is acquired by, e.g., a Fourier Transform. The SSL works together closely with the beamformer, i.e., providing the desired beam direction and its corresponding steering vectors.

The phase shifts are expressed in a complex value $e^{-j*2\pi*k*\tau}$, with k representing the frequency and τ the delay to the microphone from a reference point. By multiplying this value with the frequency-domain value of the signal, the phase of this signal is shifted. Since speech is a broadband signal, this multiplication has to be performed for each frequency bin.

Beamformers are often separated into two categories: *Conventional* and *Adaptive*. These categories are based on how the weights of each channel in the beamformer are set. These weights depict the ratio in which the signals received on each microphone should be mixed together in the final beamformed signal. Where a conventional beamformer has set weights that do not change automatically based on the signal, an adaptive beamformer adjusts its weights based on the data it receives.

Adaptive beamformers include for example the Minimum Variance Distortionless Response (MVDR) or Capon beamformer [17] and the Generalized Sidelobe Canceller [38, 17]. These beamformers often perform more complex calculations and optimization problems, which makes real-time processing more difficult. The performance of adaptive beamformers is generally better, though the more complex calculation and optimizations could cause delays in the pipeline, before the speech can be processed by the ASR.

The research in this thesis uses a conventional beamformer, specifically, the Delay-and-Sum beamformer [38]. This conventional beamformer has the weights of all microphones set to a specific value, generally to $\frac{1}{M}$, where M is the number of microphones.

A visual representation of the workings of the Delay-and-Sum beamformer is given in Figure 2.3 [1]. It shifts the phase of each microphone's received signal based on the calculated steering vectors to bring each channel in phase with each other. The figure shows the in-phase signals (top of Fig. 2.3) to be stronger, though this is not precisely the case. The signals, right before the summing step, are divided by the number of microphones and summed together in a ratio which regains the average amplitude of the input signals. Therefore, the maximum amplitude of the individual signals is approximately the same as the output signal of the summing step. The individual out-of-phase signals (bottom of Fig. 2.3) will be divided by the number of microphones as well, though as these are not superimposed, they will remain divided in the output signal, effectively attenuating the noise. The implementation of a Delay-and-Sum beamformer is explained in section 4.5.

[1] Lab Book Pages by Dr. A. Greensted:
`http://www.labbookpages.co.uk/audio/beamforming/delaySum.html`

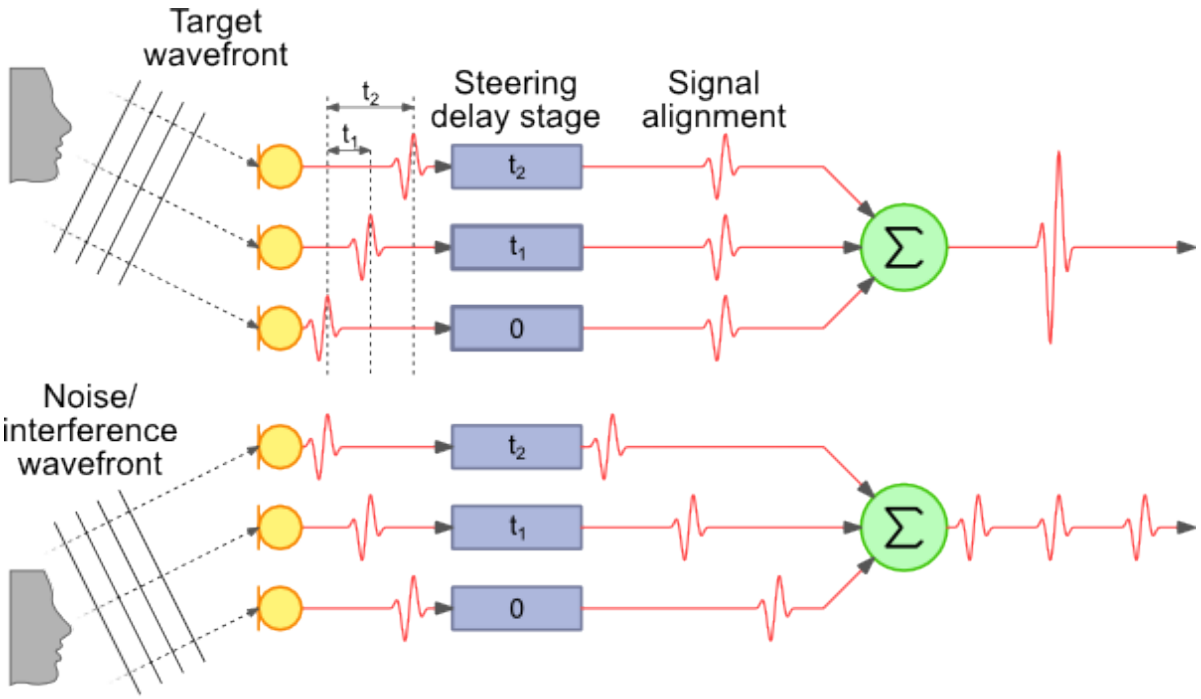


Figure 2.3: The working of a Delay-and-Sum beamformer. The top shows a simplified speech signal being steered to be in-phase, whereas the bottom shows interference from a different angle being attenuated due to being brought out-of-phase. Image used with permission from The Lab Book Pages[1]

.

3

Method

To answer the research questions, a new ASR pipeline has been created for Pepper as described in section 3.1, in which all the separate parts of the pipeline and how these have been used is described. To evaluate the performance of the new ASR pipeline, speech data had to be acquired using Pepper's microphones as a recording device. Noise data was recorded as well in order to test the newly proposed ASR pipeline in the presence of noise. The speech and noise data is acquired in-house and described in Sections 3.2 and 3.3, respectively. The setup of the four performed experiments is explained in Section 3.4.

3.1. The Proposed Pipeline

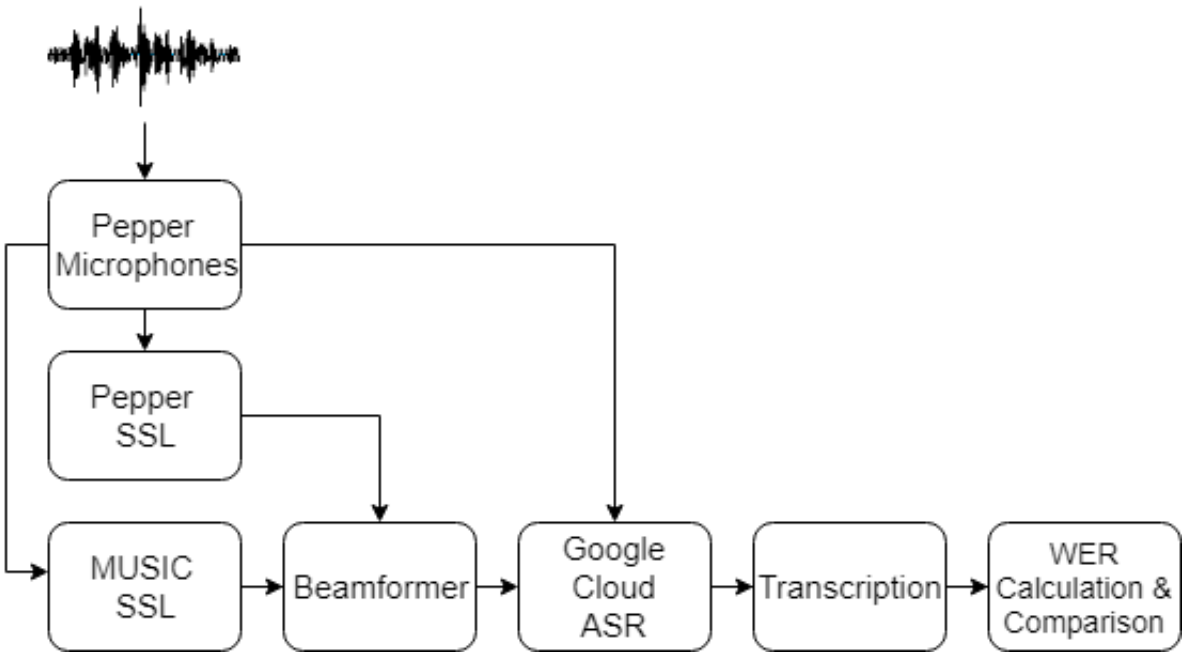


Figure 3.1: The bottom flow shows the proposed pipeline, which is compared to the pipeline using Pepper's SSL (middle flow), and the pipeline without beamformer (top flow) in Exp. 3.

The bottom flow of Figure 3.1 shows the proposed pipeline: The audio is recorded using Pepper's built-in microphones. The audio then passes through the MUSIC SSL (see Sec-

tion 3.1.2) and the proposed Delay-and-Sum beamformer (see Section 3.1.3), after which the beamformed speech signal is sent to the Google Cloud Speech-to-Text (see Section 3.1.4) for recognition. Furthermore, two baseline systems for Pepper (see Section 3.1.1) have been used, which are the unbeamformed system (the top flow of Fig. 3.1), and a pipeline based on Pepper's built-in SSL and the Delay-and-Sum beamformer. Each part of proposed system has been compared to the baseline systems in multiple experiments (see Section 3.4).

3.1.1. Pepper

Pepper is a social robot, often used for welcoming and informing people in companies or at events[1]. An image of Pepper is shown in Figure 3.2. Pepper has four built-in microphones, located on top of its head, in a rectangular or elliptical array. Figure 3.3 shows the exact layout of the microphones in the Pepper robot used in this study, where each black dot represents a microphone, and where the front of Pepper's head is facing downwards. The microphones are separated 6.7 cm from one another on the left-right axis and 5.8 cm on the front-back axis. Located below the microphones is a fan for cooling the electronics. This fan adds significant noise to the speech recordings.

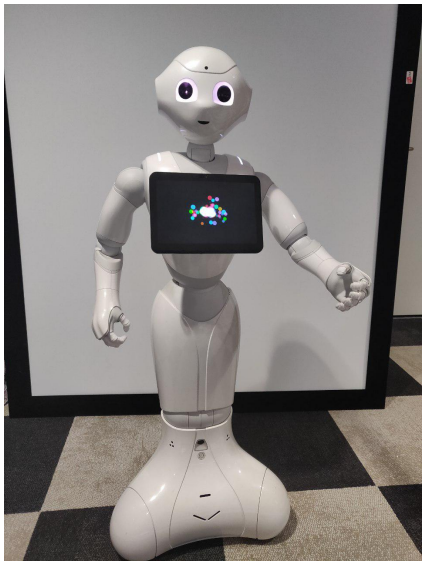


Figure 3.2: The Pepper robot

Pepper's built-in sound source localization algorithm estimates the direction of arrival (DOA) of the loudest noise it receives with about 10 degrees precision[2]. Pepper's SSL, however, cannot distinguish speech from noise. It loses reliability in noisier environments, and when multiple sound sources are received only the location of the loudest sound source will be stored.

Pepper's built-in dialogue system takes the speech recorded by Pepper's microphones as input and uses keyword spotting to parse the speech input and decide on a reply. Pepper's keyword spotter is based on the NUANCE ASR system[3]. Keyword spotting is available for all languages supported by Pepper, whereas full transcription is only possible with a limited number of languages, such as English and Japanese.

3.1.2. The MUSIC Sound Source Localization Algorithm

The MUSIC algorithm [14, 15, 36] (obtained from the Python library *PyRoomAcoustics* [39]) was chosen for the proposed pipeline since it explicitly considers noise to be part of its model [36]; moreover, it allows for a high-resolution estimation of the sound source location.

The MUSIC algorithm is a subspace method which searches for DOAs intersecting the subspace of signals of interest [36]. The MUSIC algorithm performs eigendecomposition on the covariance matrix of the received signal in the frequency-domain. By finding peaks in its power, multiple sound sources can be located, including noise. The highest eigen-

[1] `https://www.softbankrobotics.com/us/pepper`
[2] Documentation for Pepper can be found here: `http://doc.aldebaran.com/2-5/`
[3] The use of NUANCE is shown on: http://doc.aldebaran.com/2-5/

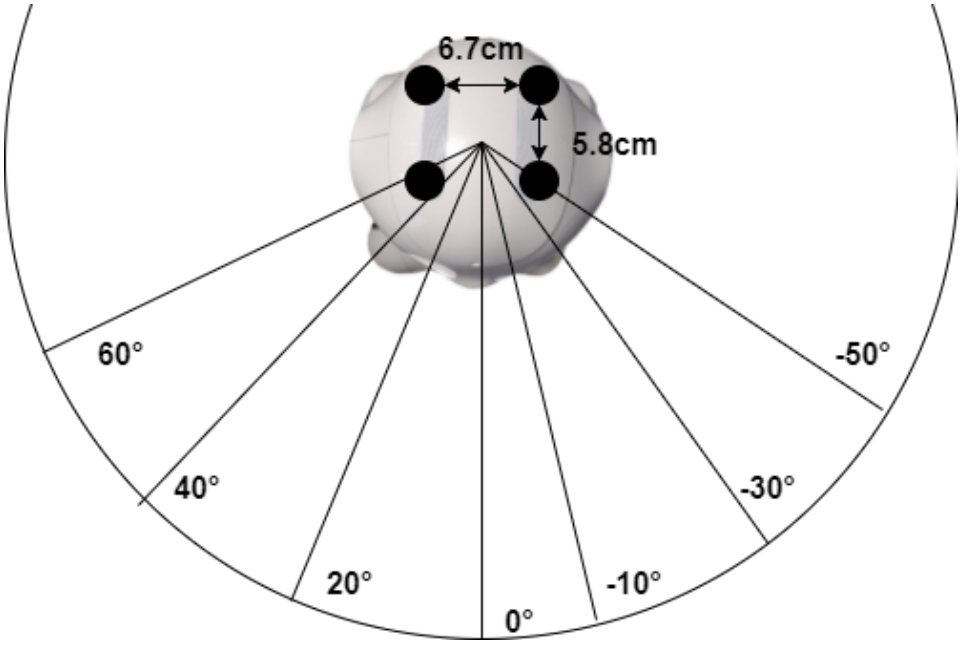


Figure 3.3: Top view of Pepper's head with the microphone locations shown as black dots, and the angles used in Exp. 1. Pepper faces downwards.

values found generally correspond with the speech signal subspace, whereas the lower eigenvalues are in the noise subspace. The subspaces are formed by dimensionality reduction based on Principle Component Analysis (PCA) [12]. By projecting a sample on the signal subspace and not on the noise subspace, the noise from the sample is partially discarded. We focus on a single speaker; therefore, only the first eigenvector is used to locate the source. The system can be extended to accommodate multiple speakers, by using the n highest eigenvalues and corresponding eigenvectors, where n is equal to the number of speakers.

Each of the possible source locations has its own set of precomputed (to reduce computation time) steering vectors. These steering vectors do not change as long as the microphone array stays the same, because the respective delays between the microphones remain the same for each direction of arrival.

The MUSIC algorithm is computationally heavy; however, when used within predefined limits regarding listening angle, resolution, frequency range, and the frequency with which the algorithm runs, it can be run in pseudo-real-time.

The limits set to MUSIC for these experiments are the following:
A maximum scanning range for sound source locations of 120 deg centered in the forward direction. Pepper always tries looking at the person it is conversing with, therefore this range should be large enough to catch any discrepancies in Pepper's look direction.

The angular resolution is the distance between sound source locations around the reference point of the microphone array, expressed in degrees. Within the 120 degree range, 73 possible sound source locations are looked into by MUSIC, which results in an angular resolution of 1.6 deg. A higher angular resolution allows for higher precision of the MUSIC algorithm in the steering test, as with the chosen resolution, it will check every 1.6 degrees for a sound source. During exploratory experimentation, 73 sound source locations was found to allow the pipeline to run in pseudo-real-time (this was later found not to be true

for live speech, see Section 5.5), with the highest precision possible. A higher precision in SSL estimations also means a beamformer can steer its beam more precisely.

MUSIC works on broadband signals by searching for peaks in the spatial spectrum over multiple frequency bins. The range of frequencies MUSIC takes into account can be limited to block out unwanted frequencies or to focus on a smaller subset. The frequency range that is used for MUSIC is 300-2000 Hz, with a frequency bin size of 62.5 Hz (due to the number of samples per processed speech chunk). This results in $(2000-250)/62.5 = 28$ frequency bins being taken into account. A range from 250-2000 has been chosen since the most essential and audible speech frequencies are below 3000 Hz. However, between 2000 and 3000 Hz the fan adds too much noise relative to the speech intensity at these frequencies to provide a correct estimation. Therefore this range has been left out.

3.1.3. The Delay-and-Sum Beamformer

The beamformer takes Pepper's audio and the estimated source location from the MUSIC algorithm and outputs a beamformed speech signal. For the experiments, a Delay-and-Sum beamformer [40, 17] was chosen, as this type of beamformer requires a low amount of processing power, which is vital for allowing the system to run as close to real-time as possible, while still being able to perform proper noise attenuation [41].

A Delay-and-Sum beamformer makes use of phase shifting based on the Time Difference of Arrival (TDOA) between the microphones to attenuate noise. This time difference can be applied to the individual channels, with respect to the center of the microphone array or a reference microphone. Delaying (i.e., the phase shift) is performed in the frequency domain since the TDOA in many cases is not equal to a real number of samples.

The frequency domain is obtained using the Short-Time Fourier Transform (STFT). The audio signal is segmented in segments of n samples for each channel using a Hanning window. Each segment $y(t)$ is transformed to $y_k(t)$, which contains multiple frequency bins. Shifting the phase of the speech signal while in the frequency domain has been done, as explained in 2.4, by multiplying each frequency bin for each microphone by $e^{-j*2\pi*k*\tau}$. After multiplying, the Delay-and-Sum beamformer sums the audio of each microphone together in pre-set (often equal) amounts. When summing the audio together, the signals that are in phase will retain their amplitude (see Section 2.4). Out-of-phase signals, like noise from a different direction than the beamformer is steered in, also gets summed. But since the peaks of out-of-phase signals do not overlap, summing these causes attenuation of these peaks.

The Delay-and-Sum beamformer does not adapt its weights for each channel automatically based on the received signal, as it is not an adaptive beamformer. Since Pepper will always focus its head on the person engaging it, and the noise generated by the fans is stronger near the rear microphones, the front microphones are given a weight of 0.3 and the rear microphones 0.2. These values have been found empirically during exploratory experiments.

After summing the signals together, a single channel audio signal in the frequency domain is obtained. This signal is reverted to the time-domain by the Inverse Short Time Fourier Transform (ISTFT). The single-channel signal resulting from this process is the beamformed speech signal, which if done correctly, has attenuated the noise from the original signal.

3.1.4. Google Cloud Speech-to-Text

The Google Cloud Speech-to-Text (GC-STT) takes either the raw speech signal from Pepper, or the beamformed speech signal as input and transcribes the speech into text. The Google Cloud Speech-to-Text is based on deep neural networks. It fully supports Dutch and can transcribe real-time streams, which is crucial for dialogues with social robots. The GC-STT is used in two ways: as an ASR that mimics a keyword spotter in Experiment 2 and a standard ASR in Experiment 3.

Keyword spotting, for Experiment 2, is mimicked by providing so-called *phrase hints* to improve the confidence scores of the provided phrases, which increases the chance of correctly recognizing the word. In Experiment 3, GC-STT is used in its standard capacity, i.e., the word-by-word transcription of a continuous speech signal.

The input to the GC-STT is a stream of Pulse Code Modulation (PCM) data, which is a digital representation of audio where the amplitude is sampled and interpolated to the nearest integer within its sample size. More specifically, 16-bit linear PCM has been used. This representation was chosen since it is uncompressed, and because Pepper also uses this representation as output for its microphones. Since it is uncompressed, the signal can be used directly for (pre)processing. The input stream for GC-STT can have a maximum length of 65 seconds due to limitations in GC-STT. Therefore each recording in the speech database has been streamed separately, before setting up a new connection to GC-STT.

Google Cloud Speech-to-Text is said to be robust against noise[4]. Preprocessing for noise reduction is not advised as it might increase the distortion of the signal[5], which could decrease the performance of the ASR engine.

Google's STT only uses a single channel in its current STT. The option for GC-STT to perform recognition on multi-channel audio specifies that multi-channel input is used in case each channel contains a different speaker, such as the two sides of a telephone conversation [6]. This implies GC-STT's acoustic models are not trained for multi-channel speech from a microphone array. As Pepper does record multiple channels, preprocessing the speech with a beamformer creates an enhanced single-channel signal, without losing any data GC-STT would require for recognition.

It is important that all processing to the signal happens without adding significant distortion. Distortion in the audio has a high chance of negatively impacting the ASR performance if the acoustic model is not trained or designed to handle this.

3.2. Speech Data

Three dedicated datasets were recorded to evaluate the proposed pipeline. Two of these have been recorded in a sound-proofed booth. Each dataset consisted of recordings of a single scripted dialogue spoken by Dutch native talkers. All talkers were recruited from the Faculty of Electrical Engineering, Mathematics, and Computer Science (EEMCS) of the Delft University of Technology, the Netherlands. They participated for free, and are native speakers of Dutch. All participants signed an informed consent form prior to the record-

[4]GC-STT's Noise-robustness is claimed on their product page: `https://cloud.google.com/speech-to-text/`

[5]Preprocessing against noise is discouraged in the GC-STT documentation: https://cloud.google.com/speech-to-text/docs/best-practices

[6]GC-STT description of multiple channel audio transcription can be found here: `https://cloud.google.com/speech-to-text/docs/multi-channel`

ings.

The dialogue consisted of 50 Dutch sentences and can be found in Appendix A. It was constructed to contain different types of phrases and utterances that could be used in different kinds of conversations with a Pepper robot, including short and long phrases, homonyms, large numbers, and dates. Moreover, a monologue, in the form of a short story, was included to test longer sentences and to investigate how the ASR would respond to phrases with ambiguous contexts. The monologue made up 12 out of the 50 sentences of the full dialogue.

The first dataset referred to as the "steering" dataset and used in Experiment 1 (see Section 3.4.1), consisted of recordings made by 8 talkers (5 males and 3 females, age range: 20 - 30 years). These recordings have not been made inside a sound-proofed booth, as this booth was relatively small. Instead it was performed in a large room (approx. 8x8 meters) in which background noise was kept to a minimum. Each participant recorded five sentences from the dialogue. For each recording, Pepper was placed 1 meter in front of the participant, to ensure constant and comfortable interpersonal distance, at each of seven different angles, which are shown in Figure 3.3. The chosen angles had a 20 degree interval except for 0 degrees and -10 degrees, since the symmetric microphone array gives the same results for each side of the robot, on equal angles. All actuators were turned off during the recording to avoid additional noise and to ensure Pepper did not move its head. Each recording from each angle took about 15 seconds.

The second dataset is referred to as the "dialogue" dataset and used in Experiments 2 and 3 (see Sections 3.4.2 and 3.4.3). This dataset consisted of recordings made by 9 participants (7 males and 2 females, age range: 20 - 30 years). Five of these participants were also recorded for the steering dataset and returned for a second recording session for the dialogue dataset. Moreover, 4 additional talkers were recorded. The recording of this dataset was performed in the sound proof booth, as these recordings only were performed in a single angle, therefore not much space is required. This booth was approximately 1 by 2 meter in size. During the recordings, Pepper was placed 1 meter in front of the participant at an approximately 0-degree angle (i.e. looking straight at the participant). Each recording of the 50 sentences in the dialogue took about 6 minutes and resulted in approximately 3 minutes of actual speech data. The other three minutes were taken up by Pepper speaking and large pauses between utterances. The latter three minutes are not included in the dataset.

During the recording of the "dialogue" dataset, the results from Pepper's keyword spotter are generated as well, because Pepper's keyword spotter cannot be evaluated using prerecorded speech. Pepper's dialogue system, which allows for a scripted dialogue between Pepper and a person, makes use of the built-in keyword spotter to decide on which reply to give. During these recordings, the dialogue system always replied with the sentence as defined in the dialogue, not depending on if the keyword was spotted correctly or not.

The third dataset referred to as the "Pepper SSL dialogue" dataset is used in the first part of Experiment 3. These recordings were made following the same procedures as the "dialogue" dataset, with the only exception that the estimations made by Pepper's SSL were also stored. Four participants (3 males and 1 female, age range: 20 - 30 years), which also have taken part in recording the "dialogue" dataset, have taken part in these recordings. The recordings for the third dataset have been made since the estimations from Pepper's SSL were not stored during the recording of the "dialogue" dataset by mistake.

During all recordings, the talkers were instructed to speak as they normally would in a conversation. Due to Pepper's dialogue engine replying overly fast after a short pause in speech, participants were instructed not to pause within a sentence. To support the participants with speaking without pauses within a sentence, punctuation such as commas have been avoided in the written text of the dialogue. Some commas did remain in the dialogue, though these were left to make the sentence more clear to avoid confusion (and therefore mistakes) while reading aloud the dialogue. Furthermore, sentences between which a pause was necessary (i.e., to split up the speech recording in separate files, and for the dialogue system to correctly recognize when to listen for the next keyword), were placed on a new line with a dash in front of them. Participants were informed about where to pause and where not, prior to the recording.

The recordings were manually cut at positive going zero-crossing into one-sentence fragments using Praat [42], leaving approximately 500 ms of preceding and trailing silence. Loud noises, such as beeps played by Pepper, that fell within the 500ms window were excluded from the 500 ms window. Pepper's speech was also removed. Each audio fragment has been normalized to 70 dB to ensure the Signal-to-Noise Ratio (SNR) can be more precisely determined in tests with additive noise.

3.3. Noise Data

Experiment 3 (see Section 3.4.3) will test the ASR pipeline's performance in noise at different SNRs, using the recordings from the "dialogue" dataset. For this experiment, noise in different SNRs needs to be mixed into the speech recording. Therefore, several minutes of cafeteria noise were recorded in the EEMCS faculty cafeteria using Pepper. To ensure that the beamformer spatial filtering could be adequately evaluated, it was made sure that most of the noise came from behind and from the sides of the robot. Noise coming from the direction Pepper is facing would require a different approach than a beamformer. Stretches of noise louder than 72 dB and silent segments were manually removed at positive-going zero-crossings to ensure a relatively stable noise level. The noise signal was normalized to 70 dB after which randomly picked stretches of the noise were automatically mixed with the speech signal at four different SNRs, i.e., 8 dB, 4 dB, 0 dB, and -4 dB, using a custom-made Praat script. Two hundred ms of preceding and trailing noise was added (in addition to the preceding and trailing silence in the speech fragments).

3.4. Experimental Set-up

To evaluate the proposed pipeline, four experiments were carried out. The first experiment investigated the performance of the SSL systems, comparing Pepper's SSL versus the MUSIC SSL. The second experiment investigated the difference in the performance of the recognition engine in a keyword spotting task, performed by Pepper's keyword spotter and Google Cloud STT. The third experiment compared the recognition performance of Google Cloud STT on the raw audio received from Pepper, a pipeline containing using Pepper's built-in SSL, and the proposed pipeline. The proposed pipeline in the third experiment used MUSIC as SSL algorithm. Comparisons to Pepper's built-in SSL as baseline have only been performed with clean speech, however the robustness of the proposed pipeline is evaluated in noisy speech. The fourth experiment, the Robot-in-the-wild experiment, was performed to evaluate the performance of the proposed pipeline in real noisy situations.

Together, these experiments can be used to evaluate the performance of the built-in and the proposed ASR pipeline, as the influence of the ASR engine, the SSL, and the beamformer have been evaluated, both separately and together, by the different experiments. All experiments have been approved by the Ethical Committee.

3.4.1. Experiment 1: Sound Source Localization Algorithm

A beamformer makes use of steering vectors to define how to delay the received speech signal, in order to emphasise the signal from a set direction. When this direction is not constantly the same, a beamformer can make use of an SSL to estimate the direction for which steering vectors should be created. An error in the estimation of the location of the sound source causes a beamformer to apply wrong delays, which causes the target speech signals to be summed out-of-phase. This error will attenuate the target speech signal and could amplify noise, instead of vice versa. Therefore the first experiment evaluates the SSL algorithm of both pipelines, i.e., Pepper's built-in SSL algorithm and the MUSIC sound source localization.

The sound source localization algorithms were evaluated in terms of the DOA in degrees. To that end, the DOAs at corresponding timestamps of Pepper's SSL and of the MUSIC SSL were compared. The Root Mean Squared Error (RMSE) between the estimated angle and the ground truth angle, which is the angle at which the recording was made, was calculated. The RMSE is calculated by the following formula:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(H-R)^2}{n}}$$

In this formula, H is the hypothesised value (i.e., the estimated DOA), R is the reference value (i.e., the actual DOA), and n the number of samples. By using RMSE as a metric, significant steering errors will have a bigger influence on the result. This measure has been chosen since significant steering errors also cause a beamformer to perform worse due to wrong steering vectors.

Given Pepper's default behavior to look at its interlocutor during a conversation, it is assumed that the person speaking to Pepper is always in front of Pepper. Given this assumption, the limits within which MUSIC SSL listened have been set to 60 degrees to each side. These limits are used to lower the number of sound source locations the algorithm has to compare, allowing the MUSIC algorithm to be used in (pseudo-)real-time. Having these limits also prevents the SSL from focusing on the fan in Pepper's head, decreasing the influence of the noise generated by the fan.

3.4.2. Experiment 2: Keyword Spotter

Pepper's built-in ASR system is a keyword spotter, while the Google Cloud system is a continuous speech recognition system. In order to investigate the performance of the Google Cloud's ASR system compared to the baseline Pepper keyword spotter, the second experiment evaluates the recognition engines, Pepper vs. GC-STT, on a keyword spotting task using the raw speech signal. This means no beamformer has been used to process the audio, which is done to ensure an equal test between keyword spotters, given Pepper's ASR system does not allow recorded audio data as input. In this task, the audio recorded by Pepper is passed both through Pepper's built-in keyword spotter and through the GC-STT,

which operates in 'keyword spotter' mode using 'phrase hints'[7].

Since Pepper's dialogue function and the keyword spotter can only be used with live audio, Pepper's keyword spotter was tested during the recordings as described in Section 3.2. Subsequently, to test the keyword spotter based on GC-STT (as in Section 3.1.4), the recordings were streamed to GC-STT. Each transcription created by GC-STT was checked for the keywords it should contain, and the keywords were marked correct if they are found in the transcription.

Performance was measured in terms of the number of correctly recognized keywords or the Keyword Error Rate (KWER), $KWER = \frac{\texttt{incorrect keywords}}{\texttt{total keywords}}\%$.

3.4.3. Experiment 3: Speech-to-Text Transcription

Experiment 3 evaluates the Delay-and-Sum beamformer and MUSIC SSL on a Speech-to-Text transcription task, and its performance in noise. To that end, the beamformed speech signal from the proposed pipeline is sent to GC-STT for transcription, and compared to the transcript made by GC-STT of the unbeamformed audio. Two versions of the beamformed speech signal were compared to the unbeamformed speech signal. Prior to the Delay-and-Sum beamformer, the speech signal was passed: 1) through Pepper's built-in SSL (as this SSL outperformed the MUSIC SSL in the steering test, see Section 5.1); 2) through the proposed MUSIC SSL.

The "Pepper SSL dialogue" and "dialogue" datasets are used for the third experiment. For the first part of the experiment, the recordings from "Pepper SSL dialogue" were used with only clean speech. The second part of the experiment, which tests the noise-robustness of the proposed pipeline, uses the "dialogue" dataset, including additive noise from the "noise" dataset.

The three systems are shown in Figure 3.1. The recordings made for the "Pepper SSL dialogue" dataset were processed through either of the three pipelines. Recordings made for the "dialogue" dataset were processed through the pipeline without beamformer and the pipeline containing MUSIC SSL.

The top two flows of Figure 3.1 show the pipelines for the baseline system. The upper flow is the baseline without any preprocessing, where speech recorded by Pepper's microphones is directly passed to GC-STT. Here the speech signal is not processed apart from normalization, and directly sent to GC-STT. The normalization is required as speech received by the microphones appeared to not be loud enough for GC-STT to reliably give a transcription. Note, increasing the gain of the microphones added significant noise to the recordings. The lack of loudness only seemed to appear when recordings instead of live audio were used. Normalization was done using the SOX audio-processing library on the recorded audio fragments. This baseline system is referred to as the raw speech signal system.

The middle flow is the baseline using Pepper's SSL instead of MUSIC SSL, before the speech data and estimated sound source location are sent to the Delay-and-Sum beamformer. Since Pepper's pipeline does not work with recorded speech, the estimations from Pepper's SSL stored together with the recorded speech were streamed to the beamformer. In this manner, Pepper's SSL estimations can be used by the beamformer without using live speech.

[7] Phrase hints (nowadays referred to as "Speech Adaptation" by Google) are described in GC-STT's documentation: `https://cloud.google.com/speech-to-text/docs/context-strength`

The speech data is sent to GC-STT, which then provides a transcription of the speech signal. The resulting WER is then used as a baseline comparison for the proposed pipeline.

The bottom pipeline (see Fig. 3.1) is the full proposed pipeline. The recorded audio is processed by MUSIC SSL, which estimates the location of the most probable sound source. This estimation is sent to the beamformer, together with speech signal. The beamformer creates a single channel speech signal which is sent to GC-STT to be transcribed.

The performance of GC-STT on the raw speech signal and the beamformed speech signals were evaluated in terms of Word Error Rate (WER) in relation to the ground truth, i.e., the written text of the dialogues. Evaluation was done for each of the noise levels separately. The WER is given by $\texttt{WER} = \frac{S+D+I}{N} * 100\%$, where S, D, and I respectively are substituted, deleted, and inserted words, and N the total number of words in the ground truth. The comparison with the ground truth is made automatically using the python library *asr-evaluation*[8], which calculates the WER automatically between two text files. After this calculation, the comparisons have been double-checked to remove errors such as comparing "19" with "nineteen", and the WER has been adjusted accordingly.

3.4.4. Experiment 4: Robot-in-the-wild

The fourth experiment is the Robot-in-the-wild test. This experiment was conducted to test the practical usage of the pipeline in a real dialogue setting. To achieve a proper evaluation of the practical use, a transcription test is done using the full pipeline, including MUSIC SSL, the beamformer, and GC-STT. This test is performed in the cafeteria of the auditorium of the university.

Only four persons participated, as the test was performed during the summer break. The experiment, furthermore, could only be performed during the lunch break. At other times than the lunch breaks, there is no background noise at this location, which is vital to this experiment. The four participants were of varying ages, between 20 and 60 years old, but only males participated. Each test took around 10 minutes, excluding time to explain the experiment.

Pepper was set up aimed forward at the participant. In this experiment, the actuators have not been turned off. This has been done to make use of Pepper as it would be in a regular usage scenario, where Pepper's actuators would be turned on. Due to this, actuator noise was still included in the unprocessed audio data. The activated actuators include the wheels. Pepper will not automatically follow persons, but it will turn around in the same location if it thinks the audio comes from behind. This action, including the movement of the head, does cause substantial differences in spatial data compared to being still. Leaving the actuators on, and performing this experiment in a real, noisy environment results in a total of three different factors compared to Experiment 3; the moving actuators, a different location (i.e., different background noise) and constantly changing spatial data.

During the Robot-in-the-wild experiment, a participant was asked to read the dialogue, used in the recordings of the "dialogue" dataset, aloud. By following the same dialogue as used in the recordings for the previous experiments, the results from the the Robot-in-the-wild experiment can be compared to the results from the third experiment, although the differences between the setups (i.e., different location, moving actuators and constantly changing spatial data) should be taken into account as well.

Pepper's dialogue system is used in this experiment to perform the dialogue, as was

[8]The asr-evaluation tool can be found here: `https://github.com/belambert/asr-evaluation`

done during the recording of the "dialogue" dataset (see 3.2). During this experiment, recording is stopped when Pepper speaks to circumvent the maximum recognition time of GC-STT, and to avoid recognition of Pepper's own voice. Pepper starts speaking when the dialogue system (which uses Pepper's keyword spotter) recognizes a keyword as defined in the dialogue. When Pepper has finished speaking, the recording is automatically started again.

Pepper's dialogue system can be adjusted to respond to text as input, instead of by using its keyword spotter. This makes it possible to use the transcriptions made by GC-STT to perform the dialogue. This has not been done during this experiment, as end-point detection would have to be added to the pipeline, to recognize when a speaker is finished with their utterance. Without end-point detection, GC-STT waits for a significant amount of time before returning its final transcription (if its input stream is not closed), making it unusable to directly replace Pepper's keyword spotter by GC-STT.

As real-time is an essential aspect of the pipeline, this has also been taken into account during the Robot-in-the-wild test. This has been done by measuring the time between the start of the pipeline, the moment the last speech data is sent to GC-STT, and the time the transcription is returned by GC-STT. Due to the low number of participants, and the problems that arose during the experiment (which is further explained in Section 5.4), this data was not properly gathered during this experiment.

4

Implementation

This chapter explains the implementation of the system. First, an overview of the implementation of the overall system is given, followed by a detailed description of the implementations of streaming and the beamformer.

4.1. Overall System

A combination of Python 2.7 and Python 3.6 is used to implement the system. This combination has been chosen since Pepper only works with Python 2.7 on the on-board system, and Python 3.6 being the newest stable version of Python at the start of this project. Combining the two versions is possible since the code written in Python 2.7 is only run on the robot itself, and the socket connection (see Section 4.3) serves as a bridging component between the two versions.

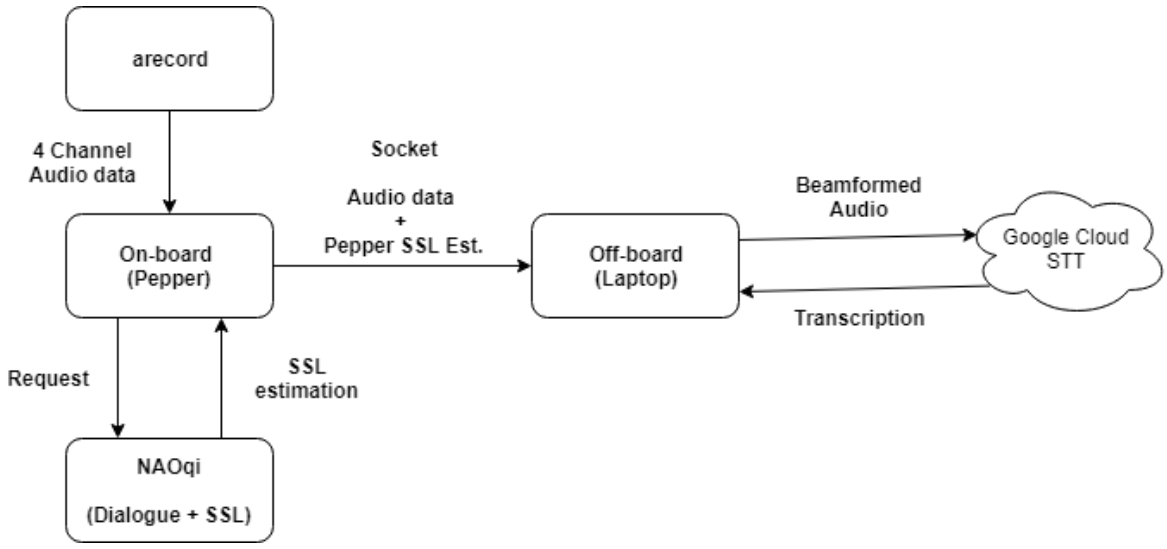


Figure 4.1: The components of the overall system, and the data sent between these components.

Figure 4.1 shows the separate components of the overall system with the data sent between these components. The system consists of two separate programs which communicate with each other. These are, firstly, the code running on-board Pepper on the left side

of the figure. Secondly, there is the code which runs on a separate instance, in this case a laptop, shown in the middle of the figure.

The on-board code performs the recording of the audio using Pepper's microphones using "arecord" and sends the data to the off-board code. Moreover, the on-board code makes calls to Pepper's API, NAOqi[1], which performs the keyword spotting task and SSL. The off-board code receives the raw audio data sent by the on-board code, after which it is processed by the proposed MUSIC SSL and Delay-and-Sum beamformer. The off-board code then streams the processed audio towards the Google Cloud services, which returns the transcription.

The code was separated into on-board and off-board for multiple reasons. It is possible to run code on Pepper's on-board system, though access to its ASR pipeline is still restricted. This means preprocessing would be possible on the on-board system, though it could limit the performance of the proposed ASR pipeline. While Pepper's hardware would allow for some complex computations, a full-scale audio processing pipeline including large matrix multiplications, would push the limits of its 4 GB of memory. This would especially be the case if the pipeline would be extended with a more computationally expensive beamformer or SSL. Moreover, it was decided to do most processing off-board to keep the off-board code as generalized as possible, making it possible to extend the proposed pipeline to cater for other, similar robots. Lastly, by separating the speech processing from Pepper's on-board system, testing and debugging code is easier and does not require a Pepper during the development.

4.2. Pepper On-board

The main focus of the on-board code is the recording of the audio. Furthermore, it communicates with Pepper's built-in NAOqi library and sends the recorded audio data to the off-board code.

The recording of speech data is done using 'arecord', a standard Linux command-line tool for sound recording and playback on a system with a soundcard using ALSA drivers. In this manner, the raw audio can be retrieved in WAVE PCM 16-bit format. The audio is retrieved with a sample rate of 16000 samples per second, which is the upper limit for recording four channels for Pepper's recording setup. The 'arecord' command is performed in a process separate from the running Python code, and its output is piped to the Python program. To prevent missing chunks of audio, the subprocess' buffer is set to 5 seconds, which is enough to catch any regular slowdowns in the connection between Pepper and the off-board code.

Pepper makes use of the built-in NAOqi library. This library performs all autonomous actions done by Pepper, such as its idle movements and the retrieval of sensor data. Several parts of the pipeline and the experiments make use of this API. The most important parts are the dialogue system and the built-in SSL.

The dialogue system is used to perform a dialogue with the user during the recording of speech data, and during the Robot-in-the-wild test. To use the dialogue system, a dialogue file is created, which contains the keywords expected from the user and Pepper's replies to these keywords. The built-in keyword spotter is used by the dialogue system to recognize the keywords set in the dialogue file.

[1]NAOqi: `http://doc.aldebaran.com/2-5/index_dev_guide.html`

The dialogue system is initialized at the same time as the recording is started. It then performs the dialogue alongside the recording. To facilitate Experiment 2 (see Section 3.4.2), the result of the keyword spotting during the dialogue (i.e., a keyword is spotted or not) is stored.

The built-in SSL, otherwise referred as Pepper's SSL, is used in Experiments 1 and 3 (see Sections 3.4.1 and 3.4.3 respectively). It is performed in the background by NAOqi at all times. An event to broadcast the SSL estimation is only triggered if a sound over a threshold is detected. When this trigger is activated, Pepper broadcasts the estimated azimuth and elevation. Furthermore, this message also contains the confidence of and energy in the estimated direction, and Pepper's current position. Energy and confidence are used by NAOqi in the decision which estimated DOA is the most probable. NAOqi returns the estimated location with the highest confidence, which is calculated using the energy.

The audio data is streamed towards the off-board code over a socket, as described in Section 4.3. Since Google has a 65 second limit on the length of an audio stream for transcription, a more extended dialogue has to be split up in several pieces. This is not the case during the recordings of the speech datasets, given these are not transcribed during the recording and are split manually. To circumvent the limit on live transcription while recording, the recordings stop when Pepper throws the event 'ALTextToSpeech/TextStarted', indicating whether Pepper is speaking. As soon as the 'TextStarted' event is negated, the recording starts again. While this does not circumvent the issue when a user is speaking for more than a minute at a time, it does work in most dialogue situations. Restarting the recording can lead to slight delays if the socket does not connect immediately again. This effect is compensated for though since the maximum throughput of the data is generally far more substantial than the actual data.

After a chunk of audio data has been recorded, this data is sent to the off-board system together with Pepper's SSL estimation. This is further explained in Section 4.3.

4.3. Streaming

Streaming is performed using Python's built-in *Socket* library. This library allows for the creation of simple socket connections, over which byte streams can be sent. The socket library has been chosen due to its simplicity, especially when dealing with byte streams.

The audio data consists of 128 kB of data per second based on the sample rate, channels and bit depth: $\texttt{data\_size} = 16000 * 4 * 2 = 128\ \texttt{kB}$. Additionally, the location is sent together with the audio data. The audio data is sent in chunks of 16384 bytes, which is approximately 120 ms of four-channel speech data. This chunk size has been chosen to allow for low latency while not creating a large overhead by sending too many messages.

To simplify the transmission of multiple different variables, i.e., the audio data and estimated location data from the Pepper SSL, protocol buffers from the ProtoBuf library have been used [43]. Protocol buffers are a method created by Google to allow for simple serialization of structured data. Given that generally only a simple byte stream can be sent over the socket, Protobuf serializes the data it contains to bytes, which can be received and deserialized without losing any structure to the data. This way, the location data and the audio data can be separated easily while still being sent in a single message.

4.4. Off-board

The off-board processing for this experiment runs on a laptop but could be run on a server or desktop as well. Its most important tasks are performing the SSL and beamformer, the streaming of the processed speech signal to GC-STT, and receiving the transcriptions from GC-STT.

Two threads have been used in the program to perform several tasks in parallel: the receiver thread and the main thread. The receiver thread only performs the acquisition of the audio data from the on-board code and the conversion of this data to the correct format. The main thread performs the processing of the data, including performing the SSL and beamformer. It also takes care of transmitting the beamformed audio to the GC-STT and the reception of the transcriptions from GC-STT.

The receiver thread receives the protocol buffer in serialized form. After deserializing, the received audio data is converted into NumPy arrays representing the raw audio samples in the used PCM 16-bit format. The received audio data contains the samples from multiple microphones in a one-dimensional array (i.e., $[m_1, m_2, m_3, m_4, m_1, ...]$, where m_i is a sample from the respective microphone). This array needs to be split into a two-dimensional array, where the samples of each channel are grouped together. The splitting of this one-dimensional array to an array with the samples for each channel separated is performed by the receiver thread as well. The NumPy arrays are stored in a queue to be used by the main thread.

The speech signal is in the time-domain, meaning it describes the amplitude at each sample. Acoustic features have to be made from this time-domain signal in order to process the audio using a beamformer. These features are created using the following two steps.

Firstly, since speech is non-stationary (or quasi-stationary) [44], and since speech processing using a Fourier transform assumes stationary signals to function optimally, the speech signal is segmented into segments of 2048 samples. These segments are windowed into segments of 256 samples in the main thread. The windowing method used is a Hanning window with a 50% overlap between segments. The windowing and overlap ensure a smooth transition between processed segments in the audio, which otherwise can cause anomalies in the processed audio signal.

Secondly, these segmented windows are converted, using a one-dimensional DFT included in the NumPy library, to a frequency-domain signal. By using windowed segments of 256 samples, 129 frequency bins will be created of equal size between 0 and 8 kHz by the DFT. The 8 kHz upper limit is set by the Nyquist frequency, which is the highest frequency that can be retrieved using, e.g., a Fourier transform [45], and it is defined as half the sample rate. By using windows with 256 samples as input for the DFT, the amount of frequency bins is $\frac{\texttt{nr. of samples}}{2} + 1 = \#bins = 129$. This transformation results in the acoustic features, represented by a complex three-dimensional array, which represent the transformed frequency-domain data. The three dimensions represent the frequency bins and the different channels for each of the windowed segments.

The acoustic features (i.e., the frequency-domain signals per segment) are fed into the MUSIC SSL algorithm from the pyRoomAcoustics [39] library. This algorithm is able to calculate the spatial data of the acoustic features and will return a list of estimated sound source locations in descending order, with the most probable source at the top of the list. The MUSIC algorithm calculates steering vectors for a pre-set range of azimuths, in our case -60 to 60 degrees, which it uses to calculate the probability of a sound coming from

that direction. These steering vectors are retrieved for use by the beamformer in the next stage.

The acoustic features are sent to the Delay-and-Sum beamformer as well as the steering vectors (see 4.5). The beamformer computes the enhanced single-channel audio signal in the frequency domain from the 4 channel data and the steering vector. This enhanced frequency-domain signal is reverted to a time-domain signal again by using the inverse DFT. After calculating the time-domain signal, the windowed segments are joined together using an overlap-add, which is the inverse step of the segmentation. It adds the segments of time-domain signal together, with the overlap defined as when segmenting the the signal. The overlap-add creates a single, complete signal again.

Using the Google Cloud libraries, the enhanced time-domain signal is streamed to the GC-STT. The GC-STT returns a list of hypothesised transcriptions after each sent segment, sorted by probability as calculated by the recognizer. An 'is final' variable is provided with the response, which is set to true if the recognizer has identified the end of an utterance. This variable ensures that only full transcriptions are returned, instead of interim results, and to recognize when GC-STT will start with an empty transcript again, which requires storing the previous transcription.

4.5. Sound Source Localization & Delay-and-Sum Beamformer

The Delay-and-Sum beamformer works on the concept of changing the phase of a complex frequency-domain signal, with respect to the delay to a reference location. These delays are stored in the steering vectors, which are created for each possible DOA during run-time. To decide which steering vectors have to be used, a Sound Source Localization algorithm is performed.

The precise location of the microphones is required to be able to calculate the steering vectors. The microphone locations are stored in a 4 by 3 matrix depicting their euclidean coördinates relative to the middle of the microphone array. The delays between the microphones (τ) are calculated by performing the dot product between the microphone locations and the unit vector of the estimated DOA.

The steering vectors are created by calculating $e^{-j*2\pi*k*\tau}$ as explained in Section 2.4. Since a single steering vector contains the delay for each of the 129 frequency bins and each of the 4 microphones, 516 computations of the exponent are required. Calculating a single steering vector is doable while still performing in a real-time system. If all possible DOAs are known, the pipeline can be optimized by calculating all steering vectors during initialisation of the pipeline (i.e., before the recording is started).

Depending on the pipeline used, the steering vectors are either calculated by MUSIC SSL or by Pepper's built-in SSL. When the MUSIC SSL is used, the steering vectors are calculated by the PyRoomAcoustics library before the recording is started. This is only done once, since steering vectors do not change if the physical orientation of the microphone array does not change. Moreover, given the range of DOAs and the angular resolution are known in advance (due to the parameters of the MUSIC algorithm), all possible DOAs are known, making it possible to precompute the steering vectors accurately.

When the beamformer code creates the steering vectors, they are calculated after Pepper's SSL estimation is received. Only a single steering vector needs to be calculated, based on the DOA estimated by Pepper's SSL. As it is unknown which precise DOAs Pepper's SSL can return (i.e., the angular resolution is unknown), precomputing the steering vectors

would be very difficult and has not been done.

It is possible to force the beamformer to always focus in a preset direction, e.g., for debugging or when the DOA is known and does not change. In this case, a static azimuth can be used to calculate the corresponding steering vector.

Applying the Delay-and-Sum beamformer is a relatively short step compared to calculating the SSL. To perform the Delay-and-Sum beamforming (i.e., the phase shifts), the complex frequency-domain signal is multiplied (using the dot product) by the steering vector. This is performed for each frequency bin separately, but over all channels at the same time. After performing the dot-product on the complex signal of all channels and the steering vector, a single complex scalar is returned. This single complex scalar is the frequency-domain representation of the beamformed single-channel speech signal, for a single frequency. Performing the aforementioned calculations for each frequency bin, results in a vector representing the beamformed frequency-domain signal of the processed speech segment.

5
Results

The results from the experiments described in Section 3.4 are given in the following sections.

5.1. Experiment 1: Sound Source Localization Algorithm

The first experiment compared the performance of Pepper's SSL and the MUSIC SSL. The RMSE, averaged over all recordings was 12.1 degrees for Pepper's SSL and 19.7 degrees for MUSIC with a standard deviation (SD) of 7.4 and 3.9, respectively. A two-tailed t-test showed that Pepper's SSL performed significantly better than the MUSIC SSL ($p < .001$).

Inspection of Pepper's results showed that Pepper's SSL changed its estimated location only a few times. This was, in most cases, only when the participant started to read aloud from a different angle. This behavior helps to avoid erroneous steering caused by noise when speech is absent or too soft. A moving sound source, on the other hand, would possibly be wrongly steered after the first estimation.

MUSIC changed its estimation for nearly every speech sample it received. While the sound source location estimation during voiced speech is reasonably good, during short pauses or unvoiced speech, MUSIC tried to estimate the location of the loudest noise source with the highest correlation between the microphones, i.e., its built-in fan. Since the RMSE gives a higher weight to more substantial errors due to its quadratic nature, MUSIC SSL's estimation error becomes much more substantial. The frequent changing of the location estimation, on the other hand, would possibly make MUSIC better suited for moving speakers, or when Pepper is allowed to move its head freely. The usage of MUSIC in the pipeline, without disabling Pepper's head movements, was later evaluated on a speech transcription task during Experiment 4.

5.2. Experiment 2: Keyword Spotter

To evaluate the difference in ASR performance between Pepper and Google Cloud Speech-to-Text, their keyword spotting performances were compared. The results showed a KWER of 34.5% (SD = 11.0) for Pepper's keyword spotter and 28.3% (SD = 11.2) for GC-STT.

To perform a two-tailed t-test, the number of correctly recognized keywords was used. Pepper's keyword spotter correctly recognized 302 keywords and GC-STT recognized 473 keywords correctly of the total of 689 keywords. A two-tailed t-test with as observations 302

and 473, with $n = 689$ showed that GC-STT significantly outperformed Pepper's keyword spotter ($p < 0.001$).

Inspection of Pepper's keyword spotter's results showed that its keyword spotter has some difficulties, especially with the monologue included in the dialogue. Pepper's dialogue system, which is used to evaluate the keyword spotting task, appeared to finish its recognition of an utterance before an utterance was finished. This caused Pepper to stop listening to the participant's utterance prematurely. In turn, this caused Pepper to listen for the next keyword, causing recognition errors. Since GC-STT was not directly linked with the flow of the dialogue system during the recordings, this problem did not affect GC-STT's results.

On the other hand, Pepper's keyword spotter was able to handle the spelling of words, whereas GC-STT was unable to perform this correctly. During a small separate test to check if GC-STT's lexicon contained distinct characters, it became clear these are in fact part of its lexicon. This suggests that GC-STT prefers to transcribe words that sound alike or fit the context of the sentence, over individual characters. The inability to spell words could make certain interactions with the Pepper more difficult, e.g., having to spell a name to schedule an appointment.

5.3. Experiment 3: Speech-to-Text Transcription

To evaluate the proposed pipeline, the Speech-to-Text transcription of the beamformed speech is compared to two baseline systems. These baseline systems are: (1 Pepper's built-in SSL and the Delay-and-Sum beamformer, and (2 The unprocessed speech signal. Both the proposed pipeline and the baseline systems make use of GC-STT to create the STT transcription.

As mentioned before, Pepper's ASR pipeline does not work with prerecorded speech. This also means Pepper's SSL can only be used on live speech. Since the estimations of Pepper's SSL in noisy conditions can differ greatly from its estimations in clean speech conditions, the estimations calculated during the recording of the "Pepper SSL dialogue" dataset cannot be used for recordings with additive noise. Therefore, the comparison between the proposed pipeline and the baseline system using Pepper's SSL will only be performed with clean speech. Using the recordings of the four participants from the "Pepper SSL dialogue" dataset, the WER was calculated for the pipeline with MUSIC SSL, the pipeline with Pepper's SSL and the pipeline without beamformer.

The WER for the pipeline without beamformer is 31.0% (SD: 4.6%), for the pipeline containing Pepper's SSL and the Delay-and-Sum beamformer is 30.1% (SD: 3.9%), and the proposed MUSIC SSL and Delay-and-Sum beamformer pipeline is 28.0% (SD: 2.9%).

A formal statistical significance test is not performed, but a conservative model is defined as done by Scharenborg, et al. [46]. If assuming that errors within a speech recording are fully correlated, and follow a Bernoulli model [47], then two ASR systems are significantly different if their WER differ by at least $\frac{50\%}{\sqrt{200}} = 3.5\%$, where 200 is the number of sentences in the performed dialogue. As such, the difference between the pipelines is not significant; however, even with the small speech dataset (n=200) used in this experiment, MUSIC SSL did achieve a higher performance than the two baselines.

The second part of the third experiment evaluated the noise-robustness of the full proposed pipeline, which is done by comparing the proposed pipeline with the unbeamformed pipeline with noisy recordings. This experiment attempts to show the proposed pipeline

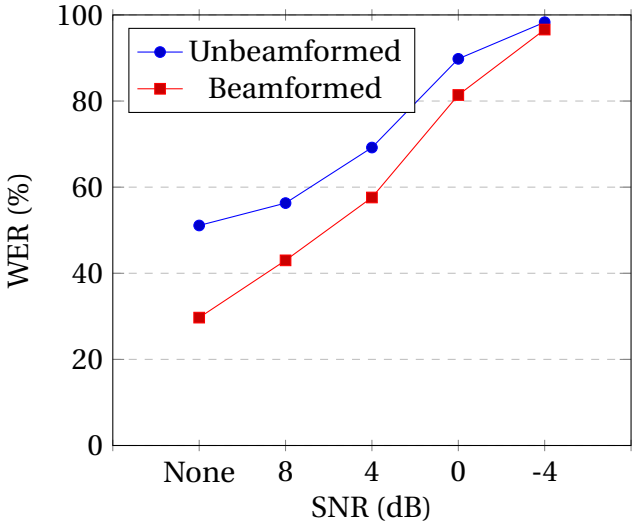


Figure 5.1: WER of the speech-to-text transcription task with the Google Cloud Text-to-Speech system on the unbeamformed and beamformed speech signal.

Table 5.1: WER, SD and total errors of the baseline system (B) and the proposed pipeline (P), followed by the decrease in WER by using P.

SNR	WER (B)	SD (B)	Total Errors (B)	WER (P)	SD (P)	Total Errors (P)	WER diff.
None	51.1%	18.5%	1770	29.7%	8.1%	1031	21.4%
8	56.3%	14.9%	1951	43.0%	8.7%	1490	13.3%
4	69.2%	14.3%	2397	57.6%	10.5%	1995	11.6%
0	89.8%	6.9%	3112	81.4%	5.2%	2822	8.4%
-4	98.4%	2.0%	3408	96.6%	2.6%	3348	1.7%

is more noise-robust than Pepper's baseline system (i.e., a pipeline without beamformer). As Pepper itself does not use a beamformer to enhance the speech signal, the unbeamformed pipeline is used as baseline to compare the proposed pipeline against Pepper's ASR pipeline. As mentioned before, the pipeline containing Pepper's SSL cannot be tested with noise, therefore it will not be used in this experiment.

The comparison between the proposed pipeline, and the unbeamformed signal is performed on the clean recordings from the "dialogue" dataset and the recordings from the "dialogue" dataset with noise mixed in from the recorded noise dataset. Figure 5.1 shows the WER for the different SNRs for the unbeamformed speech signal (dotted line with circles) and the beamformed speech signal (dashed line with crosses). The precise results are shown in table 5.1.

Significance tests are performed similarly to the first part of Experiment 3, which shows differences between WERs are significant with a difference of at least $\frac{50\%}{\sqrt{450}} = 2.4\%$ WER. A significant difference (> 2.4% WER) is achieved in most SNRs; only with an SNR of -4 dB no significant difference is found. This shows that the proposed pipeline results in a significantly improved noise-robustness over the baseline system.

Analysis of the transcription results showed that "jij/je/jou" ("you/your") were often confused. This could be due to the high phonological similarity between these words, and their occurrence in highly similar places in utterances. Moreover, both "jij" and "jou" are

often reduced to "je" when speaking. It is thus possible that the talkers unintentionally mispronounced these words. It is also possible that both possibilities happen.

5.4. Experiment 4: Robot-in-the-wild

The Robot-in-the-wild experiment has been done with four participants. The experiment with one of the participants has not been successful and did not provide a transcription, due to an error in setting up the connection between Pepper and the off-board code. From the three remaining participants, a transcription was created by the proposed pipeline. This led to an average WER of 51.3% (SD: 9.6%), though the transcriptions resulting from one participant was only complete up until the dialogue, due to the same error as the fourth participant.

Besides the connection error, several problems came up when performing the Robot-in-the-wild experiment. These problems include Pepper looking away from its interlocutor many times, and Pepper dialogue system continuing the dialogue due to "ghost utterances" (i.e., an utterance was heard by Pepper that was not uttered). These problems are examples of some of the difficulties when using the Pepper in a noisy situation, and can be used to improve the pipeline during further research.

After performing the experiment, it was discovered that Pepper's erratic looking behaviour was caused by a setting in NAOqi's Basic Awareness [1], the Engagement Mode. This setting decides if, and how Pepper keeps track of the person it is engaging with. This setting defaults to "Unengaged", which means Pepper responds to all stimuli and does not keep its focus on a single person. This setting could be set to "FullyEngaged", in which it continues to look at a person using multiple sensors, including its vision. If this setting were to be used, the assumption that Pepper always tries to look at the person it is engaging with would be correct.

Due to Pepper looking away from its interlocutor, it in some cases reached the limits set to the MUSIC algorithm. While the dialogue was performed, observations of the estimations made by MUSIC SSL showed that MUSIC SSL did appear to steer correctly when Pepper did look in the participants' direction within the limits set to MUSIC SSL.

Another problem was caused by the use of Pepper's keyword spotter to perform the dialogue with the participant, as mentioned in Section 3.4.4. During the experiment, Pepper's keyword spotter sometimes responded abnormally to utterances (i.e., replying before the interlocutor finished their utterance) or "ghost utterances", causing the dialogue system to continue to the next keyword in the dialogue prematurely. This caused the recognition and the recording to stop (for the current sentence), as Pepper starts speaking, and the system is set up to stop the recording when Pepper speaks.

5.5. Pseudo-real-time

As recognition within pseudo-real-time is important for ensuring proper Human-Robot Interaction, the processing time and latency have been calculated for the transcription test. This has been done for the recorded audio from the "dialogue" dataset and using live speech. When recorded audio was used, all speech data from this recording was available to be processed from the start. In other words, if a chunk of audio is processed, the

[1] NAOqi Basic Awareness:
http://doc.aldebaran.com/2-5/naoqi/interaction/autonomousabilities/albasicawareness.html

pipeline does not have to wait for the next chunk. When the test was performed using live speech (as in the Robot-in-the-wild experiment), the audio recorded by Pepper's microphones was streamed and used as input of the system, therefore when the processing of a chunk is finished, the pipeline had to wait for the next chunk to be received.

Figure 5.2 shows the steps in the system where a timestamp is created to calculate the run-times. The time between these measuring points are marked. T1 is the time between the start of the system and the connection between Pepper and the laptop used for off-board processing. This timing has not been taken into account as it is assumed this connection is made before the recording and the dialogue are started. T2 depicts the time it takes for all audio to be processed by the pipeline, including sending it to GC-STT. T3 depicts the latency between sending the last chunk of speech data to GC-STT and receiving the final transcription.

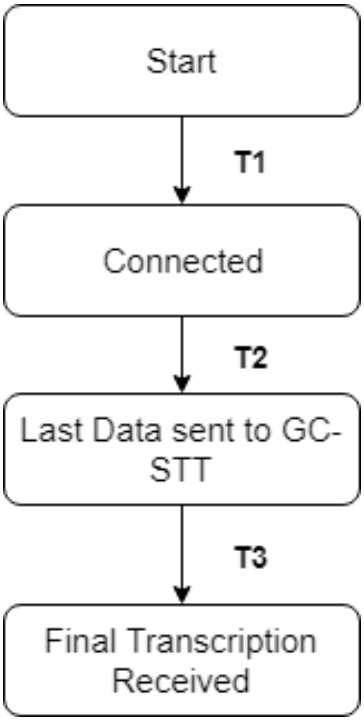


Figure 5.2: The measuring points to test processing times. T1: Time to initial connection, T2: Processing by proposed pipeline, T3: Latency until receipt final transcript

On average, all recordings per participant in the "dialogue" dataset contained 145.0 seconds of speech data. The recordings processed using the proposed pipeline required on average 134.1 seconds to process per dialogue. This shows the pipeline can work in real-time if all speech data were to be available, as soon as processing of a chunk of speech data was done.

The processing times of live speech were expected to be retrieved during the Robot-in-the-wild test. Due to the problems encountered during this experiment, this had to be done in another manner. As such, the pipeline was configured as it would in the Robot-in-the-wild experiment (e.g. performing the pipeline using live speech data). In this configuration, I have read aloud the dialogue to Pepper 5 times to calculate the processing time. This resulted in 32 streams per dialogue (160 in total) to be sent to GC-STT, given the stream is stopped each time Pepper speaks during the dialogue in the Robot-in-the-wild configuration (which it does 32 times).

The processing time and the latency by GC-STT (T2 and T3 in Fig.5.2 respectively) have been recorded. Furthermore, the length of the recording was calculated by dividing the number of samples sent to GC-STT by the sample rate of 16000.

T2 consistently fluctuating around approximately 100-150 ms processing time per 2048 frames (which represents 128 ms of speech data). The latency of T3 over the 160 recordings was approximately 300 ms per recording, with some intermittent recordings having a latency of one second. The average latency of T2 and T3 combined was around 750ms, which is more than the 500 ms defined as the upper limit for pseudo-real-time.

As T3 latency is approximately 300 ms, the remaining 450 ms could be blamed mostly on T2. When using live speech, a latency occurs in T2, since the pipeline will have to wait for the speech data to be recorded on, and received from Pepper. As the processing time of a chunk of speech is approximately 100-150 ms, the minimum latency is 100-150 ms from the end of the recording. Since a sliding window is used in processing the audio (during

the segmentation of the speech signal, see Section 4.4), all data will be processed twice, therefore leading to a latency of 200-300 ms. Furthermore, the latency between the speech being recorded on Pepper, and it being received on the off-board system adds to the total latency as well. This latency has not been taken into account though, and depends heavily on the network between Pepper and the off-board processing unit.

6

Discussion

Despite social robots, like Pepper, being increasingly more often used in human-robot dialogues and noisy environments, little is known about Pepper's ability to deal with continuous speech and background noise. To the best of my knowledge, the research in this thesis is the first to investigate an ASR pipeline for Dutch continuous speech, including preprocessing by a beamformer. Specifically, this thesis investigated 1) whether Pepper's built-in keyword spotter could be replaced by an ASR system able to deal with continuous Dutch speech; 2) whether Pepper's ASR pipeline could be made more robust against noise, without changing Pepper's hardware. To that end, in four experiments, Pepper's built-in Sound Source Localization algorithm and keyword spotter were compared to a newly proposed pipeline using SSL based on MUSIC, a Delay-and-Sum beamformer, and Google Cloud Speech-to-Text. The proposed pipeline was tested in cafeteria background noise in both an offline and online test.

The first experiment showed that Pepper's built-in SSL significantly outperformed the MUSIC SSL, with an RMSE that was 7.6 degrees lower for Pepper's SSL. The second experiment showed that changing Pepper's keyword spotter with Google Cloud Speech-to-Text yielded a significant decrease of 6.2% in Keyword Error Rate compared to Pepper's keyword spotter. This decrease shows Pepper's keyword spotter can be replaced by an ASR system able to recognise continuous speech in Dutch, with a better performance than the built-in keyword spotter.

The third experiment investigated the effect of the proposed MUSIC SSL and Delay-and-Sum beamformer on Automatic Speech Recognition. A comparison using clean speech recordings resulted in a WER of 31.0% for the unbeamformed speech, 30.1% for Pepper's SSL, and 28.0% for the proposed pipeline. Significant differences were not found though, most likely due to the small sample size, though MUSIC still outperformed the baseline.

To test the noise-robustness of the proposed pipeline, a comparison was performed using the recordings from the "dialogue" dataset with additive noise. The comparison between the pipeline containing MUSIC SSL and the beamformer showed a statistically significant decrease of 21.4% in WER in clean listening conditions compared to the raw speech system. Importantly, at an SNR of +8 dB, the proposed pipeline still showed a significant 13.3% improvement in WER over the unbeamformed speech. An improvement in the WER persisted in more difficult SNRs.

The results from the transcription experiment show that replacing Pepper's ASR pipeline by the proposed pipeline can make it more robust to noise. Despite the small datasets that

have been recorded, the difference between the proposed pipeline and Pepper's baseline is significant up to SNRs of 4 dB. The results from the transcription experiment, furthermore, show that the Delay-and-Sum beamformer can remove a large part of the noise generated by Pepper's built-in fan, as no other noise sources were present in the recordings without additive noise. Despite the decrease of WER over all tested SNRs, the WER produced by the proposed pipeline becomes very large (up to 81.4% in an SNR of 0 db). This WER would severely influence the interaction between Pepper and its interlocutor, making it (near) impossible to have a dialogue with Pepper. An average WER of 51.3% is observed in the Robot-in-the-wild experiment. This experiment was only successfully performed with three participants, but does show the pipeline performs in very noisy environments.

As indicated in the introduction, for a natural dialogue with a social robot, the dialogue needs to occur in (pseudo-)real-time. In this thesis, pseudo-real-time was assumed to be quick enough not to feel like there was a wait for the recognition result, and was set to 500 ms from the end of the utterance. The proposed pipeline had an average latency of 750 ms after the last utterance, therefore not performing in pseudo-real-time. The increase in performance though, both as keyword spotter and when performing transcription, can be worth the extra latency if Pepper is used in noisy environments.

We chose to use the MUSIC algorithm for Sound Source Localization, because the research in this thesis concentrates on evaluating Pepper's ASR in noisy backgrounds. In relatively quiet environments, I acknowledge that Pepper's SSL could be used as SSL for the pipeline, which would decrease the amount of external processing. However, its SSL performance is known to reduce in environments with an SNR below 3 dB[1]. MUSIC, on the other hand, explicitly considers noise to be part of its model. Moreover, the pipeline containing MUSIC takes the distance between the talker and the robot in account when creating its steering vectors, which allows fine-tuning of these steering vectors. Nevertheless, more research is required into finding the optimal SSL for Pepper (see [36] for a review of possible SSLs).

MUSIC SSL has been used in the creation and evaluation of the HARK robot audition system as well [7]. The HARK system showed an WER of 22% in an environment with -4 dB "Target-to-robot-noise" ratio (TTRNR), dropping to less than 13% in 0 dB TTRNRs. The WERs achieved by HARK are much better to the WERs in this thesis (97% and 81% for SNRs of -4 dB and 0 dB, respectively). HARK was evaluated on isolated words though, as opposed to the continuous speech used in this thesis. Furthermore, HARK uses the TTRNR, which includes the robot's own noise as reference for the speech volume. This measure has not been used in the research in this thesis, but as Pepper's fan adds much noise to the "clean" speech recording, the actual SNR of the recordings made with Pepper is probably much lower.

It has been attempted to use a cloud-based ASR system for the Pepper robot in other research. GC-STT has been used to improve the performance of Pepper's speech recognition, in combination with its default speech recognition [10], resulting in an improvement of recognition results. Recent research, in which the IBM Cloud services were used to evaluate the use of Pepper in patient assessment [9], show an improvement of speech recognition performance as well. While their evaluation does not aim at improving the WER specifically, they do mention that using Watson (IBM's Cloud Services) was required to overcome the problems caused by the fan in Pepper's head. The research in my thesis

[1] Pepper's SSL's reduced performance in noise is mentioned here: `http://doc.aldebaran.com/2-5/`

concurs with the assessment that Pepper's speech recognition is strongly influenced by the fan. The use of only a cloud-based ASR system (i.e., without additional preprocessing), in both the aforementioned researches, already shows an increase in the performance of the speech recognition of Pepper. The results of Experiment 2 support this statement as well (see Section 3.4.2).

It is important to note that the condition without added background noise, still contained a substantial amount of noise. The fan, located underneath the microphones in Pepper's head, creates audible noise in all recordings. To create good transcriptions, the fan noise need to be attenuated without distorting the audio. The "clean" listening condition in Experiment 3 showed that the proposed Delay-and-Sum beamformer was able to substantially attenuate the noise from the fan and actuators, although the noise was still audible in the beamformed speech signal. GC-STT also contributed to the improved recognition, due to its own noise-robustness.

The audibility of the fan noise in the speech signal beamformed by the proposed pipeline, could be caused by the MUSIC SSL algorithm responding quickly to small pauses in speech. During these silences, the beam is steered towards the fan noise, as it is the next most probable sound source.

Exploratory experiments (at the start of this thesis) showed that audio which had most of the fan noise removed (i.e., which sounded less noisy to the human ear) performed worse in the transcription experiment. This finding could be due to the human ear being able to handle small distortions in the speech, while these small distortions (e.g., audio clipping) can cause recognition problems for a trained ASR system.

Pepper's ability to recognize speech could be further improved if the fan noise could be removed completely. Alternatively, a different beamformer could be used. One of the options would be a Minimum Variance Distortionless Response (MVDR) beamformer [48]. An MVDR is an adaptive beamformer which adjusts the weights of the microphones, minimizing the variance without adding distortion in the direction of the source signal. This should allow for better noise cancellation without any distortion of the speech signal. Another option is the Generalized Sidelobe Canceller (GSC) [49]. This system combines a simple beamformer and a sidelobe canceller which aims at canceling noise from non-source directions. The GSC would create additional noise cancellation in specific directions, which could help with static noise sources such as the fan. Although these beamformers could improve the quality of the speech signal compared to the used Delay-and-Sum beamformer, they might require additional computing power to be used real-time. Moreover, due to the Pepper's limited number of microphones, these beamformers are expected to increase Pepper's performance relatively little compared to the used Delay-and-Sum beamformer.

In the experiments in this thesis, two-dimensional sound source localization was used; however, SSL algorithms often have the option to be used in three dimensions. A small-scale test using the three-dimensional algorithm showed that the difference between the WERs of the beamformed audio in two or three dimensions is at most 0.8% absolute. For Pepper, the elevation of the beamformer has seemingly far less influence on the recognition of the speech than the azimuth. Furthermore, by performing SSL in three dimensions, the number of possible locations for sound sources becomes much higher, increasing the processing time significantly. Keeping the (pseudo-)real-time limitation in mind, it would, therefore, be better to perform two-dimensional SSL with a higher angular resolution than a three-dimensional SSL.

7

Future Work

The most critical next step for further work is acquiring more speech data. The recorded speech database used in the evaluations in this thesis consisted of a limited number of participants. In order to be able to draw stronger conclusions, a more substantial number of participants, and thereby, more speech recordings, would be needed.

Noise reduction during the preprocessing for a social robot's ASR system was the main focus of this thesis. While this is an important aspect of improving the ASR results, for further work, the reverberation of a room could also be taken into account. Smaller rooms or locations with special acoustic characters, such as a small office or a church, introduce large amounts of reverberation in the recorded audio. This could strongly influence the performance of the ASR system and might require extra steps to the pipeline or changes to the parameters of the system [50].

Furthermore, different types of background noise have different characteristics than the cafeteria noise used in this thesis, possibly producing different results when using the proposed pipeline. Further research could be done to ensure the proposed pipeline is noise-robust to other types of background noise.

When Pepper is used in a very noisy environment, a different method of performing the dialogue could improve the Human-Robot interaction. One of these methods would be to use the transcriptions from GC-STT as input for the dialogue system, instead of Pepper's keyword spotter. More investigation should be done in the implementation of this method. A possible solution could be the use of a noise-robust end-point detection algorithm to detect the end of an utterance, allowing the pipeline to recognize when a person stops speaking to Pepper.

During the final stages of this thesis, Google Cloud STT has received an update which allows up to 5 minutes of streaming data per connection. A less restrictive time limit would mean the stream would have to reconnect to GC-STT less often. This would result in a more stable connection to GC-STT, given connecting to this service could add delays. Also, the chance of errors caused by reconnecting during an utterance is reduced by making use of this improvement. Future research could investigate whether this would have an impact on the ASR pipeline's recognition and computation performance.

Using different beamformers or SSL algorithms could further increase the performance of the ASR pipeline created for Pepper. While other beamformers such as an MVDR could increase the computational complexity of the system, optimizations performed or limitations set to these beamformers might still allow for real-time processing of the speech

stream. For example, if an MVDR beamformer is used together with Pepper's SSL, which does not require external processing of the audio as MUSIC does, the processing time used by MUSIC could be replaced with the necessary computations for the MVDR.

Further work could, furthermore, include the possibility to listen to multiple people simultaneously. The usage of MUSIC already allows for localization of multiple signal sources, of which only one is used in the proposed pipeline. By using a beamformer with better directionality, i.e., better separation of audio from a different DOA, in conjunction with the MUSIC SSL, multiple speakers could be separated. Achieving this would open up many more use cases for Pepper.

8

Conclusion

The research in this thesis aimed at answering two research questions: 1) *Can Pepper's built-in keyword spotter be replaced by an ASR system able to recognise continuous speech in Dutch?*, and 2) *Can Pepper's ASR pipeline be made more robust against noise, without changing its hardware?* To that end, in-house speech and noise data has been gathered, and an ASR pipeline based on the MUSIC Sound Source Localisation, Delay-and-Sum beamforming, and the Google Cloud STT has been created.

Results from the transcription experiment show that the complete, proposed pipeline performed significantly better than Pepper's baseline system. Furthermore, a series of three experiments, which investigated the contribution of each of the individual components of the proposed pipeline, showed a significant improvement of most of the proposed components, compared to Pepper's baseline. Although the first experiment showed that Pepper's SSL was able to recognize the location of a speaker more accurately than the MUSIC SSL in clean listening condition, using MUSIC SSL in the proposed pipeline did improve the transcription results compared to the Pepper's baseline. The second experiment showed, by an improvement of Keyword Error Rate, that Google Cloud SST performs better than the ASR engine (NUANCE) used by Pepper. The third experiment showed that the addition of preprocessing, using MUSIC SSL and a beamformer, to the pipeline improved the quality of the transcriptions significantly.

To answer the first question: yes, Pepper's built-in ASR engine can be replaced by a pipeline of an SSL, beamformer and the Google Cloud Speech-to-Text. In the presence of moderate background noise, the found significant difference persisted; however, in worse listening conditions, the performance difference became smaller, though the proposed pipeline still outperformed Pepper's ASR system numerically. To answer the second question, this better performance of the proposed pipeline shows an increased noise-robustness compared to Pepper's ASR system.

A

Dialogue

Pepper **Participant**

1. Hallo, ik ben Pepper, kan ik u ergens mee helpen?

   Hallo Pepper ik wil graag met jou praten

2. Dat klinkt leuk. Waar wilt u het over hebben?

   Ik zou graag over jou praten.
   - Weet je waarom ik tegen jou praat?

3. Nee, waarom is dat?

   Ik spreek nu tegen jou om testdata te vergaren.
   - Dit is om jouw spraakherkenning te verbeteren en te testen.

4. Oh, dat klinkt handig. Heb je een paar vragen voor mij?

   Ja ik wil graag weten hoe oud je bent?

5. Ik ben 4 jaar oud.

   Dan ben je dus geboren in 2015

6. Dat klopt, en wanneer bent u geboren?

   Mijn geboortedatum is 6 maart 1991

7. En waar komt u vandaan?

   Ik kom uit Den Haag

8. Dat is in de buurt, ik kom helemaal uit Japan

   Japan is wel ver weg.
   - Hoeveel talen ken je allemaal?

9. Ik ken nu 9 talen, maar ik leer er graag meer.

   Versta je ook Fries?

10. Nee, helaas versta ik dat niet.

    Dat maakt niet uit, ik kan ook geen Fries.
    - De woorden zijn lastig om uit te spreken.

11. Nu we het over lastig uit te spreken woorden hebben, heeft u nog een leuke tongbreker voor mij?

    De kat krabt de krullen van de trap.

12. Die is wel leuk. Zullen we een kleine quiz doen?

    Ok

13. Wat is 1 miljoen min 1?

(999.999) Negenhonderd-negen-en-negentigduizend-negenhonderd-negen-en-negentig

14. Goed! En wat is de hoofdstad van Denemarken?

De hoofdstad van Denemarken is Kopenhagen

15. Helemaal top! Hoe spel je “service”

Service spel je S-E-R-V-I-C-E

16. Perfect! Hoe heet de spelleider in een casino?

Een spelleider in een casino heet een croupier.

17. Leuk dat je mee hebt gedaan met deze korte quiz. We kunnen doorgaan als je wilt.

Nee dank je.
- Ik zou nog wel graag een verhaaltje willen vertellen.

18. Oh? Ik hou wel van verhalen.

Toen de wereld pas geschapen was en de maanden nog jong waren, legden ze graag een kaartje met elkaar.
- Vooral Februari hield veel van kaarten
- maar hij was zo ongelukkig in het spel dat hij altijd verloor.

Op zekere dag bemerkt hij dat hij alles verloren had
- maar nog éénmaal wilde hij het wagen
- misschien kon hij in één slag zijn gehele verlies terugwinnen.

En dus begon hij met zijn broers Januari en Maart te spelen.
- Ook nu was het geluk niet met hem
- hij verloor weer maar daar hij geen rode duit meer bezat om zijn verlies te betalen
- moest hij aan zijn medespelers elk een dag afstaan.

Zo komt het dat januari en maart 31 dagen hebben
- en februari voor zichzelf maar achtentwintig dagen overhield.

19. Ik vond dit een leuk verhaal. Zou je mij nu nog even willen helpen? Ik wil graag oefenen met mensen helpen. Zou je willen doen alsof je een afspraak wilt maken met mijn baas.

Ok, ik wil je best helpen. Begin maar!

20. Hallo, kan ik u ergens mee helpen?

    Ja, ik wil graag een afspraak maken met jouw *baas*.

21. Dat kan, en wanneer zou u deze afspraak willen?

    Ik wil een afspraak voor overmorgen, 23 februari.

22. Sorry, dan heeft mijn baas geen tijd, kunt u volgende week donderdag?

    Dan kan ik ook.
    Zou ik dan om tien voor half 12 kunnen komen?

23. Ok, u wilt een afspraak op donderdag 28 februari, om tien voor half 12. Klopt dit?

    Dat is correct.

24. Ok, dan staat uw afspraak gepland. Kan ik u nog ergens anders mee helpen?

    Ja.
    - Kan ik het beste met bus 69 vanaf het station reizen naar het kantoor?
    - Of is het makkelijker om te lopen?

25. U kunt met de bus naar het kantoor komen. Het beste kunt u echter fietsen. Deze kunt u huren bij de stalling.

    Dat is handig om te weten *dankjewel* Pepper.
    - Kan ik dan beter door het centrum heen fietsen, of daar *omheen*?

26. Het beste kunt u om het centrum heen rijden.

    Ok Pepper dan weet ik alles wat ik zou willen weten.

27. Graag gedaan! En u bedankt voor het helpen.

    Graag gedaan.

28. Kan ik nu nog ergens mee helpen?

    Nee hoor maar ik vond dit een interessant gesprek

29. Ok, ik vond het ook leuk.

    Veel succes met het onderzoek.

B

Paper Submitted to ASRU

EVALUATING A REAL-TIME ASR PIPELINE FOR SOCIAL ROBOTS

Jamey Sparreboom[†], *Koen Hindriks*[‡], *Odette Scharenborg*[†]

[†]Multimedia Computing Group,
Delft University of Technology
Delft, the Netherlands

[‡]Department of Computer Science
Vrije Universiteit
Amsterdam, the Netherlands

ABSTRACT

There has been a big increase in the use of social robots, such as Pepper, which use Automatic Speech Recognition (ASR) as the main communication between a human and the robot. Since social robots are often used in dialogues and in noisy environments, this paper investigates 1) whether Pepper's built-in keyword spotter can be replaced by an ASR system; 2) whether Pepper's ASR pipeline can be made more robust against noise, without the need to change Pepper's hardware. To that end, Pepper's built-in Sound Source Localization (SSL) algorithm and keyword spotter are compared to a newly proposed pipeline using SSL based on MUSIC, a delay-and-sum beamformer, and Google Cloud Speech-to-Text. This pipeline showed a decrease in Keyword Error Rate of 6.2% compared to Pepper's keyword spotter and a decrease of more than 20% in Word Error Rate compared to unbeamformed audio in clean listening conditions. At a signal-to-noise ratio (SNR) of +8 dB, the proposed pipeline showed a 13.3% improvement over the unbeamformed speech which persisted in more difficult SNRs. Thus Pepper's speech processing can be improved and made more robust against noise by preprocessing the audio with a beamformer and transcribing it using Google Cloud Speech-to-Text.

***Index Terms*—** Automatic Speech Recognition, Social Robot, Noise, Pepper, Beamformer

ABSTRACT

The authors would like to thank Mark Hasegawa-Johnson for fruitful discussions on the implementation of the beamformer and the interpretation of its results.

1. INTRODUCTION

Social robots are being used in many different scenarios, e.g., to welcome guests in a hotel [1], to provide information in a shopping mall [2], to interview patients to collect patient data [3], or to educate children [4]. These scenarios assume some form of communication between a human and the social robot. The most intuitive communication method when conversing with a (humanoid) social robot is using voice interaction, for which automatic speech recognition (ASR) is needed. In the scenarios in which they are used, social robots often need to deal with noisy environments. Unfortunately, typically, the performance of the automatic speech recognition (ASR) systems of these robots deteriorates in noisy conditions [5], limiting the locations and situations in which such a robot can be effectively used.

Surprisingly, although a lot of research has been carried out on the design of dialogues for social robots (e.g., [6, 7, 8]), less research has been carried out investigating the ASR pipeline of social robots. It has been reported that speech recognition for social robots poses a problem, e.g., for recognizing children's speech [9], which still suggests trying to reduce noise. More research in ASR pipelines used in robots has been done before, often under the name "Robot Audition", though these often put the emphasis on changing the ASR engine itself [10], or use robots which have less strong limitations such as larger microphone arrays [5]. In this study, we use the Pepper social robot [11].

Pepper contains a built-in ASR system, which is capable of automatic recognition of continuous speech for multiple languages including English, Japanese and Dutch [12]. In our research, we focus on Dutch as the language of interaction between the human and the Pepper robot. It has been found though Pepper does not properly support some languages by not having free-speech recognition, but only recognition of pre-set utterances. One of this languages is Dutch. The inability of Pepper's in-built ASR system to recognize Dutch free speech severely limits its functioning as a flexible social robot. The first goal of this paper is to replace this keyword spotter with an ASR system capable of recognizing Dutch continuous free speech.

There are several ways an ASR system can be made more robust to background noise. For instance, using speech enhancement [13, 14] or by training the ASR system on noisified speech [15]. However, Pepper's speech processing system is not trainable nor can it deal with preprocessed data. Consequently, in order to deal with background noise, these methods cannot be used. The second goal of this paper is to try to make Pepper more robust against the presence of background noise without changing its hardware, using a beamformer.

Specifically, this paper aims to answer two questions: 1) Can Pepper's built-in keyword spotter be replaced by an ASR system? 2) Can Pepper's ASR pipeline be made more robust against noise, without changing its hardware? To that end, we propose to replace Pepper's built-in keyword spotter with an ASR service in the cloud. Only a limited number of cloud ASR systems for Dutch were available; we chose Google Cloud Speech-to-Text (GC-STT) [16]. Using GC-STT also allows us to preprocess the speech signal to remove (some of) the background noise. Here, we propose to replace Pepper's built-in Sound Source Localization (SSL) algorithm with a pipeline using SSL based on MUSIC [17, 18] and a delay-and-sum beamformer [19, 20]. Importantly, because Pepper is typically used in social settings, all solutions should work in (pseudo-)real time.

Because Pepper cannot deal with audio processed outside of Pepper, a series of three experiments was designed to ultimately evaluate the performance of the proposed pipeline of MUSIC, the delay-and-sum beamformer, and the Google Cloud ASR. These experiments are designed to tease apart the influence of changing the

sound source localization algorithm, the addition of a beamformer, and changing the ASR engine.

Section 2 describes the general set-up of the experiments, the speech data that was collected, and the proposed ASR pipeline. Section 3 present the results of the experiments as well as an error analysis. The paper ends with a discussion of the results 4.

2. METHOD

2.1. The Proposed Pipeline

The bottom flow of Fig. 1 shows the proposed pipeline: The audio is recorded using Pepper's built-in microphones. The audio then passes through the MUSIC sound source localization (see Section 2.1.2) and the proposed Delay-and-Sum beamformer (see Section 2.1.3), after which the beamformed speech signal is send to the Google Cloud Speech-to-Text (see Section 2.1.4) for recognition. This proposed system is compared to the standard Pepper system, which is considered to be the baseline system (see Section 2.1.1).

2.1.1. Pepper

Pepper is a social robot, often used for welcoming and informing people in companies or at events [11]. Pepper has four built-in microphones, located on top of its head, in a rectangular or elliptical array. The exact layout of the microphones in the Pepper robot used in this study is shown with the black dots in Fig. 2, where each black dot represents a microphone, and where the front of Pepper's head is facing downwards. The microphones are separated 6.7 cm from one another on the left-right axis and 5.8 cm on the front-back axis. A fan for cooling the electronics is located below the microphones. This fan adds significant noise to the speech recordings.

Pepper's built-in sound source localization algorithm estimates the direction of arrival (DOA) of the loudest noise it receives with about 10 degrees precision [21]. This estimation is stored in Pepper's memory. Pepper's SSL, however, cannot distinguish speech from noise. It loses reliability in noisier environments, and when multiple sound sources are received the location of the loudest sound source will be stored [21].

The spatial data from the SSL can be used to calculate the delay between the microphones. This delay, in turn, can be used by a beamformer to adjust the phase of the speech signal it receives in each of the microphones in order to amplify the sound from the source and attenuate the sound from other directions. This is done by multiplying the audio received by the microphones with their respective steering vectors, i.e., the delays between the microphones respective to the estimated sound source. Importantly, however, Pepper does not use an built-in beamformer.

Pepper has a built-in dialogue system, which takes the speech signal recorded by Pepper's microphones as input, and uses keyword spotting to parse the speech input and decide on a reply. Wild-card characters can be used in the dialogue system instead of keywords, allowing Pepper to recognize words in a manner more similar to speech-to-text transcription. This feature is however not available for Dutch, the language we work with. Pepper's built-in ASR is created by Nuance [21].

2.1.2. The MUSIC Sound Source Localization Algorithm

In the proposed system, the MUSIC sound source localization algorithm is used [17, 18]. The MUSIC sound source localization algorithm takes Pepper's audio as input and returns an estimate of the location of the signal of the source.

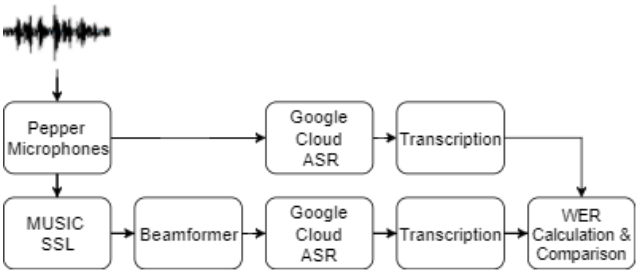


Fig. 1. The bottom flow shows the proposed pipeline with the beamformed speech signal, which is compared to the top flow, i.e., the unbeamformed speech signal, in Exp. 3.

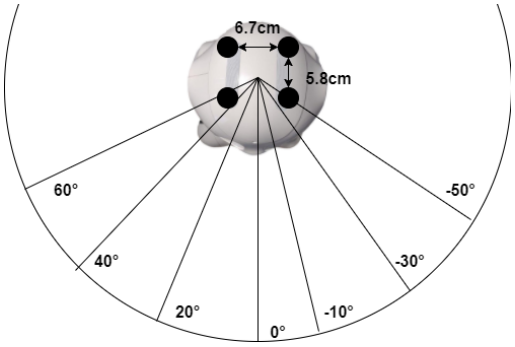


Fig. 2. Top view of Pepper's head with the microphone locations shown as black dots, and the angles used in Exp. 1.

We used the MUSIC algorithm in the Python library *PyRoomAcoustics* [22], which is based on the paper by Schmidt [17]. The MUSIC algorithm was chosen since it allows for a high resolution estimation of the sound source location taking into account the distance between the talker and the microphones. The MUSIC algorithm is a computationally heavy algorithm; however, when it is used within predefined limits regarding listening angle, resolution, frequency range, and the frequency with which the algorithm is run, it is able to run in real time.

The MUSIC algorithm performs eigendecomposition on the covariance matrix of the received signal. By finding peaks in its power, multiple sound sources can be located, including noise. The highest eigenvalues found correspond with the speech signal subspace, whereas the other eigenvalues are in the noise subspace. The subspaces are formed by dimensionality reduction based on Principle Component Analysis (PCA) [14]. By projecting a sample on the signal subspace and not on the noise subspace, the noise from the sample is discarded partially. We focus on a single speaker, therefore only the first eigenvector has been used to locate the source. The system can be extended to accommodate multiple speakers.

Each of the possible source locations has its own set of precomputed steering vectors. These are computed prior to running each experiment to reduce computation time. These steering vectors do not change as long as the microphone array stays the same, because the respective delays between the microphones stays the same for each Direction of Arrival.

2.1.3. The Delay-and-Sum Beamformer

The beamformer takes Pepper's audio and the estimated source location from the MUSIC algorithm, and outputs the beamformed speech signal. For the experiments, a delay-and-sum beamformer [23, 20] was chosen as this type of beamformer requires the least amount of processing power and time, which is important for allowing the system to run in real time.

The delay-and-sum beamformer sums the audio of each microphone together in pre-set (often equal) amounts. The delay-and-sum beamformer does not adapt its weights for each channel automatically on the basis of the received signal. Since Pepper will always focus its head on the person engaging it, and the noise generated by the fans is stronger near the rear microphones, the front microphones are given a weight of 0.3 and the rear microphones 0.2. These values have been found empirically. The delay-and-sum beamformer furthermore adds a delay to the phase of each channel, compared to the middle of the array as reference location, which brings the audio from each microphone in phase.

2.1.4. Google Cloud Speech-to-Text

The Google Cloud Speech-to-Text [16] takes either the raw speech signal from Pepper or the beamformed speech signal as input and transcribes the speech into text. The Google Cloud Speech-to-Text is based on deep neural networks. The GC-STT fully supports Dutch and is able to transcribe real-time streams, which is crucial for dialogues with social robots. The GC-STT is used in two ways: as a keyword spotter in Experiment 2 and a standard STT in Experiment 3.

Keyword spotting, for Experiment 2, is mimicked by providing so-called *phrase hints* to improve the confidence scores of the provided phrases, which increases the chance of correctly recognizing the word. In Experiment 3, GC-STT is used in its normal capacity, i.e., the word-by-word transcription of the speech signal.

The input to the GC-STT is a stream of Pulse Code Modulation (PCM) data, which is a digital representation of audio where the amplitude is sampled and interpolated to the nearest integer within its sample size. This representation was chosen since it is uncompressed, and because Pepper also uses this representation as input. This input stream can have a maximum length of 65 seconds due to limitations in GC-STT. Therefore each of the recordings in the speech database has been streamed separately, before setting up a new connection to GC-STT.

Google Cloud Speech-to-Text is robust against noise [16]. Preprocessing for noise reduction is adviced not to be used as it might increase distortion of the signal [16], decreasing the performance of the ASR engine.

2.2. Speech Data

Two dedicated datasets were recorded in a sound-proof booth to evaluate the proposed pipeline. Both datasets consisted of recordings of a single scripted dialogue by Dutch native talkers. All talkers were recruited from the Faculty of Electrical Engineering, Mathematics, and Computer Science (EEMCS) of the Technical University of Delft, the Netherlands, participated for free, and are native speakers of Dutch. All participants signed an informed consent form prior to the recordings.

The dialogue consisted of 50 sentences. It was constructed to contain different types of phrases and utterances that could be used in different types of dialogues with a Pepper robot, including short and long phrases, homonyms, large numbers and dates. Moreover, a monologue, in the form of a short story, was included to test longer sentences and to investigate how the ASR would respond to phrases with confusing contexts, such as using months instead of names. The monologue made up 12 out of the 50 sentences in the complete dialogue.

The first dataset, referred to as the "steering" dataset and used in Experiment 1, see Section 2.3.1, consisted of recordings made by 8 talkers (5 males and 3 females, age range: 20 - 30 years). Each participant recorded four sentences from the dialogue. For each recording, Pepper was placed 1 meter in front of the participant, to ensure constant and comfortable interpersonal distance, at each of seven different angles, which are shown in Fig. 2. The chosen angles had a $20\,\mathrm{deg}$ interval except for $0\,\mathrm{deg}$ and $-10\,\mathrm{deg}$, since the symmetric microphone array gives the same results for each side of the robot on equal angles. All actuators were turned off during the recording to avoid additional noise. Each recording took about 10 seconds.

The second dataset, referred to as the "dialogue" dataset and used in Experiments 2 and 3, see Sections 2.3.2 and 2.3.3, consisted of recordings made by 9 participants (7 males and 2 females, age range: 20 - 30 years. Five of these participants were also recorded for the steering dataset and returned for a second recording session for the dialogue dataset. Moreover, 4 additional talkers were recorded. During the recordings, Pepper was placed 1 meter in front of the participant at a 0 degree angle. Each recording of the 50 sentences dialogue resulted in approximately 5 minutes of speech data.

The talkers were instructed to speak as they normally would, but to ensure they speak clearly. Due to the dialogue engine replying overly fast after a short pause in speech, participants were instructed not to pause within a sentence.

The recordings were manually cut at positive going zero-crossing into one-sentence fragments using Praat [24] leaving approximately 500 ms of preceding and trailing silence. Loud noises, such as beeps played by Pepper, that fell within the 500ms window were excluded from the 500 ms window. Pepper's responses were removed. The audio fragments are normalized to 70 dB.

As Pepper robots are typically used in social contexts, several minutes of cafeteria noise were recorded in the EEMCS faculty cafeteria using Pepper. To ensure that the beamformer spatial filtering could be properly evaluated, it was made sure that most noise came from behind and to the sides of the robot. Noise stretches louder than 72 dB and silent segments were manually removed at positive-going zero-crossings to ensure a relatively stable noise level. The noise signal was normalized to 70 dB after which random stretches of the noise were automatically mixed with the speech signal at four different signal-to-noise ratios (SNR), i.e, 8 dB, 4 dB, 0 dB, and -4 dB, using a custom-made Praat script. Two hundred ms of preceding and trailing noise was used (in addition to the preceding and trailing silence in the speech fragments). The noisefied speech was only used in Experiment 3. Experiments 1 and 2 were conducted with clean speech.

2.3. Experimental Set-up

To evaluate the proposed pipeline, three experiments were carried out. The first experiment was the steering experiment and investigated the performance of the SSL systems, comparing Pepper's SSL versus the MUSIC SSL. The second experiment investigated the difference in performance of the recognition engine in a keyword spotting task performed by Pepper's ASR and Google Cloud STT. The third experiment compared the recognition performance of Google Cloud STT on the raw audio received from Pepper and the beamformed audio.

2.3.1. Experiment 1: Sound Source Localization Algorithm

An error in the estimation of the location of the sound source causes a beamformer to delay wrongly, which causes the source signals to not be out of phase. This attenuates the target signal and amplifies noise instead of vice versa. Therefore the first experiment evaluates the Sound Source Localization algorithm of both systems, i.e., Pepper's built-in sound source localization algorithm and the MUSIC sound source localization.

The sound source localization algorithms are evaluated in terms of the DOA in degrees. To that end, the DOAs at corresponding time stamps of Pepper's SSL and of the MUSIC SSL are compared. The root mean squared error (RMSE) between the estimated angle and the ground truth angle, which is the angle at which the recording was made, is calculated. By using RMSE as a metric, large steering errors will have a bigger influence on the result, just as large steering errors make a beamformer perform worse.

Given Pepper's default behaviour to look at its interlocutor during a conversation, it is assumed the person speaking to Pepper is always in front of Pepper. Given this assumption, the limits within which MUSIC SSL has to listen have been set to 60 degrees to each side. This is done to lower the number of sound source locations the algorithm has to compare, allowing the MUSIC algorithm to be used in real-time. Having these limits also prevents the SSL to focus on the fan in Pepper's head which produces noise.

2.3.2. Experiment 2: Keyword Spotter

Pepper's built-in ASR system is a keyword spotter, while the Google Cloud system is a continuous speech recognition system. In order to investigate the performance of the Google Cloud ASR system in relation to the baseline Pepper keyword spotter, the second experiment evaluates the recognition engines, Pepper vs. GC-STT, on a keyword spotting task using the raw speech signal, i.e., no beamformer was used. In this task, the audio recorded by Pepper is passed both through Pepper's built-in keyword spotter and through the GC-STT, which operates in a in 'keyword spotter' mode using 'phrase hints' [16].

Since Pepper's dialogue function and the keyword spotter both can only be used with live audio, Pepper's keyword spotter was tested during the recordings. This has been done by creating a dialogue in Pepper's built-in dialogue system, where Pepper continued the conversation after each utterance by the participant. Each correctly recognized keyword is stored in memory.

Subsequently, the recordings were streamed to GC-STT, which operated in keyword spotter mode (see Section 2.1.4). Each transcription created by GC-STT was checked for the keywords it should contain, and the keywords were marked correct if they are found.

Performance was measured in terms of the number of correctly recognized keywords or the Keyword Error Rate (KWER), which is calculated using

$$KWER = \frac{\texttt{incorrect keywords}}{\texttt{total keywords}}\%.$$

2.3.3. Experiment 3: Speech-to-Text Transcription

To evaluate the beamformer on a transcription task and its performance in noise, the GC-STT is tested on the raw speech signal, which came straight from Pepper's microphones, and the beamformed speech signal, which came from the proposed pipeline without background noise and with background noise mixed into the speech signal.

The two systems which were compared in Experiment 3 are shown in Fig. 1. The performance of the GC-STT on the raw speech and the beamformed speech was evaluated in terms of word error rate (WER) in relation to the ground truth, i.e., the dialogues, for each of the noise levels.

3. RESULTS

3.1. Experiment 1: Sound Source Localization Algorithm

The first experiment compared the performance of Pepper's SSL and the MUSIC SSL. The RMSE averaged over all recordings was 12.1 degrees for Pepper's SSL and 19.7 degrees for MUSIC with a standard deviation (SD) of 7.4 and 3.9, respectively. A two-tailed t-test showed that Pepper's SSL performed significantly better than the MUSIC SSL ($\mathrm{p} < .001$).

Inspection of Pepper's results showed that Pepper's SSL changed its estimated location only a few times, mostly at the beginning of the first sentence spoken at each angle. This behavior helps avoiding erroneous steering caused by noise when speech is absent or too soft. A moving sound source, on the other hand, would possibly be wrongly steered after the first estimation.

MUSIC changed its estimation for nearly every speech sample it received. While the location estimation during voiced speech is fairly good, during short pauses or unvoiced speech, MUSIC tried to estimate the location of the loudest noise source with the highest correlation between the microphones, i.e., its built-in fan during pauses and unvoiced speech. Since the RMSE gives a higher weight to larger errors due to its quadratic nature, this error becomes significantly larger. The frequent changing of the location estimation, on the other hand, would possibly make MUSIC be better suited for moving speakers or when Pepper is allowed to move its head freely.

3.2. Experiment 2: Keyword Spotter

To evaluate the difference in ASR performance between Pepper and Google Cloud Speech-to-Text, their keyword spotting performances were compared. The results showed a KWER of 34.5% for Pepper's keyword spotter and 28.3% for GC-STT. So, the ASR engine of GC-STT outperforms that of Pepper's built-in keyword spotter. A two-tailed t-test showed that GC-STT performed slightly better than Pepper's keyword spotter ($\mathrm{p} < 0.25$

Inspection of Pepper's results showed that its keyword spotter had difficulty with the monologue included in the dialogue. The keyword spotter lagged behind on the keywords to listen to after it had wrongly understood a keyword, which propagated through the entire monologue. Excluding the monologue, Pepper's KWER was 32.4% and that of the GC-STT 24.4%. Although the KWER of both systems decreases, the GC-STT still outperforms Pepper's keyword spotter. GC-STT performs even better compared to Pepper's keyword spotter after removing the monologue, which is caused by the lower amount of keywords left over. (TO DO: je zegt dat Pepper problemen heeft met de monoloog, je zegt niets over Google, maar als je de monoloog weghaalt uit de KWER berekening heeft Google er meer profijt van dan Pepper; dus google heeft meer last van de monoloog dan Pepper, toch??? Dit moet je denk ik anders opschrijven.)

3.3. Experiment 3: Speech-to-Text Transcription

To evaluate the SSL and beamformer pipeline, transcription of the beamformed speech was compared to that of the unbeamformed speech signal with GC-STT. Fig. 3 shows the WER for the different SNRs for the unbeamformed speech signal (dotted line with circles)

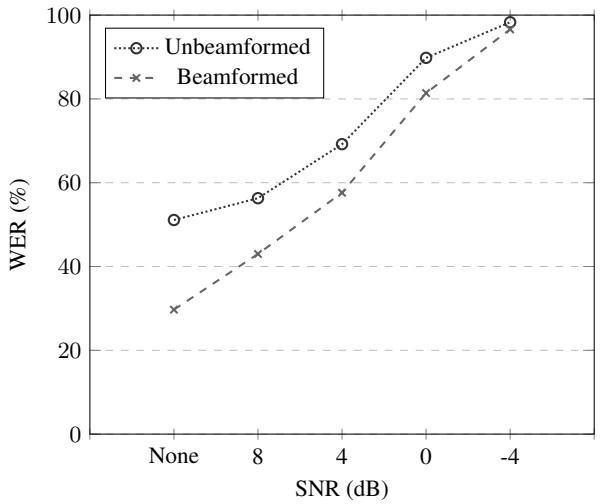


Fig. 3. WER of the transcription task with the Google Cloud Text-to-Speech system on the unbeamformed and beamformed speech signal.

and the beamformed speech signal (dashed line with crosses). As can be seen, the beamformed speech signal consistently outperforms the unbeamformed speech signal, although this difference reduces to almost 0 for the worst SNR of -4 dB. Without the presence of background noise ('None' in Fig. 3), GC-STT obtains an accuracy that is 21.4% higher for the beamformed speech signal compared to the unbeamformed speech signal. This clearly shows that the delay-and-sum beamformer is able to remove a large part of the noise generated by Pepper's built-in fan.

At SNRs of 0 dB or lower, the WER increases to more than 80%, which would make interaction with the Pepper robot in a social setting basically impossible.

Analysis of the transcription results showed that "jij/je/jou" ("you/your") were often confused. On the one hand, this could be due to the high phonological similarity between these words and their occurrence in highly similar places in utterances. On the other hand, both "jij" and "jou" are often reduced to "je". It is thus possible that the talkers unintentionally mispronounced these words.

4. DISCUSSION AND CONCLUSION

Despite social robots being increasingly more often used in human-robot dialogues and in noisy environments, little is known their ability to deal with continuous speech and background noise, and this is especially the case for the Pepper social robot. This paper is one of the first to investigate this. Specifically, we investigated 1) whether Peppers built-in keyword spotter could be replaced by an ASR system in order for it to deal with Dutch continuous speech; 2) whether Peppers ASR pipeline can be made more robust against noise, without the need to change Peppers hardware. To that end, in three experiments, Peppers built-in Sound Source Localization algorithm and keyword spotter are compared to a newly proposed pipeline using SSL based on MUSIC, a delay-and-sum beamformer, and Google Cloud Speech-to-Text. Moreover, the proposed pipeline is tested in cafeteria background noise.

The first experiment showed that Pepper's built-in SSL outperformed the MUSIC SSL, with an RMSE that was 7.6 degrees lower for Pepper's SSL. Eventhough MUSIC gave worse results in the steering test, it has been used in the pipeline as it used near-field steering vectors which take distance from the robot into account. Besides this, since MUSIC steers away mostly during silences which are not the focus of ASR systems, the effects of this should be minimal. The second experiment showed that changing Pepper's keyword spotter with Google Cloud speech-to-text yielded a decrease of 6.2% absolute in keyword error rate. The final experiment investigated the effect of the proposed delay-and-sum beamformer on automatic speech recognition. A comparison of the recognition of the raw speech signal with the beamformed speech signal showed a decrease of more than 20% in WER in clean listening conditions. At a signal-to-noise ratio (SNR) of +8 dB, the proposed pipeline showed a 13.3% improvement over the unbeamformed speech which persisted in more difficult SNRs.

It is important to note that the condition without added background noise did in fact have a substantial amount of noise. The fan, located underneath the microphones in Pepper's head, and the actuators, especially the one in Pepper's neck, create clearly audible noise in all recordings. To create good transcriptions, these noises need to be attenuated without causing distortion to the audio. The "clean" listening condition in Experiment 3 showed that the proposed delay-and-sum beamformer was able to substantially attenuate the noise from the fan and actuators, although the noise was still audible in the beamformed speech signal. Pepper's ability to recognize speech could be further improved if the fan noise and at least the usage of the neck actuator would be minimized. Alternatively, a different beamformer could be used. One of the options would be a Minimum Variance Distortionless Response (MVDR) beamformer [25]. This is an adaptive beamformer which adjusts the weights of the microphones to minimize the variance, but does not distort in the direction of the source signal. Another option is the Generalized Sidelobe Canceller (GSC) [26]. This system combines a simple beamformer and a sidelobe canceller which aims at cancelling noise from non-source directions. Although these beamformers could possibly improve the quality of the speech signal compared to the used delay-and-sum beamformer, they might increase complexity. Moreover, due to the limited number of microphones that Pepper has, these beamformers are expected to increase Pepper's performance relatively little compared to the delay-and-sum beamformer. Finally, due to their complexity, it is not an easy feat to get these beamformers to work fast enough for a real-time environment with a significant increase in performance.

In our experiments, we used two-dimensional sound source localization; however, SSL algorithms often have the option to be used in three dimensions. A small-scale test using the three dimensional algorithm showed that the difference between the WERs of the beamformed audio in two or three dimensions is at most 0.8% absolute. For Pepper, the elevation of the beamformer has far less influence on the recognition of the speech than the azimuth. Keeping the (pseudo-)real-time limitation in mind, it would therefore be better to perform two dimensional SSL with a higher angular resolution than a three dimensional SSL.

We chose to use the MUSIC algorithm for sound source localization, despite its worse results compared to Pepper's in-built SSL because MUSIC takes the distance between the talker and the robot in account when creating its steering vectors. However, for an improved pipeline with less external processing, it could be possible to use Pepper's SSL instead of an external algorithm. Pepper's SSL is always running in the background, and as its performance seems to be better than MUSIC SSL, using this would decrease processing time and might allow for a more robust beamforming algorithm.

Care should be taken though, as it is mentioned Pepper's SSL's performance decreases in environments with a less then 3 dB SNR [21].

Keeping in mind the real-time aspect of this research, the cloud solution gives an answer in pseudo real-time. Pseudo real-time is assumed to be quick enough to not feel you have to wait for it. The limit for near real-time for this research is set to 500ms. Pepper's keyword spotter has a very fast recognition which resulted in the recognition being finalized and closed during a small pause in speech, before the speaker was done with its sentence. Google Cloud STT and the full pipeline took longer than Pepper's keyword spotter. A recording of 10 seconds was processed in approximately 9 seconds, where more than half of the time was used by MUSIC. When the system will be used live, it therefore should perform in near real-time. The average latency observed of GC-STT is about 300-400 ms, not accounting for the latency caused by the set-up of the connection.

In conclusion, Pepper's keyword spotter can be replaced with an ASR system, such as Google Cloud STT. While Pepper's keyword spotter responds much faster than a cloud solution, a lower Keyword Error Rate can be achieved within near real-time. Moreover, using the Google Cloud STT instead of Pepper's ASR system makes it possible to preprocess the speech signal using a delay-and-sum beamformer to make the pipeline more robust to noise. Although the results in background noise need to be improved before Pepper can be used in a spoken dialogue scenario in a location with background noise, the results presented here show that it spoken interactions with a Pepper robot in relatively quiet conditions is already possible.

5. REFERENCES

[1] Roberto Pinillos, Samuel Marcos, Raul Feliz, Eduardo Zalama, and Jaime Gómez-García-Bermejo, "Long-term assessment of a service robot in a hotel environment," *Robotics and Autonomous Systems*, vol. 79, pp. 40–57, 2016.

[2] Masahiro Shiomi, Kazuhiko Shinozawa, Yoshifumi Nakagawa, Takahiro Miyashita, Toshio Sakamoto, Toshimitsu Terakubo, Hiroshi Ishiguro, and Norihiro Hagita, "Recommendation effects of a social robot for advertisement-use context in a shopping mall," *International Journal of Social Robotics*, vol. 5, no. 2, pp. 251–262, 2013.

[3] Koen V Hindriks, Roel Boumans, Fokke B van Meulen, Mark Neerincx, and MGM Olde Rikkert, "An interview robot for collecting patient data in a hospital," *Ercim News*, vol. 2018, no. 114, 2018.

[4] F. Tanaka, K. Isshiki, F. Takahashi, M. Uekusa, R. Sei, and K. Hayashi, "Pepper learns together with children: Development of an educational application," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, Nov 2015, pp. 270–275.

[5] Carlos Toshinori Ishi, Shigeki Matsuda, Takayuki Kanda, Takatoshi Jitsuhiro, Hiroshi Ishiguro, Satoshi Nakamura, and Norihiro Hagita, "A Robust Speech Recognition System for Communication Robots in Noisy Environments," *IEEE TRANSACTIONS ON ROBOTICS*, vol. 24, no. 3, 2008.

[6] Kerstin Fischer, "How people talk with robots: Designing dialog to reduce user uncertainty," *AI Magazine*, vol. 32, no. 4, pp. 31–38, 2011.

[7] B. Gonsior, C. Landsiedel, A. Glaser, D. Wollherr, and M. Buss, "Dialog strategies for handling miscommunication in task-related hri," in *2011 RO-MAN*, July 2011, pp. 369–375.

[8] Vasant Srinivasan and Leila Takayama, "Help me please: Robot politeness strategies for soliciting help from humans," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2016, CHI '16, pp. 4945–4955, ACM.

[9] James Kennedy, Séverin Lemaignan, Caroline Montassier, Pauline Lavalade, Bahar Irfan, Fotios Papadopoulos, Emmanuel Senft, and Tony Belpaeme, "Child speech recognition in human-robot interaction: Evaluations and recommendations," in *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, New York, NY, USA, 2017, HRI '17, pp. 82–90, ACM.

[10] Shun'ichi Yamamoto, Jean Marc Valin, Kazuhiro Nakadai, Jean Rouat, François Michaud, Tetsuya Ogata, and Hiroshi G. Okuno, "Enhanced robot speech recognition based on microphone array source separation and missing feature theory," in *Proceedings - IEEE International Conference on Robotics and Automation*. 2005, vol. 2005, pp. 1477–1482, IEEE.

[11] "Pepper - softbank robotics," `https://www.softbankrobotics.com/us/pepper`.

[12] "Pepper supported languages," `http://doc.aldebaran.com/2-5/family/pepper_technical/languages_pep.html`.

[13] R. C. Hendriks, T. Gerkmann, and J. Jensen, *DFT-Domain Based Single-Microphone Noise Reduction for Speech Enhancement: A Survey of the State of the Art*, Morgan & Claypool, 2013.

[14] Kris Hermus, Patrick Wambacq, et al., "A review of signal subspace speech enhancement and its application to noise robust speech recognition," *EURASIP Journal on Advances in Signal Processing*, vol. 2007, no. 1, pp. 045821, 2006.

[15] Andrew Maas, Quoc V Le, Tyler M Oneil, Oriol Vinyals, Patrick Nguyen, and Andrew Y Ng, "Recurrent neural networks for noise reduction in robust asr," 2012.

[16] "Google cloud speech-to-text," `https://cloud.google.com/speech-to-text/`.

[17] R. Schmidt, "Multiple Emitter Location and Signal Parameter," Tech. Rep. 3, 1986.

[18] Navin Kumar and Alka Singh, "Study of sound source localization using music method in real acoustic environment," in *International Journal of Electronics Engineering Research*. ISSN, 2017, vol. 9, pp. 545–556.

[19] Ahmed Abdalla, Suhad Mohammed, Abdelazeim Abdalla, Tang Bin, and Mohammed Ramadan, "A Study of a various Acoustic Beamforming Techniques Using a Microphone Array," *Journal of Communications Technology, Electronics and Computer Science*, vol. 1, no. 0, pp. 7, oct 2015.

[20] DJ Allred, "Evaluation and comparison of beamforming algorithms for microphone array speech processing," 2006.

[21] "Pepper documentation," `http://doc.aldebaran.com/2-5/`.

[22] Robin Scheibler, Eric Bezzam, and Ivan Dokmani, "Pyroomacoustics: A python package for audio room simulation and array processing algorithms," 10 2017.

[23] Ngoc-Vinh Vu, Hua Ye, Jim Whittington, John Devlin, and Michael Mason, "Small footprint implementation of dual-microphone delay-and-sum beamforming for in-car speech enhancement," in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2010, pp. 1482–1485.

[24] Paul Boersma & David Weenink, "Praat: doing phonetics by computer [computer program]," `http://www.praat.org/`.

[25] Byung-Chul Kim and I-Tai Lu, "High resolution broadband beamforming based on the mvdr method," in *OCEANS 2000 MTS/IEEE Conference and Exhibition. Conference Proceedings (Cat. No. 00CH37158)*. IEEE, 2000, vol. 3, pp. 1673–1676.

[26] J. Bitzer, K.U. Simmer, and K.-D. Kammeyer, "Theoretical noise reduction limits of the generalized sidelobe canceller (GSC) for speech enhancement," 2008, pp. 2965–2968 vol.5.

Bibliography

[1] Roberto Pinillos, Samuel Marcos, Raul Feliz, Eduardo Zalama, and Jaime Gomez-Garcia-Bermejo. Long-term assessment of a service robot in a hotel environment. *Robotics and Autonomous Systems*, 79:40–57, 2016.

[2] Masahiro Shiomi, Kazuhiko Shinozawa, Yoshifumi Nakagawa, Takahiro Miyashita, Toshio Sakamoto, Toshimitsu Terakubo, Hiroshi Ishiguro, and Norihiro Hagita. Recommendation effects of a social robot for advertisement-use context in a shopping mall. *International Journal of Social Robotics*, 5(2):251–262, 2013.

[3] Koen V Hindriks, Roel Boumans, Fokke B van Meulen, Mark Neerincx, and M G M Olde Rikkert. An Interview Robot for Collecting Patient Data in a Hospital. *Ercim News*, 2018 (114), 2018.

[4] F Tanaka, K Isshiki, F Takahashi, M Uekusa, R Sei, and K Hayashi. Pepper learns together with children: Development of an educational application. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 270–275, nov 2015. doi: 10.1109/HUMANOIDS.2015.7363546.

[5] Wayne Xiong, Lingfeng Wu, Fil Alleva, Jasha Droppo, Xuedong Huang, and Andreas Stolcke. The Microsoft 2017 conversational speech recognition system. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5934–5938. IEEE, 2018.

[6] Ana Rodrigues, Rita Santos, Jorge Abreu, Pedro Beça, Pedro Almeida, and Silvia Fernandes. Analyzing the performance of ASR systems. In *Proceedings of the XX International Conference on Human Computer Interaction*, pages 1–8. ACM, 2019. doi: 10.1145/3335595.3335635.

[7] Kazuhiro Nakadai, Hiroshi G Okuno, Hirofumi Nakajima, Yuji Hasegawa, and Hiroshi Tsujino. An open source software system for robot audition HARK and its evaluation. In *2008 8th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2008*, pages 561–566, 2008. ISBN 9781424428229. doi: 10.1109/ICHR.2008.4756031.

[8] Carlos Toshinori Ishi, Shigeki Matsuda, Takayuki Kanda, Takatoshi Jitsuhiro, Hiroshi Ishiguro, Satoshi Nakamura, and Norihiro Hagita. A Robust Speech Recognition System for Communication Robots in Noisy Environments. *IEEE TRANSACTIONS ON ROBOTICS*, 24(3), 2008. doi: 10.1109/TRO.2008.919305.

[9] Simone Varrasi, Alexander Lucas, Alessandro Soranzo, John McNamara, and Alessandro Di Nuovo. IBM cloud services enhance automatic cognitive assessment via human-robot interaction. In *New Trends in Medical and Service Robotics*, pages 169–176. Springer, 2019.

[10] Michiel de Jong, Kevin Zhang, Aaron M Roth, Travers Rhodes, Robin Schmucker, Chenghui Zhou, Sofia Ferreira, João Cartucho, and Manuela Veloso. Towards a robust interactive and learning social robot. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 883–891. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[11] Richard C Hendriks, Timo Gerkmann, and Jesper Jensen. Dft-domain based single-microphone noise reduction for speech enhancement: a survey of the state of the art. *Synthesis Lectures on Speech and Audio Processing*, 9(1):1–80, 2013.

[12] Kris Hermus, Patrick Wambacq, and Others. A review of signal subspace speech enhancement and its application to noise robust speech recognition. *EURASIP Journal on Advances in Signal Processing*, 2007(1):45821, 2006.

[13] Andrew Maas, Quoc V Le, Tyler M O'neil, Oriol Vinyals, Patrick Nguyen, and Andrew Y Ng. Recurrent neural networks for noise reduction in robust ASR. 2012.

[14] R. Schmidt. Multiple Emitter Location and Signal Parameter Estimation. Technical Report 3, 1986.

[15] Navin Kumar and Alka Singh. Study Of Sound Source Localization Using Music Method In Real Acoustic Environment. In *International Journal of Electronics Engineering Research*, volume 9, pages 545–556. ISSN, 2017.

[16] Ahmed Abdalla, Suhad Mohammed, Abdelazeim Abdalla, Tang Bin, and Mohammed Ramadan. A Study of a various Acoustic Beamforming Techniques Using a Microphone Array. *Journal of Communications Technology, Electronics and Computer Science*, 1(0):7, oct 2015. ISSN 2457-905X. doi: 10.22385/jctecs.v1i0.3.

[17] D J Allred. Evaluation and comparison of beamforming algorithms for microphone array speech processing. 2006.

[18] Xavier Anguera, Chuck Wooters, and Javier Hernando. Acoustic beamforming for speaker diarization of meetings. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(7):2011–2022, 2007.

[19] Bruce T Lowerre. The HARPY speech recognition system. Technical report, Carnegie-Mellon University Pittsburgh, Dept. of Computer Science, 1976.

[20] Odette Scharenborg. *Narrowing the gap between automatic and human word recognition*. PhD thesis, Radboud University, 2005.

[21] Rivarol Vergin, Douglas O'shaughnessy, and Azarshid Farhat. Generalized mel frequency cepstral coefficients for large-vocabulary speaker-independent continuous-speech recognition. *IEEE Transactions on speech and audio processing*, 7(5):525–532, 1999.

[22] Hynek Hermansky. Perceptual linear predictive (PLP) analysis of speech. *the Journal of the Acoustical Society of America*, 87(4):1738–1752, 1990.

[23] Biing Hwang Juang and Laurence R Rabiner. Hidden Markov models for speech recognition. *Technometrics*, 33(3):251–272, 1991.

[24] Michael L Seltzer, Dong Yu, and Yongqiang Wang. An investigation of deep neural networks for noise robust speech recognition. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 7398–7402. IEEE, 2013.

[25] Lawrence R Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[26] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, and Others. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

[27] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing*, 20(1):30–42, 2011.

[28] Navdeep Jaitly, Patrick Nguyen, Andrew Senior, and Vincent Vanhoucke. Application of pretrained deep neural networks to large vocabulary speech recognition. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[29] Masahiro Shiomi, Takayuki Kanda, Dylan F Glas, Satoru Satake, Hiroshi Ishiguro, and Norihiro Hagita. Field trial of networked social robots in a shopping mall. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2846–2853. IEEE, 2009.

[30] Martin Cooke, Phil Green, Ljubomir Josifovski, and Ascension Vizinho. Robust automatic speech recognition with missing and unreliable acoustic data. *Speech communication*, 34(3):267–285, 2001.

[31] Jingdong Chen, Jacob Benesty, Yiteng Huang, and Simon Doclo. New Insights Into the Noise Reduction Wiener Filter. *IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING*, 14(4):1218–1233, 2006. doi: 10.1109/TSA.2005.860851.

[32] Junhui Qian, Zishu He, Wei Zhang, Yulong Huang, Ning Fu, and Jonathon Chambers. Robust adaptive beamforming for multiple-input multiple-output radar with spatial filtering techniques. *Signal Processing*, 143:152–160, 2018.

[33] Salvador H Talisa, Kenneth W O'Haver, Thomas M Comberiate, Matthew D Sharp, and Oscar F Somerlock. Benefits of digital phased array radars. *Proceedings of the IEEE*, 104(3):530–543, 2016.

[34] Adam C Luchies and Brett C Byram. Deep neural networks for ultrasound beamforming. *IEEE transactions on medical imaging*, 37(9):2010–2021, 2018.

[35] Giulia Matrone, Alessandro Ramalli, Alessandro Stuart Savoia, Piero Tortoli, and Giovanni Magenes. High frame-rate, high resolution ultrasound imaging with multi-line

transmission and filtered-delay multiply and sum beamforming. *IEEE transactions on medical imaging*, 36(2):478–486, 2016.

[36] Caleb Rascon and Ivan Meza. Localization of sound sources in robotics: A review. *Robotics and Autonomous Systems*, 96:184–210, 2017.

[37] Joseph H DiBiase, Harvey F Silverman, and Michael S Brandstein. Robust localization in reverberant rooms. In *Microphone Arrays*, pages 157–180. Springer, 2001.

[38] Pogula Rakesh, S. Siva Priyanka, and T. Kishore Kumar. Performance evaluation of beamforming techniques for speech enhancement. In *2017 4th International Conference on Signal Processing, Communication and Networking, ICSCN 2017*, pages 1–5. IEEE, mar 2017. ISBN 9781509047406. doi: 10.1109/ICSCN.2017.8085647. URL `http://ieeexplore.ieee.org/document/8085647/`.

[39] Robin Scheibler, Eric Bezzam, and Ivan Dokmanić. Pyroomacoustics: A Python Package for Audio Room Simulation and Array Processing Algorithms. 2017. doi: 10.1109/ICASSP.2018.8461310.

[40] Ngoc-Vinh Vu, Hua Ye, Jim Whittington, John Devlin, and Michael Mason. Small footprint implementation of dual-microphone delay-and-sum beamforming for in-car speech enhancement. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1482–1485. IEEE, 2010.

[41] Ronald Mucci. A comparison of efficient beamforming algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(3):548–558, 1984.

[42] David Weenink and Paul Boersma. Praat: doing phonetics by computer [Computer program]. URL `http://www.praat.org/`.

[43] Google. Google Protocol Buffers, 2019. URL `https://developers.google.com/protocol-buffers/docs/overview`.

[44] Kuldip K Paliwal, James G Lyons, and Kamil K Wójcicki. Preference for 20-40 ms window duration in speech analysis. In *2010 4th International Conference on Signal Processing and Communication Systems*, pages 1–4. IEEE, 2010.

[45] William T Cochran, James W Cooley, David L Favin, Howard D Helms, Reginald A Kaenel, William W Lang, George C Maling, David E Nelson, Charles M Rader, and Peter D Welch. What is the fast fourier transform? *Proceedings of the IEEE*, 55(10): 1664–1674, 1967.

[46] Odette Scharenborg, Francesco Ciannella, Shruti Palaskar, Alan Black, Florian Metze, Lucas Ondel, and Mark Hasegawa-Johnson. Building an asr system for a low-resource language through the adaptation of a high-resource language asr system: Preliminary results. *Proceedings of ICNLSSP, Casablanca, Morocco*, 2017.

[47] Laurence Gillick and Stephen J Cox. Some statistical issues in the comparison of speech recognition algorithms. In *International Conference on Acoustics, Speech, and Signal Processing,*, pages 532–535. IEEE, 1989.

[48] Byung-Chul Kim and I-Tai Lu. High resolution broadband beamforming based on the MVDR method. In *OCEANS 2000 MTS/IEEE Conference and Exhibition. Conference Proceedings (Cat. No. 00CH37158)*, volume 3, pages 1673–1676. IEEE, 2000.

[49] Joerg Bitzer, Klaus Uwe Simmer, and K-D Kammeyer. Theoretical noise reduction limits of the generalized sidelobe canceller (gsc) for speech enhancement. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258)*, volume 5, pages 2965–2968. IEEE, 1999.

[50] Carlos Avendano, Sangita Tibrewala, and Hynek Hermansky. Multiresolution channel normalization for ASR in reverberant environments. In *Fifth European Conference on Speech Communication and Technology*, 1997.