Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

What are the current capabilities for a speech interface using existing speech libraries on a mobile robot with limited hardware resources?

Felix Jerye

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Marco Aiello
Supervisor:	Prof. Dr. Marco Aiello
Commenced:	July 2, 2021
Completed:	January 3, 2022

Abstract

Interaction with robots often requires a speech interface. Particularly robots with many concurrent processes, need a resource efficient solution. The available frameworks are often tightly coupled, making them less suitable for an deep integration. This work presents a modular prototype speech interface based on the Robot Operating System. The individual modules are interchangeable and logically separated. Special attention is paid to the weak hardware of a Raspberry Pi 4. Benchmarks are used to show that this platform is suitable for a real-time speech interface. This does not apply to all software components. Especially GPU-optimized components reach their limits and miss real-time.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

Acronyms

AI Artificial Intelligence. 22

DDS Data Distribution Service. 21

IDL Interface Description Language. 21

NLU Natural-language understanding. 17, 22

RNN Recurrent Neural Network. 22

ROS Robot Operating System. 17, 21

RTPS Real-Time Publish Subscribe. 21

SSML Speech Synthesis Markup Language. 23

STT Speech-to-text. 17, 22

TTS Text-to-speech. 17, 23

1 Introduction

Robots already perform a wide range of tasks these days. They take over highly automated tasks in factories[GIBS17], assist in medical operations[TDY19] or simply mow the lawn[Ver20]. The mentioned examples share the characteristic that they master (a few) specific tasks. This simplifies the design of interfaces for programming or configuring a robot, since only a certain group of people interact with these machines. Interfaces can be appropriately designed for operation, e.g. via the browser or a hardware controller.

The situation is different for robots, which are more connected to the surrounding environment and interact with people who otherwise have nothing to do with the robot. This puts the robot in situations that cannot be handled in advance due to sheer numbers. Some form of communication and ambient intelligence is required.

Example: A manager of the local supermarket wants to use a robot to stock the shelves. The main task is to retrieve goods from the storage and put them in place accordingly. However, since this is done during operation, customers come by during the work process.
It often happens that customers do not find a desired product in the store. It would be expected of a human employee to be able to provide this information, whether the store carries this product and if so, where it is located. If necessary, the employee then takes the customer there in person.
In order not to reduce the service level, the robot must therefore be able to perform at least the same tasks. To achieve this, the robot requires a speech interface and access to appropriate information. Since customers should not be encouraged to read an instruction manual, communication must be intuitive and versatile.

That comes with some challenges. The software must understand what is being said, interpret it, and provide a corresponding response. To be satisfying, the whole process must happen in an acceptable time, preferably in real time. The answer must be understandable and meaningful.
In addition to the technical challenges, there are also legal ones. In the European Union, the General Data Protection Regulation (GDPR) and national laws must be considered. Not every service or use case is compatible with it.

In this work, the focus is mainly on a speech interface. A software project was created that integrates various existing libraries/services for the Robot Operating System (ROS). It follows the usual design for speech interfaces which start by recording audio, partitioning it, transcribing it into text, trying to understand the text, generating an appropriate response, then turning the response back as an audio sample and outputting it. For a speech interface, therefore, a composition of several technologies is necessary: Speech-to-text (STT), Natural-language understanding (NLU) and Text-to-speech (TTS) are of particular importance here and will be discussed later on.

Implementation details are given in Chapter 3, technical backgrounds of the underlying technique can be accessed in Chapter 4. Performance analysis on a Raspberry Pi 4 is covered in Chapter 5. The summary and outlook can be found in Chapter 6.

2 Related Work

This chapter shows several related works that have investigated a speech interface for robots, especially for ROS. Closely related topics are also addressed.

In the work of Lane et al. [LPS+12], a framework based on ROS is presented that brings a speech input interface which relies not only on spoken language, but also on gestures and eye tracking. The results are forwarded to an NLU service, which must map the different inputs to corresponding indents and actions.

Giving a robot a multilingual speech interface in ROS is the goal of Hofer and Strohmeier [HS19]. Before the speech input is transcribed, an attempt is made to identify the correct language. The authors mention a significant delay with their framework, which still needs to be improved to achieve real-time.

Megalingam et al. [MKP21] presents a ROS speech interface that uses Google's cloud services for STT and TTS. The implementation is limited to a handful of commands. The work shows that a quick response with real time capabilities is possible, but requires in this case a continuous Internet connection.

The authors Wang and Shi [WS19] also implemented a speech interface using ROS that achieves real-time characteristics in this case on an unknown laptop computer.

Literature specializing in speech and speech interfaces combined with ROS is available to some extent, but many are outdated in terms of ROS version or used technology or are a one-time project, making further progression not possible.

3 Background

This chapter outlines various technologies and techniques used by the speech interface. It serves as an overview for a rough understanding of very extensive and complex topics that are beyond the scope of this work.

3.1 Robot Operating System

The Robot Operating System (ROS)[1] is a open-source middle-ware for robot software development. It provides a set of libraries and tools to build a robot application. ROS is designed to run on a single computer as well as in a cluster in real time.
To achieve this, the framework builds on top of the Data Distribution Service (DDS)[2] standard and the Real-Time Publish Subscribe (RTPS)[3] protocol for communication. The DDS specification describes among other details a publish-subscribe pattern:

- A *Topic* is a data structure described by the Interface Description Language (IDL).
- A *Domain* organizes a set of topics in a tree structure. For example with `my/domain/plain_text`, `my/domain/audio_result` the topics `plain_text` and `audio_result` will be part of the domain `my/domain`.
- A *Publisher* producing data by making them available to a specific topic in a DDS system.
- A *Subscriber* consumes data by listening to a specific topic in a DDS system.

The concrete implementations of DDS/RTPS are interchangeable.

To get value out of the system, ROS organizes its components in the form of nodes. A node can register any number of publishers and subscribers.
In addition to the publish-subscribe pattern, ROS also offers a request-response pattern, which is also based on the DDS.

The nodes usually run in independent processes, but can also be started as a single process with a little extra effort.

[1] https://www.ros.org/
[2] https://www.omg.org/spec/DDS/
[3] https://www.omg.org/spec/DDSI-RTPS/

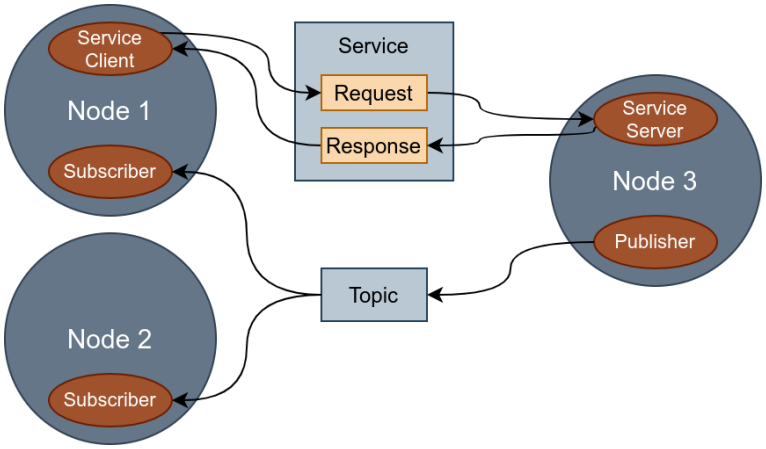

Figure 3.1: Illustration of nodes, topics, publisher, subscriber and services

When it comes to choosing an operating system, the choice[4] is limited. Only Ubuntu (20.04) gets a tier 1 support for both amd64 and arm64 architectures.

Windows 10 with VS2019 also has tier 1 support, but only for amd64, no distribution-specific packages and a number of pitfalls.

3.2 Speech-to-text

A Speech-to-text (STT) engine is software that attempts to transcribe audio samples. Some challenges are different languages, dialects, pronunciation and various types of inaccuracies.
There are many different approaches, but in recent years neural network based have dominated. These rely on well-optimized Recurrent Neural Network (RNN) training systems and a sufficiently strong CPU/GPU for runtime. With the help of large speech datasets, the models are trained. One example model is Deep Speech[HCC+14].

3.3 Natural-language understanding

When it comes to understanding a text, Natural-language understanding (NLU) comes into action. A NLU software should understand and interpret a input text with the help of an Artificial Intelligence (AI).

[4] https://www.ros.org/reps/rep-2000.html#galactic-geochelone-may-2021-november-2022

3.3.1 Intent classification

There is an unmanageable amount of possibilities what a user wants to express with a sentence. In order to break these down to an application, so-called intents are used. An AI is trained to provide an vecotor of intent probabilities for given input. Based on the this, a number of intents can then be selected. Following this, any necessary actions are executed and optional a meaningful response will provided.

3.4 Text-to-speech

The speech synthesis is an artificially way to imitate the human speech. A Text-to-speech (TTS) engine is a software that takes some kind of text as input and converts it into speech. Some software, in addition to the plain text, also supports the Speech Synthesis Markup Language (SSML)[5]. This allows among other things to integrate different voices, influence pronunciation and pauses in the synthesis process.

Listing 3.1 SSML example

```
<speak>
  I speak something.
  <break time="3s"/>
  After the break, I will clearly emphasize <emphasis>this short segment</emphasis>.
</speak>
```

There are various TTS methods. The ones described here are used by the libraries later in Section 4.5.

3.4.1 Formant synthesis

The formant synthesis method additively generates speech from a set of vowels and diphones. Base frequency, noise and intonation can be varied over time.
This method creates an artificial, unnatural sounding language, which can be clear and understandable, but never comes close to natural speech. The advantages are very fast synthesis and small memory usage.

3.4.2 Neural networks

There are different synthesis models which are based on deep learning. They mostly share the characteristic that they need a strong GPU for fast execution. The calculation steps are divided into the following parts.

[5] https://www.w3.org/TR/speech-synthesis/

1. **Text-to-Spectrogram:** A spectrogram is a representation of the spectrum of frequencies that vary with time. A Mel spectogram is a representation adapted to the Mel scale, which includes the peculiarities of human hearing.
   Models such as Tacotron[WSS+17], Tacotron2[SPW+18] and Glow-TTS[KKKY20] use neural networks to generate these mel-spectograms.
2. **Vocoders:** A spectogram is not yet an audio sample. To reverse this, so-called vocoders are used. Vocoders like HiFiGAN[KKB20] and WaveNet[ODZ+16] try to generate efficient and high-fidelity raw audio waveforms.

4 Implementation

This chapter covers the different packages, their structure and the underlying implementations.

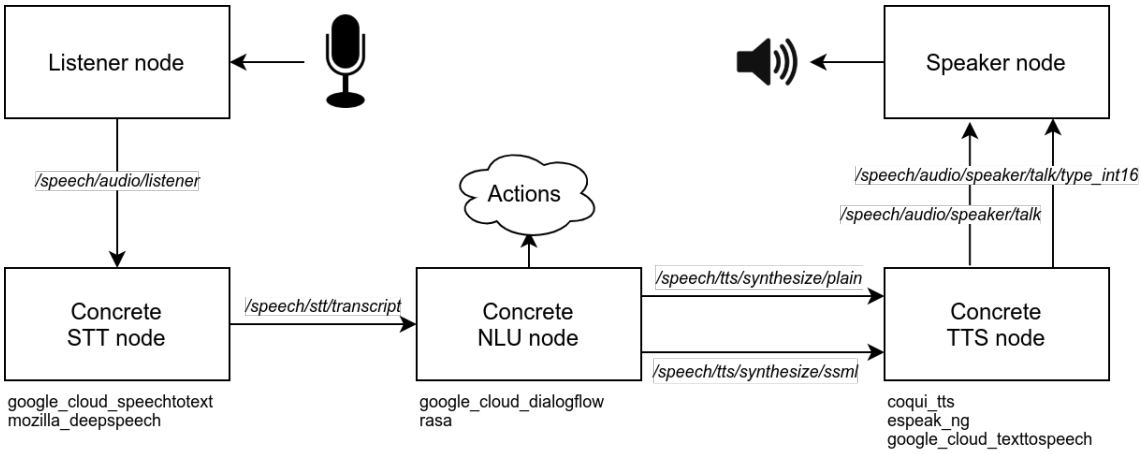

Figure 4.1: Graphical representation of the speech interface

4.1 Requirements

Due to the architecture of ROS, software components should be logically arranged in packages. As long as this is done carefully, packages can be combined in a modular way.

The programming language C/C++ and Python[1] are directly supported by ROS. There are projects that integrate other languages, such as the *ROS 2 Java client library*[2]. In this work C99/C++17 and python3 are used.

For the build system, colcon[3] is used by default in combination with cmake[4] and setuptools[5].

The hardware used for development and deployment is listed in Table 4.1. An overview of the packages can be found in Table 4.2.

[1] https://docs.python.org/3/
[2] https://github.com/ros2-java/ros2_java
[3] https://colcon.readthedocs.io/
[4] https://cmake.org/documentation/
[5] https://setuptools.pypa.io/
[6] https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

	Desktop PC	**Raspberry Pi 4 Model B**[6]
Target	Development	Deployment
ROS version	ROS2 Galactic Geochelone	
Operating System	Ubuntu Desktop (20.04 LTS)	Ubuntu Server (20.04 LTS)
Architecture	AMD64	ARM64
RAM	16GB	8GB
CPU	4x 3.6GHz	4x 1.5GHz
GPU	GeForce GTX 1060 6GB	VideoCore VI

Table 4.1: Hardware Specs

ROS package	**lang**	**type**	**internet?**	**important libs**
audio_msgs	-	Utils	-	-
audio	Python	Audio I/O	-	pyaudio, webrtcvad
google_cloud_speechtotext	Python	STT	runtime	google-cloud-speech
mozilla_deepspeech	Python	STT	during build	deepspeech
google_cloud_dialogflow	Python	NLU	runtime	google-cloud-dialogflow
rasa	Python	NLU	no	rasa
coqui_tts	Python	TTS	no	TTS
espeak_ng	C/C++	TTS	no	espeak-ng (GPLv3)
google_cloud_texttospeech	Python	TTS	runtime	google-cloud-texttospeech

Table 4.2: Package overview

4.2 Audio packages

audio_msgs

The `audio_msgs` package contains ROS messages related to audio. At the time of development there were no standard packages for audio formats. Two message types are provided for both Python and C/C++. `Wav` a int16 wav format encoded as an byte array and `WavInt16` a int16 wav format encoded as an int16 array.

audio

The `audio` package contains to major nodes, the `listener` and `speaker` node.

The `listener` node takes as input a continuous audio stream, divides it into frames (default is 200ms). These frames are stored in a ring buffer. There they are constantly classified with the help of the library `webrtcvad`[7] whether a frame has language activity or not. As soon as a certain threshold is exceeded, the ring buffer will be published as a Wav message.

The `speaker` node simply listens to the incoming `Wav` and `WavInt16` messages and outputs them sequentially to the speaker.

[7] https://github.com/wiseman/py-webrtcvad

For cross-platform interoperability with the hardware (microphones and speakers) portaudio[8] is used, respectively the python wrapper PyAudio[9].

4.3 STT packages

All STT packages working with the same principle. A concrete STT node receives an audio sample. The sample is then processed, configured and forwarded to the specific software. If the transcription is successful, the result is passed on to an NLU node.

google_cloud_speechtotext

The `google_cloud_speechtotext` package contains a node which uses Google's client library to access the cloud service. A valid json API key is required during build and runtime. The provided service is based on RNN models.

mozilla_deepspeech

The `mozilla_deepspeech` package wraps up the `DeepSpeech`[10] speech-to-text engine. During the build, an existing model with associated scorer is downloaded. There are several models available. These can be specified in advance via the config. The engine is based on RNNs, but with no precompiled AArch64 binaries. Manual compilation is required for this platform.

4.4 NLU packages

The NLU packages have the task of processing the input, executing appropriate actions and generating a corresponding response. A simple run looks like this:
An NLU node is waiting for incomming text messages. Then this is configured and forwarded to the specific software used. The text will be analyzed and if necessary a predefined action is executed. Finally, a text response is returned, which is sent as a message to an TTS node.

google_cloud_dialogflow

The `google_cloud_dialogflow` package is an integration of Google's Dialogflow as an ROS node. This intent classification based cloud service comes along with many tools and extensions. Actions can either be executed directly in the cloud (via webhooks) or sent back to the client. A valid json API key is required.

rasa_nlu

The `rasa_nlu` package implements Rasa[11], an open source conversational AI. This also uses an intent classification and model training. This software can be used completely offline.
Actions are written in advance with python. The model is trained during the ROS build process and will be deployed. Rasa uses dependencies without precompiled binaries for AArch64, which makes

[8] http://www.portaudio.com/
[9] https://pypi.org/project/PyAudio/
[10] https://github.com/mozilla/DeepSpeech
[11] https://github.com/RasaHQ/rasa

integration a more challenging.
In addition, the library does not properly work with ROS. Using a model that works in a seperate test script without problems causes runtime errors in the package. This problem could not be fixed, so rasa was tested separately outside of ROS later on.

4.5 TTS packages

A concrete TTS package can accept plain text or SSML as input. This is then processed accordingly and one or more synthesized audio samples are sent to the speaker as a result.

coqui_tts

The `coqui_tts` package integrates coquiTTS[12] a library with many different deep learning models. By default a provided Tactron2 model is used in combination with an HiFiGAN vocoder model. Other provided models can be configured in advance. Some dependencies has missing precompiled binaries for AArch64.

espeak_ng

The `espeak_ng` package integrates eSpeakNG[13]. The software uses the formant synthesis, a fast and resource-saving synthesizer, which generates a unnatural sounding speech. This synthesizer usually generates several small audio samples that are sent before the actual completion.
The non-permissive license GPLv3 must be considered.

google_cloud_texttospeech

The `google_cloud_texttospeech` package is an integration of Google's text-to-speech engine. The cloud services are based on RNN and a valid json API key is also required.

[12] https://github.com/coqui-ai/TTS
[13] https://github.com/espeak-ng/espeak-ng

5 Evaluation

This chapter analyzes and discusses the message latency caused by the DDS and ROS and the performance of the STT, NLU, and TTS packages.

5.1 Message latency

ROS integration of the DDS is an integral part of the communication between individual nodes. A separate benchmark package was developed for testing all used messages. Special attention is paid to the performance differences between Python and C/C++.
Overall, three message types were used in the implementation: `String` from the std_msgs provided by ROS; `Wav` and `WavInt16` from the custom audio_msgs package.
Data is generated deterministically for the message latency benchmarks. The length/size of this data can be set by parameter. Each message variant is measured 50 times with a different payload. The results are saved in csv files for later analysis.

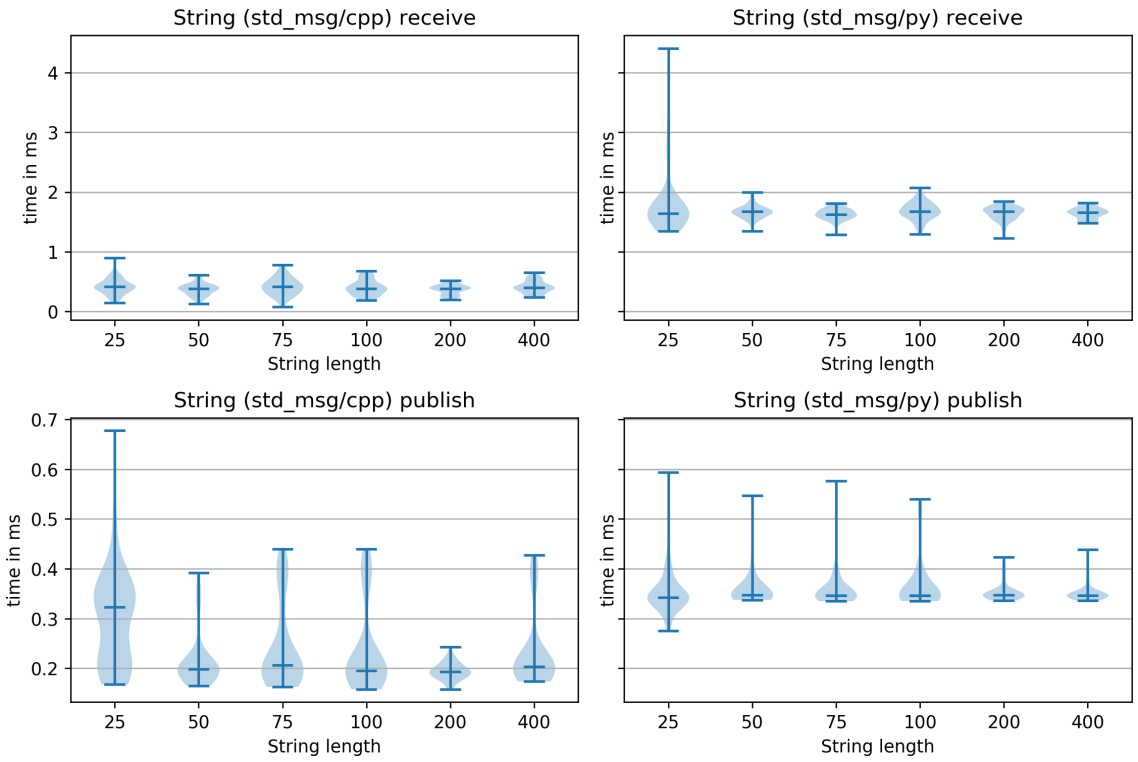

Figure 5.1: Benchmark *String* message

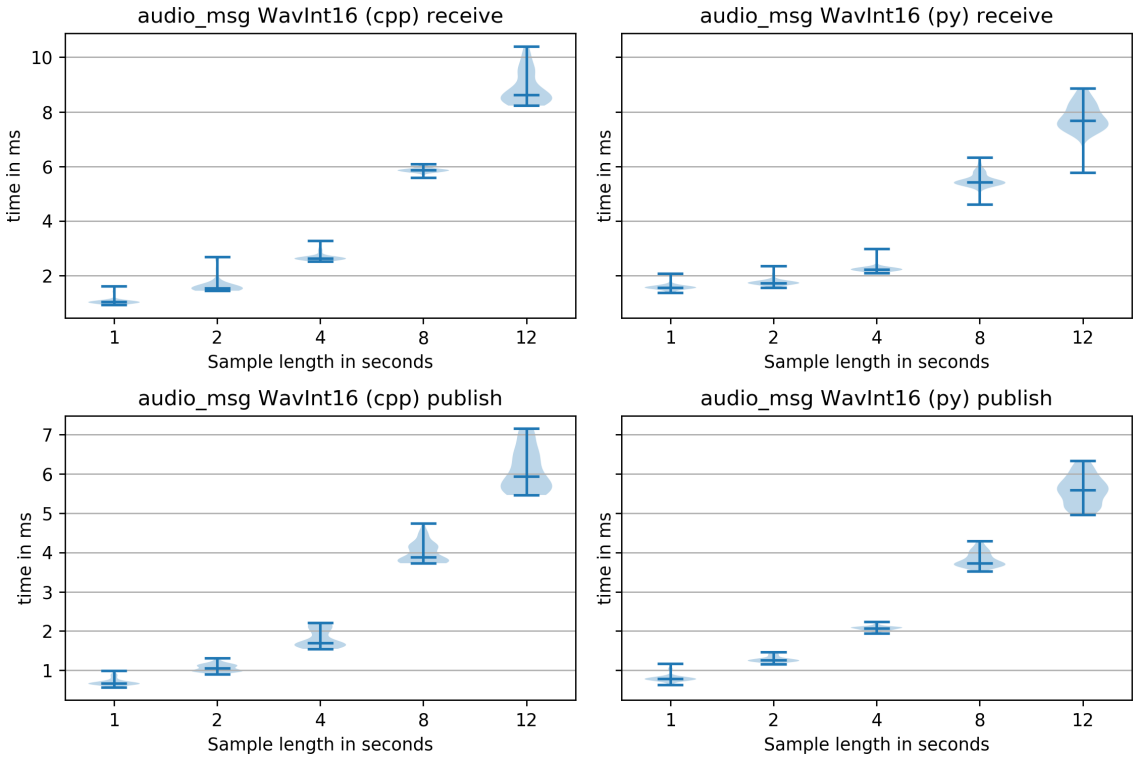

Figure 5.2: Benchmark *WavInt16* message

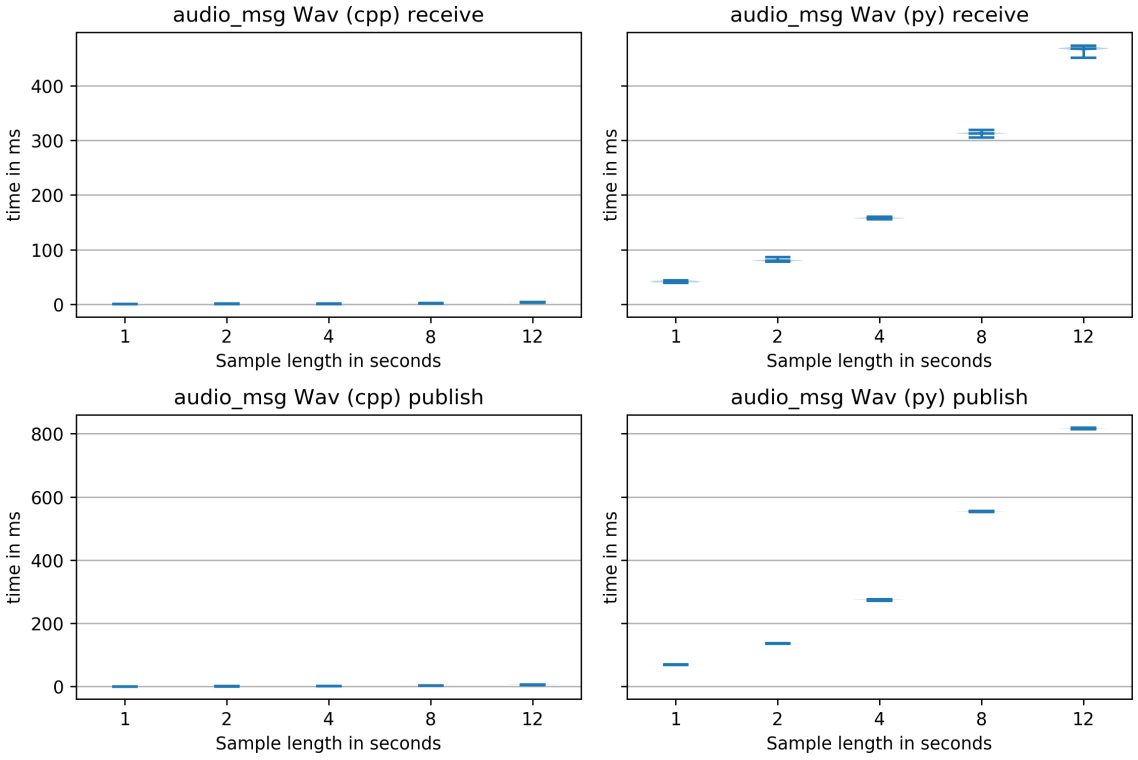

Figure 5.3: Benchmark *Wav* (byte encoded int16) message

The results show a similar results between Python and C++. A small difference between the hardware-oriented language C++ and the interpreted language Python is to be expected.
The string messages are relatively constant and more than 400 characters are usually not needed at once for a speech interface. In relation to the performance, 2ms are acceptable.
For the WavInt16 message type no significant differences are noticeable. The time increases linearly in relation to the sample length and a speed of under 7ms is sufficient.
It looks different for the Wav byte encoded int16 messages. Due to a bug in ROS bytearrays for Python have to be processed additionally over the whole length, which costs $\theta(n)$ per de-/serialization. To send 8 seconds of audio as byte encoded array, publishing and receiving costs almost a second. In comparison, a python WavInt16 message with the same data would cost less than 13ms.

5.2 Performance

5.2.1 Test design

The performance tests are designed as a black box. Only one node is tested at a time. In addition, a second BenchNode is started, which sequentially sends the input data and measures how long it takes for a result to arrive. The input data is repeated 18 times. Packages with platform limitations (see Chapter 4) were run on the development computer. The results are saved in a `csv` file for later analysis.

The STT nodes get audio samples as input (see Listing 5.1) in expectation of an transcription. The NLU nodes get a text as input (see Listing 5.2) and flat intents without successor state enables sequential sending. The TTS nodes get one or more sentences as input (see Listing 5.3) to synthesize.

Listing 5.1 STT evaluation audio samples

```
1: Yes
2: Where is the next exit?
3: How will be the weather be tomorrow?
4: When does the next train to Stuttgart leave?
5: I want a table for two. Is there anything still free today?
```

Listing 5.2 NLU evaluation inputs

```
1: set the alarm to 7 am
2: set another alarm to 8 pm
3: is my alarm set for 8 pm
4: remove all alarms
```

Listing 5.3 TTS evaluation sentences

```
1: How are you doing?
2: It's 8 o'clock.
3: Room 1.2.3 is located on the second floor on the west side.
4: You do not have authorization for this operation.

5: I will make a quick look up in the archives. One moment of patience please.
6: The message was forwarded to Mr. Lachs. Is there anything else I can do for you?
7: Two of three test results are already available. The first two results are negative.
8: The butter is in the refrigerated section next to the eggs. Shall I take you there?
```

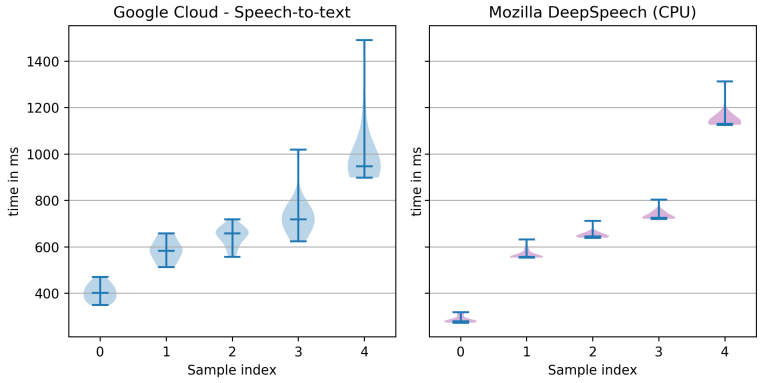

Figure 5.4: STT performance results

5.2.2 Results

The results are divided into the different package types. Coloring of the results in blue means an execution on the Raspberry Pi 4, in purple an execution on the development computer (see Table 4.1) and grey without ROS on the development computer.
The internet connection used was up to 100Mbit in download and 40Mbit in upload.

STT: The Figure 5.4 shows the test results. It can be clearly seen that as the length of the sample increases, the computation time will rise. Running a client for Google's cloud service on a weak computer is similar to CPU-based computation with Project Deepspeech on a more powerful one.

NLU: The Figure 5.5 shows a fast response of Dialogflow with a stable response time less than 250ms normally. For the CPU-based version of Rasa, the response time is similarly fast on the stronger device. Though the message latency is missing, it can be assumed that this would only cost a few milliseconds extra.

TTS: Figure 5.6 shows a super fast execution with eSpeak NG. With less 15ms response time, a significant margin to the other two implementations. Google's cloud service also fulfills real-time characteristics with less than 400ms. The situation is different for CPU-based coquiTTS with a synthesis time of often more than two seconds.

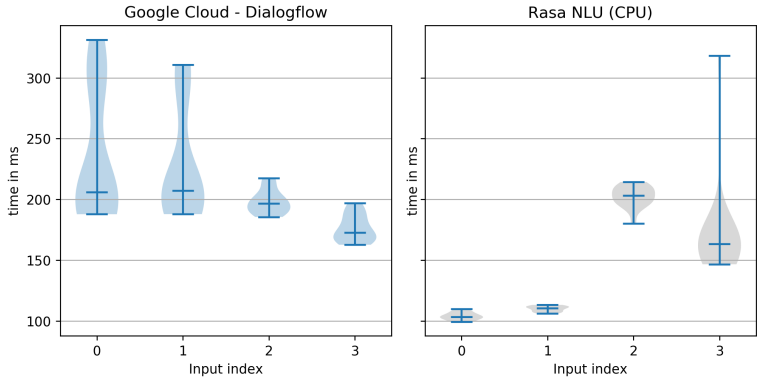

Figure 5.5: NLU performance results

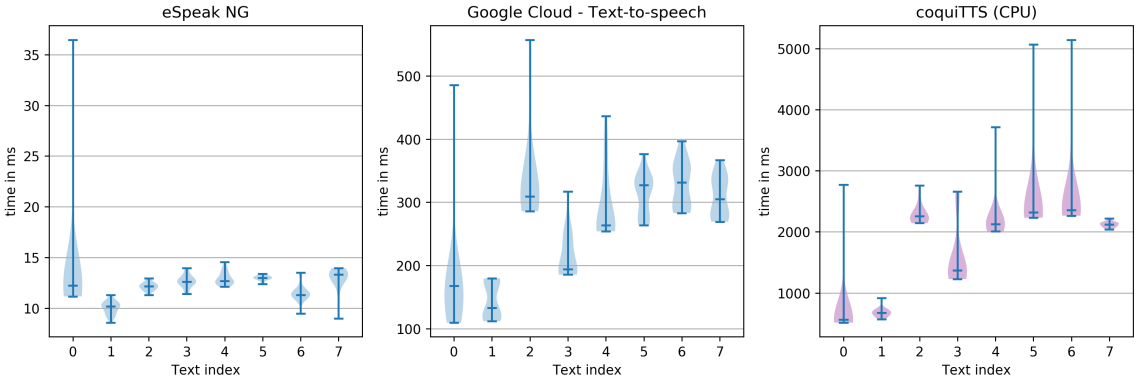

Figure 5.6: TTS performance results

5.2.3 Discussion

To provide a robot speech interface, a combination of time efficient STT, NLU and TTS packages must be chosen. The fastest pipline would be Google's STT and Dialogflow in combination with eSpeakNG with an average of less than 1s. But eSpeakNG is not considered a pleasing synthesizer. Going entirely on Google products would only increase the time by 120-320ms.

Cloud services compensate well for the weak performance of a robot. But also offline solutions, such as DeepSpeech and Rasa turn out to be promising. Due to the hardware, it can be assumed that an execution on the Raspberry Pi 4 will take a multiple of the time. If these packages are to be used, it is advisable to use hardware with the possibility of GPU-based execution. Unfortunately, this could not be tested in this work.

Another problem is that coquiTTS has to perform in the program code with two unnecessary operations. The result is converted from a float encoded sample into a byte array and then into the byte array Wav message variant.
Deepspeech also takes significantly more time to receive messages, depending on the sample size.

In addition neural network-based speech packages do not eliminate the need for high efficient but limited solutions. Especially in exceptional situations where a robot needs all resources possible, a more cost-effective alternative may be appropriate.

These tests have shown that a speech interface for a ROS based robot with limited hardware is possible. It was also shown that not all libraries are suitable for this purpose.

6 Conclusion and Outlook

This chapter summarizes the work and provides an outlook for future work.

The goal of this work was a prototype speech interface implementation with ROS targeting limited hardware. The test result shows that the execution is possible in real time depending on the package. The overall execution time ranges from less than one to a few seconds.
Embedding a speech interface in the ROS architecture works well and has potential through a modular design. There were problems especially with ARM AArch64 platforms, because necessary binaries are not included and it's difficult to get them up and running in a reasonable time. In addition there are also bugs in the ROS Client Library for Python which significantly reduces the execution time as the message latency measurements have shown.

The prototype is feature incomplete. It does not yet use the full potential. Besides optimizations, configurability and security, the following aspects should be considered:

GPU support: `DeepSpeech`, `Rasa NLU` and `coquiTTS` utilize torch and/or tensorflow. These libraries support execution on the GPU, but that requires a compatible CUDA GPU[1]. As a result, only an execution on the CPU is possible with the Raspberry Pi 4. One promising option would be to use an NVIDIA Jetson Nano, which is quite similar to the Rasperry Pi 4, but comes with CUDA cores. Other AI-based modules in a robot would also benefit.

Accessibility: Language is not limited to the spoken. There are many deaf, mute and blind people. For a robot to be socially integrated on a broad scale, it must also make use of other forms of speech. The following points should be considered:

- Sign language as a speech input.
- The speech as visually represented text.
- The speech output as Braille.

Multilingualism: There are many different languages that can often come together in a confined space. In the future, a robot speech interface should be able to understand and output common languages. As mentioned in Chapter 2, there is already such a work with ROS, which is not yet a production ready method.

ROS package index & maintaining: An unspecified number of users regularly look for ways to use a speech interface. The existing implementations are often outdated or not properly integrated into the ROS architecture. As a result, similar things are created over and over again by different people and groups. A way of collaborative development and maintenance should be figured out.

[1] https://developer.nvidia.com/cuda-gpus

Bibliography

[GIBS17] A. Grau, M. Indri, L. L. Bello, T. Sauter. "Industrial robotics in factory automation: From the early stage to the Internet of Things". In: *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*. 2017, pp. 6159–6164. DOI: 10.1109/IECON.2017.8217070 (cit. on p. 17).

[HCC+14] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, A. Y. Ng. *Deep Speech: Scaling up end-to-end speech recognition*. 2014. arXiv: 1412.5567 [cs.CL] (cit. on p. 22).

[HS19] D. P. Hofer, F. Strohmeier. "Multilingual speech control for ROS-driven robots". In: 2019. DOI: 10.1007/s00502-019-00739-y (cit. on p. 19).

[KKB20] J. Kong, J. Kim, J. Bae. *HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis*. 2020. arXiv: 2010.05646 [cs.SD] (cit. on p. 24).

[KKKY20] J. Kim, S. Kim, J. Kong, S. Yoon. *Glow-TTS: A Generative Flow for Text-to-Speech via Monotonic Alignment Search*. 2020. arXiv: 2005.11129 [eess.AS] (cit. on p. 24).

[LPS+12] I. Lane, V. Prasad, G. Sinha, A. Umuhoza, S. Luo, A. Chandrashekaran, A. Raux. "HRItk: The Human-Robot Interaction ToolKit Rapid Development of Speech-Centric Interactive Systems in ROS". In: Association for Computational Linguistics. July 2012. URL: https://aclanthology.org/W12-1817.pdf (cit. on p. 19).

[MKP21] R. K. Megalingam, A. H. Kota, V. K. T. P. "Optimal Approach to Speech Recognition with ROS". In: *2021 6th International Conference on Communication and Electronics Systems (ICCES)*. 2021, pp. 111–116. DOI: 10.1109/ICCES51350.2021.9489085 (cit. on p. 19).

[ODZ+16] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu. *WaveNet: A Generative Model for Raw Audio*. 2016. arXiv: 1609.03499 [cs.SD] (cit. on p. 24).

[SPW+18] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerry-Ryan, R. A. Saurous, Y. Agiomyrgiannakis, Y. Wu. *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions*. 2018. arXiv: 1712.05884 [cs.CL] (cit. on p. 24).

[TDY19] J. Troccaz, G. Dagnino, G.-Z. Yang. "Frontiers of Medical Robotics: From Concept to Systems to Clinical Translation". In: *Annual Review of Biomedical Engineering* 21.1 (2019). PMID: 30822100, pp. 193–218. DOI: 10.1146/annurev-bioeng-060418-052502. eprint: https://doi.org/10.1146/annurev-bioeng-060418-052502. URL: https://doi.org/10.1146/annurev-bioeng-060418-052502 (cit. on p. 17).

[Ver20] G. B. Verne. “Adapting to a Robot: Adapting Gardening and the Garden to Fit a Robot Lawn Mower”. In: *Companion of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*. HRI ’20. Cambridge, United Kingdom: Association for Computing Machinery, 2020, pp. 34–42. ISBN: 9781450370578. DOI: 10.1145/3371382.3380738. URL: https://doi.org/10.1145/3371382.3380738 (cit. on p. 17).

[WS19] J. Wang, Z. Shi. “A Speech Interaction System Based on Cloud Service under ROS”. In: *2019 Chinese Control Conference (CCC)*. 2019, pp. 4721–4725. DOI: 10.23919/ChiCC.2019.8865929 (cit. on p. 19).

[WSS+17] Y. Wang, R. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, Q. Le, Y. Agiomyrgiannakis, R. Clark, R. A. Saurous. *Tacotron: Towards End-to-End Speech Synthesis*. 2017. arXiv: 1703.10135 [cs.CL] (cit. on p. 24).

All links were last followed on December 27, 2021.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature