UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Block-based Speech-to-Speech Translation

Degree's Thesis

Audiovisual Systems Engineering

Author: Sandra Roca

Advisor: Xavier Giró i Nieto

Universitat Politècnica de Catalunya (UPC)

October, 2018

Abstract

This bachelor’s thesis explores different ways of building a block-based Speech Translation system with the aim of generating huge amounts of parallel speech data.

The first goal is to research and manage to run suitable tools to implement each one of the three blocks that integrates the Speech Translation system: Speech Recognition, Translation and Speech Synthesis.

We experiment with some open-source toolkits and we manage to train a speech recognition system and a neural machine translation system. Then, we test them to evaluate their performance.

As an alternative option, we use the cloud computing solutions provided by Google Cloud to implement the three sequential blocks and we successfully build the overall system.

Finally, we make a comparative study between an in-house software development versus Cloud computing implementation.

Resum

Aquesta tesi explora diferents maneres d'implementar un sistema de blocs de Traducció de Veu amb la finalitat de generar un gran corpus paral·lel de veu.

La primera tasca consisteix en cercar i aconseguir dominar eines adequades per a implementar cada un dels tres blocs que integra el sistema de traducció de veu: reconeixement de veu, traducció, i síntesi de veu.

Experimentem amb algunes eines de codi obert i aconseguim entrenar un sistema de reconeixement de veu i una màquina de traducció neuronal. Posteriorment, els sotmetem a test per tal d'evaluar el seu rendiment.

Com a opció alternativa, utilitzem les solucions d'Informàtica en núvol (*Cloud Computing*) proporcionades per Google Cloud per a implementar els tres blocs seqüencials i elaborem el sistema global amb èxit.

Finalment, fem un estudi comparatiu entre el desenvolupament de software intern i la implementació *Cloud computing*.

Resumen

Esta tesis explora diferentes maneras de implementar un sistema de bloques de Traducción de Voz con el propósito de generar grandes cantidades de datos para generar un gran corpus paralelo de voz.

La primera tarea consiste en buscar y conseguir dominar herramientas adecuadas para implementar cada uno de los tres bloques que integran el sistema de traducción de voz: reconocimiento de voz, traducción y síntesis de voz.

Experimentamos con algunas herramientas de Código abierto y conseguimos entrenar un sistema de reconocimiento de voz y una máquina de traducción neuronal. Posteriormente, los sometemos a test con el fin de evaluar su rendimiento.

Como opción alternativa, usamos las soluciones de Computación en la nube (*Cloud Computing*) proporcionadas por Google Cloud para implementar los tres bloques secuenciales y elaboramos el sistema global con éxito.

Finalmente, hacemos un estudio comparativo entre el desarrollo de software interno y la implementación *Cloud Computing*.

Acknowledgements

First of all, I want to thank my tutor, Xavier Giró-i-Nieto, for making possible my collaboration in this project and, most of all, for his help and guidance during the whole project.

I would also like to thank Amanda Duarte, who has been implied in the project from the very beginning and has provided her wide knowledge of deep learning.

Albert Gil also deserves appreciation for his help with the GPI servers.

Finally, I want to thank to my family and friends for their support.

Revision history and approval record

Revision	Date	Purpose
0	27/09/2018	Document creation
1	6/10/2018	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Sandra Roca Cobo	sandra.roca.cobo@alu-etsetb.upc.edu
Xavier Giró i Nieto	xavier.giro@upc.edu
Marta Ruiz	marta.ruiz@upc.edu

Written by:		Reviewed and approved by:	
Date	27/09/2018	Date	8/10/2018
Name	Sandra Roca	Name	Xavier Giró i Nieto
Position	Project Author	Position	Project Supervisor

Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

The motivation of implementing a Speech Translation system is related to the DeepLipDub project[1], carried out at UPC. The aim of that project is to synthesize speech translated from the source language of a video sequence to a target language and, simultaneously, to replace the pixels of the speaker's lips so that the substitutes are synchronized with the translated speech.

That project is divided in two branches: speech translation and lip dubbing.

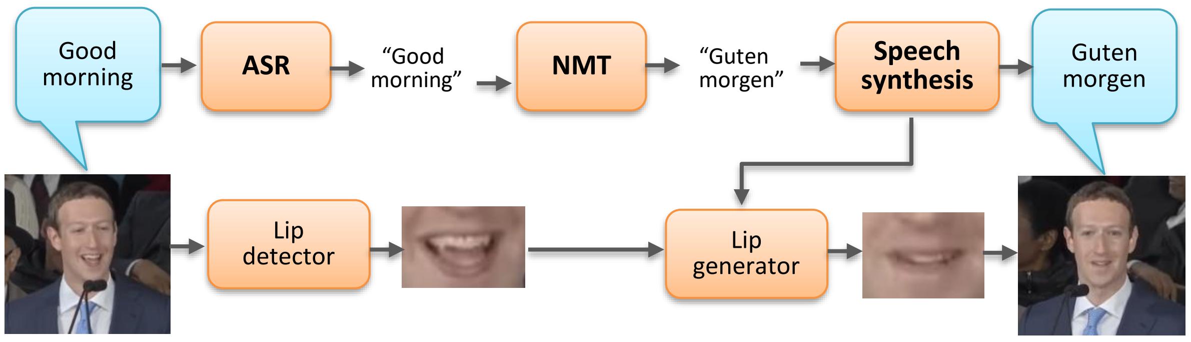


Figure 1. DeepLipDub block diagram

The first branch consists of three blocks: an Automatic Speech Recognizer (ASR) to get the transcripts of the video, a Neural Machine Translator (NMT) to translate the output of the previous block to a target language and, finally, a Speech Synthesizer to generate the audio stream of the translated transcripts.

In parallel, the second branch has a first block that detects the pixels belonging to the speaker's lips and a second block that uses the combination of the Speech Synthesizer output and the Lip detector's output to modify that pixels in accord with the content of the speech synthesizer.

Since a large amount of data is required to train deep learning, the implementation proposed in this thesis will be used as a tool to generate an English-German parallel speech corpus, which will be useful for training an end-to-end Speech Translation system.

1.2. Statement of purpose

Speech-to-speech translation (S2ST) is the process by which the phrases spoken in one language are immediately translated and spoken in another language.

The most important use case for Speech translation is, of course, in travel. It also has many potential applications in many different contexts such as medical facilities, schools, police, hotels, business and much more. This technology is also useful to perform automatic dubbing, which is the task that concerns us for the development of this thesis.

The most typical Speech Translation systems integrate three software technologies: Automatic Speech Recognition (ASR), Machine Translation (MT) and Speech Synthesis (TTS).

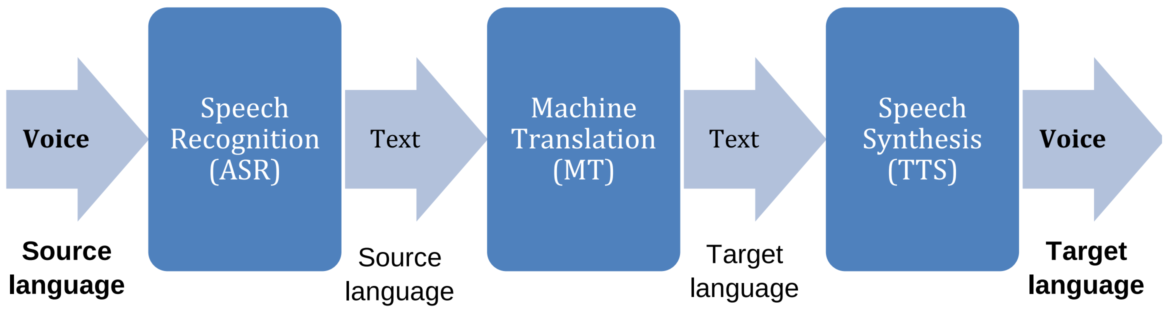


Figure 2. Typical Speech-to-Speech Translation structure

Nevertheless, it's not enough to just chain a really good ASR system with a really good MT system and a speech synthesizer. One of the most inconvenient facts is that the way people talk is not the same as the way they write. That's why corpora play a crucial role in developing speech-to-speech translation technologies.

Regarding the component technologies that integrates a S2ST system, there are so many specific corpora to train each of the blocks independently. But what about if we want to train the overall system? In that case, if we are going to train a combined model of Speech Recognition and Machine Translation, we need a corpus of input speech labelled with the corresponding transcriptions and translations into a target language.

This means collecting a huge amount of spoken data and the corresponding transcriptions in both source and target languages. As it will take a lot of time to achieve that, as well as experts in that languages would be required, we come up with a solution: a Speech Translation implementation that given audio files of spoken language as input, it automatically generates the corresponding spoken translation into the target language.

1.3. Requirements and specifications

The requirements of the project are the following:

- Find suitable open source toolkits to implement the three blocks of the S2ST system: Automatic Speech Recognition (ASR), Neural Machine Translation (NMT) and Speech Synthesis (TTS).
- Train an ASR system
- Train a NMT system
- Manage to run a Speech Synthesizer software
- Build the block-based Speech Translation system

The specifications are the following:

- Python as a programming language
- Use PyTorch implementations to train the models
- Use the Google Cloud APIs to build the Cloud-based solution

1.4. Methods and procedures

As this thesis aims to study the difference between an In-house or Cloud-based software development of Speech-to-Speech translation, it is necessary to find easy-to-use tools for both cases.

Regarding the In-house software development choice, the procedure we follow consists in training an Automatic Speech Recognition system and then training as well a Neural Machine Translation system in order to later concatenate them.

As far as the Cloud-based implementation is concerned, we use the APIs that provide Google Cloud to implement the overall system. Making the appropriate calls to the speech-to-text, translate and text-to-speech APIs, we achieve an excellent performance of the Speech Translation system.

1.5. Work plan

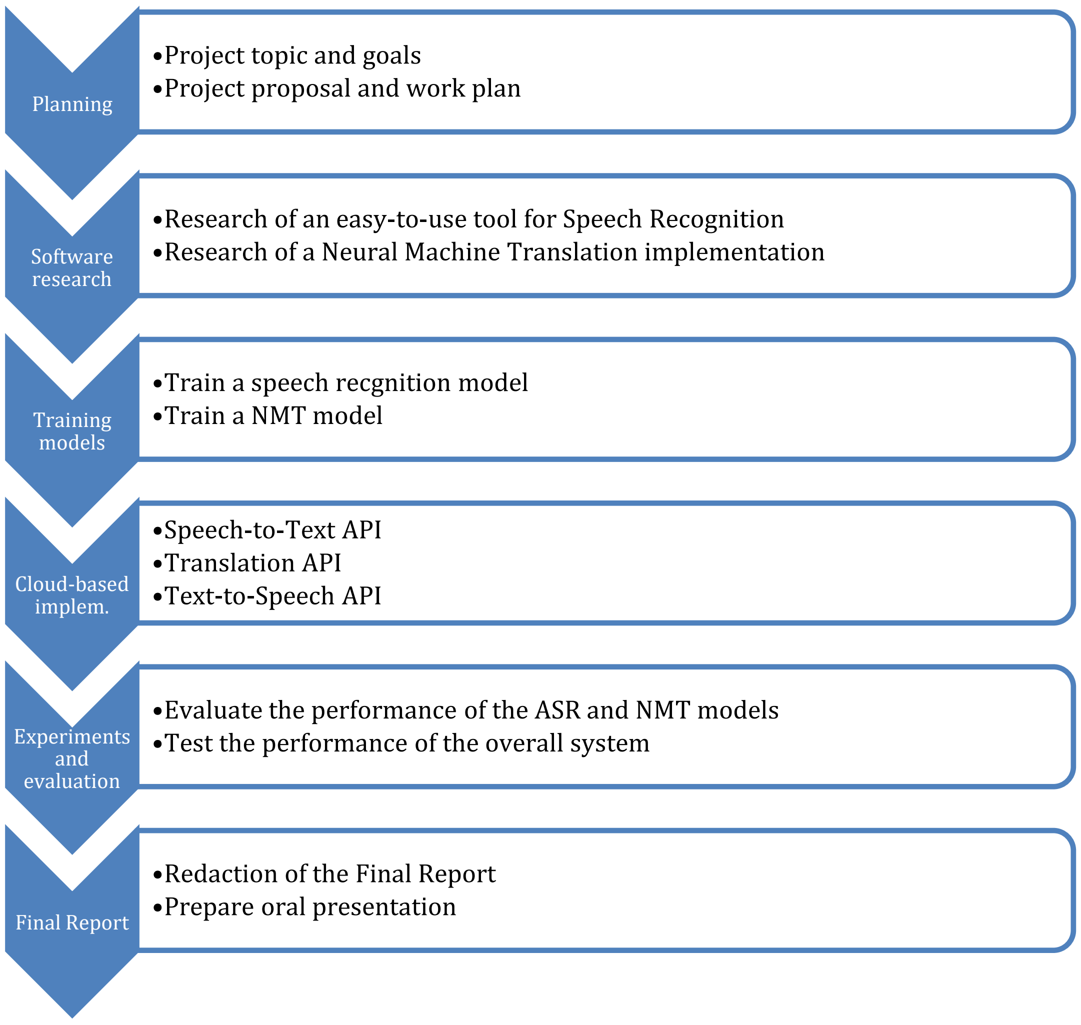

Planning
•Project topic and goals
•Project proposal and work plan
Software research
•Research of an easy-to-use tool for Speech Recognition
•Research of a Neural Machine Translation implementation
Training models
•Train a speech recgnition model
•Train a NMT model
Cloud-based implem.
•Speech-to-Text API
•Translation API
•Text-to-Speech API
Experiments and evaluation
•Evaluate the performance of the ASR and NMT models
•Test the performance of the overall system
Final Report
•Redaction of the Final Report
•Prepare oral presentation

Figure 3. Work breakdown structure

1.6. Incidences and modifications

The initial goal of this project was to build a Speech Translation system by building our own models for Speech Recognition and Neural Machine Translation.

For that purpose, I used an open source toolkit for each task. What I did not expect is that I would face so many problems with the software dependencies and the installation of extra required libraries.

Another challenge I had to face up was that there were some errors on the code and it was so difficult to identify the piece of code the errors were caused by.

After solving these issues, we finally manage to train both Speech Recognition and Neural Machine Translation models. Unfortunately, the performance of that models was not so good and, apart from that, it was not possible to concatenate the ASR system with the NMT because of incompatible data formats.

As we had spent so much time trying to run these toolkits and then training the models, it had been so laborious to look for and try other tools to achieve better results. That’s why we decided to move on to a Cloud-based implementation using Google Cloud APIs.

2. State of the art

In this chapter we will revise some of the literature involved in the process of building a speech-to-speech translation system and we will also make a review of the successes so far.

Speech translation technology has solved the language barrier problem and has enabled natural language communication between people that do not share the same language.

In today's state-of-the-art systems we see a workflow with three separated systems:

- A speech recognition system that identifies the words spoken and transcode them into text.
- A machine translation system that translate the text of the source speech into the target language.
- A speech synthesizer to go from the translated text to spoken words.

These systems are pretty robust and work well for their intended purpose. However, as it is not an end-to-end Speech translation system but a block-based system, we lose important characteristics of the voice when we transcode from the speech signal into a text representation and it makes it difficult to generate a good dynamic voice.

2.1. Automatic Speech Recognition

Speech Recognition is the process that enables the recognition and "translation" of spoken language into text. Due to the ongoing research in this area for several years, so many approaches for this task has been developed.

Based on the speech production mechanism of humans, an ASR system aims to infer the words of a speech given the observable signal.

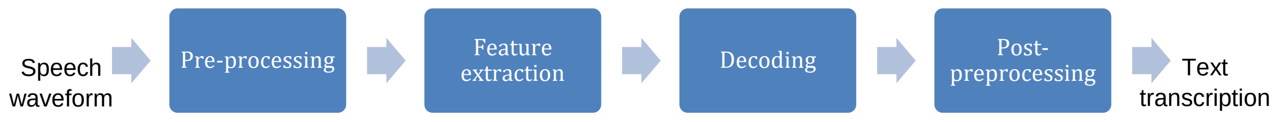


Figure 4. Typical Speech Recognition system architecture

The decoding module is as well composed by the subblocks: acoustic model, language model and pronunciation dictionary.

2.2. Neural Machine Translation

Neural machine translation (NMT) is an end-to-end approach of Machine Translation, which can be defined as the process by which a source text in one language is automatically converted to text in another language.

NMT uses neural network models to learn to translate text based on existing statistical models, and this makes it possible to condition the probability of words that are generated at each position on all the previous words in the output sentence. The deep learning architecture that the system uses is capable of learning the meaning of the text and thanks to that, the machine performs the translation task at a semantic level. Thus, the output of the system is a fluent and naturally sounding translation.

It has de ability to produce faster and higher quality output than Statistical Machine Translation (SMT).

Another characteristic of these state-of-the-art approach is that NMT systems understand similarities between words and can benefit from that.

2.3. Speech synthesis

Speech synthesis is the artificial simulation of human speech carried out by a computer or another electronic device. All speech synthesizers are based on the model of Human Speech Production in order to achieve the most possible natural sounding of the synthetic speech.

As well as written language can be seen as a sequence of elementary, we can define speech as elementary acoustic symbols also known as phonemes.

Based on this comparison, a general speech synthesizer has the following structure:

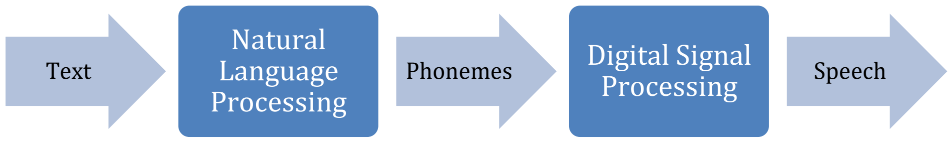


Figure 5. General speech synthesizer structure

Given some text as a input, the NLP module produces the corresponding phonetic transcription and then de DSP module is responsible for converting the phonemes into artificial speech.

The Natural Language Processing module consists in the composition of 3 major components:

- the text analyser, which study the morpho-syntactical functions of the words
- the letter-to-sound component, responsible for making the conversion between words and phonemes
- the prosody generation component, which contributes to intonation, tone and rhythm

As far as the Digital Signal Processing task is concerned, there are three main subjects involved: Linguistic, physiology and acoustics. The first one refers to how language is constructed and the grammar analysis, the second one focuses on the way the sounds are produced and, finally, acoustics is responsible for the generation and transmission of sounds.

3. Project development

This chapter presents the process followed during the development of the project.

3.1. Speech Recognition

The first important step of the project was to find an easy-to-use tool suitable for the Speech Recognition task. After testing several open-source toolkits recommended by the experts on this field, we finally managed to run a PyTorch implementation of end-to-end models for ASR, developed by Awni Hannun.

3.1.1. Dataset

The dataset used to train the speech recognition system was LibriSpeech[4] dataset, which is a corpus of read English speech suitable for training and evaluating speech recognition systems. It contains almost 1000 hours of speech sampled at 16 kHz.

The subsets used to train and to test the system are the following:

Subset	**hours**	**Total speakers**
Train-clean-360	363.3	921
Dev-clean	5.4	40
Test-clean	5.4	40

Table 1. Librispeech data subsets used in the project

3.1.2. Speech. A PyTorch implementation for Speech-to-Text

Speech is an open-source package to build end-to-end models for automatic speech recognition. At the time this project was developed, the supported configurations were: Sequence-to-sequence models with attention, Connectionist Temporal Classification and the RNN Sequence Transducer.

➔ **Data preparation**

Before starting to train a model, the first thing we need to do is to prepare the data we want to train the model with. The preprocessing task consists in two main parts:

- Convert files from flac to wav
- Generate a json file for each of the train, dev and test subsets.
  containing the duration, the transcription and the path of each of the files of the whole training subsets

Once the preprocess is done, we can proceed to the training part.

➔ **Training**

We first need to edit the configuration file, where you have to set the save_path to the choosen location to store the model. You can also modify the parameters of the neural network configuration. In our case, we have used the CTC configuration and we have set the number of epochs to 200.

- Edit Configuration file: ctc_config.json
  - Save_path: path where the model is stored
  - Train_set
  - Dev_set
  - Batch_size: 8
  - Epochs: 200
  - Class: CTC

Now we are ready to train the model. The training process takes a lot of time in executing; with the configuration we have used it took around 2 weeks to finish.

The metric used to evaluate the performance is the Character Error Rate (CER). We get a 25.3% of CER for the development subset.

➔ **Testing**

Once the model is trained, we evaluate its performance using the test subset. We can save the predictions in a json file to see the resulting transcripts from the audio files.

The evaluation of the test subset is 27% of CER.

3.2. Open-NMT

The toolkit used to train a neural machine translation system is Open-NMT: an Open-Source Toolkit for Neural Machine Translation.

➔ **Data preparation**

As a preliminary step, we have to do a pre-processing task in order to extract features for training and generate vocabulary files for the model.

The data consists of parallel source and target data containing one sentence per line.
There are parallel data for both training and validation files (the latter are required and used to evaluate the convergence of the training).
In the source data we find sentences written in the source language (English), and the target data corresponds to the translation of that sentences into the target language (German).
After running the preprocessing, the following files are generated:

- demo.trained.pt
- demo.valid.pt
- demo.vocab.pt

which are serialized PyTorch files containing training, validation and vocabulary data, respectively.

➔ **Training and Translation**

Now we are ready to train the model. We use the default model, that consists of a 2-layer LSTM with 500 hidden units on both decoder/encoder.
Once the model is trained, we can use it to predict on new data. By executing the "translate.py" command and indicating the data we want to translate, a text file is generated with the corresponding translations predicted.
As demo dataset we have used to train the model is too small, the predictions are not so good.

After that, we try a pre-trained model to repeat the process and to find out if the can manage to achieve good predictions using this implementation.
This model has been also trained for English-to-German translation. The dataset used for the training is WMT.

Using the pre-trained model it seems that the translation system has now a good performance. The problem comes when concatenating the ASR and the NMT blocks.

As the speech recognition system is character-based and it has not a high accuracy, there are some mistaken characters in the transcriptions, and that results in many non-existing words. If we then takes that transcripts as the input of the NMT system, it does not recognize the non-existing words and this makes it impossible to achieve a meaningful translation.

3.3. Implementation with Google Cloud

Google Cloud Platform is a suite of Cloud Computing services that provides several cloud services such as machine learning APIs.

In order to achieve the implementation of a block-based speech translation, the useful APIs from Google Cloud are the following:

- Cloud Speech-to-Text API
- Cloud Translation API
- Cloud Text-to-Speech API

Before starting to implement a Speech Translation system using Google Cloud APIs, there are some preliminary steps:

- Create a new project with the GCP (Google Cloud Platform) Console
- Enable the APIs listed above for that project
- Create a service account
- Download a private key as JSON
- Set the environment variable *GOOGLE_APPLICATION_CREDENTIALS* to the file path of the JSON file downloaded on the previous step
- Install the client libraries: *google-cloud-speech*, *google-cloud-translate* and *google-cloud-texttospeech*

Now we are ready to implement the three blocks of our system.

→ **Speech-to-Text**

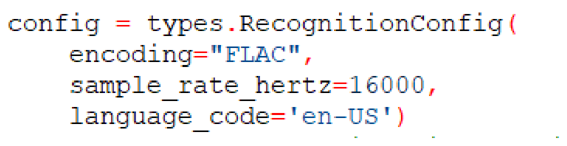
Google Cloud Speech-to-Text allows to apply the most advanced deep-learning neural network algorithms for speech recognition in an easy-to-use API.

To make a speech recognition request, it's necessary to define some configuration settings. The required fields are the following:

- Encoding → specifies the encoding scheme of the supplied audio
- Sample Rate → specifies the sample rate (in Hertz) of the supplied audio

- Language code → the language + region to use for speech recognition

As our audio input files are FLAC files, sampled at 16kHz and the source language is English, we define the configuration as follows:



```
config = types.RecognitionConfig(
    encoding="FLAC",
    sample_rate_hertz=16000,
    language_code='en-US')
```

Figure 6. Piece of code of the configuration.

The expected input of our Speech Recognition implementation is a text file containing the path of each of the audio files (.flac) we want to transcribe.

For each audio input file, we send a speech recognition request to the Speech-to-Text API.

The output is as well a unique text file containing the recognized text from each of the audio files. Each line corresponds to a single audio transcription.

→Translation

The Translation API allows us to translate an arbitrary string into more than 100 languages using state-of-the-art Neural Machine Translation.

To make a Translation API Request, we must to specify 3 parameters: the language to translate from (source), the language to translate to (target) and the text we want to translate. If the source language is not specified, the API will attempt to detect the source language automatically.

As we are building a block-based implementation, the input of our language translator is the output of the Speech Recognition block, and the output is a text file containing the corresponding English-to-German translation.

→Text-to-Speech

With Google Cloud Text-to-Speech we can synthesize natural-sounding speech with 30 voices in 14 languages and variants.

To send a request to the Text-to-Speech API, the parameters to define are the following: the voice configuration (language, voice and gender), the audio encoding format and the text to be synthesized.

Our synthesizer expects the output of the language translation as input.

For each line of the input (corresponding to the English-to-German translation of a single audio file), we send a *synthesize* request to the Cloud Text-to-Speech API.

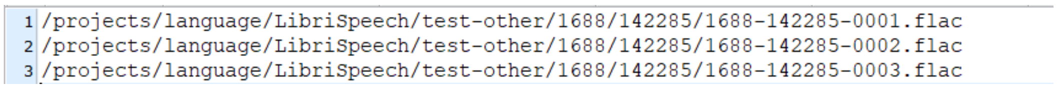
Finally, the speech synthesis process generates an audio file of natural German speech for each one of the transcripts translated into German.

Once we have the three blocks that integrates the Speech-to-Speech Translation system, we chained them in order to automatize the whole process and we finally have the overall system implemented.

To better understand the performance of our system, let us illustrate it through an example.

The example consists in translating three audio files of English speech (from the LibriSpeech test subset) into German speech.

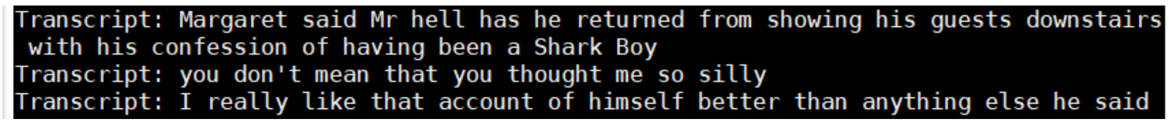
The text file we use as the input of the overall system is the following:



```
/projects/language/LibriSpeech/test-other/1688/142285/1688-142285-0001.flac
/projects/language/LibriSpeech/test-other/1688/142285/1688-142285-0002.flac
/projects/language/LibriSpeech/test-other/1688/142285/1688-142285-0003.flac
```

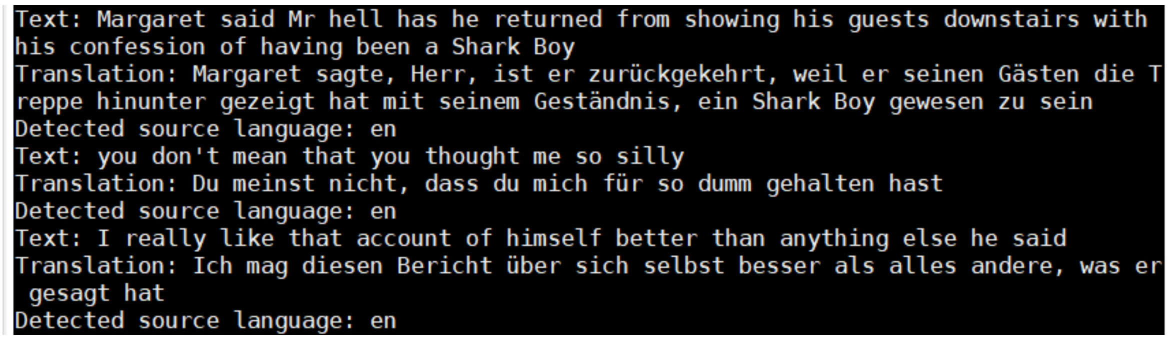
Figure 7. Input file containing the path of the three audio files

During the execution of the entire process, the terminal shows the output of each one of the blocks step by step.



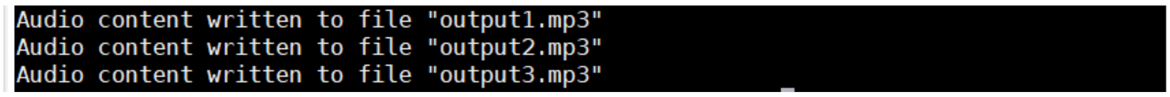
```
Transcript: Margaret said Mr hell has he returned from showing his guests downstairs
 with his confession of having been a Shark Boy
Transcript: you don't mean that you thought me so silly
Transcript: I really like that account of himself better than anything else he said
```

Figure 8. Terminal output corresponding to the Speech-to-Text process



```
Text: Margaret said Mr hell has he returned from showing his guests downstairs with
his confession of having been a Shark Boy
Translation: Margaret sagte, Herr, ist er zurückgekehrt, weil er seinen Gästen die T
reppe hinunter gezeigt hat mit seinem Geständnis, ein Shark Boy gewesen zu sein
Detected source language: en
Text: you don't mean that you thought me so silly
Translation: Du meinst nicht, dass du mich für so dumm gehalten hast
Detected source language: en
Text: I really like that account of himself better than anything else he said
Translation: Ich mag diesen Bericht über sich selbst besser als alles andere, was er
 gesagt hat
Detected source language: en
```

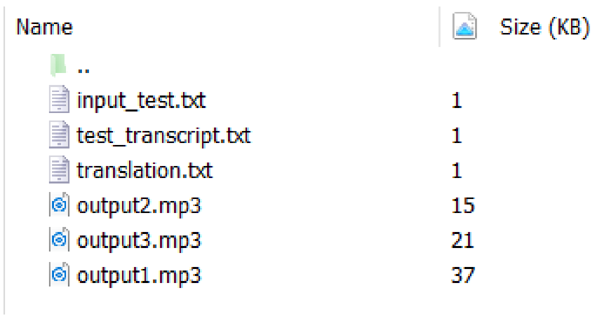
Figure 9**.** Terminal output corresponding to the translation task



```
Audio content written to file "output1.mp3"
Audio content written to file "output2.mp3"
Audio content written to file "output3.mp3"
```

Figure 10. Terminal output corresponding to the Text-to-Speech process

As it can be seen in the figure above, once the execution has finished, three audio files corresponding to the German translated speech are generated. Both the transcripts and the translation output are also saved in a text file.



Name	Size (KB)
..	
input_test.txt	1
test_transcript.txt	1
translation.txt	1
output2.mp3	15
output3.mp3	21
output1.mp3	37

Figure 11. Automatically generated files during the execution + input file

4. Results

In this chapter we present a comparative study of the costs involved between an In-house software development and a Cloud implementation of a Speech-to-Speech Recognition system.

4.1. In-house software and local server

First, we estimate the annual cost of the GPI infrastructure.

The elements we have to consider are the following:

- Virtual machines with 8 GPUs
- Access servers
- Storage
- Data centre
- Technical staff

Another element that would have to be considered is software licenses, but assuming that we would use open source tools to build the S2ST system, it would not imply any cost.

The expenses related to the electricity could have not been included in the cost estimate calculation because it's UPC who bear that costs and we have not had access to this detailed information.

To calculate the annual cost of the GPI infrastructure, the method we apply consists in dividing the total cost of each element by its useful life.

	Unit cost	Amount	Total cost	Useful Life	Annual cost
Virtual machine with 8 GPUs	20.000 €	3	60.000 €	8 years	7.500 €
Access server	12.000 €	2	24.000 €	10 years	2.400 €
Storage (30 TB)	10.000	-	10.000 €	12 years	833 €
Data centre **- Racks** **- Switches+routers**		 2 .	 1.000 € 1.500 €	 -	 2.000 € 1.500 €
Technical staff salary	35.000 €	1	35.000 €	-	35.000€

Total	**47.433 €**

Table 2. Annual cost of the GPI Infrastucture

Now that we have the total annual cost of the GPI infrastructure, let's divide it between the number of members of the Image Processing Group taking some considerations into account.

The number of professors and Phd candidates amounts to 11 and 10, respectively.

Considering that each professor has 1 student of Master and 2 Degree students assigned per year, it would be a total of 11 Master students and 22 Degree students per year.

As far as computational cost is concerned, we estimate that Phd candidates and professors use twice as much computational resources than a Master student do, as well as a Master student uses twice as much computational resources than a Degree student.

According to that estimation, we will assign different weights to the members of the group depending on their category. These weights will be 4 for Phd candidates and professors, 2 for Master students and, finally, a weight of 1 for the Degree students. Applying the corresponding weights to each member, we can calculate total number of members under the assumption that all members were Degree students.

Member category	**Real Quantity**	**Weight**	**#Equivalent Degree students**
Professors	11	4	44
Phd candidates	10	4	40
Master students	11	2	22
Degree students	22	1	22
Total			**128**

Table 3. Calculation of the equivalent number of Degree students

Based on this result, we can estimate the corresponding expense of a single Degree student that makes use of the GPI Infrastructure.

Annual cost	47.433 €
Number of "Degree students"	128
Total	**370,57 €**

Table 4. Cost estimate of the GPI infrastructure corresponding to a Degree student

4.2. Google Cloud

As the main purpose of this project is to generate a parallel speech corpus, the study will be based on the costs involved in translating the whole LibriSpeech dataset from English to German.

Cloud Speech-to-Text is priced based on the amount of audio successfully processed by the service. The price is 0,0052€ / 15 seconds, which means 1,25€ / 1 hour.

Subset	**hours**	**Cost**
Dev-clean	5,4	6,75 €
Dev-other	5,4	6,75€
Test-clean	5,3	6,625 €
Test-other	5,1	6,375 €
Train-clean-100	100,6	125,75 €
Train-clean-360	363,6	454,5 €
Train-other-500	496,7	620,875 €
Total	982,1	**1.127,625 €**

Table 5. Cost estimate of Speech-to-Text using Google Cloud

The Cloud Translation API has a price of 17,5€ / 1 million characters.

To calculate the cost estimate of translating the whole LibriSpeech dataset, we have to estimate the character count of the dataset transcripts. To do a more detailed calculation, a character count is carried out for each one of the subsets of LibriSpeech, and then, the Cloud Translation API's price is applied to calculate the total cost.

Subset	**Total characters**	**Cost**
Dev-clean	286.844	5,02 €
Dev-other	264.090	4,62 €
Test-clean	278.974	4,88 €
Test-other	271.063	4,74 €
Train-clean-100	5.427.481	91,83 €
Train-clean-360	19.053.548	333,44 €
Train-other-500	25.282.513	442,44 €
Total	50.684.513	**886,98 €**

Table 6**.** Cost estimate of translation using Google Cloud

The Cloud Text-to-Speech API is priced based on the amount of characters to synthesize into audio. The price is 3,5€ / 1 million characters.

In order to estimate the character count of the German translation of the whole LibriSpeech, we calculate the character count ratio English:German by translating a small set of the dataset from English to German. This small set has a total of 6313 characters, while its translation to German contains 7008 characters. Therefore, the estimated character count ratio is 1:1,11 (English:German). Based on this value, we can now calculate the cost estimate of speech synthesis.

Subset	**Char count (English)**	**Char count (German)**	**Cost**
Dev-clean	286.844	318.396	1,11 €
Dev-other	264.090	293.140	1,03 €
Test-clean	278.974	309.661	1,08 €
Test-other	271.063	300.880	1,05 €
Train-clean-100	5.427.481	6.024.504	21,09 €
Train-clean-360	19.053.548	21.149.438	74,02 €
Train-other-500	25.282.513	28.063.590	98,22 €
	Total	56.459.609	197,61 €

Table 7. Cost estimate of the speech synthesis using Google Cloud

Total cost of the overall system	
Speech-to-Text	1.127,625 €
Translate	886,98 €
Text-to-Speech	**197,61 €**
Total	**2.212,22 €**

Table 8. Total cost estimate of the overall system

5. Budget

As the object of this thesis was not developing or building any prototype, to calculate the cost estimate of the project we will take into account the following elements: the hardware resources required, the software used and the engineers salary.

➔ Hardware resources

This project has been developed using the resources available at the Image Processing Group (GPI) at UPC, so the hardware needed has not implied any real cost.
Nevertheless, we provide a cost estimation of the required hardware assuming that those resources had not been provided by the research group.

The total amount of hours of CPU and GPU used during the project development is 804 and 760, respectively.

The price of a GPU provided by Google Cloud Engine with the most similar configuration to the one used to develop the project is 0.36€/hour, so the total expense would be 264,25 €.

➔ Software used

Both the Automatic Speech Recognition toolkit used to train a model and the OpenNMT toolkit are open source, so they don't imply any cost.

As far as the Google Cloud APIs usage is concerned, the prices of the APIs we have used are the following:

- Speech-To-Text: 1,25€ / h
- Translate: 17,5 € / 1 million character
- Text-To-Speech: 3,5 €/ 1 million characters

However, Google Cloud offers $300 free credit during the first 12 months to get started with any GCP product. As this free credit has been enough for the project development, the usage of Google Cloud APIs has not implied any cost either.

➔ Engineers salary

Finally, we have to calculate the salary of the engineers working on the project according to the total amount of hours spent in the project.
Considering a Junior Engineer salary of 10€ per hour and a dedication of 20h/week in average during 30 weeks, the junior engineer salary amounts to 6.000€.
The professor and the Phd student who helped me with the development of the project are considered as Senior Engineers, with a salary of 50 €/h.

	Cost/hour	**Dedication**	**Total**
Hardware resources	0.35 €/h	760 h	266 €
Senior Engineers	2 x 50 €/h	2 h/week	6.000 €
Junior Engineer	10 €/h	20 h/week	6.000 €
Total			12.266 €

Table 9. Budget of the project

6. Conclusions

The main goal of this project was the implementation of a Speech-to-Speech translation system so it can be used in the *DeepLipDub* project to generate a parallel speech corpus.

We have experimented with some open source toolkits and we have indeed managed to train a model for both Speech Recognition and Translation tasks. However, we have not achieved successful results of that models and as it is required such long computational time to repeat the training experimenting with different configurations, we decided to move to a Cloud implementation.

The implementation of the Speech-to-Speech Translation system using the Google Cloud APIs has a really good performance as it uses the state-of-the-art technologies. It would be very useful to generate a parallel speech corpus and despite the fact that the usage of Google Cloud APIs is not free, it has an affordable cost.

As we have tried both In-house and Cloud-based development implementation, it would be interesting to make a brief comparison between them. Based on the degree of easy-usability, the quality of performance and the required time to be spent, we strongly recommend the Cloud-based development.

It is clear that for the purpose of building a parallel corpus, the Cloud-based choice is the most adequate. However, the *DeepLibDup* project integrates an ASR and NMT modules. Therefore, as a future work, it would be interesting to take up the In-house software solution that we have not managed to implement with success. Now that we have more control over the tools, it would be easier to train the models and to improve their accuracy.

Bibliography

[1] Amanda Cardoso Duarte, Xavier Giró-i-Nieto, Marta R. Costa-Jussà, Antonio Bonafonte, and Jordi Torres. "DeepLipDub: Putting a Mouth to Someone's Words". Universitat Politecnica de Catalunya (UPC) and Barcelona Supercomputing Center (BSC) ` Barcelona, Catalonia/Spain, 2018.

[2] Alex Graves, Abdel-rahman Mohamed and Geoffrey Hinton. "Speech Recognition with deep Recurrent Neural Networks". Department of Computer Science, University of Toronto

[3] Baidu Research – Silicon Valley AI Lab∗ Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, Zhenyao Zhu. "Deep Speech 2: End-to-End Speech Recognition in English and Mandarin".

[4] Vassil Panayotov, Guoguo Chen, Daniel Povey, Sanjeev Khudanpur. "LibriSpeech: An ASR corpus based on public domain audio books". The Johns Hopkins University, Baltimore, MD 21218, USA

[5] Guillaume Klein†, Yoon Kim, Yuntian Deng, Jean Senellart, Alexander M. Rush. "OpenNMT: Open-Source Toolkit for Neural Machine Translation".

[6] Alex Graves, Santiago Fernández, Faustino Gomez, Jürgen Schmidhuber. "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks".

[7]

Appendices

Documentation of the Google Cloud APIs:

https://cloud.google.com/speech-to-text/docs/

https://cloud.google.com/translate/docs/

https://cloud.google.com/text-to-speech/docs/

Source code:

Speech. Open source for ASR:

https://github.com/awni/speech

OpenNMT

https://github.com/OpenNMT/OpenNMT-py

Speech-to-Text:

https://github.com/GoogleCloudPlatform/python-docs-samples/tree/master/speech/cloud-client

Translate:

https://github.com/GoogleCloudPlatform/python-docs-samples/tree/master/translate/cloud-client

Text-to-Speech

https://github.com/GoogleCloudPlatform/python-docs-samples/tree/master/texttospeech/cloud-client