Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 81 (2016) 53 – 60

Procedia Computer Science

5th Workshop on Spoken Language Technology for Under-resourced Languages, SLTU 2016, 9-12 May 2016, Yogyakarta, Indonesia

Eyra - Speech Data Acquisition System for Many Languages

Matthias Petursson, Simon Klüpfel, Jon Gudnason*

Reykjavik University, Menntavegur 1, 101 Reykjavik, Iceland

Abstract

Speech data acquisition is particularly important for under-resourced languages. The data gathering is the most labour-intensive part of developing speech technologies such as automatic speech recognizers and synthesizers. It is therefore important to facilitate this process with as much automation and labour-cutting tools as possible. This paper describes a new open-source system called *Eyra* which enables distributed speech data collecting through a variety of devices. It addresses internet connectivity issues by allowing the data collectors to run the back-end server off a local laptop, thereby facilitating automatic quality control and less labour-intensive data uploading and compiling. It can also be used in a crowd-sourcing set-up where volunteers can donate voice samples through a desktop web-browser interface. An initial test shows that the system works well in an offline mode using smart-phones for data collection.

Peer-review under responsibility of the Organizing Committee of SLTU 2016

Keywords: automatic speech recognition; speech resource collection; under-resourced languages; internationalization

1. INTRODUCTION

Speech data acquisition is a crucial first step in developing an automatic speech recognizer (ASR) for a new language. Recent advances in ASR has seen the production line become more robust and accessible to a wide range of developers. A cottage industry of ASR development has been made possible with more automation in training acoustic and language models, wider availability of high quality open source software for ASR and new hardware implementation using graphical processing units. Normally, the biggest hurdle to developing an ASR system for a new language is the preparation and gathering of language resources. Building a speech corpus can be the most labour intensive part of that effort[1].

Improving the efficiency of building ASR systems is very important for under-resourced languages[2,3,4]. The number of speech data hours needed to build a robust ASR system was assessed in[5]. It was predicted that with the technology at the time, the amount of data needed to get word error rates of 10% was a prohibitive 10 thousand hours of data. ASR technology has however been revolutionized in the past few years[6] and we have seen word error rate reduce for a relatively small number of hours of data, e.g. 300 hours give 10.7% word error rate in[7]. This shows

* Corresponding author. Tel.: +354-5996325.
E-mail address: jg@ru.is

Peer-review under responsibility of the Organizing Committee of SLTU 2016
doi:10.1016/j.procs.2016.04.029

that high quality ASR performance for an under-resourced language is achievable with a realistically challenging data gathering effort.

This paper presents an open source system *Eyra* for acquiring speech data[1]. Many speech recognition projects rely on speech corpora that already exist but there are many advantages to bypass this dependence. Recording a database means for example that control is maintained over audio quality, microphone characteristics, background noise and vocabulary diversity. Also, pre-existing corpora might not be free to use or they might not even exist at all for an under-resourced language.

1.1. Relation to other work

Speech data acquisition systems, similar to the one described here, have already been proposed. Developers at Google described a system called *Datahound* which uses an Android-based smartphone app to collect the data and uploads them automatically to a tethered database on the Google App Engine[8]. The system allows the data collectors to download their prompt lists onto the client and simultaneously record and upload speech and metadata to the centralized Google App Engine server using however many devices as needed. The Icelandic speech corpus *Malromur* was, for example, collected using this system[9].

Woefzela was proposed as an open source alternative to *Datahound*[10,11]. The Android-based smartphone app addressed Datahound's reliance on internet connectivity by changing the data gathering architecture to using external memory cards for the collected speech data. This has made offline collection of speech data possible which is crucial in the environment posed by many under-resourced languages[2], but it increased the work needed to upload and organize the acquired data by field workers. Included in the process was a semi-automatic quality control process that increased the portion of recordings that are useful for speech recognition training. Early detection of low volume level and start/stop errors makes it possible for the field worker to intervene and correct any flaws in the recording process (e.g. thumb covering microphone).

Other speech data collecting projects include the volunteer-based effort VoxForge[2] and The LibriVox Project[3], where open source books are read by participants. Data from the LibriVox project is particularly useful for speech synthesis since it contains a lot of data for each speaker, but it has also been used to build a speech corpus for ASR[12]. While both projects encourage contributors to record and upload the speech manually, VoxForge also provides a recording interface in the form of a Java browser plugin. However, after recent announcement by Oracle to discontinue development of the browser plugin, the manual process remains the only choice for contributors.

1.2. Design considerations

The conception and design of *Eyra* is based on the considerations and experiences of *Woefzela* and *Datahound*. The fidelity and flexibility of *Datahound* with respect to data handling and user experience was combined with *Woefzela*'s robustness to connectivity and data quality. The following design criteria were adhered to:

1. The end user (contributor) shall only have to focus on making recordings.
2. The user interface (UI) shall be accessible on a wide range of recording-capable devices, such as smartphones and tablets but also personal laptop and desktop computers, as well as professional studio hardware.
3. The UI shall be functional on the devices without the need to install software or alter the installed system.
4. Setting up recording sessions without internet connectivity should be easily achievable.
5. The tokens to be read (usually lines of text) are provided to the end user by the system through the UI.
6. Automatic Quality Control (QC) of the recordings shall help during unsupervised recording sessions to avoid excessive gathering of unusable data.

The first criteria is achieved by using a similar user-interface design to that of *Datahound*. The metadata can easily be filled out either by the contributor or the field worker and the participation-agreement can be read and agreed to

[1] Link to the software is provided at http://www.eyra.is
[2] http://www.voxforge.org
[3] https://librivox.org

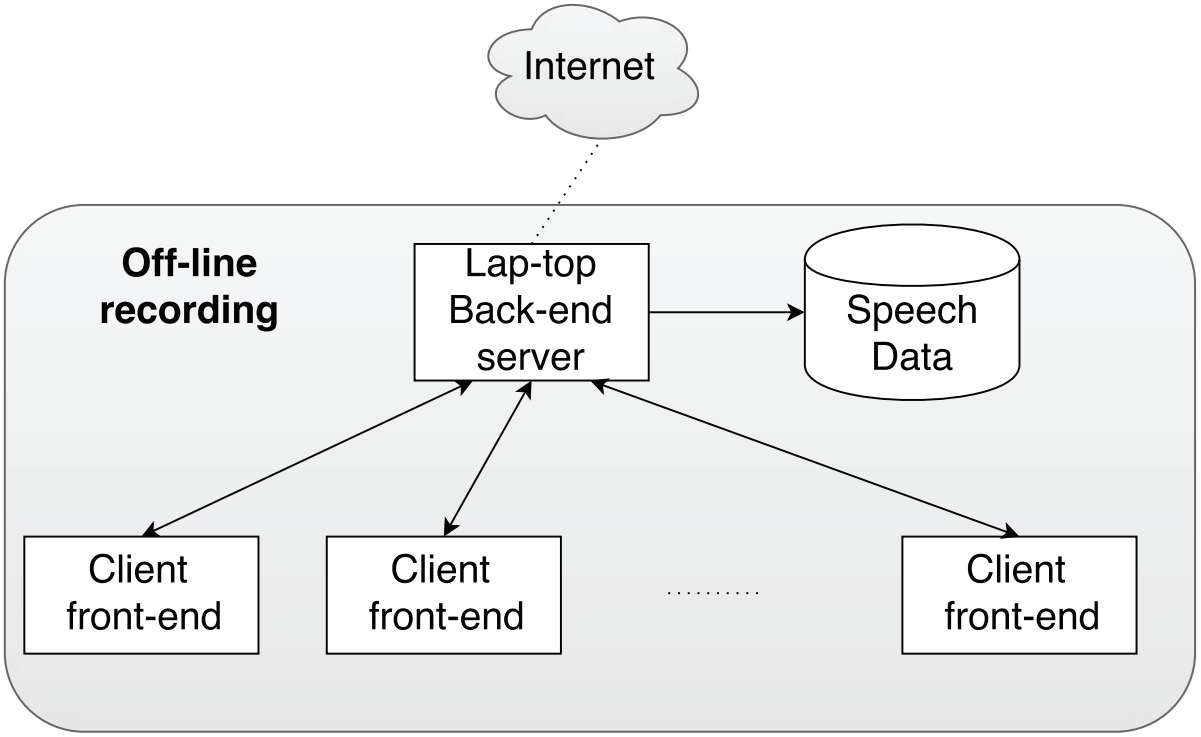

Fig. 1. A speech data acquisition session run without a connection to the internet using a laptop back-end and a wireless local area network for the client connections.

when the contributor is entered into the system. Otherwise, most of the time and effort spent by the contributor is to go through the data recording process. *Eyra* is designed using a web-client front-end to address criteria two and three. Device compatibility is therefore not an issue so long as it can run an HTML5 compatible web-browser with audio recording support. This design is described in more detail in Sec. 2.

When the client devices can have internet connectivity, the web app and backend system may typically be hosted on a centralized web-server. In cases where internet connectivity is either not available or not affordable, the clients can be served by a local server, e.g., a laptop operated by the field worker. This server runs the same software but furthermore serves as a wireless access point for the client devices. Such a set-up is depicted in Fig. 1 and addresses the fourth design criteria. The laptop runs a local version of the speech database that can be synchronized with a global database when internet connection is available. The prompts are added to the database by a script and are then handled by the back-end server and downloaded to the client front-end automatically thereby addressing criteria five. The back-end server architecture is described in Sec. 3.

The sixth criteria is addressed by running quality checks on the back-end server. This process is described in Sec. 4. The data acquisition process is outlined in Sec. 5 and results from preliminary system experiments presented.

2. CLIENT ARCHITECTURE

The client is the only component of the system that is directly exposed to the end users. To maximize the potential user base, the requirements on the client's hard- and software were kept as low as possible. The HTML5 standard offers the capabilities needed for the recording tasks and is by now sufficiently supported in modern browsers for broad deployment.

2.1. User Interface

The user interface (UI) is built solely using HTML5, JavaScript, and Cascaded Style Sheets (CSS) and is loaded on the recording client by accessing it in a web browser.

The UI is designed to be simple and user-friendly. Unless administrator actions are required (see below), a typical use case would be as follows: The contributor is entering a freely chosen username, provides information specific to the current recording session (e.g., presence of environmental noise), and accepts a participant agreement. Only if the username had not been used previously, the speaker's information, such as sex or date of birth are asked, before the main window is loaded. This main window consists of an area to display the token, and one button each to start and stop the recording.

Settings and administrator options are not exposed to the end user, but can be accessed through a menu after authenticating with the server using admin credentials. Administrative tasks include adding an instructor (e.g., name, phone, email), registering the device (set a device ID and send it to the server/save in local storage),[4] and synchronizing locally stored sessions (metadata and recordings) with the server.

Using the Mozilla local forage library (tokens, recordings and metadata), along with the HTML5 application cache (the website itself), the app is entirely self-contained. By this, once it has been initialized, it requires no further connection to the server until it is dumping the gathered data. An example setup using this feature could be as follows: In the field the laptop is installed and the app is initialized on several handheld devices. The devices are then used to gather data, moving around to reach more contributors, and moving outside of the laptop's wireless range. When the devices are returned they reconnect to the laptop wirelessly and synchronize their data.

2.2. Token processing

The client receives a list of tokens from the server and stores it locally for the current session. Any natural language processing involved in the token selection happens on the server while the client only queues and records the provided tokens.

For the majority of applications, tokens will be in the form of text. Peculiarities of particular texts, such as directionality, fonts, etc., are expected to be covered by the use of Unicode as the token format and using HTML for the presentation.

The design of the client does, however, not restrict tokens to be plain text. A general token is more abstract and can consist of any data that can be presented to the contributor and be expressed in spoken form. Examples of non-textual tokens could be images or colours, providing use-cases for recording speech responses when written text tokens are not appropriate (e.g. ill-defined written form of the language and illiterate contributors).

2.3. Audio Processing

Audio is recorded through the Web Audio API using `navigator.getUserMedia`. Each recording is made into a JavaScript blob on the client and if it passes *Instant Quality Control* (see below) an attempt is made at sending the recording with its associated metadata to the server. On a failed send, the data is saved locally. The data from these locally stored sessions can then be sent to the server once connectivity has been restored.

3. SERVER ARCHITECTURE

3.1. Application Programming Interface

The communication from client to server is governed by a RESTful API through HTTP requests. Only a specific set of operations are allowed. Currently, these operations are:

- *POST instructor data, receive in return ID of instructor.*

[4] This is needed because the browser alone can not access any unambiguous information to identify the device.

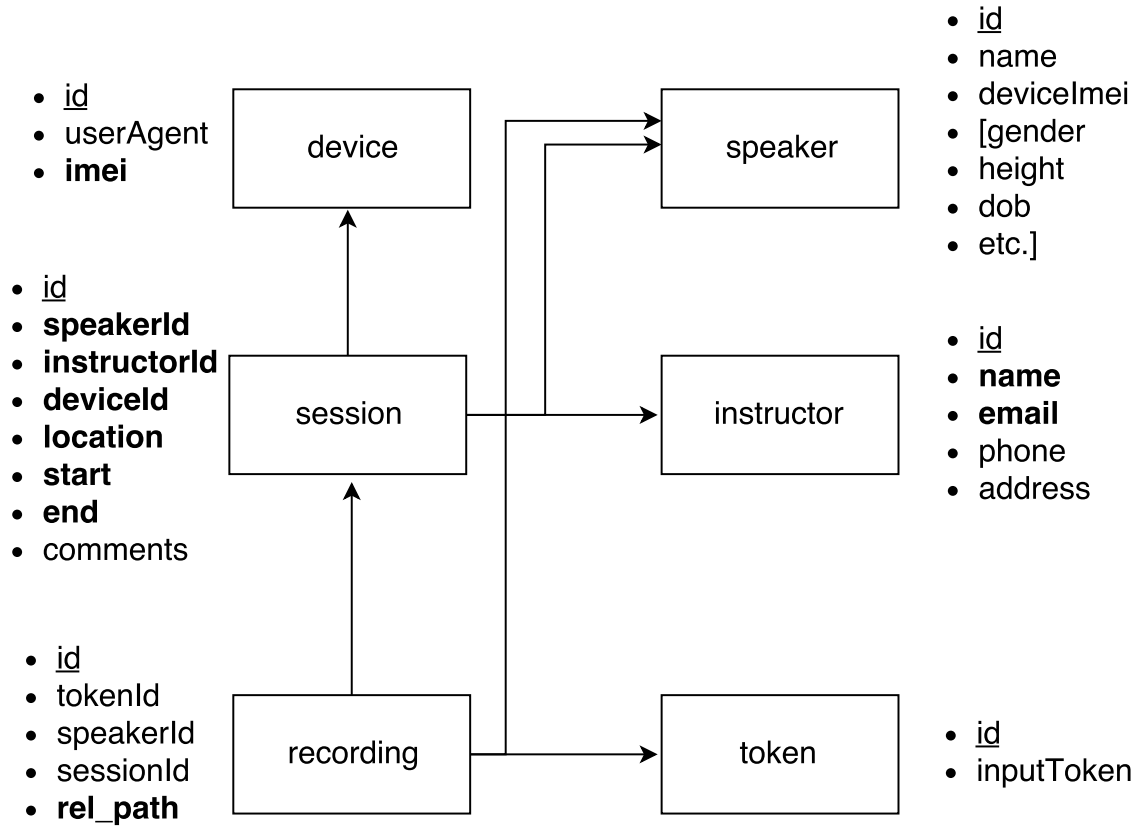

Fig. 2. Relational model for the database schema.

- *POST device data.*
- *GET a list of tokens.*
- *POST a recording including its metadata.*
- *GET a report from quality control.*

All the metadata is transmitted as simple JSON strings, may it be data from the client (e.g. speaker or utterance information), or from the server (e.g. the list of tokens). The server parses the content of the JSON objects and either stores received information in a relational database, or otherwise reacts on the received request if need be.

3.2. Database design

After a successful recording, or during a later sync, the client pushes the data and metadata to the server. Here, the recordings are stored in the file system, and the remaining data is organized in a MySQL database. The database schema can be seen in Fig. 2, where primary keys are underlined, the bold keys should form a unique set for each entry, and arrows indicate foreign keys. For example, *recording* has an arrow to *session*, *speaker*, and *token*, meaning that in order to add an item to *recording*, items from these three tables need to be referenced in it.

The tables in the database are structured as follows:

- **recording** (References: *token*, *speaker*, *session*)
  Signifies a recording of a token. Contains a relative path to the audio file and information on the recording session and speaker.
- **session** (References: *speaker*, *instructor*, *device*)
  Signifies a session of recordings by a speaker. Has a start and an end time, location (e.g. GPS) and some comments (e.g. environmental noise).
- **device** Signifies the recording device used. Has a *userAgent* string from the browser (includes whether it's a mobile, which browser, etc.), and an optional International Mobile Station Equipment Identity (IMEI), or any other type of device identity.
- **speaker** Signifies a speaker (who records tokens). Has a user name, an optional reference to a device's ID, and other features depending on the nature of the data gathering. Examples are sex, height and date of birth.
- **instructor** Signifies an instructor (field worker), who is responsible for the speakers that use the device he is registered on. Has a name and an email, and optionally a phone or an address. This is mainly to make it possible to identify and contact the instructor.
- **token** Signifies a token. The *inputToken* can be data in form of Unicode text, or any other data to be presented on the client.

3.3. Validation processes

Recordings received by the server are being queued for validation exceeding the one done directly on the client. The details of this quality control are described in more detail in the next section.

4. QUALITY CONTROL

Quality Control (QC) of the collected recordings plays an important role in the creation of a high quality speech corpus. It is necessary to verify the final set of recordings for correctness and compliance with desired quality requirements. While transcription errors may be corrected to yield still usable data, other errors or problems will inevitably result in unusable data. To minimize the rejects, it is desirable to monitor and interfere with the recording process in or close to real time. By this, the client can inform the contributor of problems affecting the recordings and adjustments can be made to improve the quality.

4.1. Instant QC

Due to the low requirements set for the client devices, the QC tasks that can be accomplished on those instantly during or after the recording are limited. Nevertheless, relatively simple tasks can still be implemented, such as to identify an insufficient recording volume, too high background noise, utterances being cut off at the beginning or end, or samples being clipped. In these cases the subject can be informed of the problem, and the recordings being discarded instantly on the client device.

4.2. Online QC

Other analyses may require more processing power and are thus better suited to be handled by the server: After receiving a recording from the client, the data is processed, graded, and if need be, a report is prepared for retrieval by the client. The Online QC tasks may vary widely, so that it can not be known *a priori* how long the particular analysis will actually take. Hence, it does not appear sensible to pause the recording session until the current recording passed Online QC. By querying the server for new QC reports in regular intervals, e.g., after each transmitted recording, the client still can follow the QC in a timely fashion and react upon it with neither risking to produce excessive amounts of rejects, nor causing the contributor to be idle.

For target languages with pre-existing ASR resources, the recorded utterances can, e.g., be compared to the token by means of online decoding or forced alignment, and recordings containing pronunciation or reading mistakes can be flagged as such. If such resources are not available, the provided recordings can be used to bootstrap an ASR

system for this purpose. This can be repeated with the increasing amount of data, by which the ASR system will incrementally increase in quality throughout the process.

Depending on the amount of data received and the workload needed for processing, the server may distribute the Online QC tasks to other networked computers (if connectivity is granted), while himself merely taking over the book-keeping and report generation based on their QC results.

4.3. Offline QC

The framework employed in the Online QC strategy can readily be used for quality control of the final corpus or subsets thereof. Here, e.g., a completed recording session by one speaker can be thoroughly tested for its quality, but also for other quantitative or statistical features. The reports generated here can then guide the further processing and finalization of the speech corpus. Furthermore, more detailed insights might be gained regarding possible improvements of further data gathering processes.

5. DATA COLLECTION

The *Eyra* system can be easily used in different types of data gathering efforts, such as, a) personal, b) in-house, c) in-field, or d) crowd-sourced projects.

The design only requires the client to be able to communicate (occasionally) with the server. By this, one can easily implement the several different setups above: For the personal data gathering effort, a), the server and client are run on the same personal computer, using *Eyra* merely to streamline the recording process, as no more than one client is intended to contribute. If more clients are contributing to a private project, the in-house solution, b), is easily implemented with the server hosted in the same LAN the clients are. For work in the field, c), a laptop can be deployed as the server acting furthermore as a wireless access point for the connected recording devices. Being run on a public server, also crowd-sourced projects, d), are easily implemented, lifting the burden of token selection, manual recording and uploading from the shoulders of the contributors.

The system could therefore fit in well into existing volunteer based projects, reinforcing them and extending the base of possible contributors, while not excluding small or big closed projects to make use of it as well.

The focus of the initial implementation has, however, been on decentralized, in-field data gathering using smartphones as the clients. This setup was beta-tested in-house by running a continuation of the *Almannaromur* project described in [9]. The tokens were selected as only containing the news stories and the rare tri-phone sentences. A standalone server was set up on a Dell Latitude E5450 laptop using an Intel 7265 Wireless Card as the access point device. Five Samsung Galaxy (S5, A5 and J5) smart-phones running Android 5.X were used as the speech data collection clients, each using Chrome 43 as the browser for the user interface. The recorded speech was sampled at 48 kHz in mono and no compression was used to store the samples. Five contributors were each asked to donate between 300-500 utterances and the recordings happened simultaneously. In about half an hour 1,544 recordings were collected successfully. Moving out of beta stage further field tests are planned to test the physical limitations of this setup of the *Eyra* system.

ACKNOWLEDGEMENTS

This project was made possible through the support of Google. The authors would also like to thank Oddur Kjartansson and Martin Jansche of Google Research and Sveinn Tryggvason at Taeknivorur ehf. for their valuable advice and contribution during this work.

References

1. N. T. Kleynhans and E. Barnard, "Efficient data selection for ASR," *Language Resources and Evaluation*, vol. 49, no. 2, pp. 327–353, 2015.
2. L. Besacier, E. Barnard, A. Karpov, and T. Schultz, "Automatic speech recognition for under-resourced languages: A survey," *Speech Communication*, vol. 56, pp. 85–100, 2014.
3. R. Sahraeian, D. Van Compernolle, and F. de Wet, "Under-resourced speech recognition based on the speech manifold," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.
4. P. Smit, J. Leinonen, K. Jokinen, M. Kurimo *et al.*, "Automatic speech recognition for northern Sámi with comparison to other Uralic languages," in *Proceedings of the Second International Workshop on Computational Linguistics for Uralic Languages*. The Research Group on Artificial Intelligence (RGAI), 2016.
5. R. K. Moore, "A comparison of the data requirements of automatic speech recognition systems and human listeners." in *INTERSPEECH*, 2003.
6. G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.
7. T. N. Sainath, B. Kingsbury, G. Saon, H. Soltau, A. Mohamed, G. Dahl, and B. Ramabhadran, "Deep convolutional neural networks for large-scale speech tasks," *Neural Networks*, vol. 64, pp. 39–48, 2015.
8. T. Hughes, K. Nakajima, L. Ha, A. Vasu, P. J. Moreno, and M. LeBeau, "Building transcribed speech corpora quickly and cheaply for many languages," in *INTERSPEECH*, 2010, pp. 1914–1917.
9. J. Gudnason, O. Kjartansson, J. Johannsson, E. Carstensdottir, H. H. Vilhjalmsson, H. Loftsson, S. Helgadottir, K. Johannsdottir, and E. Rognvaldsson, "Almannaromur: an open Icelandic speech corpus." in *SLTU*, 2012, pp. 80–83.
10. N. J. De Vries, J. Badenhorst, M. H. Davel, E. Barnard, and A. De Waal, "Woefzela - an open-source platform for ASR data collection in the developing world," in *INTERSPEECH*, 2011.
11. N. J. De Vries, M. H. Davel, J. Badenhorst, W. D. Basson, F. De Wet, E. Barnard, and A. De Waal, "A smartphone-based ASR data collection tool for under-resourced languages," *Speech communication*, vol. 56, pp. 119–131, 2014.
12. V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an ASR corpus based on public domain audio books," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE*, 2015.