Flexible Stand-Alone Keyword Recognition Application Using Dynamic Time Warping

Miquel Ferrarons[1,2]*, Xavier Anguera[1] and Jordi Luque[1]

[1] Telefonica Research, Edificio Telefonica-Diagonal 00, 08019, Barcelona, Spain,
[2] Universitat Autonoma de Barcelona, Barcelona, Spain,
{xanguera, jls}@tid.es

Abstract. We introduce a Query-by-Example (QbE) application for smart-phone devices that implements a recently proposed memory-efficient dynamic programming algorithm [1] for the task of keyword search. The application compares acoustic keywords with the audio input from the microphone and reacts to detected keywords with actions in the phone. These keywords are recorded by the user, who also defines what actions will be performed by each one. One of these keywords is defined to be a trigger keyword, which is used to *wake up* the system and thus reduce false detections. All keywords can be freely chosen by the user. In Monitor mode, the application stays listening to audio acquired through the microphone and reacts when the trigger + some keyword are matched. All processing is done locally on the phone, which is able to react in real-time to incoming keywords. In this paper we describe the application, review the matching algorithm we used and show experimentally that it successfully reacts to voice commands in a variety of acoustic conditions.

Keywords: Mobile search, dynamic time warping, query-by-example, keyword recognition

1 Introduction

Currently smartphones are found everywhere. The usage of smartphones is not only driven by the need to make phone calls, as more and more people use them for activities like gaming, browsing the internet, reading, etc. Voice recognition is quickly gaining popularity and acceptance among smartphone users, probably due to the small keyboard footprint of these devices which makes it more complicated to type on it than to speak to it. Companies like Google [3] of Microsoft [4] have recently proposed powerful speech-enabled applications in the cellphone that are changing the general public miss-conception that speech recognition *does not work*. Driven by this trend, we decided to experiment with small-footprint but flexible voice-enabled command-and-control applications for the cellphone. In this paper we present one of these experiments. It corresponds to a keyword recognition application we developed for Android devices which implements a flexible Query-by-Example (QbE) algorithm to enable many functionalities in the phone by just giving spoken orders to it. QbE algorithms [5]

* M. Ferrarons was visiting Telefonica Research at the time this work was performed.

are used to search for matches of a given spoken query within a set of spoken utterances. In this case we use it to match pre-recorded acoustic keywords against online audio captured from the smartphone's microphone. The proposed application is a proof of concept of a totally offline (no server connectivity required) speech enabled tool to control the telephone's functionalities by voice. In addition, recorded keywords can be chosen by the user to be whatever word/sound he/she wants, and each one can be recorded multiple times to improve matching accuracy. There are two kinds of keywords, "trigger" and "action" keywords. A "trigger" keyword is recorded by the user to "wake up" the application (i.e. indicate that an "action" keyword will be spoken next). Then, "action" keywords are detected and associated to actions in the phone. In this proof of concept some actions have been implemented, like taking a picture, recording a voice note or picking up a call.

The rest of the paper is organized as follows: Section 2 describes the application implementation details, including the user interface (Subsection 2.1) and each of the two modes of operation (Subsections 2.2 and 2.3). Then, Section 3 describes and performs an evaluation of usage of the application among real users, to show that it does perform as expected. Finally, in Section 4 we draw some conclusions from the presented work, and draw some lines of future research.

2 System description

The proposed application has two main modes of operation: Monitor mode, and Recording mode. In Recording mode the user records new acoustic keywords, assigns actions to them, and can set some parameters to each of these keywords. In Monitor mode the application constantly records the ambient sound and reacts accordingly when the user says any of the keywords recorded in the Recording mode. Next we describe the application user interface and how each of these modes works in more detail.

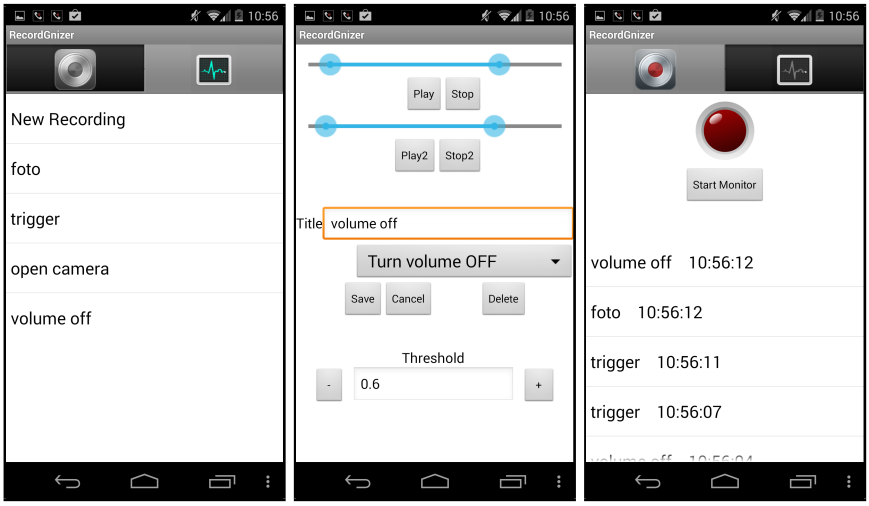
2.1 Application User Interface

We implemented the keyword recognition application in Android. Our development was tested using a Samsung GalaxyS phone, although any smartphone with similar (or superior) capabilities should be able to successfully run the application. Figure 1 shows three screenshots of the application as it is being used. Subfigure 1a shows the main screen for the Recording mode and Subfigure 1c shows the Monitor mode. The user can switch between the two modes by clicking the tab present on top of either of these screens.

Within the Recording mode the user sees the list of keywords that he has already recorded, and has the option (by clicking in "New Recording") to record a new keyword. There is no limit to the number of different keywords that a user can record, although at some point the application might not be able to monitor all of them in parallel (we have experimented with more than 10 concurrent keywords with no felt slowdown on the phone). Clicking an already recorded

keyword or in "New Recording" launches a screen similar to Subfigure 1b. This user interface allows the user to record up to two instances of the keyword, define a name with which this keyword will be later identified (or change a current name), select an action to be taken by the phone when this keyword is selected and modify the detection threshold. Although by default every keyword has a precomputed detection threshold, it can be modified by the user whenever it is not producing successful results. Similarly, after the Speech Activity Detection (SAD) module has automatically identified the start-end points of the speech part in the keyword, these boundaries can be played back and modified through this user interface.

Within the Monitor mode the user initially sees a ON/OFF button to toggle the monitoring status. Once ON, the system continuously scans the audio entering the microphone to detect the "trigger" keyword and then, once detected, either of the "action" keywords. All detections are shown on screen as they appear, and actions are then taken on the phone.



(a) Recording mode (b) Keywords editor (c) Monitor mode

Fig. 1. Application user interface

2.2 The Recording Mode

In "Recording mode" the user is able to record up to two speech instances per keyword. These are processed and stored in the system to later compare them with the input audio. The full processing pipeline is shown in Fig. 3 and is described next.

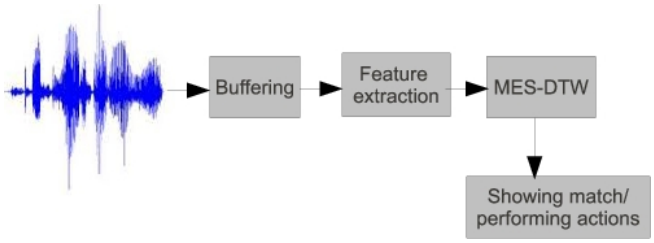


Fig. 2. Online keyword matching in Monitor mode

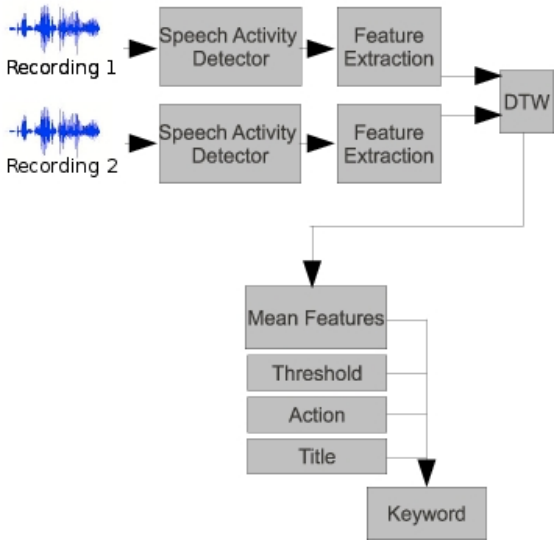


Fig. 3. Keyword acquisition in Recording mode

Speech Activity Detection A speech activity detector (SAD) is applied to detect start and end points of the spoken keywords. We implemented a simple SAD algorithm described in [6] which focuses on energy and zero crossing rate to determine the endpoints of the speech utterance. The algorithm first automatically determines some parameters using the information provided by the first 100ms of the signal (considering it as noise). Then it uses these values to filter the whole signal and determine correct start (N1) and end (N2) points. However, in some cases N1 and N2 might not be found correctly. To solve this, the user interface depicted in Fig. 1b adds the possibility to manually modify these endpoints.

Feature Extraction Once we have the SAD endpoints, we extract the features of each keyword. We used standard 39-dimensional MFCCs: 13 statics (12+energy), deltas and acceleration. Input signal is first filtered with a 25ms long

Hamming window and features are extracted every 10ms. Global cepstral mean and variance normalization (CMVN) is applied to all features in each keyword.

DTW Alignment Whenever the user records the keyword twice we apply a DTW alignment to transform both keyword instances into one (stronger) keyword. The purpose of doing so is to improve the system by using a mean keyword, instead of just one instance of that keyword. This can be useful when keywords are recorded in a noisy environment, where the SNR needs to be improved by emphasizing those features in the keyword that are common and averaging out those that belong to noise. In this case, the second keyword instance is aligned to the first at MFCC feature level and then each frame of the second recording is averaged with its corresponding MFCC frames of the first recording. If more than one frame of recording 2 corresponds to one frame of recording 1, then the average between all those is computed. Finally we obtain a mean feature vector that has as many frames as recording 1.

The same procedure can be potentially applied to more than two recorded keywords. In our system we limited it to two to keep the UI and the user interaction pleasant. In this respect, if only one keyword is recorded, this step is skipped.

Keyword Storage The newly created keyword is saved into the system together with all the extra information provided (keyword features, name, SAD boundaries, action and detection threshold). Automatic values of SAD and detection threshold are initially stored, although these can be manually changed using the interface in Figure 1b. Once stored, the keyword is immediately available for matching.

2.3 The Monitor Mode

The "Monitor Mode" listens to audio captured from the cellphone's microphone and reacts when one of the prerecorded keywords is matched. In order to reduce false alarms and to allow a free selection of keywords leading to actions in the phone, the system first expects to match a "trigger" keyword and then listens for 2 seconds to match either of the "action" keywords. If after 2 seconds no action keyword is detected, the trigger keyword is forgotten and the system goes back to searching for a trigger. Both keyword matching steps are technically identical, as shown in Fig. 2 and described next.

Audio Buffering and Online Feature Extraction Incoming audio is constantly stored in a circular buffer. Once the buffer contains enough audio, an acoustic feature vector is computed. Note that unlike in the Recording mode, in here, the feature extraction module works online. This affects the way that features are normalized, as we do not have access to the whole signal at once. We use a running mean and variance estimation as follows.

The running mean μ' of a feature vector x is estimated as $\mu' = \alpha\mu + (1-\alpha)x$ where we set $\alpha = 0.995$ and μ is the running mean value at the previous frame. To compute the running variance, we use a similar approach. Given that $\sigma^2 = E(x^2)' - E(x)^2$, we compute $E(x)^2 = \mu^2$ and $E(x^2)' = \alpha E(x^2) + (1-\alpha)(x^2)$.

Keyword Matching Algorithm The core of the application is the keyword matching algorithm. As mentioned above, the system constantly monitors the audio input and only reacts when the "trigger" keyword is detected. Then a span of 2 seconds is allowed to detect any of the "action" keywords, or the system resets and starts again looking for a "trigger" keyword. The use of a "trigger" keyword is used in many modern systems such as [7] , Kinect voice recognition or Google Now. In this application we first run a single keyword matching instance to match the input audio to the "trigger" keyword. Once it is detected, several (one for every "action" keyword) DTW instances are run in parallel to detect whether any of the prerecorded keywords is spoken.

In our application we implement keyword matching by using a modification of the dynamic time warping (DTW) algorithm as proposed in [1]. Standard DTW algorithms cannot be used here as the input audio is not bounded, thus we cannot build standard global alignment matrices to compare both patterns. Instead, we use the MES-DTW algorithm [1] in online mode.

As acoustic frames become available, these are first compared to the "trigger" keyword. The MES-DTW algorithm allows us to perform the comparison by using only 3 support vectors, which is sufficient to find matches when no alignment path (i.e. matrix trackback) is required. The main difference of the current implementation with the MES-DTW algorithm in [1] is that in the later the system tries to align small queries with big references, but in here, we need to align an infinite(online) query with small references.

The matching algorithm works as follows. For every new input feature vector we first compute its distance to all features in the keyword(s) we want to match with. In this work the normalized cosine distance has been used. The distance vector is then used to update the global distance matrix, which in this case is limited to a $2xN$ matrix, where N is the number of frames of a keyword, stored for the previous frames $t-2$ and $t-1$. To update the matrix we use standard DTW local constraints of insertion, deletion and assignment. In addition, a single N-dimensional vector is kept to count how many matching frames each optimum path in position t contains. This is used to perform a local normalization of all values before local constraints are applied. For more details on the MES-DTW algorithm please refer to [1]. In order to impose a local version of the global Sakoe-Chiba band [8], we disallow more than 3 continuous insertions or deletions by keeping a count of previous decisions. After processing every frame we compare the normalized score at position N of time t. If the value is lower than the predefined threshold for that keyword a match is hypothesized. If more than one keyword instance finds a match, the one with the best score is selected. Once a match is granted, all vectors are reset so that a transitory time (of N frames) passes before more matches are allowed.

Showing Matches and Performing Actions Once a keyword has been detected (i.e. the normalized similarity is better than the selection threshold) if the keyword is one of the "action" keywords and less than 2 seconds passed after the "trigger" keyword was detected, then the action associated to that keyword is launched on the phone. We implemented the following 8 actions through calls to the cellphone core API: Open music player, turn volume ON/OFF, pick up a call, start/end voice note, Open camera and take a photo.

3 Evaluation

In this section we first describe a database we recorded in order to evaluate the system, the evaluation protocol we used and the experimental results we obtained.

3.1 Database Description

In order to obtain experimental results of the accuracy of our system, a database was recorded and labelled. We recruited two users to record a set of keywords in isolation and embedded inside longer utterances, in different acoustic conditions. Every user recorded five keywords and 120 utterances. The keywords were short (less than one second), and the utterances were of around 15 seconds long. Overall we recorded about 30 minutes of audio (15s/utterance * 120 utterances) for each user. The 120 queries were recorded in 4 different backgrounds:

1. Quiet background.
2. Quiet background in a small room (lots of reverberation).
3. Background with music.
4. Street with lots of noise.

In every background condition we recorded the 5 keywords contained in 6 utterances each (totaling 30 utterances recorded per background). The first three recordings contained the keyword alone, with the rest of the utterance containing just background noise. The other three utterances consisted on long sentences that contained the keyword within. In total the database contains around 1h of data. All recordings were done using a smartphone decide, using a standard voice recording application.

For every recording, we manually labelled the time when the user starts and stops saying the keyword. We used that information to evaluate the system, and to determine if a match found was correct or not, or if a miss occurred.

3.2 Evaluation Setup

In this section we're going to explain how the accuracy of the system was evaluated.

All of the 120 recordings available per user were run through the MES-DTW algorithm and compared with the 5 keywords, thus simulating a real use-case

scenario (we simulated the process of extracting the live features and computing the live DTW). We did this process for a variety of detection thresholds, trying to find the optimum threshold values and obtaining some curves depending on each threshold. Instead of using a different threshold for each keyword, we used the same threshold for all keywords of each user. Better results would be achieved by using an appropriate threshold for each keyword, but as threshold setting is not a user-friendly activity, and while we do not find a way to automatically set the optimum threshold, we set a unique threshold for all backgrounds.

Then, for a given threshold t and an utterance $q1$ we detect how many matches or false alarms appear. To do this, if a match does not appear when it was expected to appear (between its endpoints previously labelled, plus a margin of 0.2 seconds), we count that as a false negative. If a match appears when it doesn't have to appear, we count that as a false positive. And if a match appears when it has to appear, then we have a true positive. Finally, if a match appears twice within its endpoints, we count one true positive and one false positive.

In order to compute meaningful statistics we also consider the following values:

- The max number of true positives is 120 (the total number of queries)
- The max number of false positives depends on the total length of all the queries. If the total length is 1800s, and we can have an alarm every 1.2s for each reference, we can find up to 1800*5/1.2 alarms, so 7500 alarms, where 120 of them are true positives. So, the difference is the maximum number of false positives. Actually, the total length wasn't 1800s, it depended on every user. The total number of possible alarms was computed for every user.
- The max number of false negatives is 120 (the total number of queries)

We measure performance using standard information retrieval metrics:

$$\text{Precision} = \frac{tp}{tp+fp} \qquad \text{Recall} = \frac{tp}{tp+fn} \qquad \text{Fscore} = 2\frac{\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}}$$

$$\text{Misses} = \frac{max\_tp - tp}{max\_tp} \qquad \text{FalseAlarms} = \frac{fp}{max\_fp}$$

Where tp denotes "true positives", fp denotes "false positives", fn denotes "false negatives", max_tp denotes the possible number of true positives, and max_fp denotes the possible number of false positives, as defined above.

3.3 Experimental Results

We performed experiments individually for each of the two speakers that recorded the database. Table 1 shows results for user 1, and Table 2 shows them for user 2. We show results using just the first recording the user made for each keyword, the second one, or the mix (mean features between the two aligned sets of

features). All thresholds were set per speaker and task, using a held-out development set, and are reported in the last column of each table. We can see that results are very different between the two users. While user 1 obtains very satisfactory results (very high Fscore and a low number of misses and false alarms) user 2 obtains much worse Fscore, mostly due to the very high rate of misses (e.g. the application did not detect the right keyword when he spoke it). The reason for this behavior is probably because user 1 knows how to clearly speak to the application and how to use it better (he was involved with the application much more before recording the database) than user 2.

We can also see that the fact of using the mean features is not helping the system to get better results. That probably happened because the keywords in the tests were recorded in clear environments or due to the method we used to align the features from both keywords. More research is due to find out the reasons and to correct them.

Table 1. Results for user 1

Type of test	F score	misses	false alarms	Thres.
Mean Features	0.8559	0.067	0.009	0.43
First Recording	0.8219	0.25	0.001	0.44
Second Recording	0.8642	0.083	0.004	0.45

Table 2. Results for user 2

Type of test	F score	misses	false alarms	Thres.
Mean Features	0.4976	0.5667	0.004259	0.36
First Reference	0.5225	0.5167	0.006033	0.39
Second Reference	0.5804	0.4583	0.0045	0.38

4 Conclusions and future work

The use of smartphones is becoming ubiquitous. As the keyboards in these devices are quite small and uncomfortable to use, there is a very good opportunity for speech technology to help users be most efficient when using these devices. In this paper we present a prototype application we developed as a proof of concept of a small-footprint flexible voice-enabled command-and-control application for the cellphone. With the use of the application the user can record acoustic keywords and associate them to actions in the phone. Then the phone can be set to listen to the environment and react whenever one of the keyword is detected. We have implemented the application to work on an Android device totally offline (no connectivity required) and in real-time. In this paper we evaluate the

application in terms of matching accuracy in different acoustic environments to see whether it would be useful in a real-live setting. We observe that results are positive but vary a lot depending on the user. Future work will include further quantitative tests with a bigger database as well as a qualitative test to gather insights on how/when users would use this application.

References

1. X. Anguera and M. Ferrarons, "Memory efficient subsequence dtw for query-by-example spoken term detection," in *2013 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2013, pp. 1–6.
2. J. Schalkwyk, D. Beeferman, F. Beaufays, B. Byrne, C. Chelba, M. Cohen, M. Kamvar, and B. Strope, ""your word is my command": Google search by voice: A case study," in *Advances in Speech Recognition*, A. Neustein, Ed. Springer US, 2010, pp. 61–90. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-5951-5_4
3. A. Acero, N. Bernstein, R. Chambers, Y. Ju, X. Li, J. Odell, P. Nguyen, O. Scholz, and G. Zweig, "Live search for mobile:web services by voice on the cellphone," in *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, March 2008, pp. 5256–5259.
4. W. Shen, C. M. White, and T. J. Hazen, "A comparison of query-by-example methods for spoken term detection," DTIC Document, Tech. Rep., 2009.
5. L. R. Rabiner and M. R. Sambur, "An algorithm for determining the endpoints of isolated utterances," *Bell System Technical Journal*, vol. 54, no. 2, pp. 297–315, 1975.
6. H. Lee, S. Chang, D. Yook, and Y. Kim, "A voice trigger system using keyword and speaker recognition for mobile devices," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 4, pp. 2377–2384, 2009.
7. H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 26, no. 1, pp. 43–49, 1978.