A PITCH EXTRACTION ALGORITHM TUNED FOR AUTOMATIC SPEECH RECOGNITION

Pegah Ghahremani[1], *Bagher BabaAli*[2], *Daniel Povey*[1],
Korbinian Riedhammer[3], *Jan Trmal*[1], *Sanjeev Khudanpur*[1]

[1] Johns Hopkins University, {pghahre1,khudanpur}@jhu.edu,{dpovey,jtrmal}@gmail.com
[2] University of Tehran, Iran, babaali@ut.ac.ir [3] ICSI, Berkeley, korbinian@ieee.org

ABSTRACT

In this paper we present an algorithm that produces pitch and probability-of-voicing estimates for use as features in automatic speech recognition systems. These features give large performance improvements on tonal languages for ASR systems, and even substantial improvements for non-tonal languages. Our method, which we are calling the Kaldi pitch tracker (because we are adding it to the Kaldi ASR toolkit), is a highly modified version of the getf0 (RAPT) algorithm. Unlike the original getf0 we do not make a hard decision whether any given frame is voiced or unvoiced; instead, we assign a pitch even to unvoiced frames while constraining the pitch trajectory to be continuous. Our algorithm also produces a quantity that can be used as a probability of voicing measure; it is based on the normalized autocorrelation measure that our pitch extractor uses. We present results on data from various languages in the BABEL project, and show a large improvement over systems without tonal features and systems where pitch and POV information was obtained from SAcC or getf0.

Index Terms— Automatic Speech Recognition, Pitch, Tone, Probability Of Voicing

1. INTRODUCTION

Our goal in this work was to obtain good-performing pitch and Probability of Voicing (POV) features for use in speech recognition, and specifically to produce a standardized pitch feature for use in the Kaldi Automatic Speech Recognition (ASR) toolkit [1]. In Section 2 we review our work to select the best previously published pitch extraction algorithms; in Section 3 we describe our proposed method. In Section 4 we describe the pitch post-processing method we used for the baseline pitch and POV features, and in Section 5 we describe the post-processing we use with our proposed method. We describe our ASR system and data-sets in Section 6, give experimental results in Section 7, and conclude in Section 8.

The work of Pegah Ghahremani, Jan Trmal, Daniel Povey and Sanjeev Khudanpur was partially supported by IARPA BABEL contract No W911NF-12-C-0015. Povey and Khudanpur were partially supported by DARPA BOLT contract No HR0011-12-C-0015 and are affiliated with the Human Language Technology Center of Excellence. Bagher BabaAli began this work while he was a Visiting Scholar at Johns Hopkins University, where he received partial support from Cisco Research Award CG#574560 and from a gift by GoVivace Inc. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors alone and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DARPA/DoD, or the U.S. Government.

2. EXISTING PITCH EXTRACTION METHODS

We started our work by obtaining various off-the-shelf pitch extractors, namely Yin [2], Getf0 [3], SAcC [4], Wu [5], SWIPE [6] and YAAPT [7]. We compared their accuracy as pitch trackers (see Sec. 7). For this we used the Keele database [8], which consists of about half an hour of speech manually labeled for pitch and voicing. We selected three of the best-performing methods for further study; these were SAcC, Yin and getf0 (we did not consider YAAPT at this point because of its greater complexity; it is based on getf0).

Next, as will be seen in the experimental section, we compared the pitch features of SAcC, Yin and getf0 in an ASR task. For this comparison we processed the pitch as described in Section 4 and used the voicing feature from SAcC. These experiments did not show very large differences between the various pitch extractors, so we used getf0 as our starting point as it seemed to perform slightly better than Yin, and it is a relatively simple algorithm to implement (SAcC gave better performance but it is a fairly complex method).

Due to space limitations we cannot give a general overview of pitch extraction methods; we refer the interested reader to the references, particularly to [3] which describes getf0.

3. THE KALDI PITCH EXTRACTOR

Parameter	Value	Explanation
min-f0	50	Minimum possible frequency value (Hz)
max-f0	400	Maximum possible frequency value (Hz)
window-width	0.025	Length in seconds of window used for NCCF
window-shift	0.01	Frame-shift, in seconds (should match that used for baseline features e.g. PLP)
soft-min-f0	10	Minimum f0, applied in soft way; must not exceed min-f0.
nccf-ballast	0.625	Increasing this factor reduces NCCF for quiet frames, helping ensure pitch continuity in unvoiced regions
penalty-factor	0.1	Factor that penalizes frequency change
delta-pitch	0.005	Smallest relative change in pitch that our algorithm measures
lowpass-cutoff	1000	Low-pass cutoff that we apply to the raw signal
lowpass-filter-width	2	Integer that determines filter width of low-pass filter (more gives wider filter with sharper cutoff)
resample-frequency	4000	Sample frequency for NCCF; must exceed twice lowpass-cutoff.
upsample-filter-width	5	Integer that determines filter width when upsampling NCCF

Table 1. Parameters of our algorithm, and their default values

Our algorithm is a highly modified version of the getf0 algo-

rithm. It is based on finding lag values that maximize the Normalized Cross Correlation Function (NCCF). Like most pitch extraction algorithms, our algorithm has a number of parameters that were set by hand. We show these and their values in Table 1. It should not be necessary to change any of these values when applying it to data of different sampling rates.

Probably the most important change from getf0 is that rather than making hard decisions about voicing on each frame, we treat all frames as voiced and allow the Viterbi search to naturally interpolate across unvoiced regions. In order to make this happen we had to make a few changes. We do not limit the search to the approximate local maxima of the NCCF– we allow it to take any value on a reasonably fine grid. We alter the penalty on the change Δ in log-pitch from proportional to $\mathrm{abs}(\Delta)$ to Δ^2, which causes the algorithm to linearly interpolate across constant regions of the NCCF. And we add a "ballast" term to the NCCF formula which makes it approach zero for "quiet" regions of the signal; for this to work, we have to globally energy-normalize the signal. This requires lookahead, as doe the Viterbi search; in future we plan to create a modified version of the algorithm for online use.

We also low-pass the signal to 1kHz; this improves accuracy as well as making the algorithm more efficient by allowing us to work with a sub-sampled signal. And we obtain a feature based on the NCCF which is related to the probability of voicing and which helps in ASR.

3.1. Resampling method

For completeness we will specify the method we use to resample signals (you may skip this subsection if the details are not needed). Let the sampled source signal be viewed as a continuous function of time $s(t)$, where the n'th sample x_n becomes a Dirac delta function shifted to time n/S where S is the sampling rate, and scaled by x_n/S. We define a filter function $f_{C,w}(t)$, parameterized by a cutoff frequency $C \leq S/2$, and an integer width factor $w \geq 1$. Let the window function $w(t)$ be a raised-cosine (Hanning) window with support on $\left[\frac{-w}{2C}, \frac{w}{2C}\right]$. Then define

$$f_{C,w}(t) = 2C\,\mathrm{sinc}(2Ct)w(t) \tag{1}$$

where sinc is the normalized sinc function. To take a sample of the signal at an arbitrary time t, we simply evaluate $\int_u s(u)f(t-u)$ which is the sum $s'(t) = \sum_n x_n \frac{f_{C,w}(t-n/S)}{S}$. Naturally we only evaluate this for the values of n for which the summand is nonzero.

3.2. Subsampling and normalization

Let the input to the algorithm be a discretely sampled signal, sampled with sampling frequency S. The first stage is to use the resampling method above, with the filter parameterized by lowpass-cutoff and lowpass-filter-width, to sample the signal at sampling frequency resample-frequency. Next we normalize the resampled signal's dynamic range by dividing by the root-mean-square signal value (if it is nonzero). Let the result be the signal x_n, with $n = 0, 1, \ldots N-1$.

3.3. Computing the NCCF

First we need to establish the range of lags over which to compute the NCCF. These depend on the frequency range we search over. Define the quantities min-lag $= 1/$max-f0, max-lag $= 1/$min-f0, which are the minimum and maximum lags(in seconds) at which we need the NCCF, and furthermore define upsample-filter-frequency as resample-frequency$/2$ which is the filter cutoff we will use when upsampling the NCCF. Then with filter-width w (in seconds) defined as upsample-filter-width$/$upsample-filter-frequency, let outer-min-lag $=$ min-lag $- w/2$ and outer-max-lag $=$ max-lag $+ w/2$, which gives us a slightly larger range of lags over which to compute the NCCF (we need to extend the range by half the width of the filter function we'll use when up-sampling the NCCF).

Consider the frame-index $t = 0, 1, \ldots$. The time span of the signal that we need to process starts at the closest sample to the time $t \cdot$ window-shift and is of length window-width $+$ outer-max-lag (in seconds). We produce output for all frame-indices t such that this time span is wholly within the time span of the input file. Let $\mathbf{w}_t = (w_{t,0}, w_{t,1}, \ldots)$ be the sequence of samples used for frame t; this is a subsequence of the sequence x_n, of length $\lceil$(window-width $+$ outer-max-lag) $\cdot$ resample-frequency$\rceil$ samples, but with its mean subtracted away. Let $\mathbf{v}_{t,i}$ represent the sub-sequence of $\mathbf{w}_t$ starting at position i and of length $n = \lceil$window-width $\cdot$ resample-frequency$\rceil$, so for instance $\mathbf{v}_{t,3} = (w_{t,3}, \ldots, w_{t,n+3})$. Where convenient we will view these sequences as vectors. The NCCF for frame t and lag-index l is

$$\phi_{t,l} = \frac{\mathbf{v}_{t,0}^T \mathbf{v}_{t,l}}{\sqrt{||\mathbf{v}_{t,0}||_2^2 ||\mathbf{v}_{t,l}||_2^2 + n^4\,\text{nccf-ballast}}}, \tag{2}$$

where $||\mathbf{x}||_2^2 = \mathbf{x}^T\mathbf{x}$. We compute this for all l s.t. outer-min-lag $\leq l/$resample-frequency $\leq$ outer-max-lag.

3.4. Upsampling the NCCF

Next we upsample the NCCF in a non-linear way: that is we measure the NCCF at the geometrically increasing sequence of lag values

$$L_i = \text{min-lag}\,(1 + \text{delta-pitch})^i\,,\ i \geq 0,\ L_i \leq \text{max-lag}, \tag{3}$$

where the condition $L_i \leq$ max-lag determines the maximum index i. For each index i we compute the NCCF $\Phi_{t,i}$ which is the NCCF ϕ_{t,L_i} measured on frame t at lag L_i, using the resampling method described in Section 3.1 parameterized by upsample-filter-frequency and upsample-filter-width.

3.5. Defining the cost function

Suppose the range of the frame-index t is $0 \leq t < T$ and the range of the lag index i is $0 \leq i < I$ (we have mentioned in Sections 3.3 and 3.4 how these ranges are determined). The pitch trajectory is obtained by minimizing a cost function defined on a sequence of indices $\mathbf{s} = (s_t)_{t=0}^{T-1}$; each element s_t is interpreted as a lag-index i, so $0 \leq s_t < I$. The cost function consists of a local cost for each time t plus a term that penalizes changes in frequency:

$$C(\mathbf{s}) = \sum_{t=0}^{T-1} \text{local-cost}(t, s_t) + \sum_{t=1}^{T-1} \text{penalty-factor}(\log(L_{s_t}/L_{s_{t-1}}))^2, \tag{4}$$

where the configuration value penalty-factor controls how strongly we penalize changes in frequency, and we define

$$\text{local-cost}(t, i) = 1 - \Phi_{t,i}(1 - \text{soft-min-f0}\,L_i) \tag{5}$$

View $1 - \Phi_{t,i}$ as the basic local-cost, and the multiplicative factor on $\Phi_{t,i}$ as a kind of penalization of high lags, which will tend to keep the selected lags substantially below $1/$soft-min-f0.

3.6. Optimizing the cost function

The algorithm we use to compute the sequence $\mathbf{s}$ that minimizes $C(\cdot)$ is based on the Viterbi algorithm. A naïve implementation would take time quadratic in the number I of lag-indices. Let the Viterbi back-trace on time $t > 0$ and lag i be $b(t, i)$; this evaluates to an integer index (like i) that is the optimal lag-index on time $t-1$ that we "point back to" from (t, i). Due to convexity in i of the transition-cost, we can show that $b(t, i + 1) \geq b(t, i)$. We can use this to obtain an exact search algorithm that in takes time closer in practice to linear in I (although not provably so; it is data-dependent). Let the forward-cost be written as $c(t, i)$. Ignoring all end effects for purposes of exposition, the basic idea is that, on time t, we first do a "forward pass" for $i = 0$ to $I-1$, and set $c(t, i)$ and $b(t, i)$ while only considering the previous forward-costs $c(t-1, j)$ for $j = b(t, i-1)$ to i. Then, in a "backward pass" for $i = I-1$ to 0, we see whether we can get a better forward-cost and corresponding backtrace than we already have by considering the previous forward-costs $c(t-1, j)$ for $j = i + 1$ to $b(t, i + 1)$. Let the result of this computation be the state-sequence $\mathbf{s} = (s_t)_{t=0}^{T-1}$.

3.7. Output

The output of this algorithm is the pitch on each frame and also the NCCF on each frame. The pitch on frame t equals $1/L_{s_t}$, with lags L_i as defined in (3). The NCCF values are computed at the selected lags, so on frame t we output Φ_{t,s_t} (see Sec. 3.4); however, for purposes of this output we compute the NCCF without the nccf-ballast term of (2) (and treating $0/0$ as zero in case of a zero sequence in the signal). This means that we need to do the upsampling computation of the NCCF twice. Below we describe how we post-process the output for use as features for ASR.

4. BASELINE PITCH POST-PROCESSING METHOD

This post-processing that we used for all the non-Kaldi pitch features is based on [9, pp.46,54], and is similar to the system of the "Swordfish" team [1] in the BABEL program (IARPA-BAA-11-02); our own experimental setup is part of the "Radical" team's larger system. First, if there are regions where the pitch extractor says there is no voicing, we interpolate the pitch values from the adjacent voiced region in a straight line across the gap; or for unvoiced regions at file boundaries, we simply continue the first or last pitch value. We also add a little noise to the pitch values at this point. Then, we take the log of the resulting pitch values. We then apply mean subtraction, subtracting the mean of a window of length 151 frames, centered on the current frame. To the resulting pitch, we apply short-time smoothing, averaging over a centered window of 5 frames. The reason why it is necessary to add noise and to do short-time averaging, is that many pitch extractors (including SAcC) output pitch that is quantized to discrete values, which produces a "blocky-looking" pitch trace. These operations help make the pitch trace smoother.

The POV feature is obtained as follows, and note that for all baseline systems we used the POV estimates from SAcC. We used $\log((p + 0.0001)/(1.0001 - p))$ as the feature, where $0 \leq p \leq 1$ is the POV estimate from SAcC. So the output is a two features representing pitch and POV. We append these to the PLP features, and from then on treat them the same way as we would treat extra PLP coefficients (i.e. we apply cepstral mean subtraction, and delta computations or splicing followed by LDA).

[1] Thanks to Arlo Faria who developed the pitch processing for that system

5. PITCH POST-PROCESSING FOR KALDI PITCH EXTRACTOR

5.1. Processing NCCF into a POV measure

The basis for our POV measures are the NCCF values for each frame (see Sec. 3.7). Its range is $[-1, 1]$, but it is usually positive. We process the raw NCCF value in two ways, for reasons that will become clear.

5.1.1. Method intended to give accurate probability of voicing

The first method is only used as part of the pitch mean-subtraction algorithm we describe below; it processes the NCCF value into a reasonably accurate probability of voicing measure. The following formula was obtained by plotting the log of count(voiced) / count(unvoiced) on the Keele database as a function of the NCCF, and manually creating a function to approximate it.

Let the NCCF on a given frame be written c. Compute its absolute value: let $a = \mathrm{abs}(c)$. Then let $l = -5.2 + 5.4\exp(7.5(a-1)) + 4.8a - 2\exp(-10a) + 4.2\exp(20(a - 1))$. Here, l is intended to approximate the log-likelihood ratio $\log(p(\text{voiced})/p(\text{unvoiced}))$. Then let $p = 1/(1 + \exp(-l))$, and p is a reasonable approximation to the probability of voicing on this frame.

5.1.2. Method for use as a feature

The other method we use to process the NCCF, gives a value that seems to give good performance when used as a feature. This method was designed to give the feature a reasonably Gaussian distribution (although there are still noticeably separate peaks for voiced and unvoiced frames). If $-1 \leq c \leq 1$ is the raw NCCF, we let the output feature be $f = 2\left((1.0001 - c)^{0.15} - 1\right)$.

5.2. Normalization of pitch

We use the short-time mean subtraction approach of [9], but with POV weighting: on each time t we subtract a *weighted* average pitch value, computed over a window of width 151 frames centered at t and weighted by the probability of voicing value p described in Section 5.1.1. Note: the improvement in WER from incorporating the weighting in the mean subtraction was quite small: of the order of 0.1% WER, averaged across various languages.

5.3. Delta feature

Since the initial version of this paper was written we have extended our post-processing by adding to the pitch and POV features a third feature consisting of the delta-log-pitch computed directly from the un-normalized log pitch, in the normal way (using ± 2 frames of context). The motivation was to get an exact delta-pitch feature without inaccuracies caused by the moving-window mean subtraction. This, together with the previous two features, is appended to the raw MFCCs or PLPs. This gave us around 0.4% absolute improvement on top of the results we show below.

6. SYSTEM DESCRIPTION

6.1. Kaldi BABEL pipeline

Our system is mostly as described in [10], although we have made various improvements since then. We measure our systems using %WER and using Actual Term Weighted Value (ATWV), which is a measure of keyword search effectiveness [11]; larger values are better. We train on the LimitedLP training data, which is around 10 hours per language.

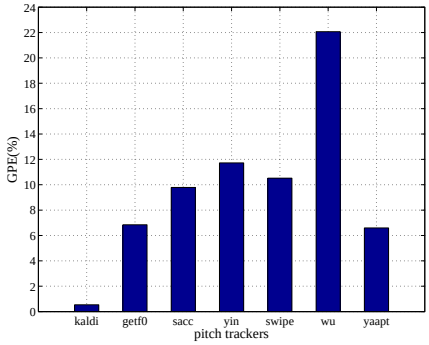

Fig. 1. Gross Pitch Error (% voiced frames >10% pitch error): Keele database

Our current recipe is based on combination of three types of Kaldi system, but for speed we only test one of them here: it is a Subspace Gaussian Mixture Model (SGMM) system [12], discriminatively trained with boosted MMI (bMMI) [13].

Of the languages we tested, only Vietnamese[2] and Cantonese[3] have tone marked in the dictionary. We configure Kaldi in such a way that the acoustic decision tree can ask about tone. Of the other languages, Zulu[4] is the only one that is considered to be a tonal language, but its lexicon was not marked for tone. We also tested on Assamese[5] and Bengali[6] In all cases we tested on the official Dev10h development set. For the keyword search, the development keyword phrases (usually performer provided) lists were used. Note: while we get improvements on pitch in even the atonal BABEL languages, separate experiments on Switchboard English showed no improvement from our features, compared with just MFCC: possibly English is exceptional in some way.

In some of our experiments we appended Fundamental Frequency Variation (FFV) features [14]; these are seven-dimensional features which are informative about pitch changes. These features were part of our standard pipeline when our pitch features were based on SAcC, but we find that they are not helpful in combination with the features from our improved pitch tracker.

7. EXPERIMENTAL RESULTS

Figure 1 compares our pitch tracker with various baselines, using the Keele corpus. Our pitch tracker gives substantially better accuracy than the others (but bear in mind that we tuned it on this setup and that some baselines' error may be inflated because they classified some frames as unvoiced).

In Table 2, we compare the Yin, Getf0 and SAcC pitch trackers on Vietnamese data. Because not all the pitch trackers provide a POV, we used the SAcC POV. We also added FFV [14] features, as these were part of our original SAcC-based recipe. SAcC was still the best of the original pitch trackers we tested, but due to the simplicity of getf0 we felt it was the best starting point for our work.

[2]Language collection release `IARPA-babel-107b-v0.7`.
[3]Language collection release `IARPA-babel-101b-v0.4c_sub-train1`
[4]Language collection release `IARPA-babel-206b-v0.1d`
[5]Language collection release `IARPA-babel-102b-v0.4`.
[6]Language collection release `IARPA-babel-103b-v0.3`.

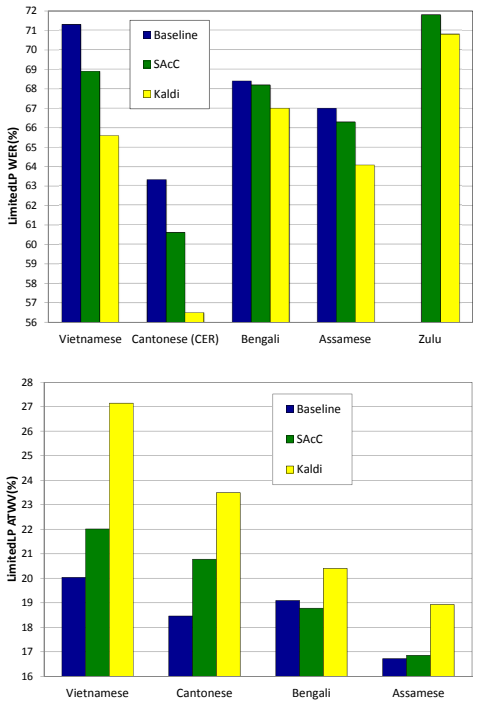

Fig. 2. No-pitch vs. (SAcC vs. Kaldi) (Pitch+POV)

Features	WER
Yin pitch + SAcC POV + FFV	68.1
GetF0 pitch + SAcC POV + FFV	68.0
SAcC pitch + SAcC POV + FFV	67.6

Table 2. Off-the-shelf pitch extractors (Vietnamese LimitedLP)

		Vietnamese		Cantonese	
Pitch	POV	%WER	%ATWV	%CER	%ATWV
-	-	71.3	20.0	63.3	18.5
SAcC	SAcC	68.9	22.0	60.6	20.8
Getf0	SAcC	68.8	21.3	60.1	20.07
Kaldi	SAcC	67.1	24.0	58.1	24.1
Kaldi	Kaldi	65.6	27.14	56.5	23.49

Table 3. Comparing pitch and POV algorithms on tonal languages

Table 3 shows various combinations of pitch and POV features, on tonal languages, without FFV features. It can be seen that the Kaldi pitch and POV features are each better than the corresponding SAcC-based feature.

8. CONCLUSIONS

We have proposed a robust pitch tracking algorithm based on normalized cross-correlation. The performance evaluation on Keele dataset showed that our algorithm improves pitch tracking as measured by GPE, versus the off-the-shelf methods we tested. Experiments on four different languages shows that the technique gives consistent improvements versus a PLP based system, with a typical improvement of 6 % absolute on tonal and 2 % absolute on atonal languages, which is roughly double the improvement we get from the baseline SAcC-based features.

9. REFERENCES

[1] D. Povey, A. Ghoshal, et al., "The Kaldi Speech Recognition Toolkit," in *Proc. ASRU*, 2011.

[2] Alain De Cheveigné and Hideki Kawahara, "Yin, a fundamental frequency estimator for speech and music," *The Journal of the Acoustical Society of America*, vol. 111, pp. 1917, 2002.

[3] David Talkin, "A robust algorithm for pitch tracking (rapt)," *Speech coding and synthesis*, vol. 495, pp. 518, 1995.

[4] Daniel PW Ellis and Byunk Suk Lee, "Noise robust pitch tracking by subband autocorrelation classification," in *13th Annual Conference of the International Speech Communication Association*, 2012.

[5] M. Wu, D.L. Wang, and G.J. Brown, "A multipitch tracking algorithm for noisy speech," *IEEE Transactions on Speech and Audio Processing*, vol. 11, no. 3, pp. 229–241, 2003.

[6] A. Camacho and J. G. Harris, "A sawtooth waveform inspired pitch estimator for speech and music," *Journal of the Acoustical Society of America*, vol. 124, no. 3, pp. 1638–1652, 2008.

[7] Kavita Kasi and Stephen A Zahorian, "Yet another algorithm for pitch tracking," in *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*. IEEE, 2002, vol. 1, pp. I–361.

[8] F. Plante, Georg F. Meyer, and William A. Ainsworth., "A pitch extraction reference database," in *Eurospeech*, 1995, pp. 837–840.

[9] Xin Lei, *Modeling Lexical Tones for Mandarin Large Vocabulary Continuous Speech Recognition*, Ph.D. thesis, University of Washington, 2006.

[10] Guoguo Chen, Sanjeev Khudanpur, Daniel Povey, Jan Trmal, David Yarowsky, and Oguz Yilmaz, "Quantifying the value of pronunciation lexicons for keyword search in lowresource languages," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, 2013, pp. 8560–8564.

[11] J. G. Fiscus, Jerome Ajot, John S. Garofalo, and George Doddington, "Results of the 2006 spoken term detection evaluation," in *ICSLP*, 2007.

[12] D. Povey, L. Burget, et al., "The Subspace Gaussian Mixture Model–A Structured Model for Speech Recognition," *Computer Speech & Language*, vol. 25, no. 2, pp. 404–439, April 2011.

[13] D. Povey, D. Kanevsky, B. Kingsbury, B. Ramabhadran, G. Saon, and K. Visweswariah, "Boosted MMI for model and feature-space discriminative training," in *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, 2008, pp. 4057–4060.

[14] Kornel Laskowski, Mattias Heldner, and Jens Edlund, "The fundamental frequency variation spectrum," in *FONETIK*, 2008.