Dictionary-based pitch tracking with dynamic programming

Ewout van den Berg and Bhuvana Ramabhadran

IBM T.J. Watson Research Center
{evandenberg,bhuvana}@us.ibm.com

Abstract

Pitch detection has important applications in areas of automatic speech recognition such as prosody detection, tonal language transcription, and general feature augmentation. In this paper we describe Pitcher, a new pitch tracking algorithm that correlates spectral information with a dictionary of waveforms each of which is designed to match signals with a given pitch value. We apply dynamic programming techniques on the resulting coefficient matrix to extract a smooth pitch contour while facilitating pitch halving and doubling transitions. We discuss the design of pitch atoms along with the various considerations for the pitch extraction process. We evaluate the performance of Pitcher on the PTDB database and compare its performance with three existing pitch tracking algorithms: YIN, IRAPT, and Swipe’. The performance of Pitcher consistently outperforms the other methods for low-pitched speakers and is comparable in performance to the best of the other three methods for high-pitched speakers.

Index Terms: pitch detection, dynamic programming

1. Introduction

Pitch determination is a fundamental problem in audio processing with applications ranging from feature augmentation in speech recognition of tonal languages to prosody in speech synthesis, automatic melody detection and musical transcription in music, and yet others in speech coding and speaker identification (See [5, 7]). For most music applications the signals are polyphonic and therefore require the detection and tracking of multiple pitch contours at the same time. In this paper we limit ourselves to speech signals in which at most one speaker is active at any given time.

Over the years many pitch determination algorithms (PDA) have been developed. Typically, these fall in one of two groups: time-based PDAs and short-term analysis PDAs [5]. The first group of algorithms works directly on the (filtered) audio signal, whereas the second works on data obtained using short-term segmentation and transformation. Our algorithm, Pitcher, is based on the spectrogram and therefore falls in the second category. We evaluate the performance of Pitcher on the PTDB-TUG [7] database and compare it against three existing methods: YIN [3], IRAPT [1], and Swipe’ [2]. The well-known YIN algorithm is based on the classic autocorrelation method and uses several additional steps to improve performance. IRAPT is a PDA for instantaneous pitch estimation and is based on RAPT [9], which uses the normalized cross-correlation function for its initial pitch estimation. Swipe’, like our proposed method, extracts pitch information from the spectrogram by applying a set of filters. The important differences between Swipe’ and Pitcher will be detailed later in the article.

We develop a dynamic programming model to ensure that the pitch contour extracted from the data is piecewise smooth. The use of dynamic programming in PDAs itself, however, is not new: the technique is also used in RAPT and appeared much earlier in [6, 8]; for more information, see [5, 9].

In speech, pitch is meaningful only in voiced segments and the classification in voiced and unvoiced regions is therefore an important component in pitch tracking. Such voicing detection algorithms (VDA) can be incorporated at various stages of the PDA, or be a separate stand-alone postprocessing step. For the current version of Pitcher, we decided to focus on pitch tracking itself. Nevertheless we discuss extensions to the algorithms for voicing classification in more detail later.

2. Algorithm

The proposed pitch-tracking algorithm, called Pitcher, works in the frequency domain and takes as input a spectrogram, $F = [f_1, \ldots, f_n]$, with vector $f_k$ representing the frequency spectrum at time $t_k$. In voiced regions, pitch and its harmonics appear as a series of equally spaced peaks of varying intensity in the corresponding vector $f$. A natural way of detecting the pitch therefore is to correlate $f$ with waveforms that correspond to different pitch values and choose the one with the highest correlation. Below we give a more refined version of this, but both approaches rely on a dictionary of waveform atoms $D = [d_1, d_2, \ldots, d_m]$ that are designed to correlate well with corresponding pitch values $p = [\varphi_1, \varphi_2, \ldots, \varphi_m]$. For example, given target pitch value $\varphi$, a sampling rate $s$ (Hz), and a window length $w$, we could choose a cosine atom $d$ with entries

$$d[k] = \cos \left( \frac{ks}{w\varphi} \right), \quad k = 0, \ldots, w/2 - 1 - 1. \quad (1)$$

The main design parameters in general are the shape of the waves, the offset and amplitude envelopes, and the normalization. To account for weak fundamentals or otherwise fluctuating intensity levels of pitch harmonics it may be desirable to have several waveforms corresponding to a single pitch value with one waveform for each variation. The exact shape of the atoms depend in part on the way the spectrogram is computed. For example, their shape depends on the various preprocessing steps and whether the intensity is represented on a linear, logarithmic, or square-root scale.

Given the dictionary we compute the correlation coefficients as

$$C = D^T F,$$

where column $k$ of $C$ represents the coefficients at time $t_k$. From this information we extract a path of prominent coefficients, which are then mapped to pitch values based on $p$. Because pitch values are only valid in regions of voiced speech, a post-processing step is used to invalidate pitch values in regions marked at unvoiced. We now describe the different parts of the algorithm in more detail.
2.1. Dictionary generation

The dictionary design is intricately linked with the type of spectrum, so throughout the paper we use a spectrum with intensity values on the logarithmic scale that are clipped below and normalized such that the minimum is always zero.

Now, suppose we are given a vector $f$ from the spectrum with pitch level $\varphi$. Of all unit-norm vectors, the best atom we could choose for this instance is $f / \|f\|_2$. That is, we would like to correlate the spectrum with atoms that match the waveform in $f$ as closely as possible. Preliminary experiments showed that the cosine atoms in (1) did not work well, mostly because of their pronounced valleys with negative values. The first improvement therefore was to take only the positive part, $\max(-, 0)$, which we denote by $[\cdot]_+$. Although the resulting dictionary led to improved results it was found that the peaks in the atoms were too wide and bulky, especially for high pitch values. Consequently, the second improvement was to raise the atom entries to a power $\alpha > 1$, which increases the sharpness of the peaks:

$$d_k[k] = \left[\cos \left(\frac{k 8}{W^\varphi} \pi\right)\right]_+^{\alpha}.$$  

We found that a value between 2 and 3 worked well and we therefore settled for $\alpha = 2.5$.

For further processing of the atoms we follow Swipe' [2] in that we omit the first peak at the origin and divide the remaining entries $d[k]$ by the square root of the frequency, $ks/w$. Aside from this Swipe' differs in many aspects. First of all Swipe' works with square-root intensity values instead of logarithmic values in order to avoid problems with silence or small intensity values. Second, the wave forms implemented in Swipe' are piecewise sinusoidal and, as its name implies, only the peaks around to the pitch value itself and its prime-number multiples are used. A third difference in the implementation is that instead of using a single spectrogram, Swipe' computes a series of spectrograms with different window lengths, each used for different ranges of pitch values. Finally, unlike Swipe' we do not normalize the atoms to be unit norm because it was found to negatively affect the performance of our algorithm.

When the pitch values in $p$ are sorted and relatively closely spaced we expect that peaks in the coefficients at a given time are clustered around indices for which $\varphi_k$ closely matches the correct pitch, or, to a lesser extend, matches octave shifts of this value. Individual outliers can affect the pitch extraction process and we therefore first apply a diagonal smoothing matrix $S$ to the coefficients. The new coefficients are given by $SC = SD^T F$, and it follows that we can equivalently work with a new dictionary $SD$ that includes the smoothing step.

Depending on the sampling rate it is often the case that the first, say, ten multiples of the highest feasible pitch value span only a small portion of the spectrogram. In this case it is possible to discard the high-frequency information from the spectrum along with the corresponding rows in the dictionary. This allows us to substantially reduce the dictionary size and lower the computational cost without affecting the performance.

2.2. Dynamic programming

Given the correlation coefficients of the pitch atoms with the spectrogram, we need to extract the desired pitch values. The simplest way of doing this is to take the maximum entry in each column of $C$ and return the pitch value corresponding to the atom for that entry. This approach is fast and suitable for on-line processing, but it is also very sensitive to outliers and fluctuations. The pitch in voiced areas can be expected to be mostly smooth, aside from occasional and near-instantaneous pitch halving or doubling. To enforce this structure we developed a dynamic programming model that is tailor made for pitch tracking. The goal is to find a path along the columns such that the summation of the entries is maximal. However, this would simply coincide with taking the columnwise maximum and we therefore need to add constraints and penalties on the possible trajectories. Hence, for any given pitch value $\phi_k$ at time $k$ we only allow pitch values $\phi_i$ at time $k + 1$ which satisfy one of the following conditions. First, we allow relative pitch changes with factor $0 < \beta_1 < 1$, giving the interval $[1 - \beta_1] \phi_k, (1 + \beta_1) \phi_k]$. Second, we allow pitch values around $\phi_k / 2$ and $2 \phi_k$, each with a margin $\beta_2$, to account for instantaneous pitch halving and doubling. To avoid spurious octave jumps, we penalize such jumps by a multiplicative factor $\gamma$. In general we can define a matrix of penalty values $\gamma[i, j]$ for jumps from pitch $\varphi_i$ to $\varphi_j$, and find a path $\{i\}_k$ that maximizes

$$f(\{i\}) = C[1, i_1] + \sum_{k=2}^n \gamma[i_{k-1}, i_k] \cdot C[k, i_k].$$  

In this case we set $\gamma[i, j] = +\infty$ to all jumps for $i$ to $j$ that are infeasible. For simplicity we choose $\gamma[i, j] = 1$ for all pitch changes that are within a factor of $\beta_1$ relative to the current pitch values, but other penalties would be possible as well. Minimizing the objective in (2) consists of a relatively straightforward forward pass over the data followed by a reverse pass to extract the optimal path.

Some comments regarding the choice of $\beta_1$ and $\beta_2$ are in place here. First, choosing $\beta_1$ too small limits the ability of the algorithm to track rapidly changing pitch values and causes estimates that trail behind. On the other hand, choosing $\beta_1$ values that are too large decreases the smoothness of the trajectory and allows the inclusion of more outliers. Second, it was observed that pitch halving and doubling tends to happen in regions where the pitch is otherwise fairly constant. This suggests that we can restrict ourselves to a much more limited search region, and consequently use a value for $\beta_2$ that is much smaller than $\beta_1$. The values $\beta_1 = 7\%$ (175.7 semitones per second) and $\beta_2 = 1\%$ (25.8 st/s) worked well for our experiments, even though $\beta_1$ exceeds the maximum speed of pitch changes reported in [10].

For some applications, such as feature augmentation in tonal languages, it may be undesirable to report instantaneous halving or doubling in the pitch. This could be dealt with either by limiting the set of feasible steps and allowing only limited pitch changes around the current value, or by adjusting pitch values in regions where pitch halving or doubling is detected.

2.3. Post processing

During the post-processing phase refinements and corrections to the pitch estimates can be made. Perhaps the most important type of correction is the exclusion of estimated pitch values in unvoiced regions. This requires an additional phase of estimating whether a given frame is voiced or unvoiced. The easiest way to do this is by checking whether the coefficients corresponding to the pitch values are above a given threshold after volume normalization. Although the reuse of the correlation coefficients can help save time, there is an inherent price to doing so. For accurate determination of the pitch values we need to use high-resolution spectral information, which corresponds...
to large window sizes and therefore leads to a loss in temporal resolution. By contrast, when determining which parts of the utterance are voiced or unvoiced it is temporal resolution that we are after, which requires shorter windows. For our experiments we focus on the accuracy of the pitch estimation alone and therefore use ground-truth information to determine which regions are voiced.

Other post-processing steps could include the additional removal of very short voiced regions and individual spikes in pitch values, possibly followed by interpolation of pitch values in the small gaps formed by regions spuriously misclassified as unvoiced or by the removal of individual outliers. In our algorithm we did not include any steps of this type.

3. Numerical results

We evaluate the performance of Pitcher on the PTDB-TUG [7] database. The database consists of 2342 sentences taken from the TIMIT corpus, each recorded by at least one of the ten male and one of the ten female speakers. For each speaker PTDB-TUG records approximately 230 utterances at 48kHz along with laryngograph measurements to establish the ground-truth pitch values and voice classification. We compare the performance of Pitcher with YIN [3, 4], IRAPT [1], and Swipe' [2]. For IRAPT we tried both the IRAPT1 and IRAPT2 variations but found little difference in performance except that IRAPT2 was significantly slower. Likewise, we also tried the base version of Swipe, in which the filters contain peaks at every multiple of the pitch frequency instead of only at the first and prime-number multiples, but no big difference was found. We therefore report on the IRAPT1 and Swipe’ versions of these algorithms. All solvers are used with their default parameters.

We look at the percentage of gross outliers (deviations of more than 20% relative to the ground-truth pitch); the average percentage deviation in pitch estimates for non-outliers; the percentage of gross outliers when pitch halving and doubling errors are corrected by an oracle; and the runtime of the algorithms. The results are summarized in Table 1. The runtime is reported only for indication only and are not directly comparable. In particular, the results reported for YIN are based on the default time shift of 1ms. Because the other methods have much larger time shifts we decided to report the runtime based on a time shift of 6.7ms second. Running YIN with this larger time shift led to worse results, so, aside from runtime, we report the results obtained with default settings. The time shifts for IRAPT and Pitcher are 5ms and 6.7ms respectively. Swipe’ has nonuniform time shifts in that it computes a series of spectrograms each with different window lengths but with the fraction of window overlap fixed. The results in the database are provided at 10ms intervals.

There is a marked difference in the relative performance of the algorithms between female (high-pitch) and male (low-pitch) speakers, and we therefore discuss these results in turn. Starting with the results for female speakers on the left half of Table 1, we see that there is no uniformly best method. Swipe’ and Pitcher have the most accurate pitch estimates and also have the lowest number of gross outliers for all but one speaker (F6). Comparing the entries in the first and third row of each block, we see that most of the gross errors in IRAPT are caused by octave misclassifications and it thus follows that IRAPT has the highest potential if pitch halving and doubling errors could be corrected. Overall, the performance of IRAPT is slightly better than YIN, but neither method compares favorably with Swipe’ or Pitcher. Moving on to the results on male speakers in the right half of Table 1, we see that that Pitcher decisively outperforms the other methods. The gross error rate of Pitcher is uniformly lower and the non-outlier pitch estimates are more accurate for all but three speakers. When looking at the potential of the methods with octave errors corrected, we see that Pitcher outperforms the other methods. Moreover, even when single octave errors could be corrected, YIN and IRAPT generally still trail behind Pitcher. The results by Swipe’ are better than the former two, but no longer match those of Pitcher. In addition, the computation of various spectrograms in Swipe’ (typically four) means that the method is substantially slower than the other three. Finally, from visual inspection of the data it seems that the large percentage of outliers for some speakers in the database is caused by classification errors in the ground truth, rather than by the methods.

4. Discussion

The resolution of the spectrogram has a notable effect on the performance of Pitcher. In particular, it was found that the accuracy of the pitch estimates improved substantially when increasing the window size from 2048 to 4096, especially for female speakers. Further increasing the window size did not improve the results; in fact, doing so slightly deteriorated the performance. As mentioned in Section 2.1, it typically suffices to work with only part of the spectrum. For the experiments with Pitcher reported in Table 1 we worked with a dictionary of size 350 × 3500, instead of the full 2048 × 3500. That is, only the intensity values for the lowest 350 rows in the spectrum are used when computing the correlation coefficients C. The number of columns, 3500, indicates the granularity in the pitch values, which in this case were chosen log-linearly between 50Hz and 400Hz. Increasing the resolution can help improve the accuracy of the non-outlier pitch estimates, but also leads to a quadratic increase in the computational cost of the dynamic programming (both the number of coefficients, and the search window size increase linearly). As mentioned above, we allow 7% relative jumps in pitch, along with 1% jumps relative to single octave shifts up or down. Decreasing the 7% relative jumps much further led to an increase in the numbers of gross outliers and a lower relative accuracy caused by failure to keep up with rapidly changing pitch values. We imposed a moderate penalty value for octave shifts (γ = 0.95) to help avoid rapid switching back and forth between octaves.

For the best performance it is important that the waveform atoms for a given pitch correlate well with the corresponding features in the spectrum. One possible way of improving performance is to use the smooth and slowly varying component of the intensity values in the spectrogram for each given time instance, and correlate only with the oscillating part. In conjunction, it may also be possible to improve the waveform atoms by learning them from training data. As a preliminary experiment we generated a grid of pitch values 1/4Hz apart and averaged the vectors in the spectrogram corresponding to each of the bins for all utterances in the database. From this we generated a synthetic model in which the smooth component of the spectral features is interpolated over the entire range of pitch values based on samples at a small number of pitch values. Likewise, we estimate the amplitude envelope of the oscillatory part and fitted waveforms of the form

\[ w_{\omega}(x) = \frac{2 \times (\exp(\alpha(\omega) \cos(x)) - \exp(-\alpha(\omega)))}{\exp(\alpha(\omega)) - \exp(-\alpha(\omega))} \times 1, \]

where \( \alpha(\omega) \) is based on a smooth approximation of the best fit
Table 1: Performance of the different algorithms on ten female (left) and ten male (right) speakers from PTDB-TUG. The numbers below each speaker indicate the median pitch level and the duration of all the utterances, including silence. Per block we record from top to bottom (1) The percentage gross outliers, defined as pitch values that differ more than 20% from the ground truth; (2) the average relative difference in percent in pitch for non-outlier estimates; (3) the percentage of gross outliers if single octave corrections are provided by an oracle; and (4) the runtime of the different methods (see main text for an explanation of the runtime for YIN).

<table>
<thead>
<tr>
<th>Method</th>
<th>F01</th>
<th>F02</th>
<th>F03</th>
<th>F04</th>
<th>F05</th>
<th>F06</th>
<th>F07</th>
<th>F08</th>
<th>F09</th>
<th>F10</th>
<th>F11</th>
<th>F12</th>
<th>F13</th>
<th>F14</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Yin</strong></td>
<td>2.675</td>
<td>0.985</td>
<td>4.026</td>
<td>9.243</td>
<td>4.801</td>
<td>5.728</td>
<td>2.354</td>
<td>0.718</td>
<td>2.095</td>
<td>17.321</td>
<td>1.357</td>
<td>1.111</td>
<td>5.983</td>
<td>3.577</td>
</tr>
<tr>
<td><strong>IRAPT</strong></td>
<td>2.447</td>
<td>0.664</td>
<td>2.218</td>
<td>8.258</td>
<td>5.295</td>
<td>9.538</td>
<td>2.155</td>
<td>0.831</td>
<td>1.729</td>
<td>17.115</td>
<td>1.345</td>
<td>1.068</td>
<td>5.938</td>
<td>3.579</td>
</tr>
<tr>
<td><strong>Swipe'</strong></td>
<td>2.159</td>
<td>0.738</td>
<td>2.111</td>
<td>7.720</td>
<td>2.111</td>
<td>9.323</td>
<td>2.372</td>
<td>0.663</td>
<td>2.216</td>
<td>16.127</td>
<td>1.205</td>
<td>1.166</td>
<td>5.969</td>
<td>3.601</td>
</tr>
<tr>
<td><strong>Pitcher</strong></td>
<td>2.338</td>
<td>0.697</td>
<td>2.250</td>
<td>7.184</td>
<td>2.250</td>
<td>8.967</td>
<td>2.236</td>
<td>0.612</td>
<td>2.216</td>
<td>16.101</td>
<td>1.418</td>
<td>1.206</td>
<td>5.969</td>
<td>3.599</td>
</tr>
<tr>
<td><strong>Synth</strong></td>
<td>3.420</td>
<td>0.845</td>
<td>2.908</td>
<td>8.128</td>
<td>3.850</td>
<td>8.637</td>
<td>2.536</td>
<td>0.560</td>
<td>2.533</td>
<td>13.092</td>
<td>2.809</td>
<td>2.800</td>
<td>7.375</td>
<td>2.344</td>
</tr>
</tbody>
</table>

6. References


