A Fast and Accurate Zebra Finch Syllable Detector

Ben Pearre[1,a,¤], L. Nathan Perkins[1,b], Jeffrey E. Markowitz[2,c], Timothy J. Gardner[1,d]

[1] Department of Biology, Boston University, Boston, Massachusetts, United States of America

[2] Department of Neurobiology, Harvard Medical School, Boston, Massachusetts, United States of America

Author [a] contributed the original MATLAB and LabVIEW implementations and most of the manuscript. [b] contributed the Swift implementation and its description herein, helped debug the training code, and measured the timing for most of the detectors. [c] performed the fibre photometry experiment, wrote several types of filter-bank software against which we compared, made suggestions for the neural network detector, and reformatted the MATLAB code for distribution. [d] provided a stream of suggested techniques, metrics, and goals.

[¤] Corresponding author: bwpearre@gmail.com (BP)

Abstract

The song of the adult male zebra finch is strikingly stereotyped. Efforts to understand motor output, pattern generation, and learning have taken advantage of this consistency by investigating the bird's ability to modify specific parts of song under external cues, and by examining timing relationships between neural activity and vocal output. Such experiments require that precise moments during song be identified in real time as the bird sings. Various syllable-detection methods exist, but many require special hardware, software, and know-how, and details on their implementation and performance are scarce. We present an accurate, versatile, and fast syllable detector that can control hardware at precisely timed moments during zebra finch song. Many moments during song can be isolated and detected with false negative and false positive rates well

under 1% and 0.001% respectively. The detector can run on a stock Mac Mini with triggering delay of less than a millisecond and a jitter of $\sigma \approx 2$ milliseconds.

1   Introduction   1

The adult zebra finch (*Taeniopygia guttata*) sings a song made up of 2–6 syllables,   2
with longer songs taking on the order of a second. The song may be repeated   3
hundreds of times per day, is almost identical each time, and several brain areas reflect   4
this consistency in highly stereotyped neural firing patterns. This consistency makes   5
the zebra finch one of the most popular models for the study of the neural basis of   6
learning, audition, and control.   7

This consistency allows a variety of experiments if precise moments in song can   8
reliably be detected quickly enough to trigger other apparatus during singing. A   9
common area of study with this song-triggering technique is the anterior forebrain   10
pathway (AFP), a homologue of mammalian basal ganglia consisting of a few distinct   11
brain areas concerned with the acquisition and learning of song. For example, [1]   12
stimulated the lateral magnocellular nucleus of the anterior nidopallium (LMAN)—the   13
output nucleus of the AFP—at precisely timed moments during song and showed that   14
this area repeatably controls specific variables in song output. [2] stimulated LMAN   15
and the high vocal centre (HVC) in one hemisphere or the other, and found that   16
control of song rapidly switches between hemispheres. [3] investigated the response of   17
cells in field L and the caudolateral mesopallium during song with and without   18
feedback perturbations as well as external sounds, and concluded that these regions   19
may hold a representation of song against which auditory signals are compared. [4]   20
played a disruptive sound during renditions of a syllable that were slightly above (or   21
below) its average pitch, and showed that the presence of this apparently random   22
natural variability in songbird motor output is used to drive change in the song. [5]   23
extended this experiment and showed that AFP produces a corrective signal to bias   24
song away from those disruptions. [6] showed that the song change was isolated to   25
within roughly 10 milliseconds (ms) of the stimulus, and further proposed an elegant   26
model that predicts the shape of the learned response and suggests a simple   27
mechanism. [7] showed that that AFP transfers the error signal to the robust nucleus   28

of the arcopallium (RA) using NMDA-receptor–mediated glutamatergic transmission. 29
By looking at song recovery after applying the same pitch-shift paradigm, [8] showed 30
that the caudal medial nidopallium is implicated in memorising or recalling a recent 31
song target, but for neither auditory processing nor directed motor learning. 32

Despite the power and versatility of such experiments, there is no standard syllable 33
detector. Considerations include: 34

Accuracy: How often does the system produce false positives or false negatives? 35

Latency: The average delay between the target syllable being sung and the detection. 36

Jitter: The amount that latency changes from instance to instance of song. Our 37
measure of jitter is the standard deviation σ of latency. 38

Versatility: Is detection possible at "difficult" syllables? 39

Ease of use: How automated is the process of programming a detector? 40

Cost: What are the hardware and software requirements? 41

A variety of syllable-triggering systems have been used, but few have been 42
documented or characterised in detail. In 1999, [9] used groups of IIR filters with 43
hand-tuned logical operators. Their system had a latency of 50 or 100 ms, and they do 44
not report on jitter or accuracy. As access to computational resources has improved, 45
approaches have changed: in 2009, [5] still used hand-tuned filters, but ran them on a 46
Tucker-Davis Technologies digital signal processor. They report a latency of around 4 47
ms, but as with other filter-bank techniques, it is not strictly a syllable detector but 48
rather a pitch and timbre detector—it cannot identify a frequency sweep, or 49
distinguish a short chirp from a long one—and thus requires careful selection of target 50
syllables. Furthermore, the method is neither inexpensive nor, based on our experience 51
with a similar technique, accurate. [3] applied a neural network to a spectral image of 52
song. They report a jitter of 4.3 ms, but further implementation and performance 53
details are not available. In 2011, [7] matched spectral templates to stable portions of 54
syllables in 8-ms segments. They report a jitter of 4.5 ms, and false-negative and 55
false-positive rates of up to 2% and 4%, respectively. Hardware requirements and ease 56
of use were not reported. In 2013, [10] described in detail a system that matches 57

spectral images of template syllables using a correlation coefficient. With a fast ₅₈
desktop (Intel i7 six-core) running Linux and equipped with a National Instruments ₅₉
data acquisition card, it boasts a hardware-only (not accounting for the time taken to ₆₀
compute a match with a syllable) latency and jitter of just a few microseconds, and ₆₁
the detection computation they use should not much increase that. Drawbacks are ₆₂
that some hand-tuning may still be required, and that they report false-negative rates ₆₃
around 4% and 7% for zebra finches and Bengalese finches respectively, measured on a ₆₄
small dataset. In much other work, an allusion is made to a syllable detector, but no ₆₅
further information is provided. ₆₆

We developed a standalone detector that learns to match moments in the song ₆₇
using a neural network applied to the song spectrogram, and outputs a TTL pulse (a ₆₈
brief 5-volt pulse) at the chosen moment. The approach consists of three steps: ₆₉

1. Record and align a corpus of training songs. The technique has been published ₇₀
   in [11]. ₇₁

2. Choose one or more instants in the song that should create a trigger event, and ₇₂
   train a neural network to recognise them. This step is carried out offline. While ₇₃
   any neural network software would produce equivalent results, we used ₇₄
   MATLAB 2015b's neural network toolbox. ₇₅

3. Once trained and saved, the neural network is used by a realtime detection ₇₆
   program that listens to an audio signal and indicates detection of the target ₇₇
   syllables via a TTL pulse. We present three implementations, in MATLAB, ₇₈
   LabVIEW, and Swift, that trade off hardware requirements, ease of maintenance, ₇₉
   and performance. ₈₀

This method makes the following contributions: ₈₁

- Fast: sub-millisecond latencies, with jitter around 2 ms. ₈₂

- Accurate: false positive rates under 0.001% and false negative rates under 1% for ₈₃
  a variety of syllables. Balancing these rates depends on a single relative-cost ₈₄
  parameter. ₈₅

- State-of-the-art performance with default parameters. ₈₆

- Runs on inexpensive hardware. 87

- Described in detail here, with reference implementations provided and 88
benchmarked. 89

2 Materials and Methods 90

2.1 Learning a detector 91

We begin with a few hundred recordings of a given bird's song, as well as calls, cage 92
noise, and other non-song audio data. Male zebra finch song is most highly 93
stereotyped when a female is present ("directed song") and slightly more variable 94
otherwise ("undirected"); we train and test on undirected song since this is more 95
challenging. Recordings were made inside a sound-isolating chamber in which was 96
mounted a recording microphone (Audio-Technica AT803). The songs were 97
time-aligned as described in [11]. 98

We rely on two circular buffers: 99

Audio buffer: This contains the most recent audio samples, and is of the length 100
required to compute the Fast Fourier Transform (FFT)—usually 256 samples. 101

FFT buffer: The results of each new FFT are placed here. It contains the n_{fft} most 102
recent FFTs, which will serve as inputs to the neural network (described below). 103

Audio is recorded at some sample rate $1/t_{\text{sample}}$ (for example, 44.1 kHz), and new 104
data are appended to the circular audio buffer. 105

The spectrogram is computed at regular intervals—the useful range seems to be 106
roughly 1–5 milliseconds, which we refer to as the frame interval t_{fft}. At each frame, a 107
spectrum is computed from the most recent 256 audio samples in the audio buffer, and 108
the result is appended to the FFT buffer. For example, if $t_{\text{fft}} = 1$ ms and the recording 109
sample rate $1/t_{\text{sample}} = 40$ kHz, then in order to compute a new FFT, 110
$40000 \cdot 0.001 = 40$ new audio samples must be appended to the audio buffer, the FFT is 111
computed using the most recent 256 samples in that buffer, and the result is appended 112
to the FFT buffer. Over time, the successive spectra will look something like Fig. 1. 113
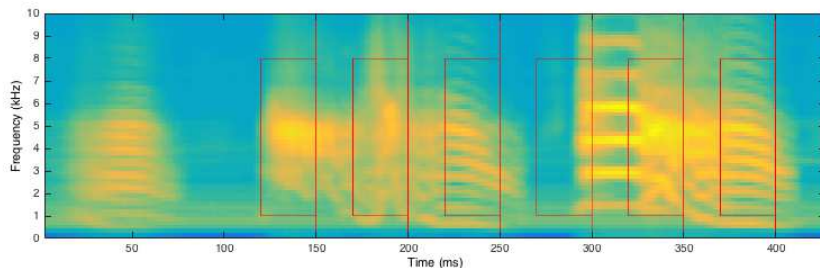
Figure 1. This image was made by superposing the spectra of our 2818 aligned songs. Our example detection points, $t_1^* \ldots t_6^*$, are shown as red lines, with the recognition regions (30 ms × 1–8 kHz) marked as rectangles.

Note that this results in time being discretised into chunks of length t_{fft}. Because each Fourier transform computation contains a small number n_{fft} of new audio samples (in the example above, 40 new samples and $256 - 40$ that have already been examined), we tried implementing the Sliding Discrete Fourier transform (SDFT) [12]. This allows $t_{\text{fft}} = t_{\text{sample}}$. Practically, the operating system retrieves new audio samples from the hardware several at a time, so the full benefit of the SDFT is difficult to see in practice. Furthermore, we found that FFT implementations are sufficiently highly optimised that discretising time into chunks of t_{fft} as we have done produced similar results with simpler software.

When using consumer audio hardware that operates best at 44.1 kHz, the requested frame interval may not line up with the sample rate, so the actual frame interval may be different from the intended. For example, at 44.1 kHz a 1-ms frame interval requires a new FFT every 44.1 samples. This must be rounded to 44 samples, resulting in $t_{\text{fft}} = \lfloor 44.1 \rfloor / 44.1 \approx 0.9977$ ms.

One or more target moments during the song must be chosen. Our interface presents the time-aligned spectrogram averaged over training songs, and requires manual input of the target times, t^*. Then we assemble the training set from the song data, train the network, compute optimal output unit thresholds, and save the network object and an audio test file.

Recognition region   The neural network's inputs are the FFT values from a rectangular region of the spectrogram covering a predefined range of frequency values F (for example, 1–8 kHz) at some number of the most recent frames n_{fft}. Any time t falls within frame $\tau(t)$, and $t - t_{\text{fft}}$ falls within the previous frame, so the recognition

region that the neural network receives as input consists of the spectrogram values 137
over F at $\tau(t)$ and those from the contiguous set of recent frames: $T = \{ \tau(t), 138
\tau(t - t_{\text{fft}}), \tau(t - 2t_{\text{fft}}) \ldots \tau(t - n_{\text{fft}} t_{\text{fft}}) \}$. We have found that a 30-ms time 139
window—which will yield $n_{\text{fft}} = |T| = \lfloor 30 \text{ ms}/t_{\text{fft}} \rfloor$ frames—of frequencies spanning 140
1–8 kHz works well for most syllables. 141

Six examples of chosen target moments in the song, and their corresponding 142
recognition regions F and T, are shown in Fig. 1. 143

Building the training set   The training set is created in a fashion typical for 144
neural networks: at each time frame t the rectangular $|F| \times |T|$ recognition region in 145
the spectrogram as of time t is reshaped into a vector ξ_t, which will have length $|F||T|$ 146
and contain the spectra in F taken at all of the times in the set T: from $\tau(t - n_{\text{fft}} t_{\text{fft}})$ 147
to $\tau(t)$. These vectors are placed into a training matrix, Ξ, such that each column ξ_t 148
holds the contents of the recognition region—containing multiple frames from the 149
spectrogram—as of one value of t. 150

Training targets y_t are vectors with one element for each desired detection syllable. 151
That element is, roughly, 1 if the input vector comes from the target time ($t = t^*$), 0 152
otherwise, for each target syllable (of which there may be any number, although they 153
increase training time, and in our implementations the number of distinct output 154
pulses is constrained by hardware). Since the song alignment may not be perfect, and 155
due to sample aliasing, a strict binary target may ask the network to learn that, of two 156
practically identical frames, one should be a match and the other not. Thus it is 157
preferable to spread the target in time, such that at the target moment it is 1, and at 158
neighbouring moments it is nonzero. We found that a Gaussian smoothing kernel 159
around the target time with a standard deviation of 2 ms serves well. 160

With inputs well outside the space on which a neural network has been trained, its 161
outputs will be essentially random. In order to reduce the false positive rate it is 162
necessary to provide negative training examples that include silence, cage noise, wing 163
flapping, non-song vocalisations, and perhaps songs from other birds. Although it will 164
depend on the makeup of the non-song data, we have found that training with as low 165
as a 1:1 ratio of non-song to song yields excellent results on most birds. These data 166
are also included in our accuracy tests. 167

Normalisation   In order to present consistent and meaningful inputs to the neural   168
network, we normalise the incoming data stream so that changes in the song structure   169
of the sound are emphasised over changes in volume, and to maximise the effectiveness   170
of the neural network's training algorithm.   171

The first normalisation step is designed to eliminate differences in amplitude due to   172
changes in the bird's location and other variations in recording. Each recognition   173
region vector ξ_t—during training, each *column* of the training matrix Ξ—is normalised   174
using MATLAB's `zscore` function $\hat{\xi}_t = (\xi_t - \mu_{\xi_t})/\sigma_{\xi_t}$, so that the content of each   175
window has mean $\mu_{\hat{\xi}_t} = 0$ and standard deviation $\sigma_{\hat{\xi}_t} = 1$.   176

The second step is designed to ensure that the neural network's inputs have a   177
range and distribution for which the training function can easily converge. Each   178
element i of $\hat{\xi}$ is normalised across the entire training set—each *row* of Ξ—in the same   179
way: $\check{\xi}^i = (\hat{\xi}^i - \mu_{\hat{\xi}^i})/\sigma_{\hat{\xi}^i}$, so that the values of that point across the whole training set   180
have mean $\mu_{\check{\xi}^i} = 0$ and standard deviation $\sigma_{\check{\xi}^i} = 1$. This is accomplished during   181
training by setting MATLAB's neural network toolbox normalisation scheme to   182
`mapstd`, and the scaling transform is saved as part of the network object used by the   183
realtime detector so that it may be applied to unseen data at runtime.   184

These two normalisation steps provide a set of inputs that are more robust to   185
outliers and less likely to produce false positives during silence than other   186
normalisation schemes, such as linear or L_2 normalisation.   187

Neural Networks   While any learned classifier might suffice, we chose a two-layer
feedforward neural network. In brief, our network takes an input vector ξ_t—as
described above—and produces an output vector y_t, and when any element of y_t is
above a threshold (described below), the detector reports a detection event. The
network uses two matrices of weights, W_0 and W_1, and two vectors of biases, b_0 and b_1.
The first takes the input ξ_t to an intermediate stage—the "hidden layer" vector. To
each element of this vector is applied a nonlinear squashing function such as `tanh`, and
then the second weight matrix W_1 is applied. This produces output y_t:

$$y_t = W_1 \tanh(W_0\xi_t + b_0) + b_1$$

During the network's training phase, the elements of the matrices and bias vectors are learned by back-propagation of errors over a training set. A more detailed explanation of neural networks may be found in [13].

Essentially, after training, the network is available in the form of the two matrices W_0 and W_1 and the two vectors b_0 and b_1, and running the network consists of two matrix multiplications, two vector additions and the application of the squashing function.

Training the network   The network is trained using MATLAB's neural network toolbox, with Scaled Conjugate Gradient (`trainscg`). We tried a variety of feedforward neural network geometries, from simple 1-layer perceptrons to geometries with many hidden nodes, as well as autoencoders. Perhaps surprisingly, even the former yields excellent results on many syllables, but a 2-layer perceptron with a very small hidden layer—with a unit count 2-4 times the number of target syllables—was a good compromise between accuracy and training speed. For more variable songs, deep structure-preserving networks may be more appropriate, but they are slow to train and unnecessary for zebra finch song.

Computing optimal output thresholds   After the network is trained, outputs of the network for any input are now available, and will be in (or, due to noisy inputs and imperfect training, close to) the interval $(0, 1)$. We must choose a threshold above which the output is considered a positive detection. Finding the optimal threshold requires two choices. The first is the relative cost of false negatives to false positives, C_n. The second is the acceptable time interval: if the true event occurs at time t, and the detector triggers at any time $t \pm \Delta t$, then it is considered a correct detection. Then the optimal detection threshold is the number that minimises [false positives] $+ C_n \cdot$ [false negatives] over the training set, using the definitions of false positives and negatives given in Section 2.3. Since large portions of the cost function are flat, we use a brute-force linear search, which requires fractions of a second. For the results presented here, we have used $\Delta t = 10$ ms, and arbitrarily set $C_n = 1$.

De-bouncing During runtime, the network may produce above-threshold responses $\quad$ 217
to nearby frames. Thus, after the first response, subsequent responses are suppressed $\quad$ 218
for 100 ms. However, for the accuracy figures presented here, we used the $\quad$ 219
un-de-bounced network response. $\quad$ 220

Our parameter choices We used an FFT of size 256; a Hamming window; and $\quad$ 221
chose a target spectrogram frame interval of $t_{\mathrm{fft}} = 1.5$ milliseconds, resulting in a true $\quad$ 222
frame interval of $t_{\mathrm{fft}} = \lfloor 1.5 \cdot 44.1 \rfloor / 44.1 \approx 1.4966$ ms. We set the network's input space $\quad$ 223
to 30 ms long, and to span frequencies from 1–8 kHz, which contains the fundamentals $\quad$ 224
and several overtones of most zebra finch vocalisations. $\quad$ 225

$\quad$ We found these parameters to work well across a variety of target syllables, but $\quad$ 226
various other parameter sets yield results similar to those presented here. Some of the $\quad$ 227
parameters trade off detection accuracy or temporal precision vs. training time. For $\quad$ 228
example, decreasing the frame interval generally decreases both latency and jitter, but $\quad$ 229
also increases training time. $\quad$ 230

2.2   Realtime detection $\quad$ 231

The architecture of the realtime detector requires that the most recent n_{fft} $\quad$ 232
spectrograms be fed to the neural network every frame interval. Audio samples from $\quad$ 233
the microphone are appended to the circular audio buffer. Every t_{fft} seconds a new $\quad$ 234
spectrogram is calculated by applying the Hamming window to the contents of the $\quad$ 235
buffer, performing an FFT, and extracting the power. Outputs of the spectrogram $\quad$ 236
from the target frequency band are appended to the circular FFT buffer. The $\quad$ 237
spectrograms are sent to a static implementation of the previously trained neural $\quad$ 238
network. $\quad$ 239

$\quad$ We tested three implementations of the realtime detector. For all of these tests, we $\quad$ 240
ran the detector processes under the operating systems' default schedulers and process $\quad$ 241
priorities, running typical operating system daemon processes but no user loads. The $\quad$ 242
computers had ample memory resources. $\quad$ 243

Swift This detector uses the Swift programming language and Core Audio interface $\quad$ 244
included in Apple's operating systems. $\quad$ 245

The Core Audio frameworks provide an adjustable hardware buffer size for reading ₂₄₆
from and writing to audio hardware (different from our two circular buffers). Tuning ₂₄₇
this buffer size provides a tradeoff between the jitter in the detection and the ₂₄₈
processor usage needed to run the detector. We used buffer sizes ranging from 8 ₂₄₉
samples (0.18 ms at 44.1 kHz) to 32 samples (0.7 ms at 44.1 kHz) depending on the ₂₅₀
frame size used by the detector. ₂₅₁

Vector operations—applying the Hamming window, the FFT, input normalisation, ₂₅₂
matrix multiplication, and the neural network's transfer functions—are performed ₂₅₃
using the Accelerate framework (vDSP and vecLib), which use modern vector-oriented ₂₅₄
processor instructions to perform calculations. ₂₅₅

When the neural network detects a match, it instructs the computer to generate a ₂₅₆
TTL pulse that can be used to trigger downstream hardware. This pulse can be either ₂₅₇
written to the computer's audio output buffer (again, in 8- to 32-sample chunks) or ₂₅₈
sent to a microcontroller (Arduino) via a USB serial interface. Sending the trigger ₂₅₉
pulse via the serial interface and microcontroller is noticeably faster (2.2 ms lower ₂₆₀
latency), likely due to the fact that the audio buffer goes through hardware mixing ₂₆₁
and filtering prior to output. ₂₆₂

The above code can be run on multiple channels of audio on consumer hardware ₂₆₃
(such as a 2014 Mac Mini) with little impact on CPU usage ($< 15\%$). Depending on ₂₆₄
the experimental needs, latency can potentially be further decreased (at the expense of ₂₆₅
processor usage) by adjusting the audio buffer sizes. ₂₆₆

We ran the Swift detector on a Late 2014 Mac Mini with a Intel Core i5 processor ₂₆₇
at 2.6GHz with 16 gigabytes of RAM, running Mac OS X 10.11. ₂₆₈

LabVIEW This implementation requires special software and hardware: LabVIEW ₂₆₉
from National Instruments—we used 2014 service pack 1—and a data acquisition ₂₇₀
card—we use the National Instruments PCI-6251 card on a PC with an Intel Core ₂₇₁
i5-4590 processor at 3.7GHz (a relatively low-end machine) with 32 gigabytes of RAM, ₂₇₂
running Microsoft Windows 8.1 Pro. ₂₇₃

This implementation has several drawbacks: it requires Windows and expensive ₂₇₄
hardware and software and due to the programming language it is difficult to modify ₂₇₅
and debug—indeed, a persistent bug in our implementation currently renders it ₂₇₆

substantially less accurate than the other detector implementations on some syllables. ₂₇₇
However, our test configuration usually achieved excellent performance, and further ₂₇₈
gains should be possible if the implementation were retargeted onto ₂₇₉
field-programmable gate array (FPGA) hardware—which would have the additional ₂₈₀
benefit of providing deterministic "hard realtime" guarantees—or just run on a faster ₂₈₁
desktop system. ₂₈₂

MATLAB   This detector uses the built-in audio input and output hardware on a ₂₈₃
compatible computer. We tested on a 2014 Mac Mini (the same machine used for the ₂₈₄
Swift detector described above) and 2015 Mac Pro. The Mac Pro does not have an ₂₈₅
audio input jack, so input was through an M-Audio MobilePre external USB audio ₂₈₆
interface. Despite the faster processor, the latter system did not achieve higher ₂₈₇
performance than the former, due to USB data transfer overhead. ₂₈₈

Because of how MATLAB's DSP toolbox interfaces with audio hardware, there is a ₂₈₉
double buffer both reading from and writing to audio hardware. As a result, much of ₂₉₀
the code focuses on a lightweight audio hardware interface, in order to have the ₂₉₁
smallest achievable audio buffer. To help achieve this, the MATLAB implementation ₂₉₂
spreads data acquisition and processing across two threads, due to the higher ₂₉₃
computational overhead of the interpreted programming language. ₂₉₄

The most versatile implementation, MATLAB runs on a variety of hardware and ₂₉₅
operating systems, and is perhaps the easiest to modify. While it did not perform as ₂₉₆
well on our test system as the other implementations, the convenience may outweigh ₂₉₇
the slight timing performance penalty for some experiments. Key to minimising jitter ₂₉₈
is the size of the audio buffer: on a 2014 Mac Mini running MATLAB 2015b the ₂₉₉
smallest buffer size that did not result in read overruns was about 4 ms. ₃₀₀

Similar to the Swift implementation, it is also possible to modify the MATLAB ₃₀₁
implementation to generate the TTL pulse from a microcontroller (Arduino) ₃₀₂
controlled via a serial over USB interface. This eliminates the double buffer required ₃₀₃
when sending triggers via the audio interface, reducing latency by half. ₃₀₄

2.3 Quantification

<small>305</small>

We divided the data into training and test sets. For the results reported in the <small>306</small> following section, the training set consisted of 1000 instances of the song of one bird, <small>307</small> and 2000 song-length samples of non-song (cage noise, calls, other birds' song). Our <small>308</small> test set consisted of an additional 1818 iterations of the bird's song and 3636 <small>309</small> song-length samples of non-song. <small>310</small>

The MATLAB neural network toolbox further divides our "training" set into <small>311</small> internal training, validation, and test sets. Because we do our own testing on a <small>312</small> distinct and much larger data set, we instructed the toolbox to use 80% of its training <small>313</small> set for training, and the remaining 20% for validation, with no hold-out test data. <small>314</small>

Because the data set consists of temporally aligned songs, ground truth is available <small>315</small> for each song (albeit with minor variations due to the alignment process used [11]), <small>316</small> and thus the detector can be checked by presenting the recorded training songs as well <small>317</small> as the canonical detection events. To this end, besides the trained network object, our <small>318</small> learning code produces an audio file consisting of all of the training data on the left <small>319</small> audio channel and a delta function at each aligned moment of target syllable <small>320</small> presentation on the right channel. Thus, when played on any audio player, the left <small>321</small> channel may be provided as input to the detector, and the the detector's output pulses <small>322</small> may be compared against the right channel. <small>323</small>

Accuracy   We define the accuracy of the network based on its classification <small>324</small> performance per frame. In order to avoid the apparent problem of every non-detected <small>325</small> non-syllable counting as a true negative, we also tried defining accuracy on a per-song <small>326</small> basis, such that a song without the target syllable counted as a single true negative. <small>327</small> Computing the optimal output thresholds on a per-frame basis resulted in higher <small>328</small> thresholds and thus a lower false-positive rate, with minimal consequences to the <small>329</small> false-negative rate, while also providing a valid definition of false-positive rate for data <small>330</small> streams that had not been segmented into song-sized chunks and thus allowing easy <small>331</small> interpretation of false-positive rates from calls, cage noise, etc. <small>332</small>

The accuracy as defined above is used for computing the optimal thresholds above <small>333</small> which the network's output should be interpreted as a match on the training data as <small>334</small> described in Section 2.1, for evaluation of the detectors on the training songs, and <small>335</small>

while live.   336

Timing   We evaluate the time taken from the presentation of each instance of the   337
target syllable to the firing of the detector's TTL pulse—for example, if the target   338
trigger point is at 200 ms with respect to the corpus of aligned songs, and for a given   339
presentation of the song the detector fires at 203 ms, we take the latency on that trial   340
to be 3 ms.   341

While playing the audio test file from any audio playback device, we used the TTL   342
output from the ground-truth channel of the audio output as the trigger pulse for an   343
oscilloscope, and compared it to the TTL pulse produced by the detector, which sees   344
only the birdsong channel of the audio file. For this purpose we used a pulse generator   345
(Philips PM 5715, with a listed latency of 50 ns, jitter of $\leq 0.1\%$ or 50 ps, whichever is   346
greater) to widen the detector's output spike to a number much larger than the jitter   347
(~ 100 ms). This obviates pulse length variability in the output device by essentially   348
discarding the falling edge of the output pulse. The oscilloscope is then set to   349
averaging mode (128-trigger average) in order to collect timing data. The canonical   350
signal is the trigger at $t = 0$, and the average of the detector's detection events will be   351
seen as a low-to-high transition with form approximating the cumulative probability   352
distribution function (CDF) of the detector's output in response to the chosen song   353
event, with individual detection events visible as steps in this curve.   354

Mean latency is then given as the average time between the canonical signal and   355
the corresponding detection event. It is a helpful number, but not a critical one, since   356
in zebra finch a detector with high but constant latency can often be trained to trigger   357
at a point somewhat before the true moment of interest (for example, if triggering   358
should occur at the very beginning of a syllable, the detector's time window may   359
include previous syllable). Often more important is the variability in the latency? We   360
define jitter as the standard deviation of the latency.   361

When measuring timing, it is useful to compare against a theoretical optimal, in   362
order to control for any imprecision in the song extraction and alignment of our real   363
data inherent in the method we use (described in [11]). We propose two measures   364
thereof: 365

First we test the "ideal" latency and jitter in the time shift used in calculating new   366

columns of the spectrogram. By passing a recorded audio sequence into the detector offline, and assuming no input or output latency, we compare how many additional audio samples beyond the syllable are needed before the spectrogram can match the inputs needed to trigger the neural network. This latency reflects the FFT size used for calculating the spectrogram, the FFT time shift between columns in the spectrogram, and the width of the Gaussian smoothing kernel applied to the ground truth data when training the neural network, but ignores computation times, audio buffers, and other operating system overhead.

Next we use a "δ-syllable" consisting of a δ function in the time domain, and train the network to trigger 5 ms after this pulse. This song is fed into the live detector. The results for this measurement show the latency and jitter inherent in the complete detector including audio read and write buffers, classifier computation, and FFT time shift aliasing, but excluding imprecision in the song alignment as well as detection timing effects due to differences between instances of the bird's song.

Finally, we measure latency on real bird song aligned as in [11]. This extracts and aligns the centres of the sampled songs, and the canonical signal is given with respect to that alignment. These timing measurements reflect not only detector performance, but also the variability of the bird's song.

2.4   Ethics

All procedures were approved by the Institutional Animal Care and Use Committee of Boston University (protocol number 14-029).

3   Results

We have used this detector for song alignment and feedback experiments in 5 behaving birds, and for offline analysis on a few others. For simplicity we present performance on 6 randomly chosen points in one bird's song, but the results presented here are representative of real-world performance in our lab.

Ok?

Fig. 1 shows the song for which we present accuracy and timing results. Six example target trigger points are shown, at 150, 200, 250, 300, 350, and 400 milliseconds after the beginning of the aligned samples, as indicated by the red lines,

and referred to henceforth as $t_1^* \ldots t_6^*$ respectively. The rectangles show the recognition window (30 ms, 1–8 kHz) for each instant in the song.

The first two triggers, t_1^* at 150 ms and t_2^* at 200 ms, are far from pure tones, but are closer to coloured noise. They would be difficult for a harmonic-stack filter bank to detect, and especially to pinpoint in time. t_3^* (250 ms) and t_6^* (400 ms) are rich in overtones and contain downward slides with changing timbres. t_4^* (300 ms) occurs near the beginning of a more typical harmonic stack amenable to a variety of recognition techniques, although consistently detecting a point partway through the syllable demands a detector that can use multiple time steps. t_5^* (350 ms) shows a steady pitch followed by a complex changing tone structure.

An example of the detector's output for three of these syllables is shown in Fig. 2. The total audio energy of each song is shown as a single grayscale row. By sorting according to time of each syllable's detection, the rest of the song is shown in the context of the timing for that moment. This gives an intuition of the timing variability within each song and across songs (which may be responsible for some of our measured jitter).

3.1 Accuracy

We allowed our training software to allocate our default of 4 hidden units per syllable, and computed a new FFT every 1.5 ms. Because our timing test files are designed for our stereo playback software, allowing only one channel for the ground-truth pulse, we trained one detector for each syllable for the following results. In order to eliminate the large number of variables involved in microphone, cage, playback and re-digitising, we evaluated the neural network's accuracy directly on the digitised recording. When training and runtime data are gathered on the same hardware setup, this is the digital signal that the detector will see—only the non-song data may be different.

Accuracies for our six test syllables are shown in Fig. 3 (accuracies for the synthetic δ-function songs were always 100%). For each test syllable we trained 100 different networks, which differ from each other in the random initialisation of network weights and on which random subset of our songs was used for training. Each network's accuracy is shown as a single point in each chart in Fig. 3, and the means
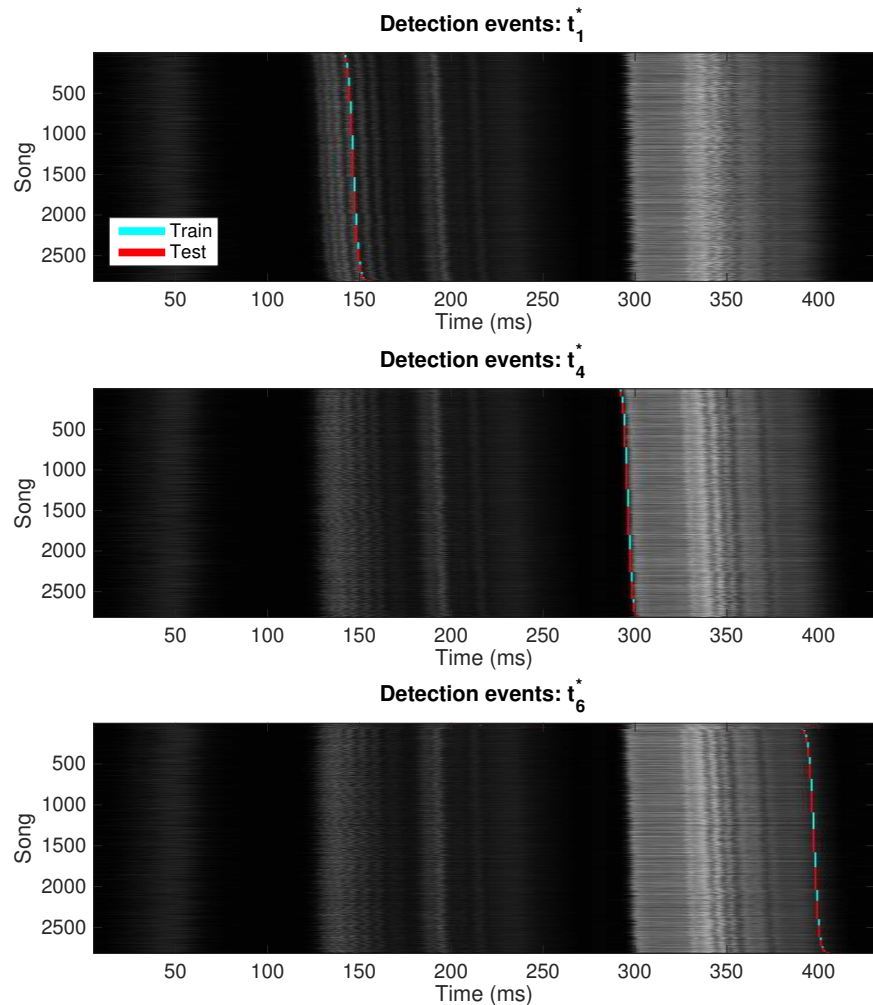
Figure 2. Each plot shows one network output unit's responses to all 2818 presentations of the song shown in Fig. 1. For simplicity, we show only the syllables t_1^*, t_4^*, and t_6^*, and we do not show the non-response to presentation of non-song. The horizontal axis is time relative to the beginning of the aligned song, and the vertical axis is an index for the 2818 individual song presentations. The grey shading shows the audio amplitude of song Y at time T. Detection events on training songs are shown in cyan, with detections of unseen test songs in red. To provide an intuition of intra-song variability, songs have been stably sorted by the time of detection events; thus, each of the three detection graphs shows the songs in a different order.
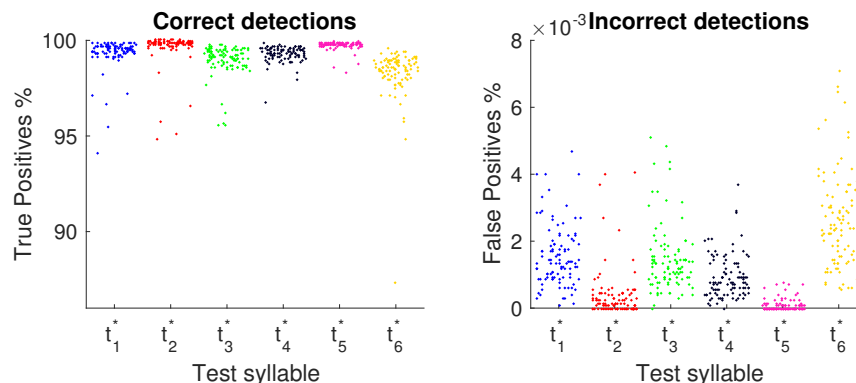
Figure 3. Accuracy variability over 100 different training runs for each of the six test syllables. Each dot shows the test-set accuracy for an independently trained detector. Because the horizontal positions have been randomised slightly so as not to occlude same-valued measurements, test syllable is also indicated by colour. The means are given in Table 1.

Target time	% True positives	% False positives
t_1^*	99.34	0.0015
t_2^*	99.65	0.00040
t_3^*	98.98	0.0015
t_4^*	99.30	0.00093
t_5^*	99.73	0.000094
t_6^*	98.31	0.0026

Table 1. Mean values for the detection accuracies shown in Fig. 3.

are shown in Table 1. Some syllables are easier to identify reliably than others, showing that detection point choice can impact the detector's accuracy. As can be seen from Fig. 3, for difficult syllables, the training process occasionally yields a network that performs unusually poorly, but it is easy to spot these networks by their performance either as measured as part of the training process or on the test audio file, and re-train when necessary.

We repeated the above experiment using only 2 hidden units per syllable. Training under this condition is much faster, but the mean true positive rate per syllable decreases by an average of 18% across our test syllables, and the mean false positive rate increases by roughly 23%.

3.2 Timing

We evaluate variability in timing performance across three variables: FFT frame interval; syllable choice; and detector implementation.
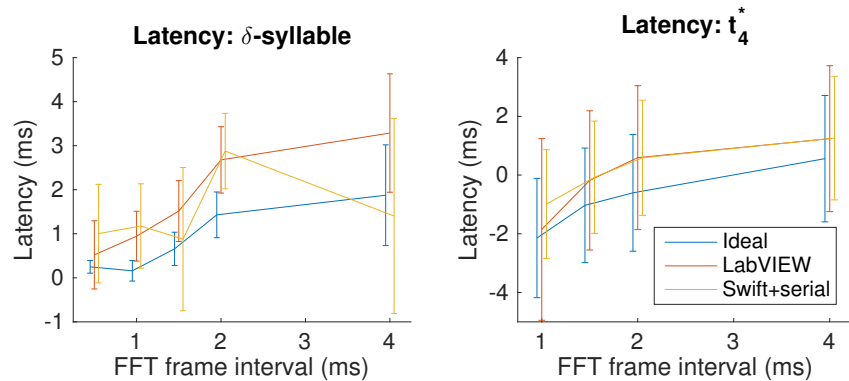
Figure 4. Timing varies as the FFT frame interval changes. Here we show results for the ideal detector and the LabVIEW and Swift+serial implementations, for the constructed δ-syllable and for trigger t_4^* of the bird song. The lines show latency; error bars are standard deviation (jitter). Points have been shifted horizontally slightly for clarity; original positions are [0.5 1 1.5 2 4] ms.

FFT frame interval   Detector latency and jitter depend on the FFT frame rate. Our 1.5-ms–frame default is a compromise: shorter frames increase the precision of timing, but also increase computational requirements both during training and at runtime. Fig. 4 shows how these numbers vary over a useful range of frame rates on our ideal detector, the Swift detector with serial output, and for the LabVIEW detector, for both the δ-syllable and for the bird song at t_4^*.

Syllable choice   Syllable choice impacts detector performance, but despite the variety of syllables chosen here, performance was fairly stable across syllables. Fig. 5 shows measured timing data for the Swift+serial detector compared to the ideal, with $t_{\text{fft}} = 1.5$ ms. Given the variety of the test syllables, the variability is surprisingly low, and is summarised here with 95% confidence intervals (with n=6):

Detector	Latency	Jitter
Ideal	-0.8 ± 0.3 ms	2.1 ± 0.1 ms
Swift+serial	0.0 ± 0.6 ms	2.0 ± 0.1 ms

The negative latency is due to the way in which the network responds to the song through time: as the recognition region looks increasingly similar to the trained match, the network's evaluation of similarity rises, and will generally cross the triggering threshold before it reaches its maximum value. A heuristic as simple as triggering at an apparent turning point after crossing the threshold might improve timing
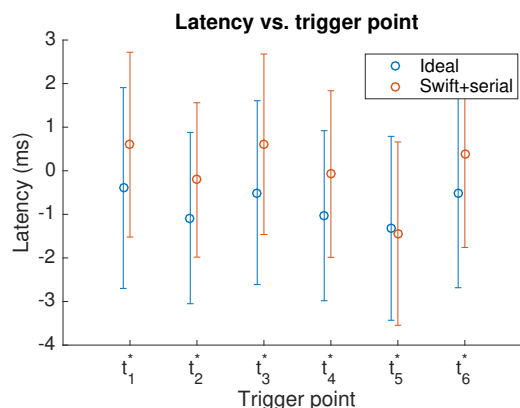
Figure 5. Timing data for our 6 test syllables, for the ideal and the Swift+serial detectors, with an FFT frame rate of 1.5 ms. Point centres show latency; error bars show jitter.

consistency at the expense of latency, but we did not test this. ₄₅₆

Detector Implementations We compared latency and jitter across our different 456 457
detector implementations for the δ-syllable and t_4^*, again with $t_{\text{fft}} = 1.5$ ms. Results 458
are as follows: 459

	δ-syllable		Bird: t_4^*	
Detector	Latency (ms)	Jitter (ms)	Latency (ms)	Jitter (ms)
Ideal	0.66	0.38	-1.0	2.0
LabVIEW	1.5	0.69	-0.18	2.4
Swift+serial	0.88	1.6	-0.075	2.0
Swift+audio	5.9	2.6	1.2	4.9
MATLAB+serial	8.1	2.3	3.8	2.0
MATLAB+audio	13	5.9	8.4	5.9

These data are also plotted in Fig. 6, and Fig. 7 gives a more detailed view of what 461
the timing curves look like for the five implementations of our detector on t_4^*. 462

4   Discussion 463

This syllable detector is appropriate for zebra finch song, and although our tests were 464
carried out on songs from that species, it is also likely to work well for Bengalese 465
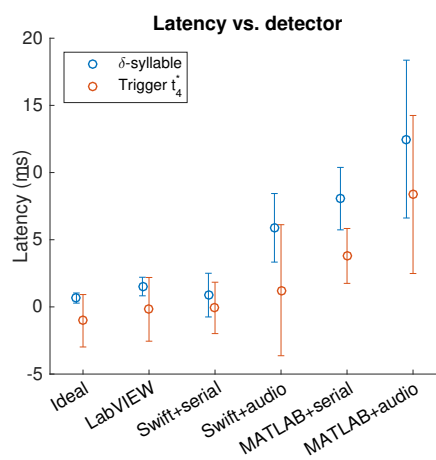finches due to their similarly stereotyped syllables. It offers the following benefits: 466

Figure 6. The different detectors for the constructed δ-syllable and for the bird song at t_4^*. Point centres show latency; error bars show jitter.
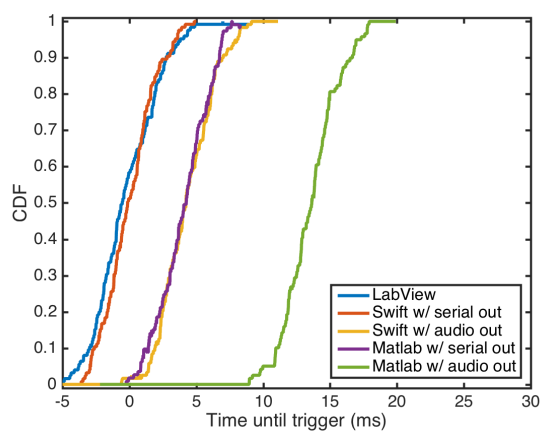

Figure 7. Raw timing curves for all detectors measured during detection of t_4^* using 1.5-ms frames. We extract the trigger events from each curve, from which we obtain the mean—latency—and standard deviation—jitter.

- The detector is accurate. False negative and false positive rates can be well under 1% and 0.001% respectively, and trading these two numbers off against each other is through a single relative-cost parameter.

- Latency is generally under a millisecond, with jitter around 2 ms.

- Works on a wide range of target syllables using the default values described here, generally eliminating the need for hand-tuning.

- Runs fast enough for syllable-modification experiments on inexpensive consumer-grade hardware, although we recommend that the training phase be run on a fast desktop system with 32 GB of RAM.

- A single detector can generate different target pulses for multiple syllables at almost no additional computational cost during runtime, although training time will increase.

Although there are differences, the Swift+serial detector and the LabVIEW implementation are roughly comparable in performance. We prefer the Swift implementation due to its lower hardware and software requirements and the difficulty of debugging LabVIEW programmes. With serial output, MATLAB's performance is good, although its buffer handling is sensitive to system load.

The song presented here was recorded with a fixed microphone mounted inside the cage. We found that higher accuracy is achieved when a microphone is instead mounted on the bird's head, which maintains volume and reduces changes in timbre as the bird moves.

A common experimental paradigm requires detecting the frequency of syllables. Many pitch detection techniques rely on the spectrum, which incurs no additional computational cost here since it is already available. For example, [8] achieved good results with the Harmonic Product Spectrum algorithm [14].

In order to monitor syllable duration, the beginning and end of a syllable may be detected by looking at the ratio of total energy in the singing frequency band to the total energy, over some small time window. Any syllable thus identified that also contains a trigger event may be monitored for duration. Alternatively, the network can be trained to recognise both the beginning and the end of the syllable of interest.

In feedback experiments such as frequency- or duration-shifting, vocal output changes over the course of the experiment. The neural network excels at identifying syllables close to its training set, so as vocal output changes the detector may not recognise a match. If the detector must be robust to this shift, it may be retrained as often as necessary as the bird learns, or data consisting of synthetically pitch-shifted or duration-shifted target syllables over the recognition region may be added to the training set. We will test these approaches in future work.

5 Acknowledgments

A Resources

If accepted, before final submission, all code will be moved to the Gardner Lab github repository, final release versions generated, and DOIs created as described in https://guides.github.com/activities/citable-code/ so that our results may be reproduced and the software deployed. For now, in lieu of final release DOIs, we provide these links to our development repositories. If you would like final versions and test datasets for review, please contact the corresponding author (bwpearre@bu.edu).

Song alignment: Jeff Markowitz's song alignment software [11] can be found at

> `https://github.com/jmarkow/zftftb`

Our implementation of the syllable detector is available under an open-source license *(we are still evaluating which open source license is the most suitable for this software):*

Training the neural network:

> `https://github.com/bwpearre/birds/align`

> **Installation:**

Clone the Github repository onto your computer. 523

Usage: 524

- The main programme is `learn_detector.m`. 525

- The top section of `learn_detector.m` contains global configuration 526
  variables. The important ones are the data directory and the bird 527
  name: as supplied, the software looks for `<data directory>/<bird` 528
  `name>/song.mat`. Modify as necessary. 529

- The format for the data file `<data directory>/<bird` 530
  `name>/song.mat` is specified in `birds/align/README.md`: a MATLAB 531
  file containing aligned song, nonsong (calls, cage noise, etc), and the 532
  sampling frequency. 533

- The configuration file `<data directory>/<bird name>/params.m` is 534
  also described in `README.md`. If the file is empty, the programme will 535
  display an image showing the average of the song spectrograms 536
  (frequency vs. time), tell the user how to define syllables of interest, 537
  and exit. If `times_of_interest` is defined, the programme will train 538
  the neural network. `params.m` can also override any of the 539
  configuration parameters at the top of `learn_detector.m`. 540

- The software will produce a neural network file, 541
  `detector_bird_times_....mat` (the ellipses signify other information 542
  encoded in the filename). It will also produce the audio test file 543
  described in Methods. 544

Runtime: 545

Jeff: Can you flesh this out?

MATLAB: `https://github.com/gardner-lab/syllable-detector-matlab` 546

Installation: 547

Clone the Github repository onto your computer. 548

Connection: 549

Usage: 550

Swift: `https://github.com/gardner-lab/syllable-detector-swift` 551

Installation: 552

Download the most recent release from the Github repository onto your   553
computer.   554

Connection: 555

- Either connect the microphone directly into the microphone jack or   556
  through a preamp into the audio input jack. If the input jack   557
  supports multiple channels (e.g., audio input jacks are usually   558
  stereo), you can connect multiple audio sources and run different   559
  detectors on each channel simultaneously.   560

- To generate TTL signals via the audio output, use the headphone   561
  jack. The audio output must have at least the same number of   562
  channels as the input.   563

- To generate TTL signals via an Arduino, load the MATLAB   564
  arduino IO sketch onto the Arduino (the sketch is available in the   565
  Github repository, and provides a simple serial protocol for   566
  controlling pins). Connect the Arduino to the computer via USB.   567
  The TTL signal for the first input will be available on pin 7, the   568
  second input on pin 8, etc.   569

Usage: 570

- Use `convert_to_text.m` to convert the Matlab output file to a   571
  format easily readable by Swift. This will generate a text version of   572
  the neural network.   573

- Launch the application downloaded above. It will provide an menu   574
  to select an audio input and either an audio output or an Arduino   575
  output (if detected). Once you select an input and an output, a   576
  new window will show all audio channels associated with the input.   577
  For each channel, select a the text-encoded version of the detector   578
  created in the previous step. Once configured, press run to begin   579
  monitoring the inputs.   580

Customization: 581

- You can clone the Github repository and use Xcode to edit the   582

project, if you are interested in extending or modifying the 583
functionality. To hook into either the raw input or output, look at 584
the "ViewControllerProcessor.swift" file. The `Processor` class acts 585
as a coordinator, receiving income audio, passing that to the 586
detector and generating appropriate output TTL signals. 587

LabVIEW: `https://github.com/bwpearre/birds/align` 588

Installation: 589

Clone the Github repository onto your computer. 590

Connection: 591

- The microphone should be connected through a preamplifier to the 592
data acquisition card, by default on Analog Input 0 (AI0). 593
- Outputs are on the Digital I/O (DIO) channels. See the block 594
diagram of `nnfft.vi` for documentation, or just connect to DIO0. 595

Usage: Use LabVIEW to open `nnfft.vi`. A file chooser is provided in the 596
front panel. Use it to point to the network file produced by 597
`learn_detector.m`, and use LabView's "Run" button to begin 598
detection. 599

References 600

1. Kao MH, Doupe AJ, Brainard MS. Contributions of an avian basal 601
ganglia–forebrain circuit to real-time modulation of song. Letters to Nature. 602
2005 February;433:638–643. Available from: 603
`http://www.nature.com/nature/journal/v433/n7026/abs/nature03127.html`. 604

2. Wang CZH, Herbst JA, Keller GB, Hahnloser RHR. Rapid Interhemispheric 605
Switching during Vo- 606
cal Production in a Songbird. PLOS Biology. 2008 October;6(10). Available from: 607
`http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.00602` 608

3. Keller GB, Hahnloser RHR. Neural processing of auditory feedback during 609
vocal practice in a songbird. Nature. 2009 January;457:187–190. Available from: 610
`http://www.nature.com/nature/journal/v457/n7226/abs/nature07467.html`. 611

4. Tumer EC, Brainard MS. Performance variability enables adaptive plasticity of 'crystallized' adult birdsong. Nature. 2007 December;450:1240–1244. Available from: http://www.nature.com/nature/journal/v450/n7173/abs/nature06390.html.

5. Andalman AS, Fee MS. A basal ganglia-forebrain circuit in the songbird biases motor output to avoid vocal errors. Proceedings of the National Academy of Sciences of the United States of America. 2009 July;106(30):12518–12523. Available from: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2709669/.

6. Charlesworth JD, Tumer EC, Warren TL, Brainard MS. Learning the microstructure of successful behavior. Nature Neuroscience. 2011;14(3):373–380. Available from: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3045469/.

7. Warren TL, Tumer EC, Charlesworth JD, Brainard MS. Mechanisms and time course of vocal learning and consolidation in the adult songbird. Journal of Neurophysiology. 2011 October;106(4):1806–1821. Available from: http://jn.physiology.org/content/106/4/1806.full.

8. Canopoli A, Herbst JA, Hahnloser RHR. A Higher Sensory Brain Region Is Involved in Reversing Reinforcement-Induced Vocal Changes in a Songbird. The Journal of Neuroscience. 2014;34(20):7018–7026. Available from: http://www.jneurosci.org/content/34/20/7018.full.

9. Leonardo A, Konishi M. Decrystallization of adult birdsong by perturbation of auditory feedback. Nature. 1999 June;399:466–470. Available from: http://www.nature.com/nature/journal/v399/n6735/abs/399466a0.html.

10. Skocik M, Kozhevnikov A. Real-time system for studies of the effects of acoustic feedback on animal vocalizations. Frontiers in Neural Circuits. 2013 January;6(111). Available from: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3539774/.

11. Poole B, Markowitz JE, Gardner TJ. The Song Must Go On: Resilience of the Songbird Vocal Motor Pathway. PLOS One. 2012 June;7(6). Available from: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0038173.

12. Jacobsen E, Lyons R. The sliding DFT. IEEE Signal Processing Magazine. 2003   641
    Mar;20(2):74–80. Available from: 642
    http://www.cmlab.csie.ntu.edu.tw/DSPCourse/reference/Sliding%20DFT.pdf643

13. Hertz JA, Krogh AS, Palmer RG. Introduction to the Theory of Neural   644
    Computation. Perseus Books; 1991.   645

14. Noll AM. Pitch determination of human speech by the harmonic product   646
    spectrum, the harmonic sum spectrum, and a maximum likelihood estimate. In: 647
    Proceedings of the symposium on computer processing in communications.   648
    vol. 19; 1970. p. 779–797.   649