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] 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~~ bu.edu (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~~0.005% 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

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

~~This consistency allows a variety of experiments if~~ If precise moments in song can reliably be detected quickly enough to trigger other apparatus during singing, then this consistency of behaviour allows a variety of experiments. A common area of study with ~~this song-triggering technique~~ song-triggered experiments is the anterior forebrain pathway (AFP), a homologue of mammalian basal ganglia consisting of a few distinct brain areas concerned with the learning and production of song. For example, ~~[1] stimulated~~ stimulation of the lateral magnocellular nucleus of the anterior nidopallium (LMAN)—the output nucleus of the AFP—at precisely timed moments during song ~~and~~ showed that this area ~~repeatably~~ controls specific variables in song output ~~. [2] stimulated~~ [1]. Song-synchronised stimulation of LMAN and the high vocal centre (HVC) in one hemisphere or the other ~~, and found~~ showed that control of song rapidly switches between hemispheres ~~. [3] investigated the response of cells in field~~ [2]. Feedback experiments have shown that Field L and the caudolateral mesopallium ~~during song with and without feedback perturbations as well as external sounds, and concluded that these regions~~ may hold a representation of song against which auditory signals are compared ~~. [4] played a disruptive sound during~~ [3]. The disruption by white noise of renditions of a syllable that were slightly above (or below) ~~its average pitch, and~~ the syllable's average pitch showed that the ~~presence of this~~ apparently random natural variability in songbird motor output is used to drive change in the song ~~. [5] extended this experiment and showed that~~ [4], and AFP produces a corrective signal to bias song away from those disruptions ~~. [6] showed~~

[5]. The song change is isolated to within roughly 10 milliseconds (ms) of the stimulus, and the shape of the learned response can be predicted by a simple mechanism [6]. AFP transfers the error signal to the robust nucleus of the arcopallium (RA) using NMDA-receptor–mediated glutamatergic transmission [7]. The course of song recovery after applying such a pitch-shift paradigm showed that the caudal medial nidopallium is implicated in memorising or recalling a recent song target, but in neither auditory processing nor directed motor learning [8].

Despite the power and versatility of vocal feedback experiments, there is no standard syllable detector. Desiderata for such a detector include:

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

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

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

Versatility: Is detection possible at "difficult" syllables?

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

Cost: What are the hardware and software requirements?

A variety of syllable-triggering systems have been used, but few have been documented or characterised in detail. In 1999, detection was achieved by a group of IIR filters with hand-tuned logical operators [9]. The system had a latency of 50 or 100 ms, and accuracy and jitter were not reported. As access to computational resources has improved, approaches have changed: in 2009, hand-tuned filters were implemented on a Tucker-Davis Technologies digital signal processor, bringing latency down to around 4 ms [5]. But as with other filter-bank techniques, it is not strictly a syllable detector but rather a pitch and timbre detector—it cannot identify a frequency sweep, or distinguish a short chirp from a long one—and thus requires careful selection of target syllables. Furthermore,

the method is neither inexpensive nor, based on our experience with a similar technique, accurate. 2009 saw the application of a neural network to a spectral image of song [3]. They reported a jitter of 4.3 ms, but further implementation and performance details are not available. In 2011, stable portions of syllables were matched to spectral templates in 8-ms segments [7]. This detector achieved a jitter of 4.5 ms, and false-negative and false-positive rates of up to 2% and 4% respectively. Hardware requirements and ease of use were not reported. In 2013, spectral images of template syllables were compared to song using a correlation coefficient [10]. With a fast desktop (Intel i7 six-core) running Linux and equipped with a National Instruments data acquisition card, it boasts a hardware-only (not accounting for the time taken to compute a match with a syllable) latency and jitter of just a few microseconds, and the detection computation they use should not much increase that. They reported false-negative rates around 4% and 7% for zebra finches and Bengalese finches respectively, measured on a small dataset. In much other work, a syllable detector is alluded to, but not described.

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

1. Record and align a corpus of training songs. The technique has been published in [11]. As few as 200 songs can yield acceptable results, but here we standardise on 1000-song training sets.

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

3. Once trained and saved, the neural network is used by a realtime detection programme that listens to an audio signal and indicates detection of the target syllables via a TTL pulse. We present three detection implementations, in MATLAB, LabVIEW, and Swift, that trade off hardware

requirements, ease of maintenance, and performance.   89

This method makes the following contributions: 90

- Fast: sub-millisecond latencies, with jitter around 2 ms.   91

- Accurate: false negative rates under 1% and false positive rates under   92
  ~~0.001~~0.005% for a variety of syllables. ~~Balancing these rates depends on a single~~   93
  ~~relative-cost parameter.~~   94

- State-of-the-art performance with default parameters.   95

- Requires almost no programming experience.   96

- Runs on inexpensive hardware.   97

- Described in detail here, with reference implementations provided and   98
  benchmarked.   99

2 Materials and Methods   100

2.1 Learning a detector   101

We begin with a few hundred recordings of a given bird's song, as well as calls, cage   102
noise, and other non-song audio data. Male zebra finch song is most highly   103
stereotyped when a female is present ("directed song") and slightly more variable   104
otherwise ("undirected"); we train and test on undirected song since this is both more 105
commonly studied ~~, as well as~~ and more challenging. Recordings were made inside a   106
sound-isolating chamber in which was mounted a recording microphone   107
(Audio-Technica AT803), using methods similar to those described in [12], Chapter 2.   108
The songs were time-aligned as described in [11].   109

    We rely on two circular buffers: 110

Audio buffer: This contains the most recent audio samples, and is of the length   111
    required to compute the Fast Fourier Transform (FFT)—usually 256 samples.   112

FFT buffer: The results of each new FFT are placed here. It contains the n_{fft} most   113
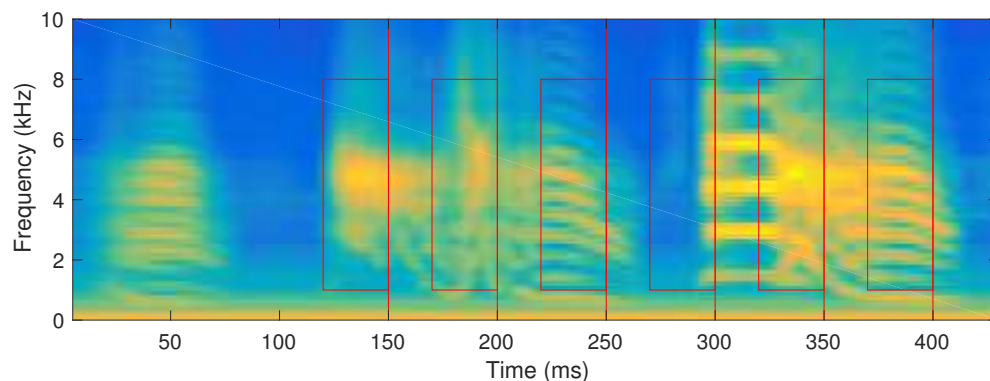   recent FFTs, which will serve as inputs to the neural network (described below).   114

Figure 1. The spectrogram of ~~a typical~~ the song of the bird "lny64", used as ~~the~~ an example throughout this paper. 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~~ example recognition regions ~~(of~~ 30 ms × 1–8 kHz ~~)~~ marked as rectangles.

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

The spectrogram is computed at regular intervals—the useful range seems to be roughly 1–5 milliseconds, which we refer to as the frame interval t_{fft}. At each frame, a spectrum is computed from the most recent 256 audio samples in the audio buffer, and the result is appended to the FFT buffer. For example, if $t_{\mathrm{fft}} = 1$ ms and the recording sample rate $1/t_{\mathrm{sample}} = 40$ kHz, then in order to compute a new FFT, $40000 \cdot 0.001 = 40$ new audio samples must be appended to the audio buffer, the FFT is computed using the most recent 256 samples in that buffer, and the result is appended to the FFT buffer. Over time, the successive spectra will look something like Fig. 1.

~~Note that this~~ 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) [13]. This allows $t_{\mathrm{fft}} = t_{\mathrm{sample}}$. 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 over F at $\tau(t)$ and those from the contiguous set of recent frames: $T = \{ \tau(t), \tau(t - t_{\text{fft}}), \tau(t - 2t_{\text{fft}}) \dots \tau(t - n_{\text{fft}} t_{\text{fft}}) \}$. ~~We have found that a 30-ms time window—which will yield $n_{\text{fft}} = |T| = \lfloor 30 \text{ ms}/t_{\text{fft}} \rfloor$~~ Time windows of 30–50 ms—the latter will yield $n_{\text{fft}} = |T| = \lfloor 50 \text{ ms}/t_{\text{fft}} \rfloor$ frames—of frequencies spanning 1–8 kHz ~~works well~~ ~~for most syllables~~ generally work well.

Six examples of chosen target moments in the song, ~~and their corresponding recognition regions F and T~~ with recognition regions $F = 1$–8 kHz and $T = 30$ ms, are shown in Fig. 1.

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

Training targets y_t are vectors with one element for each desired detection syllable. That element is, roughly, 1 if the input vector comes from the target time ($t = t^*$), 0

otherwise, for each target syllable (of which there may be any number, although they   166
increase training time, and in our implementations the number of distinct output   167
pulses is constrained by hardware). Since the song alignment may not be perfect, and   168
due to sample aliasing, a strict binary target may ask the network to learn that, of two   169
practically identical frames, one should be a match and the other not. Thus it is   170
preferable to spread the target in time, such that at the target moment it is 1, and at   171
neighbouring moments it is nonzero. We found that a Gaussian smoothing kernel   172
around the target time with a standard deviation of 2 ms serves well.   173

With inputs well outside the space on which a neural network has been trained, its   174
outputs will be essentially random. In order to reduce the false positive rate it is   175
necessary to provide negative training examples that include silence, cage noise, wing   176
flapping, non-song vocalisations, and perhaps songs from other birds. Although it will   177
depend on the makeup of the non-song data, we have found that training with as low   178
as a 1:1 ratio of non-song to song—or roughly ~~10-20~~ 10 minutes of non-song   179
audio—yields excellent results on most birds. ~~These data are also included in our~~   180
~~accuracy tests.~~   181

Normalisation   In order to present consistent and meaningful inputs to the neural   182
network, we normalise the incoming data stream so that changes in the song structure   183
of the sound are emphasised over changes in volume, and to maximise the effectiveness   184
of the neural network's training algorithm.   185

The first normalisation step is designed to eliminate differences in amplitude due to   186
changes in the bird's location and other variations in recording. Each recognition   187
region vector ξ_t—during training, each *column* of the training matrix Ξ—is normalised   188
using MATLAB's `zscore` function $\hat{\xi}_t = (\xi_t - \mu_{\xi_t})/\sigma_{\xi_t}$, so that the content of each   189
window has mean $\mu_{\hat{\xi}_t} = 0$ and standard deviation $\sigma_{\hat{\xi}_t} = 1$.   190

The second step is designed to ensure that the neural network's inputs have a   191
range and distribution for which the training function can easily converge. Each   192
element i of $\hat{\xi}$ is normalised across the entire training set—each *row* of Ξ—in the same   193
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   194
have mean $\mu_{\check{\xi}^i} = 0$ and standard deviation $\sigma_{\check{\xi}^i} = 1$. This is accomplished during   195
training by setting MATLAB's neural network toolbox normalisation scheme to   196

`mapstd`, and the scaling transform is saved as part of the network object used by the realtime detector so that it may be applied to unseen data at runtime.

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

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 [14].

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. ²¹⁷

Computing optimal output thresholds   After the network is trained, outputs of ²¹⁸
the network for any input are now available, and will be in (or, due to noisy inputs ²¹⁹
and imperfect training, close to) the interval (0, 1). We must choose a threshold above ²²⁰
which the output is considered a positive detection. Finding the optimal threshold ²²¹
requires two choices. The first is the relative cost of false negatives to false positives, ²²²
C_n. The second is the acceptable time interval: if the true event occurs at time t, and ²²³
the detector triggers at any time $t \pm \Delta t$, then it is considered a correct detection. ²²⁴
Then the optimal detection threshold is the number that minimises ²²⁵
[false positives] $+ C_n \cdot$ [false negatives] over the training set, using the definitions of ²²⁶
false positives and negatives given in Section 2.3. Since large portions of the cost ²²⁷
function are flat, we use a brute-force linear search, which requires fractions of a ²²⁸
second. For the results presented here, we have used $\Delta t = 10$ ms, and arbitrarily set ²²⁹
$C_n = 1$. ²³⁰

De-bouncing   During runtime, the network may produce above-threshold responses ²³¹
to nearby frames. Thus, after the first response, subsequent responses are suppressed ²³²
for 100 ms. However, for the accuracy ~~figures~~ measurements presented here, we used ²³³
the un-de-bounced network response. ²³⁴

Our parameter choices   We used an FFT of size 256; a Hamming window; and ²³⁵
chose a target spectrogram frame interval of $t_{\text{fft}} = 1.5$ milliseconds, resulting in a true ²³⁶
frame interval of $t_{\text{fft}} = \lfloor 1.5 \cdot 44.1 \rfloor / 44.1 \approx 1.4966$ ms. We set the network's input space ²³⁷
to ~~30~~ 50 ms long, and to span frequencies from 1–8 kHz, which contains the ²³⁸
fundamentals and several overtones of most zebra finch vocalisations. ²³⁹

   We found these parameters to work well across a variety of target syllables, but ²⁴⁰
various other parameter sets yield results similar to those presented here. Some of the ²⁴¹
parameters trade off detection accuracy or temporal precision vs. training time. For ²⁴²
example, decreasing the frame interval generally decreases both latency and jitter, but ²⁴³
also increases training time. Sometimes the effects are syllable-specific: for example, ²⁴⁴
using a 30-ms time window rather than 50 ms speeds training while usually having a ²⁴⁵
minimal effect on detector performance (as is the case, for example, for all of the ²⁴⁶

detection points for the bird "lny64"), but occasionally a syllable will be seen for   247
which extending the window to 80 ms is helpful.   248

2.2   Realtime detection   249

The architecture of the realtime detector requires that the most recent n_{fft} 250
spectrograms be fed to the neural network every frame interval. Audio samples from   251
the microphone are appended to the circular audio buffer. Every t_{fft} seconds a new   252
spectrogram is calculated by applying the Hamming window to the contents of the   253
buffer, performing an FFT, and extracting the power. Outputs of the spectrogram   254
from the target frequency band are appended to the circular FFT buffer. The   255
spectrograms are sent to a static implementation of the previously trained neural   256
network.   257

We tested three implementations of the realtime detector. For all of these tests, we   258
ran the detector processes under the operating systems' default schedulers and process   259
priorities, running typical operating system daemon processes but no loads from user   260
processes. The computers had ample memory resources.   261

Swift   This detector uses the Swift programming language and Core Audio interface   262
included in Apple's operating systems.   263

The Core Audio frameworks provide an adjustable hardware buffer size for reading   264
from and writing to audio hardware (different from our two circular buffers). Tuning   265
this buffer size provides a tradeoff between the jitter in the detection and the   266
processor usage needed to run the detector. We used buffer sizes ranging from 8   267
samples (0.18 ms at 44.1 kHz) to 32 samples (0.7 ms at 44.1 kHz) depending on the   268
frame size used by the detector.   269

Vector operations—applying the Hamming window, the FFT, input normalisation,   270
matrix multiplication, and the neural network's transfer functions—are performed   271
using the Accelerate framework (vDSP and vecLib), which use modern vector-oriented   272
processor instructions to perform calculations.   273

When the neural network detects a match, it instructs the computer to generate a   274
TTL pulse that can be used to trigger downstream hardware. This pulse can be either   275
written to the computer's audio output buffer (again, in 8- to 32-sample chunks) or   276

sent to a microcontroller (Arduino) via a USB serial interface. Sending the trigger pulse via the serial interface and microcontroller is noticeably faster (2.2 ms lower latency), likely due to the fact that the audio buffer goes through hardware mixing and filtering prior to output.

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

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

LabVIEW   This implementation requires special software and hardware: LabVIEW from National Instruments—we used 2014 service pack 1—and a data acquisition card—we use the National Instruments PCI-6251 card on a PC with an Intel Core i5-4590 processor at 3.7GHz (a relatively low-end machine) with ~~32~~ 24 gigabytes of RAM, running Microsoft Windows 8.1 Pro ~~.~~ and Windows 10.

This implementation has several drawbacks: it requires ~~Windows and~~ expensive hardware and software ~~and~~ from National Instruments (a data acquisition card and LabVIEW), Windows (we use hardware features of LabVIEW that are unavailable on MacOS or Linux), and due to the programming language it is difficult to modify and debug—indeed, a persistent bug in our implementation currently renders it substantially less accurate than the other detector implementations on some syllables. However, our test configuration ~~usually achieved~~ proved itself capable of excellent performance, and further gains should be possible if the implementation were retargeted onto field-programmable gate array (FPGA) hardware—which would have the additional benefit of providing deterministic "hard realtime" guarantees—or just run on a faster desktop system.

MATLAB   This detector uses the built-in audio input and output hardware on a compatible computer. We tested on a 2014 Mac Mini (the same machine used for the Swift detector described above) and 2015 Mac Pro. The Mac Pro does not have an audio input jack, so input was through an M-Audio MobilePre external USB audio

interface. Despite the faster processor, the latter system did not achieve higher performance than the former, due to USB data transfer overhead.

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

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

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

2.3   Quantification

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, and to test robustness to imperfect choice of "syllable", we present performance on~~ <u>measure accuracy and characterise timing for</u> 6 randomly chosen points in one bird's song <u>(lny64), and present accuracy results for seven more birds (the detector's timing does not appreciably vary among birds)</u>. The results presented here are representative of real-world performance ~~over~~ <u>in feedback experiments on</u> many birds in our lab.

~~We divided the data into training and test sets. For the results reported in the following section, the training set consisted of~~

<u>Data consisted of extracted, aligned songs, and additional</u> song-length samples of non-song (cage noise, calls, other birds' song). ~~Our test~~ <u>These were divided into</u>

Bird	Test songs = Test non-songs	Target time (ms)
lny64	1818	150:50:400
lny46	1000 ~~instances of the song of one bird, and~~	195
lny42	1245	380
lny4rb	1000	250
lr12	2000	180
lr13	2000	300
lr28	3000	140
lr77	2000	320

Table 1. For each bird, detectors were trained for the specified timepoints, using 1000 songs and 1000 non-song samples of the same length. The test set consisted of the remaining songs and an equal number of non-songs.

training and test sets. For all birds, we used 1000 songs and 1000 non-song samples for training. The test set consisted of ~~an additional 1818 iterations of the bird's song and 3636 song-length samples of non-song~~ the remaining songs and an equal number of non-songs. Table 1 lists the test dataset sizes and target detection points. For lny64, we arbitrarily chose six linearly spaced target times. For each of the other birds, we eyeballed one promising-looking detection point.

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

Because the data set consists of temporally aligned songs, ~~ground truth is available for each song~~ the moment at which detection should occur during each song sample is available (albeit with minor variations due to the alignment process used [11])~~, and thus the~~. The detector can be checked by ~~presenting~~ comparing its response to the recorded training songs ~~as well as the canonical~~ against these "canonical" detection events. To this end, besides the trained network object, our learning code produces an audio file consisting of all of the training data on the left audio channel and a delta function at each aligned moment of target syllable presentation on the right channel. Thus, when played on any audio player, the left channel may be provided as input to the detector, and the the detector's output pulses may be compared against the "ground truth" given by the canonical detection event provided by the right channel.

Accuracy We define the accuracy of the network based on its classification performance per frame. In order to avoid the apparent problem of every non-detected

~~non-syllable~~ <u>non-matching frame</u> counting as a true negative, we also tried defining accuracy on a per-song basis, such that a song-length sample without the target syllable counted as a single true negative. Computing the optimal output thresholds on a per-frame rather than a per-song basis resulted in higher thresholds and thus a slight reduction in both the false-positive and true-positive rates (reversible by increasing C_n), while also providing a valid definition of false-positive rate for data streams that had not been segmented into song-sized chunks and thus allowing easy interpretation of false-positive rates from calls, cage noise, etc.

The accuracy as defined above is used for computing the optimal thresholds above which the network's output should be interpreted as a match on the training data as described in Section 2.1, for evaluation of the detectors on the training songs, and while live.

Timing   We evaluate the time taken from the complete presentation of each instance of the target syllable to the firing of the detector's TTL pulse. For example, if the target trigger point is at 200 ms with respect to the corpus of aligned songs, and if the detection region is 30 ms long, then when the detector has seen the region from 170–200 ms, it should recognise the syllable. If for a given presentation of the song the detector fires at 203 ms, we take the latency on that trial to be 3 ms.

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

~~Mean latency is then given as the average~~ We define latency as the time between the ~~canonical signal~~ training target (during our tests, this is indicated by the canonical signal in the test file) and the corresponding detection event. It is a helpful number, but not a critical one~~, since in zebra finch~~; due to the stereotyped song of the zebra finch, a detector with high but constant latency can often be trained to trigger at a point somewhat before the true moment of interest (for example, if triggering should occur at the very beginning of a syllable, the detector ~~'s time window may include previous syllable). Often more important is~~ may be trained to recognise the previous syllable and the gap thereafter). Usually the variability in ~~the latency~~ latency is the more important number. We define jitter as the standard deviation of ~~the latency~~ detection latency, measured over the test songs.

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

First we test the "ideal" latency and jitter in the time shift used in calculating new 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,   423

but also the variability of the bird's song.   424

2.4 Ethics   425

All procedures were approved by the Institutional Animal Care and Use Committee of   426

Boston University (protocol number 14-029).   427

3 Results   428

Fig. 1 shows ~~the songfor which we present accuracy and timing results. Six example~~ a   429

typical song. The image is produced by averaging the aligned spectrograms for the   430

bird lny64. Six target trigger points are shown for this song, at 150, 200, 250, 300, 350,   431

and 400 milliseconds after the beginning of the aligned samples, as indicated by the   432

red lines~~, and referred to henceforth~~. In discussion, we will refer to these six as   433

$t_1^* \ldots t_6^*$ respectively. ~~The rectangles show the recognition window (~~To avoid overlap in   434

the graphic, the rectangles show a shorter recognition window than our standard: here   435

30 ms, but still spanning 1–8 kHz~~) for each instant~~, for each target time in the song.   436

The first two triggers, t_1^* at 150 ms and t_2^* at 200 ms, are far from pure tones, but   437

are closer to ~~coloured~~ bandpass-filtered white noise. They would be difficult for a   438

harmonic-stack filter bank to detect, and especially to pinpoint in time. t_3^* (250 ms)   439

and t_6^* (400 ms) are rich in overtones and contain downward slides with changing   440

timbres. t_4^* (300 ms) occurs near the beginning of a more typical harmonic stack   441

amenable to a variety of recognition techniques, although consistently detecting a   442

point partway through the syllable demands a detector that can use multiple time   443

steps. t_5^* (350 ms) shows a steady pitch followed by a complex changing tone structure.   444

An example of the detector's output for three of these syllables is shown in Fig. 2.   445

The total audio energy of each song is shown as a single grayscale row. By sorting   446

according to time of each syllable's detection, the rest of the song is shown in the   447

context of the timing for that moment. This gives an intuition of the timing variability   448

within each song and across songs (which is responsible for a small amount of our   449

measured jitter, but note that few of the songs are significantly timeshifted).   450
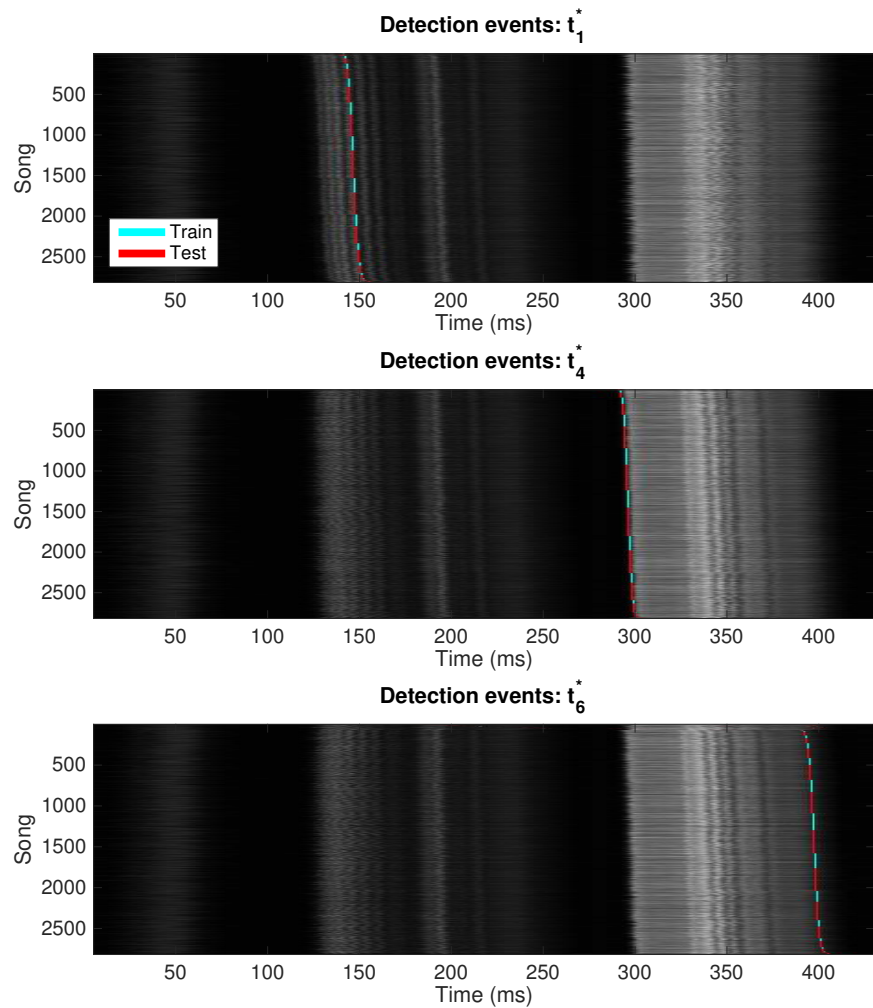
Figure 2. Each plot shows one network output unit's responses to all 2818 presentations of ~~the~~ lny64's song shown in Fig. 1. ~~For simplicity, we~~ We show only the syllables t_1^*, t_4^*, and t_6^*, and we do not show the non-response to presentation of non-song. The horizontal axis is time relative 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.

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~~see.

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 _both_ in the random initialisation of network weights and ~~on which~~ _in the_ random subset of ~~our songs~~ _the song-and-nonsong corpus that_ was used for training. Each network's accuracy is shown as a single point in each chart in Fig. 3, and the means are shown in Table 2. Evidently some syllables are easier to identify reliably than others (for example, _for lny64, t_3^* (250 ms) and t_6^* (400 ms)_ are occasionally confused due to their similar appearance, as could be guessed from Fig. 1); thus, if the experimental design permits, it can be worthwhile to choose an easily identifiable syllable. Furthermore, as can be seen from Fig. 3, for difficult syllables the training process occasionally yields a network that performs unusually poorly. 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, perhaps taking the best of a few training runs.

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%.

Changing the FFT interval to 2 ms significantly reduces the computational requirement during learning. For most syllables, results were identical, but some syllables are slightly easier to detect and some slightly more difficult, with no significant difference on average.
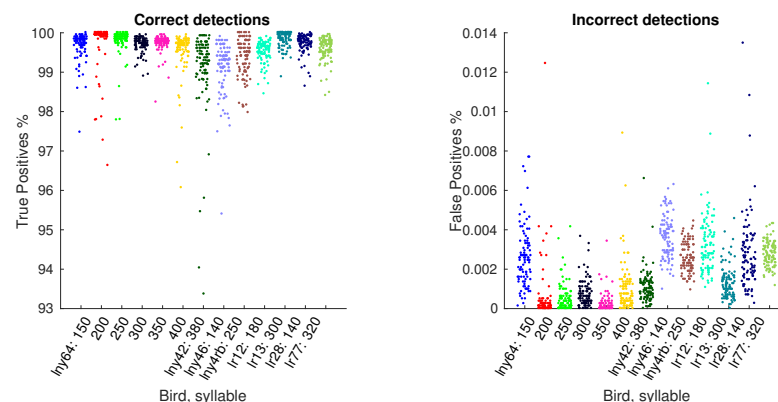
Figure 3. Accuracy variability over 100 different training runs for each of the ~~six~~ test ~~syllables~~detection points. Each dot shows the test-set accuracy for an independently trained detector. Because the horizontal positions have been randomised slightly so as not to occlude same-valued measurements, test syllable is also indicated by colour. The means are given in Table 2.

Bird	Target time (ms)	% True positives	% False positives
~~t_1^*~~ lny64	~~99.34~~ 150 $[t_1^*]$	~~0.0015~~ 99.66	0.0026
t_2^*	~~99.65~~ 200 $[t_2^*]$	~~0.00040~~ 99.75	0.00052
t_3^*	~~98.98~~ 250 $[t_3^*]$	~~0.0015~~ 99.79	0.00051
t_4^*	~~99.30~~ 300 $[t_4^*]$	~~0.00093~~ 99.70	0.00077
t_5^*	350 $[t_5^*]$	99.73	~~0.000094~~ 0.00022
t_6^*	~~98.31~~ 400 $[t_6^*]$	99.52	0.0010
~~Target time~~ lny42	380	99.09	0.0011
lny46	140	99.11	0.0036
lny4rb	250	99.37	0.0026
lr12	180	99.52	0.0033
lr13	300	99.83	0.0013
lr28	140	99.73	0.0028
lr77	320	99.58	0.0028

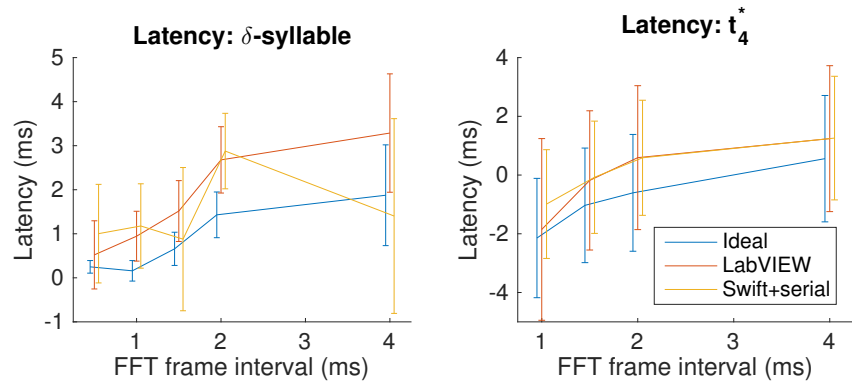**Table 2.** Mean values for the detection accuracies shown in Fig. 3.

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~~ lny64's 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.

3.2 Timing

We evaluate variability in timing performance across three variables: FFT frame interval; syllable choice; and detector implementation.

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~~ lny64's song at t_4^*.

Syllable choice Syllable choice impacts detector performance, but despite the variety of syllables ~~chosen~~ shown here, performance was ~~fairly~~ stable across syllables. Fig. 5 shows measured timing data for lny64's test syllables 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):

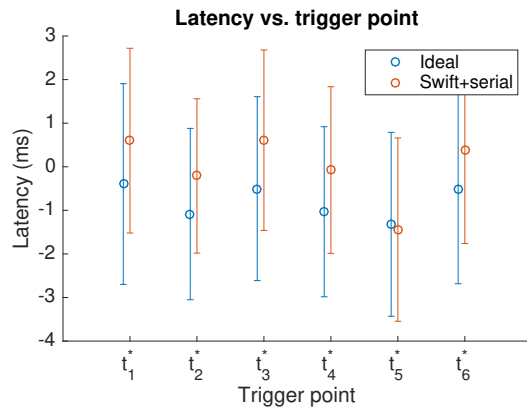| Detector	Latency	Jitter
Ideal	-0.8 ± 0.3 ms	2.1 ± 0.1 ms
Swift+serial	0.0 ± 0.6 ms	2.0 ± 0.1 ms

Figure 5. Timing data for ~~our~~ lny64's 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.

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

Detector Implementations We compared latency and jitter across our different detector implementations for the δ-syllable and lny64's t_4^*, again with $t_{\mathrm{fft}} = 1.5$ ms. Results are as follows:

Detector	δ-syllable		Bird: t_4^*	
	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 the timing curves look like for the five implementations of our detector on t_4^*.
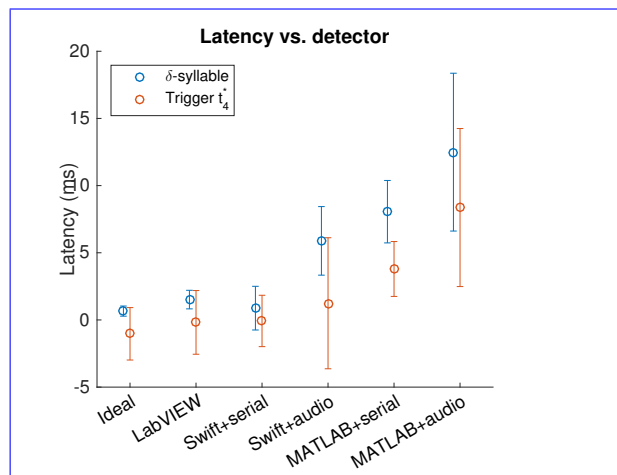
Figure 6. The different detectors for the constructed δ-syllable and for ~~the bird~~ lny64's song at t_4^*. Point centres show latency; error bars show jitter.
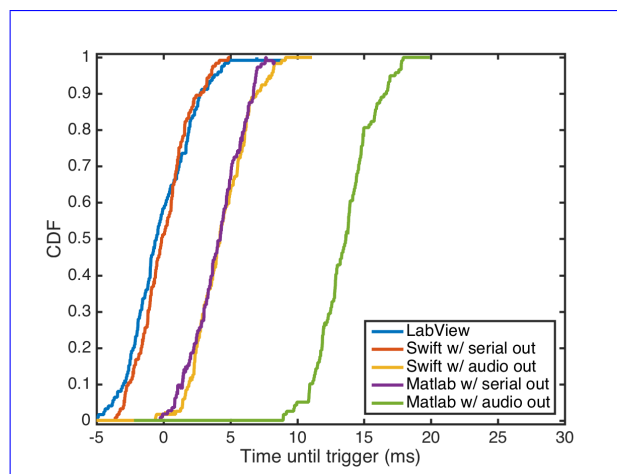

Figure 7. Raw timing curves for all detectors measured during detection of lny64's t_4^* using 1.5-ms frames. We extract the trigger events from each curve, from which we obtain the mean—latency—and standard deviation—jitter.

4   Discussion

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

- The detector is accurate. False negative and false positive rates can be well under 1% and ~~0.001~~0.005% 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~~ songs presented here were recorded with fixed microphones mounted inside the ~~cage~~cages. 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 [15].

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, DOIs for the latest code will be created as described in https:~~

Data: Data and training configuration files for the 8 birds discussed here are available from the Open Science Framework at DOI: 10.17605/OSF.IO/~~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).~~BX76R . DOIs for the versions of the programmes used here are provided for reproducibility; however, we generally recommend

downloading the latest release. 567

Song alignment: Jeff Markowitz's song alignment software [11] can be found at 568

`https://github.com/jmarkow/zftftb` 569

~~Our implementation of the syllable detector is available under an open-source~~ 570

~~license (Tentatively GPL, but we are still evaluating which open source license is~~ 571

~~the most suitable for this software):~~ 572

Training the neural network: 573

`https://github.com/gardner-lab/syllable-detector-learn` 574

DOI: 10.5281/zenodo.437555 575

Installation: 576

See `README.md` in the repository for complete, up-to-date instructions. 577

- Clone the Github repository onto your computer, and add it to your 578
  MATLAB path. 579

- You also need the following packages from MATLAB File Exchange 580
  ~~somewhere~~ in your MATLAB path: 581

  Significant Figures by Teck Por Lam. File ID #10669. 582

  Generate maximally perceptually-distinct colors by Tim Holy. 583
  File ID #29702. 584

Usage: 585

- The main programme is `learn_detector.m`. 586

- The top section of `learn_detector.m` ~~contains global~~ describes the 587
  configuration variables. ~~The important ones are the data directory and~~ 588
  ~~the bird name: as supplied, the software looks for~~ 589

- The current directory should contain the data in ~~<data~~ 590
  ~~directory>/<bird name>/~~song.mat. ~~Modify as necessary.~~ 591

- The format for ~~the data file <data directory>/<bird~~ 592
  ~~name>/song.mat~~this file is specified in 593
  ~~syllable-detector-learn/~~README.md: a MATLAB file containing 594

aligned song, nonsong (calls, cage noise, etc), and the audio sampling frequency.

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

- Configuration may also be controlled by specifying parameter-value pairs to the function call in the common MATLAB fashion; for example, `learn_detector('times_of_interest_ms', [150 200])`.

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

Runtime:

MATLAB:

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

Installation:

- Clone the Github repository onto your computer.
- Usage requires MATLAB with the Data Acquisition Toolbox and the Parallel Computing Toolbox.

Connection:

- Either connect the microphone directly into the microphone jack or through a preamp into the audio input jack. The current implementation only supports a single channel of audio.

- To generate TTL signals via the audio output, use the headphone ₆₂₄ or audio output jack. ₆₂₅

- To generate TTL signals via an Arduino, switch to the `arduino` ₆₂₆ branch, load the MATLAB Arduino IO sketch onto the Arduino ₆₂₇ (the sketch is available in the Github repository, and provides a ₆₂₈ simple serial protocol for controlling pins). Connect the Arduino to ₆₂₉ the computer via USB. The TTL signal will be available on pin 7. ₆₃₀

Usage: ₆₃₁

- Switch to the directory of the repository and run `nndetector_live`. ₆₃₂ You can optionally specify parameters as string/value pairs. When ₆₃₃ you run the command, you will be prompted to select an input ₆₃₄ device, output device and select a ".mat" file containing the trained ₆₃₅ detector. ₆₃₆

- Note that you may need to tune the input and output buffer based ₆₃₇ on computer performance. Try decreasing the size of the buffers ₆₃₈ until you see warnings about buffer under/overruns, then increase ₆₃₉ the value slightly. We usually found 2-3ms worked reliably. To ₆₄₀ specify the buffer sizes, use the `buffer_size_input` and ₆₄₁ `buffer_size_output` parameters. ₆₄₂

- Information about the detector is logged to a file called ₆₄₃ "detector_status.log". ₆₄₄

Swift: ₆₄₅

`https://github.com/gardner-lab/` ₆₄₆

`syllable-detector-swift`~~`/releases/tag/0.3.1`~~ ₆₄₇

DOI: 10.5281/zenodo.437559 ₆₄₈

Installation: ₆₄₉

Download the most recent release from the Github repository onto your ₆₅₀ computer. ₆₅₁

Connection: ₆₅₂

- Either connect the microphone directly into the microphone jack or ₆₅₃ through a preamp into the audio input jack. If the input jack ₆₅₄

supports multiple channels (e.g., audio input jacks are usually 655 stereo), you can connect multiple audio sources and run different 656 detectors on each channel simultaneously. 657

- To generate TTL signals via the audio output, use the headphone 658 jack. The audio output must have at least the same number of 659 channels as the input. 660

- To generate TTL signals via an Arduino, load the MATLAB 661 Arduino IO sketch onto the Arduino (the sketch is available in the 662 Github repository, and provides a simple serial protocol for 663 controlling pins). Connect the Arduino to the computer via USB. 664 The TTL signal for the first input will be available on pin 7, the 665 second input on pin 8, etc. 666

Usage: 667

- Use `convert_to_text.m` to convert the Matlab output file to a 668 format easily readable by Swift. This will generate a text version of 669 the neural network. 670

- Launch the application downloaded above. It will provide an menu 671 to select an audio input and either an audio output or an Arduino 672 output (if detected). Once you select an input and an output, a 673 new window will show all audio channels associated with the input. 674 For each channel, select the text-encoded version of the detector 675 created in the previous step. Once configured, press run to begin 676 monitoring the inputs. 677

Customisation: 678

- You can clone the Github repository and use Xcode to edit the 679 project, if you are interested in extending or modifying the 680 functionality. To hook into either the raw input or output, look at 681 the "Processor.swift" file. The `Processor` class acts as a 682 coordinator, receiving incoming audio, passing that to the detector 683 and generating appropriate output TTL signals. 684

LabVIEW: 685

```
https://github.com/gardner-lab/                                        686
syllable-detector-runtime-labview                                      687
```

10.5281/zenodo.437558   688

Installation: 689

Clone the Github repository onto your computer.   690

Connection: 691

- The microphone should be connected, optionally through a   692
  preamplifier, to the data acquisition card, by default on Analog   693
  Input 0 (AI0).   694
- Outputs are on the Digital I/O (DIO) channels. See the block   695
  diagram of `nnfft.vi` for documentation, or just connect to DIO0. 696

Usage: 697

- Use LabVIEW to open `nnfft.vi`.   698
- A file chooser is provided in the front panel of `nnfft.vi`. Use it to 699
  point to the network file produced by `learn_detector.m`, and use 700
  LabView's "Run" button to begin detection.   701

References   702

1. Kao MH, Doupe AJ, Brainard MS. Contributions of an avian basal   703
   ganglia–forebrain circuit to real-time modulation of song. Letters to Nature.   704
   2005 February;433:638–643. Available from: 705
   http://www.nature.com/nature/journal/v433/n7026/abs/nature03127.html. 706

2. Wang CZH, Herbst JA, Keller GB, Hahnloser RHR. Rapid Interhemispheric   707
   Switching during Vo-   708
   cal Production in a Songbird. PLOS Biology. 2008 October;6(10). Available from: 709
   http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.00602 710

3. Keller GB, Hahnloser RHR. Neural processing of auditory feedback during 711
   vocal practice in a songbird. Nature. 2009 January;457:187–190. Available from: 712
   http://www.nature.com/nature/journal/v457/n7226/abs/nature07467.html. 713

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. Tchernichovski O. Sound Analysis Pro User Manual; 2011. Available from: 743

    http://soundanalysispro.com/manual-1. 744

13. Jacobsen E, Lyons R. The sliding DFT. IEEE Signal Processing Magazine. 2003 745

    Mar;20(2):74–80. Available from: 746

    http://www.cmlab.csie.ntu.edu.tw/DSPCourse/reference/Sliding%20DFT.pdf. 747

14. Hertz JA, Krogh AS, Palmer RG. Introduction to the Theory of Neural 748

    Computation. Perseus Books; 1991. 749

15. Noll AM. Pitch determination of human speech by the harmonic product 750

    spectrum, the harmonic sum spectrum, and a maximum likelihood estimate. In: 751

    Proceedings of the symposium on computer processing in communications. 752

    vol. 19; 1970. p. 779–797. 753