A Fast and Accurate Zebra Finch Syllable Detector

Ben Pearre[1,¤], L. Nathan Perkins[1], Jeffrey E. Markowitz[1], Timothy J. Gardner[1]

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

¤ 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 specific moments during song be identified as the bird sings in real time. 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. Most moments during song can be isolated and detected with $> 95\%$ accuracy, easier syllables can be detected 99.5% of the time, with fewer than 0.1% of songs producing false positives. The detector can run on a stock Mac Mini with a triggering delay around 3 milliseconds and trigger jitter with $\sigma \approx 1.2$ 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, is almost identical each time, and several brain areas reflect this consistency in highly stereotyped neural firing patterns. This consistency makes the

Might be fun to throw in the obligatory raster plot?

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 precise moments in song can reliably be detected quickly enough to trigger other apparatus during singing. A common area of study with this song-triggering technique is the anterior forebrain pathway (AFP), a homologue of mammalian basal ganglia consisting of a few distinct brain areas concerned with the acquisition and learning of song. For example, [1] stimulated 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] showed that AFP produces a corrective signal to bias song away from error conditions, which are generated by playing a disruptive sound for any detected rendition of a syllable that is slightly above or below its average pitch. [3] showed that that AFP transfers this signal to the robust nucleus of the arcopallium (RA) using NMDA-receptor–mediated glutamatergic transmission. By looking at song recovery after applying the same pitch-shift paradigm, [4] showed that the caudal medial nidopallium is implicated in memorising or recalling a recent song target, but for neither auditory processing nor directed motor learning.

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

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, [5] used groups of IIR filters with hand-tuned logical operators. Their system had a latency of 50 or 100 milliseconds

(ms), and they do not report on jitter or accuracy. As access to computational resources has improved, approaches have changed: in 2009, [2] still used hand-tuned filters, but ran them on a Tucker-Davis Technologies digital signal processor. They report a latency of around 4 ms, but as with other filter bank techniques, it is not strictly a syllable detector but rather a pitch 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. [6] applied a neural network to a spectral image of song. It reported a latency of 4.3 ms, but further implementation and performance details are not available, and the authors claim that their detector is too deeply integrated into their experimental apparatus to be useful to the community [personal communication]. In 2011, [3] matched spectral templates to stable portions of syllables in 8 ms segments. It reported a jitter of 4.5 ms, and false-negative and false-positive rates of 0-2% and 0-4%, respectively. Hardware requirements and ease of use were not reported. In 2013, [7] described in detail a system that matches spectral images of template syllables using a correlation coefficient. With a fast desktop (Intel i7 six-core) running Linux and equipped with a National Instruments data acquisition card, it boasts a hardware-only (without accounting for the time taken to compute a match with a syllable) latency and jitter of just a few microseconds, and the detection computation should not much increase that. Drawbacks are that some hand-tuning is still required, and that they report false-negative rates around 4% and 7% (for zebra finches and Bengalese finches, respectively) measured on a small dataset. In much other work, an allusion is made to a syllable detector, but no further information is provided.

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 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 [8].

2. Choose one or more desired trigger syllables, and train a neural network to recognise them. This step is carried out offline. While any neural network

Tim: that's my understanding of your conversation with Richard. Is this problem worth mentioning?

Terminology: "syllable" is a unit, but this detector triggers on "song moments" or something?

software would produce equivalent results, we used Matlab 2015b. 66

3. Once trained and saved, the neural network is used by a realtime detection 67
   program that listens to an audio signal and indicates detection of the target 68
   syllables via a TTL pulse. We present three implementations that trade off 69
   hardware requirements, ease of maintenance, and performance. 70

This method makes the following contributions: 71

- Fast: latencies in the 3-5 ms range, with jitter around 1.3 ms. 72

- Accurate: false positive rates under 0.1%, and false negative rates well under 1%, 73
  for a variety of syllables. Balancing these rates depends on a single relative-cost 74
  parameter. 75

- State-of-the-art performance with default parameters. 76

- Runs on inexpensive hardware. 77

- Described in detail here, with reference implementations provided and 78
  benchmarked. 79

We present the method in Section 2. Section 2.3 describes how we define and test 80
performance. Section 3 presents our measurements. Section **??** gives an example usage 81
case of the detector. We conclude in Section 4, and point to software resources in 82
Appendix A. 83

2 Materials and Methods 84

2.1 Learning a detector 85

We begin with a few hundred time-aligned recordings of a given bird's song. 86
Time-alignment is described in [8]. 87

The spectrogram is computed at regular intervals—every 1-5 milliseconds, which 88
we refer to as the frame time t_f. 89

One or more moments during the song must be chosen. Our interface presents the 90
time-aligned spectrogram averaged over training songs, and requires manual input of 91

the target times. The user may next tune the region in frequency and time used for   92

syllable identification. Then we assemble the training set from the song data, train the   93

network, compute optimal output unit thresholds, and save the network file and an   94

audio test file.   95

Recognition region   The neural network uses a contiguous set of the most recent   96

timesteps (frames) from the song spectrogram. The frequencies F and duration T of   97

this recognition region should be chosen in order to contain unique features of the   98

target syllable and surrounding areas. This process could perhaps be automated, but   99

we have not done so, as hand-choosing is neither difficult nor particularly error-prone.   100

Song micro-realignment   As conditions change, and especially during undirected   101

song, syllable length and relative timing may vary slightly, which introduces variations   102

in the precise timing of each syllable. The alignment software we use ensures that   103

songs are aligned at the point midway through the song, but if the target syllable is   104

not at that point, it is helpful to re-align the songs at the point of interest. This may   105

be accomplished by looking for peaks in the correlation of the time-domain signal with   106

the song whose spectrogram is closest to average over the training set.   107

I hope that's the right thing to do. Seems to work, anyway. . .

Normalisation   To accommodate differences in amplitude, due to changes in the   108

relative position between the microphone and bird or due to other minor variations in   109

recording, normalisation ensures that the detector is sensitive only to the relative   110

spectral content. We found that a two-step normalisation works. The spectrogram in   111

the time and frequency windows, in decibels, is normalised to have mean zero and unit   112

standard deviation. This eliminates relative differences in song amplitude due to   113

microphone position or line levels. Next, each time-frequency bin is normalised based   114

on the training set, such that that bin has mean zero and unit variance. This scales   115

each time-frequency bin based on the amplitudes seen in the training data. These two   116

normalisation steps provide a set of inputs that were more robust to outliers and less   117

likely to produce false positives during silence when evaluated against other   118

normalisation schemes, such as linear or L2 normalisation.   119

What data is each norm computed over? Why does this do more than just the second step?

Over what set of measurements?

Over what set of measurements? This seems clearer than step 1, but the description of step 1's result sounds like what this would do.

Would it be worth putting in data to this effect?

Building the training set   The neural network's training set is created in the   120
typical fashion: the rectangular $|F| \times |T|$ recognition region in the spectrogram is   121
simply reshaped into a vector of length $|F||T|$. The frequency range is constant, and   122
every possible time interval of length $|T|$ is converted into a training input vector.   123

Training targets are, roughly, 1 if the input vector comes from the target time, 0   124
otherwise, for each target syllable (of which there may be any number, although they   125
increase training time, and in our implementations the number of distinct output   126
pulses is constrained by hardware). Since the song realignment may not be perfect,   127
due to sample aliasing, and because the song spectrogram appears not to vary faster   128
than the frame rate we chose, a strict binary target may ask the network to learn that   129
practically identical samples should have opposite targets. Thus it is preferable to   130
spread the output vector in time, such that at the target moment it is 1, and at   131
neighbouring moments it is nonzero. We found that a Gaussian smoothing kernel   132
around the target time with $\sigma \simeq 3$ ms serves well.   133

With inputs well outside the space on which a neural network has been trained, its   134
outputs will be essentially random. In order to reduce the false positive rate it is   135
useful to provide negative training examples that include silence, cage noise, non-song   136
vocalisations, and perhaps songs from other birds. We have found that training with   137
as low as a 1:1 ratio of non-song to song yields excellent results, although this will   138
depend on the makeup of the non-song data.   139

Training the network   The network is trained using Matlab's neural network   140
toolbox. We tried a variety of feedforward neural network geometries, from simple   141
1-layer perceptrons to more complicated forms and many hidden nodes. Perhaps   142
surprisingly, even the former yields excellent results on many syllables, but a 2-layer   143
perceptron with a very small hidden layer—just 1 or 2 more units than the number of   144
target syllables—was a good compromise between accuracy and training speed.   145
Various other neural network geometries could be tried, as well as any other classifier   146
that executes quickly. For more variable songs, deep structure-preserving networks   147
may be more appropriate, but they are slow to train and unnecessary for zebra finch   148
song.   149

Matlab's neural network toolbox defaults to Levenburg-Marquardt training. This is   150

a fast algorithm, but is memory-intensive, so multiple output syllables or high FFT frame rates require a large amount of RAM and increase training time to hours. Other training algorithms that use less RAM are much slower, and by default they will often terminate before converging due to their performance gradient going to 0.

Computing optimal output thresholds When the network is trained, outputs of the classifier for any input are now available, and will be in 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. 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 τ is the one that minimises [false positives] $+ C \cdot$ [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, random-restart hillclimbing would be effective, but a brute-force search requires fractions of a second. For the results presented here, we have used $C = 1$ and $\Delta t = 20$ ms.

Our parameter choices We used an FFT of size 128 or 256; a Hamming window; and chose a spectrogram frame time of $t_f = 1.5$ milliseconds. We usually define the network's input space to be 20-80 ms long, and to span frequencies from 1-7 kHz, which contains the fundamentals and several overtones of most zebra finch vocalisations.

We found these parameters to work well across a variety of target syllables, but various other parameter sets yield results similar to those presented here. Some of the parameters trade off detection accuracy or detection timing vs. training time. For example, detection latency and jitter cannot be less than the frame rate, but increasing the frame rate increases the size of the training set and thus the training resources required and the detection latency.

Didn't you guys try reducing input size with an autoencoder? Wasn't that effective?

I chose this large value for Δt before I did target syllable realignment, and it could probably be much reduced now. Does this number make our results look bad? Should I explain it, change it and rerun, etc?

2.2 Realtime detection

The architecture of the realtime detector requires that the most recent $|T|/t_f$
spectrograms be fed to the neural network at every timestep (1.5 ms in our example).
Audio samples from the microphone are appended to a circular buffer. Every t_f
seconds a new spectrogram is calculated by applying the Hamming window to the
contents of the buffer, performing an FFT, and extracting the power. Outputs of the
spectrogram from the target frequency band are appended to a second circular buffer.
The spectrograms are sent to a static implementation of the previously trained neural
network.

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

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

The Core Audio frameworks provide an adjustable buffer size for reading from and
writing to audio hardware. Tuning this buffer size provides a tradeoff between the
jitter in the detection and the processor usage needed to run the detector. We settled
on a buffer size of 32 samples (0.7 ms at 44.1 kHz), as this created minimal system
load while achieving detection within the desired lag and jitter.

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

When the neural network detects a match, it instructs the computer to generate a
TTL pulse that can be used to trigger downstream hardware. This pulse can be either
written to the computer's audio output buffer (again, in 32-sample or 0.7 ms chunks)
or 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 be further decreased (at the expense of processor usage) by adjusting the audio buffer sizes.

LabView   This implementation requires special software and hardware: LabView from National Instruments, 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 gigabytes of RAM, running Microsoft Windows 8.1 Pro.

This implementation has several drawbacks: due to the programming language it is difficult to modify and debug, and it requires Windows. However, our test configuration achieved excellent performance, and further gains should be possible if the implementation were retargeted onto 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 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, perhaps 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 the easiest to modify. While it did not perform as well on our test system as the other implementations, the convenience may outweigh the slight

timing performance penalty for some experiments. Key to minimising jitter is the size ₂₃₉ of the audio buffer: on a 2014 Mac Mini running Matlab 2015b the smallest buffer size ₂₄₀ that did not result in read overruns was about 4 ms. ₂₄₁

2.3 Quantification ₂₄₂

Ground truth is given on the training set, and can be measured by presenting the ₂₄₃ recorded training songs as well as the canonical detection events. To this end, besides ₂₄₄ the trained network object, our learning code produces an audio file consisting of all of ₂₄₅ the training data on the left audio channel and a delta function at each moment of ₂₄₆ target syllable presentation on the right channel. Thus, when played on any audio ₂₄₇ player, the left channel may be provided as input to the detector, and the the ₂₄₈ detector's detection pulses may be compared against the right channel. ₂₄₉

Accuracy The Matlab neural network toolbox breaks the given training set into ₂₅₀ three groups: data on which the network is trained, data used to validate the progress ₂₅₁ of the training algorithm, and holdout test data used only as a final measure of ₂₅₂ performance. We further withhold a portion of the training data in order to provide ₂₅₃ another evaluation of performance on unseen data. ₂₅₄

It's a little more complex than that (cross-validation), but can we go with this?

But another test set layer is just paranoia!

We define the accuracy of the network based on its classification performance per ₂₅₅ frame. In order to avoid the apparent problem of every non-detected non-syllable ₂₅₆ counting as a true negative, we also tried defining accuracy on a per-song basis, such ₂₅₇ that a song without the target syllable counted as a single true negative. Computing ₂₅₈ the optimal output thresholds on a per-frame basis resulted in higher thresholds and ₂₅₉ thus a lower false-positive rate, with minimal consequences to the false-negative rate, ₂₆₀ while also providing a valid definition of false-positive rate for data streams that had ₂₆₁ not been segmented into songs. ₂₆₂

The accuracy as defined above is used for computing the optimal thresholds above ₂₆₃ which the network's output should be interpreted as a match on the training data as ₂₆₄ described in Section 2.1, for evaluation of the detectors on the training songs, and ₂₆₅ while live. ₂₆₆

We present the resulting confusion matrix for a few sample songs, and, for ₂₆₇ simplicity, we present a summary of detector accuracy using the area under the ROC ₂₆₈

curve. 269

Timing We evaluate the time taken from the presentation of the target syllable to 270
the firing of the detector's TTL pulse. While playing the audio test file from another 271
device (such as a mobile phone), the TTL output from the ground-truth channel of the 272
audio output may be used as the trigger pulse for an oscilloscope, and compared to 273
the TTL pulse produced by the detector implementation, which sees only the birdsong 274
channel of the audio file. For this purpose we used a pulse generator (Philips PM 5715, 275
with a listed latency of 50ns, jitter of $\leq 0.1\%$ or 50ps, whichever is greater) to widen 276
the detector's output spike to a number much larger than the jitter (100 ms). This 277
obviates pulse length jitter in the output device by essentially discarding the falling 278
edge of the output pulse. The oscilloscope is then set to averaging mode (128-trigger 279
average) in order to collect timing data. The canonical signal is the trigger at $t = 0$, 280
and the average of the detector's detection events will be seen as a low-to-high 281
transition with form approximating the cumulative probability distribution function 282
(CDF) of the detector's output in response to the chosen song event. 283

Mean latency is then given as the halfway point of that detection sigmoid. It is a 284
helpful number, but not a critical one, since a detector with high but constant latency 285
can be trained to trigger at a point somewhat before the true moment of interest. 286
Often more important is latency jitter: how much random variability is there in the 287
latency? 288

We obtain both of these numbers by performing a maximum-likelihood fit of a 289
Gaussian distribution to the timing data obtained from the oscilloscope. 290

It is possible to calculate the ideal latency and jitter, which reflects the inherent 291
latency and jitter in the time shift used in calculating new columns of the spectrogram. 292
By passing a recorded audio sequence into the detector and assuming no input or 293
output latency, it is possible compare the how many additional audio samples beyond 294
the syllable are needed before the spectrogram matches the inputs needed to trigger 295
the neural network. Given the network and FFT parameters used for benchmarking 296
the LabView and Swift implementations of the detector, the ideal implementation will 297
have a latency of $\mu = 1.1$ ms; $\sigma = 1.0$ ms. This latency reflects the FFT size used for 298
calculating the spectrogram, the FFT time shift between columns in the spectrogram 299

and the width of the Gaussian smoothing kernel applied to the ground truth data   300

when training the neural network.   301

3   Results   302

Fig. 1 shows a typical song, with six target moments selected. We trained the network   303

with 8 hidden units for the 6 detection targets spaced 10 ms apart. Matlab's   304

self-report of detection performance for each iteration of the song is shown in Fig. 2,   305

with ROC curves shown in Fig. 3. The beginning of the syllable is difficult to detect,   306

with a few detection events considerably earlier than the correct moment. But the   307

syllable quickly becomes reliably identifiable. By the time the detector has seen 50 ms   308

of the syllable—the sixth detection point—performance is good: the area under the   309

ROC curve is 0.992.   310

Accuracy for a typical good syllable is identical across the three detectors. For one   311

of our examples of a well-chosen syllable from a random bird, the confusion matrix   312

was as follows: 313

	True	
	pos	neg
Detected pos	632	1 (0.07%)
neg	4 (0.63%)	1271

314

Fig. 4 shows the latency and jitter for the three implementations of our detector   315

running on a single syllable from another bird, using the test audio file generated   316

during training on that song (not shown). The maximum-likelihood estimates of the   317

mean μ and standard deviation σ of the detector latencies on this training set are as   318

follows: 319

I should generate, say, 100 random detection points in a few songs, so I can say what something like "average accuracy" is.

Should be! Verify! And/or explain that there is a bug in the LabView implementation, if there in fact is. ALSO need to present a picture of the test syllable or etc...
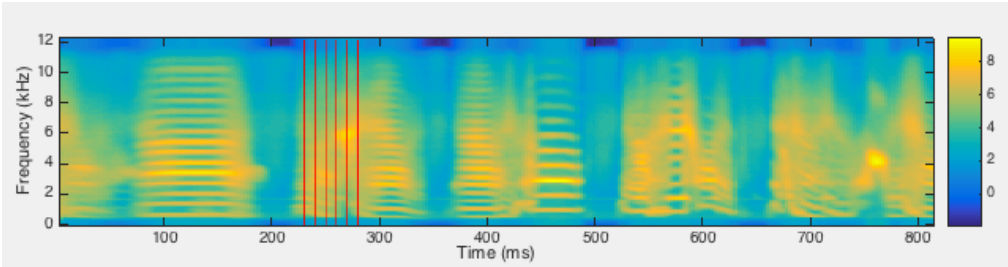
Figure 1. A spectrogram made by averaging over 526 songs. The 6 target syllables, spaced 10 ms apart, are marked by red lines.

| Detector	Latency (ms)	
	μ	σ
Ideal	1.1	1.0
Swift with serial out	2.8	1.2
LabView	3.2	1.4
Matlab with serial out	6.7	1.3
Swift with audio out	6.8	1.3
Matlab with audio out	16	2.9

4   Discussion

This syllable detector is appropriate for zebra finch song, and although we did not test in other species, it is likely to work well for Bengalese finches and probably other species. It offers the following benefits:

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

- Latency can be around 3 ± 1.2 ms or better depending on hardware.

- Works on a wide range of target syllables.

- A single detector can easily generate different target pulses for multiple syllables, at almost no additional computational cost.

- Requires no hand-tuning, although performance on certain syllables can be further improved if desired.
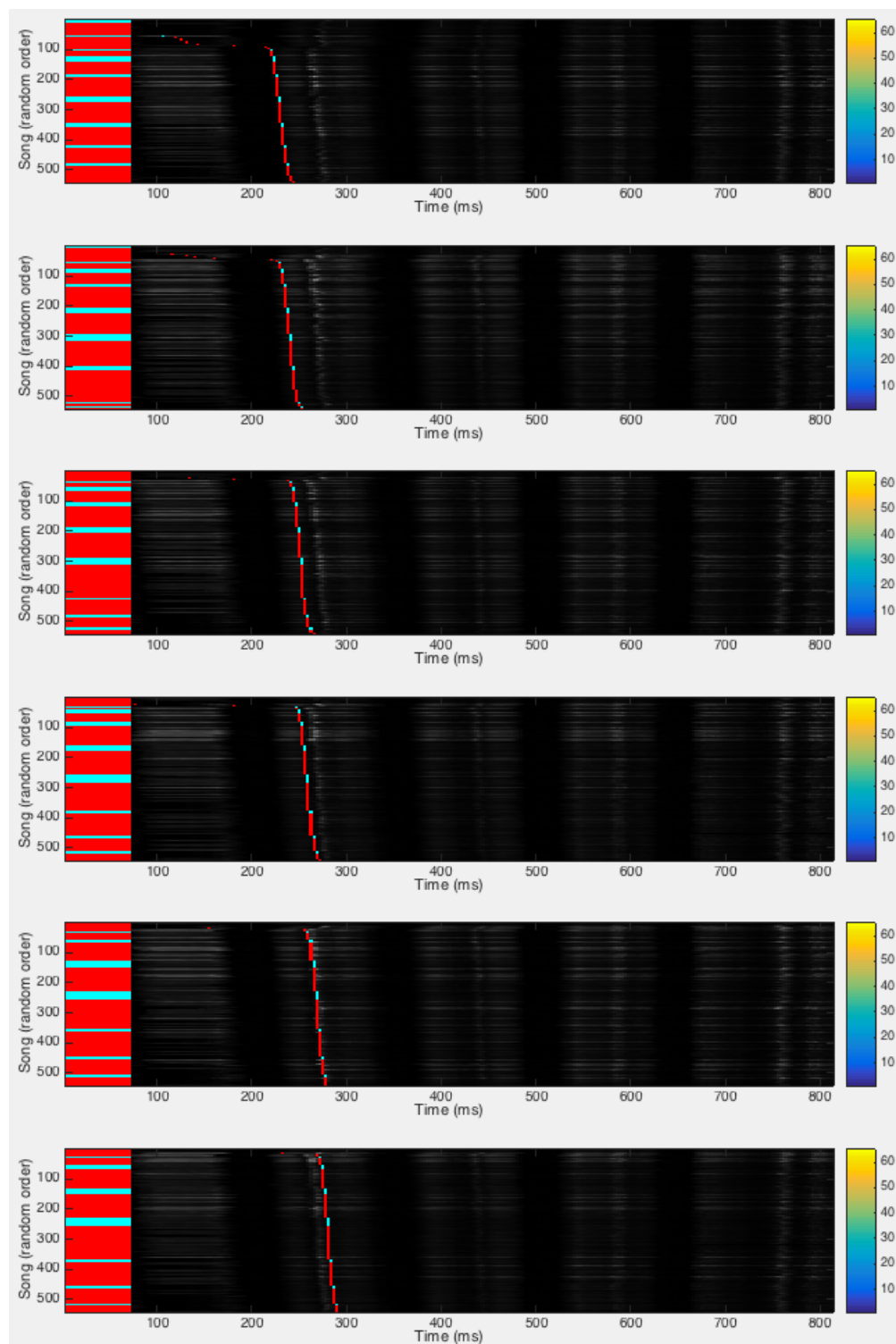
Figure 2. Detection results. Each plot shows detection events for the corresponding syllable as shown in Fig. 1. The horizontal axis is time, and the vertical axis is the index of the 526 single song presentations. The grey shading shows the total audio energy of song Y at time X. The bar on the left is the same width as the detection window, so no detection events can happen within that region, and its colour code shows training songs to the right of red regions and unseen test songs to the right of cyan regions. For visualisation of the distribution, songs have been stably sorted by the time of detection events.
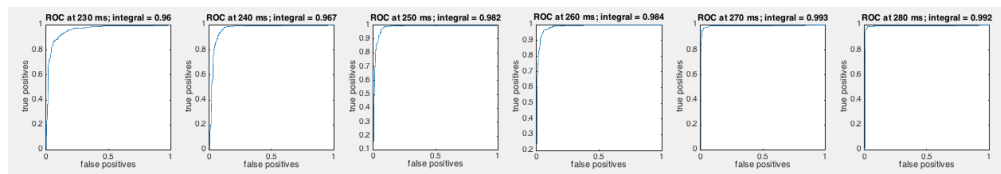
Figure 3. ROC curves for the detection of targets shown in Figure 1. The first one, at the beginning of a syllable, is the most difficult to detect, and as more structure emerges in the current syllable, accuracy approaches 100%.
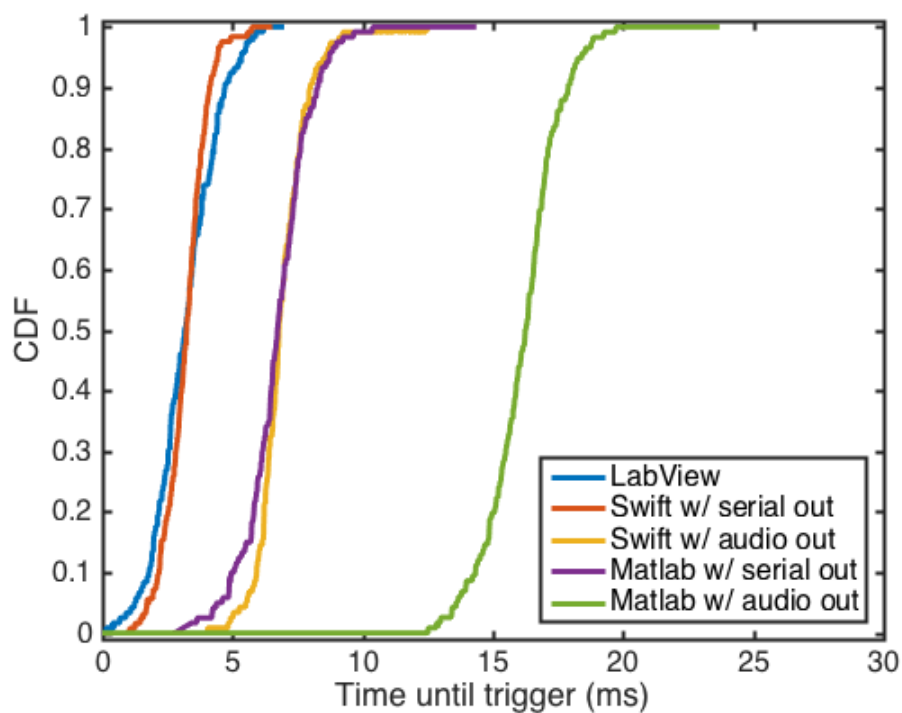
Figure 4. Timing curve for a syllable running a test file generated by training.

- Runs in near realtime on inexpensive consumer-grade hardware, although we   334
  recommend that the training phase be run on a computer with at least 32GB of   335
  RAM.   336

As described, the detector only identifies matches that are similar to the training   337
syllables. A common experimental paradigm requires detecting the frequency of   338
syllables. Many pitch detection techniques rely on the spectrum, which is already   339
available. For example, [4] achieved good results with the Harmonic Product   340
Spectrum algorithm [9].   341

Duration detection is similarly easy. The beginning and end of a syllable may be   342
detected by looking at the ratio of total energy in the singing frequency band   343
(approximately 1-7 kHz) to the total energy, over some small time window. Any   344
syllable thus identified that also contains a trigger event may be monitored for   345
duration. Alternatively, the network can be trained to recognise both the beginning   346
and the end of the syllable of interest.   347

In feedback experiments such as frequency- or duration-shifting, vocal output   348
changes over the course of the experiment. The neural network excels at identifying   349
syllables close to its training set, so as vocal output changes the detector may not   350
recognise a match. If the detector must be robust to this shift, it may be retrained   351
every so often as the bird learns, or data consisting of synthetically pitch-shifted or   352
duration-shifted target syllables over the recognition region may be added to the   353
training set. We will test these approaches in future work.   354

5   Acknowledgments   355

A   Resources   357

Song alignment: Last we checked, Jeff Markowitz's song alignment software could   358
be found at   359
`https://github.com/jmarkow`.   360

But by whom?

PLOS doesn't really say where such things should go.

Cleanup: mostly done. But it would be nice to move all this to the lab's github group.

Training the neural network: Our implementation of the syllable detector training   361
code is available under the GNU GPL at: 362

`https://github.com/bwpearre/`   363

Runtime: The Matlab, Swift, and Labview implementations for executing the   364
trained network: 365

`https://github.com/nathanntg/syllable-detector`   366

References   367

1. Kao MH, Doupe AJ, Brainard MS. Contributions of an avian basal   368
ganglia–forebrain circuit to real-time   369
modulation of song. Letters to Nature. 2005 February;433:638–643. Available from: 370
`http://www.nature.com/nature/journal/v433/n7026/abs/nature03127.html`.   371

2. Andalman AS, Fee MS. A basal ganglia-forebrain circuit in the songbird biases   372
motor output to avoid vocal errors. Proceedings of the National Academy of   373
Sciences of the United States of America. 2009 July;106(30):12518–12523.   374
Available from: `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2709669/`.   375

3. Warren TL, Tumer EC, Charlesworth JD, Brainard MS. Mechanisms and time   376
course of vocal learning and consolidation in the adult songbird. Journal of   377
Neurophysiology. 2011 October;106(4):1806–1821. Available from: 378
`http://jn.physiology.org/content/106/4/1806.full`.   379

4. Canopoli A, Herbst JA, Hahnloser RHR. A Higher Sensory Brain Region Is   380
Involved in Reversing Reinforcement-Induced Vocal Changes in a Songbird. The   381
Journal of Neuroscience. 2014;34(20):7018–7026. Available from: 382
`http://www.jneurosci.org/content/34/20/7018.full`.   383

5. Leonardo A, Konishi M. Decrystallization of adult birdsong by perturbation of   384
auditory feedback. Nature. 1999 June;(399):466–470. Available from: 385
`http://www.nature.com/nature/journal/v399/n6735/abs/399466a0.html`.   386

6. Keller GB, Hahnloser RHR. Neural processing of auditory feedback during vocal 387
   practice in a songbird. Nature. 2009 January;(457):187–190. Available from: 388
   http://www.nature.com/nature/journal/v457/n7226/abs/nature07467.html. 389

7. Skocik M, Kozhevnikov A. Real-time system for studies of the effects of acoustic 390
   feedback on animal vocalizations. Frontiers in Neural Circuits. 2013 391
   January;6(111). Available from: 392
   http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3539774/. 393

8. Poole B, Markowitz JE, Gardner TJ. The Song Must Go On: Resilience 394
   of the Songbird Vocal Motor Pathway. PLOS One. 2012 June;7(6). Available from: 395
   http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0038173. 396

9. Noll AM. Pitch determination of human speech by the harmonic product 397
   spectrum, the harmonic sum spectrum, and a maximum likelihood estimate; 398
   1970. . 399