# A Fast and Accurate Zebra Finch Syllable Detector

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

[1] Department of Biology, Boston University, 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. [c] wrote several types of filterbank software against which we compared, made suggestions for the neural network detector, and prepared the final version of the Matlab code for distribution. [d] provided a stream of suggested techniques, metrics, and goals.

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

**Abstract**

The song of the adult male zebra finch is strikingly stereotyped. Efforts to understand motor output, pattern generation, and learning have taken advantage of this consistency by investigating the bird's ability to modify specific parts of song under external cues, and by examining timing relationships between neural activity and vocal output. Such experiments require that precise moments during song be identified in real time as the bird sings. Various syllable-detection methods exist, but many require special hardware, software, and know-how, and details on their implementation and performance are scarce. We present an accurate, versatile, and fast syllable detector that can control hardware at precisely timed moments during zebra finch song. Many moments during song can be isolated and detected, with false negative and false positive rates well under 0.5% and 0.01% respectively. The detector can run on a stock Mac Mini with a triggering delay around 3 milliseconds and trigger jitter with $\sigma \approx 1.3$ 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 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] played a disruptive sound during renditions of a syllable that were slightly above (or below) its average pitch, and showed that AFP produces a corrective signal to bias song away from those disruptions. [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 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. [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. 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 up to 2% and 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 (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. 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 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 equivalent results, we used Matlab 2015b's neural network toolbox.

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

This method makes the following contributions:

- Fast: latencies around 3 ms, with jitter around 1.3 ms.
- Accurate: false positive rates under 0.02% and false negative rates under 0.5% for a variety of syllables. Balancing these rates depends on a single relative-cost parameter.
- State-of-the-art performance with default parameters.
- Runs on inexpensive hardware.
- Described in detail here, with reference implementations provided and benchmarked.

# 2 Materials and Methods

## 2.1 Learning a detector

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

The spectrogram is computed at regular intervals—the useful range seems to be roughly 1–5 milliseconds, which we refer to as the frame time $t_f$.

One or more 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. 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 uses a contiguous set of the most recent timesteps (frames) from the song spectrogram. The frequencies $F$ and duration $T$ of this recognition region may be chosen in order to contain unique features of the target syllable and surrounding areas. We have found that a 30-ms time window of frequencies spanning 1–8 kHz works well for most syllables.

**Normalisation** To accommodate differences in amplitude due to changes in the relative position between the microphone and bird or other variations in recording, normalisation ensures that the detector is sensitive only to the relative spectral content. We settled upon the following two-step normalisation scheme. First, each spectral time-frequency window is normalised so that the content of that window has mean zero and unit standard deviation. This eliminates relative differences in song amplitude due to microphone position or line levels. Next, each time-frequency point is scaled so that the values of that point across the whole training set have mean zero and unit standard deviation, and this scaling factor is saved as part of the network object used by the realtime detector. These two normalisation steps provide a set of inputs that are more robust to outliers and less likely to produce false positives during silence when evaluated against other normalisation schemes, such as linear or L2 normalisation.

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

Training targets are, roughly, 1 if the input vector comes from the target time, 0 otherwise, for each target syllable (of which there may be any number, although they increase training time, and in our implementations the number of distinct output pulses is constrained by hardware). Since the song alignment may not be perfect, due to sample aliasing, and because the song spectrogram appears not to vary faster than the frame rate we chose, a strict binary target may ask the network to learn that

practically identical samples should have opposite targets. Thus it is preferable to spread the target in time, such that at the target moment it is 1, and at neighbouring moments it is nonzero. We found that a Gaussian smoothing kernel around the target time with $\sigma \approx 2$ ms serves well.

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

**Training the network** The network is trained using Matlab's neural network toolbox. 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 3 times the number of target syllables, or even fewer—was a good compromise between accuracy and training speed. Various other neural network geometries could be tried, as well as any other classifier that executes quickly. For more variable songs, deep structure-preserving networks may be more appropriate, but they are slow to train and unnecessary for zebra finch song.

Matlab's neural network toolbox defaults to Levenburg-Marquardt training. This is a fast algorithm, but is memory-intensive, so many target 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 prematurely due to their performance gradient going to 0.

**Computing optimal output thresholds** After 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, $C_n$. The second is the acceptable time interval: if the

true event occurs at time $t$, and the detector triggers at any time $t \pm \Delta t$, then it is considered a correct detection. Then the optimal detection threshold $\tau$ is the one that minimises [false positives] $+ C_n \cdot$ [false negatives] over the training set, using the definitions of false positives and negatives given in Section 2.3. Since large portions of the cost function are flat, hillclimbing is ineffective. Random-restart hillclimbing would work, but a brute-force search requires fractions of a second. For the results presented here, we have used $C_n = 1$ and $\Delta t = 10$ ms.

**De-bouncing** Since the network may produce above-threshold responses to nearby frames, after the first response, subsequent responses are suppressed for 100 ms. However, for the accuracy figures presented here, we used the un-de-bounced network response.

Nathan, what de-bounce time do you use?

**Our parameter choices** We used an FFT of size 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 30 ms long, and to span frequencies from 1–8 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 temporal precision vs. training time. For example, detection latency and jitter cannot be much less than the frame interval, but increasing the frame rate increases the size of the training set and thus the training resources required and the detection latency.

## 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 and expensive hardware and software. 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 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 slight 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.

## 2.3 Quantification

We divided the data into training and test sets. For the results reported in the following section, the training set consisted of 1000 instances of the song of one bird and 1000 instances of non-song data. Our test set consisted of an additional 1818

iterations of the bird's song and 1818 more instances of non-song data.

The Matlab neural network toolbox further divides our "training" set into internal training, validation, and test sets. We did not stray from the toolbox's default values, and do not discuss Matlab's internal subdivision of our training set.

Because the dataset consists of temporally aligned songs, ground truth is available for each song, and thus the detector can be checked by presenting the recorded training songs as well as the 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 detection pulses may be compared against 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 counting as a true negative, we also tried defining accuracy on a per-song basis, such that a song without the target syllable counted as a single true negative. Computing the optimal output thresholds on a per-frame basis resulted in higher thresholds and thus a lower false-positive rate, with minimal consequences to the false-negative rate, while also providing a valid definition of false-positive rate for data streams that had not been segmented into songs.

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 presentation of the target syllable to the firing of the detector's TTL pulse. While playing the audio test file from any audio playback device, the TTL output from the ground-truth channel of the audio output may be used as the trigger pulse for an oscilloscope, and compared to the TTL pulse produced by the detector implementation, 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 ($\sim 100$ ms). This obviates pulse length jitter 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.

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

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

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

## 3 Results

The top panel of Fig. 1 shows the song for which we present results. Three target syllables are shown, at 150, 315, and 405 milliseconds after the beginning of the aligned samples, as indicated by the red lines. The rectangles show the recognition

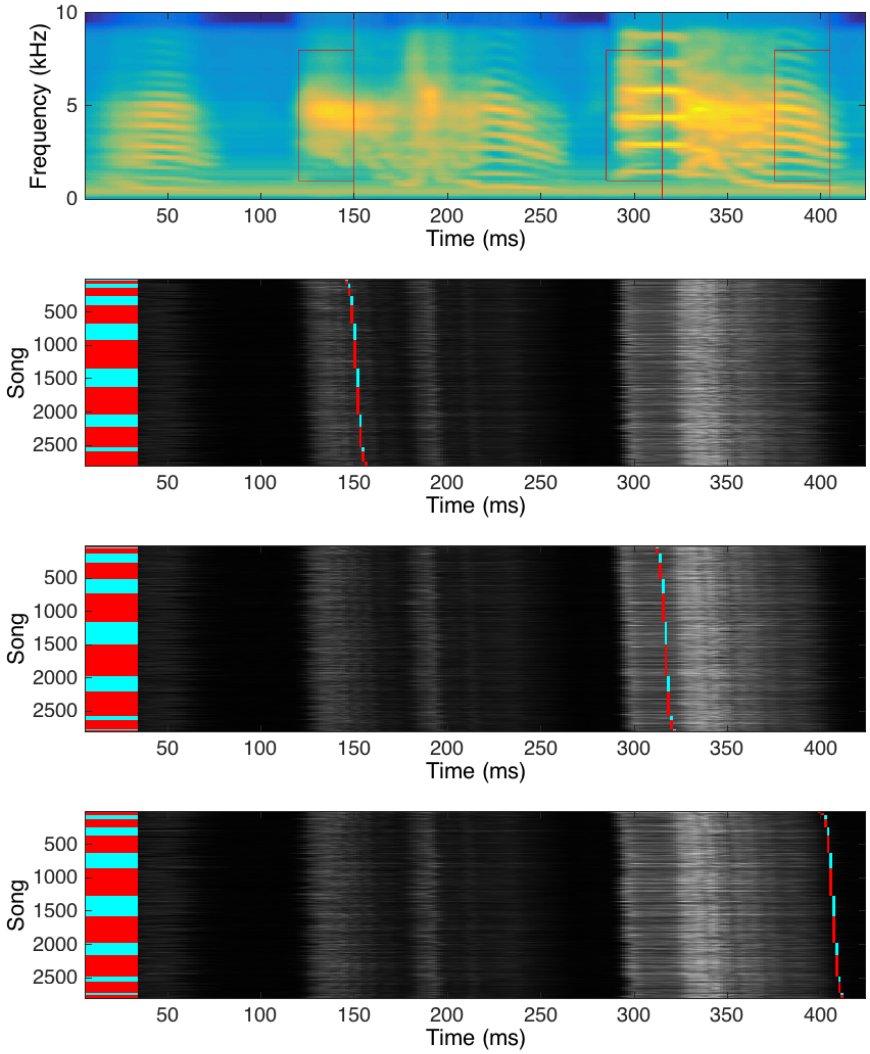


**Figure 1.** The top plot shows the superposition of the spectra of our 2818 aligned songs. The detection points are shown as red lines, with the recognition regions (30 ms × 1–8 kHz) marked as rectangles. Each subsequent plot shows one network output unit's responses to all 2818 presentations of the song. The horizontal axis is time relative to the beginning of the aligned song segment and corresponds to the top figure, and the vertical axis is an index for the 2818 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, and its colour code shows training songs adjacent to cyan regions and unseen test songs adjacent to red regions. For visualisation of the distribution, 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. By sorting according to time of each syllable's detection, the rest of the song is shown in the context of the timing for that moment, giving an intuition of the variability of intrasong timing.

window (30 ms, 1–8 kHz) for each instant in the song.

The first trigger, at 150 ms, is far from a pure tone, but is closer to white noise over the region. It would be difficult for a harmonic-stack filterbank to detect, and especially to pinpoint in time. The second trigger, at 315 ms, is a more typical harmonic stack amenable to a variety of recognition techniques, although consistently detecting a point midway through the syllable demands a detector that can use multiple timesteps. The third syllable, at 405 ms, shows a steady pitch followed by a downward slide along with a change in timbre, and was chosen in order to test the detector's ability to discriminate between it and the similar region at 250 ms.

## 3.1 Accuracy

We allowed our training software to allocate our default of 3 hidden units per syllable, for a total of 9. 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. The confusion matrices are:

| **At 150 ms** | True | |
|---|---|---|
| | pos | neg |
| Detected pos | 99.66611% | 0.00153% |
| neg | 0.33389% | 99.99847% |

| **At 315 ms** | True | |
|---|---|---|
| | pos | neg |
| Detected pos | 99.83306% | 0.00024% |
| neg | 0.16694% | 99.99976% |

| **At 405 ms** | True | |
|---|---|---|
| | pos | neg |
| Detected pos | 99.66611% | 0.00243% |
| neg | 0.33389% | 99.99757% |

### 3.2 Timing

Fig. 2 shows the latency and jitter for the three implementations of our detector running on a single syllable from another bird, using the test audio file generated during training on that song (not shown). The maximum-likelihood estimates of the mean $\mu$ and standard deviation $\sigma$ of the detector latencies on this training set are as follows:

| | Latency (ms) | |
|---|---|---|
| Detector | $\mu$ | $\sigma$ |
| 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 |

The best performance is achieved by the Swift detector using USB serial out, with our LabView implementation performing almost as well. We prefer the Swift implementation due to its lower hardware and software requirements and the difficulty of modifying and debugging LabView programmes. With serial output, Matlab's performance is excellent, although its buffer handling is sensitive to system load. Matlab's failure to produce precisely timed audio output makes that system the least desirable.

## 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 negative and false positive rates can be well under 0.5% and 0.01% respectively, and trading these two numbers off against each other is through a single relative-cost parameter.
- Latency can be around $3 \pm 1.2$ ms or better depending on hardware.

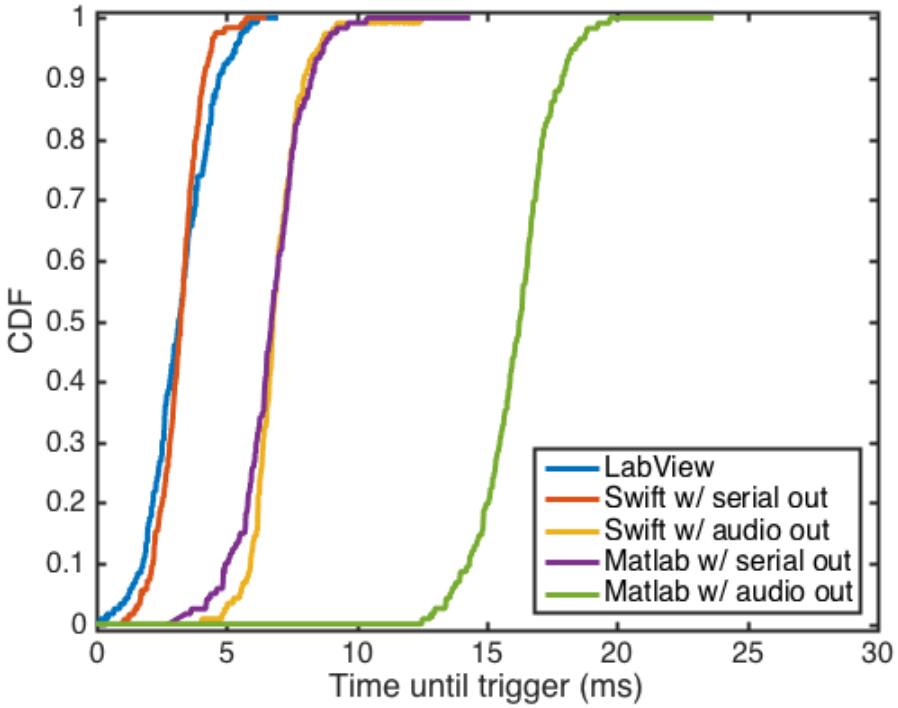


**Figure 2.** Timing curve for a syllable running a test file generated by training.

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

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

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

We would like to thank William A. Liberti for his improvements to the draft. This work was funded by someone.

But by whom?

## A Resources

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.

**Song alignment:** Last we checked, Jeff Markowitz's song alignment software could be found at

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

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

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

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

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

## References

1. Kao MH, Doupe AJ, Brainard MS. Contributions of an avian basal ganglia–forebrain circuit to real-time

modulation of song. Letters to Nature. 2005 February;433:638–643. Available from: http://www.nature.com/nature/journal/v433/n7026/abs/nature03127.html.

2. 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/.

3. 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.

4. 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.

5. 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.

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

7. 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/.

8. 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.

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