A fast, accurate, and robust zebra finch syllable detector

Ben Pearre, Nathan Perkins, Jeffrey E. Markowitz, Timothy J. Gardner

November 25, 2015

Abstract

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 exceed 99.5% detection, and false positive rates can be near 0. The detector can run on a stock Mac Mini with a triggering delay around 2 milliseconds and trigger jitter with $\sigma \approx 2$ milliseconds.

How to explain how to quantify this in the abstract?

1 Introduction

The adult zebra finch 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. This consistency presents a unique opportunity to study the neural basis of learning, audition, and control: we do not know of another model in which neural activity may be so easily correlated with motor output.

Quantify? Remove vacuous handwaving? Figure?

In order to take advantage of this consistency, it is useful to be able to detect selected moments during song, in order to align data or trigger other systems. To the best of our knowledge, current systems require expensive hardware, extensive hand-tuning, and careful choice of an easy syllable, and no standalone solution is available.

Examples?

We developed a standalone detector based on the song spectrogram. Given a set of aligned songs, our software computes spectrograms and trains a neural network to output a TTL pulse at the chosen moment. The approach consists of three steps:

1. Align songs. This is outside the scope of this paper, but we provide a pointer to our song alignment software in Section 7.

2. Choose one or more desired trigger syllables, and train a neural network to recognise them. This is carried out offline using any recent version of Matlab, and we recommend a computer with at least 32GB of RAM.

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

3. Once trained, the Matlab output file is read into code written in Swift. Audio input uses the Mac's stereo audio jack and the CoreAudio interface. Detection of the target syllables is indicated by TTL pulses. Hardware requirements for this system are minimal: we use the current (2015) Mac Mini, which is overkill.

We present the method in Section 2. Section 3 describes how we define and test performance. Section 4 presents our measurements. Section 5 gives an example usage case of the detector. We conclude in Section 6, and point to software resources in Section 7.

2 Method

We begin with a few hundred time-aligned recordings of a given bird's song.

One or more moments during the song must be chosen. Our interface presents the time-aligned spectrogram and requires manual input of the target times. The user may next tune the region in frequency and time used for syllable identification. Then we assemble the training set from the song data, train the network, compute optimal output unit thresholds, present ROC curves for the final network, and save the network file and an audio test file.

2.0.1 Recognition region

The neural network uses several concurrent timesteps from the song spectrogram. The range of frequencies F and length of the history T of this recognition region should be chosen in order to contain unique features of the target syllable and surrounding areas. This process could perhaps be automated, but we have not done so, as hand-choosing is neither difficult nor particularly error-prone.

2.0.2 Song micro-realignment

As conditions change, and especially during undirected song, syllable length and relative timing may vary slightly, which introduces variations in the precise timing of each syllable per song. The alignment software we use ensures that songs are aligned at the point midway through the song, but if the target syllable is not at that point, it is helpful to re-align the songs at the point of interest.

Quick overview of how? I just filched Jeff's realignment code...

2.0.3 Normalisation

Jeff and Nathan changed this. What's our current state of the art?

FIXME

2.0.4 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 FT. Each possible region of size $F \times T$ is converted into a

training input vector. For more variable songs, structure-preserving networks may be more appropriate, but they are slow to train and unnecessary for zebra finch song.

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 negative rate it is necessary 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 roughly a 8:1 ratio of non-song to song yields excellent results, although this will depend on the makeup of the non-song data.

Training targets are, roughly, 1 if the input vector comes from the target time, 0 otherwise, for each target syllable. Since the song realignment may not be perfect, due to sample aliasing, and because the zebra finch's song seems not to vary faster than this, this output vector may be spread 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 $\sigma \simeq 3$ms serves well.

2.0.5 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 more complicated forms and many hidden nodes. Perhaps surprisingly, even the former yields excellent results on many syllables, but a 2-layer perceptron with a very small hidden layer—just 1 or 2 more units than the number of target syllables—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.

Matlab's neural network toolbox defaults to Levenburg-Marquardt training. This is a fast algorithm, but is memory-intensive, so multiple output syllables or high FFT frame rates may require a large amout 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.

2.0.6 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 positives to false negatives, 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 3.1. Since large portions of the cost function are flat, random-restart hillclimbing would be ef-

fective, 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.

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?

Check with Nathan and Jeff on what they've used.

2.1 Our parameter choices

We use a size-256 FFT, a Hamming window, and a spectrum sampled every 2 milliseconds. We usually define the network's input space to be around 80ms long, and to span frequencies from about 1.5-7kHz, 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 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 FFT frame rate, but reducing the frame rate increases the size of the training set and thus the training resources required (and also increases the computational burden on the realtime detector, but that seems low to begin with).

3 Quantification

Ground truth is given on the training set, and can be measured by presenting the recorded training songs as well as the canonical detection events. To this end, besides the trained network object, our Matlab code produces an audio file consisting of all of the training data on the left audio channel and delta functions at the moment of correct detection (basically a TTL pulse, although the voltage will fluctuate with audio output volume) on the right channel. Thus, when played on any audio player, the left channel may be provided as input to the Swift detector, and the right channel may be compared against the Swift detector's detection pulse.

3.1 Precision and Recall

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

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

We define accuracy on a per-song basis, as follows:

True positive: A song for which the detector fires within 20ms of the intended moment.

True negative: So that every non-detected non-syllable does not count, a true negative is a complete song during which the detector did not fire at any point outside the true positive region.

False positive: A detection event more than 20ms from the target syllable.

False negative: A song during which a detection event should have occurred, but didn't.

These definitions are 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 2.0.6, for evaluation of the detector on the training songs in Matlab and Swift, and while live.

Given those definitions, we can compute the false-positive and false-negative rates. For simplicity, we present detector accuracy using the area under the ROC curve.

Would be good to measure FP and FN rates on Nathan's detector! With the test audio file, it's easy—see the LabView code for my spaghetti implementation, or do it anew.

3.2 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 another device (such as a mobile phone), the TTL output from the ground-truth channel of the audio output may be used as the trigger pulse of an oscilloscope, and compared to the TTL pulse produced by the Swift detector, which sees only the birdsong channel of the audio file. For this purpose we used a pulse generator to widen the detector's output spike to about 100ms and set the oscilloscope to averaging mode. 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.

Model? Ramp time? Latency?

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 constent latency can be trained to trigger at a point somewhat before the true moment of interest.

Often of more importance is latency jitter: how much random variability is there in the latency? This we obtain by eyeballing the width of the detection sigmoid in the oscilloscope.

4 Results

Bases for comparison? Jeff's matched filterbank? Nothing really informative.

Figure 1 shows detection on a typical song. The top image is the average of the spectrograms of 528 songs. We trained the network with 8 hidden units and 6 output units, for 6 moments in the song spaced 10ms apart. The beginning of the syllable is difficult to detect, with a few detection events considerably earlier than the correct moment. But the syllable quickly becomes reliably identifiable. ROC curves for this example are shown in Figure 2. By the time the detector has seen 50ms of the syllable, detection is around 99% accurate: the area under the ROC curve is 0.992.

Figure 3 shows the realtime Swift detector running on a single syllable from another bird, using the test audio file generated during training on that song (not

shown). The oscilloscope is triggering on the true signal on CH2 (blue), while CH1 (yellow) shows the average of the detection events, using a pulse generator as described above (a single detection event, non-averaged, would be a single low-to-high transition). Each grid square is 10ms long, so average detection latency appears to be around 1ms, while jitter appears to have $\sigma \simeq 4$ms or so (eyeballed).

Figure 4 is similar to Figure 3 but tested using the LabView implementation, on a different syllable from a different bird, with different parameters. Latency is much higher ($\simeq$ 11ms, but latency jitter is more like $\sigma \simeq 2$ms—note that the grid spacing is 2.5ms. More comparisons required.

Figure this out!

Figure **??** could show a comparison between Jeff's filtering approach on the TDT and my FFT-NN. But how much time do I want to spend describing Jeff's filters? Not much, since it's not in use anywhere besides our lab. Or I could simply insert a photo of Jeff, holding my FFT-NN and a finch, and smiling, with "My name is Dr. Jeffrey E. Markowitz and I approve of this software!"

FIXME

5 Case Study

Jeff is using the detector for something cool, and it might be fun to get him to describe it in a paragraph or two.

Failing that, I can immediately do song-aligned X spike rasters if lw95rhp is still willing to sing... but I have not heard him do so lately.

Do we want to include a case study? I think the paper is on the long side already, but it might be interesting and let Jeff earn that coauthorship ;)

6 Conclusion

This syllable detector is appropriate for zebra finch song. It offers the following benefits:

- False positive and false negative rates well under 1%.
- Latency and jitter are both in the range of 2 milliseconds.
- Works on a wide range of target syllables.
- Requires minimal hand-tuning.
- Runs in realtime on inexpensive compact consumer-grade hardware.
- For training, we recommend a system with at least 32GB of RAM.

It has not been evaluated in other species, and the high accuracy presented here relies on zebra-finch–like consistency of song.

7 Resources

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

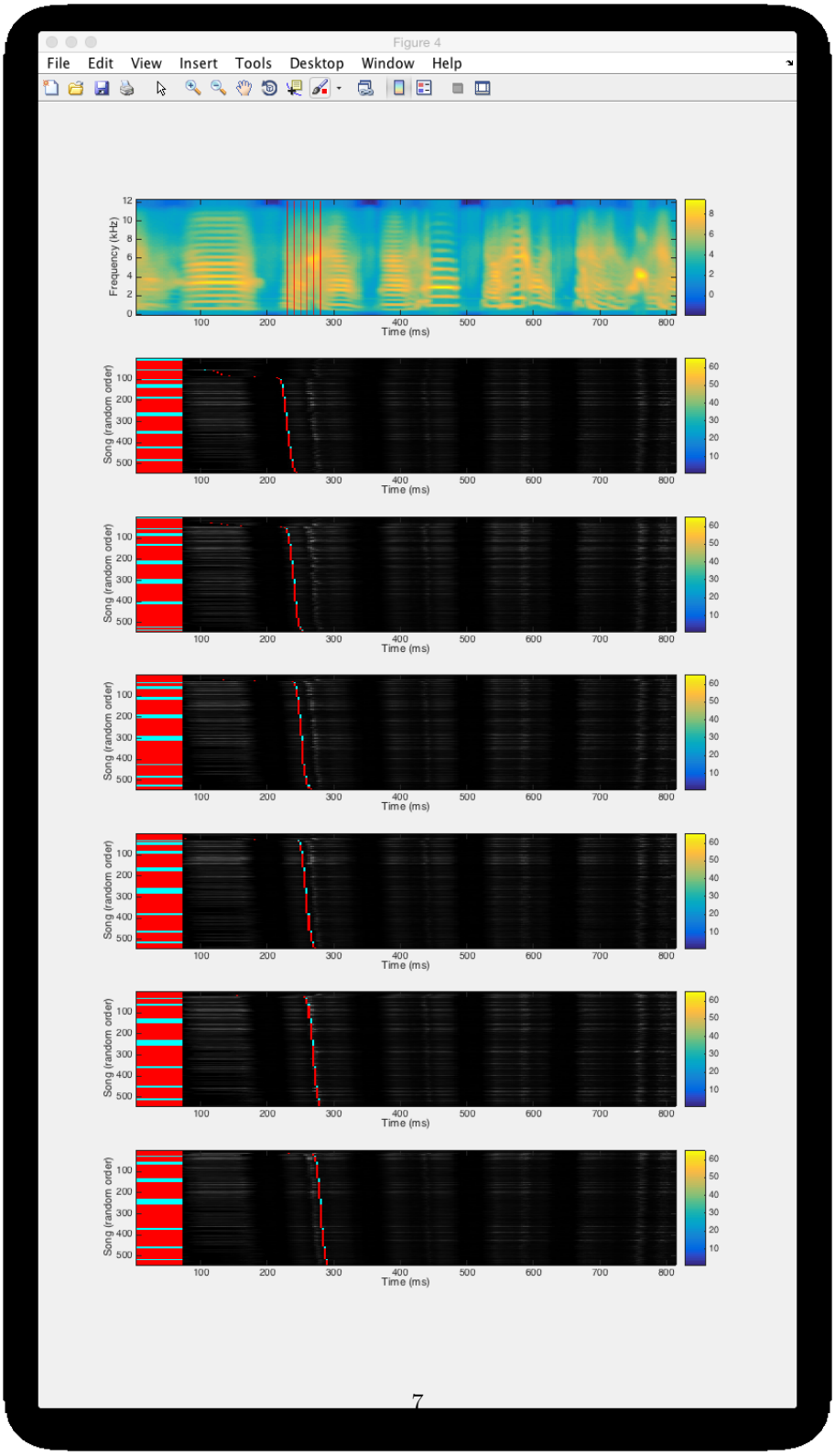

Figure 1: Top plot: a spectrogram made by averaging over 500 songs. The 6 target syllables, spaced 10ms apart, are marked by red lines. Below, each plot shows detection events for the corresponding syllable. The bar on the left is the same width as the detection window, so no detection events can happen 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.

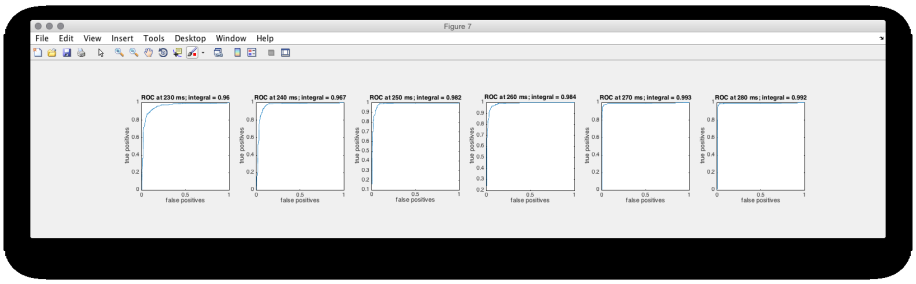

Figure 2: ROC curves for the detection of targets shown in Figure 1. The first one, at the beginning of a syllable, is the most difficult to detect, and as more structure emerges in the current syllable, precision and recall both approach 100%.

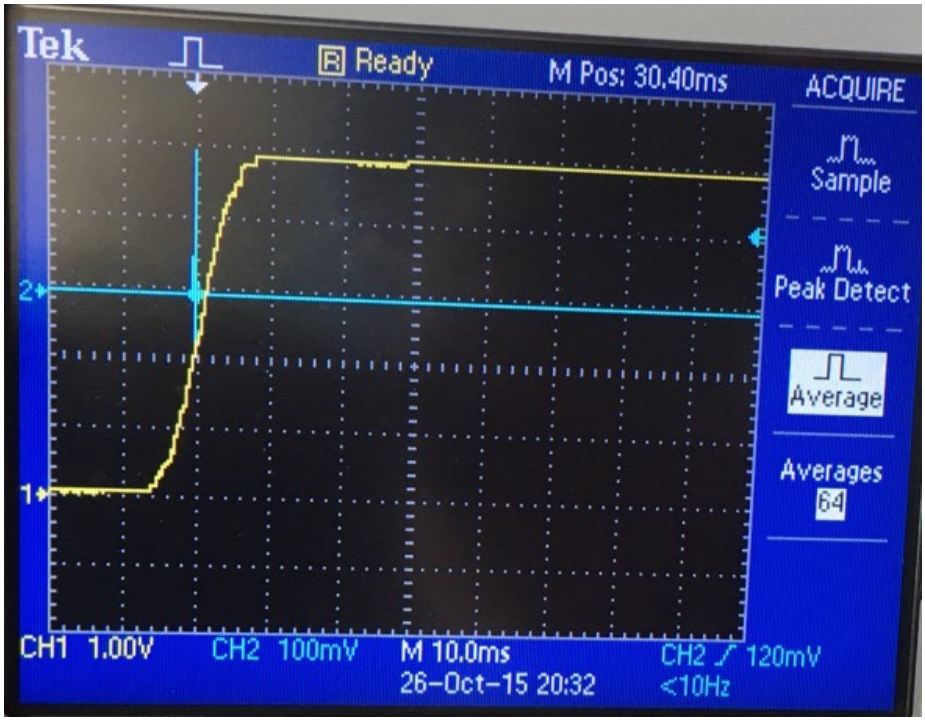

Figure 3: Timing curve for a syllable running a test file generated by training.

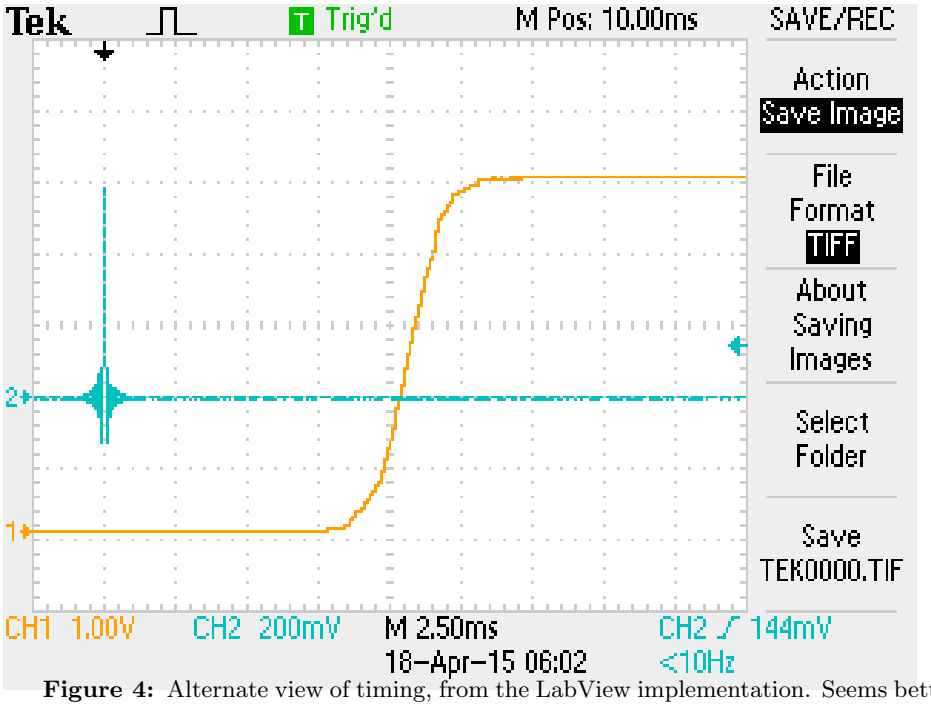

Figure 4: Alternate view of timing, from the LabView implementation. Seems better than that in Figure 3, doesn't it? Hmmm. . . have to discuss with Nathan!

Training the neural network: Our implementation of the syllable detector training code is available under the GNU GPL at `https://github.com/bwpearre/-`

Fix the location!

Runtime: The Swift implementation for executing the trained network...

References