An Optimal Control Model of Zebra Finch Vocalization

Mike Schachter [*]
Helen Wills Neuroscience Institute
University of California, Berkeley
Berkeley, CA 94720
`mike.schachter@gmail.com`

Abstract

In this work, a nonlinear oscillator modeling the syringeal folds of the Zebra Finch is controlled by the state of higher level linear dynamical system. We formulate an optimal control cost function and solution for the learning of bird song.

1 Introduction

First we'll discuss the Zebra Finch vocalization system and it's mathematical formulation, and then the control of that system. All code used to generate figures and results in this paper can be found at:

`http://github.com/mschachter/birdy`

1.1 Zebra Finch Song

The Zebra Finch is an *oscine songbird*, it vocalizes complex sounds in order to defend terroritory and attract mates. A male juvenile Zebra Finch learns a unique song within 90 days of hatching, usually from his father.

The song of a Zebra Finch is organized hiearchically. At the largest time scale is a repeating *motif* that lasts 500-1000ms. The motif is comprised of smaller 20-50ms subunits called *syllables*. The syllables are comprised of sound primitives such as frequency sweeps, noise bursts, and harmonic stacks, collectively called *tones*. Figure 1 illustrates an example of a typical Zebra Finch song. For this project we have focused on the time scale of sound primitives, and more specifically the class of harmonic stacks.

1.2 The Zebra Finch Vocalization System

Like human speech, Zebra Finch song is generated at it's source by oscillations in airflow generated by vibrating vocal cords [3]. Figure 2 shows the Zebra Finch synrinx.

Each syrinx is modeled as a symmetric pair of nonlinear mass-springs. A model for these nonlinear oscillations is taken from [4]:

$$\begin{aligned} \dot{x} &= v \\ \dot{v} &= \gamma^2\alpha + \gamma^2\beta x - \gamma^2 x^3 - \gamma x^2 v + \gamma^2 x^2 - \gamma x v \end{aligned} \tag{1}$$

[*]With a metric ton of help and code from Hédi Soula: `hsoula@gmail.com`

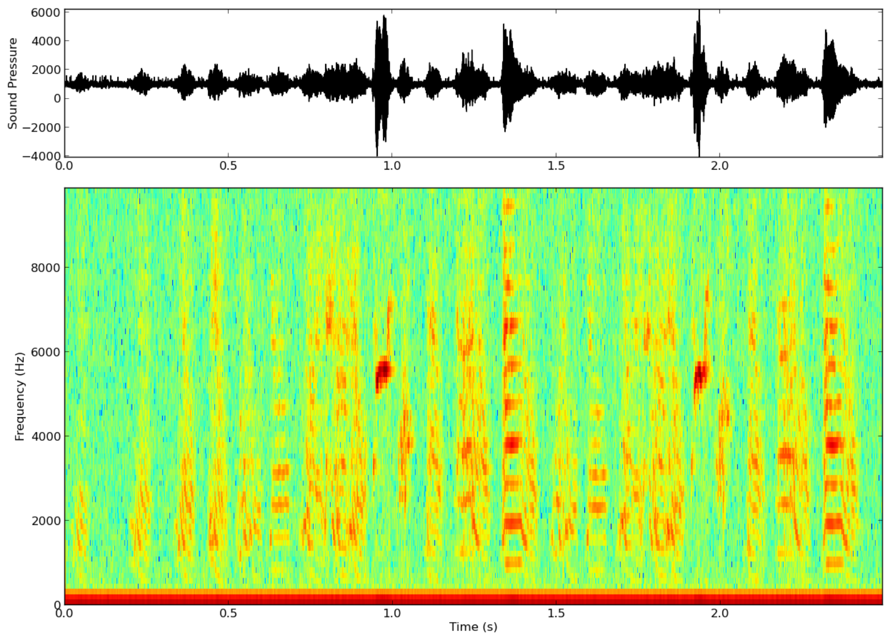


Figure 1: Zebra Finch song, the sound pressure waveform (top) and associated spectrogram (bottom).

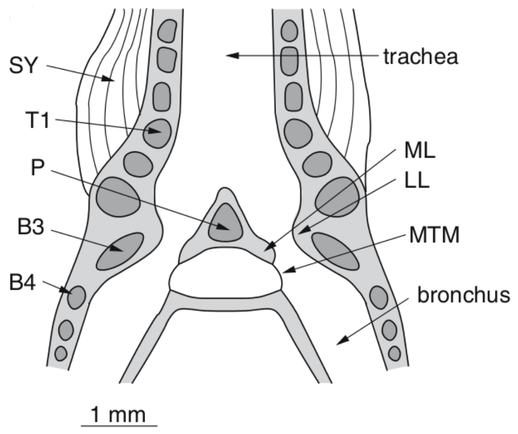


Figure 2: The Zebra Finch synrinx, with two independently controlled vocal cords. Taken from [3]

The control parameters of the model are α and β. γ = 23500 is a constant. The control space has a rich bifurcation structure, with both Hopf, homoclinic, and saddle-node in a limit cycle bifurcations leading to the birth of oscillations, as illustrated in figure 3. The type of bifurcation that occurs can determine the overall spectral content of the sound produced, as noted in [4].

1.3 Control of the Syrinx Model

The goal is to control the parameters α and β in order to produce an observed vocalization. Let $\boldsymbol{\phi}(t) = [\alpha(t)\ \beta(t)]^T$ be the state vector.

Assume that the temporal evolution of $\boldsymbol{\phi}(t)$ is defined by a controlled linear dynamical system:

$$\dot{\boldsymbol{\phi}} = A\boldsymbol{\phi}(t) + \boldsymbol{u}(t)$$

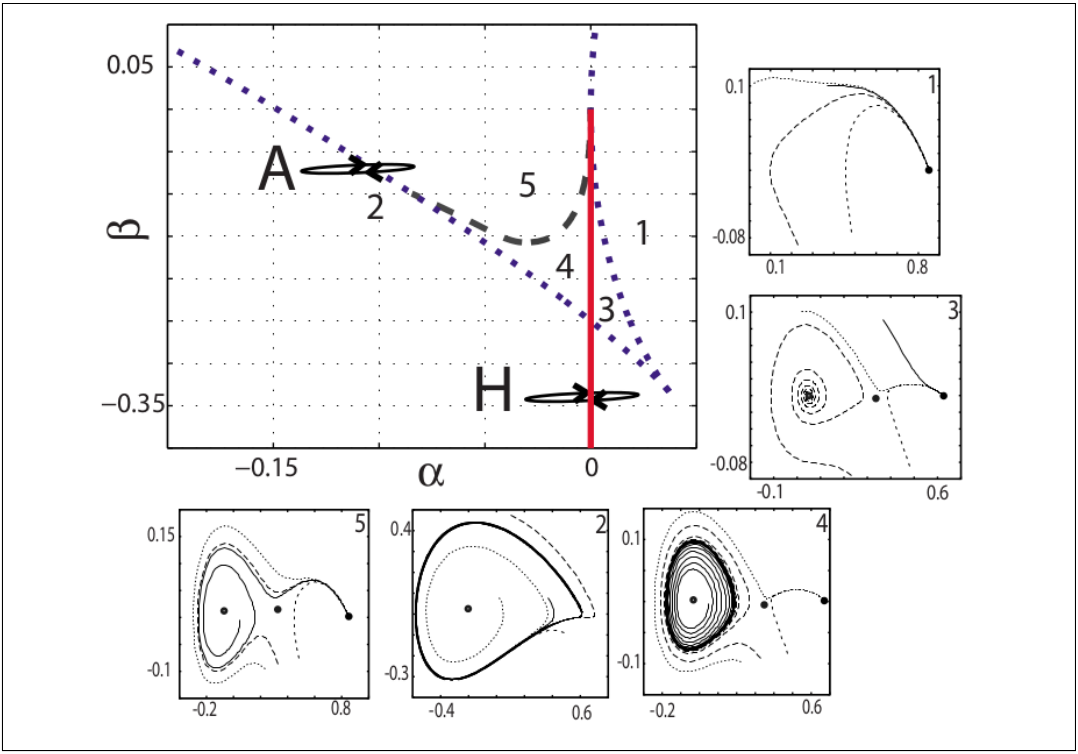


Figure 3: A bifurcation diagram with trajectories, taken from [4]. The red line is a Hopf bifurcation, the dashed black line is homoclinic, and the dotted blue line is a saddle-node bifurcation. Paths A and H represent trajectories that produce Hopf and Saddle-node in a limit cycle bifurcations.

where A is a 2x2 matrix, chosen to make the passive dynamics of the control system decay to some physiologically relevant rest state $\boldsymbol{\phi}_{rest}$.

We discretized time to simplify the analysis. Let $\Delta\tau$ be the time step for simulation of the control system, and define $t_k = k\Delta\tau$. Also define $\boldsymbol{\phi}_k = \boldsymbol{\phi}(t_k)$. Note that the time step for the simulation of the control is significantly larger than that of the oscillator. In practice we used the discrete time map representing the control system as given by the forward Euler step:

$$\boldsymbol{\phi}_{k+1} = \left(A\left(\boldsymbol{\phi}_k - \boldsymbol{\phi}_{rest}\right) + \boldsymbol{u}_k\right)\Delta\tau + \boldsymbol{\phi}_k \tag{2}$$

In optimal control theory [5], a cost function is specified and is to be minimized over time to produce an optimal control law. But what form should the cost function take? We'll make the following assumptions:

1. The control wants to keep the system's instantaneous energy low: $\boldsymbol{\phi}_k^T\boldsymbol{\phi}_k$.
2. The control wants to keep it's own instantaneous energy low: $\boldsymbol{u}^T\boldsymbol{u}$
3. The control wants to produce an instantaneous fundamental frequency that matches that of a stored template.

To elaborate on the last assumption, say we are the given time-varying fundamental frequency of a song syllable that we would like to learn, represented as the function $F(t_k)$. Let:

$$\hat{F} = g(\boldsymbol{\phi}_k) \tag{3}$$

be the function that gives the steady state fundamental frequency for a given control $\boldsymbol{\phi}_k$. The function g can be empirically estimated through simulation and approximated through interpolation. To follow this frequency means to keep the quantity $(F(t_k) - g(\boldsymbol{\phi}_k))^2$ small.

Given these assumptions, the cost at time t_k of applying control $\boldsymbol{u}$ is:

$$\ell_k\left(\boldsymbol{\phi}_k, \boldsymbol{u}\right) = \boldsymbol{\phi}_k^T \boldsymbol{\phi}_k + \boldsymbol{u}^T \boldsymbol{u} + \left(F(t_{k+1}) - g(\boldsymbol{\phi}_{k+1})\right)^2$$

Let N be the number of time points we want to control, and let $\boldsymbol{\pi}_j = \{\boldsymbol{u}_j(\boldsymbol{\phi}_j), ..., \boldsymbol{u}_N(\boldsymbol{\phi}_N)\}$ be the control law applied from time j to time N. Note that the control law is a sequence of functions! Each control uses feedback information about most recently observed state. The total cost of applying a control law $\boldsymbol{\pi}_0$ given initial state $\boldsymbol{\phi}_0$ is:

$$v\left(\boldsymbol{\phi}_0, \boldsymbol{\pi}_0\right) = \sum_{k=1}^{N} \ell_k\left(\boldsymbol{\phi}_k, \boldsymbol{u}_k\right)$$

1.4 Dynamic Programming Solution to Optimal Control

The optimal cost-to-go is a function that represents the total cost from a time j to time N given that the optimal control function is applied:

$$v_j^*\left(\boldsymbol{\phi}_j\right) = \min_{\boldsymbol{\pi}_j} \sum_{k=j}^{N} \ell_k\left(\boldsymbol{\phi}_k, \boldsymbol{u}_k\right)$$

Dynamic programming is typically used to solve for the optimal cost-to-go [1]. When applying DP, we work backwards in time, creating a recursive algorithm that determines the functional form of the entire optimal control policy $\boldsymbol{\pi}_0^*$. To illustrate this, we are going to take $N = 2$ with a specified initial condition $\boldsymbol{\phi}_0$.

First the solution must be found for $N = 2$:

$$v_2^*\left(\boldsymbol{\phi}_2\right) = \min_{\boldsymbol{u}_2} \ell_2\left(\boldsymbol{\phi}_2, \boldsymbol{u}_2\right)$$

A straighforward but expensive way to solve would be to perform a search over a grid of values for $\boldsymbol{\phi}_2$. For each value on the grid, we then will use gradient descent to minimize $\ell_2\left(\boldsymbol{\phi}_2, \boldsymbol{u}_2\right)$ with respect to $\boldsymbol{u}_2$. The end result is that we have a lookup table of values, giving us a function $\boldsymbol{u}_2(\boldsymbol{\phi}_2)$, and a function $v_2^*\left(\boldsymbol{\phi}_2\right)$.

The optimal cost-to-go at time $N - 1 = 1$ is

$$v_1^*\left(\boldsymbol{\phi}_1\right) = \min_{\boldsymbol{u}_1} \left[\ell_1\left(\boldsymbol{\phi}_1, \boldsymbol{u}_1\right) + v_2^*\left(\boldsymbol{\phi}_2\right)\right] \tag{4}$$

We know the functional form of $\ell_1\left(\boldsymbol{\phi}_1, \boldsymbol{u}_1\right)$ and the numerical form of $v_2^*\left(\boldsymbol{\phi}_2\right)$. We also know that:

$$\boldsymbol{\phi}_2 = \left(A\boldsymbol{\phi}_1 + \boldsymbol{u}_1\right)\Delta\tau + \boldsymbol{\phi}_1$$

so we can again use gradient descent for a grid on $\boldsymbol{\phi}_1$ to create a lookup table for $\boldsymbol{u}_1(\boldsymbol{\phi}_1)$ and $v_1^*\left(\boldsymbol{\phi}_1\right)$. We now have a functional form for the optimal control policy: $\boldsymbol{\pi}_0^* = \{\boldsymbol{u}_1(\boldsymbol{\phi}_1), \boldsymbol{u}_2(\boldsymbol{\phi}_2)\}$

Now going foward in time, with the knowledge of a start state $\boldsymbol{\phi}_0$, we can compute $\boldsymbol{\phi}_1$, and then the optimal control $\boldsymbol{u}_1\left(\boldsymbol{\phi}_1\right)$. Given $\boldsymbol{u}_1$, we can then compute $\boldsymbol{\phi}_2$ and then $\boldsymbol{u}_2$. The technique is generally applicable for any N.

1.4.1 A More Efficient Implementation

Solving the dynamic programming problem at a time t_k requires finding a function $\boldsymbol{u}_k(\boldsymbol{\phi}_k)$ that minimizes $v_k(\boldsymbol{\phi}_k)$ for any $\boldsymbol{\phi}_k \in \mathcal{P}$, where $\mathcal{P}$ is a pre-specified set of realistic values. This requires us to uniformly sample from $\mathcal{P}$ and solve an optimization problem for each sample, a very computationally expensive prospect!

To reduce the computational burden, we first model the function $\boldsymbol{u}_k(\boldsymbol{\phi}_k)$ as a linear combination of M radial basis functions $\{\psi_1, ..., \psi_M\}$. Each basis function takes the form:

$$\psi_i(\boldsymbol{\phi}) = exp\left(\frac{\|\boldsymbol{\phi} - \boldsymbol{\theta}_i\|^2}{\sigma_i^2}\right) \tag{5}$$

where $\boldsymbol{\theta}_i$ is the center of the basis function, σ_i is the bandwidth of the basis function. Each basis function is multiplied by a coefficient $\boldsymbol{c}_i = [c_{i1}\ c_{i2}]^T$. Let $\boldsymbol{C} = [\boldsymbol{c}_1, ..., \boldsymbol{c}_M]$ be the $2xM$ matrix of coefficients, the control is computed for a given $\boldsymbol{C}$ as:

$$\boldsymbol{u}_k(\boldsymbol{\phi}_k, \boldsymbol{C}) = \sum_{i=1}^{M} \boldsymbol{c}_i \psi_i(\boldsymbol{\phi}_k) \tag{6}$$

We then formulate a single optimization problem to minimize the sum of cost across all points in $\mathcal{P}$ for a given $\boldsymbol{C}$:

$$\boldsymbol{C}^* = \operatorname*{argmin}_{\boldsymbol{C}} \sum_{\boldsymbol{\phi} \in \mathcal{P}} v_k(\boldsymbol{\phi}, \boldsymbol{C}) \tag{7}$$

The optimal control at time t_k is then given as $\boldsymbol{u}_k(\boldsymbol{\phi}_k, \boldsymbol{C}^*)$.

To create an initial guess for the optimization problem, each $\boldsymbol{c}_i$ is initialized independently by finding the minimum cost control at $\boldsymbol{\theta}_i$, the center of basis function i:

$$\boldsymbol{c}_i^0 = \operatorname*{argmin}_{\boldsymbol{c}_i} v_k(\boldsymbol{\theta}_i, \boldsymbol{c}_i) \tag{8}$$

2 Methods and Results

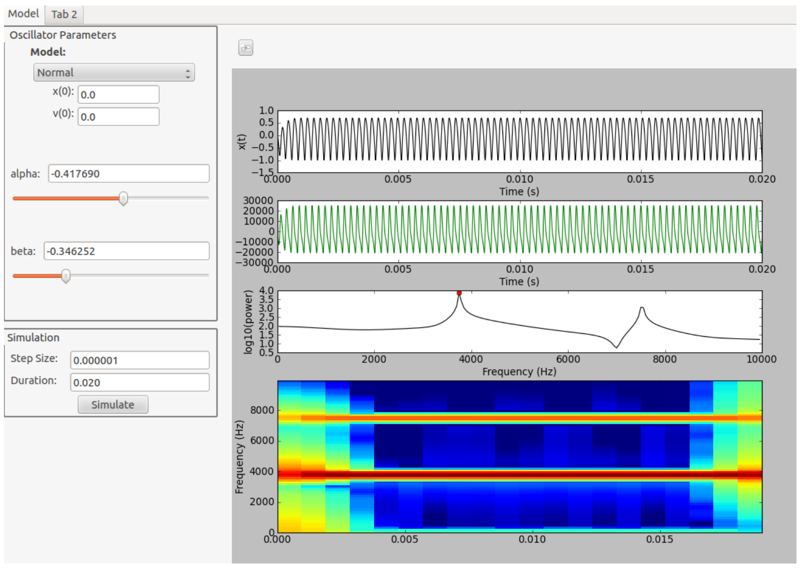
2.1 Simulation

Equation (1) was implemented in C++ and utilized the GNU Scientific Library. It was accessed and simulated using Cython/Python. The integration method used was an 8th order Runga-Kutta method [2] with a step size of 1 μs. A GUI was created using PyQT in order to examine the dependence of oscillations with the control parameters, as shown in figure 4.

2.2 Mapping Controls to Fundamental Frequency

Equation (3) maps a control $\boldsymbol{\phi}$ to a steady-state fundamental frequency $\hat{F}$, and required simulation to determine. First, a grid of points was constructed that spanned $\alpha \in [-1.25, 0.05]$ and $\beta \in [-1.10, 1.10]$, with a spacing of 0.05. For each point in the grid, the oscillator was simulated with a zero initial condition and the control specified by the grid point, for 15 ms.

The power spectrum of the resluting oscillation was taken and the fundamental frequency $\hat{F}$ was identified as it's maximum. This produced samples for a real-valued function of two variables. These samples were fit with a radial basis function implementation from SciPy with a bandwidth 0.8 and Gaussian kernel. A RBF kernel was placed at each of the sample points, and allowed for a continuously-varying representation of equation (3). The sampling and it's RBF interpolation are shown in figure 5.



Model: Normal

x(0): 0.0

v(0): 0.0

alpha: -0.417690

beta: -0.346252

Step Size: 0.000001

Duration: 0.020

Figure 4: The UI for the simulation environment.

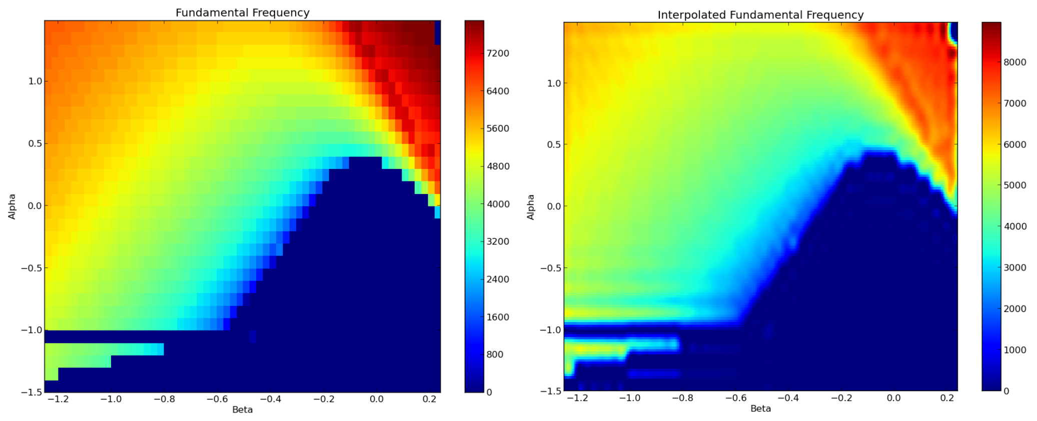


Figure 5: The simulated steady-state fundamental frequency as a function of control (left), and it's interpolation by radial basis functions (right).

2.3 Optimal Control

The first step in solving the optimal control problem was to choose the passive dynamics of the system defined by (2). We chose the following matrix:

$$A = \begin{bmatrix} -1000 & 0 \\ 0 & -900 \end{bmatrix}$$

and specified $\boldsymbol{\phi}_{rest} = [-0.30\ 0.30]^T$, a value where only one stable fixed point exists and no oscillations can occur. The passive dynamics had the phase plot specified in figure 6. The values of A were specified such that both α and β decayed to their rest points within 10ms.

Once the passive dynamics were selected, we applied the algorithm described in section 1.4.1. First, we created a grid of points for $\boldsymbol{u}$ that spanned $[-1, 1]X[-1, 1] \subset \mathbb{R}^2$, with a spacing of 0.25. Each point in the grid was a center $\boldsymbol{\theta}_i$ as listed in equation (5). The bandwidths were chosen uniformly as $\sigma_i = 0.30$.

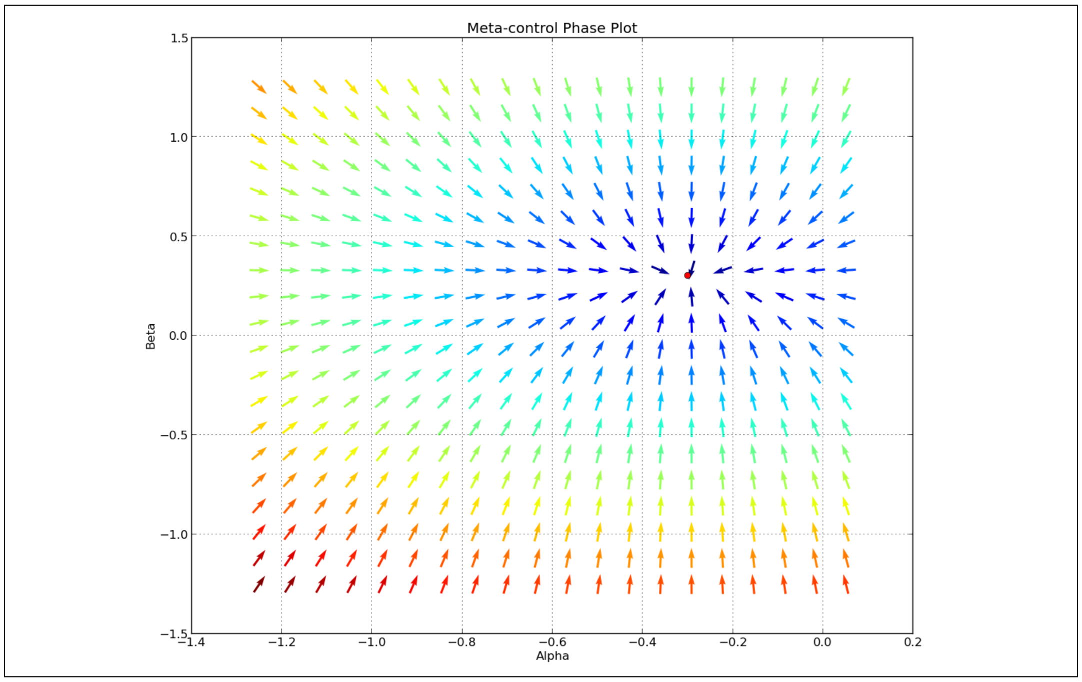


Figure 6: A phase plot for the passive dynamics of the control system. Color indicates magnitude, red is large, blue is small.

In the next step, the desired sequence of fundamental frequencies was specified with a sample spacing of $1ms$. The last frequency in the sequence was selected, and the dynamic programming optimization began.

The goal of the DP algorithm, for a given time step, was to determine the matrix $\boldsymbol{C}^*$ of equation (7). $\boldsymbol{C}^*$ was then used in equation (6) to determine the optimal control.

Because the optimization problem of (7) is (probably) difficult and (possibly) nonconvex, and definitely expensive, the first step was to find a good initial guess for each element of $\boldsymbol{C}^*$. So we evaluated a locally-optimal value at each center, solving the optimization problem specified by equation (8). This was accomplished using gradient descent, with an initial guess of $\boldsymbol{c}_i^0 = [0\ 0]^T$. All gradients were computed using a finite-difference approximation.

Once $\boldsymbol{C}$ was initialized, we then ran the full optimization of equation (7). We only let it run for 10 iterations, with the intention of actually getting at least one result before the due date of this project!

The technique above was applied to the last time point. Using dynamic programming, we then solved for the second-to-last time point, applying a recursion to the cost similar to that shown in equation (4). The time for optimization increased linearly as a function of how far back in time we went, as shown in figure 7.

Given time constraints, we only had the opportunity to apply the equation to a single example, that of a constant desired fundamental frequency of 3000Hz for 10ms. The total time to solve this problem, computed as the cumulative sum of the times in figure 7, was 314 minutes.

For each time point, we determined the form of the function specified in (6). One such example of this function is shown in figure 8. Some explanation is in order. The upper left is the optimal cost-to-go function $v_{10}^*(\boldsymbol{\phi})$. It shows that the most expensive states are those that produce a mismatch of the fundamental frequency (see figure 5).

The lower left panel shows the value of u_0 for a given value of $\boldsymbol{\phi}$. Strongly negative values of u_0 will push α to be significantly more negative. The control at this time point seems to have the ability to push α to two different regions of the space, to the extreme left and to the extreme right. Given that time has run out to check for bugs, we can only hope that this makes sense...

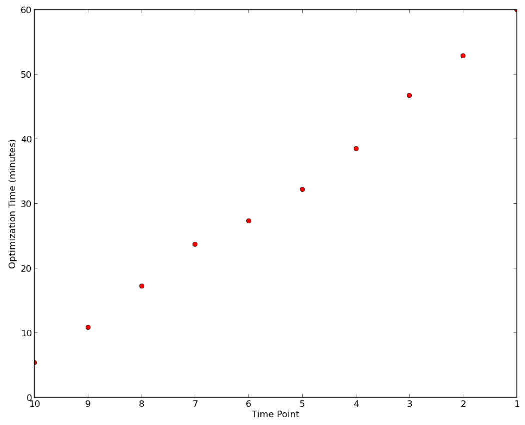


Figure 7: The computation time for optimization at each time point.

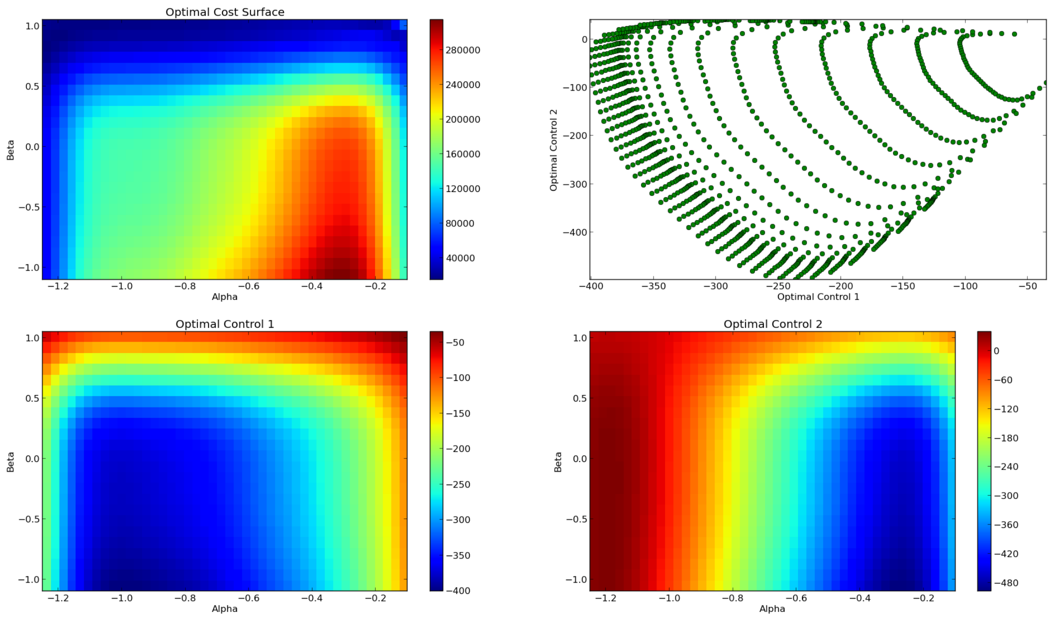


Figure 8: Upper Left: the optimal cost-to-go function at time point $N = 10$. Upper Right: the optimal control vector elements plotted against eachother. Lower Left: optimal control 1 as a function of ϕ. Lower Right:optimal control 1 as a function of ϕ.

The lower right panel shows a tendency for the control u_1 to push β into a negative range, dependent on the value of α. It's probably best not to over-interpret this figure in the absence of a more rigorous analysis of bugs. The upper right panel shows u_0 vs u_1, showing a very much nonlinear relationship between the two.

You may ask yourself at this point, "does it work"? The answer for now is "not yet". Figure 9 is a spectrogram of a 10ms simulation for the optimal control law applied to a given initial condition. There's plenty of power in the 3000Hz range, but clearly the fundamental frequency is closer to 1000Hz. More debugging is necessary!

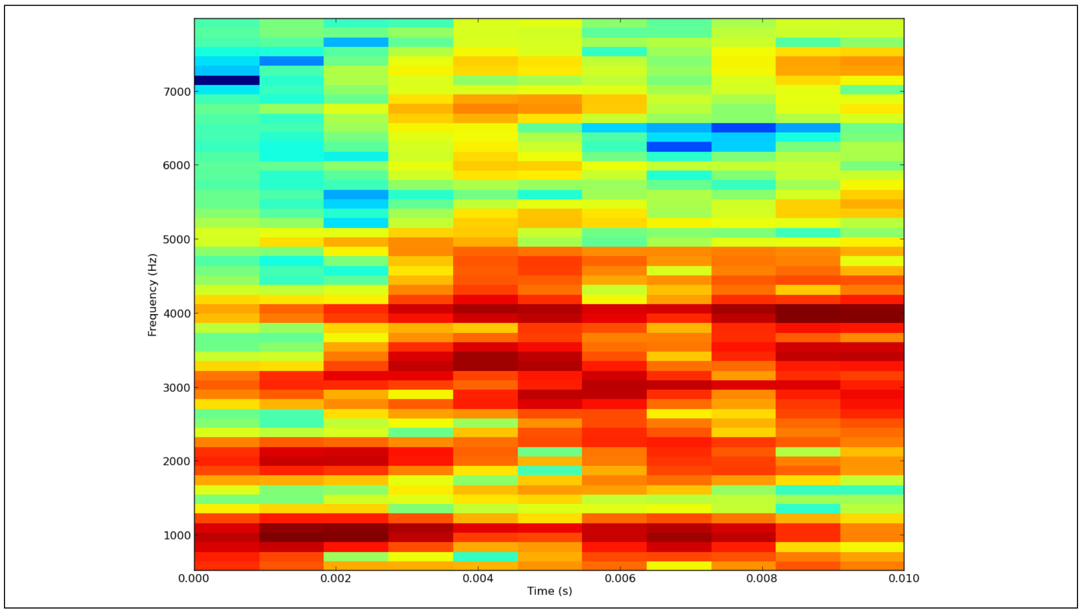


Figure 9: A spectrogram for an optimally-controlled vocalization.

3 Conclusion

This work was an exciting foray into the world of nonlinear oscillators and optimal control theory. It's a work-in-progress, setting the stage for my thesis work, which will require the generation of synthetic Zebra Finch syllables.

The basic idea for future work will be to identify the "motor primitives" of Zebra Finch song. These motor primitives are the motor commands, i.e. the feedback control functions $\boldsymbol{u}_k(\boldsymbol{\phi}_k)$, that dictate the spectral content of a song for durations of 5-10ms.

The major bottleneck right now is computation. The dynamic programming algorithm takes a long time. There are many ways to speed it up. Some ways to do so may include coming up with a better parameterization for the control functions than RBFs, faster generation of initial guesses, implementation of an analytic gradient (if possible), and an optimization algorithm such as scaled conjugate gradient descent that should converge faster.

Note that there was no actual feedback used in this work. The Zebra Finch auditory system cannot provide sensory feedback in less than 20-50ms, which is often longer than the duration of the syllable. Instead of using feedback, we predicted the fundamental frequency through an emprically estimated function (equation (3)). We used what is called a *corrolary discharge*, a copy of the motor command, and ran it through a *forward model*, a model that takes a motor command and produces a prediction of the output state. By doing so we formulated the error in the predicted fundamental frequency, and tried to minimize that error.

The use of forward models and corrolary discharge is very general and has been used to implement a Zebra Finch learning model in [6]. We hope to extend this methodology to sound features that span longer periods of time. The idea would be to start building associations between the properties of a syllable at several time scales, and in doing so actually simplify the function form of the feedback control laws. We can begin to explore the various timescales that make learning easier, and explore their representation by neurons in the Zebra Finch auditory cortex.

Acknowledgments

Thank you to Hédi Soula, who created a C++/Python implementation of this model in our lab and has provided ideas, expertise, and feedback on this project. A thousand thanks to Jim Crutchfield, who facilitated an excellent NCASO course and offered lots of advice on this project.

References

[1] Bertsekas, D. P. *Dynamic Programming and Optimal Control*. Athena Scientific, 2000.

[2] Dormand, J., and Prince, P. A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics 6* (1980), 19–26.

[3] Mindlin, G. B., and Laje, R. *The Physics of Birdsong*. Springer, 2005.

[4] Sitt, J. D., M, E., Arneodo, Goller, F., and Mindlin, G. B. Physiologically driven avian vocal synthesizer. *Physical Review E 81, 031927* (2010).

[5] Todorov, E., and Jordan, M. I. Optimal feedback control as a theory of motor coordination. *Nature Neuroscience 5 no 11* (2002).

[6] Troyer, T. W., and Doupe, A. J. An association model of birdsong sensorimotor learning i. efference copy and the learning of song syllables. *J. Neurophysiol 84* (2000), 1204–1223.