ECE 5557 - Final Report

Sound Controller

Rayan El Helou

April 28, 2015

Contents
Project Outline ................................................................. 2
Speaker-Microphone Setup (Physical Dynamics) ........................................ 3
  Speaker ............................................................................. 3
  Microphone .................................................................... 3
Output Measurements ........................................................................ 4
  Amplitude Estimation ....................................................... 4
  Frequency Measurement ..................................................... 5
Open-Loop Plant .................................................................. 5
  Input - Output Definition ................................................ 5
  Open Loop Control and System Identification ............................... 6
MIMO Controller with variable set-points ......................................... 7
APPENDIX ........................................................................... 8
  Simulink Models ............................................................... 8
  Test Plots and Results ....................................................... 11
  MATLAB® Functions ....................................................... 17
Project Outline

The objective of this project is to create a computer that can ‘sing’ a mathematically defined waveform (e.g. sine wave) using a single speaker, and measure its own voice using a single microphone to attempt to change it to achieve desired features. One of the main challenges in this project is that the system is MIMO (multi-input multi-output). For example, by changing the frequency sent to the speaker, the microphone will detect varying amplitudes during that change.

A sine wave may be defined by its amplitude and frequency. Those two features, as described above may be initially defined by the user. The outline of the system is shown below.

- **Speaker - Microphone setup (Physical Dynamics):**
  At the start of the simulation, the computer will start sending the waveform to the speaker and receiving the physical sound through a microphone, performing both actions at a specified sampling rate. This is the only physical non-computational project portion.

- **Output Measurements:**
  The microphone data collected is sent in clusters every 20ms to two MATLAB® functions. One algorithm measures, or estimates, the frequency, while the other does so for the amplitude.
  - Hence, every 20ms, the system has two outputs: frequency and amplitude.
  - This restricts the ‘control frequency’ to 50 changes per second.

- **Open-Loop Plant:**
  The inputs of the system are the multiples by which one would like to scale the frequency and amplitude. For example, if we currently hear a 5 sin(3t) waveform and we would like to hear a 10 sin(9t) waveform, then the two control inputs for amplitude and frequency ought to be 2 and 3 respectively, since these are the amounts by which each output was scaled.
  - Basically, an increase in control input of each feature of sound causes a proportional increase in the output feature. Is there a chance for one feature to affect the other?

- **MIMO controller with variable set-points:**
  The two outputs are fed into controllers with or without coupling along with desired values of amplitude and frequency into two controllers, where the result of the controllers is the control input of the plant.

Obligations:

MATLAB® and Simulink were used to complete this project. I would like to thank Prof. K.M. Passino, Prof. L.C. Potter, L.F. Giraldo Trujillo and Y. Yang for their help and support on this project.
What this project is not:

This project is intended to make the speaker attempt to speak and hear itself without any human input. Any of such would be considered a disturbance. This project is not a way to auto-tune one’s voice, since that would require two speakers: the human’s throat and an electronic speaker. One may only define a set-point for the frequency and amplitude of the waveform at any given point in time. Otherwise, the computer will act on its own to adjust itself. There will be no sound - or noise - filtering in this project since it has little effect on the frequency control, but since it has some effect on the amplitude control, we will ignore it and consider it an obstacle or challenge to further enforce our will to apply better control theories in attempt to fix it.

Using the Appendix

Please refer to the appendix for reference to the labelling system that will be used in the Report.

Speaker-Microphone Setup (Physical Dynamics)

Speaker

In order to send a signal to a physical speaker in Simulink, one must do so in segments of data at a specified sample rate. When a physical speaker receives a segment of data from Simulink, it takes each element and assigns it to the displacement of a physical diaphragm (inside the speaker), and the waiting time between each data point assignment and the next is defined as the sampling time, or the reciprocal of the sampling rate. The continuous displacement of the physical diaphragm is what causes sound to be heard since it pushes the molecules (air) in its vicinity to create most likely a periodic wave that propagates to one’s ear and is detected by the brain as sound.

By defining the buffer size, the size of the data segment, and the sampling rate one is essentially telling the physical speaker how often to receive the data and how quickly to play it respectively. For the purposes of this project, the sampling rate of the speaker found in Mod. 4 of the Appendix is set as 8000 Hz. The buffer size is set as 1024. Mod. 4 also shows a gain for sampling which is basically receiving data and sampling it at the rate which the speaker desires. There is also a block labelled Unbuffer which collects the first-entered piece of data from the buffer instantaneously to avoid waiting for the buffer to fill up with data to play the sound. This makes the flow of sound smoother and more pleasant to hear.

INPUT: Continuous- or discrete-time data

OUTPUT: Physical Sound

Microphone

Please refer to Mod. 5 of the Appendix. Physically speaking, the microphone performs the reverse action of the speaker; its diaphragm gets displaced instantaneously and by detecting the displacement at a fixed sampling rate, it would be ideally recording the data that the speaker had
sent. The obvious reason it is not ideal in any project is because the propagations of the waves in air (or any other medium) are always subject to some disturbance or dampening.

Again, the buffer size is how often the device compiles the data, which may be automatically determined by the microphone. What the user rather sets in Mod. 5 is the frame size, since after the microphone compiles the data it outputs it in segments, the constant length of which is defined as the frame size. Similar to the speaker, the sampling rate of the microphone is 8000 Hz. Its frame size is 512. As seen in Mod. 5, there is an Unbuffer - Buffer system which is similar to that of the speaker’s but in reverse order due to data output rather than input. The buffers’ sizes are set to be the same as the microphone’s frame size.

**INPUT:** Physical Sound

**OUTPUT:** Continuous- or discrete-time data

**Output Measurements**
The golden blocks in Mod. 5 represent two MATLAB® functions in which the live data is collected and algorithms are applied to measure/estimate the amplitude and frequency every 20 milliseconds. This is a crucial step in feedback control, especially when attempting to reach set-points or desired outputs. This means that control decisions may be made 50 times a second.

Please note that the inputs of the golden blocks are fed through buffers (see Mod. 5), which have overlaps set to 352. Frame size - Overlap = 512 - 352 = 160. The golden blocks would only have to wait 160 data points to make new calculations. At a sampling rate of 8000 Hz (data points per second), this would mean that a new pair of output is calculated every 160/8000 or every 20 milliseconds. That’s where the control time comes from. One may make the overlap larger to increase the control rate, but when I attempted that (on my Windows 8 PC with Intel core i7 processor) the speaker would produce ‘hiccups’ in sound.

As for the measurements themselves, please find the used MATLAB codes attached in the Appendix, but they behave as follows.

**Amplitude Estimation**
To estimate the amplitude of a segment of data, I chose to first divide it into four segments. The maximum of the absolute value of each is taken and the four maximums of the divided data are noted. Their mean is then considered to be the estimated amplitude. This algorithm forces the estimated amplitude to act as an outline to the signal which is a functionality that we desire, but there is a time delay of less than 50 milliseconds due to computation time. As an example to prove it working, I used the algorithm to estimate the amplitude and track a signal. The signal used is a recording of my voice while I say “Hello, my name is Rayan.” Please see Fig. 1 in the Appendix for the results and note that for each word I said, there is almost a pulse of sound. The units are not relevant to our project, so I do not rescale the microphone’s raw data after
MATLAB® detects it. Also note that the estimated amplitude curve in Fig. 1 looks like a stairs function; that is due to the fact that the control time is 20 milliseconds as mentioned before.

**Frequency Measurement**

To estimate the frequency, some assumptions must be made.

- First, the type of signal that we are measuring might have many frequencies, so we must perform a Fourier analysis to detect which of the many has the highest magnitude. The frequency with the highest magnitude in a spectrum of frequencies is to be considered the output or measured frequency in the system.
- Second, to avoid aliasing, accept measurements of frequencies only under half the sampling rate, i.e. less than 4000 Hz. I chose the range of accepted frequencies to be from 200 Hz to 3200 Hz.
- Third, if the waveform being sent to the speaker is a square wave, the microphone will detect it to have multiple frequencies and in many situations, mistake one dominant frequency for the other if the speaker does not perform well in playing the square wave. Hence, it’s best to stick with sine waves for the purposes of practicing Multi-input Multi-output control rather than signal processing.

Please refer to the appendix for the MATLAB® function to measure frequency. A function obtained from the online community has been included; it is to find the discrete-time Fourier transform of a segment of data. First the data labelled \( y_{mic} \) in Mod. 5 is transformed into frequency domain using the attached DTFT function. Next, the indices of 200 Hz and 3200 Hz are located to chop off the Fourier transform into the desired region only. The magnitude function of the transform then has its maximum located within the specified range of frequencies. That location represents the dominant or output frequency.

Fig. 2 shows how entering sine waves with various frequencies into the speaker every 3 or 4 seconds will be read by the frequency measurement algorithm. Notice how there is a delay of about 1 second. That is the delay for the speaker to send the manually changed data in Simulink. However, the frequencies are detected accurately except of course when the speaker sometimes skips certain pieces of data, which I believe is due to issues in the speaker itself.

**Open-Loop Plant**

**Input - Output Definition**

In order to manually control the behavior of the audible waveform, there are two main features that need to be considered: time and space, or frequency and amplitude. To control the amplitude, one may simply multiply the signal by a scalar. However, to control the frequency, one needs to stretch the time by also multiplying by a scalar if there is access to the clock that is running the signal, but in case all the signal provides is data points at every moment in time, one needs to perform a complex pitch shift phase vocoder which causes delay in response. For the
purposes of this project, which is more controls based rather than signal processing based, we will assume access to the clock that’s running the signal and scale it up or down to change the frequency. Mod. 3 shows a simply sine wave as the original signal with an amplitude of 0.1 and a frequency of 700 Hz. The output of this subsystem is the signal itself, and the input is the clock which sets the pace of time. Mod. 3 is a subsystem represented in Mod. 2 with the red block as part of the Open-Loop Plant. This plant model has two inputs, each of which are entered in product blocks as seen in Mod. 2. The first input is labelled \( f_{scale} \) and is the amount by which the original frequency is multiplied - since the blocks of frequency and clock are multiplied inside the sine wave subsystem (Mod. 3), multiplying by the clock is the same as multiplying by the frequency. The second input of the plant is labelled \( A_{scale} \) and is the amount by which the signal is multiplied, hence by which the amplitude is multiplied.

As described in the previous section for amplitude estimation and frequency measurement, the algorithms are ready in place to detect the outputs of the model plant that are the frequency of the audible waveform and its amplitude. Therefore, this system is a MIMO (Multi-Input Multi-Output) system.

**INPUT 1:** \( f_{scale} \) or \( u_f \). The amount by which the frequency is scaled.

**INPUT 2:** \( A_{scale} \) or \( u_A \). The amount by which the amplitude is scaled.

**OUTPUT 1:** \( f \) or \( y_f \). The measured frequency.

**OUTPUT 2:** \( A \) or \( y_A \). The estimated amplitude.

**Open Loop Control and System Identification**

By leaving the original frequency and amplitude from Mod. 3 as they are - 700 Hz and 0.1, we can provide the system first with varying \( f_{scale} \) values and constant \( A_{scale} \) values to see the effect of the frequency control input, then we may reverse the control to test the effect of the amplitude control. Please refer to Mod. 1 that is the overall closed-loop MIMO plant model, but for this section, both switches labelled Frequency Control Switch and Amplitude Control Switch are turned to the left to open the loop.

**NO CONTROL CASE:** By setting \( f_{scale} \) or \( A_{scale} \) as 1, **not** 0, that is the case of no control correspondingly.

First, let’s keep \( f_{scale} \) fixed at 1 and ramp \( A_{scale} \) from 1 to several values to test the behavior of the system. Please see the results of this experiment in Fig. 3 of the appendix.

The original signal’s amplitude was set to 0.1, and there seems to be a dampening of about 0.01 between the data sent to the speaker and that received by the microphone. This leads to a total gain of about 0.001 from the input \( u_A \) to the output \( y_A \). However, there is a saturation to how loud the microphone detects data when the frequency is at 700 Hz. The second and fourth subplots of Fig. 3 show that the microphone saturates at 0.01 amplitude. Due to noise, its lower
saturation bound is about 0.002. So is it the fault of the microphone or of the speaker? You will have to trust me that I heard no increase in volume as the speaker data was increasing. This means that the speaker receives the data (subplot 1 of Fig. 3) and saturates it. The final relationship between input and output is as follows:

\[ u_f = 1 \rightarrow A(t) = s(0.001|u_A(t-1)|), s(x) = \begin{cases} 
0.002, & x \leq 0.002 \\
0.002 < x < 0.01 & \\
0.01, & x \geq 0.01
\end{cases} \]

Interestingly enough, the upper saturation limit of the speaker has nothing to do with safety restrictions set by the manufacturer, but with the frequency that’s being sent. As an example, let’s repeat the experiment by controlling the input frequency control to 2 instead of 1. Fig. 4 shows that not only did the upper saturation limit rise by increasing the frequency, but the amplitude doubled at every point in time when the speaker was turned on. As a piece of evidence, observe the difference between Fig. 3 and Fig. 4. The data sent to the speaker is the same but that received from the microphone is doubled.

Ignoring saturation, disturbance and delay for now, a good model for the amplitude is as follows:

\[ A(t) = K_A|u_f(t) \cdot u_A(t)| \]

Second, for the control of the frequency, the amplitude will this time be kept constant and the frequency control input will be stepped up every second. Fig. 5 shows a very simple proportional relationship between the control input and frequency output whose gain is defined by the original frequency - 700 Hz in our case. From the previous sections, it was explained how the frequencies are taken only between 200 Hz and 3200 Hz, so assuming we are dealing with all values in between, then model for the frequency is as follows:

\[ f(t) = f_{original} \cdot u_f(t) \]

**MIMO Controller with variable set-points**

Let’s say we would like to reach a desired set-point for the frequency of 1400 Hz. We must therefore apply a control signal of \( u_f = 2 \) to double the original frequency. However, the system is not allowed to know the original frequency, so it must rely on output measurements in order to decide that the control signal must settle at 2. The challenge is that if the system immediately multiplies the output by 2, the new output will be 1400 Hz, but then it will think that it should multiply by 1 since it already reached the desired value. This will cause oscillations that do not end.

Therefore, we use an integrator as shown in Mod. 6 to detect the difference rather than quotient between the current and desired frequencies. The error is sent to an integrator controller (PID without derivative or proportional sums) whose gain is tuned so that the system will have almost
no oscillations and as least settling time for the frequency as possible. The reason there is no derivative control is so that disturbances and noise do not make the system go unstable.

The same theory has been applied to control the amplitude and the combined de-coupled controlled has the results shown in Fig. 6. De-coupled means that we have not yet chosen to take feedback from frequency to affect amplitude or vice versa.

**Note:** Mod. 1 shows the $u_f$ is passed through a low pass filter to make the sound more pleasant. This has a negative effect on the settling time, but it does explain why in Fig. 6 the output frequency and input frequency control are proportional as described in the previous section.

Next, to attempt to couple the frequency and amplitude to improve results, let’s take a look at the equations derived in the earlier sections. Frequency seems to not depend on the amplitude whatsoever but the latter depends on the first. Since an increase in frequency causes an increase in the amplitude according to previous results, it would make sense to take the amplitude control signal and divide it by something proportional to the frequency. This is shown in Mod. 7 where the frequency is fed back with a gain of about 0.001 to compensate for the large numerical value of frequency, and added to 1, since 1 is the equilibrium (no control) level and this new value is what $u_A$ is divided by to improve results. Please see the results in Fig. 7.

The control signal of the amplitude settles faster and the amplitude itself has less overshoot.

--

Thank you for your time,

Rayan El Helou

**APPENDIX**

The labelling system in the Appendix and the Report is as follows:

- **Fig. ##** refers to a single or set of plots.
- **Mod. ##** refers to a model, subsystem or schematic from Simulink.

MATLAB® commands may be found at the end of the Appendix.

**Simulink Models**
Mod. 1: Overall (Top level) Closed-Loop MIMO Control Model

Mod. 2: Open-Loop Plant

Mod. 3: Original Signal
Mod. 4: Speaker as Physical Input to Plant

Mod. 5: Microphone Data Collection as Physical Output to Plant

Mod. 6: Discrete-Time MIMO Controller
Mod. 7: Frequency feedback to update Amplitude control

**Test Plots and Results**

![Graph](image)

**Fig. 1:** “Hello, my name is Rayan.” A test for how accurate the estimation of amplitude is.
Fig. 2: Steps in frequency of sine wave sent to speaker. A test of frequency measurement.
Fig. 3: Variable amplitude open-loop control with no frequency control
Fig. 4: Variable amplitude open-loop control with fixed frequency control
Fig. 5: Variable frequency open-loop control for fixed amplitude control input
Fig. 6: MIMO De-coupled controller

Fig. 7: MIMO Coupled controller
**MATLAB® Functions**

MATLAB function used for dominant frequency measurement (Made by me)

```matlab
function fmic = fcn(y_mic)

coder.extrinsic('evalin');
fs = 0;
fs = evalin('base', 'fs');  % Receive sampling rate from workspace
fmic = 0;

[H, W] = dtft(y_mic, length(y_mic));  % See attached function

band_min = round((length(y_mic) - 1)/fs*(200 + fs/2)+1);
band_max = round((length(y_mic) - 1)/fs*(3200 + fs/2)+1);

W = W(band_min:band_max);
H = H(band_min:band_max);

[index_max, ~] = find(abs(H) == max(abs(H)));
fmic = W(max(index_max))*fs/2/pi;
```

MATLAB function used for amplitude estimation (Made by me)

```matlab
function Amic = fcn(y_mic)

Amic = mean([ max(abs(y_mic(1:end/4)))
              max(abs(y_mic(end/4+1:end/2)))
              max(abs(y_mic(end/2+1:3*end/4)))
              max(abs(y_mic(3*end/4+1:end)))
            ]);```

MATLAB function used to find discrete-time Fourier transform of data (see copyright)

```matlab
function [H, W] = dtft(h, N)

% DTFT  calculate DTFT at N equally spaced frequencies
%------
% Usage: [H, W] = dtft(h, N)
%  h : finite-length input vector, whose length is L
%  N : number of frequencies for evaluation over [-pi,pi)
%  ==> constraint: N >= L
%  H : DTFT values (complex)
%  W : (2nd output) vector of freqs where DTFT is computed

%------------------------------------------------------------------------
% copyright 1994, by C.S. Burrus, J.H. McClellan, A.V. Oppenheim,
% T.W. Parks, R.W. Schafer, & H.W. Schussler.  For use with the book
% "Computer-Based Exercises for Signal Processing Using MATLAB"
% (Prentice-Hall, 1994).
%------------------------------------------------------------------------
```
N = fix(N);
L = length(h);  h = h(:);  %<-- for vectors ONLY !!!!
if( N < L )
    error('DTFT: # data samples cannot exceed # freq samples')
end
W = (2*pi/N) * [ 0:(N-1) ]';
mid = ceil(N/2) + 1;
W(mid:N) = W(mid:N) - 2*pi;  % <---- move [pi,2pi) to [-pi,0)
W = fftshift(W);
H = fftshift( fft( h, N ));  %<--- move negative freq components