Vowel Formant Creation by Formant Wave Synthesis Techniques

Stuart J. Wichman
ECE Department
New Mexico State University
Las Cruces, NM
swichman@nmsu.edu

***Abstract*—Human speech uses various phonetic mechanics to create distinguishable sounds that allow our ideas to be modulated into speech for the purpose of exchanging ideas. Formants are the phonetic device that are created by positioning the back and middle regions of tongue in a way that alters the acoustic resonance properties of the mouth, causing local maxima in the frequency domain, with more energy concentrated around these maxima regions. Formant Wave Synthesis can recreate this frequency response by creating individual formants and placing them in correct bands to create the same vowel sounds and timbre as the human voice.**

Keywords— Additive synthesis, Speech analysis, Speech synthesis, Vocoders

I. Introduction

Modeling and synthesis of the human voice has been a topic of interest in research since the early 20th century, when Homer Dudley developed the Voder, a vocoding device. This device used two waveforms, a square wave generator, and a noise generator. The square waves created a multifrequency spectrum that was manipulated by applying filters controlled by a set of "spectrum keys" very similar to a manual on an organ to subtract unwanted frequencies and retain formants at desired frequencies. The noise channel was to facilitate non-formant based sound structures (plosives, fricatives, and affricates) and to also simulate a "breath" effect on the sound, which acts as a raising of the noise floor to smooth the gaps in frequency domain caused by the spectral spread of square waves.

This device was a purely analog device that required trained operators to tease out even the most basic speech capability. It was a notoriously complicated machine to operate.

With the arrival of digital systems, speech analysis became a much more refined area of research. The capability to produce an accurate spectrogram of recorded speech samples enabled a clearer image of the phonetic devices in play for various speech mechanics. This also allowed for new synthesis techniques to be applied to speech synthesis.

In 1978, Werner Kaegi and Stan Tempelaars published a method called 'VOSIM' [1] that involved generating a burst of sine waveforms with a decay envelope. The sine bursts were triggered by the rising edge of an external clock trigger, and the control variables acting as "knobs" to tune the formants were the period time of the sine (T), the decay envelope (b), burst length (N), and trigger rate.

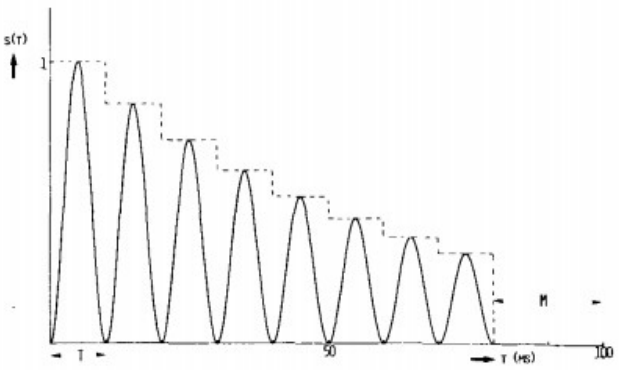

Fig. 1. The VOSIM time function. N=8, b=.85, T=10ms

A natural progression to this is the FOF (Fonction d'onde formantique) or Formant Wave Synthesis technique developed by Xavier Rodet and published in 1984, which still uses a sinusoid with decay envelope, but with added attack envelope, which gives more control of the formants synthesized in the form of a direct control over formant bandwidth, impacting

warmth and color of the generated tones, giving it a less "robotic" characteristic.

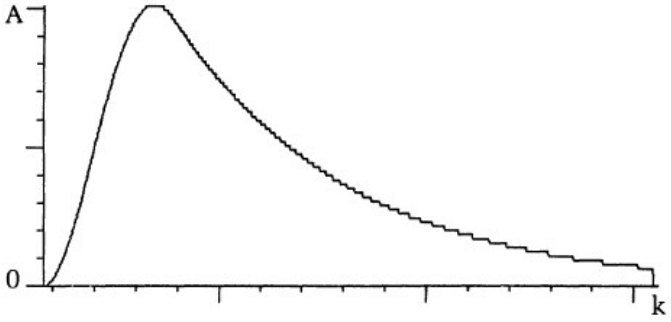

Fig. 2. The FOF Time Function

Replication of this latter method will be the focus of this study. The FOF will be implemented, and various formants will be generated using this method for a qualitative analysis of the formants synthesized by this technique, and a qualitative analysis of the vowel sounds produced by the formants.

II. FOF Overview

The Formant Wave Synthesis technique uses a sinusoid that is designed to be the peak of the formant (F_c), with an attack (β) and decay (α) envelope that affect the bandwidth (BW) and pass band transition ($\frac{\pi}{\beta}$) of the formant. The formula for the enveloped sinusoid is defined:

$$s(k) = \begin{cases} 0, k \leq 0 \\ \frac{1}{2}(1-\cos(\beta k))e^{-\alpha} \ \sin(\omega_c k + \phi), 0 \leq k \leq k_1 \\ e^{-\alpha k}\sin(\omega_c k + \phi), k_1 < k \end{cases}$$

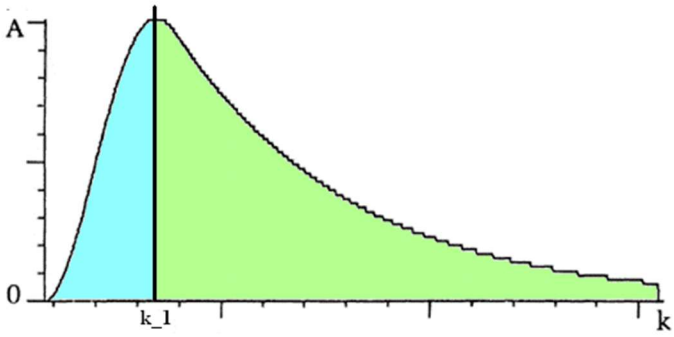

Fig. 3. The FOFlet with the blue region indicating the attack component where 0<k<k1, the green region indicating the decay component where k > k1.

This equation states that there is no excitation before the impulse arrives ($0, k \leq 0$). When the impulse arrives, the sin wave is excited ($\sin(\omega_c k + \phi)$) ad will oscillate at the desired frequency, causing the center frequency of the formant. The variable k_1 defines the transition point between the attack profile and the decay profile, so while $k \leq k_1$, the attack profile will increase amplitude from 0 to maximum amplitude (A). The attack profile $\frac{1}{2}(1-\cos(\beta k))$ relies on $\beta = \frac{\pi}{k_1}$ so that when k approaches k_1, the output of the attack function will move from 0 to 1, with 1 being the peak of the attack profile, and also the transition to the decay profile. For the decay profile, the $e^{-\alpha k}$ term will envelope the sinusoid, bringing the amplitude of the sinusoid to 0 as k approaches the end of the impulse train. This should be designed such that $\frac{F_s}{F_{impulse}}$ is the maximum value of k. This ensures that the impulse response is reasonably damped before the next impulse arrives, avoiding an unbounded divergence in the system output.

When transformed to the Frequency Domain, the temporal structure gives us a formant as defined:

$$S(\omega) = \mathcal{F}\{s(k)\}(\omega) = \frac{M+1}{2}\frac{1}{\sqrt{\alpha^2+\omega^2}}$$

where $M = e^{-\frac{\alpha\pi}{\beta}}$, $\alpha = BW\pi$, all variables are shared between the time-domain and frequency-domain representation of the formants. An amplitude coefficient is applied after the convolution of the impulse and formant structure to give an amplitude profile to the formant $A_n S(\omega)$.

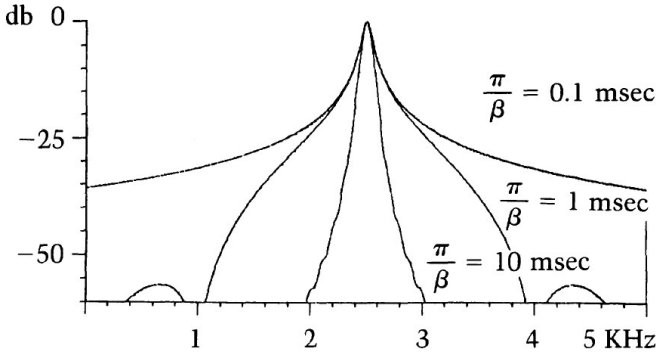

Fig. 4. Formant Strucure in the Frequency Domain. The peak amplitude will be at A_n dB.

Of note is that the BW values define the formant bandwidth between the -6 dB from off peak cutoff frequencies, and the β defines the shape of the formant outside of the cutoff frequencies. This allows β to control the 'smoothness' or 'warmth' of the formant.

III. Vowel Formant Structure

Vowels are made of a combination of formants spread to multiple frequencies and with varying amplitude. This creates a profile that can be parameterized for each specific vowel representation. This paper will concentrate on two cases with several examples for each case. The first case will be 5 formants per vowel sound and will look at the case of short vowel sounds of the major 5 vowels available to the English language. Table I outlines the parameters used to define each vowel sound for the case of the 5 formant structures. These values were defined in Table 1 of Xavier Rodet's paper on FOF synthesis [2].

The k1 variables (and thus the β's) are statically defined for all vowel sounds as:

$$k_1 = [0.002ms, 0.0015ms, 0.0015ms, 0.003ms, 0.001ms]$$

The β is simple $\frac{\pi}{k_1}$ for each element of k_1.

TABLE I. PARAMETER VECTORS OF 5 FOF FORMANTS [3]

Vowel Sound Parameters					
	'A'	***'E'***	***'I'***	***'O'***	***'U'***
Frequency (Hz)					
F_{c1}	600	400	220	400	350
F_{c2}	1040	1620	1720	750	600
F_{c3}	2250	2400	2600	2400	2400
F_{c4}	2450	2800	3050	2600	2675
F_{c5}	2750	3100	3340	2900	2950
Amplitude (dB)					
A_1	0	0	0	0	0
A_2	-7	-12	-30	-11	-20
A_3	-9	-9	-16	-21	-32
A_4	-9	-12	-22	-20	-28
A_5	-20	-18	-28	-40	-36
BW (Hz)					
BW_1	60	40	60	40	40
BW_2	70	80	90	80	80
BW_3	110	100	100	100	100
BW_4	120	120	120	120	120
BW_5	120	120	120	120	120

Additionally, a three-frequency formant implementation was performed on a wider spectrum of formant sounds:

TABLE II. PARAMETER VECTORS OF 3 FOF FORMANTS [4]

	Fc1	***Fc2***	***Fc3***
B'**ea**'t	270	2300	3000
B'**i**'t	400	2000	2550
B'**e**'t	530	1850	2500
B'**a**'t	660	1700	2400
P'**ar**'t	730	1100	2450
P'**o**'t	570	850	2400
B'**oo**'t	440	1000	2250
B'**oo**'k	300	850	2250
B'**u**'t	640	1200	2400
P'**er**'t	490	1350	1700

For the case of the three formant vowels, each vowel shared the Amplitude profile, k_1 profile, and BW profile which simplified the implementation process and didn't seem to interfere terribly with the quality of the formants, as these values are within range or human speech formants F1, F2, and F3:

$$A = [0dB, -4dB, -7dB]$$

$$BW = [60Hz, 70Hz, 110Hz]$$

$$k_1 = [.002ms, .0015ms, .0015ms]$$

IV. MODEL IMPLEMENTATION

The synthesis model was implemented in MATLAB. Fundamentally, the implementation can be applied in most any language capable of convolution operations. Octave implementation is also possible, but the function spectrogram() from the signal processing toolbox is used, which has no direct implementation in Octave, and the code would have to be adjusted to rely instead on specgram(), which is part of the signal package in Octave.

A sample rate was chosen at 44.1 kHZ, which is a fairly standard sample rate to capture audible sound. This is technically oversampled, as frequencies over 10kHZ are typically not present in human speech, but this oversampling allows for a finer spectral analysis of the frequency bands of interest, and ample resolution for the formant to develop. Since there is no need for modulation of formants for proper speech synthesis, a sample period of 1 second was chosen. This simplifies all of the conversion of individual variables into the time domain, as we can base all conversions on the 44100 samples with a sample rate of 44100 Hz, a sort of 1-to-1 conversion between sample set length, sample rate, and sample time.

There are two functions created for processing of inputs to vowel sounds. The first function is the function that takes all of the formant parameters and creates the time-domain signal and envelope. The signal is then convolved with the input impulse train to create the FOFlet, which translates to the formant in the frequency domain.

The second function takes the parameters for multiple FOFlets and iterates through them calling the FOFlet generating function for each set of parameters. This allows N FOFlets to be created with the function, which allows for faster experimentation with the vowel formant structures. This function also has an optional graph associated with it that will create a plot of the N FOFlets generated with the associated k_1 parameter annotated on the plots. This is useful to verify the correct transition from attack profile to decay profile, and can help distinguish one data set from another, given that they are using different k_1 input vectors.

The main script creates input vectors from defined parameters for each vowel (e.g. the 5-formant vowel 'A' will have a vector of length for frequency, a vector of length 5 for amplitude profile, another for bandwidth, k_1 values, and β values). These are then passed to the vowel function. The output of this is then passed into a Welch power spectral density function that plots the PSD of the vowel, and highlights the values in the frequency input vector, to show any deviation from desired frequency and the formant realization's center frequencies. The amplitude is not annotated, as the ear is less discerning with this characteristic when it comes to distinguishing one vowel from another.

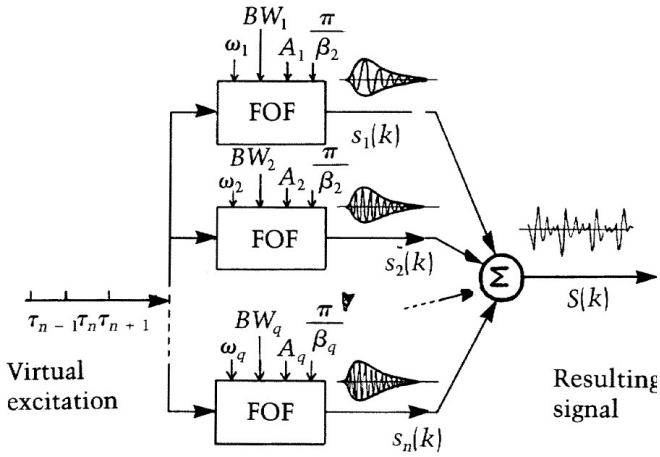

Fig. 5. Rodet's multi-formant FOF engine to synthesize speech.

Figure 5 illustrates the two primary functions in the model. The blocks containing the FOF in the figure is associated with the FOFlet synthesis function, and the summation and control of calling the FOFlets with proper arguments are handled by the vowel function.

For this project, for the virtual excitation, an impulse train operating at 120 Hz was used. This give us 367.5 samples (rounded to 367) per FOFlet, with FOFlets being generated every $\tau \cong 0.008ms$.

V. Results

All of the results were generated using the same k_1 vector, β vector, sample rate, and impulse train. It stands that since the resulting FOFlets are in samples, we should define the k1 vector in samples rather than in time. Since we are multiplying floating point numbers for this, there will be some error introduced, as the floating-point results will have to be typecast to integer types for indexing purposes. I will define the vector with floating-point numbers for exact results with the understanding that the decimals will be truncated.

$$k_1 = [88.2, 66.15, 66.15, 132.3, 44.1]\ samples$$

Converting the k_1 values thusly will make reading the resulting charts simple, as the marker on the figures will represent the sample at k_1, which is the marker for the transition from attack envelope to decay envelope.

First the results of synthesis of vowel formants with 5 FOFlets will be reviewed, then the vowels formed with 3 FOFlets. There are some notable differences that will be discussed

A. 'A' Vowel Formants

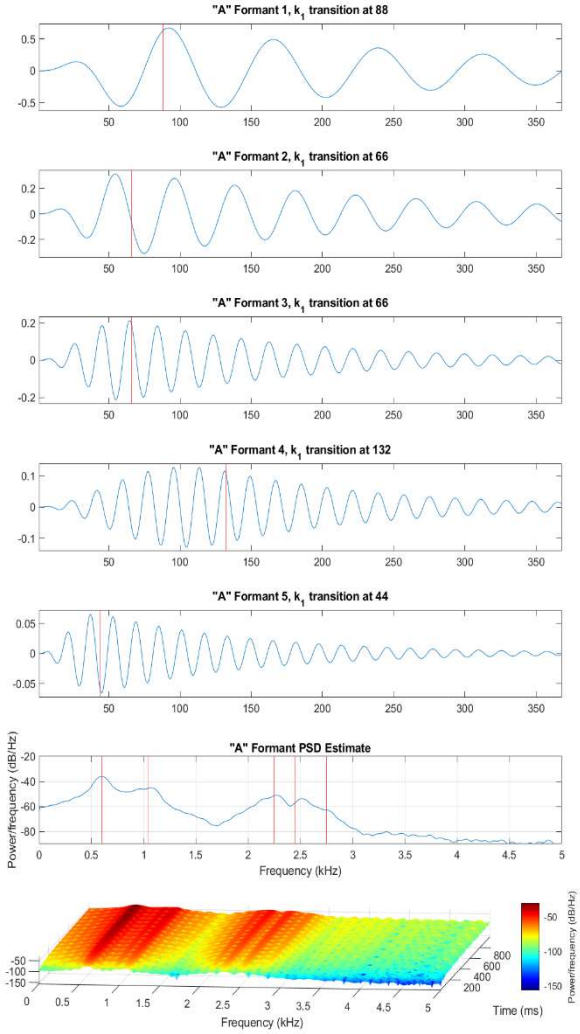

Fig. 6. 'A' synthesized with 5 FOFlets.

The 'A' formants as described by input parameters:

Fc(Hz) = [600 1040 2250 2450 2750];

A(dB) = [0 -7 -9 -9 -20];

BW(Hz) = [60 70 110 120 130];

B. 'E' Vowel Formants

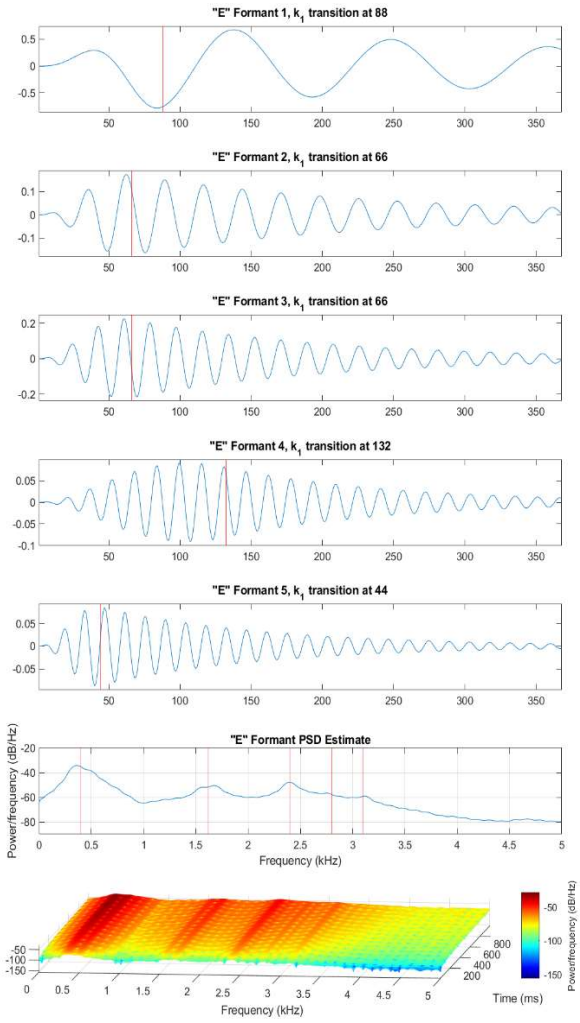

Fig. 7. 'E' synthesized with 5 FOFlets

The 'E' formants as described by input parameters:

Fc(Hz) = [400 1620 2400 2800 3100];

A(dB) = [0 -12 -9 -12 -18];

BW(Hz) = [40 80 100 120 120];

C. 'I' Vowel Formants

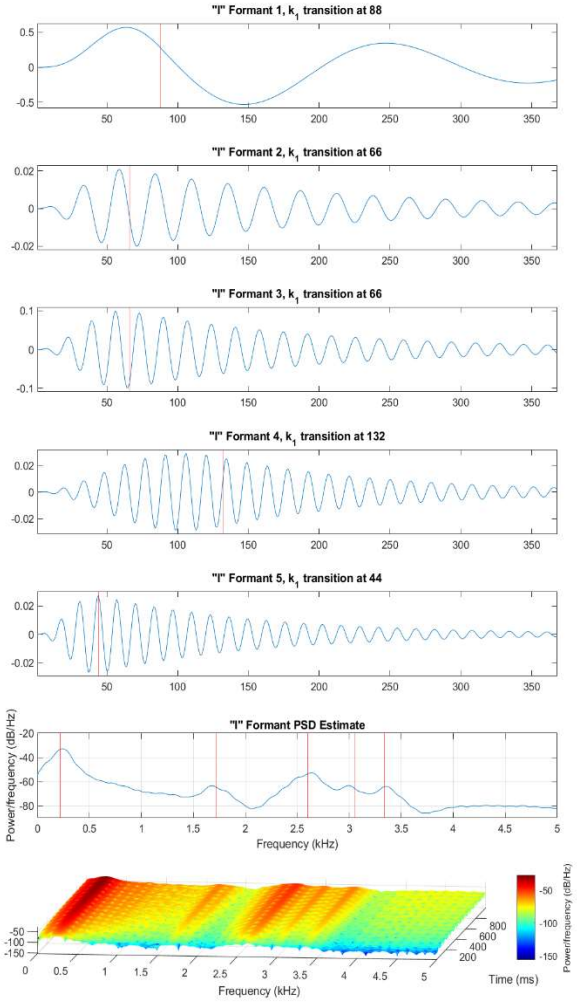

Fig. 8. 'I' synthesized with 5 FOFlets

The 'I' formants as described by input parameters:

Fc(Hz) = [220 1720 2600 3050 3340];

A(dB) = [0 -30 -16 -22 -28];

BW(Hz) = [60 90 100 120 120];

D. ‘O’ Vowel Formants

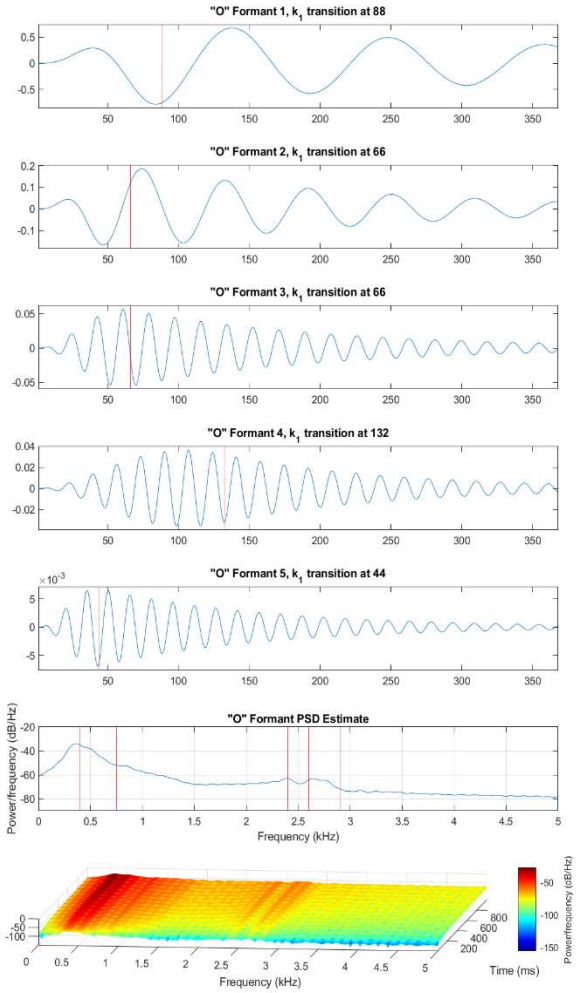

Fig. 9. 'O' synthesized with 5 FOFlets

The ‘O’ formants as described by input parameters:

Fc(Hz) = [400 750 2400 2600 2900];

A(dB) = [0 -11 -21 -20 -40];

BW(Hz) = [40 80 100 120 120];

E. ‘U’ Vowel Formants

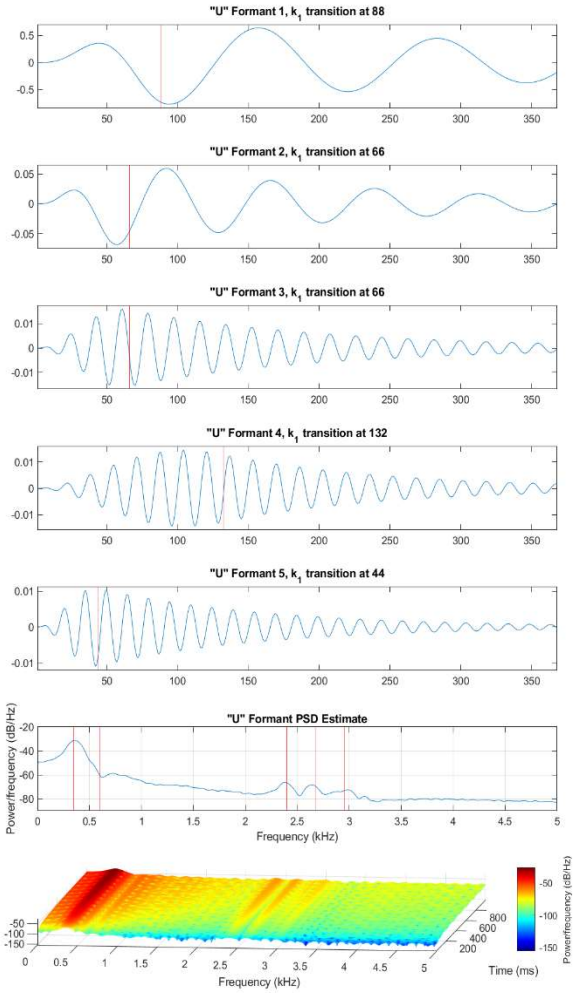

Fig. 10. 'U' synthesized with 5 formants.

The ‘U’ formants as described by input parameters:

Fc(Hz) = [350 600 2400 2675 2950];

A(dB) = [0 -20 -32 -28 -36];

BW(Hz) = [40 80 100 120 120];

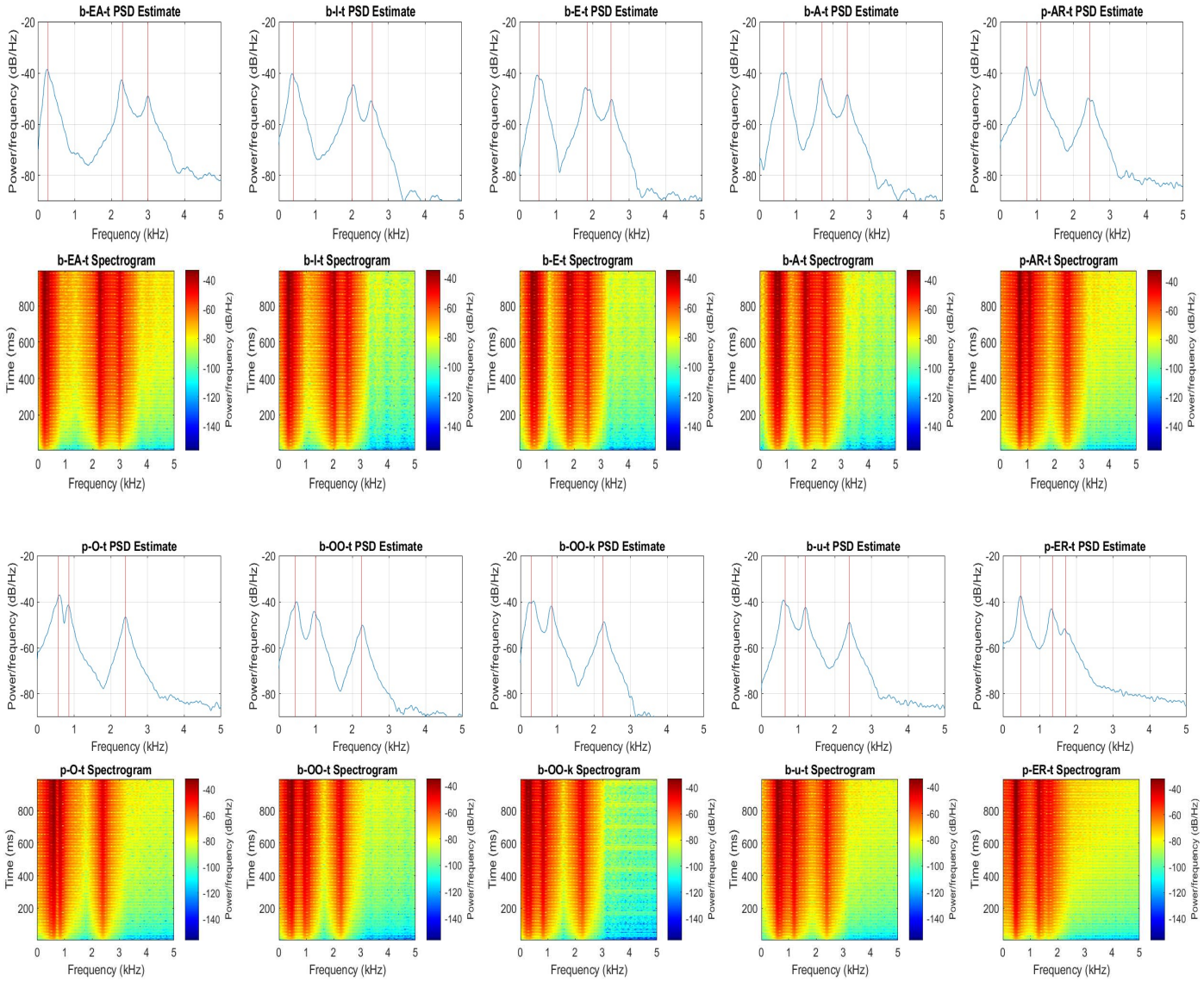

Fig. 11. Vowel formants formed by 3 FOFlets. The lack of the 4th and 5th formant is evident, and the issues associated with the 4th FOFlet are diverted for these formants. Qualitatively, the vowel sounds are very clear and distinct, despite reduced FOFlet synthesis.

The 5 FOFlet vowels are qualitatively what they are supposed to be (they sound like the appropriate vowel sounds), but there are some flaws in the implementation that could cause some issues with full speech synthesis. Two items are of note.

As can be seen from the spectrogram of the formants, and (although not shown) is present in the waveform in the time-domain for the entire sample set, there is a tendency of the algorithm to grow in amplitude as the tone is sustained. I believe that this is due to the design of the system response to the impulse train. The impulse train is exciting the waveform at 120 Hz, but the sample rate of the system is 44100, which creates a remainder form the modulo of the impulse train frequency with the sample rate. This fractional sample's energy is being inserted into the next impulse excitation and creating a cascade of increasing energy present that is unbounded. This will eventually saturate the output medium and the signal will likely be clipped by any audio device, creating distortion. This can be managed via either analytical normalization, or reassessment of the algorithm to line the impulse train's frequency with the sampling frequency of the output signal.

The second notable flaw is evident in the fourth formant (F4). The k_1 value should be noting the transition from attack to decay envelopes, but not in this case. The k_1 is at the expected marker, but the signal is in obvious decay. This is due to one coefficient in the attack envelope being overpowered by another coefficient. The $\frac{1}{2}(1-\cos(\beta k))$ term that should describe the attack rate is getting overpowered by the $e^{-\alpha k}$ term, implying the particular value of k_1 is beyond the upper bounds of where it should be for this system's sample rate and impulse train frequency. The formants shown Figure 4 show an expected response with arbitrary (but within bounds) system parameters.

This was replicated, and as suspected, the more sustained attack/decay transitions can fall out of bounds with system parameters, leading to a destabilization of the formant in the frequency domain. While this didn't seem to have an effect on the resultant sound, sustained usage would most assuredly produce a use case where this would be an issue that would need to be addressed by recharacterization of the excitation impulse or sampling frequency.

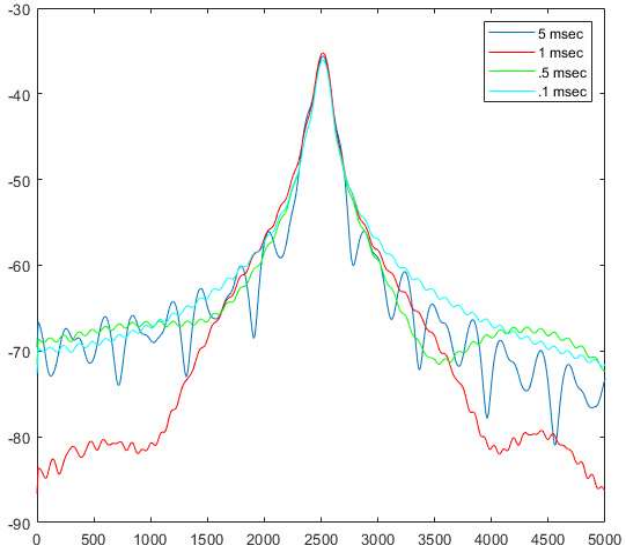

Fig. 12. Frequency domain representation of formants synthesized with increasing atta/decay transition points.

The vowel formants formed with 3 FOFlets, while still prone to the unbounded growth, are deficient of the premature transition, as the 4^{th} k_1 value is truncated from the system. This can be seen in Figure 11 on the previous page.

VI. Conclusion

Rodet's FOF technique for formant synthesis using time-domain based methods give reasonable results, while being computationally simple, as the input is an impulse train, and the convolution with the attack/decay enveloped filter is straight forward because of this. The variables that translate from time to frequency domain are a clever way to parameterize a signal for implementation. Some work needs to be done on the front end to ensure that the system sample rate and excitation times line up. Post processing of the signal is minimal. The only serious post processing that was implemented in this project was normalization of the resultant waveforms.

Some additional post-processing that could be done to improve the quality of the formants would be to apply an amplitude envelope across the entire signal to provide a static amplitude and correct the unbounded amplitude growth from mis-matched impulse input frequencies and sample rate of the signal.

Replacement of the impulse train with an excitation arch (sync function) can smooth out some of the noise that exists on the higher frequency formants. This can be as simple as placing a low pass filter after the impulse train, but before the input to the FOFlets, with a compensating filter to bring the maxima of the sync pulses back to unity amplitude.

It would be an interesting algorithm to implement in hardware. A DSC with a MAC (Multiplier Accumulator) and DAC (Digital to Analog Converter) should be capable. The only computationally heavy components would be the sinusoid calculations, as those are not implemented in an atomic fashion, and would require several cycles to complete. The low fidelity output by converting impulse trains to formants implies that a lower quality DAC (read cheaper DSC) could handle this sort of formant synthesis. More work would need to be done with consonants and other phonetic devices to assess the full range of speech. There are likely much more effective methods to generate clear speech with more modern techniques, but this was an interesting approach and very clever for the time (it's still clever now).

The Casio CZ series used this sort of VOSIM or FOF resonant formant synthesis techniques for their synthesis engine. It seems a good choice, as they were nice entry level keyboards and the hardware platform had oscillators to spare to act as excitation impulse triggers. This technique has a classic 80's feel that takes me back to the times when talking toys were the most amazing novelties available.

The results of this project were uploaded and made available through the presentation and sound board that has all of the formants generated, and can be viewed at:

Presentation:

https://md.hellcorp.org/p/H1l-ABYjD#/

Sound board:

https://md.hellcorp.org/s/HJZPsV1hD

The associated code for this project, while included in the appendix can also be accessed on github:

https://github.com/swichman/FOFlets

All of the associated debugging and extracurricular graphing was left in the matlab scripts, and just commented out so that it could be enabled as needed.

ACKNOWLEDGMENT

This paper was based on a paper done by Xavier Rodet. His work on time based resonant function formant synthesis was brilliant, andhas been replicated likely countless times in universities the world over. I found so many examples of implementing this method, that I am inspired to upload this paper for future reference by numerous students and seed a new generation of synthesizer enthusiasts and signal processing students.

J.O. Smith was a stellar resource. He implemented this method using FAUST IDE, which is an online IDE for making sound-based apps. There is an API for creating knobs that act as control parameters, which allowed me to tune the formants with real time outputs, allowing a fantastic resource to develop my parameterizations. The scripting that was used to build this web app was also very useful to help develop the MATLAB script, although not exactly a one-to-one conversion as the web app is designed for real time operation, while the MATLAB script was written with full knowledge of the complete sample set from the outset.

Lastly, I would like to acknowledge those that enabled me the temporal resources to complete this project. My coworkers have had to put up with my uninspired performance over the duration of this project, and my family that hasn't really complained too much about me locking myself in my room for long stretches to code, write, code, and write some more.

REFERENCES

[1] W. Kaegi, and S Tempelaars, "VOSIM--A New Sound Synthesis System" Journal of the Audio Engineering Society, vol. 26 num. 6, pp. 418–425, June 1978.

[2] X. Rodet, "Time-Domain Formant-Wave-Function Synthesis", Computer Music Journal, vol. 8 num. 3, pp. 9-14, Autumn 1984.

[3] Smith, J.O. "Formant Synthesis Models", in *Physical Audio Signal Processing,* http://ccrma.stanford.edu/~jos/pasp/Formant_Synthesis_Models.html, online book, 2010 edition, accessed 2020.

[4] MamonuLabs "Average Formant Frequencies", http://mamonu.weebly.com/formant-frequencies-table.html, accessed 2020.

Appendix A

FOF.m

```
%=========================================================================
% Formant Synthesis using FOF methods. This is designed to take an impulse
% train at 120 Hz and generate 5 formats that are mixed together to
% create various vocal formants.
%=========================================================================
% Nice audio sample rate
Fs = 44100;
%=============================================
% Impulse Train Creation
%=============================================
input = zeros(1,Fs);
F_imp = 120;
input(1:int32(Fs/F_imp):Fs) = 1;
%=============================================
% Formant synthesis with five parallel FOFs
%=============================================
% Center Frequencies of the FOF Synthesizer output
Fc_a = [600 1040 2250 2450 2750];
Fc_e = [400 1620 2400 2800 3100];
Fc_i = [220 1720 2600 3050 3340];
Fc_o = [400 750 2400 2600 2900];
Fc_u = [350 600 2400 2675 2950];
% Max Amplitudes of the FOF Synthesizer output
A_a = [0 -7 -9 -9 -20];
A_e = [0 -12 -9 -12 -18];
A_i = [0 -30 -16 -22 -28];
A_o = [0 -11 -21 -20 -40];
A_u = [0 -20 -32 -28 -36];
% Bandwidth of the FOF
BW_a = [60 70 110 120 130];
BW_e = [40 80 100 120 120];
BW_i = [60 90 100 120 120];
BW_o = [40 80 100 120 120];
BW_u = [40 80 100 120 120];
%=============================================
% Formant synthesis with three parallel FOFs
%=============================================
% Center Frequencies of the FOF Synthesizer output
Fc_beat = [270 2300 3000];
Fc_bit = [400 2000 2550];
Fc_bet = [530 1850 2500];
Fc_bat = [660 1700 2400];
Fc_part = [730 1100 2450];
Fc_pot = [570 850 2400];
Fc_boot = [440 1000 2250];
Fc_book = [300 850 2250];
Fc_but = [640 1200 2400];
Fc_pert = [490 1350 1700];
% Max Amplitudes of the FOF Synthesizer output
A_3 = [0 -4 -7];
% Bandwidth of the FOF
BW_3 = [60 70 110];
%=============================================
% System Constants
%=============================================
% k_1 values (where the attack transitions to decay)
k_1 = [.002*Fs .0015*Fs .0015*Fs .003*Fs .001*Fs];
k_1_3 = [.002*Fs .0015*Fs .0015*Fs];
% Beta Values
beta = [pi/k_1(1), pi/k_1(2), pi/k_1(3), pi/k_1(4), pi/k_1(5)];
%=============================================
% Synthesize each vowel from formants
%=============================================
out_a = vowel(Fc_a, A_a, BW_a, k_1, beta, Fs, input, 0, '"A" Formant');
out_e = vowel(Fc_e, A_e, BW_e, k_1, beta, Fs, input, 0, '"E" Formant');
out_i = vowel(Fc_i, A_i, BW_i, k_1, beta, Fs, input, 0, '"I" Formant');
out_o = vowel(Fc_o, A_o, BW_o, k_1, beta, Fs, input, 0, '"O" Formant');
out_u = vowel(Fc_u, A_u, BW_u, k_1, beta, Fs, input, 0, '"U" Formant');
```

```
out_beat = vowel(Fc_beat, A_3, BW_3, k_1_3, beta(1:3), Fs, input, 1, 'bEAt Formant');
out_bit = vowel(Fc_bit, A_3, BW_3, k_1_3, beta(1:3), Fs, input, 1, 'bIt Formant');
out_bet = vowel(Fc_bet, A_3, BW_3, k_1_3, beta(1:3), Fs, input, 1, 'bEt Formant');
out_bat = vowel(Fc_bat, A_3, BW_3, k_1_3, beta(1:3), Fs, input, 1, 'bAt Formant');
out_part = vowel(Fc_part, A_3, BW_3, k_1_3, beta(1:3), Fs, input, 1, 'pARt Formant');
out_pot = vowel(Fc_pot, A_3, BW_3, k_1_3, beta(1:3), Fs, input, 1, 'pOt Formant');
out_boot = vowel(Fc_boot, A_3, BW_3, k_1_3, beta(1:3), Fs, input, 1, 'bOOt Formant');
out_book = vowel(Fc_book, A_3, BW_3, k_1_3, beta(1:3), Fs, input, 1, 'bOOk Formant');
out_but = vowel(Fc_but, A_3, BW_3, k_1_3, beta(1:3), Fs, input, 1, 'bUt Formant');
out_pert = vowel(Fc_pert, A_3, BW_3, k_1_3, beta(1:3), Fs, input, 1, 'pERt Formant');

%=============================================
% Write all files to disk for qualitative
% analysis (listen to them).
%=============================================
audiowrite('_a.wav', out_a, Fs)
audiowrite('_e.wav', out_e, Fs)
audiowrite('_i.wav', out_i, Fs)
audiowrite('_o.wav', out_o, Fs)
audiowrite('_u.wav', out_u, Fs)

audiowrite('_beat.wav', out_beat, Fs)
audiowrite('_bit.wav', out_bit, Fs)
audiowrite('_bet.wav', out_bet, Fs)
audiowrite('_bat.wav', out_bat, Fs)
audiowrite('_part.wav', out_part, Fs)
audiowrite('_pot.wav', out_pot, Fs)
audiowrite('_boot.wav', out_boot, Fs)
audiowrite('_book.wav', out_book, Fs)
audiowrite('_but.wav', out_but, Fs)
audiowrite('_pert.wav', out_pert, Fs)
```

FOFlet.m

```
function [output] = FOFlet(Fs,bw,k_1,beta,f_c, A, in_train)
%FOFLET Summary of this function goes here
%   Detailed explanation goes here
Ts = 1/Fs;
fof = zeros(1,Fs);    %only 1 second of audio output
%f_c = f_c/120;

j = 1;
while j < Fs
    for k = 1:k_1
        fof(j) = .5*(1 - cos(beta*(k-1)))*exp(-bw*pi*Ts*(k-1))*sin(2*pi*f_c*(k-1)*Ts);
        j = j + 1;
    end

    for k = k_1:Fs/120
        fof(j) = exp(-bw*pi*Ts*k)*sin(2*pi*f_c*k*Ts);
        j = j + 1;
    end
end
% figure
% plot(conv(in_train,fof(1:2000)))

output = conv(in_train,db2mag(A)*fof);
%output = conv(in_train,fof);
```

vowel.m

```
function [out] = vowel(Fc, A, BW, k_1, beta, Fs, impulse, graph, graphtitle)
%VOWEL Summary of this function goes here
%   Detailed explanation goes here
FOF = zeros(1,Fs);
if graph == 1
    h = figure('Position',[0 0 800 1300]);
end
for i = 1:length(Fc)
    tmp = FOFlet(Fs, BW(i), k_1(i), beta(i), Fc(i), A(i), impulse);
    if graph == 1
        subplot(length(Fc)+2,1,i)
        plot(tmp(1:368))
        xline(k_1(i), 'r')
        ylim([min(tmp(1:368)) + min(tmp(1:368))*.1, max(tmp(1:368)) + max(tmp(1:368))*.1])
        xlim([1, 368])
        k1_value = strcat(', k_1 transition at ', compose(' %0.f',k_1(i)));
        FOFlet_num = strcat(graphtitle, compose(' %d', i));
        titleText = strcat(FOFlet_num, k1_value);
        title(titleText)
    end
    FOF = FOF + tmp(1:Fs);
end
% Normalize the output to avoid clipping
FOF = FOF/max(abs(FOF));
if graph == 1
    window = 512;
    noverlap = 256;
    nfft = 4096;
    rot_x = 7;
    rot_y = 63;

    subplot(length(Fc)+2,1,length(Fc)+1)
    pwelch(FOF, window, noverlap, nfft, Fs);
    ax = gca;
    xlim(ax, [0,5])
    ylim(ax, [-90 -20]);
    xline(Fc(1)/1000, 'r')
    xline(Fc(2)/1000, 'r')
    xline(Fc(3)/1000, 'r')
    if length(Fc) > 3
        xline(Fc(4)/1000, 'r')
        xline(Fc(5)/1000, 'r')
    end
    title(strcat(graphtitle, ' PSD Estimate'));
%     subplot(length(Fc)+2,1,length(Fc)+2)
%     spectrogram(FOF, window, noverlap, nfft, Fs)
%     view(rot_x,rot_y)
%     ax = gca;
%     xlim(ax, [0,5])
%     colormap('jet')
    saveas_name = strcat(erase(graphtitle, '"'),'.png');
    saveas(h, saveas_name);
end
out = FOF;
end
```

Plot3FOFlet.m

```
this = figure('Position',[0 0 1980 500]);

window = 512;
noverlap = 256;
nfft = 4096;
rot_x = 0;
rot_y = 90;

plot_fof(out_beat,Fs,Fc_beat,1,'b-EA-t',this)
plot_fof(out_bit,Fs,Fc_bit,2,'b-I-t',this)
plot_fof(out_bet,Fs,Fc_bet,3,'b-E-t',this)
plot_fof(out_bat,Fs,Fc_bat,4,'b-A-t',this)
plot_fof(out_part,Fs,Fc_part,5,'p-AR-t',this)

saveas(this, 'top_5_3-FOFlets.png');

plot_fof(out_pot,Fs,Fc_pot,1,'p-O-t',this)
plot_fof(out_boot,Fs,Fc_boot,2,'b-OO-t',this)
plot_fof(out_book,Fs,Fc_book,3,'b-OO-k',this)
plot_fof(out_but,Fs,Fc_but,4,'b-u-t',this)
plot_fof(out_pert,Fs,Fc_pert,5,'p-ER-t',this)

saveas(this, 'bottom_5_3-FOFlets.png');
```

plot_fof.m

```
function [out] = plot_fof(FOF, Fs, Fc, placement, strTitle, fig)
%PLOT_FOF Summary of this function goes here
%   Detailed explanation goes here
window = 512;
noverlap = 256;
nfft = 4096;
rot_x = 0;
rot_y = 90;
figure(fig)

subplot(2,5,placement)
pwelch(FOF, window, noverlap, nfft, Fs);
ax = gca;
xlim(ax, [0,5])
ylim(ax, [-90 -20]);
xline(Fc(1)/1000, 'r')
xline(Fc(2)/1000, 'r')
xline(Fc(3)/1000, 'r')
title(compose('%s PSD Estimate', strTitle));

subplot(2,5,placement+5)
spectrogram(FOF, window, noverlap, nfft, Fs)
view(rot_x,rot_y)
ax = gca;
xlim(ax, [0,5])
title(compose('%s Spectrogram', strTitle));
colormap('jet')
out = 0;
end
```