Investigation of Aliasing Effects*

University Of Washington Dept. of Electrical Engineering

This work is produced by The Connexions Project and licensed under the
Creative Commons Attribution License †

Abstract

This lab investigates the effect of aliasing.

This development of these labs was supported by the National Science Foundation under Grant No. DUE-0511635. Any opinions, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

1 Introduction

Aliasing literally means "by a different name" and is used to explain the effect of under-sampling a continuous signal, which causes frequencies to show up as different frequencies. This aliased signal is the signal at a different frequency. This is usually seen as higher frequencies being aliased to lower frequencies. For a 1d signal in time, the aliased frequency components sound lower in pitch. In 2d space, such as images, this can be observed as parallel lines in pinstripe shirts aliasing into large wavy lines. For 2d signals that vary in time, an example of aliasing would be viewing propellers on a plane that seem to be turning slow when they are actually moving at very high speeds.

NOTE: The Nyquist sampling rate is twice the highest frequency of the signal. This is the minimum rate needed to prevent aliasing.

2 Signals and Aliasing

In Figure 1 a 500Hz cosine signal is shown in red, and an under-sampled version of the signal in blue.

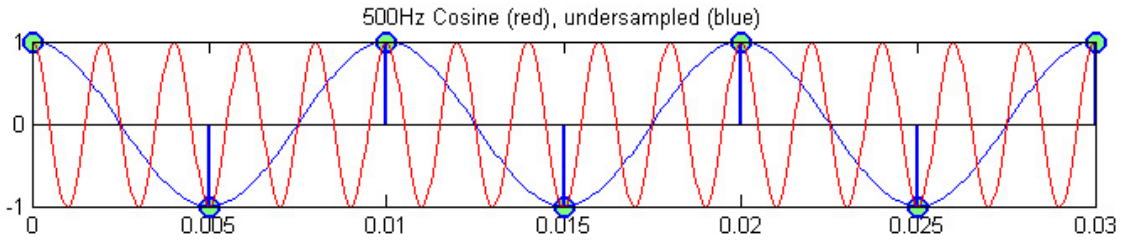

Figure 1: Aliased Signal

*Version 1.8: Nov 24, 2007 6:58 pm US/Central
†http://creativecommons.org/licenses/by/2.0/

Exercise 1
To see the effects of aliasing on a 1kHz cosine signal create an over-sampled, under-sampled, and critically-sampled version of the signal.

1. Plot a cosine at 1kHz showing at least twenty periods. Use a step size (sampling period) of 1/10kHz. This will be our over-sampled signal. Try playing this signal with `soundsc`. How many samples are needed to make the sound last 2 seconds if the step size is 1/10kHz?
2. Plot the critically-sampled version by applying what you know about Nyquist. Make sure the plot contains at least twenty periods and that you sample at a non-zero point. Listen to this signal with `soundsc`, does it sound the same?
3. Plot the under-sampled version. Make sure the plot contains at least twenty periods. Listen to this signal with `soundsc`, how does it sound now?
4. Plot all three signals stacked on top of each other using subplot. Note that the plot command uses straight line interpolation, so your plots will not look smooth like Figure 1 (which actually uses a much finer sampling period an knowledge of the aliased frequency to generate the smooth undersampled result).

3 Temporal Aliasing

Have you ever seen an old western movie and noticed that the wagon wheels appear to turn backwards even though the coach is moving forward? This phenomenon is sometimes referred to as the wagon-wheel effect, but is really an effect of temporal aliasing. You can see the same effect easily on anything with a spoked wheel, such as wheels on a stage coach and airplane propellers.

Wagon-wheels, stage coaches, horses, and airplane propellers?? What's this have to do with signal processing? Actually, quite a lot, not the wagon-wheels directly, but how the images of the wagon-wheels are captured. The video you watch from a movie or tv show is actually sampled in time (hence temporal). Typically a movie is captured at 24 frames per second (FPS).

Exercise 2
Now it's your turn to be the cinematographer. For this problem you will take an image of a wagon-wheel and "capture" a MATLAB movie at different frame rates of the wheel rotating. After the movie is made, you will be able to play it back, and if everything worked, be able to see the wheel spin.

A movie of a rotating wheel is a signal in time, and at each instant in time, instead of just one point (like a normal x(t) signal), you have a whole image defined. Thus, if you have an image of an arrow rotating, Figure 2, where the image rotates ten times per second, then the period is 1/10 second, because every 1/10 second the image (signal) is at the same value again. Thus image(t+n/10) = image(t) for all integers n.

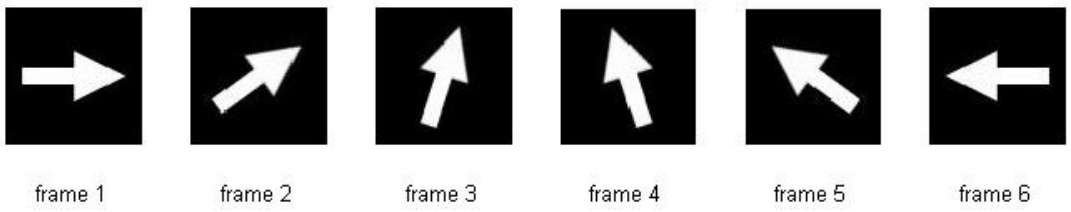

Figure 2: Frames of rotating arrow.

If an image rotates at 10 Hz (10 rotations per second), then what is the Nyquist sampling rate so that you can reconstruct the temporal signal? Recall that the signal will be critically sampled when using a sampling rate that is twice the highest frequency in the signal (20 Hz, in this case). Anything above that will be over-sampled, and fewer samples/second will be under-sampled.

Check your understanding: standard film is captured at 24 frames per second. What's the highest frequency of motion that can be reconstructed without aliasing?

Create three movies to show the wheel being over-sampled (appears to be rotating clockwise), under-sampled (appears to be rotating counter-clockwise), and critically-sampled (appears stationary). In each case rotate the wheel at the same rate and only change the frame rate in the `movie2avi` command (keep the FPS under 30).

Write a Matlab function named `wheel.m` to create a movie showing the spokes image (download it here[1]) rotate clockwise at a constant speed. The function should take parameters to change the frame rate and the speed of the rotation. Save the movies as `wheel-oversample.avi`, `wheel-undersample.avi`, and `wheel-critsample.avi`. Label the plot with the frame rate used for each of the movies and the degrees per frame. Here some tips below to help you get started.

- You will need to use the following Matlab commands: `imread`, `imshow`, `imrotate`, `getframe`, and `movie2avi`.

  NOTE: Passing a negative angle in the `imrotate` command rotates clockwise, and a positive angle rotates counterclockwise.

  Another useful command you can use to help formatting labels for the figure is `sprintf`. For more information use the help system in Matlab.
- Use `myImageRotated = imrotate(myImage, theta, 'bilinear', 'crop')` for the rotate command.
- One way to do this is rotate the image by a number of degrees for each frame. The angle can be split into two variables; `degPerFrame` will be our speed and `theta` will be the actual number of degrees to rotate for the rotate command. Remember to change `degPerFrame` to reflect the same speed when changing the frame rate. Now we can setup a `for` loop something like this,

```
for i = 1:FPS*TIME
    % rotate the image

    % display the image

    % label the plot showing the FPS and speed of the wheel

    pause(0.01)                     % allows time for the plot to draw
    myMovie(i) = getframe(gcf);     % Capture the frame
    theta = theta + degPerFrame;    % Calculate the angle for next frame
end

% save the avi file
```

- Can you use `degPerFrame` to relate to degrees per second? Given some frame rate, how many degrees pass each frame to make a rotation of 360° take 1 second? At a given frame rate, can you calculate the number of frames are needed to last a given amount of time, say 3 seconds?.

[1] http://cnx.org/content/m13687/latest/spokes.tif

- Once your `for` loop is done, you will need to save the movie as an avi to watch it. Use the `movie2avi` function to save the movie. Why can't we just watch the wheel as it is drawing in the `for` loop?

Now try the same problem with a different picture of your choice. Can you get it to appear to move backwards? Save the movie as `myMovie.avi`.

Show the TA the following files:

```
wheel.m
wheel-oversample.avi
wheel-undersample.avi
wheel-critsample.avi
myMovie.avi
```