EE-394 Digital Signal Processing
Bonus Task 01 - Sampling and Aliasing

Saad Mashkoor Siddiqui, EE-16163
Section D, TE-EE, Fall 2018
Submitted To: Mr Muhammd Omar

November 24, 2018

Contents

List of Figures

Listings

1 Objective

To investigate the effects of temporal aliasing in video files using `MATLAB`.

2 Introduction

2.1 Sampling

Sampling is the process of discretizing a continuous-time signal along its time axis. This transforms the a continuous-time, continuous value signal $x_a(t)$ into a discrete-time, continuous-value signal $x(nT_s)$ where n is the sample index and T_s is the sampling period.

The sampling rate $F_s = \frac{1}{T_s}$ determines whether or not the discrete-time signal can be used to recreate the original continuous-time signal without any loss of information.

2.2 Aliasing

Temporal (time-based) aliasing of a continous-time signal $x_a(t)$ with frequency F occurs when it is sampled at a rate that does not fulfill the Nyquist Criteria. Concretely, when the relationship between the sampling frequency F_s and signal frequency F is such that

$$F_s < 2F$$

$$f_d = \frac{F}{F_s} < \frac{1}{2}$$

the samples of signal $x_a(t)$ are practically indistinguishable from those of lower frequency signals upon reconstruction. This process effectively gives the signal $x_a(t)$ an 'alias' or false identity of another signal.

3 Overview

The code described in this report investigates the occurrence of aliasing in a video signal. This is done by first generating a video of a rotating car wheel using the wheel's image (shown in figure 1) and sampling this video at specific intervals.

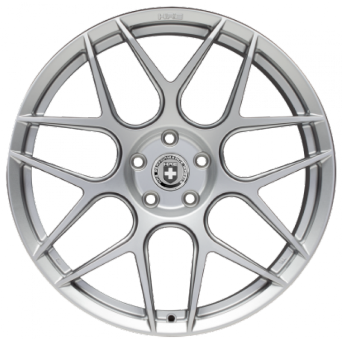

Figure 1: A picture of a wheel that is used to generate videos

Temporal aliasing can be observed by varying the frame rate of the video while keeping the wheel's speed of rotation constant.

4 Task Requirements

Define `MATLAB` function that will read in an image of a wheel, rotor, or similar object, rotate it about its origin at a specified speed, and use each rotation as a single frame of a video. This will create a video of a rotating wheel with both a specified frame rate and a rotation speed. The function must use at least two arguments to vary the frame rate and rotation speed respectively.

Specific requirements include:

- Wheel image must be rotated clockwise.
- Frame rate of video must not exceed 30 FPS.
- Rotation speed must remain constant.

- The following conditions must be investigated
  - Oversampling
  - Critical Sampling
  - Undersampling - Stationary
  - Undersampling - Slow, clockwise rotation
  - Undersampling - Slow, counterclockwise rotation

5 Code and Explanation

5.1 Test Script

Code 1: Test Script

```
%% EE-394 Digital Signal Processing - Bonus Task 1
% Saad Mashkoor Siddiqui, EE-16163, Section D, TE-EE,
    ↪ Fall 2018
% Test File - sampling_test.m
% Tests function generate_movie.m

%% Script

% args - fps, rotation speed, title, duration
generate_movie( 30, 10, 'Oversampling', 5 );
```

Code 1 is a simple test script for invoking the `generate_movie.m` function. The `generate_movie` function itself was originally intended to have only two arguments, `frame_rate` for the video's FPS and `rotation_speed` for the number of circular cycles completed by the wheel per second. During testing, it was necessary to add additional arguments `title` and `duration` for setting the video's heading annotation and duration respectively, as this allowed for more flexibility in experimenting with frame rates, durations, and descriptions.

5.2 Generate Movie Function

Code 2: Generate Movie Function

```matlab
%% EE-394  Digital Signal Processing - Bonus Task 1
% Saad Mashkoor Siddiqui, EE-16163, Section D, TE-EE, Fall 2018
% Function File - generate_movie.m
% generates and saves a video of a rotating wheel

%% Function Definition
function generate_movie( frame_rate, rotation_speed, title, vid_length )
    % read image file - assumes file is in same directory as m script
    wheel = imread( 'wheel.png' );

    % create a video writer object - MPEG 4 profile specified
    video = VideoWriter( title, 'MPEG-4' );

    % set the framerate as defined in argumeont
    video.FrameRate = frame_rate;

    % set duration to argument
    video_duration = vid_length;

    % calculating total frames and total angular displacement
    total_frames = video_duration * frame_rate;
    total_angle = video_duration * rotation_speed * 360;     % 360 deg in 1 rotation

    % amount by which image rotates with each frame
    angle_per_frame = total_angle / total_frames;

    % rotation angle decrements by angle_per_frame with each iter for cw
    img_angle = 0;

    % Open the video writer stream to write frames to it
    open( video );

    % for every frame in the specified duration
    for var = 1:(total_frames + 1 )
        % create annotations for frame
        % angle per frame label
        apf_text = sprintf( 'Angle/Frame = %.2f\n',
   ↪ angle_per_frame );

        % clockwise angular displacement label
        angle_text = sprintf( 'CW Displacement = %.2f\n'
   ↪ , -img_angle );

        % frame rate and rotation speed (function
   ↪ argument) labels
        args = [ sprintf( 'Frames/Second = %.2f\n',
   ↪ frame_rate )...
             sprintf( 'Rotations/Second = %.2f\n',
   ↪ rotation_speed ) ];

        % will be passed as argument to insertText
        caption = [ args, apf_text, angle_text ];

        % Rotate image for new frame
        new_img = imrotate( wheel, img_angle, 'bilinear'
   ↪ , 'crop' );

        % Add title, FPS, rotations/second, angle labels
   ↪  to frame
            % title
        new_img = insertText( new_img, [ 0 0 ], title, '
   ↪ FontSize', 24, 'BoxColor',...
            'red', 'BoxOpacity', 0.4, 'TextColor', '
   ↪ white' );
            % other data
        new_img = insertText( new_img, [ 0 34 ], caption
   ↪ ,'FontSize',18,'BoxColor',...
            'green','BoxOpacity',0.4,'TextColor','white'
   ↪ );

```

```
60          % render frame after rotation
61          imshow( new_img );
62
63          % save currently rendered frame; add to video
64          frame = getframe( gcf );
65          writeVideo( video, frame );
66
67          % decrement angle for next frame - clockwise
      ↪ rotation
68          img_angle = img_angle - angle_per_frame;
69      end
70
71      % close video file stream
72      close( video );
73  end
```

5.2.1 Overview

Code 2 is the `MATLAB` function that generates the videos of the wheel rotating. The function first reads in the `'wheel.png'` image file and stores it in memory (line `9`). It then creates a `MATLAB VideoWriter` object which can be used as an interface to both read from and write to a video file. On line `12`, the `MATLAB VideoWriter` is used to create an MP4 file with the same name as a passed to the function in the `title` argument. The video file's frame rate and duration are set to their corresponding values passed to the function as arguments on lines `15` and `18` respectively.

The function then calculates the total frames (line `21`) and total angular displacement(line `22`) for the duration and frame rate specified, and uses this information to derive angular displacement per frame.

Once the video file stream has been opened on `31`, new frames can be added to the video.

For every frame in the specified duration, the function will rotate the image clockwise by `angle_per_frame`, render the rotated image (line `61`) using `imshow`, and use `MATLAB`'s built-in `gcf` (get current frame) command to save the rotated image to a variable (line `frame (64)`). This `frame` is then written to the video file, and the image's rotation angle is increased in preparation for the next iteration.

Once all frames have been written to the video file, the function closes the connection to the video filestream and terminates. The result is an MP4 video of the rotating wheel image that will exhibit oversampling, critical sampling, or undersampling depending on the relationship between `frame_rate` and `rotation_speed`.

5.3 Comments on Specific Code Elements

imread The `MATLAB` command `imread` is used to read and store digital images into memory. Each image is stored as an array of 3-dimensional information corresponding to each pixel: its x coordinate, y coordinate, and RGB pixel intensity. An ordered collection of these RGB pixels is what constitutes a digital image.

VideoWriter The `VideoWriter` object creates an interface to a video file that can be written to. In this case, the object is used to define both the title of the video file to be written to, as well as the **video profile**, a video formatting specification that determines, amongst other things, video quality, compression, and size. A reference to the `VideoWriter` is also used to set the frame rate of the video to the value specified in the function's argument.

Angle per Frame This is perhaps the single, most important component of the entire function. This quantity links `frame_rate` and `rotation_speed` arguments and, in doing so, establishes a relationship between the **sampling frequency** and **signal frequency**.

$$\begin{aligned} per - frame\ displacement &= \frac{total\ angular\ displacement}{total\ frames} \\ total\ angular\ displacement &= (360)(rotation\ speed)(duration) \\ total\ frames &= (frame\ rate)(duration) \end{aligned}$$

For Loop Bounds `MATLAB` is not a zero-index programming envirionment, which means the first index of any array, list or other iterable always begins at 1. This is why the upper bound of the `for` loop must also be incremented by 1 as in `total_frames + 1`. This results in complete, whole numbers of wheel rotations over the duration of the video.

Creating Annotations `MATLAB` provides functionality to add labels/annotations to a rendered image. The `for` loop makes use of this functionality between lines `35` and `57`, along with the built in formatting string printing functionality `sprintf` to create labels for each frame of the video. These labels show the title (oversampling, undersampling, or critical sampling) of the current sampling investigation, the frame rate and rotation speed passed as arguments to the function, and total angular displacement. This information (2) is useful to illustrate whether or not aliasing is occurring and how visual and actual displacements differ from each other.

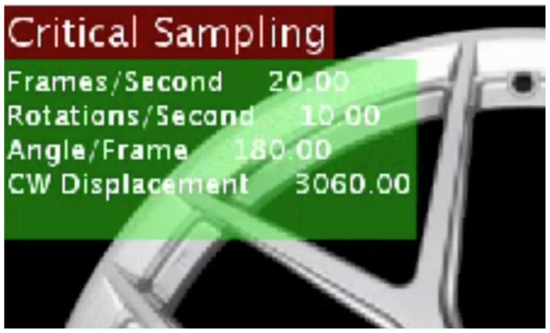

Figure 2: Labels to show sampling type and other relevant data

imrotate One of the most trivial, albeit obstacles in developing this program was to get the image to rotate while maintaining constant frame dimensions. As the image orientation changes, so too does its length and width, which can cause errors when working with `VideoWrite` streams, as they expect all frames to be of the same size. The `crop` argument passed to the `imrotate` command ensures that the frame dimensions remain constant despite rotation by resizing the image with an appropriate scale factor. This is an approximate operation, which is why the wheel in the video does not rotate about exactly the same fixed axis. However, for the purposes of demonstrating aliasing, this has no noticeable effect.

6 Video Analysis

Parameters The values of the `frame_rate`, `rotation_speed`, `duration`, and `title` arguments for each test case are as follows.

Table 1: Video Parameters

Name	FPS	RPS	Direction	Angle	Comments
Oversampling	30	10	Clockwise	120	-
Critical Sampling	20	10	Not evident	180	Worst case
Undersampling(a)	10.03	10	Counterclockwise	358.92	Aliasing
Undersampling(b)	10	10	No rotation	360	Aliasing
Undersampling(c)	9.97	10	Clockwise	361.08	Aliasing

6.1 Sampling Cases

Oversampling In this case, the Nyquist criteria is fulfilled. The sampling frequency (`frame_rate`) is larger than twice the signal frequency(in this case, the `rotation_speed`). As a result, the clockwise rotation of the wheel is representative of the actual angular displacement signal. Neither the angular displacement nor the displacement direction information is lost in this sampling process.

Critical Sampling In this case, the Nyquist criteria is fulfilled, but the sampling frequency is exactly equal to twice the signal frequency. As a result, while the rotation angle information is conserved, the direction is not. There is no way to ascertain whether the wheel is rotating in a clockwise or anticlockwise direction. This is the worst case of sampling - the minimum acceptable quality of sampling.

Undersampling (a) The sampling frequency is just barely above the signal frequency, and far below twice the sampling frequency. The Nyquist criteria is not fulfilled, and aliasing is taking place. As a result, even though the wheel rotates by an angle of 358.92 degrees after every frame, the sampling rate is insufficient to capture this information. Instead, it appears as if the wheel is rotating counterclockwise by 1.08 degrees. Thus, in this case both the direction and angle of rotation has been lost due to aliasing.

Undersampling (b) This video shows the famous 'wagon wheel' effect. The frame rate of the video is exactly the same as the wheel's speed of rotation, which means a new frame is captured every time the wheel completes a single rotation. Combining several of these images results in an optical illusion which makes the wheel seem stationary. However, the angular displacement reading in the annotation continues to increase with each frame, implying that the wheel is rotating and only appears to be stationary.

Undersampling (c) The sampling frequency is slightly lower than the signal frequency. The direction of rotation is still clockwise, but the angular displacement information is lost due to aliasing. The sampling makes it seem as if the wheel rotates counterclockwise by 1.08 degrees with each frame, even though it is rotating with 361.08 degrees.

6.2 Choosing Frame Rates for Undersampling (a)/(c)

To best demonstrate the slow clockwise/counterclockwise rotation characteristic of undersampling and the wagon wheel effect, the wheel must appear to be moving slowly while covering large angular displacements. This is best exemplified when the angular displacement is close to 360 degrees. The method I used to derive the appropriate frame rates was as follows.

$$\begin{aligned}
Let\ video\ duration &= t \\
Let\ total\ frames &= n \\
Let\ angle\ per\ frame &= \angle \\
Let\ total\ angular\ displacement &= \theta = (360)(rotation\ speed)(t) \\
So\ angle\ per\ frame\ is\ \angle &= \frac{\theta}{n} \\
Which\ means\ frames\ n &= \frac{\theta}{\angle}
\end{aligned}$$

$$As\ frame\ rate = \frac{n}{t}$$
$$FPS = \frac{\frac{\theta}{\angle}}{t}$$

Thus, frame rates for undersampling(a) and (c) were derived by assuming per frame angles of 365 and 361, and working backwards form there.

7 Conclusions

7.1 What I Learnt About Sampling

- FPS is analogous to sampling frequency, while rotation speed is analogous to continuous-time signal frequency.
- When critical sampling occurs, it is impossible to ascertain the direction of rotation.
- The slow clockwise/counterclockwise rotation of wheels and rotors always occurs when sampling frequency is close to, but not exactly the same as, the speed of rotation.
- Aliasing can also give the impression that the direction of oscillation of a signal has been reversed, which means there are two pieces of information that can be lost through aliasing.

7.2 What I Learnt About MATLAB

- The `imread, imshow, gcf, VideoWrite, and getframe` commands.
- Passing the `crop` argument to `imrotate` command is a quick fix for enforcing uniform frame size for videos.
- Documentation is a blessing, but one that I tend to forget about.

7.3 Other Observations

- Procrastination fuelled by exam-prep avoidance can be a great motivating tool for learning LaTeX.