ECEn 380—Laboratory Assignment 5
Discrete-Time Filters

Overview

Next semester, in ECEn 390, you will construct a laser tag system comprising two major elements:

1. A hand-held laser tag unit that produces an infrared signal with a sinusoidally varying intensity. The frequency of the sinusoidal variations is unique to each player and is one of ten possible frequencies.
2. An IR detector, an amplifier, an A/D converter, and a signal processor.

The arrangement of these elements is illustrated in Figure 1. The right-hand side of the figure is the receiving end of the system and the signal processing operating on the A/D converter output is the focus of this lab. Of the ten possible frequencies in the signal output by the continuous time circuit in the receiver, the highest is 4,167 Hz. The sample rate must satisfy $F_s > 2 \times 4{,}167 = 8{,}334$ samples/s. As explained in L&G, the sample rate is always higher than the minimum, so a sample rate of $Fs = 10$ ksamples/s is desired. However, operating the A/D converter at 10 ksamples/s imposes strict constraints on the continuous-time anti-aliasing filter that precedes it. To relax the requirements on the continuous-time anti-aliasing filter, the A/D converter operates at 10 times the desired sample rate: $F_s = 100$ ksamples/s. The reasons for this are explained in Appendix A.

A high-level block diagram of the discrete-time signal processing tasks is also shown in Figure 1. The first operation is to “sample” the sample sequence produced by the A/D converter operating at 100 ksamples/s. Before sampling “sampling” the sample sequence, a discrete-time “anti-aliasing” filter is required. This filter is implemented as a finite impulse response (FIR) filter. Sampling a discrete-time sequence is called *downsampling* because it changes the sample rate. The design of discrete-time FIR filters is explained in Appendix B.

After reducing the sample rate to a more manageable 10 ksamples/s, the shot detector and shooter ID are designed. The detector is based on a bank of bandpass filters, each one tuned to one of the possible shooter frequencies in the sampled-data domain. To reduce computational complexity, the filters are implemented as infinite impulse response (IIR) filters. The design of IIR filters is explained in Appendix C.

This lab assignment is organized into three tasks as follows:

Task 1: FIR lowpass filters

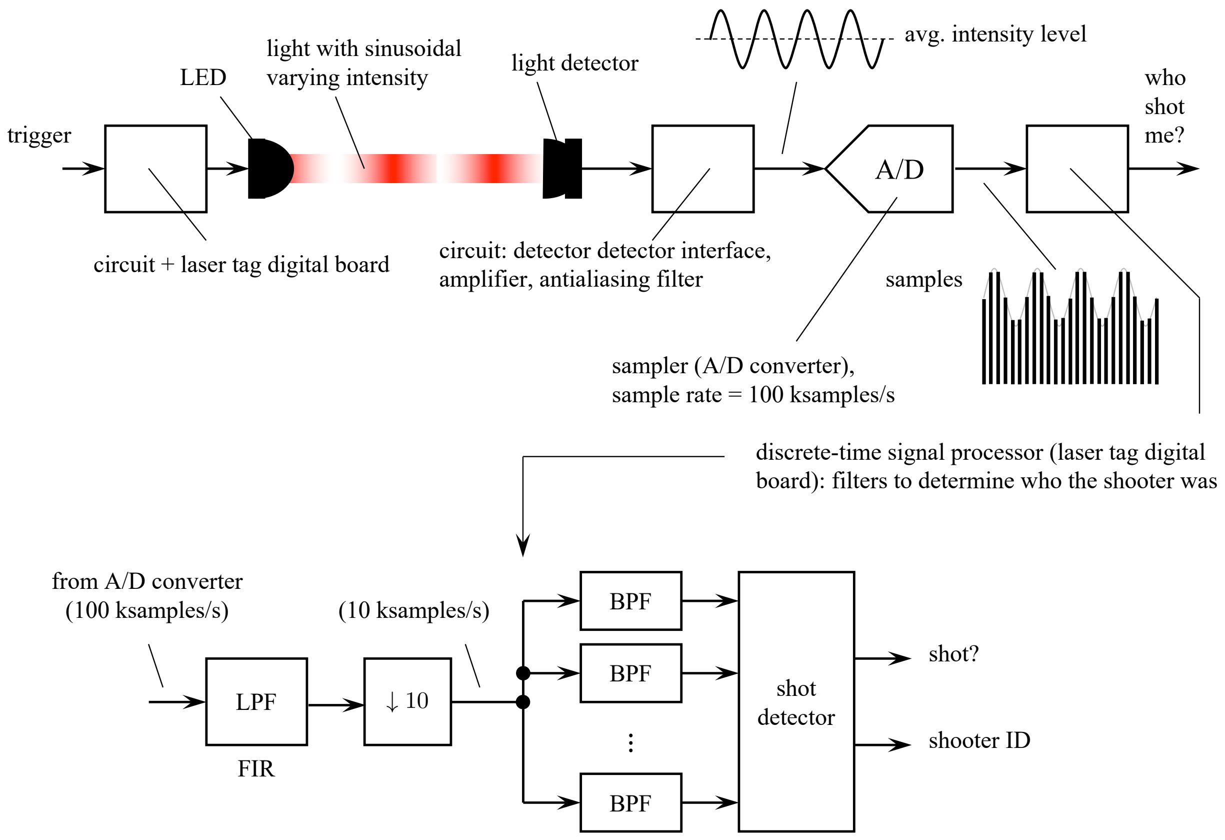
- (a) Design the ideal (IIR) lowpass filter.
- (b) Explore the use of windows to create an FIR filter from the lowpass filter.

Task 2: IIR bandpass filters

- (a) Design a bank of IIR band pass filters for use with the laser tag system.
- (b) Test the filters using sampled data files.

Task 3: Practical realization of the IIR bandpass filter bank

- (a) Poles, zeros, and stability.
- (b) Decomposition into a cascade of second-order sections.
- (c) Testing the cascade of second-order sections.



trigger
LED
light with sinusoidal varying intensity
light detector
avg. intensity level
who shot me?
A/D
circuit + laser tag digital board
circuit: detector detector interface, amplifier, antialiasing filter
samples
sampler (A/D converter), sample rate = 100 ksamples/s
discrete-time signal processor (laser tag digital board): filters to determine who the shooter was
from A/D converter (100 ksamples/s)
(10 ksamples/s)
LPF
FIR
↓ 10
BPF
BPF
BPF
shot detector
shot?
shooter ID

Figure 1: A block diagram of the laser tag system.

Lab Schedule and Submission Requirements

You have **two** lab periods to complete this assignment. The lab exercises are organized into three main tasks described above. Work through the exercises and neatly track your progress in your electronic lab book. Save all of your MATLAB code, and answer all questions in your lab book. Lab book submission is through Learning Suite. The due date is posted on Learning Suite.

Objectives

1. Develop a better understanding how to design an FIR discrete-time filter using windowing.
2. Develop a better understanding how to design an IIR discrete-time filter.
3. Develop an understanding of practical implementation issues for discrete-time IIR filters.

Prelab

Before showing up to your designated lab section, complete the following:

- Read this entire lab assignment and highlight important information that may be hard to find later.

- Be familiar with the concepts from L&G 4.1–4.2, 4.3-3, 4.4, 4.6-3 (cascade realizations) and 4.12-2, and 4.12-3.
- Bring your breadboard, circuit components, and textbook to the lab.

TASK 1

Introduction

Introductory remarks.

Task 1 Procedure

1. Read and understand the advantages and disadvantages of oversampling in Appendix A. Summarize, in your lab notebook, why the laser tag digital system oversamples by as much as it does.
2. Read and understand lowpass filter design using windowing in Appendix B.
3. Use the windowing method to design the FIR lowpass filter you will use in the laser tag system next semester. The design specifications are
   - Continuous-time corner frequency: $f_c = 4600$ Hz.
   - Sample rate: $F_s = 100000$ samples/s.
   - Length = 81.

   To perform the design, complete the following steps:

   (a) Derive the formula for $h_{\text{ideal}}[n]$ for $\Omega_c = \dfrac{2\pi f_c}{F_s} = 2\pi \times 0.046$ rads/sample.

   (b) Explore the DTFT of the windowed filter $h[n] = h_{\text{ideal}}[n]w[n]$ using three different window functions. The window functions available in MATLAB are listed on line at

   `https://www.mathworks.com/help/signal/ug/windows.html`

   on in MATLAB's help documentation

   Documentation → Signal Processing Toolbox → Spectral Analysis → Windows

   The following MATLAB code can be used:

```
N = 40;
n = (-N:N)';         % Make this a column vector to make hideal
                     % a column vector because MATLAB's window
                       % functions produce a column vector.

hideal = myfunc(n); % This is your function to compute hideal.
                    % The equation for hideal was derived
                    % in the previous step.

L = 2*N + 1;
w1 = one of MATLAB's windows of length L;
w2 = another one of MATLAB's windows of length L;
w3 = another one of MATLAB's windows of length L;
```

```
h0 = hideal;         % a simple rectangular window
h1 = hideal .* w1;
h2 = hideal .* w2;
h3 = hideal .* w3;

F = 0:0.001:0.5;     % frequency axis for frequency response plot.
E = exp(1i*2*pi*F); % see comments below
H0 = polyval(h0,E);
H1 = polyval(h1,E);
H2 = polyval(h2,E);
H3 = polyval(h3,E);

figure(11);
subplot(221); stem(n,h0); grid on; ylabel('hideal[n]');
subplot(222); stem(n,h1); grid on; ylabel('h1[n]');
subplot(223); stem(n,h2); grid on; ylabel('h2[n]');
subplot(224); stem(n,h3); grid on; ylabel('h3[n]');

figure(12);
plot(F,20*log10(abs(H)),...
    F,20*log10(abs(H1)),...
    F,20*log10(abs(H2)),...
    F,20*log10(abs(H3)));
grid on;
xlabel('frequency (cycles/sample)'); ylabel('magnitude (dB)');
legend('H0(\Omega)','H1(\Omega)','H2(\Omega)','H3(\Omega)');
```

The DTFT calculations require an explanation. Let $h[n]$ be one of the four impulse responses produced in the MATLAB code segment. The frequency response is the DTFT of $h[n]$

$$H(\Omega) = \sum_{n=-40}^{40} h[n] e^{j\Omega n}. \tag{1}$$

Because $e^{j\Omega n}$ can be written $(e^{j\Omega})^n$, the DTFT may be expressed as

$$H(\Omega) = \sum_{n=-40}^{40} h[n] (e^{j\Omega})^n. \tag{2}$$

This looks like a polynomial in $e^{j\Omega}$ with coefficients $h[n]$. That is, for a given Ω the DTFT is

$$H(\Omega) = h[-40](e^{j\Omega})^{-40} + \cdots + h[0] + \cdots + h[40](e^{j\Omega})^{40}. \tag{3}$$

The MATLAB code segment defines a vector `F`. (Recall Ω is $2\pi\texttt{F}$). The vector `E` in the MATLAB code segment defines the the vector $e^{j\Omega} = e^{j2\pi\texttt{F}}$. The command `polyval` evaluates the degree-81 polynomial with coefficients $h[-40], \ldots, h[40]$ at each element of `E`. Note that in reality `polyval` computes

$$\texttt{polyval} = h[-40] + h[-39](e^{j\Omega}) + h[-38](e^{j\Omega})^2 + \cdots + h[40](e^{j\Omega})^{80} \tag{4}$$

but we want (3). Factoring out $(e^{-j\Omega})^{40}$ from (3) produces

$$\begin{aligned} H(\Omega) &= (e^{j\Omega})^{-40}\left[h[-40] + \cdots + h[0](e^{j\Omega})^{40} + \cdots + h[40](e^{j\Omega})^{80}\right] && (5) \\ &= (e^{j\Omega})^{-40} \times \texttt{polyval}. && (6) \end{aligned}$$

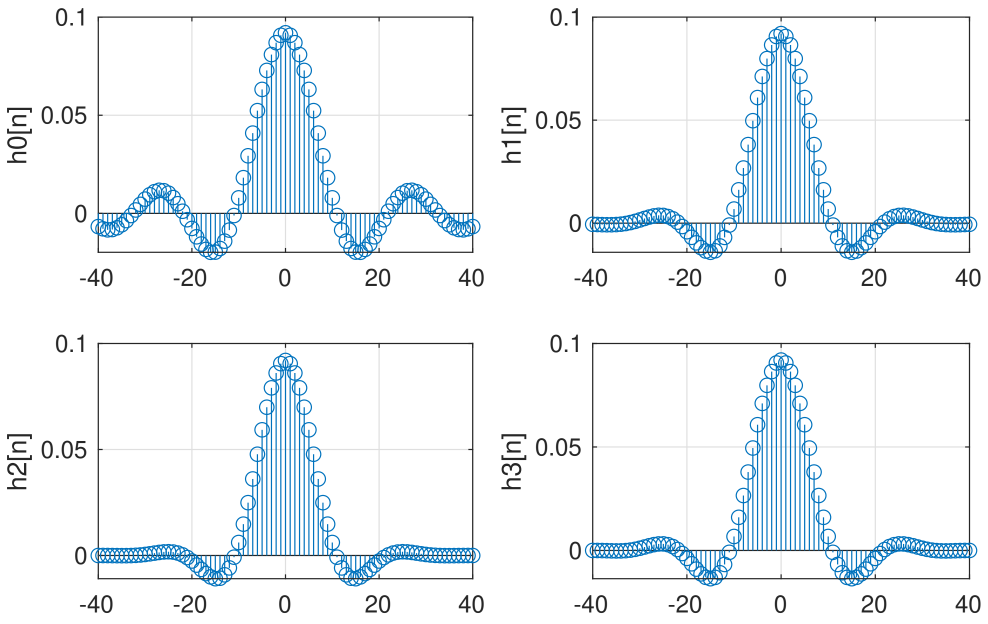


Figure 2: The filter impulse responses produced by the MATLAB code segment.

Because $|e^{j\Omega}| = 1$, $|H(\Omega)| = |\texttt{polyval}|$. Consequently, the differences between (3) and (4) are not important for the *magnitude* of $H(\Omega)$, the quantity of interest here. But the differences are important for the *phase* of $H(\Omega)$.

An example of the plots produced by the MATLAB code segment with three (undisclosed) windows is shown in Figures 2 and 3. Observe that it is hard to tell the differences between the different windows by observing the impulse responses in Figure 2. The frequency-domain plots reveal a lot about how the windows impact the frequency response of the filter. Observe the trade-off between pass band ripple, transition band, and stop band attenuation.

Include the plots you generate for your choice of windows in your lab notebook. Document, in your lab notebook, trade-offs between pass band ripple, transition band, and stop band attenuation. Which window you want to use for the downsampling filter in the laser tag system. Explain your choice in your lab notebook.

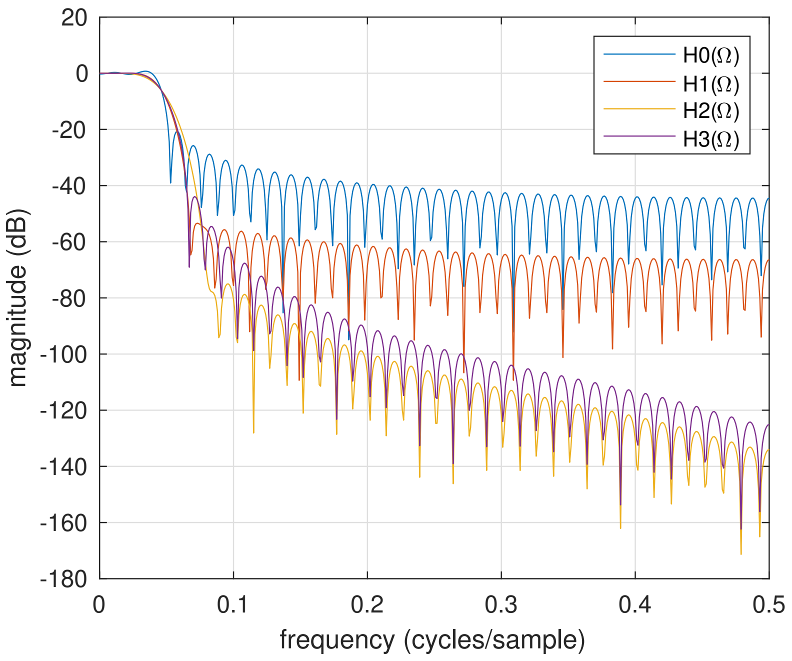


Figure 3: The frequency responses of the filter impulse responses in Figure 2.

TASK 2

Introduction

In this part, ten discrete-time infinite impulse response (IIR) filters are designed. Collectively, these ten filters comprise a "filterbank." The purpose of the filterbank is to monitor the downsampled output from Part 1 to determine if a player was "shot" and, if so, who the shooter was.

The laser gun for each player modulates the laser at a different frequency. These frequencies are

Player	Frequency
1	1,471 Hz
2	1,724 Hz
3	2,000 Hz
4	2,273 Hz
5	2,632 Hz
6	2,941 Hz
7	3,333 Hz
8	3,571 Hz
9	3,846 Hz
10	4,167 Hz

Each filter in the filterbank monitors a small band around one of the ten possible frequencies. An illustration showing $H_1(\Omega), \ldots, H_{10}(\Omega)$ plotted on the same set of axes is illustrated in Figure 4. A shooter may be identified by detecting energy at the frequency corresponding to the shooter's frequency.

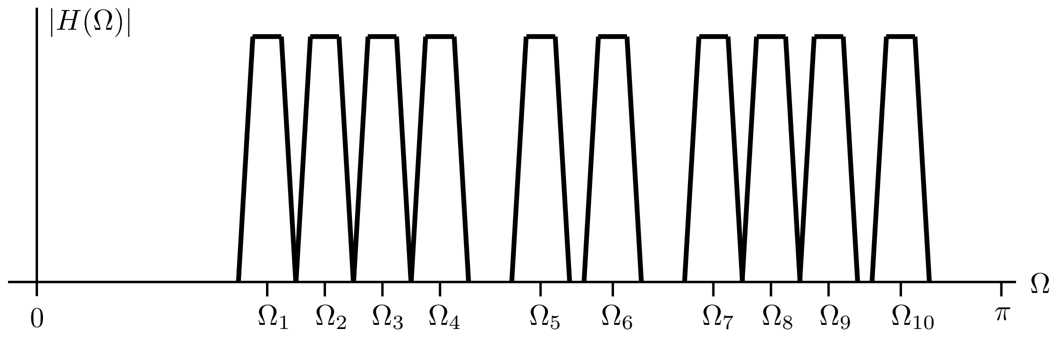


Figure 4: The DTFT magnitudes of the ten filters in the filterbank used to detect a shooter. Following the convention used in filter design, only the positive Ω axis is shown.

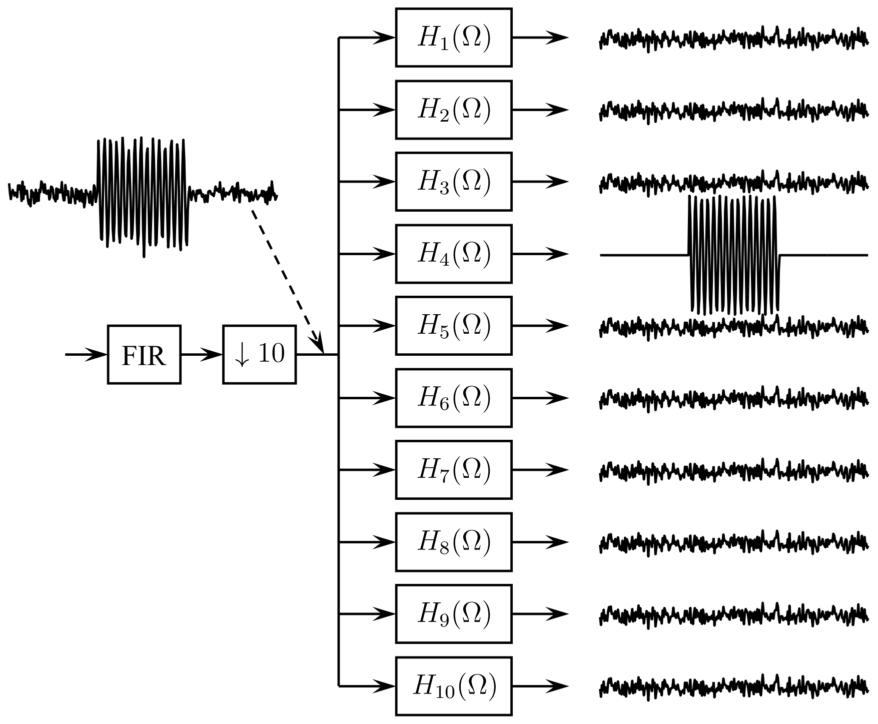


Figure 5: The filterbank comprising ten bandpass filters used to detect a shot and identify the shooter.

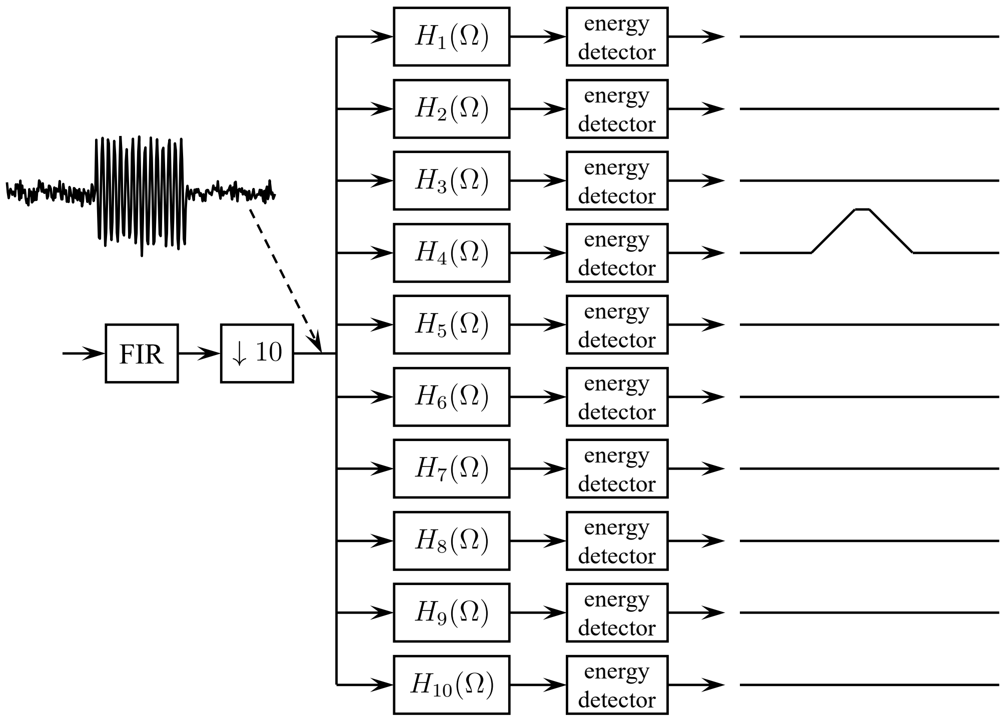


Figure 6: The filterbank comprising ten bandpass filters used to detect a shot and identify the shooter. Here, an energy detector is applied to output of each filter in the filterbank (cf., Figure 5).

The idea is illustrated by the block diagram in Figure 5. The block diagram represents software modules to be implemented in the ESP32 processor on the laser tag digital board in ECEn 390. The output of the downsample-by-10 operation forms the input to the filterbank. Each filter is identified by its DTFT. $H_1(\Omega)$ is the DTFT of a bandpass filter centered at Ω_1; $H_2(\Omega)$ is the DTFT of a bandpass filter centered at Ω_2; and so on.

In the figure, the input to the filterbank shows what a shot looks like in the time domain. There is a segment where the signal amplitude is low: no shot was fired. This segment is followed by a discrete-time sinusoidal signal with relatively large amplitude: this is what a shot looks like. The shot is followed by segment where the signal amplitude is low: no shot was fired. In this example the frequency of the shot is Ω_4. The output of bandpass filter centered described by $H_4(\Omega)$ passes the sinusoidal signal. The other nine bandpass filters do not respond to the sinusoidal input because the frequency of the input does not correspond to the pass band of these filters.

Detecting a shot occurs when the amplitude of one of the bandpass filter outputs is "large." A simple way to monitor the output amplitude is to compute the energy over an interval corresponding to the duration of a shot. Let $y[k]$ be the input to the filterbank, let $y_i[k]$ be the output of the bandpass filter described by the transfer function $H_i(\Omega)$ for $i = 1, \dots, 10$, and let N the the duration of a shot in samples. The energy at the output of $H_i(\Omega)$ is

$$E_i[k] = \sum_{\ell=k-N+1}^{k} y_i^2[\ell]. \tag{7}$$

The filterbank of Figure 5 is modified to include the energy detector in Figure 6. In the example illustrated in the figure, the energy at the output of $H_4(\Omega)$ is the only energy detector output that is "big" during a shot.

The tasks focus on the design of the ten bandpass filters that comprise the filterbank.

Task 2 Procedure

1. Design the bandpass filter filter bank.

   (a) The bandpass filters operate at 10 ksamples/s. Find the frequencies $F_{0,1}, \ldots, F_{0,10}$ cycles/sample using the sample rate and the list of frequencies.

   (b) To simplify the design process, all filters in filterbank use are of the same order and have the same bandwidth. The only difference between them is the center frequency. Find the two frequencies that are the closest. The bandpass filters centered at these two frequencies are the limiting case.

   (c) Let F_a and F_b cycles/sample be the two frequencies that are the closest. Design a pair of discrete-time Butterworth bandpass filters to meet the following specifications:

   - Order of the prototype lowpass filter is not too big.
   - The filters provide at least 40 dB of isolation. The two bandpass filters that are closest together should just meet this requirement. The rest of the filters should easily meet the requirement when designed using the same parameters except for the center frequency.

   I recommend using n and BW (the bandwidth in cycles/sample) as parameters. The following segment of MATLAB code will be useful. (What the MATLAB code segment is doing is explained in Appendix C.)

```
n = something not too big
BW = something small

% design the bandpass filter centered at Fa
F1 = Fa - BW/2;
F2 = Fa + BW/2;
[B1,A1] = butter(n,[F1,F2]*2);

% design the bandpass filter cenetered at Fb
F1 = Fb - BW/2;
F2 = Fb + BW/2;
[B2,A2] = butter(n,[F1,F2]*2);

% plot the magnitude of the frequency responses
% to examine the filter design
F = 0:0.001:0.5;
E = exp(1i*2*pi*F);
H1 = polyval(B1,E) ./ polyval(A1,E);
H2 = polyval(B2,E) ./ polyval(A2,E);

figure(21);
plot(F,20*log10(abs(H1)),F,20*log10(abs(H2))); grid on;
xlabel('frequency (cycles/sample)'); ylabel('magnitude (dB)');
xlim([0 0.5]); ylim([-60 5]);
```

   The design is iterative. Try different values of n (the order) and BW (the bandwidth). An n and BW pair that produces DTFTs whose magnitudes do not touch until the magnitude no greater than -40 dB meets the design criteria. Record your final choice of n and BW in your lab notebook.

(d) Once you have determined a value of n and BW that meet the design requirements, design the remaining eight bandpass filters.

(e) Create a pole-zero plot for each of the ten bandpass filters in the filterbank. The following segment of MATLAB shows how to use the MATLAB function `zplane` to produce the pole-zero plot of the first bandpass filter described by the polynomials whose coefficients form the vectors `B1` and `A1`:

```
figure(22);
zplane(B1,A1);
grid on;
```

The closer the poles are to the unit circle, the more chance there is finite precision arithmetic causes the poles to "migrate" outside the unit circle. How close are the poles to the unit circle?

(f) Plot the magnitudes of the DTFTs of all ten filters on the same set of axis to verify the overall DTFT of the frequency bank looks something like Figure 4.

```
F = 0:0.001:0.5;
E = exp(1i*2*pi*F);
H1 = polyval(B1,E) ./ polyval(A1,E);
H2 = polyval(B2,E) ./ polyval(A2,E);
H3 = polyval(B3,E) ./ polyval(A3,E);
H4 = polyval(B4,E) ./ polyval(A4,E);
H5 = polyval(B5,E) ./ polyval(A5,E);
H6 = polyval(B6,E) ./ polyval(A6,E);
H7 = polyval(B7,E) ./ polyval(A7,E);
H8 = polyval(B8,E) ./ polyval(A8,E);
H9 = polyval(B9,E) ./ polyval(A9,E);
H10 = polyval(B10,E) ./ polyval(A10,E);

figure(23);
plot(F,20*log10(abs(H1)),...
   F,20*log10(abs(H2)),...
   F,20*log10(abs(H3)),...
   F,20*log10(abs(H4)),...
   F,20*log10(abs(H5)),...
   F,20*log10(abs(H6)),...
   F,20*log10(abs(H7)),...
   F,20*log10(abs(H8)),...
   F,20*log10(abs(H9)),...
   F,20*log10(abs(H10)));
grid on;
xlabel('frequency (cycles/sample)'); ylabel('magnitude (dB)');
xlim([0 0.5]); ylim([-60 5]);
```

Include the plot in your lab notebook.

2. Test the bandpass filter bank using the sampled data in `lasertag_data_sets.mat`. There are ten data sets in the `.mat` file. Five of the data sets are easy and five of the data sets are hard. The names of the vectors indicate which is which. Each data set contains one of the ten possible frequencies plus some interference and noise.

   (a) For the first data set, filter the data using all ten of your bandpass filters. Use MATLAB's `filter` function whose use is illustrated in Appendix C.

   (b) Unlike the examples illustrated in Figures 5 and 6, the input to the filter bank does not have a segment where no shot has been fired. Consequently, the energy of the entire filter output vector is of interest. Compute the energy (a scalar) of the output of each of the ten bandpass filters in the filterbank by summing the squares of the filter outputs:

$$E_i = \sum_{\ell=1}^{L} y_i^2[\ell], \quad i = 1, \ldots, 10 \tag{8}$$

where $y_i[\ell]$ is the output of the i-th bandpass filter and L is the length of $y_i[\ell]$.

(c) Create a bar graph (with ten bars) showing the $E_1, \ldots, E_{10}$. Include the bar graph in your lab notebook.

(d) Repeat steps (a)–(c) for the remaining nine data sets in `lasertag_data_sets.mat`.

3. Determine if your filterbank is working by examining the bar graphs created in the previous step. Can you identify a player hit in the sample data? If so, what player in each case? Record your observations in your lab notebook.

TASK 3

Introduction

In Task 2, a bank of IIR bandpass filters was designed. The bandpass filters were of even order and you should have found you needed an order greater than 2. Direct implementation of high-order discrete-time filters can be challenging. This is discussed in L&G Section 5.10-8 (pp. 572–574). The solution is to partition the discrete-time filter into a cascade of second-order filters.

This is exactly what you did to implement the 4-th order Butterworth filter in Lab 4. In Lab 4, you decomposed the 4-th order Butterworth filter into the cascade of two second-order systems and implemented each of the second-order systems using the Sallen-Key circuit.

In this task you will follow the same general idea: decompose each of your high-order bandpass filters into a cascade of second-order discrete-time systems.

Task 3 Procedure

1. Read L&G Section 5.10-8 (pp. 572–574). Record, in your lab notebook, the reason given for decomposing high-order discrete-time IIR filters into a cascade of second-order sections.

2. The ESP32 uses single-precision floats for all computations. Explain how the use of single-precision floats instead of double-precision floats might provide an additional motivation for using a cascade of second-order systems.

3. Decompose each of the ten IIR bandpass filters (of order $2n$) into n second order sections. The following MATLAB code shows how to do this for a simple 4-th order ($n = 2$) bandpass filter centered at frequency for player 1. (Note that this filter does not meet the specifications for the laser tag system. It is used here to illustrate the process.)

```
Fs = 1e4;      % sample rate
n = 2;         % BPF filter order is 2n
BW = 0.05;     % bandwidth
F0 = 1471/Fs; % player 1 frequency cycles/sample
```

```
% create Butterworth IIR bandpass filter
[B,A] = butter(n,[F0-BW/2 F0+BW/2]*2);

% decompose the 4-th order filter into
% 2 second order sections
[sos,g] = tf2sos(B,A);
```

In the MATLAB code segment, the function `tf2sos` (for "transfer function to second order system") produces two outputs.

`sos`	=	A matrix containing the numerator and denominator polynomials that describe each second order section. The format of this matrix is described below.
`g`	=	Gain. The "b_0" coefficient in each of the sections is normalized to 1. The constant `g` is the gain needed to make the cascade of the second order sections have the same gain as the original filter.

The variable `sos` produced by the code segment is

```
sos =

    1.0000   -2.0000    1.0000    1.0000   -0.9157    0.7855
    1.0000    2.0000    1.0000    1.0000   -1.2563    0.8164
```

The first row describes the first of the two second-order sections. The first three numbers in row one are the coefficient of the numerator polynomial. The second three numbers in row one are the coefficients of the denominator polynomial. Thus the first second-order section is described by the transfer function

$$H_1(z) = \frac{B_1(z)}{A_1(z)} = \frac{1 - 2\,z^{-1} + z^{-2}}{1 - 0.9157\,z^{-1} + 0.7855\,z^{-2}}. \tag{9}$$

The second row defines the second second-order section. As before, the first three numbers are the coefficients of the numerator polynomial and the second three numbers are the coefficients of the denominator polynomial:

$$H_2(z) = \frac{B_2(z)}{A_2(z)} = \frac{1 + 2\,z^{-1} + z^{-2}}{1 - 1.2563\,z^{-1} + 0.8164\,z^{-2}}. \tag{10}$$

The gain `g` is 0.0201. Putting the pieces together:

$$\frac{B(z)}{A(z)} = g\frac{B_1(z)}{A_1(z)}\frac{B_2(z)}{A_2(z)}. \tag{11}$$

Perform the decomposition into second order systems for all ten of your IIR bandpass filters.

4. Test the decomposition. The decomposition is tested in both the DTFT domain and the time domain.

   (a) Frequency domain test. The following MATLAB code segment illustrates how to test the frequency response using the filter and its decomposition in step 3.

```
% transfer function for the first second-order section
B1 = sos(1,1:3);
A1 = sos(1,4:6);

% transfer function for the second second-order section
B2 = sos(2,1:3);
A2 = sos(2,4:6);

F = 0:0.001:0.5;
E = exp(1i*2*pi*F);

% frequency response of original bandpass filter
% with transfer fuction B/A
H = polyval(B,E) ./ polyval(A,E);

% frequency response of first second-order system
H1 = polyval(B1,E) ./ polyval(A1,E);

% frequency response of second second-order system
H2 = polyval(B2,E) ./ polyval(A2,E);

% plot the two to see if they match
figure(41);
plot(F,20*log10(abs(H)),F,20*log10(g*abs(H1).*abs(H2)),'.');
grid on;
xlabel('frequency (cycles/sample)'); ylabel('magnitude (dB)');
xlim([0 0.5]); ylim([-60 5]);
```

   This code produces the plot shown in Figure 7. Observe that the two match. Test the frequency response of the cascade of second-order systems for all ten of your IIR bandpass filters. Include the plots in your lab notebook.

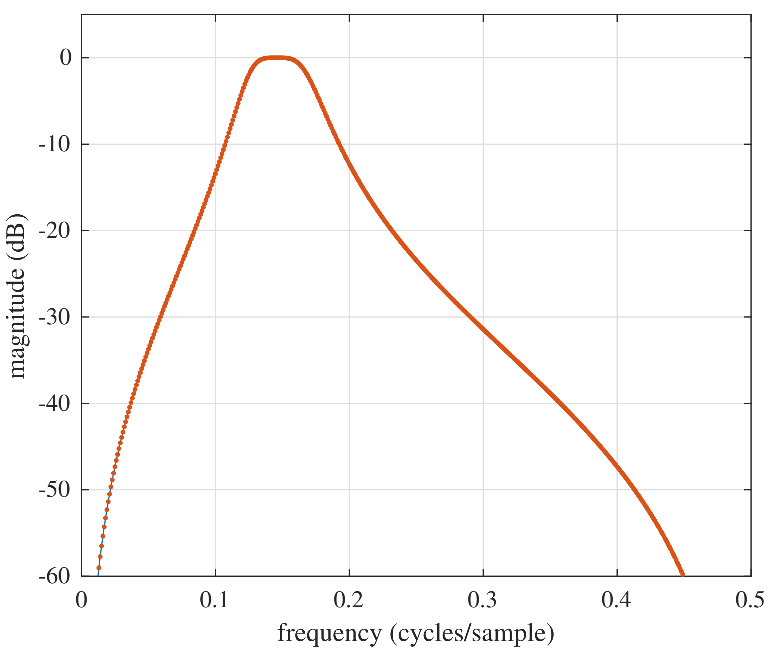


Figure 7: A comparison of the frequency response for the 4-th order IIF Butterworth bandpass filter (solid line) and the overall response of the cascade of two second-order sections (dots).

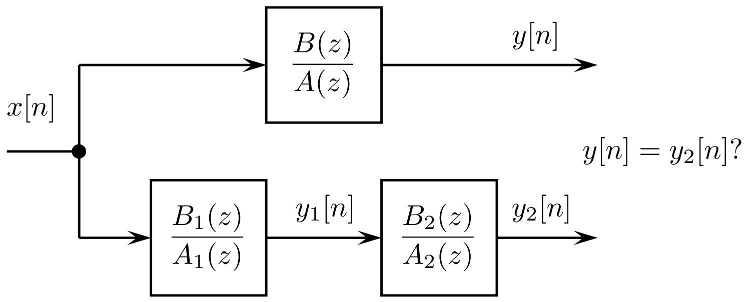


Figure 8: A block diagram illustrating the time domain test for the cascade of second-order systems obtained in for the filter in step 3.

(b) Time domain test. The time-domain test is organized as shown in Figure 8. The following MATLAB code illustrates a method for testing the cascade in the time domain using the the filter and its decomposition in step 3.

```
% transfer function for the first second-order section
B1 = sos(1,1:3);
A1 = sos(1,4:6);

% transfer function for the second second-order section
B2 = sos(2,1:3);
A2 = sos(2,4:6);

% test input
x = randn(1,1000);

% output of original filter
y = filter(B,A,x);

% output due to first second order section
y1 = filter(B1,A1,x);

% output due to second second order section
y2 = filter(B2,A2,y1);

% plot y2 vs. y. If they are the same,
% the plot should produce a diagonal line with unity slope
figure(42);
plot(y,y2,'.'); grid on; axis square;
xlable('y'); ylabel('y2');
```

The code segment produces the plot shown in Figure 9. The data forms a diagonal line with unity slope. This means `y = y2`. Perform this test for all ten bandpass filters and their corresponding cascade of second-order forms. Include the plots in your lab notebook.

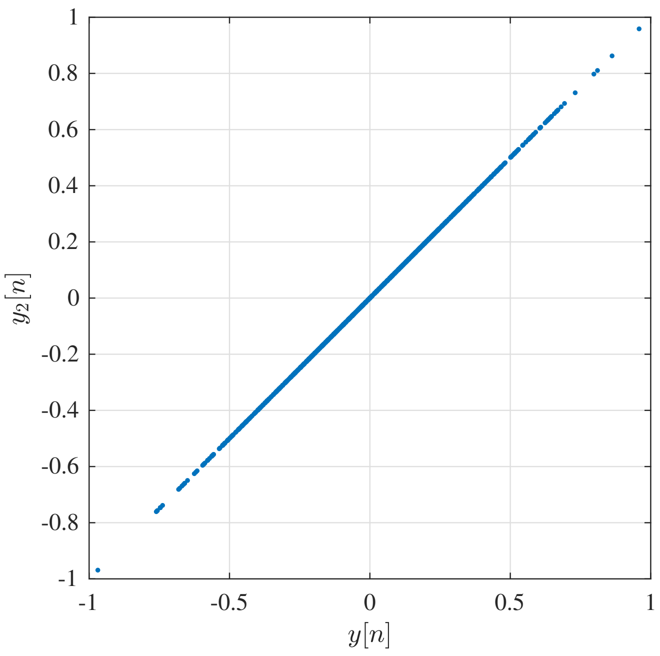


Figure 9: A plot of $y_2[n]$, the output of cascade of the two second order systems derived from the 4-th order Butterworth IIR bandpass filter designed in step 3 vs. $y[n]$, the output of the 4-th order Butterworth IIR bandpass filter.

Appendix A: Oversampling: The Advantages and Disadvantages

In the laser tag receiver, the IR detector output is either noise alone, or one (or more) sinusoids at one (or more) of the frequencies listed in the table in the Task 2 Introduction. Let $x(t)$ be the continuous-time signal representing the IR detector output. Let $X(f)$ be the "f"-version of the continuous time Fourier transform of $x(t)$. A representation of $X(f)$ is shown in Figure 10 (a). The frequencies, from 1,471 to 4,167 Hz are represented by the white box in the figure. The white box spans all ten possible frequencies. This is what one would observe if all ten hand held units were firing at the receiver simultaneously. It will rarely be the case that all ten laser tag units are firing at the receiver simultaneously. The box represents all ten frequencies to remind the reader and system designer what the possibilities are.

Let $x[n] = x(nT)$ be T-spaced samples of $x(t)$. L&G Section 8.1 explores the frequency domain properties of the signal

$$\overline{x}(t) = \sum_{n=-\infty}^{\infty} x(nT)\delta(t - nT). \tag{12}$$

The result is Equation (8.2) in L&G Section 8.1, re-written here in its "f"-version

$$\overline{X}(f) = \frac{1}{T}\sum_{n=-\infty}^{\infty} X\left(f - \frac{n}{T}\right). \tag{13}$$

In words, the relationship (13) says the Fourier transform of $\overline{x}(t)$ comprises periodically-spaced replicas of $X(f)$; the spacing is the sample rate $1/T$.

Let $X(\Omega)$ be the DTFT of the sample sequence $x[n]$. The relationship between the Fourier transform $X(f)$ and the DTFT $X(\Omega)$ is explored in L&G Section 9.5. The relationship leverages $\overline{X}(f)$: $X(\Omega)$ (the DTFT) is a frequency-scaled version of $\overline{X}(f)$.

The discussion in L&G Section 8.1 focuses on *reconstruction*: recovering $x(t)$ from $\overline{x}(t)$. Here, the discussion focuses on the role of the continuous-time anti-aliasing filter and the tradeoff between anti-aliasing filter requirements, sample rate, and discrete-time signal processing.

Suppose the goal is to sample the IR detector output $x(t)$ at 10 ksamples/s. A plot of $\overline{X}(f)$ for this case is shown in Figure 10 (b). The frequency range $-5 \leq f < 5$ kHz is the DTFT for $-\pi \leq \Omega < \pi$ after appropriate scaling of the frequency axis. This is also shown in Figure 10 (b).

If the IR detector output were described by the Fourier transform in Figure 10 (a), then the system design would be simple: sample the IR detector output at 10 ksamples/s and move on. Unfortunately, nature is not this cooperative. The Fourier transform of the IR detector output looks more like the spectrum shown in Figure 11 (a). There is a lot of noise and signals due to other IR sources in the environment. The spectral components of these unwanted signals are shown in gray. If this signal were sampled at 10 ksamples/s, applying (13) produces the spectrum shown in Figure 11 (b). Multiple portions of the unwanted spectrum alias into the desired band. The result is a mess.

The first solution to this problem is to apply a continuous time filter, called an *anti-aliasing filter*, to the IR detector output. A frequency domain representation of this solution is shown in Figure 12 (a). The frequency response of the anti-aliasing filter is shown by $H_a(f)$. If the anti aliasing filter removes all spectral components above the half-sample rate 5 kHz, then the filter output, shown by $Y(f)$ in Figure 12 (b), is the result. Sampling the anti-aliasing filter output produces the pleasing spectrum shown in Figure 12 (c).

The requirements for the anti-aliasing filter in Figure 12 (a) are

1. The pass band goes from below 1,471 Hz to just above 4,167 Hz.
2. The stop band must attenuate everything above 5 kHz.

The transition band of this filter is only $5{,}000 - 4{,}167 = 833$ Hz. The only way to accomplish this is to use a (very) high-order all-pole filter. If Lab 4 taught you anything, it was that high-order continuous-time filters are hard.

The alternative is to reduce the order of the anti-aliasing filter sample the IR detector output at a higher sample rate. The reason this might work is illustrated by considering the case where the sample rate is increased to 50 ksamples/s. In this case, the anti-aliasing filter must now eliminate (or sufficiently attenuate) its input above 25 kHz. The situation is illustrated in Figure 13 (a). Instead of requiring an anti-aliasing filter with a transition band of 833 Hz, this new approach requires an anti-aliasing filter with a transition band of $25{,}000 - 4{,}167 = 20{,}833$ Hz. This is much easier to accomplish.

Note that the anti-aliasing filter output $y(t)$, whose spectrum is shown in Figure 13 (b), contains an attenuated portion of the unwanted spectrum. This is acceptable because the removal of spectral components above 25 kHz eliminates aliasing. The fact that no aliasing occurs is shown in Figure 13 (c). This is why the filter is called an *anti-aliasing* filter.

The DTFT of the sample sequence $y[n] = y(nT)$ is also indicated in Figure 13 (c). (Note a frequency-axis scaling must be applied to obtain the DTFT frequencies—see L&G Section 9.5.) The gray business in the DTFT is still unwanted, and it must be removed by filtering. But the required filtering can be performed in discrete-time processing. It is easier to design and implement a good lowpass filter in discrete-time processing than it is in continuous-time processing. This is the purpose of the filter-and-downsample operation in Task 1. (Except the sample rate is 100 ksamples/s instead of 50 ksamples/s used here to illustrate the concepts.)

Oversampling as a method to reduce the anti-aliasing filter requirements is a good way to go, *as long as the discrete-time signal processor can support operation at the high sample rate*. Your professors have worked through the tradeoffs and found that the ESP 32 can support an input signal at 100 ksamples/s as long as the sample rate is immediately reduced 10 ksamples/s before doing any further processing.

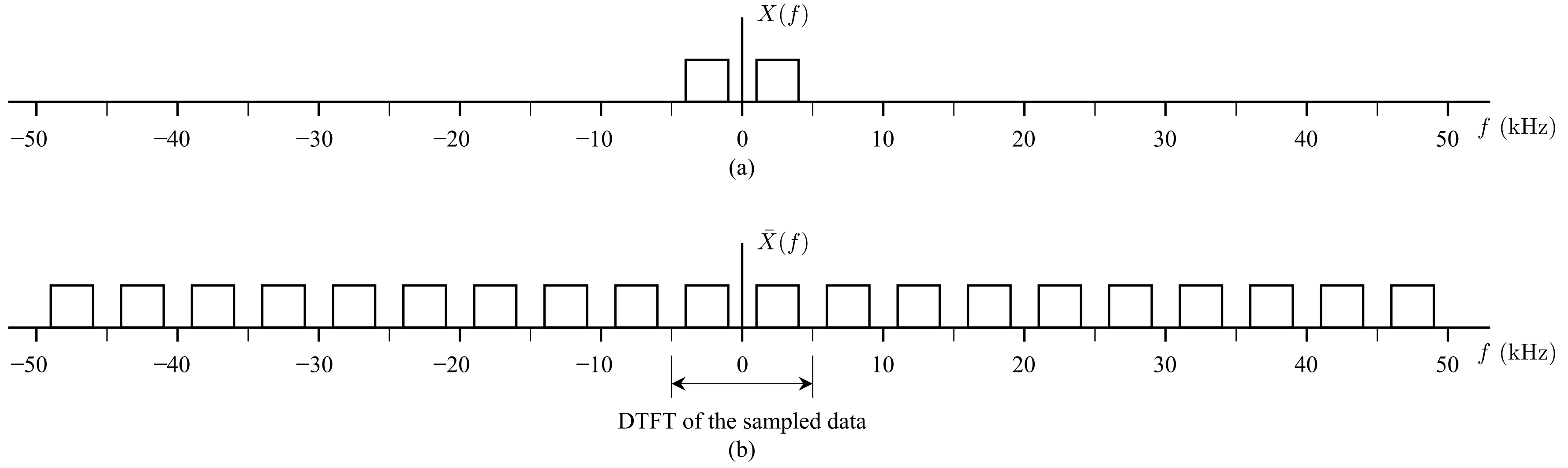


Figure 10: An example of sampling the IR output $x(t)$ at 10 ksamples/s: (a) an idealized view of the Fourier transform of the IR detector output; (b) the Fourier transform of the corresponding $\overline{x}(t)$.

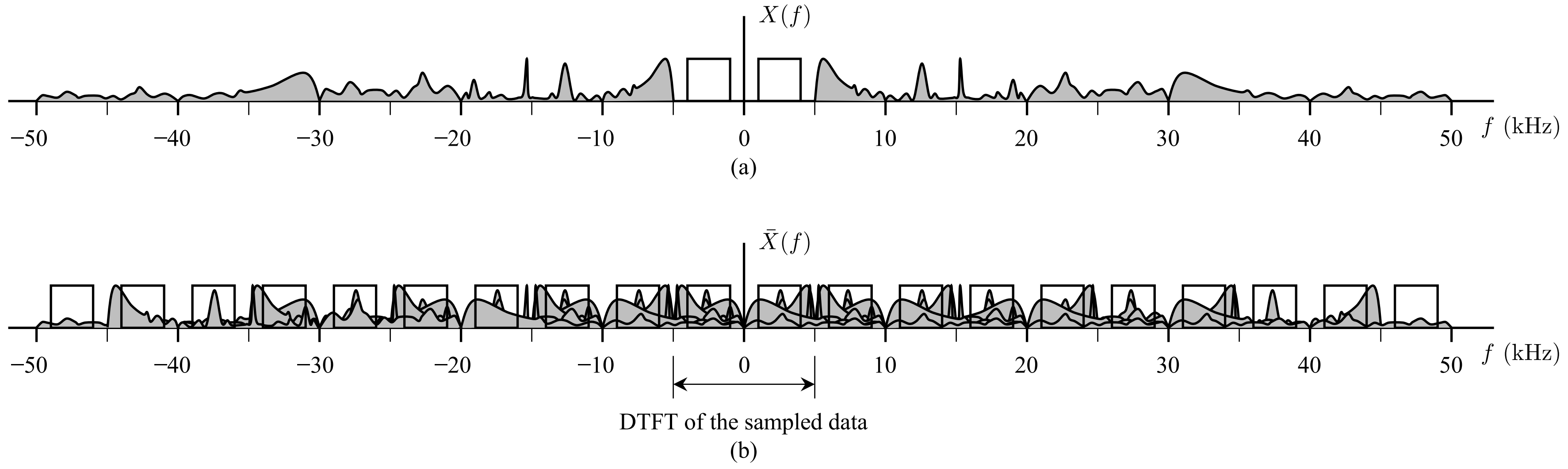


Figure 11: An example of sampling the IR output $x(t)$ at 10 ksamples/s: (a) a more realistic Fourier transform of the IR detector output: (b) the Fourier transform of the corresponding $\overline{x}(t)$.

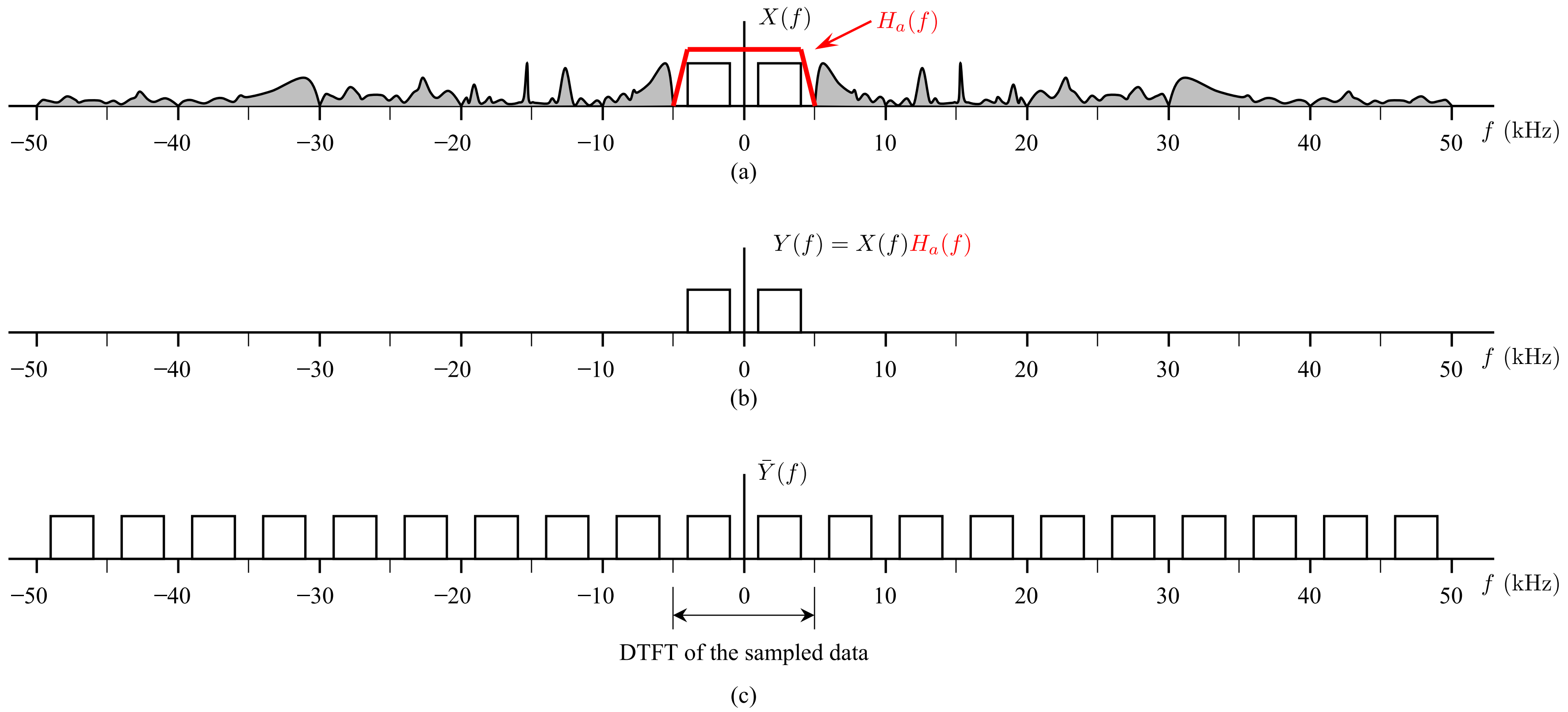


Figure 12: An example of sampling the IR output $x(t)$ at 10 ksamples/s: (a) a realistic Fourier transform of the IR detector output along with the frequency response of an anti-aliasing filter; (b) the Fourier transform $Y(f)$ of the anti-aliasing filter output $y(t)$; (c) the Fourier transform of the corresponding $\overline{y}(t)$.

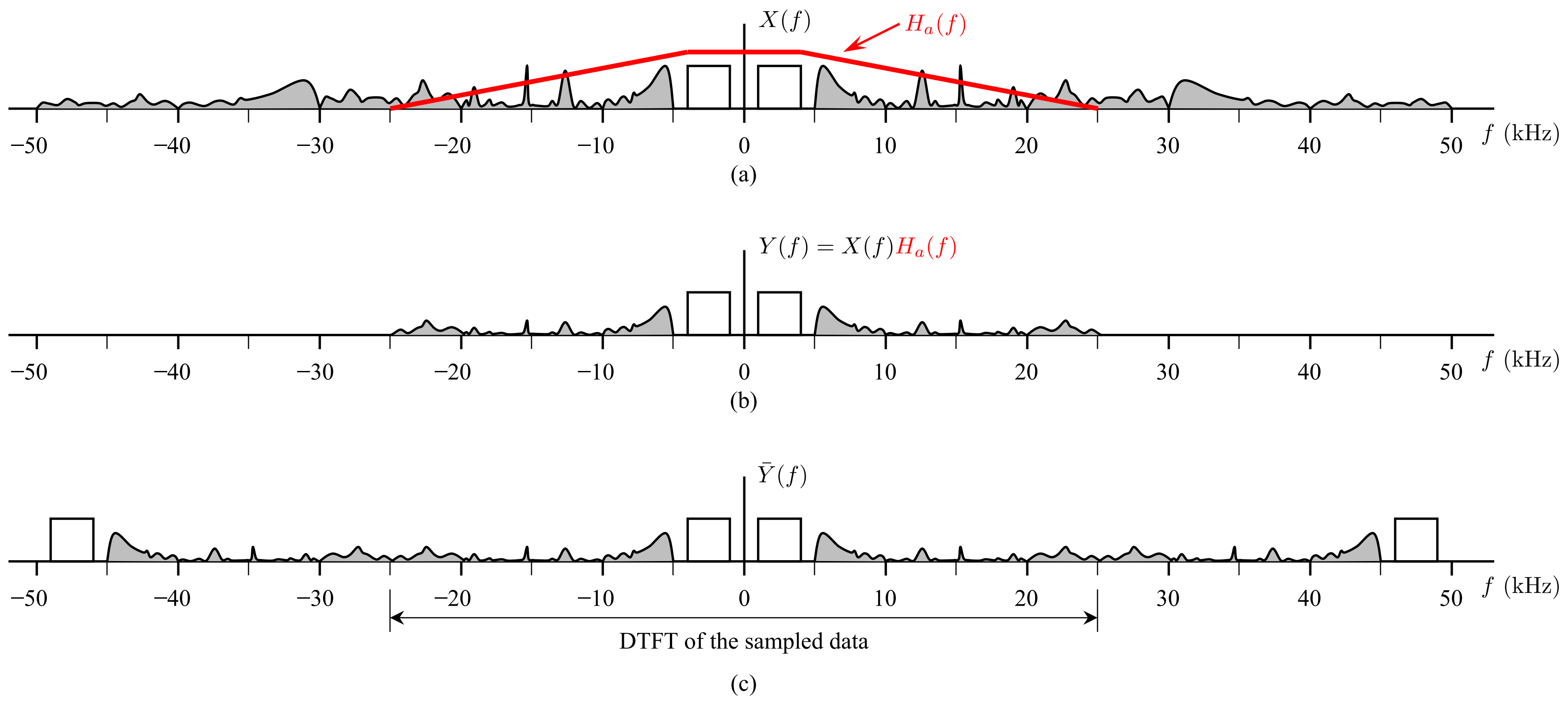


Figure 13: An example of oversampling the IR output $x(t)$ at 50 ksamples/s: (a) a realistic Fourier transform of the IR detector output along with the frequency response of an anti-aliasing filter; (b) the Fourier transform $Y(f)$ of the anti-aliasing filter output $y(t)$; (c) the Fourier transform of the corresponding $\overline{y}(t)$.

Appendix B: FIR Lowpass Filter Design Using Windowing

A discrete-time filter is described in the discrete-time frequency domain by its DTFT $H(\Omega)$, where Ω has units rads/sample. Because $H(\Omega)$ is a DTFT, it is periodic in Ω with period 2π. In the plots shown below, it is customary to only plot one period of $H(\Omega)$ and requires the reader to imagine the infinite number of periodic replicas. All of the work in this lab will uses the period $-\pi \leq \Omega < \pi$ rads/sample, or the postive frequency half of it $0 \leq \Omega < \pi$ rads/sample.

The inverse DTFT is the impulse response $h[n]$. The impulse response is, in general, not periodic. There are two broad classes of discrete-time filters:

- The impulse response is zero for all n outside the interval $N_1 \leq n \leq N_2$ for finite integers N_1 and N_2. The length of the impulse response is $N_1 + N_2 + 1$. Because the length of the impulse response is finite, this filter is called a *finite impulse response* (FIR) filter. Note that the adjective *finite* modifies the length of the impulse response, not its amplitude. The most commonly used FIR filters have $N_1 = -N_2$. A common notation is to designate $N = N_2$ so that $h[n] = 0$ outside the interval $-N \leq n \leq N$. The length filter is $L = 2N + 1$. The reason this is common is that it is easy to produce a symmetric filter that has *linear phase*: one of two requirements for distortionless transmission described in L&G Section 9.4-1. Symmetric filters possess symmetry about $n = 0$.

- If the impulse response is defined for $-\infty \leq n < \infty$, the length of the impulse response is infinite. Such a filter is called an *infinite impulse response* (IIR) filter. Note that the adjective *infinite* modifies the length of the impulse response, not its amplitude. The most commonly used IIR filters are *causal*. A causal IIR filter is one where the impulse response is of the form

$$h[n] = \begin{cases} 0 & n < 0 \\ \text{something} & n \geq 0. \end{cases} \tag{14}$$

  Causal IIR filters produce a time-domain input/output relationship that is recursive. It is very difficult (impossible?) to produce an IIR filter with linear phase.

The appendix focuses on the design of FIR filters. The next appendix focuses on the design of IIR filters.

A discrete-time FIR filter is described by the $L = 2N + 1$ *coefficients*

$$h[-N], h[-N+1], \cdots, h[0], \cdots, h[N-1], h[N].$$

The filter output, due an input $x[n]$, is given by the convolution sum

$$y[n] = \sum_{k=-N}^{N} h[k]x[n-k]. \tag{15}$$

FIR filter *design* means calculating the L coefficients to meet a given set of requirements. The requirements are usually given in the discrete-time frequency domain using the DTFT. When the filter is intended for use with sampled data, it is sometimes the case that the filter requirements are specified in the continuous time frequency domain using the (continuous-time) Fourier transform. In such cases, the continuous-time frequency domain characteristics plus the sample rate at which the filter must operate are enough to derive the discrete-time frequency domain requirements.

Of the many discrete-time FIR filter design techniques, you will use a technique called "windowed filter design" because it does not require any principles beyond those taught in ECEn 380. First, some

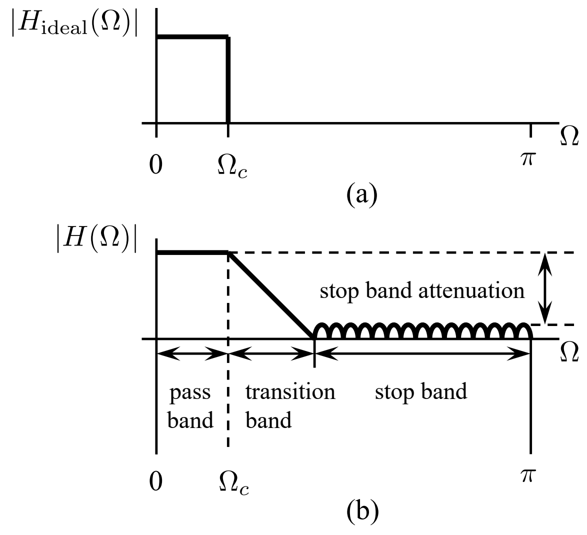


Figure 14: Frequency-domain characterizations of a lowpass filter: (a) The transfer function of an ideal lowpass filter; (b) The transfer function of a practical lowpass filter.

terminology. The transfer function of an ideal lowpass filter with corner frequency Ω_c rads/sample is, for $-\pi < \Omega < \pi$,

$$H_{\text{ideal}}(\Omega) = \begin{cases} 0 & -\pi \le \Omega < -\Omega_c \\ 1 & -\Omega_c < \Omega < \Omega_c \\ 0 & \Omega_c < \Omega < \pi \end{cases} \tag{16}$$

The transfer function for the ideal lowpass filter is illustrated in Figure 14 (a). Following the conventions in filter design, only the positive Ω axis is shown in the figure. A practical lowpass filter is shown in Figure 14 (b). There are two important differences between the ideal lowpass filter and a practical lowpass filter.

- A practical lowpass filter is characterized by three frequency bands: the pass band, the transition band, and the stop band. The pass band approximately corresponds to the frequency range where the ideal lowpass filter is 1. The stop band approximately corresponds to the frequency range where the ideal lowpass filter is zero. Whereas the ideal lowpass filter transitions from pass band ($|H(\Omega)| = 1$) to stop band ($|H(\Omega)| = 0$) over a zero-width interval on the Ω axis, a practical filter requires an interval of non-zero width on the Ω axis. The non-zero width interval is called the transition band. *All practical lowpass filters require a transition band.*
- Another important difference is the stop band. The magnitude of transfer function of an ideal lowpass filter is zero in the stop band. Filter designers say "the filter has a stop band gain of zero (or $-\infty$ dB)." The magnitude of transfer function of a practical lowpass filter in the the stop band is a non-zero value that is (hopefully) much less than the magnitude of the transfer function in the pass band. The ratio of the magnitude of the transfer function in the pass band to the magnitude of the transfer function in the stop band is called the *stop band attenuation.* The stop band attenuation is almost always expressed in dB. Good practical lowpass filters have a stop band attention in the 40 to 60 dB range.

The impulse response of the ideal lowpass filter is given by the inverse DTFT of $H_{\text{ideal}}(\Omega)$ defined in (16). The impulse response $h_{\text{ideal}}[n]$ is the DTFT transform pair no. 8 in L&G Table 9.1 (p. 860). An example of the computations required to compute the inverse DTFT of $H_{\text{ideal}}(\Omega)$ is given in Example 9.6 of

L&G (pp. 865–866). The property of $h_{\text{ideal}}[n]$ that stands out the most is that it is defined for $-\infty < n < \infty$: the ideal lowpass filter is an IIR filter.

To create an FIR version of the lowpass filter, early discrete-time filter designers observed that $h_{\text{ideal}}[n]$ becomes very small as $|n|$ becomes large. Consequently, the first FIR filters were truncated versions of $h_{\text{ideal}}[n]$:

$$h[n] = \begin{cases} h_{\text{ideal}}[n] & -L \leq n \leq L \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

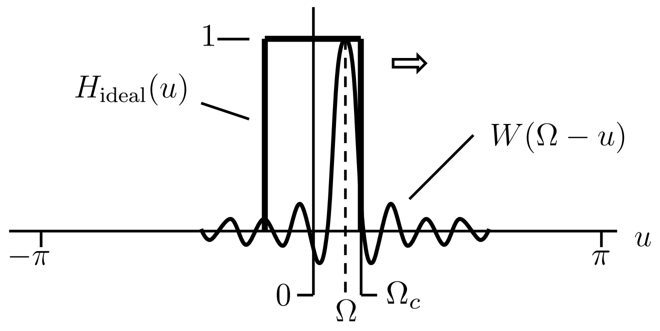
The results were not always what one hoped. A short time later it was observed that the truncation in (17) is a form of time-domain *windowing*. That is $h[n] = h_{\text{ideal}}[n]w[n]$ where

$$w[n] = \begin{cases} 1 & -L \leq n \leq L \\ 0 & \text{otherwise.} \end{cases} \tag{18}$$

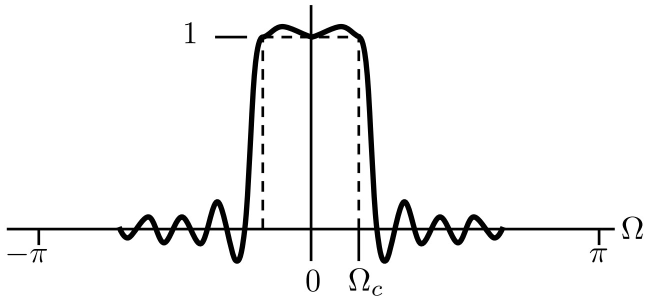
The impact of time-domain windowing on the frequency response of lowpass filter is best understood by thinking of windowing as multiplication: $h[n] = h_{\text{ideal}}[n]w[n]$. The Frequency Convolution property in Table 9.2 shows that multiplication in the time domain is convolution in the frequency domain:

$$h[n] = h_{\text{ideal}}[n]w[n] \quad \Leftrightarrow \quad H(\Omega) = \frac{1}{2\pi}\int_{2\pi} H_{\text{ideal}}(u)W(\Omega - u)du$$

The "flip and slide" operation defined by convolution is illustrated below.



The ideal window is a frequency-domain impulse. Practical windows display a (hopefully narrow) main lobe and (hopefully small) side lobes in the frequency domain. A narrow main lobe and small side lobes are competing demands and the variety of windows offer tradeoffs between main lobe width and side lobe height. The passband response of $H(\Omega)$ is generated as the main lobe of $W(\Omega)$ slides through the pass band of $H_{\text{ideal}}(\Omega)$. The transition band of $H(\Omega)$ is created as the main lobe of $W(\Omega)$ exits the pass band of $H_{\text{ideal}}(\Omega)$. Thus, the width of the transition band is approximately equal to the width of main lobe of $W(\Omega)$. The stop band of $H(\Omega)$ is created as the side lobes of $W(\Omega)$ slide through the pass band. Thus, the stop band attenuation is determined approximately by the ratio of main lobe height to side lobe height. The result of the frequency domain convolution is illustrated below.



The small ripples in both the pass band and the stop band are due to the side lobes in $W(\Omega)$.

Appendix C: IIR Filter Design: Lowpass Filters, Bandpass Filters, and Stability Issues

An infinite impulse response (IIR) filter is a filter whose impulse response is infinitely long. Most discrete-time IIR filters are causal:

$$h[n] = \begin{cases} 0 & n < 0 \\ \text{something} & n \geq 0. \end{cases} \tag{19}$$

Given the infinitely long impulse response, the filter output $y[n]$, due an input $x[n]$, is given by the convolution sum

$$y[n] = \sum_{k=0}^{\infty} h[k]x[n-k]. \tag{20}$$

From this point of view, an IIR filter is described either by an infinite list of numbers $h[n]$ or by a formula for computing $h[n]$ that is valid for $0 \leq n < \infty$.

The most important class of IIR filters are those described by the z-domain transfer function

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{a_0 + a_1 z^{-1} + \cdots + a_N z^{-N}}. \tag{21}$$

Comments:

1. The z-transform is denoted $H(z)$ instead of L&G's "$H[z]$." The L&G notation is *not* the convention in the discipline.

2. See L&G Section 5.4 for a discussion of systems of the form (21).

3. The transfer function $H(z)$ is the ratio of two polynomials in z^{-1}. Note that L&G (5.29) is the ratio of two polynomials in z. Defining $H(z)$ in terms of polynomials in z^{-1} is the convention in discrete-time signal processing and is the definition used by the MATLAB tools you will be using. Polynomials in z are encountered in digital control.

4. The form (21) is called a *rational form* where the word rational refers to the *ratio* of two polynomials.

5. Following the convention in discrete-time signal processing, the two polynomials that defined $H(z)$ are

$$\begin{aligned} B(z) &= b_0 + b_1 z^{-1} + \cdots + b_M z^{-M} \\ A(z) &= 1 + a_1 z^{-1} + \cdots + a_N z^{-N} \end{aligned} \tag{22}$$

   Note that following convention, a_0 in (21) is 1. Using the polynomial definitions (22), the transfer function (21) may be expressed as

$$H(z) = \frac{B(z)}{A(z)}. \tag{23}$$

6. $B(z)$ is a degree-M polynomial in z^{-1}. Consequently, $B(z)$ has M roots. These M roots are the *zeros* of $H(z)$.

7. $A(z)$ is a degree-N polynomial in z^{-1}. Consequently, $A(z)$ has N roots. These N roots are the *poles* of $H(z)$.

A filter whose transfer function is of the form (21) is described by two lists of numbers:

$$\{a_1, \ldots, a_N\} \qquad \{b_0, b_1, \ldots, b_M\}. \tag{24}$$

These two lists comprise the *filter coefficients* of the IIR filter. Thus, an IIR filter, whose transfer function is a rational function, is described completely by $N + M + 1$ coefficients.

The input/output relationship for an IIR filter defined by a rational transfer function is derived as follows. Let $X(z)$ be the z-transform of the input $x[n]$ and let $Y(z)$ be the z-transform of the output $y[n]$. Then

$$Y(z) = X(z)H(z) = X(z)\frac{B(z)}{A(z)} \tag{25}$$

$$\Rightarrow A(z)Y(z) = B(z)X(z) \tag{26}$$

$$\Rightarrow \left[1 + a_1 z^{-1} + \cdots + a_N z^{-N}\right] Y(z) = \left[b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}\right] X(z) \tag{27}$$

$$\Rightarrow Y(z) + a_1 z^{-1} Y(z) + \cdots + a_N z^{-N} Y(z) = b_0 X(z) + b_1 z^{-1} X(z) + \cdots + b_M z^{-M} X(z) \tag{28}$$

Computing the inverse z transform on a term-by-term basis produces

$$y[n] + a_1 y[n-1] + \cdots + a_n y[n-N] = b_0 x[n] + b_1 x[n-1] + \cdots + b_M x[n-M]. \tag{29}$$

Moving all but the $y[n]$ term to the other side of the equation produces

$$y[n] = -a_1 y[n-1] - \cdots - a_n y[n-N] + b_0 x[n] + b_1 x[n-1] + \cdots + b_M x[n-M]. \tag{30}$$

Equation (30) defines a *recursion*: the n-th output is a linear combination of the N previous outputs plus a linear combination of the current and M previous inputs. The linear combinations are defined by the sets of filter coefficients (24).

IIR filter *design* means calculating the $N + M + 1$ filter coefficients to meet a given set of requirements. Because, in general, $N + M + 1 \ll \infty$, the class of filters defined by rational transfer functions is easier to design: fewer numbers have to be calculated.

There are a number of design procedures for discrete-time IIR filters. One of the first to be developed is based transforming a continuous-time filter designed using well-established continuous-time designs, such as the Butterworth filter. This is because (1) all continuous-time filter designs produce stable systems, and (2) all continuous-time filter designs have a continuous-time impulse response $h(t)$ that has infinite duration.

The design method is based on transforming an s-domain transfer function describing a continuous-time filter to a z-domain transfer function describing the corresponding discret-time filter. The transformation is based on a conformal mapping called the "bilinear transform." The bilinear transform defines the relationship between the complex variable s and the complex variable z as

$$s = \frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}} \tag{31}$$

where T s/sample is the sample time. In the context of filter design, the bilinear transform has two important properties:

1. The imaginary axis in the s-plane maps to the unit circle in the z-plane. The imaginary axis in the s-plane is the radian frequency axis. This feature is illustrated in Figure 15 (a). If the region of

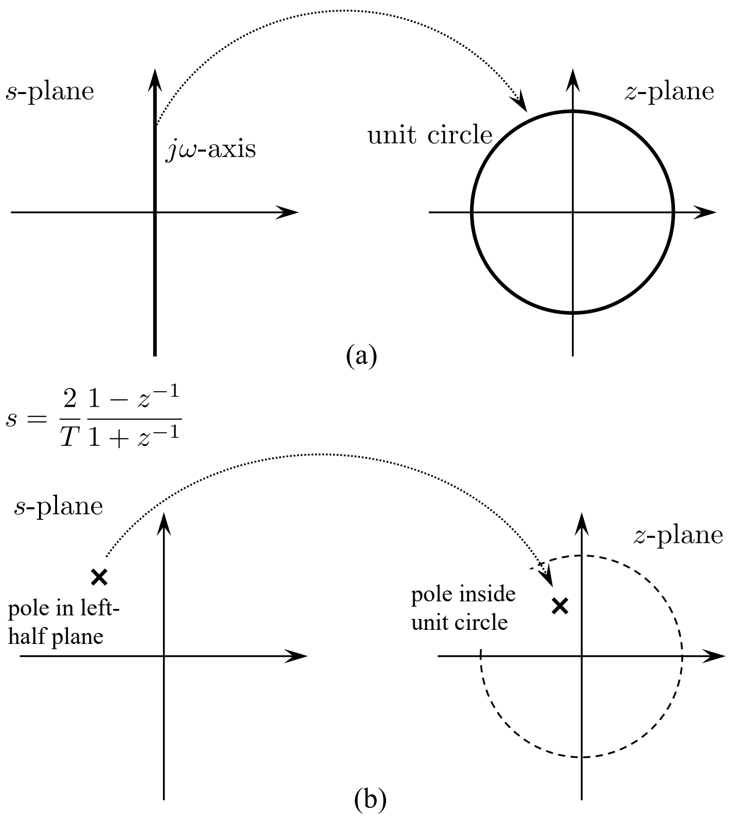


Figure 15: The conformal mapping defined by the bilinear transform (31): (a) the imaginary axis in the s-plane maps to the unit circle in the z-plane; (b) all point in the left-half s-plane map to points inside the unit circle in the z-plane.

convergence for the Laplace transform includes the imaginary axis then the Fourier transform is the Laplace transform evaluated at $s = j\omega$. The unit circle in the z-plane is the discrete-time frequency axis. If the region of convergence for the z-transform includes the unit circle, then the DTFT is the z-transform evaluated at $z = e^{j\Omega}$. Consequently, *the bilinear transform maps the continuous-time frequency axis to the discrete-time frequency axis.*

2. All points in the left-half s-plane (i.e., the region described by $\mathrm{Re}\{s\} < 0$) map to points inside the unit circle in the z-plane. Because a pole is point in either the s- or z-planes, the bilinear transform maps s-plane poles in the left-half s-plane to z-plane poles inside the unit circle. This feature is illustrated in Figure 15 (b). The transfer functions of causal, stable continuous-time systems have poles in the left-half s-plane. The transfer function of causal, stable discrete-time systems have poles inside the unit circle in the z-plane. Consequently, *the bilinear transform transforms a causal, stable continuous-time system to a causal, stable discrete-time system.*

Because the mapping of the $j\omega$-axis in the s-plane to the unit circle in the z-plane is nonlinear, the points on the $j\omega$-axis are "warped" in the mapping. In the filter design methods described below, the continuous-time frequencies are "pre-warped" so that the nonlinear mapping from $s = j\omega$ to $z = e^{j\Omega}$ produces the desired outcome.

The discrete-time IIR lowpass filter design method based on transforming a continous-time filter design works as follows.

1. Specifications: define the filter order n and the corner frequency Ω_c rads/sample.
2. Design the normalized continuous-time filter. The transfer function is of the form
$$H_c(s) = \frac{B_c(s)}{A_c(s)}. \tag{32}$$
3. Given a sample rate $1/T$ samples/s, compute $\omega_c = \Omega_c/T$ rads/s.
4. Pre-warp ω_c to produce $\tilde{\omega}_c$ given by
$$\tilde{\omega}_c = \frac{2}{T}\tan\left(\frac{\omega_c T}{2}\right). \tag{33}$$
5. Make the corner frequency of the continuous-time filter $\tilde{\omega}_c$:
$$H_c\left(\frac{s}{\tilde{\omega}_c}\right) = \frac{B_c\left(\frac{s}{\tilde{\omega}_c}\right)}{A_c\left(\frac{s}{\tilde{\omega}_c}\right)} \tag{34}$$
6. Apply the bilinear transform to convert the continuous-time system described by $H_c(s/\tilde{\omega}_c)$ to a discrete-time system described by $H_d(z)$:
$$H_d(z) = H_c\left(\frac{2}{\tilde{\omega}_c T}\frac{1-z^{-1}}{1+z^{-1}}\right) = \frac{B_c\left(\frac{2}{\tilde{\omega}_c T}\frac{1-z^{-1}}{1+z^{-1}}\right)}{A_c\left(\frac{2}{\tilde{\omega}_c T}\frac{1-z^{-1}}{1+z^{-1}}\right)} = \frac{B_d(z)}{A_d(z)}. \tag{35}$$

Example 1: Second-Order Discrete-Time Butterworth Lowpass Filter

1. Filter specifications: $n = 2$ and $\Omega_c = 2\pi \times 0.1$ rads/sample.
2. Normalized second-order Butterworth filter: $H_c(s) = \dfrac{1}{s^2 + \sqrt{2}s + 1}$.
3. Sample rate: $1/T = 10^4$ samples/s $\rightarrow \omega_c = 2\pi \times 10^3$ rads/s.
4. Pre-warped frequency $\tilde{\omega}_c = 2 \times 10^4 \tan\left(\dfrac{2\pi \times 0.1}{2}\right) = 2\pi \times 1034$ rads/s.
5. Continuous-time system with corner frequency $\tilde{\omega}_c$:
$$H_c\left(\frac{s}{\tilde{\omega}_c}\right) = \frac{1}{\left(\frac{s}{\tilde{\omega}_c}\right)^2 + \sqrt{2}\left(\frac{s}{\tilde{\omega}_c}\right) + 1}.$$
6. Apply bilinear transform.
$$H_d(z) = \frac{1}{\left(\frac{2}{2\pi \times 0.1034}\frac{1-z^{-1}}{1+z^{-1}}\right)^2 + \left(\frac{2}{2\pi \times 0.1034}\frac{1-z^{-1}}{1+z^{-1}}\right) + 1}$$
$$= \frac{0.0675 + 0.1349\,z^{-1} + 0.0675\,z^{-2}}{1 - 1.1430\,z^{-1} + 0.4128\,z^{-2}}.$$

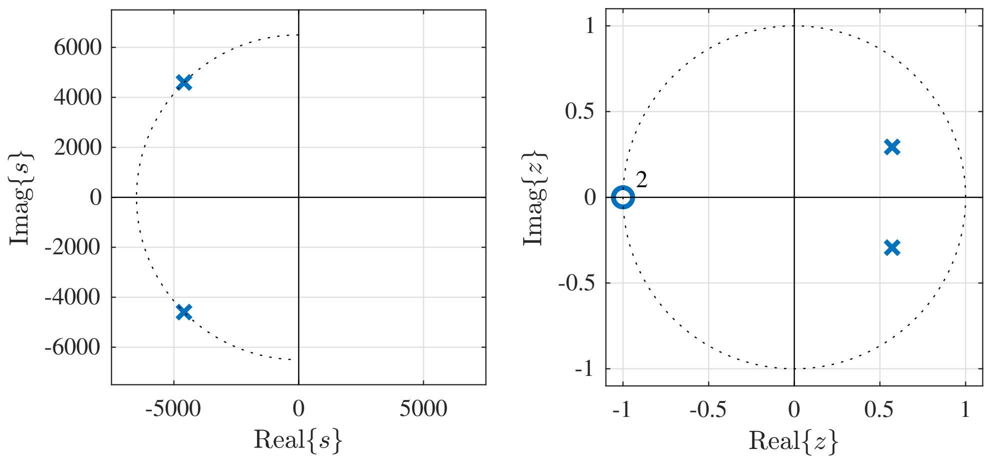


Figure 16: The mapping of the poles of $H_c(s/\tilde{\omega}_c)$ to the poles and zeros of $H_d(z)$ for Example 1.

The mapping of the poles of $H_c(s/\tilde{\omega}_c)$ to the poles and zeros of $H_d(z)$ is illustrated in Figure 16. Comments:

1. As expected, $H_c(s/\tilde{\omega}_c)$ has two poles in the left-half s-plane at complex-conjugate positions on a circle of radius $\tilde{\omega}_c$. $H_c(s/\tilde{\omega}_c)$ has no zeros at finite values of s. That this is true is clear from the mathematical expression for $H_c(s/\tilde{\omega}_c)$: the numerator is a constant and the denominator is a second-degree polynomial in s. That is, the continuous-time Butterworth filter is *an all-pole filter*. This is true of all the standard continuous-time lowpass filter designs: all of them produce all-pole filters.

2. $H_d(z)$ has two poles at complex-conjugate positions inside the unit circle. $H_d(z)$ also has two zeros (repeated zeros at $z = -1$). The zeros were introduced by the bilinear transform. Note that the mathematical expression for $H_d(z)$ is the ratio of two second-degree polynomials in z^{-1}. This observation generalizes: when the applying the bilinear transform to an all-pole s-domain transfer function, the result is a z-domain transfer function with the same number of poles and zeros. The bilinear transform transforms an n-th order all-pole continuous-time system to a discrete-time system with n zeros and n poles.

The magnitude of the frequency response $H_d(\Omega) = H_d(2\pi F)$ is computed using

```
Bd = [0.0675 0.1349 0.0675];
Ad = [1 -1.1430 0.4128];
F = 0:0.001:0.5;
E = exp(1i*2*pi*F);
H = polyval(Bd,E) ./ polyval(Ad,E);

plot(F,20*log10(abs(H)));
grid on;
xlabel('frequency (cycles/sample)'); ylabel('magnitude (dB)');
xlim([0 0.5]); ylim([-40 5]);
```

The frequency response is plotted in Figure 17. Observe that at low frequencies, the pass band has unity gain and drops to -3 dB at $F = 0.1$ cycles/sample, as required by the design specifications. Pre-warping the continuous-time corner frequency was needed to place the corner frequency of the frequency response at 0.1 cycles/sample.

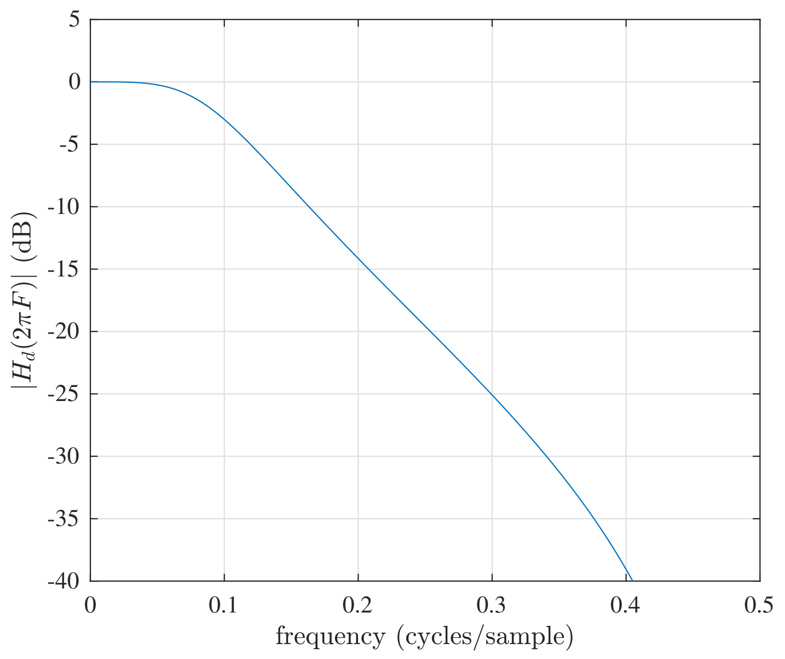


Figure 17: The frequency response of the discrete-time second-order Butterworth filter of Example 1.

Filtering a discrete-time input sequence $x[n]$ to produced an output $y[n]$ is accomplished using the recursion

$$y[n] = 1.1430\,y[n-1] - 0.4128\,y[n-2] + 0.0675\,x[n] + 0.1349\,x[n-1] + 0.0675\,x[n-2]. \quad (36)$$

$\square$

MATLAB provides functions to automate the filter design process and the recursion that defines the input/output relationship. The code segment

```
n = 2;
Fc = 0.1;
[Bd,Ad] = butter(n,2*Fc);
```

produces two vectors `Bd` and `Ad` whose elements are the coefficients of the polynomials $B_d(z)$ and $A_d(z)$, respectively, in (35). The second argument in `butter` in the code segment requires an explanation. The frequencies supplied to all of MATLAB's discrete-time filter design functions are normalized versions of radian frequencies: π rads/sample is 1. In the example above,

$$\Omega_c = 2\pi F_c \rightarrow \text{normalized frequency} = \frac{\Omega_c}{\pi} = \frac{2\pi F_c}{\pi} = 2F_c. \quad (37)$$

The recursion (36) is performed by the MATLAB command

```
y = filter(Bd,Ad,x);
```

where `x` is the input sequence (a vector) and `y` is the output sequence (a vector). The MATLAB command `filter` is useful for completing this laboratory assignment. Unfortunately, the ESP32 microcontroller

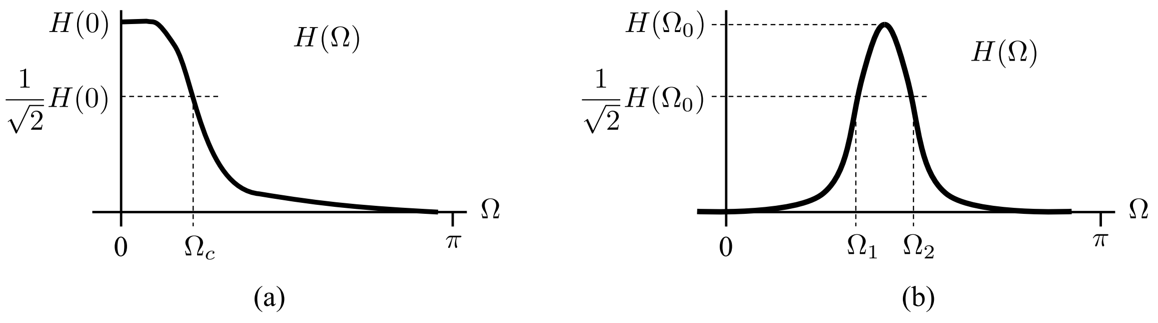


Figure 18: Frequency response specifications for: (a) a lowpass filter; (b) a bandpass filter.

does not have a command like `filter`. You will have to write your own code to perform the filtering defined by the recursion.

So far, the development has focused on lowpass filters. Because bandpass filters are needed for the laser tag system, the design of bandpass filters is of interest. Bandpass filter specifications are similar to, but different from, lowpass filter specifications. The comparison is illustrated by the frequency responses shown in Figure 18. The frequency response of a lowpass filter is specified by its corner frequency Ω_c as shown in Figure 18 (a). The corner frequency is the frequency for which the magnitude of the frequency response is $1/\sqrt{2}$ its magnitude at $\Omega = 0$. In contrast, the frequency response of a bandpass filter is specified by three frequencies: Ω_0 (the center frequency) and two two other frequencies Ω_1 and Ω_2 illustrated in Figure 18 (b). Ω_1 and Ω_2 play a role similar to the corner frequency for a lowpass filter. Because of the shape of the bandpass filter frequency response, there are two frequencies for which the magnitude of the frequency response is $1/\sqrt{2}$ its peak magnitude. These two frequencies are Ω_1 and Ω_2. The peak magnitude is usually assumed to be at $\Omega = \Omega_0$. as shown. The filter design algorithms summarized here require specification of Ω_1 and Ω_2.

The design of bandpass filters follows the same outline as the design of lowpass filters. First, a normalized lowpass filter of the desired order is designed. Second, the continuous-time lowpass filter is transformed to a continuous-time bandpass filter based on pre-warped frequencies. The bilinear transform is applied to the continuous-time bandpass filter to produce the desired discrete-time bandpass filter.

The new step is the transformation of the transfer function for continuous-time lowpass filter to the transfer function for a continuous-time bandpass filter. The lowpass to highpass transformation is accomplished by the substitution

$$s \rightarrow Q\left(\frac{s}{\omega_0} + \frac{\omega_0}{s}\right) \tag{38}$$

in the lowpass filter transfer function. Equation (38) means every occurrence of s in the lowpass filter transfer function is replaced by the expression on the right-hand side of (38). For a continuous time bandpass filter defined by ω_1 and ω_2 rads/s, the constants in (38) are defined as follows. ω_0 is the geometric mean of ω_1 and ω_2:

$$\omega_0 = \sqrt{\omega_1\omega_2}. \tag{39}$$

The Q-factor is the ratio of center frequency to bandwidth. Using $\omega_2 - \omega_1$ as the measure of bandwidth, the Q-factor is

$$Q = \frac{\omega_0}{\omega_2 - \omega_1}. \tag{40}$$

For those who like to design continuous-time filters, the tranformation (38) has an interesting interpretation. The starting point of the interpretation is to recall the s-domain equivalent impedances of series

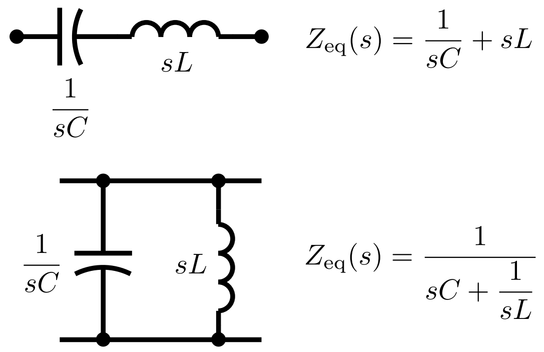

$\frac{1}{sC}$
sL
$Z_{\mathrm{eq}}(s) = \frac{1}{sC} + sL$
$\frac{1}{sC}$
sL
$Z_{\mathrm{eq}}(s) = \frac{1}{sC + \frac{1}{sL}}$

Figure 19: The equivalent impedance of a series combination of capacitor and inductor (top) and the parallel combination of a capacitor and inductor (bottom).

and parallel combinations of a capacitor and an inductor shown in Figure 19. The impedance of a single capacitor undergoes the following transformation

$$\frac{1}{sC} \to \frac{1}{Q\left(\frac{s}{\omega_0} + \frac{\omega_0}{s}\right)C} = \frac{1}{s\frac{QC}{\omega_0} + \frac{1}{s\frac{1}{Q\omega_0 C}}} = \frac{1}{sC' + \frac{1}{sL'}}. \tag{41}$$

The result is of the form of the equivalent impedance of a capacitor C' and inductor L' in parallel. The capacitance and inductance are

$$C' = \frac{QC}{\omega_0} \qquad L' = \frac{1}{Q\omega_0 C}. \tag{42}$$

Should the original lowpass filter have an inductor (most do not), the transformation (38) of the inductor impedance is

$$sL \to Q\left(\frac{s}{\omega_0} + \frac{\omega_0}{s}\right)L = s\frac{QL}{\omega_0} + \frac{1}{s\frac{1}{Q\omega_0 L}} = sL'' + \frac{1}{sC''}. \tag{43}$$

The result is of the form of the equivalent impedance of an inductor L'' and a capacitor C'' in series, where

$$L'' = \frac{QL}{\omega_0} \qquad C'' = \frac{1}{Q\omega_0 L}. \tag{44}$$

The relationships (41) – (44) define the continuous-time circuit transformation:

> Starting with the circuit for a normalized lowpass filter, replace each capacitor with capacitance C with a parallel capacitor-inductor pair with capacitance and inductance given by (42). Replace each inductor with inductance L (if there are any) with a series inductor-capacitor pair with inductance and capacitance given by (44).

The discrete-time IIR bandpass filter design method based on transforming a continuous-time filter goes as follows:

1. Specifications: define the order n and corner frequencies Ω_1 and Ω_2 rads/sample.
2. Design the normalized continuous-time lowpass filter. The transfer function is of the form

$$H_c(s) = \frac{B_c(s)}{A_c(s)}. \tag{45}$$

3. Given a sample rate $1/T$ samples/s, compute $\omega_1 = \Omega_1/T$ and $\omega_2 = \Omega_2/T$ rads/s.

4. Pre-warp ω_1 and ω_2 to produce $\tilde{\omega}_1$ and $\tilde{\omega}_2$ given by

$$\begin{aligned}\tilde{\omega}_1 &= \frac{2}{T}\tan\left(\frac{\omega_1 T}{2}\right) \\ \tilde{\omega}_2 &= \frac{2}{T}\tan\left(\frac{\omega_2 T}{2}\right).\end{aligned} \tag{46}$$

5. Perform the lowpass-to-bandpass transformation using the pre-warped frequencies:

$$\tilde{\omega}_0 = \sqrt{\tilde{\omega}_1\tilde{\omega}_2} \tag{47}$$

$$Q = \frac{\tilde{\omega}_0}{\tilde{\omega}_2 - \tilde{\omega}_1} \tag{48}$$

$$H_c\left(Q\left(\frac{s}{\tilde{\omega}_0} + \frac{\tilde{\omega}_0}{s}\right)\right) = \frac{B_c\left(Q\left(\frac{s}{\tilde{\omega}_0} + \frac{\tilde{\omega}_0}{s}\right)\right)}{A_c\left(Q\left(\frac{s}{\tilde{\omega}_0} + \frac{\tilde{\omega}_0}{s}\right)\right)}. \tag{49}$$

6. Apply the bilinear transform to compute the continuous-time system with transfer function $H_c(\cdot)$ to a discrete-time system with transfer function $H_d(z)$:

$$H_d(z) = H_c\left(Q\left(\frac{s}{\tilde{\omega}_0} + \frac{\tilde{\omega}_0}{s}\right)\right)\Bigg|_{s=\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}} = \frac{B_d(z)}{A_d(z)}. \tag{50}$$

An important detail that is easy to overlook is the that the lowpass-to-bandpass transformation (38) *doubles* the order of the filter.

Example 2: Second Order Discrete-Time Butterworth Bandpass Filter

1. Filter specifications: $n = 2$, $\Omega_1 = 2\pi \times 0.15$ rads/sample, $\Omega_2 = 2\pi \times 0.25$ rads/sample.

2. Normalized second-order Butterworth filter: $H_c(s) = \dfrac{1}{s^2 + \sqrt{2}s + 1}$.

3. Sample rate: $1/T = 10^4$ samples/s $\rightarrow \omega_1 = 2\pi \times 1500$ rads/s and $\omega_2 = 2\pi \times 2500$ rads/s.

4. Pre-warped frequencies:

$$\begin{aligned}\tilde{\omega}_1 &= 2 \times 10^4 \tan\left(\frac{2\pi \times 0.15}{2}\right) = 2\pi \times 1622 \text{ rads/s} \\ \tilde{\omega}_2 &= 2 \times 10^4 \tan\left(\frac{2\pi \times 0.25}{2}\right) = 2\pi \times 3183 \text{ rads/s}\end{aligned}$$

5. Lowpass-to-bandpass transformation:

$$\tilde{\omega}_0 = 2\pi \times \sqrt{1622 \times 3183} = 2\pi \times 2272$$

$$Q = \frac{2\pi \times 2272}{2\pi(3183 - 1622)} = 1.455$$

$$H_c\left(Q\left(\frac{s}{\tilde{\omega}_0} + \frac{\tilde{\omega}_0}{s}\right)\right) = \frac{9.6226 \times 10^7\, s^2}{s^4 + 1.3873 \times 10^4\, s^3 + 5.0385 \times 10^8\, s^2 + 2.8274 \times 10^{12}\, s + 4.1539 \times 10^{16}}.$$

6. Apply the bilinear transform

$$H_d(z) = \frac{0.0675 - 0.1349\, z^{-2} + 0.0675\, z^{-4}}{1 - 1.0212\, z^{-1} + 1.4128\, z^{-2} - 0.6396\, z^{-3} + 0.4128\, z^{-4}}.$$

The result is a rational expression of the form

$$H_d(z) = \frac{B_d(z)}{A_d(z)} \tag{51}$$

where

$$\begin{aligned} B_d(z) &= 0.0675 - 0.1349\, z^{-2} + 0.0675\, z^{-4} \\ A_d(z) &= 1 - 1.0212\, z^{-1} + 1.4128\, z^{-2} - 0.6396\, z^{-3} + 0.4128\, z^{-4}. \end{aligned} \tag{52}$$

The pole-zero plot of the transfer function in step 2 is shown in Figure 1 of Lab 4 and comprises two poles in the left half s-plane on a circle of radius 1. The lowpass-to-bandpass transformation in step 5 produces a transfer function with the pole-zero plot illustrated by the left-hand plot in Figure 20. Observe that the lowpass-to-bandpass transformation doubled the number of poles and introduced two zeros. The four poles are arranged in two complex-conjugate pairs. Complex-valued poles must occur in complex-conjugate pairs because the polynomial coefficients of the transfer function in step 5 are all real valued.

The bilinear transform maps the four poles in the left-half s-plane to four poles inside the unit circle in the z-plane. The z-plane poles are illustrated in right-hand plot in Figure 20. Note that the bilinear transform also introduces two additional zeros for a total of four zeros (two at $z = -1$ and two at $z = 1$). Because the z-plane poles are inside the unit circle, the causal inverse z-transform is stable.

The magnitude of the frequency response $H_d(\Omega) = H_d(2\pi F)$ is plotted in Figure 21. Observe that the frequency response magnitude in the pass band—the range of frequencies around $F = 0.2$ cycles/sample—is unity. At the two corner frequencies $F_1 = 0.15$ cycles/sample and $F_2 = 0.25$ cycles/sample, the frequency response magnitude is -3 dB. The pre-warping performed in step 4 was needed to place the corner frequencies at the desired frequencies.

Filtering a discrete-time input sequence $x[n]$ to produced an output $y[n]$ is accomplished using the recursion

$$\begin{aligned} y[n] = 1.0212\, y[n-1] - 1.4128\, y[n-2] + 0.6396\, y[n-3] - 0.4128\, y[n-4] \\ + 0.00675\, x[n] - 0.1349\, x[n-2] + 0.0675\, x[n-4]. \end{aligned} \tag{53}$$

□

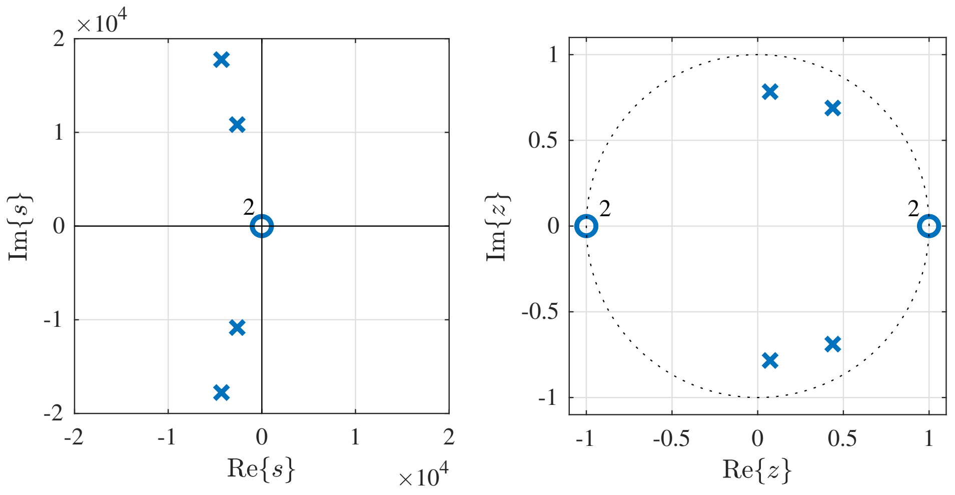


Figure 20: The mapping of the poles of $H_c(s/\tilde{\omega}_c)$ to the poles and zeros of $H_d(z)$ for Example 1.

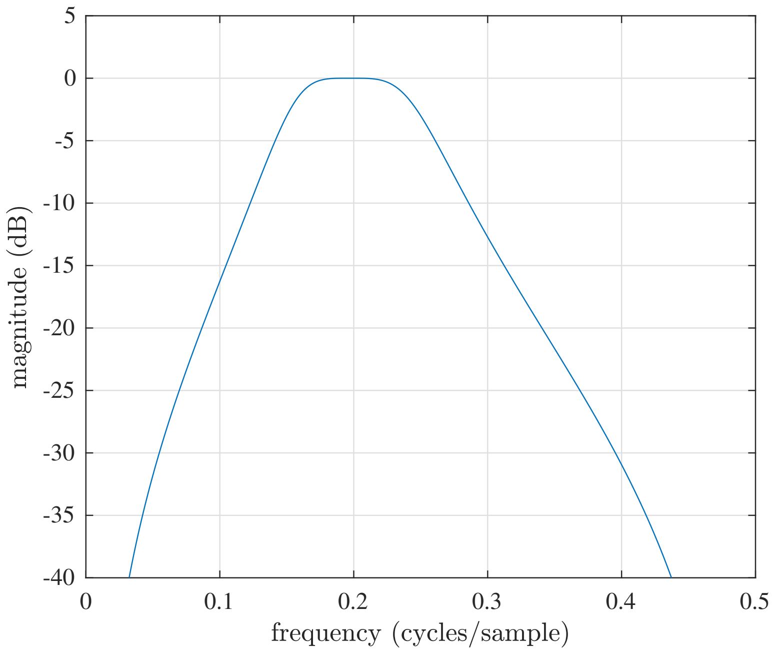


Figure 21: The frequency response of the discrete-time second-order Butterworth filter of Example 1.

The MATLAB `butter` function automates these steps. Example 2 may be completed as follows:

```
n = 2;
F1 = 0.15;
F2 = 0.25;
[B,A] = butter(n,[2*F1 2*F2]);
```

Comments:

1. The MATLAB command `butter` knows a bandpass filter design is intended because the second argument is a 1×2 vector.
2. The frequencies supplied to all of MATLAB's discrete-time filter design functions are normalized versions of radian frequencies: π rads/sample is 1. In the example code above

$$\Omega_1 = 2\pi F_1 \rightarrow \text{normalized frequency} = \frac{\Omega_1}{\pi} = \frac{2\pi F_1}{\pi} = 2F_1 \tag{54}$$

$$\Omega_2 = 2\pi F_2 \rightarrow \text{normalized frequency} = \frac{\Omega_2}{\pi} = \frac{2\pi F_2}{\pi} = 2F_2. \tag{55}$$

3. The function `butter` returns two vectors `B` and `A`. Both of these are 1×4 vectors and contain the polynomial coefficients for $B_d(z)$ and $A_d(z)$, respectively, in (52).

Finally, filtering is performed using the recursion (53). MATLAB has an efficient function to compute perform the recursion:

```
y = filter(B,A,x);
```

where `x` is the input sequence (a vector) and `y` is an output sequence (a vector). The MATLAB command `filter` is useful for for completing this laboratory assignment. Unfortunately, the ESP32 microcontroller does not have a command like `filter`. You will have to write your own code to perform the filtering defined by the recursion.