L314
MPHIL IN ADVANCED COMPUTER SCIENCE
COMPUTER SCIENCE TRIPOS Part III

Monday 4 November 2024 12:00 to Monday 11 November 2024 12:00

Module L314 – Digital Signal Processing – Assignment 2

This assignment involves programming. The recommended programming language is Julia *and library functions referred to in the problems may be found in the Julia packages* `DSP.jl`, `WAV.jl`, `FFTW.jl`, `Plots.jl`, `Colors.jl`, `ImageIO.jl`, `FileIO.jl`, *and* `ImageShow.jl`. *[Implementations in other suitable languages, using equivalent library functions, such as* MATLAB*'s* Signal Processing Toolbox*, or the Python packages* `matplotlib` *and* `scipy.signal`*, are also acceptable.]*

Prepare the solutions and answers to all parts as a single PDF file and include all source code written, along with any required outputs produced by the programs. A `Pluto.jl` *notebook provides a convenient way to combine answer text, Julia source code and outputs into a single PDF. [Alternatives include a* Jupyter *notebook for either a Julia or Python solution, or* MATLAB*'s* `publish` *function or* Live Editor*.]*

Submit your work via

`https://www.vle.cam.ac.uk/course/view.php?id=254472`

no later than 12:00 on Monday 11 November 2024.

Students will be required to sign an undertaking that work submitted will be entirely their own; no collaboration is permitted.

(*a*) Analog "touch tone" push-button telephones use a system called "dual-tone multi-frequency signaling (DTMF)" to communicate to the telephone switch which button is being pressed. Each button produces a combination of two sine tones:

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

You receive a digital telephone signal with a sampling frequency of 8 kHz. You cut a 256-sample window out of this sequence, multiply it with a windowing function and apply a 256-point DFT. What are the indices where the resulting vector $(X_0, X_1, \ldots, X_{255})$ will show the highest amplitude if button $\boxed{9}$ was pushed at the time of the recording? [*Note:* No program expected here.]

(*b*) The audio file `touchtone.wav` on the course-materials web page contains the recording of a DTMF-encoded sequence of buttons pressed on a telephone. Use the short-term Fourier transform, to produce a spectrogram image (similar to slide 90, but without using any existing `spectrogram` library function).

- Determine the sampling frequency used in `touchtone.wav` (Julia: `wavread`, MATLAB: `audioread`).
- Chose (via experimentation) a DFT window size that provides a good visual tradeoff for the resulting time and frequency resolution.
- Load the audio file and split it into a series of blocks where each block overlaps 50% with the previous block (i.e., each sample appears in two blocks)
- Chose and apply a window function
- Calculate the DFT of each block (Julia/MATLAB function `fft`)
- Convert the results into a raster image in which the amplitude (magnitude) spectrum of each DFT appears as a vertical column, discarding redundant frequencies, such that the vertical axis shows frequency and the horizontal axis the time index at which the respective DFT block starts in the recording.
- Display the spectrogram as a raster image (e.g., using Julia function `heatmap`), appropriately cropped, and label the horizontal axis in seconds and the vertical axis in Hz, with ticks at the eight DTMF frequencies.
- Experiment with different colourmaps and experiment with showing linear versus logarithmic displays of the magnitude.
- Read from this spectrogram the touch-tone number dialed.

(*c*) Construct seven band-pass filters, with centre frequencies chosen according to the above table (skipping 1633 Hz, which is not used here). To choose the bandwidths of these filters, consult ITU-T Recommendations Q.23 and Q.24. Then pass the same `touchtone.wav` signal through each of these seven filters. Plot the result for each of the seven filters such that you can again read off the sequence of digits pressed. Experiment with different filter types and orders until the seven tones are clearly detected and separated across the seven receiver channels. Summarise the reasons for your choices.

(*d*) The audio files `fsk1.wav` and `fsk2.wav` on the course-materials web page each contain a frequency-modulated ASCII text string, encoded as follows: transmission rate 100 bits/s, each character is transmitted as one start bit (0), followed by eight data bits (least-significant bit first), followed by two stop bits (1). A 0 bit is modulated as a 1180 Hz tone, a 1 bit as a 980 Hz tone. Any idle period after a character is filled with the same tone as is used for the stop bits.

To read these messages, you could adapt the techniques from parts (*b*) or (*c*), but let's try something else. Instead of e.g. modulating a low-pass FIR filter to convert it into a band-pass filter, we can also first shift the frequency spectrum of the input signal such that the frequency of interest is at 0 Hz, and then apply a low-pass filter. This is a technique commonly used in AM radio receivers. Implement two such amplitude demodulators, one for each of the audio frequencies used.

- Load the waveform from the WAV file and make two copies.
- Multiply both waveform copies with a complex phasor rotating at $f_0 = 1180$ Hz and $f_1 = 980$ Hz, respectively. This multiplication will shift the Fourier spectrum of the waveform by $-f_i$, such that the respective target frequency of f_i ends up at 0 Hz, and its negative pair $-f_i$ ends up at $-2f_i$.
- Apply a low-pass filter to each product from the previous step, to eliminate any other frequencies not of interest, including the shifted negative target frequency.
- Calculate the absolute values of the (still complex valued) output waveform of each low-pass filter and plot the resulting real-valued waveforms on top of each other, each labeled with the corresponding bit value. [*Note:* you should see a clear increase in signal level when there is a tone present in the respective frequency channel.]
- Experiment with different low-pass filter types, cut-off frequencies and orders until the two bit tones are clearly detected and separated.
- From the amplitude difference between these two frequency channels, locate the leading edge of each start bit, then read which eight data bits follow and look up the corresponding ASCII letter.

The message in "`fsk1.wav`" is "`DSP`". What is it in `fsk2.wav`?

(*e*) Write a program to deconvolve the blurred stars from slide 31.

The files `stars-blurred.png` with the blurred-stars image and `stars-psf.png` with the impulse response (point-spread function) are available on the course-material web page.

In Julia you may find the functions `load`, `Float64`, `Gray`, `fft`, `ifft`, `circshift`, `nextpow`, `conv`, `maximum`, `findall`, `axes`, of use. (To display a grayscale image in a Pluto notebook, load packages `ImageShow` and `Colors` and return a matrix of elements of type `Gray`.)

In MATLAB, you may find the functions `imread`, `double`, `imagesc`, `daspect`, `circshift`, `fft2`, `ifft2` of use.

A typical implementation will involve the following steps:

- Load both the blurred grayscale image and the point-spread image and convert both matrices into ones that use a floating-point number type.
- Gradually reduce the pixel values in the blurred image near its edges, to reduce high-frequency components caused by the steps created in the blurred image as the result of cropping. (Try with and without this step.)
- Zero pad both images to the same size, which should in each direction be at least the sum of the respective sizes of the blurred and the psf image. Round these sizes to the next higher power of two.
- Apply the two-dimensional FFT forward transform to both padded images.
- Take care of near-zero values in the FFT of the psf image (e.g., by replacing them with a larger value), before combining both with complex division and applying a two-dimensional inverse FFT to the result.
- Depending on how the psf image was aligned, the resulting deblurred image may have been shifted and may need realignment before cropping to the image's original size.
- Display both the original and the deblurred image.

Experiment with different ways or parameters to control the noise (see slide 89) caused by division of near-zero values and distortions emanating from cropped stars near the edges (edge tapering).

[*Note:* Do not use any ready-made functions for image deconvolution, such as `deconvwnr`, `deconvreg`, `deconvlucy`, `edgetaper` from the MATLAB Image Processing Toolbox.]

END OF PAPER