Final Project

Due December 12, 2017

1 Overview

In this project you will simulate a digital communication system by sending and receiving an image over a channel with noise, similar to an actual communication system (e.g. cell phones, HDTV). You will implement this digital communication system in Matlab.

You will work on this project in a group of two. Any different group size will need approval from the professor and is granted only under special circumstances (for example, an odd number of students in the class). Each group just needs to do one presentation and turn in one project report. In addition, each student in a group needs to turn in a separate one-page report on their contribution to the project. The grading for each person in a group may differ from the other person if one person is doing much more work than the other.

Your grade will be based primarily on your presentation and your report. We will need to be able to run your codes as is to verify the results in your report. Please do not include codes in your report, but you need to *answer all the questions in detail and include the output image for relevant questions.*

Project deadlines:

- December 1: mid-project report
- December 12: class presentation (12 minutes for presentation + 3 minutes for Q&A)
- December 12: final project submission (by 10pm)

The mid-project report:

You need to hand in a report that includes answers to Questions 1 – 9.

The final project submission includes:

– The complete project report in pdf format.

– Complete Matlab codes (in a zip file) and source images. You do not need to print out the Matlab codes.

– Individual one-page reports of contribution.

All submission is by email to both the professor and TA.

This project is the most complicated system you will implement in MATLAB in this course, start early in case you run into difficulties.

2 Communication System

We will construct a model of a communications system that transmits an image from one place to another, much like digital television (HDTV). It is very common in image processing to break up the image into 8 pixel by 8 pixel blocks and take the discrete cosine transform (DCT) of each block – we will provide more information on the DCT later in the description. The DCT frequency information for each block can then be coded, modulated, and transmitted, as will be explained now.

A typical communications scheme begins at the *transmitter*. Here a discrete-time signal $x[n]$ is *quantized*, meaning for each n, $x[n]$ is rounded to a binary number of finite precision. That is, for any fixed n, $x[n]$ is a real number mathematically, so it has infinite precision. In order to transmit it digitally, we must, for example, round it to the closest integer from 0 to 255 because we can only transmit finite information digitally. After quantization, the signal is often coded. *Coding* is a method of representing the signal using some alphabet of symbols in a way that allows for error detection and correction, compression, and other things. We will not do any coding here. *Modulation* is the process of transforming each sample of the signal (typically coded in real world applications) into an analog (continuous time) waveform that is physically transmittable through the communications channel. This is necessary because essentially all signals in nature are fundamentally continuous (pedantic quantum mechanical arguments aside). The *channel* is the physical medium through which the modulated waveforms travel; for example, the atmosphere, a telephone wire, or a coaxial cable could all carry an electromagnetic wave. As the modulated waveforms travel along the channel, they are distorted according to the frequency response of the channel (we assume that it is an LTI system, in practice this is a good approximation for a variety of mediums). Noise is also added to the waveforms. Once they reach the end of the channel, called the *receiver*, one must try to undo all the distortion caused by the channel. Then the waveforms must be *detected* and *demodulated*, or mapped from the analog domain into decoded symbols; in our case, just the 2-D discrete-time signal we started with.

2.1 System Structure

A block diagram of the communications system is shown in the figure below. Here we give a brief overview of the system structure, followed by more detailed descriptions of what you will need to do in MATLAB.

The system consists of the following blocks:

- **Image pre-processing:** This block includes the transformation and quantization of an image. We break the image into 8 by 8 blocks and perform the DCT on each block. Then we quantize the DCT coefficient values (which are continuous) into 256 levels, using 8-bit unsigned binary numbers.

- **Conversion to a bit stream:** For serial transmission over a channel, we need to convert the transformed and quantized image into a bit stream. To do this, we first group the 8 by 8 discretized DCT blocks into groups of size N (we will pick it later). We then reshape each group into a 1-D bit stream (i.e. a vector) for transmission.

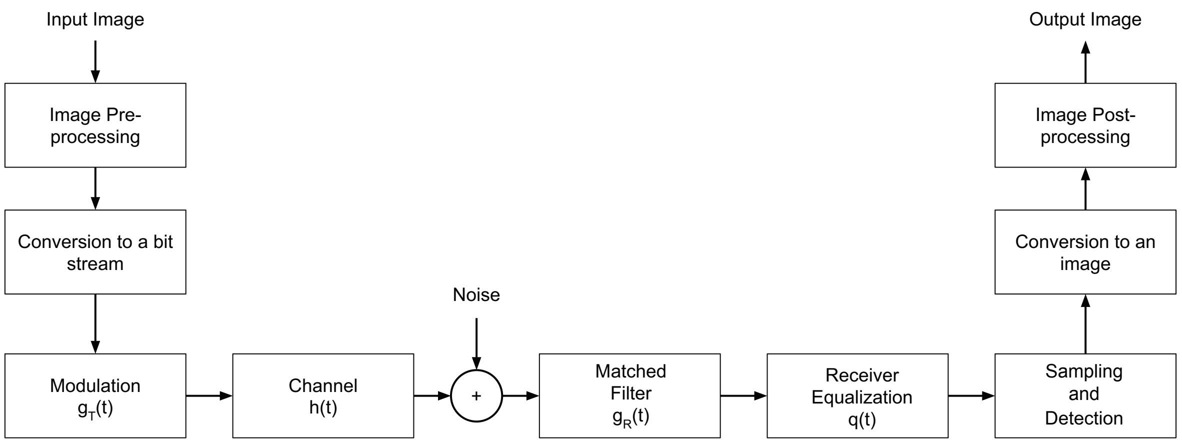

Figure 1: System block diagram.

- **Modulation:** To send digital bits over a physical channel, we need to represent each bit with a waveform, a process called modulation. In this project, we will use binary PAM (Pulse Amplitude Modulation) with pulse shaping filter $g_T(t)$. We will study two different pulse shaping filters in order to compare the effect of pulse shaping on the transmission bandwidth and the error performance. Specifically, we will compare between a half-sine pulse, and a square-root raised cosine (SRRC) pulse with details given later in the implementation section.

- **Channel:** The channel is a filter with a given impulse response that characterizes the transmission medium, such as a copper wire, a cable, or the air. The channel will be modeled as an LTI system with an impulse response $h(t)$. We will provide the specific impulse response in the implementation section.

- **Noise:** At the receiving end, noise (such as circuit thermal noise, interference) is added to the received signal. Therefore the received signal may be written as

$$r(t) = s(t) * h(t) + n(t) \tag{1}$$

  where $*$ is the convolution operator, $s(t)$ is the modulated signal and $n(t)$ is the additive noise. Often the noise is assumed to be white.

- **Matched filter:** The receiver attempts to recover the transmitted signal that has been filtered by the channel and corrupted by additive noise. A block in the receiver that deals with the effect of noise is the received pulse shaping filter, which is matched to the transmit pulse shaping filter as

$$g_R(t) = g_T(T - t). \tag{2}$$

  The matched filter is the best receive filter to deal with the effect of white noise as it results in the highest SNR at the sampling point later. Note that at the output of the matched filter, the overall pulse shape is the combination of the transmit and receive pulse shaping filters. This is the reason for using a SRRC pulse shaping filter at the transmitter, since after combining with the receive matched filter, the overall pulse shape is a raised-cosine pulse.

- **Equalization:** A block in the receiver that compensates for the effects of the channel is the equalizer, which is another linear filter. In this project, we will study two types of equalization filters: the zero-forcing filter (which is the inverse of the channel), and the MMSE filter (which also takes into account noise power). We will discuss each filter in detail in Section 2.8.
- **Sampling and Detection:** The output signal of the equalizer is then sampled at the optimal points, which are at multiple symbol intervals. Here we need to perform some simple synchronization to determine when the first received symbol begins and when is the optimal sampling time.

  After sampling, the received signal is detected, meaning it is examined to determine whether a 0 or 1 was initially sent by the transmitter. Here we will use a simple threshold detection method to recover the transmitted bit stream.
- **Conversion to an image:** This block reshapes the detected bit streams in the original order into the form of an image. This is the reverse step of conversion to a bit stream.
- **Image post-processing:** The demodulated image is converted back into groups of 8 by 8 DCTs, which are then transformed back to the image domain.

In writing your codes, try to write a function for each block to make the code as modular as possible. Modularity not only saves you time by making it easier to test each component separately but also promotes portability. Thus you can reuse each function later as needed.

We now discuss the implementation details for each block, together with possible testing methods and items to be included in the report.

2.2 Image pre-processing

2.2.1 Some image processing background

Suppose we have a *grayscale*, or black and white, image loaded in MATLAB, which we will treat as an m by n matrix I. One of the most straightforward ways to analyze the two dimensional frequency content present in the image would be to take the discrete Fourier Transform (DFT) of each column of I to get a matrix J, then take the DFT of each row of J to get a matrix K. In fact, this is precisely how the *2-D DFT* is defined. However, for this project we will use a slightly different (but closely related) technique called the *Discrete Cosine Transform*, or DCT for short. This is an extremely popular transform; for example it is used to perform data compression in JPEG images and in MPEG coding including MP3 audio compression. There are many other transforms besides the DFT and the DCT, such as STFTs, wavelets and Karhunen-Loeve, and as you might expect, people have written many books and papers on this most useful subject.

The transformed image in the DCT domain is a discrete-frequency signal where each value is a sample in the frequency domain. Since the samples have real values, we will need to quantize these samples into a bit stream to be transmitted. As a background note, even though the original image in the spatial (time) domain is already in a digital format, we actually transmit the frequency response of the image after doing the DCT. This is because by transforming an image into the DCT domain, we can compress the image before transmission (compression is not done in this project). We can also transmit different portions of the transformed image at different rates. For example, the low frequency portion needs stronger protection from error will be transmitted at a lower modulation index (for example, binary PAM), whereas the high frequency portion needs less protection and can be transmitted at a higher modulation index (such as 8-PAM or 4-QAM / QPSK).

2.2.2 Image pre-processing

First find an image to use, we encourage you to **find your own image**. To load the image, use `imread`. You will need to convert the image to the double data type using `im2double`.

From here on we assume the image is m by n pixels (make sure that m and n are both divisible by 8). Perform a DCT on the image in 8 by 8 blocks using the `blkproc` function. To prepare the transformed image for quantization, first scale all the values linearly so that the largest is 1 and the smallest is 0. Set aside the scaling constants you use for later; they will be needed in the post-processing step. Now you can quantize the scaled, transformed image using the quantization code in Homework 7. Next, use `reshape` and `permute` to convert the scaled image to a 3 dimensional array of dimension 8 by 8 by $m * n/64$ such that at each point in the 3rd dimension, the first two are one of the 8 by 8 DCT blocks.

2.3 Conversion to a bit stream

We will transmit DCT blocks in groups of size N. Write your code such that N is a changeable parameter. Given the current group of N blocks, `reshape` it into one long column vector. Convert each element of this vector into a row with that element's 8 bits as the row's 8 entries using `de2bi`.

2.4 Modulation

We will implement the modulation using two different pulse shaping functions and compare their performance in terms of the transmission bandwidth and the error probability. The first pulse shaping function is a half-sine wave where the data bits are multiplied by one half period of a sine wave. The second wave will be a square-root raised-cosine (SRRC).

Specifically, suppose that a bit b has duration T, for the half-sine pulse shaping function, we send the following signal:

$$g_1(t) = \begin{cases} +\sin\left(\frac{\pi}{T}t\right) & \text{if } b = 1 \\ -\sin\left(\frac{\pi}{T}t\right) & \text{if } b = 0 \end{cases} \qquad \text{for } 0 \leq t \leq T. \tag{3}$$

For the square-root raised-cosine pulse shaping function, we send a pulse $g_2(t)$ as

$$g_2(t) = \begin{cases} +A \cdot x(t) & \text{if } b = 1 \\ -A \cdot x(t) & \text{if } b = 0 \end{cases} \quad \text{for } -K \cdot T \leq t \leq K \cdot T. \tag{4}$$

where A is a normalization factor that makes the energy in the SRRC pulse the same as the energy in the half-sine pulse, and K is the truncation length that we will discuss below. The SRRC pulse has the following spectrum:

$$X(f) = \begin{cases} \sqrt{T}, & 0 \leq |f| \leq \frac{1-\alpha}{2T} \\ \sqrt{T} \cos\left\{ \frac{\pi T}{2\alpha} \left[|f| - \frac{1-\alpha}{2T} \right] \right\}, & \frac{1-\alpha}{2T} \leq |f| \leq \frac{1+\alpha}{2T} \\ 0, & |f| > \frac{1+\alpha}{2T} \end{cases} \tag{5}$$

Here α is the roll-off factor. This spectrum is the square root of the raised cosine spectrum as we studied in class. The impulse response of this SRRC pulse is given as

$$x(t) = \frac{\sin\left(\pi \frac{t}{T}(1-\alpha)\right) + 4\alpha \frac{t}{T} \cos\left(\pi \frac{t}{T}(1+\alpha)\right)}{\pi \frac{t}{T}\left(1 - \left(4\alpha \frac{t}{T}\right)^2\right)}. \tag{6}$$

At special points where the above denominator is zero, the values of the SRRC impulse response can be computed to be

$$x(t) = \begin{cases} 1 - \alpha + 4\frac{\alpha}{\pi} & t = 0 \\ \frac{\alpha}{\sqrt{2}} \left[\left(1 + \frac{2}{\pi}\right) \sin\left(\frac{\pi}{4\alpha}\right) + \left(1 - \frac{2}{\pi}\right) \cos\left(\frac{\pi}{4\alpha}\right) \right] & t = \pm \frac{T}{4\alpha} \end{cases} \tag{7}$$

Technically, the SRRC pulse goes on for an infinite time duration. For practical implementation, we need to truncate it to $-KT \leq t \leq KT$ where typically $K = 6$. In this project, you should make K a parameter that you can change its value to be between $2 \leq K \leq 6$. That way you can observe the effect of the truncation length K on system performance.

The fact that the pulse is nonzero outside of its own bit duration implies that there will be overlapping of the modulating waveforms for different bits at the transmitter (even before the signal goes through the channel). Note that the SRRC pulse itself *does not* satisfy the Nyquist's criterion since it does not cross zero at multiple bit intervals nT. However, if we apply a matched filter at the output and sample at the correct times (as we will do in this project), the output of the matched filter will have the raised cosine spectrum which satisfies the Nyquist's criterion, and hence we will recover the signal without inter-symbol interference at the receiver.

For the half-sine pulse shaping function, define the modulating sine wave $\sin \pi t$ for $0 \leq t \leq 1$ (assuming the bit duration is $T = 1$); we suggest using 32 samples.

For the SRRC function, select $\alpha = 0.5$ (make it a parameter so you can change it later). Each bit will span a time duration of $2K * T$. *Any two consecutive bits will have their* *modulated waveform overlapping by* $2K-1$ *bit durations* ($(2K-1)*T$). The transmit signal will then be the sum of all the overlapping SRRC waves. For each bit duration, we suggest you use the same number of samples (32 samples) as you use for the half-sine wave. Thus the total number of samples for each SRRC pulse will be $2K*32$.

The eye diagram is obtained by superimposing the waveforms of subsequent bits on top of each other. The middle part of this eye should now be open. The vertical width of the opening is the voltage margin, and the horizontal width is the timing margin. These play an important role in signal sampling and reconstruction.

Q1) Plot the two pulse shaping functions and the frequency responses of both pulses. Which pulse uses more bandwidth? Discuss the meaning of these results.

Implementation note: When plotting these pulse, make sure that you use the same number of samples *per bit duration*, so that the SRRC pulse should have more samples than the half-sine pulse ($2*K$ times more samples). For the frequency response, you need to show both the magnitude and the phase plots. Make sure to plot the magnitude in dB and not in linear scale.

If you increase the length of the truncated SRRC pulse, what do you expect to see in the frequency response? What happens if you change α? You can experiment with the number of bit truncation K and the roll-off factor in an SRRC pulse here. Discuss what you observe, does it make sense?

Q2) Plot the spectrum of the modulated signal (at the output of the pulse shaping filter) and compare with the pulse spectrum for each pusle.

Q3) For each pulse shape, plot the modulated signal in time for 10 random bits. Comment on what you observe. You should see that the modulated signal using the SRRC pulse has the pulses overlap each other.

Q4) Plot the transmit eye diagram for both pulses. Do you observe the eye opening? Comment on what you see.

Implementation note: When plotting the eye diagram, you should superimpose bit durations and not the whole pulse durations (note the SRRC pulse spans several bit durations and the transmitted SRRC pulses overlapping each other). Both eyes should span only *a single bit duration* in time. Each eye should be approximately symmetric both vertically and horizontally.

2.5 Channel

Since we process the signal in discrete-time, assuming that we sample the channel at the same rate as we sample the received signal, which is at the bit intervals nT, we can use the impulse response $h[n] = h(nT)$ of the equivalent discrete-time system. This is an FIR (finite impulse response) filter with four taps given as

$$h[n] = h(nT) = \delta[n] + \frac{1}{2}\delta[n-1] + \frac{3}{4}\delta[n-2] - \frac{2}{7}\delta[n-3]\ . \tag{8}$$

The three time delayed echoes here might represent attenuated reflections of the signal in the environment (e.g. reflections off buildings, cars, hills, trees in wireless communications). These echoes cause distortion in the received signal since the received waveforms of different bits will overlap in time, leading to ISI.

Represent $h[n]$ in MATLAB as a vector. Note that the four taps will need to be spaced equally so that there is a full bit duration T in between every two consecutive taps. So if you use 32 samples to represent each bit in the modulation part, then there should be 31 zeros in between every two consecutive taps. Take care to make the total number of samples a power of 2, so your code will run faster.

Now send the modulated bit stream through the channel. You may use `conv` to apply the filter given above to each row of the modulated signal matrix from the previous step.

Q5) Plot the frequency response (both phase and magnitude) and the impulse response of the channel. (You may use `freqz` or `fft`.)

Implementation note: Since in Matlab we represent each bit duration by a number of samples (32 samples in this case) and the frequency response of a sampled channel is periodic, you only need to display *one period* of the channel frequency response.

Q6) Plot the eye diagram of the channel output for each pulse shaping function. Comment on your results. The opening of the eye should be smaller due to ISI.

2.6 Noise

Now generate a Gaussian random noise matrix with the command σ`*randn(`$\cdot$`,`$\cdot$`)`, where σ is the square root of the noise power. The arguments of `randn` depends on the size of the input stream in your implementation. Add this Gaussian noise to the output of the channel.

Q7) Plot the two eye diagrams again of the channel output with noise added. Comment on your results. (You can experiment with several values for the noise power σ^2 here.)

2.7 Matched Filter

The matched filter is designed to match to the transmit pulse shaping filter as in (2). For each of the two given pulse shaping functions, implement the impulse response of this matched filter in the same way as you construct the pulse shaping function.

Q8) Plot the impulse response and frequency response of the matched filter for each pulse shaping function.

Q9) Plot the eye diagram for each pulse shape at the output of the matched filter. What is the best sampling point according to your eye diagram? Comment on your results.

2.8 Equalizer

Here you will apply an equalization filter in an attempt to recover the modulated signal as it was before passing through the channel. In this project we will try two different equalizers.

2.8.1 Zero-Forcing (ZF) Equalizer

This filter is just the inverse of the channel response. Suppose that $H(f)$ is the frequency response of the channel $h[n]$, then the frequency response of the ZF filter is

$$Q_{\mathrm{ZF}}(e^{j\omega}) = \frac{1}{H(e^{j\omega})} \tag{9}$$

You may use `filter` to implement this ZF equalizer.

Q10) Describe in words how you implement the ZF equalizer.

Plot the frequency response and impulse response of this filter. You may use `freqz` to plot both the frequency response and (in conjunction with `ifft`) to recover the impulse response.

Discuss the resulting frequency response. Is this zero-forcing filter stable? Can you expect the inverse of the channel to always be stable?

Implementation note: The ZF is the inverse of the channel, so its impulse response should ideally be infinitely long (it is an IIR – infinite impulse response – filter). When plotting the impulse response of this filter, you can truncate it after some time as the amplitude will decay with time. Keep in mind though that the length of this ZF impulse response should be much longer than the channel impulse response. The ZF frequency magnitude response should be the inverse of the channel magnitude response.

Q11) Plot the eye diagrams for each pulse shape at the output of the equalizer. Remember to specify the noise level σ in your plot. Comment on your result.

2.8.2 MMSE Equalizer

If there were no noise and the channel were invertible, the ZF filter would perfectly recover the modulated signal. In practice, however, there is always noise. In addition, we often don't know the channel response precisely since we can only make noisy measurements of it. It can also change quite suddenly over time (imagine a person holding a cell phone while walking or traveling in a car) and hence the channel measurements need to be updated frequently, making it more noise prone. Hence we now study a different filter that takes both the channel and the noise into account.

Using detection and estimation theory, it is possible to design filters that simultaneously invert the channel response and try to undo some of the effects of the noise. One popular filter of this type is called the minimum mean square error filter, or MMSE filter, since it minimizes the mean, or average, of the square of the difference between the transmitted signal and the received signal. The MMSE filter takes the noise into account via the signal-to-noise ratio, or SNR, defined as the ratio of the signal power to the noise power (calculated at the sampling point in the receiver). The signal power is obtained from the pulse shaping functions used in Step 2.4. For the half-sine pulse, its average power is 1/2. For the SRRC pulse, we choose the normalization factor A so that the pulse energy (over its truncated duration) is the same as the half-sine energy, so that the signal power at the sampling point in the receiver is the same for the two pulse shaping functions. (In fact, in the codes for pulse shaping and matched filter that we gave you, the pulses are already normalized in energy so you don't need to worry about further normalization.)

The noise power is σ^2, as used in generating the additive Gaussian noise in Step 2.6. The frequency response of the MMSE equalizer is then given as

$$Q_{\text{MMSE}}(e^{j\omega}) = \frac{H^*(e^{j\omega})}{|H(e^{j\omega})|^2 + 2\sigma^2} \tag{10}$$

where * is complex conjugation. Notice that when the noise is weak, $\sigma^2 \to 0$, the MMSE filter approaches the ZF filter. Thus when the noise power is small, the two filters should perform similarly, but when the noise power is large, the MMSE is expected to outperform the ZF equalizer.

To compute the equalizer frequency response $Q_{\text{MMSE}}(e^{j\omega})$ in Matlab, you need to use the channel impulse response *with the zeros insertion*. That is, the MMSE frequency response can be obtained by first doing `freqz` with the 'whole' option, or doing `fft`, *on the channel impulse response with zero insertion and appropriate length* to obtain $H(e^{j\omega})$, then apply it equation (10).

There are different ways of implementing the MMSE equalization. We suggest you implement it in the frequency domain, but you can try implementing it the time domain as well. Here are several different ways to implement the MMSE equalizer.

- *fft:* To perform the equalization in the frequency domain, you need to convert the received signal to the frequency domain by doing `fft`, multiply it with the equalizer frequency response (see text above), and convert the result back into the time domain through `ifft`. Take care to make sure the frequency domain multiplication is applied to two vectors of the same length.

- *Convolution:* Compute the MMSE frequency response as above, then use `ifft` to convert it back to the time domain to obtain the MMSE impulse response. Convolve that impluse response with the data in time. You will need to adjust the sampling point in the signal detection block to account for the fact that convolution output length is longer than either input vector.

- *Filter:* You can also use `filter` but this is the most complicated option. You will need to work out the coefficients for the numerator and denominator of the `filter` command from the channel impulse response given in (8), using the DTFT equation. You cannot use `fft` and `ifft` to obtain filter coefficients, they will not work correctly because of issues with sampling in both time and frequency domains.

Q12) For the report, describe in words the method you used to implement the MMSE filter (no codes please).

Plot the frequency response and impulse response of this filter. You may use `freqz`. Comment on the results and compare to those of the zero-forcing equalizer.

Q13) Plot the eye diagrams at the output of the MMSE equalizer. Comment on your results.

2.9 Sampling and Detection

In converting from the received waveform to discrete-time, we need to sample the signal. Here we sample at the output of the equalizer at the bit intervals. But before sampling, you need to determine when the first received bit begins in time and when is the best sampling instance. The subsequent sampling times are just multiple of the bit intervals from the first sampling instance.

Using a simple zero threshold, if the sample is positive, then we say we received a 1, otherwise a 0. From this, construct a matrix of bits with the nth row containing eight entries corresponding to the eight bits of the nth received and detected pixel.

2.10 Conversion to an image

Now convert the detected matrix to a vector of integers with `bi2de` and use `reshape` and `permute` to place the received 8 by 8 DCT blocks into the proper place in the full image.

2.11 Image post-processing

Invert the DCT blocks using `blkproc` and rescale the resulting image using the inverse of the linear scaling from the pre-processing stage.

Q14) Display the result. Was your image recovered completely? Comment on what you see.

Include in your report 3 output images: an output image at a noise level that allows perfect reconstruction, and another output image at a noise level that results in some errors in reconstruction, and the third one that has a lot of errors in reconstruction. Remember to specify the noise power for each image. Perform this experiment for both input pulse shapes.

3 Project Implementation and Analysis

3.1 Implementation hints

For the project implementation, you should test each functional block separately before connecting all of them into a system. For example, you may want to send just 10 random bits through the channel first, without noise, then apply the ZF equalizer to make sure that you recover all the bits completely. Then add noise and test the MMSE equalizer. After that, connect the image blocks and test with an actual image.

Take care to notice when your data is double precision (floating point) and when it is uint8 (unsigned byte).

3.2 Analysis

After you have built the whole communication system in MATLAB and tested that it works, you have in your hand a powerful tool to experiment and analyze the effects of several different factors on system performance.

3.2.1 Effect of noise and equalization

Experiment with different values of the noise power σ^2 until you find the critical one that just barely results in errors in the recovered image. What SNR does this corresponds to? Since we use a matched filter receiver, you can compute the SNR by taking the pulse energy and dividing it to the noise power σ^2. The pulse energy is computed by taking the average of the squares of sample values in each pulse.

Q15) Report the critical SNR threshold values (in dB) you find for both equalizers. Which of the two equalizer types has the lower critical value? Why does this make sense?

Q16) Try 2 different images. For each image, pick a value of σ of your choice and include in your report the recovered images using the two equalizers and the corresponding SNR. Comment on your results.

3.2.2 Effect of the pulse shaping function

We implemented two different pulse shaping functions, the half-sine and the SRRC pulses.

Q17) Do these pulses satisfy the Nyquist's criterion for zero ISI? At which point in the system (after the match filter or after the equalizer) do you expect zero ISI at the sampling point? Provide reasoning. Is your understanding in agreement with your observation in this implementation?

Q18) At the same noise power and provided that the two pulse shaping functions have the same energy, do you expect any difference in error performance? Provide reasons. Do you observe this in your simulation? Include in your report an output image for each pulse shaping function at the same noise level.

Q19) Compare and comment on the effects of these pulse shaping functions on the transmission bandwidth. Discuss the effect on system performance of the pulse length in number of bit durations.

Q20) Based on all the above points, discuss which pulse shape is better. Also discuss any drawback it may have.

3.2.3 Effect of the channel

After completing the whole simulation system, try two other channels with the impulse response given below. These are actual wireless channel models used in designing cellular systems. An outdoor channel is given as

$$h_1[n] = 0.5\delta[n] + \delta[n-1] + 0.63\delta[n-3] + 0.25\delta[n-8] + 0.16\delta[n-12] + 0.1\delta[n-25] \, ,$$

and an indoor channel as

$$h_2[n] = \delta[n] + 0.4365\delta[n-1] + 0.1905\delta[n-2] + 0.0832\delta[n-3] + 0.0158\delta[n-5] + 0.003\delta[n-7] \, .$$

Notice the long delays in these channels, especially in the outdoor one. Once you plot the frequency response of these channels, you should be able to see where the channel attenuates significantly (those are called "deep fades") and the difference between an outdoor and an indoor channel response.

Q21) Did your communication work seamlessly when you plug it into a new channel? Include in your report an output image from each new channel. Comment on your system performance under different channels.

Implementation note: Since the outdoor channel has very long delay, its equalizers should have much longer impulse responses. You can also experiment with the length N of the transmission block (see the *Conversion to a bit stream* section) that goes through the channel and equalizer each time to see if that makes a difference in the output image quality.