

Welcome to The Video and Image Systems Engineering (VISE) Lab

February 27, 2007

1 Introduction

The purpose of this lab is to introduce you to the basics of signal, speech and image processing using digital computers, the Matlab (MATrix LABoratory) software environment and the XV image processing software.

2 Signal Processing

The concept of signals arise in an extremely wide variety of fields, and the ideas and techniques associated with signal processing play an important role in such diverse areas as communications, acoustics, seismology, biomedical engineering, circuit design, aeronautics and chemical process control.

In this part of the lab, you will use the Simulink extension to Matlab to design a simple system to view a sine signal and to synthesize a square wave using sine waves.

Down load [SignalLabUtilities](#)

To get the library of Simulink functions for this section, download the [SignalLabUtilities](#) . Then start Matlab on your workstation by typing the command

```
matlab
```

in a command window. After starting up, you will get a Matlab prompt.

```
>>
```

Once Matlab is started, type

```
SignalLab
```

to bring up the library of Simulink components shown in Fig. 1. This library contains a full library of Simulink blocks, a sine wave generator, a scope, a spectrum analyzer, and a pre-designed system for the square wave synthesis experiment you will be running.



Figure 1: Simulink utilities for the signal processing section.

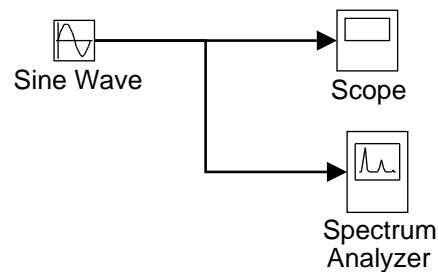


Figure 2: Simulink model for the introductory example.

2.1 Viewing A Sine Wave

In order to familiarize yourself with Simulink, you will first build the system shown in Fig. 2. This system consists of a sine wave generator that feeds a scope and a spectrum analyzer.

1. Open a window for a new system by using the *New* option from the *File* pull-down menu, and select *Model*.
2. Drag the *Sine Wave*, *Scope*, and *Spectrum Analyzer* blocks from the *Lab3* window into the new window you created.
3. Now you need to connect these three blocks. With the left mouse button, click on the output of the *Sine Wave* and drag it to the input of the *Scope*. Now use the right button to click on the line you just created, and drag to the input of the *Spectrum Analyzer* block. Your system should now look like Fig. 2.
4. Double click on the *Scope* block to make the plotting window for the scope appear.
5. Set the simulation parameters by selecting *Parameters* from the *Simulation* pull-down menu. Under the *Solver* tab, set the *Stop time* to 100, and the *Max step size* to 0.02. Then select *Close*. This will allow the Spectrum Analyzer to make a more accurate calculation.

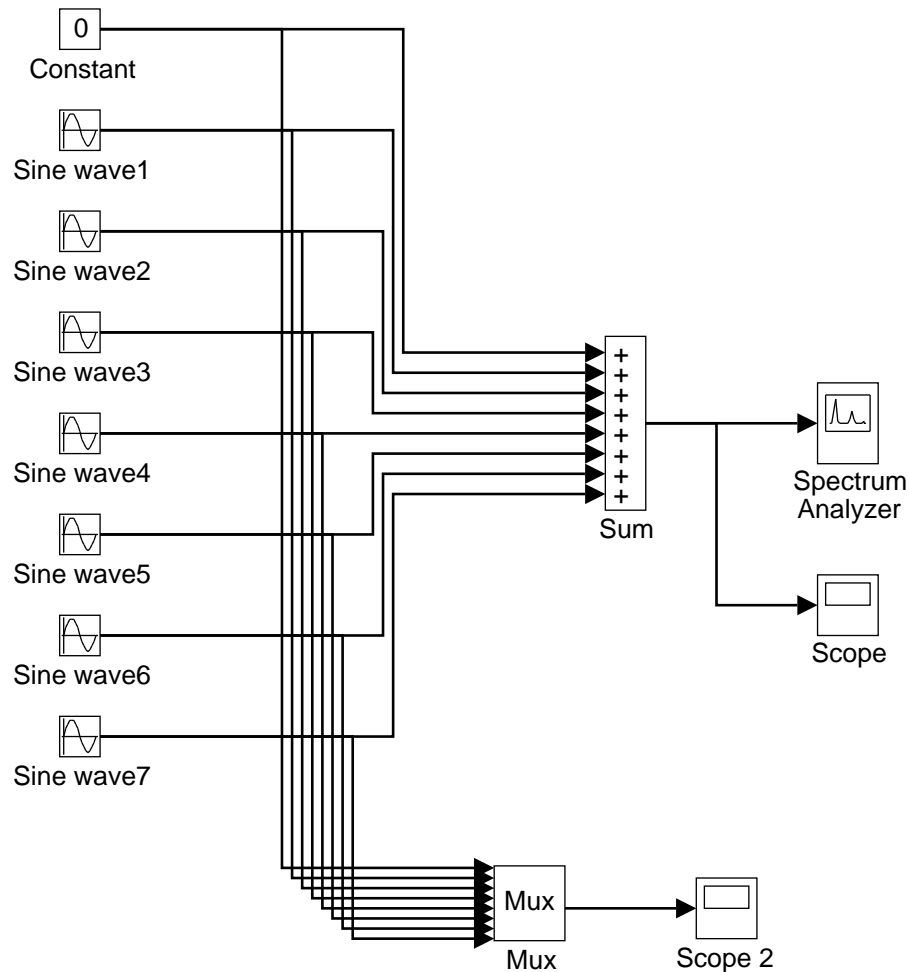


Figure 3: Simulink model for the square wave synthesis experiment.

6. Start the simulation by using the *Start* option from the *Simulation* pull-down menu. A window entitled *Figure 1* will pop up showing the output of the *Spectrum Analyzer*.
7. Change the frequency of the sine wave to 5 rad/sec by double clicking on the *Sine Wave* icon and changing the number in the *Frequency* field. Restart the simulation. Observe the change in the waveform and its spectral density.
8. When you are done, close the system window you created by using the *Close* option from the *File* pull-down menu. You don't need to save this model.

2.2 Synthesizing A Square Wave

It is a well-known fact from signal processing that any periodic signal (signal which repeats itself in time) can be written as a sum of infinitely many sine waves. In this section of the experiment, you will create a periodic square wave by adding up 7 sine waves. Now, double click the icon labeled *Synthesizer* to bring up a model as shown in Fig. 3. This system may

be used to approximately synthesize any periodic signal by adding together 7 sine waves.

In the model given to you, only one of the sine wave generators, *Sine wave 1*, is generating a non-zero output. The amplitudes of the rest of the sine wave generators are set to 0.

1. Run the model by selecting *Start* under the *Simulation* menu. A graph will pop up that shows the 1st sine wave signal and its spectral density which is just a spike. This is the output of the device called *Spectrum Analyzer*. The spectral density plot is the plot of the frequency components in the given signal. The fact that you see only one spike means that there is only one frequency component, only one sine wave, in your given signal. After the simulation runs for a while, the *Spectrum Analyzer* element will update the plot of the spectral density plot and the incoming waveform. You may see the plot of the 1st sine wave by double clicking on the *Scope2* icon.
2. Then, sequentially change the amplitudes of the 2nd, 3rd, 4th, 5th, 6th and 7th sine wave generators from 0 to $4/(3*\pi)$, $4/(5*\pi)$, $4/(7*\pi)$, $4/(9*\pi)$, $4/(11*\pi)$ and $4/(13*\pi)$ respectively by double clicking on the corresponding *Sine Wave* icons and changing the number in the *Amplitude* fields. Make sure that you wait long enough after every change you make for the correct spectral density plot to appear. Note the changes in the plot of the synthesized waveform (entitled *Signal*), its spectral density plot, and the graph window of *Scope 2*. With the addition of each new sine wave, your synthesized waveform should look more and more like a square wave. Also note that each sine wave you add causes a new spike (a new frequency component) to appear in the spectral density plot. You can see the simultaneous plot of all the sine waves in the graph window of *Scope 2*.
3. Now that you have successfully created a periodic square wave, you can shift it up and down. Double click on the *Constant* icon and change the number in the *Constant Value* field from 0 to 1. Observe its effect on the synthesized waveform. Repeat the procedure for *Constant Value* of -1.
4. When you are done, close the *Synthesizer* and the *SignalLab* system windows by using the *Close* option from the *File* pull-down menus and close your figure windows by typing
`close all`

3 Speech Processing

A rapidly developing branch of signal processing is speech processing. Sophisticated signal processing algorithms and hardware are being used for speech recognition and synthesis in a wide range of systems such as security systems, automatic voice response systems and consumer items like audio systems, learning aids, and even toys.

In this section of the experiment, you will filter a contaminated speech signal to reveal a secret message and then, you will change the sampling rate of the speech signal to see how it affects the audio output.

Down load [noisyspeech.mat](#)
Down load [Matlab audio utilities](#)
Down load [FilterSpeech function](#)

1. To run the experiment, first download the contaminated speech signal [noisyspeech.mat](#) , the [Matlab audio utilities](#) , and the [FilterSpeech function](#) .
2. Load the contaminated speech signal into Matlab by typing the command
`load noisyspeech`
in your Matlab command window. This will load the signal *nspeech* and its spectral density *nspeechspec* into your workspace.
3. Play *nspeech* by typing the command
`autoplay(nspeech)`
You will hear a beep. In fact, there is also a speech signal in it that is masked by the beep.
4. The beep that you hear is a sine wave. If you plot the first 50 points of this speech signal by typing the following two lines
`set(figure(1), 'pos', [800 700 400 250])`
`plot(nspeech(1:50))`
you will see that it looks almost like a sine wave.
5. To look at the spectral components of this signal, first bring up another plotting window by typing
`set(figure(2), 'pos', [800 350 400 250])`
Then, plot the spectral density of the signal by typing
`plot(frequency,nspeechspec)`
You will see a big spike around 1.35. This spike corresponds to the beep. In addition, you will see considerably smaller spikes between 0.0 and 0.5, and between 1.5 and 2.0. These correspond to the masked hidden message.
6. You will use the *FilterSpeech* function to filter out this beep and reveal the hidden message. Type
`FilterSpeech`
This command will filter out the beep from the noisy signal *nspeech*, amplify the output, and create a filtered signal called *speech*. It will also create *speechspec*, the spectral density of the filtered signal.
7. Play *speech* by typing the command
`autoplay(speech)`
Listen to the hidden message, note that most of the beep is filtered out.

8. You can verify that the beep is gone by looking at *speechspec*. Bring up another plotting window by typing
`set(figure(3), 'pos', [800 10 400 250])`
Then, plot the spectral density of the filtered signal by typing
`plot(frequency,speechspec)`
Note that the big spike around 1.35 is no longer there. The spikes that you now see are the amplified versions of the smaller spikes you have observed between 0.0 and 0.5, and between 1.5 and 2.0 in the plot of *nspeechspec*, the spectral density of the contaminated speech signal.
9. Matlab is designed to play audio files that are sampled at 8kHz, that is it will play the audio signal correctly if 8000 samples are taken from the analog audio signal per second. If you try to play audio signals sampled at different rates, you get interesting consequences. For example, sample this speech signal at 16kHz by typing the command
`speech16=interp(speech,2);`
Do not forget to put a semicolon at the end of the command. Play this new speech signal by typing the command
`autoplay(speech16);`
and note the difference.
10. Sample the speech signal at 4kHz by typing the command
`speech4=decimate(speech,2);`
Play this new speech signal by typing the command
`autoplay(speech4);`
and note the difference.
11. When you are done, quit Matlab by typing
`quit`

4 Image Processing

Image processing is the manipulation and analysis of images by a computer. Image processing is closely related to how humans perceive the outside world. As a result, it has a large number of existing and potential applications in a wide variety of areas like communications, medicine, defense and robotics. Some everyday appliances that employ image processing are television sets, monitors, CAT and MRI scanners, and digital cameras. In this section, you will experiment with grayscale and colored images. You will apply grayscale or color transformations to them and you will filter them to make them sharper or to find the edges in them.

Down load [imagedata.tar](#)

5 Grayscale Transformations

In this part we will look at some simple grayscale transformations that can have dramatic effects on images. We will use a program called *xv* to take a look at three different transformations.

5.1 Contrast Reversal

1. Type `./begin` followed by the *Return* key in the lower-right text window. This will start two copies of the *xv* program. After a short delay, two windows with an image of the Teton Mountains will appear in the top corners of the screen.
2. In the lower-left corner of the screen is a window containing the *xv color editor*. Changes made in this window will affect the upper-left image.
3. Grayscale transformations are performed by manipulating the *Intensity* graph located at the bottom of the center column of the *xv color editor* window. To reverse the contrast of the image, we need to do the following:
 - (a) Remove the two intermediate handles by twice pressing the third button up from the bottom of the *Intensity* graph (it has a scissors on the button). Note: The “handles” are small squares at a quarter and three quarters of the way up the line. (Handles are also at the ends of the line.)
 - (b) Drag the bottom-left handle to the top-left position and the top-right handle to the bottom-right position. You can see that the straight line in the *Intensity* graph now goes from top-left to bottom-right. The upper left image should look like a photographic negative of the upper right image.
4. After comparing the two images, press the reset button (the second button up from the bottom on the *Intensity* graph).

5.2 Thresholding

In this section we will explore thresholding. Our original image contains up to 256 different levels of gray. In this section we will reduce the image to just two levels of gray (black and white).

1. Press the button second from the top on the *Intensity* graph in order to use straight lines to connect the handles.
2. Press the button third from the top on the *Intensity* graph twice in order to add two handles.
3. Drag the two intermediate handles to manipulate the *Intensity* graph so that it looks like the graph in Figure 1(a). You can see that the image now contains only black and white information.

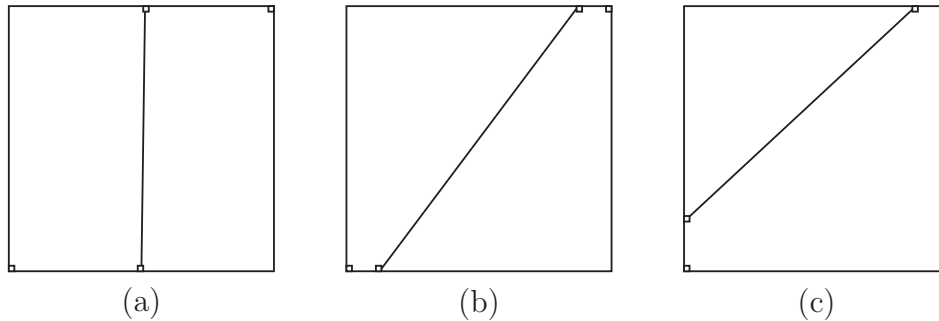


Figure 4: Intensity graphs for (a) thresholding and (b) and (c) contrast enhancement

4. After comparing the two images, leave the images as they are and move on to the next section.

5.3 Contrast Enhancement

1. Return to the *Intensity* graph and make the appropriate adjustments so that it looks like the graph in Figure 1(b).
2. The clouds now appear more vibrant and dramatic.
3. In addition, simple modifications to the *Intensity* graph can reveal details hidden in the original image. Modify the *Intensity* graph to look like the graph in Figure 1(c).
4. It should be evident that the mountains and grass in the foreground are more visible in this image.
5. Experiment with other possible configurations of the *Intensity* graph. Are there other details that are hidden in the original image?

In this part we will take a look at some of the finer details of how color images are displayed on computer monitors and television sets. Computer monitors and televisions form color images by presenting three primary colors (red, green, and blue). Your eyes perceive these three colors which your brain interprets as a single color. We will explore this concept by selectively removing the red, green, and blue color components of an image.

1. Click on the upper-left image and press the space bar. An image of the fountain on Purdue's engineering mall should appear. Do the same for the upper-right image.
2. We will now focus our attention away from the *Intensity* graph and toward the third column of the *xv color editor* window. This column contains three *Intensity* graphs — one for red, one for green, and one for blue. These graphs function in exactly the same way as the *Intensity* graph, except they only effect one of the three colors.
3. Go to the *Intensity* graph for the red component and press the button third from the bottom twice in order to remove the two middle handles.

4. Move the top-right handle to the bottom-right corner. In doing so you have removed the red component from the image. Observe that the image takes on a greenish cast and the pink flowers have turned blue.
5. In the same way, remove the green component from the image. Now the image appears *very* blue. You may also notice that the image is darker.
6. Experiment with different combinations of these operations to see the effects of taking out any one or two color components from the image.
7. We are now finished with these images. In order to remove them from the screen, move the mouse arrow on top of the image and press `q` for *quit*. Do this for both images.

6 Spatial Filtering

In this section we will investigate some of the effects of spatial filtering on images. We will explore three different types of spatial filtering — smoothing, sharpening, and edge detection. Each function is performed in the same manner. The only thing that changes is the weightings that we give to the pixel and its neighbors.

Each operation is performed using Matlab, an environment for processing data on a computer.

6.1 Image Smoothing

1. Type the command `./matlab` into a Unix window. This will bring up a Matlab command window.
2. Then type the following commands into Matlab.

```
kernal1 = [0.06 0.13 0.06 0.13 0.25 0.13 0.06 0.13 0.06];
FilterImg('tree.tif',kernal1);
```

3. Both the original and filtered image will appear. The filtered image should appear smoother or more blurry.
4. The first three numbers in the above command correspond to the top row of the filter kernel. The next three correspond to the middle row, and the last three correspond to the bottom row. Hence the filter used above is:

$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$
$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{8}$
$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$

Note: $\frac{1}{16} \approx 0.06$, $\frac{1}{8} \approx 0.13$, and $\frac{1}{4} \approx 0.25$.

5. When you are finished examining the images, remove them from the screen by typing `close all` into the Matlab command window.

6.2 Image Sharpening

1. In this part we perform an operation very similar to the previous one, but the result is the opposite. Instead of blurring the image we will sharpen it. The only difference here is that we use a different set of weightings for the filter kernel.
2. Here we would like to filter the image with the following kernel:

$\frac{-1}{9}$	$\frac{-1}{9}$	$\frac{-1}{9}$
$\frac{-1}{9}$	$\frac{17}{9}$	$\frac{-1}{9}$
$\frac{-1}{9}$	$\frac{-1}{9}$	$\frac{-1}{9}$

We do this by typing the following commands into the Matlab command window:

```
kernal2 = [-0.11 -0.11 -0.11 -0.11 1.89 -0.11 -0.11 -0.11 -0.11];
FilterImg('tree.tif',kernal2);
```

3. The original and filtered images will appear on the screen. Observe that the filtered image looks sharper than the original. You may enlarge the windows to get a better look at the images.
4. When you are finished examining the images, remove them from the screen by typing `close all` into the Matlab command window.

6.3 Edge detection

1. We can use the same technique (with yet other weightings) to create an image that indicates where edges are present in the original image.
2. An appropriate set of filter weightings are:

$\frac{-10}{9}$	$\frac{-10}{9}$	$\frac{-10}{9}$
$\frac{-10}{9}$	$\frac{80}{9}$	$\frac{-10}{9}$
$\frac{-10}{9}$	$\frac{-10}{9}$	$\frac{-10}{9}$

3. To do this type the following commands into the Matlab command window.

```
kernal3= 10*[-0.11 -0.11 -0.11 -0.11 0.89 -0.11 -0.11 -0.11 -0.11];
FilterImg('edge_in.tif',kernal3);
```

4. The filtered image is bright where an edge is present in the original image and dark where the original is smooth.