LABORATORY 3: The Signals and the Noise

Getting Started!

Lab Submission Instructions and Partners:

1) Keep your partners! If a student dropped the course, let us know!
2) Decide if you'd like to do an Alpha Experiment or Omega Exploration.
3) The format of this lab is different from Lab 01 and Lab 02. You will go through each lab section and **answer the questions posed in the lab in a Gradescope assignment by entering measurements, observations and screenshots of your model and results.** There are no "Proof of Concept" submissions for this lab.

Purpose: ECSE is more than just circuits – broadly speaking, it is the study of using electrical energy to solve a problem. Electrical energy takes the form of *electrical signals*, which can be used to convey information or deliver power. The objective of this experiment is to introduce you to the idea representing data as an electrical signal, explore how information can be extracted from data and how that information can be used to make decisions, and learn the basics of how data can be transmitted wirelessly as part of a communication system. As preparation for this lab, you will complete a series of Matlab tutorials in Proof of Skills, which will introduce you to the basics of signal processing, image processing, and machine learning. In Part A, you will use image processing methods to extract features, such as color, from a set of images of traffic signs. In Part B, you will use those extracted features and a simple machine learning model to try to classify the traffic signs by their type (stop, crosswalk, etc.). If you choose to do an Alpha Experiment, you will learn the basics of how data is transmitted as an analog signal by simulating an amplitude modulation (AM) communication system in Simulink. If you choose to pursue an Omega Exploration, you will apply digital filters to audio signals to extract the specific information or create an audio synthesizer in Simulink by modifying common waveforms, such as sine, square, and sawtooth waves. **Our world runs on the power and information conveyed by electrical signals and ECSE is the home of the knowledge for how these technologies work.**

Student Preparation BEFORE doing this experiment: (Students should be able to)

- Download and install software on a Windows machine or Mac machine
- Plot data in Matlab
- Write and run a simple script in Matlab

Learning Outcomes AFTER doing this experiment: (Students will be able to)

- Generate a signal in Matlab and plot its power spectrum
- Apply basic digital filters in Matlab to signals to extract the desired information
- Apply basic preprocessing techniques to images in Matlab to prepare them for feature extraction
- Extract basic features from an image in Matlab, such as the color of the most prominent objects in the image
- Apply the KNN algorithm to image data in Matlab to classify them based on their extracted features
- Alpha: explain the role of modulation in transmitting a signal
- Alpha: explain the roles of demodulation and filtering a received signal

- Omega: apply more advanced and/or multiple digital filters to an audio signal to remove noise or unwanted parts of a signal
- Omega: implement a phase or frequency modulation scheme to model the transmission and reception of an audio signal within a communication system in Simulink

Software and Equipment Required:

- Matlab
- Simulink (Alpha and some Omega options)
- Matlab Image Processing Toolbox
- Matlab Machine Learning Toolbox
- Matlab Digital Signal Processing (DSP) Toolbox (Alpha and some Omega options)
- Matlab Audio Toolbox (Alpha and some Omega options)
- Matlab Communications Toolbox (Alpha and some Omega options)

Learning from Proof of Skills applied to this lab:

Matlab and Simulink:

- I have learned how to plot and interpret signals in the frequency domain and apply basic filters to them by completing the Matlab Signal Processing Onramp
- I have learned how to import images into Matlab, apply basic image preprocessing techniques, segment images and apply filters to images by completing the Matlab Image Processing Onramp
- I have learned how to extract features from data, the importance of splitting data into training and testing sets, the basics of classification methods, how to train a model, and how to evaluate the model's performance by completing the Matlab Machine Learning Onramp

PART A [Core] – Extracting Features from Traffic Signs Using Image Processing

In Part A of the lab, you will be further exploring the following concepts and how they relate to extracting useful features from traffic signs. All of the following terms should have been covered by the Matlab Image Processing Onramp. Please review the sections of the onramp dedicated to these concepts if you feel uncomfortable with them:

- *Data Import*: loading the image files from your storage to your workspace
- *Preprocessing*: enhancing brightness, sharpness, etc. to improve the clarity of the image being processed
- *Pixel Intensity*: the amount of a certain color value contained in a pixel, represented as a number between 0 and 255. For black and white images, each pixel contains one value. For color images, each pixel contains three values: one each for red, blue and green.
- *Thresholding*: an image processing technique in which a pixel intensity threshold is set and pixels are kept or removed from the image data based on their values relative to the threshold.
- *Masking*: an image processing technique where a filter or "mask" is applied to an image in order to keep or remove parts of the image based on whether or not they are contained in mask's pixels.
- *Morphological Operators*: an image processing technique in which a shape centered on a particular pixel is applied to a part of the image and used to determine whether or not to remove the pixel based on its relevance to other nearby pixels. This technique is often used to remove noise or small, connected areas of images.
- *Edge Detection*: an image processing technique that uses properties like color to divide images into lines.

Lab Activity Instructions:

Follow the instructions below to complete the Part A for Lab 03. Instead of submitting a proof of concept for this experiment, you will answer questions regarding your results and submit screenshots of them on Gradescope (assignment: "Lab 03 Part A: Image Processing"). At the end of the Gradescope assignment, you will answer questions about what you conceptually learned by doing the experiment.

Pre-Lab Tasks

1. You should have already completed the Matlab Image Processing Onramp as part of Proof of Skills: Lab 03.
2. Download the Matlab script template for Part A and the training images data set.

Data Import

1. Begin by copying "Lab03_PartA.m" and the "training_images" folder (after extracting it from the .zip file) to your current Matlab directory. Make sure the "training_images" is also on your Matlab path and that the "training_images" folder contains only images (not another folder called "training_images" – the script will give you an error if this is the case).

2. Open "Lab03_PartA.m" in Matlab. The provided code stores the "training_images" folder as datastore variable and displays image number *n* in 8-bit RGB. By default, *n* is 47, an image of a stop sign in very good lighting conditions, called "road59.png". Run the script to verify that you see the same image as below in Figure 1.

Figure 1: Default stop sign image

Separating the Colors

1. Separate the image into three color bands, red, green, and blue and plot them in the same figure alongside the original image using "tiledlayout" as shown in Figure 2. The resulting images will all be in grayscale, but the pixel intensity will correspond to how much of that color is present in each pixel; the lighter parts of the image indicate a higher intensity of that color, and darker parts indicate a lower intensity of that color.

Figure 2: Initial plots of color band images

2. Plot the color intensity histograms of each of the red, blue, and green color band images on the same set of axes, using "imhist()". Then, create a histogram for each individual color band image and plot them in the same Matlab figure alongside the histogram with all of the colors represented. Your results should look similar to Figure 3.

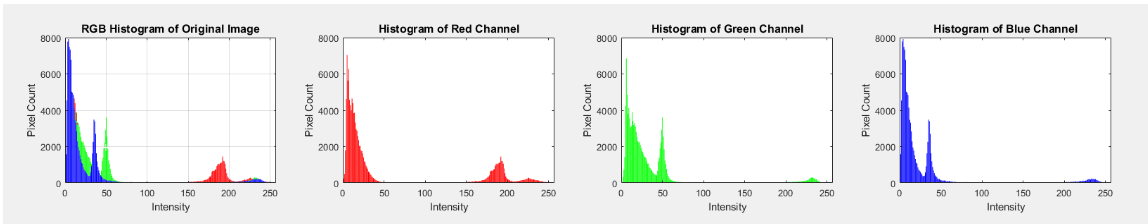

Figure 3: Plots of color band images with corresponding color band intensity histograms.

Identifying the Sign by Thresholding

1. Next, you will attempt to define what part of the image is the red stop sign by creating color thresholds. To do this, inspect the images and the histograms for each color band, then try to identify which range of intensities best represents parts of the image with those colors. Lighter pixels in a color band (values closer to 255) signify more of that color in the RGB image, while darker pixels (values closer to 0) signify less of that color in the RGB image. Choose any set of lower and upper thresholds for each color for now and move on to the next step. You can adjust your threshold values later based on the resulting images.
2. Using the threshold values, create masks for each color band that will indicate which pixels have intensities that lie between the lower and upper threshold values you defined above. For example:

   redMask = (imR >= minthreshold) & (imR <= maxthreshold)

   finds all pixels in the image "imR" with red intensities between the values of "minthreshold" and "maxthreshold". After creating the masks, plot each of them in the same figure as above, under their corresponding color band images, as shown in Figure 4. Based on the thresholds you set, your mask images may look very different.

Figure 4: Red, green, and blue mask images

3. Based on how you set your thresholds, you may see that the "STOP" text in all of your mask images is white. This is because white contains all 3 colors of the RGB spectrum (in RGB color space, white = [255, 255, 255]). To filter out the other colors while only keeping red, create a third mask by subtracting the green and blue masks from the red mask, then setting all values in the resulting mask that are negative to 0. If your thresholds have been set well, the stop sign octagon should be white, the "STOP" text should be dark, and a few smaller areas may be white, as in Figure 5 below. If your image does not look like Figure 5 (or better!), adjust your thresholds until you obtain a similar result.

Figure 5: Isolated stop sign

Further Refining the Image

1. Further steps can be taken to improve the identification of only the red parts of the traffic sign. First, any connected areas in the mask with a total number of pixels below a certain threshold can be removed, using the function "bwareaopen". Try removing all of the white areas with fewer than 100 pixels and plot the resulting mask. Adjust the minimum area as needed to obtain the best result. In the example in Figure 6, you can see that opening the image removed the small white specks in the background of the original mask image.

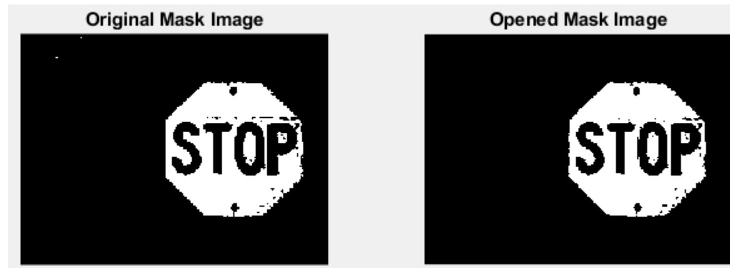

Figure 6: Opened mask image

2. Next, any "holes" inside the mask can be assumed to belong to the object that has been identified as a red traffic sign and can be closed using the function "imfill". For a stop sign, this function will eliminate the "STOP" text in the red object mask. Apply "imfill" with the "holes" option to your opened mask image. You should obtain a result similar to Figure 7 below.

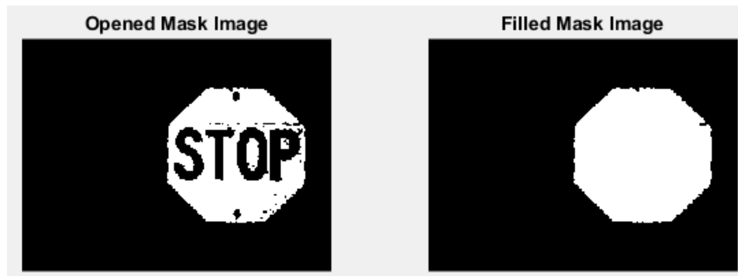

Figure 7: Filled mask image

3. Any rough edges can be smoothed out by using a morphological closing operation. Use the function "imclose" with a "disk" structuring element to smooth the edges of the stop sign. Adjust the radius of the disk as needed for the best results. After following all the refinement steps, you should end up with a sign image similar to that in Figure 8 below.

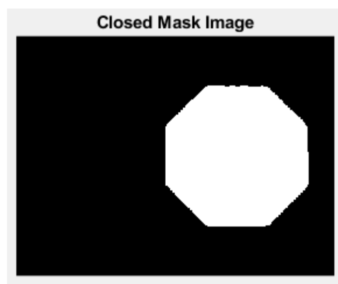

Figure 8: Refined stop sign mask image after smoothing edges

4. Finally, use the refined object mask to create a new image that only contains the red part of the STOP sign. You can do this by applying the mask to each of the color band images (for example, “redImage.*redObjectMask” applies the mask to the red color band image), then using the “cat” function to concatenate them into an “*n* x *m* x 3” RGB image array. Note: you may have to convert your mask into an integer array to apply it to your color band images. To do this, apply the “uint8” function to it first.

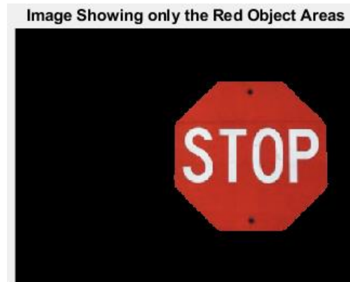

Figure 9: Final image showing only areas identified as “red” by the mask

Edge Detection

1. Edges of objects can provide useful data like the shape of the object. Use the function “edge” with the method “sobel” on the image in Figure 8Figure (the smoothed red object mask) to obtain the shape of the object. You should end up with an octagon similar to the image in Figure 10.

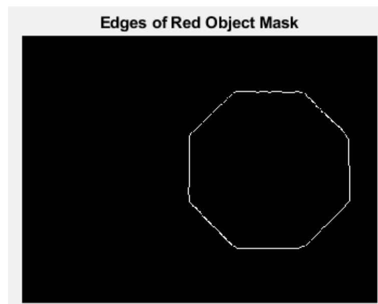

Figure 10: Edges of mask in Figure 8

Feature Extraction (Extra Credit)

1. Now you have two features that you can use to tell that a STOP sign is a STOP sign and not any other sign: the amount of red in the sign and its shape. Come up with a way to convert each of these two features into a quantity that you can use to identify whether or not a sign is a STOP sign.
    - How can you use the color data from your stop sign image to distinguish a stop sign from a speed limit sign or a crosswalk sign?
    - What information from the edges of the image can you use to determine the shape of the sign?
2. Load a given image (stop sign, speed limit sign or crosswalk sign) and calculate any quantities (features) that you think are useful for distinguishing a stop sign from the other sign types. To determine if it is a stop sign, compare your extracted features to values that you have determined are typical for a stop sign
3. Apply your algorithm to 5 different images that you didn’t use for setting your thresholds and record the results in a table like the one below:

Image Name	Actual Sign Type	Value of Feature #1	Value of Feature #2	Predicted Sign Type

4. How well does your method work? What is its success rate for these 5 test images? For which images does it fail and why?

Activity Summary

In this activity, you imported an RGB image as a 3-D array of its pixels, with the x-y axis containing the pixel's location, the z-axis containing the color, and the value of the element containing its intensity. Next, the images were pre-processed to enhance its brightness and contrast for easier feature extraction and was divided into 3, 2D vectors, each representing a different color band of the RGB image. The histograms for each of the color bands were then plotted to see the number of pixels that contained each intensity of that color and was used to determine thresholds for color identification. These thresholds were used to further remove parts of the image so that only the colors specified by the threshold remained. Next, morphological operators, such as "opening" to remove noise like small bits of red in the background and closing to smooth out rough patches where effects like shade might have reduced the intensity below your defined threshold, were applied. Finally, the edges of the image were identified by seeing where the color changed drastically, which was used to determine the shape of the object of interest. With this information, you were able to then write and test out an algorithm that identified images based on their color and shape characteristics.

Part B [Core] – Road Sign Classification Using Machine Learning

In Part B of this lab, you will implement a basic supervised machine learning algorithm to classify road signs into three categories: Stop, Speed Limit, and Crosswalk. You will gain hands-on experience with data import, feature extraction, model training, evaluation, and explore potential model improvements, using MATLAB.

The classes of signs to identify and their main characteristics are listed below and shown in Figure 11:

- **Stop Sign**: red, hexagonal shape.
- **Speed Limit Sign**: white, circular shape with a red border.
- **Crosswalk Sign**: square shape with yellow and blue colors.

Figure 11: Different images of the three classes of traffic signs to be classified in this lab

First, you will extract features from the images that will help to classify the images by sign type, then you will train a machine learning model using a training data set, and finally, you will evaluate your model's performance using a different, test set of images.

Background

Features and Feature Extraction

Features are the measurable attributes of an image that help the machine learning model distinguish between different classes, which in this case, are sign types. In this lab, we focus on:

- **Shape Properties**: area, perimeter, and aspect ratio.
- **Color Properties**: average intensity values of the red, green, and blue channels.

Feature extraction simplifies the dataset by reducing it's the complexity of the information it contains while preserving essential information required for classification.

Steps for Feature Extraction:

1. Read each image using the filenames provided by the .xml annotations.
2. Use image processing techniques:
    - Convert images to grayscale using "rgb2gray".
    - Extract shape properties using "regionprops".
    - Calculate average RGB intensities for color properties.
3. Save the extracted features and their corresponding labels in a structured format for training.

Training a Machine Learning Model

- In this lab, we use the **K-Nearest Neighbors (KNN)** algorithm, a straightforward yet powerful supervised learning technique.

- KNN classifies new data points by identifying the majority class among its nearest neighbors in the feature space.

Training Process Overview:

1. Normalize the extracted features to ensure uniform scaling across attributes.
2. Train the model using MATLAB's "fitcknn" function with the training features and labels.
3. Save the trained model for evaluation in the next step.

Model Evaluation

Testing data is critical for assessing the model's ability to generalize to unseen data. This ensures the model doesn't merely memorize the training data.

Steps for Evaluating the Model

1. Extract features from the test images following the same process as for the training images.
2. Use the trained KNN model to classify the test images.
3. Compare the predicted labels with the actual labels to evaluate the model's performance.

Generating a Confusion Matrix

A confusion matrix provides insights into the model's performance:

- **Rows** represent the true classes.
- **Columns** represent the predicted classes.
- **Diagonal values** indicate correctly classified samples.

A sample confusion matrix is shown below in Figure 12.

		True Classes	
		Positive (1)	Negative (0)
Predicted Classes	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 12: Confusion matrix

Lab Activity Instructions:

Follow the instructions below to complete the Part B for Lab 03. Instead of submitting a proof of concept for this experiment, you will answer questions regarding your results and submit screenshots of them on Gradescope (assignment: "Lab 03 Part B: Machine Learning"). At the end of the Gradescope assignment, you will answer questions about what you conceptually learned by doing the experiment.

Pre-Lab Tasks

1. You should have already completed the Matlab Machine Learning Onramp as part of Proof of Skills: Lab 03.
2. Download the following files for the lab:
    - the Matlab script template for Part B
    - the Matlab function parseAnnotations
    - the images and annotations data set data set. Extract this to your working directory and make sure all folders are on your Matlab path.

Data Import

In this section, you will load and visualize the dataset containing training and test images. You will load images from the training_images and test_images folders, parse the .xml annotation files to retrieve the class labels, and assign appropriate labels (Stop, Speed Limit, Crosswalk).

The Matlab script template already contains code for:

1. Loading images from training_images and test_images folders.
2. Running the function "parseAnnotations" to parse the .xml annotation files to retrieve filenames and labels.
3. Displaying a few samples of each class using MATLAB to ensure that the assigned labels match the sign class.
4. Defining a function "extractFeatures", to which you will add your own code for extracting features from the training and test data sets.

Run the Matlab script to verify that the displayed sample images have the correct labels assigned to them, as shown below in Figure 13.

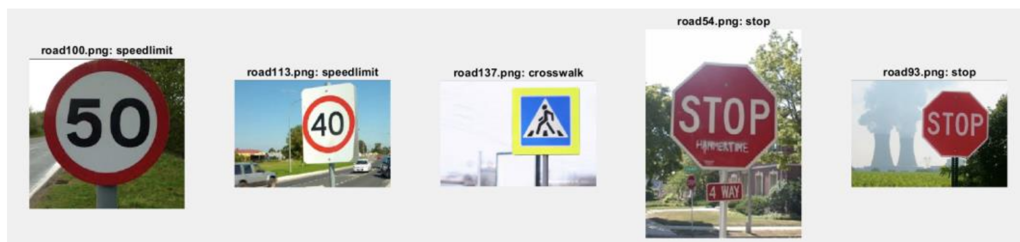

Figure 13: Sample traffic sign images with assigned class annotations

Feature Extraction

In this section, you will extract meaningful features from the images, such as shape and color, similar to what you did in Part A of this lab. Just as with the data import step, some of the Matlab code has already been provided for you to help you get started, such as the definition of the function "extractFeatures" (at the end of the script) and

code within that function that loads an image. You will add your feature extraction code to the “extractFeatures” function, then eventually call that function in the main body of your code to extract features from each of your data sets.

There are many ways to extract features from images, but you will follow the approach below to get started, then you can optimize feature extraction later to improve the performance of the model. Perform the steps below to create a basic feature extraction algorithm for your images.

1. Convert the image to grayscale and binarize it. Your image will be converted to black and white, as shown in Figure 14 below.

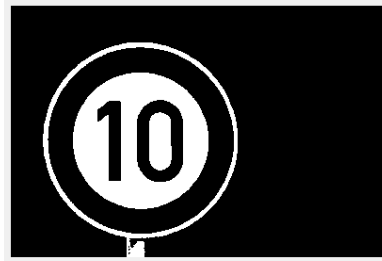

Figure 14: Binarized image of a speed limit sign

2. Fill in any small areas of bright pixels, using the “bwareaopen” function and a specified minimum connected area size to keep in the image.

3. Fill in any connected areas of dark pixels that are entirely enclosed by connected areas of bright pixels by using the “imfill” function. The opened, filled version of the speed limit sign in Figure 14 is shown below in Figure 15.

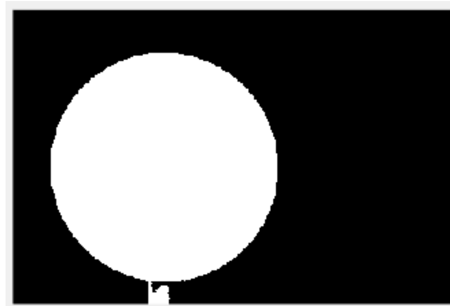

Figure 15: Opened, filled speed limit sign image

4. In this step, you will find the largest area of connected bright pixels in the image. In many cases, this should be the sign itself. To do this, use the “regionprops” function with the “Area” option, which will return a struct with the area information (in pixels) of each of the connected areas in the image. Then, find the index of the element in the struct with the largest area by using the “max” function on the area property of your struct. Finally, use that index to select only the largest connected area.

   Obtain the bounding box of the largest area by adding the option “BoundingBox” to your original “regionprops” function call. The “BoundingBox” option will return the x- and y-coordinates of the top left and bottom right corners of the smallest rectangle that includes the largest area of connected pixels. Plot the bounding box of the largest area of connected pixels to verify that it has identified the most relevant region by using the “rectangle” function and the coordinates of corners of the bounding box.

Once you do this, you should see that the bounding box frames the filled sign in the image, as shown in Figure 16 below.

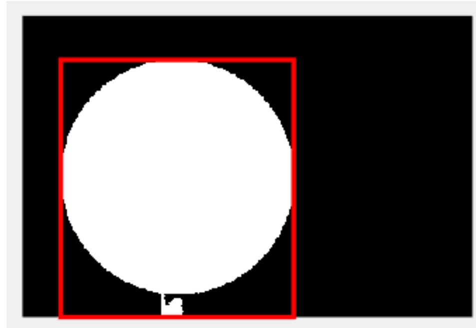

Figure 16: Bounding box of largest connected area of bright pixels

5. Extract geometric features of the largest connected area by adding additional options to your "regionprops" function call. What shape features will help you tell the difference between a stop sign, speed limit sign, and crosswalk sign? Area? Perimeter? Aspect ratio? Circularity? Look at the documentation for the "regionprops" function to see the entire list of which region properties are available for you to use. Choose one or two to extract and use in your initial machine learning algorithm.

6. It will also be useful to extract features from the sign image based on its color properties. Adding the "SubarrayIdx" option in the "regionprops" function will give you a cell array containing the indices of the pixels contained inside the bounding box. Use these indices to create a mask that will only keep the area of the original RGB image that is within the bounding box, as shown in Figure 17 below. This will limit the extracted color information to the region of the image that we've identified as belonging to the traffic sign only.

Figure 17: Original RGB image masked by the bounding box of the largest area of connected pixels

7. Extract color information from the image within the bounding box that will help you distinguish between the different sign classes. To start, try calculating the mean value of the red, green, and blue pixel intensities for the image. What other color information could you extract that would be helpful?

8. When you are satisfied with the code that will extract features from your data, run the template code to extract features from the training data set (via running the "extractFeatures" function).

9. Once you have a set of extracted features, store them in a new row of a matrix that contains all of you extracted features. Also create a column vector with a label denoting the class of the sign image for which you just extracted features. The machine learning algorithm must know which labels belong to which set

of extracted features. When you run your code, it should loop through all of the images in the datastore, extract features for each, and store them in a matrix that will be used in the machine learning algorithm.

Model Training

Now that features have been extracted, the machine learning model can be trained using the features from the training data set:

1. Although the features have been extracted, they must be normalized to ensure fair comparison of the features across different scales. Normalize your features matrix using the function "normalize". The function "normalize" calculates a z-score and normalizes the extracted feature values such that their mean is 0 and their standard deviation is 1. Along with the normalized features themselves, you'll need to save the "centering values" and "scaling values" by providing variables for those function outputs as shown below:

   ```
   [N,C,S] = normalize(___)
   ```

   where "N" are the normalized feature values, "C" is the centering value and "S" is the scaling value. You will need to supply these as inputs when you normalize the features you extract from the test data.

2. Fit the KNN model by calling the "fitcknn" function and sending your extracted features matrix and labels vector as arguments. Also specify the number of neighbors to use in the classification algorithm. Use 5 neighbors to start. You can adjust this later to improve the performance of your model.

Evaluate the Performance of the Model with Test Data

After running the "fitcknn" function, the model will have been trained on the data from the training images. In order to test the model, features need to be extracted from the test data and sent to the model for evaluation. The performance of the model will be evaluated using a confusion matrix.

1. Using the function "extractFeatures", extract the features from the test data and store it in a matrix. Also store the labels from the test data in an array.
2. Normalize the features from the test data using the centering and scaling values from when you normalized the training data by using the syntax:

   ```
   N2 = normalize(A2,"center",C,"scale",S)
   ```

3. Use the "predict" function to have the model predict labels for your test images based on the trained model and features extracted from the test data. Depending on your feature extraction algorithm, this may take a few minutes.
4. Once the model has finished evaluating the test data, create and display a confusion matrix by calling the function "confusionmat". Calculate the model's accuracy by dividing the number of correct predictions by the total number of test images evaluated by the model. An example confusion chart is shown in Figure 18.

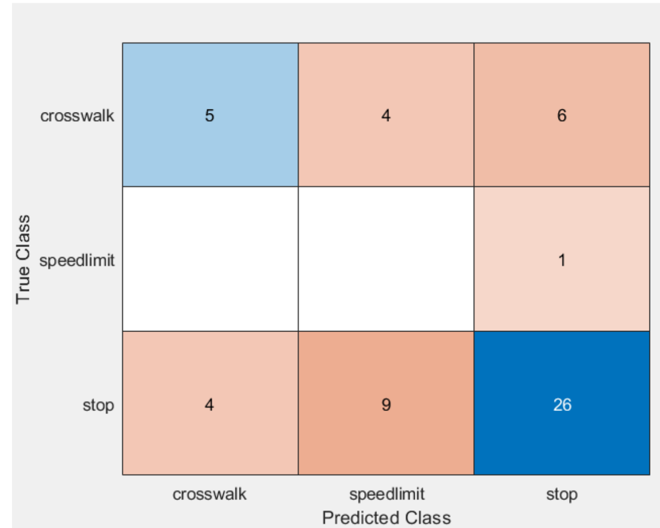

Figure 18: Confusion chart for KNN model predicting the classes of three types of traffic signs: stop, speed limit and crosswalk.

Improve the Performance of Your Model

Likely, your model will not have achieved 100% accuracy in predicting the labels of the test data. Answer the questions below to start improving your model:

1. For which images did your model incorrectly predict the class of the traffic sign in the image?
2. Why do you think the model incorrectly predicted the class of those images? Was it an error in feature extraction? What happened? You will need to investigate individual images to answer this question.
3. What improvements could you make in feature extraction to increase the accuracy of your model? Below are a few suggestions:
    - Consider other aspects of the traffic sign images for which you could extract additional or better features. What additional shape information could you use? What additional color information could you use?
    - Apply techniques like rotation, scaling, and brightness and contrast adjustments to your images before feature extraction.
    - Optimize your KNN algorithm by adjusting parameters like the number of neighbors or the distance metric used.
4. Implement at least one improvement to your model and show that it increases its accuracy (above 50% at the minimum). Which one did you choose? What led to it increasing the accuracy of your model?

Activity Summary

In this activity, you trained a KNN (k-nearest neighbors) algorithm to predict the class of images of three different types of traffic signs: stop, speed limit and crosswalk. Machine learning uses algorithms to identify patterns in data and make predictions (or classifications) on that data, based on how it has been trained. The algorithms are trained using features, which are measurable properties or characteristics extracted from the data. In order to train an algorithm, features are extracted from a training data set and the extracted features are sent to the model along with the labels identifying the class of each piece of data. Once a model is trained, its performance can then be evaluated by having it predict the classes of a separate set of test data and creating a confusion matrix, which shows true and false positives and negatives. Further improvements in the accuracy of a model can be made by improve feature extraction and adjusting model parameters, such as how many nearest neighbors are used in the classification scheme.

Part C [Alpha/Omega] – Alpha Experiment and Omega Exploration

Alpha Experiment: Amplitude Modulation (AM) Communication System in Simulink

Communication systems can transmit information wirelessly by modulating an electrical signal and sending it over the air via an antenna. Modulation takes a signal at a lower frequency, such as an audio recording, and encodes it onto a higher frequency signal that can be transmitted over the air. In this alpha experiment, you will simulate all the steps of an amplitude modulation (AM) communication system in Simulink and explore the effect of channel noise on your transmitted signal.

Omega Exploration: Signal Modification and Filtering Background

Electrical signals can convey information in a variety of ways, whether it be a coded message, a recorded song, a live instrument, or via a communication system. In this omega exploration, you will explore the different ways in which signals can be filtered to obtain desired information or modify recorded audio, as well as how signals can be modulated to create new sounds and send messages wirelessly.

Choose Your Adventure! Alpha Experiment or Omega Exploration

Alpha Experiment

Explore the transmitting information wirelessly by simulating an amplitude modulation (AM) communication system:

1. Create an AM communication system in Simulink
2. Explore the effect of channel noise on the quality of the signal you send through the communication system
3. Send an audio signal through your communication system to better understand the effects of modulation and demodulation on the frequency components of your signal.

See below for more instructions

Omega Explorations

Explore applications of signals and filters in one of the following:

- **Decode the Hidden Message** – design a system in Simulink that isolates and amplifies the message hidden in the noisy signal.
- **Modify your Favorite Song using Filters** – choose and design filters that will isolate or eliminate certain instruments in your favorite song in Simulink or a DAW.
- **Software Synthesizer** – design audio waveforms in Simulink by applying filters and other signal modification methods.
- **Analog Communication System** – design an FM or PM communication system in Simulink, including filtering, modulation and demodulation.

Technical and functional requirements for your application are listed below.

Alpha Experiment Guide

Follow the instructions below to complete the alpha experiment for Lab 03. Instead of submitting a proof of concept for this experiment, you will answer questions regarding your results and submit screenshots of them on Gradescope (assignment: "Lab 03 Alpha Experiment"). At the end of the Gradescope assignment, you will answer questions about what you conceptually learned by doing the experiment.

In this Alpha Experiment, you will learn about a method used for transmitting signals wirelessly, called Amplitude Modulation (or simply AM) and build a model AM communication system in Simulink. Amplitude modulation encodes a message signal as the amplitude of a higher frequency wave, the carrier wave, which is used to transmit the message over the air wirelessly. While the signal is traveling through the air, it will encounter sources of noise, which will add unwanted components to our signal across the frequency spectrum and make it more difficult to decipher our message at the receiver. Once the signal has made it to a receiver, it must be demodulated, filtered and amplified to recover the original message. You will start by transmitting a sine wave to better understand what AM does to a signal, then you will send an audio signal through your communication system to investigate the effects of noise and the modulation process on your signal's quality.

Required Software

- Matlab
- Simulink
- Matlab Communications Toolbox (install first, since it should install the other toolboxes below)
- Matlab Signal Processing Toolbox (required for Communications Toolbox)
- Matlab Digital Signal Processing (DSP) System Toolbox (required for Communications Toolbox)
- Matlab Audio Toolbox (required for Communications Toolbox)

Lab Activity

Background: Amplitude Modulation (AM)

Amplitude Modulation (AM) is a type of modulation scheme used to transmit radio waves in which the amplitude of the carrier wave $c(t)$ varies according to the amplitude of the message wave $m(t)$.

Carrier waves are pure waves with a set frequency (you can imagine this as being something similar to regular sine waves) and do not carry much relevant information on their own. The frequency of a carrier wave correlates to the frequency of the communication channel. If you are listening to a radio, the frequency of the radio channel you are on is the same frequency as the carrier wave. Carrier waves are described by the mathematical form:

$$c(t) = A_c \sin(2\pi f_c t) \quad (1)$$

where A_c is the amplitude of the carrier wave and f_c is the frequency of the carrier wave.

A message wave is the wave that holds the information that we want to transmit (such as speech, text, data, etc.). It is also sometimes called the modulating signal, or the input wave. Message waves are described by the mathematical form:

$$m(t) = A_m \sin(2\pi f_m t) \quad (2)$$

where A_m is the amplitude of the message wave and f_m is the frequency of the message wave.

To perform amplitude modulation, we impose the message wave onto the carrier wave. What this ultimately does is change the amplitude of the carrier wave to correspond to the information held by the message wave. What amplitude modulation looks like in the time domain is shown in Figure 19.

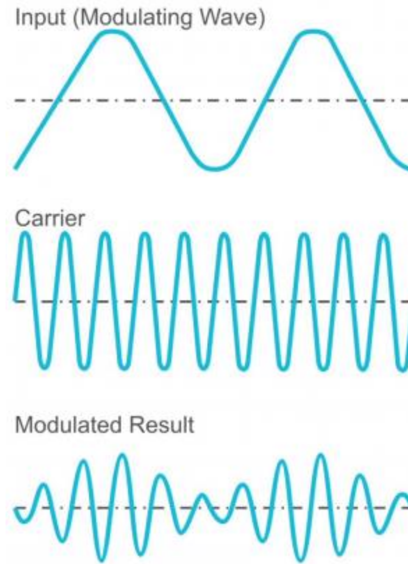

Figure 19: Amplitude modulation (from taitradioacademy.com)

The mathematical expression for amplitude modulation is described to be:

$$f(t) = A_c\left(1 + \frac{A_m}{A_c} sin(2\pi f_m t)\right)\sin(2\pi f_c t) \tag{3}$$

The Amplitude of Modulation parameter μ (also known as the Modulation Index) can be found by taking the ratio of the amplitude of the message wave and the amplitude of the carrier wave:

$$\mu = \frac{A_m}{A_c} \tag{3.1}$$

Thus, the resulting mathematical form of final modulated wave would look like this:

$$f(t) = A_c\big(1 + \mu\, sin(2\pi f_m t)\big)\sin(2\pi f_c t) \tag{4}$$

where A_c and A_m are the amplitudes of the carrier and message waves respectively, and f_c and f_m are the frequencies of the carrier and message waves.

Once the modulation is complete, the modulated signal will have three corresponding frequencies. There will be the carrier signal frequency at f_c, and two frequencies corresponding to the message signal at:

$$f_{m1} = f_c - f_m \tag{5.1}$$

$$f_{m2} = f_c + f_m \tag{5.2}$$

These three frequencies are plotted in the frequency domain in Figure 20 below.

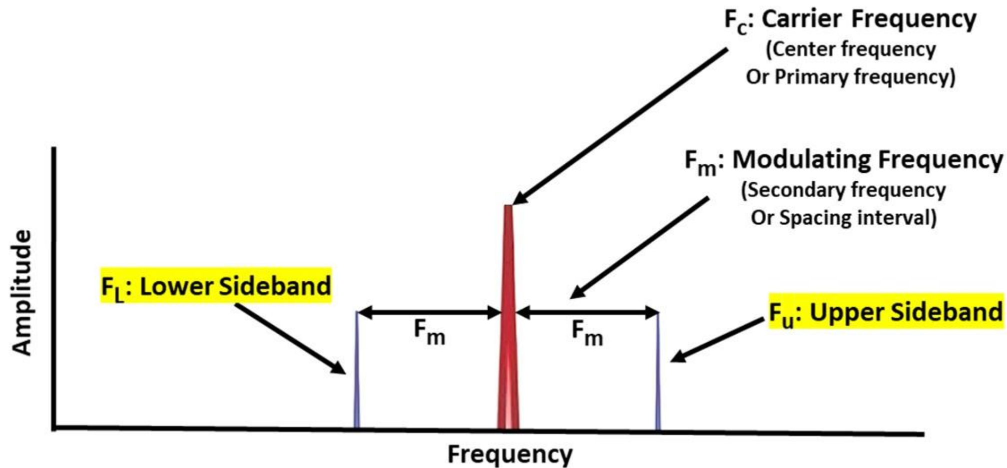

Figure 20: Three frequencies for the modulated signal (from fluidlife.com)

Amplitude Modulation

1. Open a blank Simulink model. Under the "Modeling" tab, click on "Model Explorer". In the explorer pane, click on "Base Workspace". Under the "Add" menu at the top of the window click on "Simulink Parameter"; it should appear in the main pane of the window as "Param". Name the parameter "fs" (for the sampling frequency) and set its numerical value to 10000 Hz. Add another parameter "fc" for the carrier wave frequency and set its value to 2500Hz.
2. Close out of Model Explorer. In the main Simulink model, add the following blocks to your model as shown in Figure 21:
   a. Message Signal: Sine Wave (DSP) with amplitude 1V and frequency $f_m = 500$ Hz. Set the sample time of the block to "1/fs". Label the block "Message".
   b. Time Scope connected to the message signal.
   c. Spectrum Analyzer (DSP) connected to the message signal.

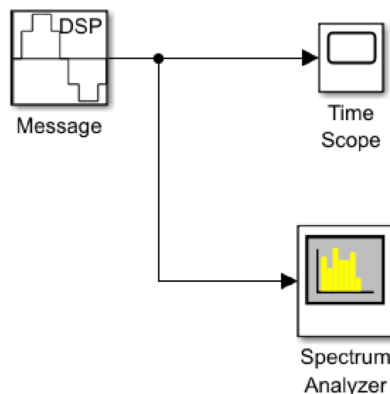

Figure 21: Message signal with time scope & spectrum analyzer

3. Go to the "Simulation" tab and set the simulation Stop Time to 1s.

4. Run the simulation, then look at the signal in the frequency domain by clicking on the Spectrum Analyzer. (To have a better view of the signal, zoom in by scrolling on the graph so that the x-axis ranges from 0 Hz to 700 Hz. Ensure that the peaks of your frequency graph are visible by aligning your graph so the y-axis ranges from 18 to 30 dBm.) Is the signal located in the frequency domain where you expect it to be?
5. Look at the message signal in the time domain by clicking on the Scope. (If your computer is crashing when clicking on the Scope, set the Stop Time to 0.02s. Remember to reset the time back to 1s after, as the Spectrum Analyzer would not display correctly otherwise. You can also drag your mouse across the screen horizontally to view a shorter sample.) What is its maximum value?
6. Add a “Constant” block to the model and set its value to what you determined to be the maximum value in the previous step.
7. Add a “Sum” block that sums the message signal and the constant. This is needed to properly modulate the signal.
8. Now add a Sine Wave (DSP) block with amplitude 1V and frequency “fc” to serve as the carrier signal. Set the sample time of the block to “1/fs”. Name this block “Carrier”.
9. Add a “Product” block to the model and multiply the output of the sum block and the carrier wave. This amplitude modulates the wave. Your model should now look like the model in Figure 22.

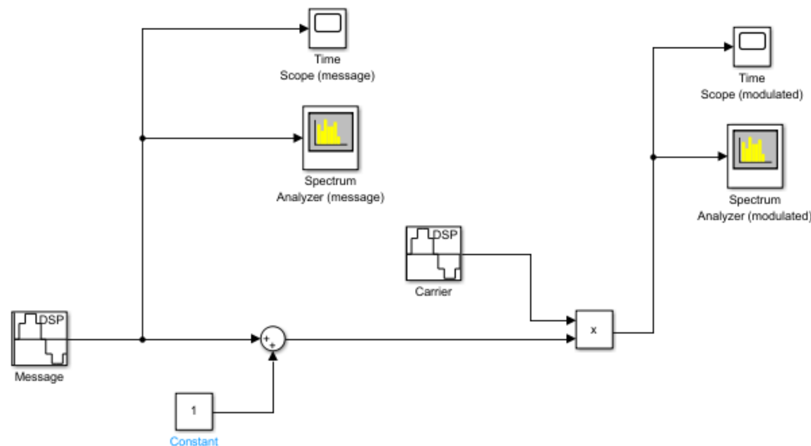

Figure 22: Amplitude modulator with message & carrier waves

10. Run the simulation again and view the result in the time scope. To get a better view, go to the “View” tab and click on “Configuration Properties”. In the Display tab, you can adjust the y-axis view. For the x-axis, drag your mouse horizontally across the screen to get a better time range to view your signal. Can you discern both the message signal and the carrier signal in the *time* domain? How?
11. View the result in the spectrum analyzer. Configure your display so the x-axis ranges from 2 kHz to 3.5 kHz. Can you discern both the message signal and the carrier signal in the frequency domain? How? What did multiplying the message signal with the carrier signal do to the location of the information of the message signal in the *frequency* domain?

Signal Transmission and Channel Noise

1. Once the signal is modulated it needs to be transmitted over the air. While the physical process of transmission and reception won't be simulated, the noise that the signal picks up during the time it travels between the transmitter and receiver will be modeled with an "AWGN Channel" block from the Communications Toolbox. Add this block to your model and connect its input to the output of the product block.
2. Open the AWGN Channel block. Set the input signal power to 1W and the SNR (dB) to 25. What is SNR?
3. Connect a scope and spectrum analyzer to before the AWGN Channel and after the AWGN Channel. Your model should now look like the model in Figure 23.

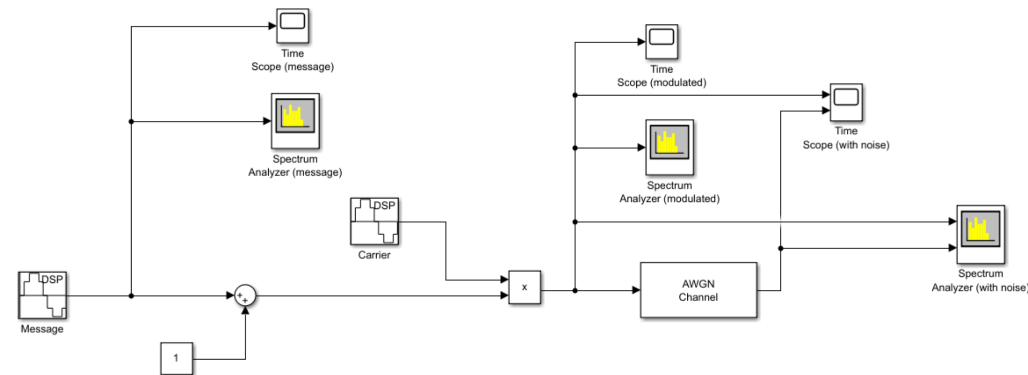

Figure 23: AWGN channel with scope & spectrum analyzer

4. Inspect the two signals using the scope. Was any noise added? How can you tell?
5. Decrease the SNR to 1 and inspect the signal in the time domain again. What has changed?
6. Inspect the two signals in the frequency domain. At which frequencies was the noise added?
7. You can leave the SNR at 1 for the rest of the lab.

Amplitude Demodulation

As mentioned previously, after modulating the original message signal with the carrier signal, there are components at three frequencies, corresponding to:

- The carrier signal at $f = f_c$
- A message signal at $f = f_c - f_m$
- A message signal at $f = f_c + f_m$

If the modulated signal is multiplied with another signal at the carrier frequency, new frequency components are added to the signal. Similar to what we had previously done, multiplying the signal gives us a new modulated signal centered around three frequencies (see Figure 20 above for a reminder). Since the signal we are multiplying by has three frequency peaks, we will have a total of 9 total peaks if we look at the whole frequency spectrum. This is because we will have three center frequencies that will each have three peaks corresponding to our initial message and carrier waves.

As a result, there is now a copy of the modulated signal centered at $f = 2f_c$ and one centered at $f = 0$ Hz. The original signal (with a reduced amplitude) can be recovered by applying a band-pass filter at the center frequency of the original message signal $f = f_m$.

1. Add another Sine Wave (DSP) block and set the sampling frequency (1/sample time) and signal frequency to match that of the original carrier signal.
2. Add another Product block to multiply the signal leaving the AWGN Channel block with the sine wave from the previous step. Your model should now look like the model in Figure 24.

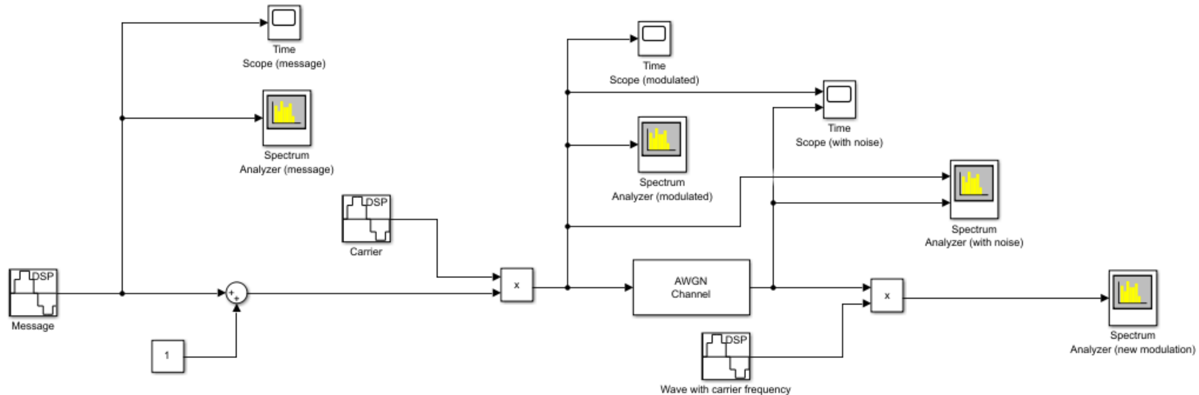

Figure 24: Amplitude demodulator with scope & spectrum analyzer

3. Inspect the signal leaving the Product block in the spectrum analyzer. At which frequencies are there peaks in the spectrum now? Do any correspond to the frequency of the original message f_m? (Hint: There are two for the spectrum range of 0 to 5kHz).
4. Add an “Analog Filter Design” block after the product block. The “Analog Filter Design” block implements filters that can be built out of analog circuit components. Set the “Design method” to “Butterworth”, the “Filter type” to “Bandpass”, and the “Filter order” to 2. Set the “Lower passband edge frequency (rad/s)” to 300*2*pi and the “Upper passband edge frequency (rad/s)” to 700*2*pi. This creates a passband filter with a passband between 400 Hz and 600 Hz, which is centered on f_m.
5. View the time domain signal after the Butterworth filter in a scope, along with the original message signal. Does it resemble the original signal? What is different about it? What could be the cause for the differences between the two? How does the resulting time domain signal change if you make the lower and upper frequency limits of the pass band 400*2*pi and 600*2*pi?
6. To return the demodulated signal to the same amplitude as the original message signal, add a “Gain” block after the Butterworth filter with a gain of 2. Your model will now look like the model in Figure 25. View the resulting signal alongside the original message signal. What effect would decreasing the SNR of the AWGN Channel block have on the demodulated signal? Verify this by varying the SNR.

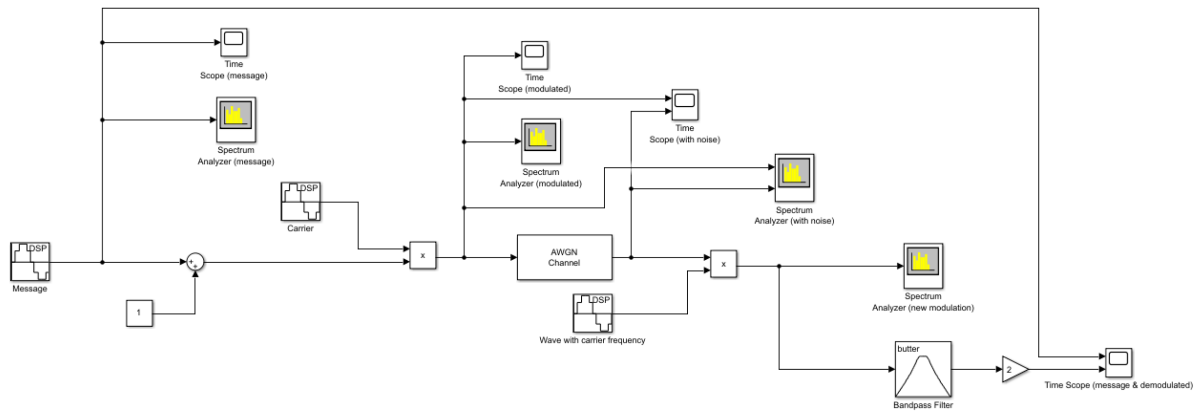

Figure 25: Final model for amplitude modulation, transmission, demodulation and amplification.

Transmitting an Audio Signal

Having transmitted a simple sinusoid via AM, you can now try to transmit an audio signal.

1. First, set the sampling frequency parameter $f_s = 250000$ Hz.
2. Next, replace the original "Message" Sine Wave block with a "From Multimedia File" block. Set the "Samples per audio channel" to 1. The default file "speech_dft.mp3" is 5 seconds long, so update your Simulation Stop Time to 5s. Listen to the input file by finding it at "C:\Program Files\MATLAB\R2023b\toolbox\dsp\samples", where your Matlab version ("R2023b") may differ.
3. Run the simulation and view the audio signal in the time domain. What is its maximum amplitude (look at both positive and negative polarities of the signal)? Enter this value as the constant that is added to the signal.
4. The carrier frequency must be larger than the frequencies present in the signal itself. Inspect the signal in the spectrum analyzer. What is the maximum frequency present in the signal? Set the carrier frequency to be approximately 5 times the maximum frequency.
5. Run the simulation again and inspect the "Modulated Signal" in the frequency domain. You will need to insert a "Rate Transition" block with a sampling time of "$1/f_s$" in front of the Spectrum Analyzer to view the signal. Can you identify the carrier signal in the frequency domain?
6. Since the signal should reside in the frequency range $0 < f \leq 10$ kHz after demodulation, you will have to change your filter parameters. Change the "Lower passband edge frequency (rad/s)" to 50*2*pi and set the "Upper passband edge frequency (rad/s)" to 10000*2*pi.
7. Run the simulation again and compare the final, filtered signal and the original message signal in the time domain. How are they different? What could cause this?
8. Add a "Rate Transition" block to your model with a sampling time "$1/f_s$" and connect the output of the filter to its input. To listen to your transmitted signal, add a "To multimedia file" block after the Rate Transition block. This will save the output to an audio file format in your Matlab directory. Run the simulation again to generate the audio file and open it to play it. How does the file sound? What caused the audio noise?

9. Increase the SNR in the AWGN Channel block to 25, run the simulation again, inspect the signal in the time domain and listen to the output audio file again. Has the clarity of the transmitted message improved? Does it resemble the original message in the time domain?

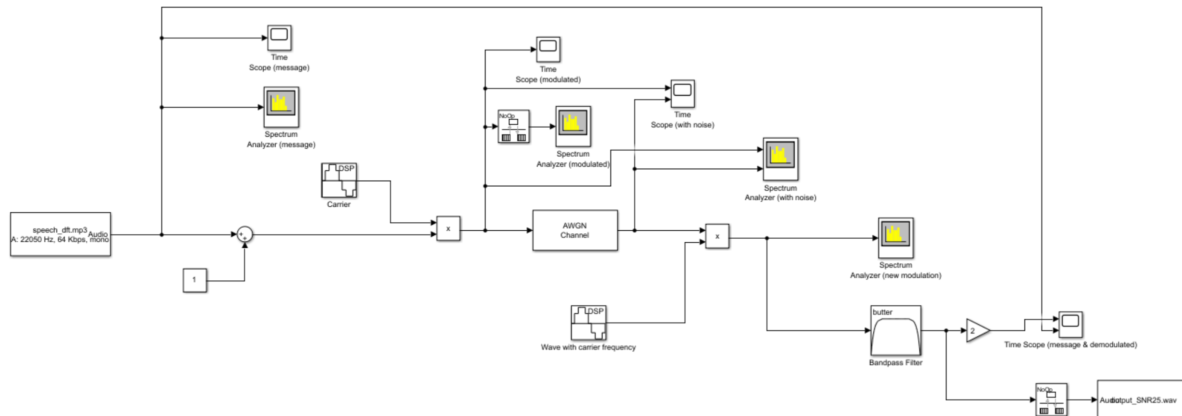

Figure 26: Final model for transmission of an audio signal via AM.

Activity Summary

Modulation is a signal processing tool that lets us encode information we want to transmit onto a high frequency wave, known as the carrier wave. Modulation can help us get higher quality transmissions by reducing noise and interference and increasing transmission power. We use this in our day-to-day telecommunication systems in various ways, ranging from securing the quality of our transmissions to allowing multiple transmissions to happen simultaneously.

Amplitude Modulation (AM) is one type of modulation that changes the amplitude of the carrier wave based on the amplitude of the message signal. We can do this by multiplying the carrier wave with the message signal to get a modulated signal that can be transmitted reliably across communication channels.

The modulated signal will encounter and pick up noise between its transmission and reception. In real life, this can range from other sounds such as conversations to walls that block waves from passing. In this lab, we used the AWGN channel to simulate white gaussian noise. If the signal to noise ratio (SNR) is small enough, it can interfere with our comprehension of our message signal.

After we receive the modulated wave that has gone through some noise, we can demodulate it by multiplying it with another wave with the same frequency as our carrier wave. This will give us our modulated signal centered around 0 Hz and $2f_c$, and our message signal in its initial frequency range.

To access our message, we must use a bandpass filter to extract only the frequency components of the signal corresponding to our message signal, which eliminates the carrier wave and other noise that the signal has picked up. With that, we have successfully transmitted and received our message!

Omega Exploration Application Requirements

A short description and the technical specifications for each Omega Exploration option are listed below.

Decode the Hidden Message

Filtering can be used to separate wanted information (the "message") from unwanted information (the noise) in a signal. In this Omega Exploration, you will try to decode the message hidden in a very noisy audio clip by designing and applying filters in Simulink to eliminate the noise. No physical circuit is required for this project – your simulation counts as your experiment in this case.

A .wav file of the noisy signal can be downloaded here. *Note*: turn your volume down before opening it!

Requirements:

- ☐ *System architecture*: in Simulink
  - You must filter the signal to isolate the message and will likely need to use multiple filters. In each case where you've chosen a filter design, discuss at least one other filter type that could have performed the same function and justify why you chose one filter type over the other.
  - You must also include one amplifier to boost the signal volume.
- ☐ *Output signal*:
  - You must display both your original signal and filtered signal in the frequency domain and time domain.
  - You must play your filtered signal through speakers.
- ☐ *Experimental Verification / Test Cases*:
  - Show your original signal and filtered signal in both the frequency domain and time domain.
  - Play your filtered signal through speakers (include in your presentation video).
  - Decode the message hidden in the signal. *Hint*: what are different ways of encoding information via taps or dots and dashes?

Modify your Favorite Song using Filters

Active filters (filters that can also amplify certain parts of a signal) are used extensively during the music production process to shape the sound to meet an artist's aesthetic goals. This stage of music production in which different audio tracks are adjusted so that they can later be combined together into a single, final signal (or track) is called mixing.

On the other hand, once a song is finished, you can also use filters to try to single out or eliminate individual instruments. In this Omega Exploration, you will design filters to change the sound of a chosen song, and isolate (keep) and eliminate (remove) individual instruments from it. While you can accomplish this in Simulink, many programs are much more convenient for audio editing.

Some free options for audio editing software include digital audio workstations (DAWs) such as Reaper and Waveform Free. Other programs, such as Audacity, provide many tools for audio editing, but offer fewer conveniences than DAWs for manipulating music tracks.

No physical circuit is required for this project – your simulation counts as your experiment in this case.

Requirements:

- ☐ *System architecture*: in Simulink or digital audio workstation (DAW)
  - You must apply filters to modify your chosen song. In each case where you've chosen a filter design, discuss at least one other filter type that could have performed the same function and justify why you chose one filter type over the other. *Note*: in audio engineering, there are multiple types of high pass, low pass, band pass, and band stop filters.
  - You must use at least three different types of filters.
- ☐ *Output signal*:
  - You must display both your original and filtered signals in the frequency domain.
  - You must play your filtered signal through speakers to hear the audible difference between the original and filtered version of your chosen song.
- ☐ *Experimental Verification / Test Cases*:
  - You must list the frequency ranges corresponding to each of the instruments you are trying to isolate or eliminate.
  - You must try to isolate a single instrument (and filter out all others).
  - You must try to eliminate a single instrument.
  - You must try to isolate **OR** eliminate **two** instruments.
  - In each case, you must show your original signal and the filtered signal in the frequency domain for one point in time when the difference is clear.
  - In each case, you must play your original signal and the filtered signal through speakers to verify that your filter designs have achieved their audio goals (include this in your presentation video).

Software Synthesizer

A synthesizer is a musical instrument that can produce a wide variety of sounds by modifying and combining simple waveforms, such as sine, square, sawtooth, and triangle waves. Although the most basic function of a synthesizer is to produce a waveform at with specific fundamental frequency (musical note), synthesizers also use a variety of tools, such as filters, envelopes, and modulators, to modify the character of the waveform's sound. In this Omega Exploration, you will design your own signals by experimenting with combining waveforms and applying filters, envelopes, and modulators. No physical circuit is required for this project – your simulation counts as your experiment in this case.

Requirements:

- ☐ *System architecture*: in Simulink
  - Input stage: blocks that generate a sine, square or triangle wave
  - Signal modification stages must include
    - Filter(s). In each case where you use a filter, justify why you chose that filter to achieve your goals instead of another filter.
    - At least one of the following special effects:
      - Amplitude modulator (you may not use Communication Toolbox blocks specifically for AM – you must do some math to implement AM yourself!)
      - Envelope generator (you don't have to follow the ADSR model, but you should show that you can control the shape of the envelope to some extent)
      - Frequency modulator (you may not use Communication Toolbox blocks specifically for FM – you must do some math to implement FM yourself!)
  - Output stage:
    - "To Multimedia File" block – to save the signal as an audio file
    - Spectrum Analyzer
- ☐ *Output signal*:
  - Must play resulting waveforms through speakers
  - Must display the waveform after the modification stage and after the filter stage
- ☐ *Experimental Verification / Test Cases*:
  - You must design "instruments" with the following characteristics using your synthesizer. In each case, you should describe the initial idea for the sound you wanted and the design choices you made to try to produce that sound.
    - Bass instrument: primarily low/bass frequencies
    - Rhythm instrument: primarily middle frequencies
    - Lead instrument: primarily high/treble frequencies
  - Each of your designed instruments must include a filter
  - One of your designed instruments must be sum of at least **two** signals
  - One of your designed instruments must use the chosen special effect in your synthesizer
  - You must play samples of each of your designed instruments through speakers
  - You must show the frequency spectrum of each of your designed instruments and explain how it corresponds to the sound of that instrument

Analog Communication System

Communication systems transfer information from once place to another by sending and receiving electromagnetic signals. Each of these systems consists of multiple stages meant to prepare the information, or signal, for transmission from one place to another. For example, in order to send a signal wirelessly, a process called modulation is required, which "encodes" the data onto a wave which can travel long distances through the atmosphere. The signal is then sent through the air via an antenna and can be picked up by a receiver. Once a receiver picks up the signal, it must be demodulated to recover the original signal. In this Omega Exploration, you will design a basic communication system in Simulink and send an audio signal of your choice through it. No physical circuit is required for this project – your simulation counts as your experiment in this case.

Requirements:

- ☐ *System architecture*: in Simulink
  - Input stage: audio file
  - You must include the following stages in your system
    - Modulation stage – frequency modulation (FM) **OR** phase modulation (PM).
    - Added white Gaussian noise (AWGN) channel block – simulates noise added to the signal between the transmitter and receiver. *Note*: you will need the Communications Toolbox for this block.
    - Demodulation stage
    - Filtering stage
    - *Note*: you may not use Communications Toolbox blocks specifically for AM or FM in your modulation or demodulation stages – you must implement them yourself!
  - Output stage:
    - "To Multimedia File" block – to save the signal as an audio file
    - Spectrum Analyzer and Time Scope
- ☐ *Operating conditions*
  - AWGN channel block settings → mode: SNR; SNR (dB): 0; Input Signal Power: ≥10 mW
- ☐ *Output signal*:
  - Must play resulting signal through speakers
  - Must display the signal in the frequency and time domains
- ☐ *Experimental Verification / Test Cases*:
  - Your input audio signal must pass through a modulation stage, an AWGN channel block, a demodulation stage, and a filter stage.
  - You must display the signal in the frequency and time domains after each stage.
  - You must compare the audio quality of the signal at three stages: at the input, immediately after demodulation and after the final filtering stage.

Note: Due to the Nyquist-Shannon sampling theorem the sampling frequency for all of your signals and spectrum analyzer must be at least twice the maximum frequency of any of your signals. For this reason, you may want to low-pass filter your input signal before modulating it.