Simultaneous Call Transmission (SCT)

Final Report

sddec22-13

Client: Collins Aerospace
Advisor: Dr. Andrew Bolstad

Team Members
Sullivan Jahnke
Json Rangel
Tyler Mork
Austin Rognes
Hani El-Zein

Email: sddec22-13@iastate.edu
Website: sddec22-13.sd.ece.iastate.edu

December 7th, 2022

Executive Summary

Development Standards & Practices Used

- IEEE 2755-2017 - *IEEE Guide for Terms and Concepts in Intelligent Process Automation*
- IEEE 1139-2008 - *Standard Definitions of Physical Quantities for Fundamental Frequency and Time Metrology - Random Instabilities*
- IEEE 1641-2010 - *IEEE Standard for Signal and Test Definition*

Summary of Requirements

- The algorithm must output a model that can be used to detect SCT
- Training data for the algorithm will be simulated
- The alert must come within 1 second of the signals being transmitted
- The model should be able to run on the hardware that it will be implemented on, meaning that it should be efficient enough for a small device

Applicable Courses from Iowa State University Curriculum

- COM S 228 - *Introduction to Data Structures*
- EE 224 - *Signals and Systems I*
- EE 321 - Communication Systems I
- EE 422 - Communication Systems II
- ENGL 314 - *Technical Communication*

New Skills/Knowledge acquired that was not taught in courses

- Machine Learning Concepts
- Python Development
- Complex Baseband Modulation
- IQ Mismatch/ Imbalance

Knowledge/ Experience from Classes

- Sampling
- Digital signal processing
- Amplitude/ Frequency Modulation
- Amplitude/ Frequency Demodulation
- Digital Filter Design

PROBLEM STATEMENT

The problem we are trying to solve is Simultaneous Call Transmission (SCT). An SCT event is when multiple aircraft communication devices are trying to transmit messages to the same person at the same time. This frequently causes interference between the signals due to their carrier frequencies being very similar which causes noise disruption or a loss of information. Our job is to develop an algorithm that can be implemented into a hardware device that would detect and alert users when multiple calls are being transmitted simultaneously.
Our algorithm does not need to correct the frequency error/ aliasing/ overlap, but only detect when it is happening, making the task a little simpler.

REQUIREMENTS & CONSTRAINTS

- Functional Requirements
    - The algorithm must alert the user when an SCT event has occurred.
        - An SCT alert must come within 1 second of the signals being transmitted.
    - The algorithm should be able to run on the hardware that it will be implemented on, meaning that it should be efficient enough for a small device.
    - Accuracy of the algorithm should be reasonably accurate
        - 70% accuracy for general scenarios.
        - 90% accuracy for specific environments (such as at airports).

- Non-functional Requirements
    - Training data must be diverse and random.
        - Training data must contain audio from pilots speaking different languages, at different volumes, at different rates, and with different dialects.
    - Documentation must be adequate for a second team in the future to be able to pick the project up with little hiccups.

- Constraints
    - Training data for the algorithm is simulated, as we do not have access to real data from radios.
        - MATLAB and Simulink must be used to simulate communication.
    - Communication parameters are not immediately known.
        - The distance between the transmitter and receiver varies.
        - Signal amplitude, frequency offset, and gain vary.
        - Time and length of SCT events vary.
        - Background noise, volume of noise, and amount of noise can vary.
    - Hardware must fit seamlessly into the cockpit or wing of a plane and an air traffic control tower.

- Total cost of the project cannot exceed $8,000, as this is the budget allocated by Collins Aerospace.

STANDARDS

IEEE 2755-2017: *IEEE Guide for Terms and Concepts in Intelligent Process Automation*

https://ieeexplore.ieee.org/document/8070671

This standard is called the "IEEE Guide for Terms and Concepts in Intelligent Process Automation." This was relevant to our project because our software automatically processes these signals without any human help. Our project is not just taking data and displaying it; it is learning from simulated data then processing new data to make a prediction.

IEEE 1139-2008: *Standard Definitions of Physical Quantities for Fundamental Frequency and Time Metrology - Random Instabilities*

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6581834

This link listed above takes you to a PDF of the IEEE Standard titled, "IEEE Standard Definitions of Physical Quantities for Fundamental Frequency and Time Metrology - Random Instabilities." It covers randomness and instabilities in signal processing. This was useful for our project because we needed to factor in randomness, such as noise, when designing our machine learning algorithm. The standard helps with understanding how to recognize it and how to predict it.

IEEE 1641-2010: *IEEE Standard for Signal and Test Definition*

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4410235

The standard above is known as the "Standard for Signal and Test Definition." Seeing as much of our training data originated from self-made simulation signals, a standard to define those signals mathematically proved very useful. Not only that, we can make the signal undergo to change its "viewed" characteristics.

SECURITY CONCERNS AND COUNTERMEASURES

The biggest vulnerability is our dataset not modeling real world radio audio data. This would result in an inaccurate model and when implemented in an actual radio system, would produce false positives and false negatives. We've done our best to mitigate this by using a diverse amount of training data and comparing the model predictions of different datasets to each other.

We also had to do many checks and tests at every stage in dataflow that our datasets were not being mis-shaped or misrepresented.

DESIGN

Design Context

Broader Context

Area	Description	Examples
Public health, safety, and welfare	Project affects people both indirectly and directly. Affects the aviation community (ATC and pilots) and the welfare of passengers.	Decreases safety risk of aircraft collision and/or runway excursions at airports.
Global, cultural, and social	All cultural groups may be affected. It would be a global influence within the aviation workplace. For all end and indirect users, this project values safety to people and aircraft which would reflect positively on the aims of all parties involved.	An addition of SCT event detection in radio receivers would not cause drastic change to systematic processes of ATCs. No cultural or global communities would be adversely affected by such an implementation.
Environmental	Under specific analysis, this project would have a front end environmental effect due to material demand of manufacturing the radio system. Unsure as to the hardware choice of filtering products and processor at this time.	Increasing use of non-recyclable materials in radio construction. Although minute, increase power demand of radio (higher energy use)
Economic	If software based with SCT detection, may drive radio prices higher. This can be dictated by FAA regulation in the future by requiring SCT detection in all towered airports.	Cost of production is expected to be low. Highest expense will be the processor required to run the algorithm in real time. Embedded software application so it may drive the cost of hardware down.

User Needs

ATC needs a system application that allows for him/her to be alerted to the event of an SCT. The ATC can then perform a SOP (Standard Operating Procedure) to prevent any runway mishaps. An ATC may only receive a single aircraft transmission or static tone due to the interference of two received transmissions.

Prior Work/Solutions

– Many pending patents exist. Various types exist with use of some form of signal demodulation and decimation. Most, if not all, prior designs incorporate some use of the Fast Fourier Transform (FFT) to derive the frequency spectrum of the signals as well as a systematic process of mixing and

down converting the RF signal to baseband frequency. The designs further incorporate logic sequences, error optimization, buffers, and additional filtering. Some designs incorporate deep learning algorithms and some do not. Of the algorithms that exist, there have been applications that utilize Gaussian Mixtures Model, fuzzy clustering, neural networks, support vector machines, etc.

– We are not following any prior work. We have dissuaded from relying on external designs to implement the SCT event.

– We can not accurately give pros or cons versus any other particular designs seeing as our algorithm has not been implemented quite yet and we will be unable to credit any hardware capabilities until that milestone is reached.

Technical Complexity

1. Digital signal processing (DSP) is a vast realm with many already modern applications. This project is of sufficient technical complexity as it requires not only Digital Signal Processing, but efficient use of a deep learning algorithm that is capable of providing real time event detection in an estimated 1 second.

2. Many of the engineering principles reside in the DSP portion of the project where an RF signal is modeled as a complex sinusoidal function. This complex sinusoidal input undergoes FFT transformation through use of its mathematical representation in programming. This is preprogrammed as a function in the machine learning algorithm platform we utilize. From there, the signal undergoes more signal manipulation such as mixing, filtering and decimation such that we are required to design the LPF according to specifications of the project and design the ADCs to produce the necessary sample input to our algorithm. All of these instances require an understanding of complex signals and their corresponding transformation functions. Once the basis of the process is established, we will further look into other processing techniques that can be used to increase processing time, reduce total samples required, or create looping mechanisms to promote efficiency. Furthermore, we must distinguish between mathematical model and real world variations considering we are working with RF signals that are subject to many distortions in its process such as Doppler Effect (objects moving at a distance and speed relative to each other), reflections (circulating signal within coaxial cable), or noise (acquired signal distortion due to hardware or transmission through air).

Design Exploration

Design Decisions

A key decision for our design was to implement the deep learning algorithm on TensorFlow-Keras with the plan to utilize a classification type network.

Another key decision of the design was to pursue IQ demodulation in our application as it allows for full analysis of any RF signal received, no matter how it was modulated to be transmitted.

Lastly, a key decision of the design was to not worry about the hardware and its capabilities while drafting the algorithm. The client suggested first to get a working algorithm, then once that is done we will fine tune it with hardware capability in mind.

Ideation

Our decision to use TensorFlow-Keras to create the machine learning algorithm came after looking at all of the available options and finding what we valued in the framework.

Theano looked appealing as it was a highly optimized matrix based machine learning tool. We didn't use this as it was difficult to set up and looked painfully above our heads as it allowed too much control over everything.

We also looked at Numpy quickly before seeing that it was even more barebones than Theano.

Matlab ML is built into Matlab so we could have used it there. Not many of us have enough experience with Matlab to delve into more complex parts of the language. We figured piping a file into a python script would be better.

Pytorch looked interesting but it didn't have the best debugging or visualization tools, which would help us greatly.

Using TensorFlow alone was also an option, but using the Keras library made it easier to use.

Decision-Making and Trade-Off

The process we used to identify pros and cons was to try and get a real example of what each option would be like to develop with. They all could have achieved the same outcome, however if we had a tough time learning or developing with an option, we would not choose it. So there weren't several factors in our decision making, it was just "we like this, let's use it."

Proposed Design

The beginning of our project entailed creating simulatory input signals within Simulink where an SCT event could be modeled by summing two RF signals together that are slightly offset in carrier frequency. This model utilized DSB-AM modulation and demodulation to generate the signals and analyze them. It just so happened that there was a more efficient means of undergoing the demodulation such that an IQ demodulation process is now being implemented. This model was also created in simulink, where the filters, local oscillator, and ADCs can be designed and implemented as well. With this model, we can simulate received signals to a receiver that are subjected to signal distortions and randomized in signal strength and frequency offsets so that the data to the machine learning algorithm is random in nature.

We plan to embed a matlab code that allows for random variable generations within our simulink model and run multiple iterations to acquire files to use as training data in our machine learning algorithm. The proposed idea is to create a random function generator in Matlab using trueRand() or using a seed that is based on real time. From there, the output of the simulink model can be sent to the Matlab Workspace or exported as a .csv file for implementation into Keras.

Design Visual and Description

This model is the beginning construct of an I-Q demodulator with the goal of taking a received RF signal into a receiver and implementing I-Q demodulation. A summed RF signal of two different frequency signals is the input to the I-Q demodulator. This is a modeled input to a RF signal with two transmitted signals at the receiver. From that point, the transmitted signal is multiplied by a LO (Local Oscillator) Frequency. The I phase of the received signal is multiplied by a sinusoid with no phase offset present, but the Q phase of the received signal is multiplied by a sinusoid with a 90 degree phase offset present. From there, the two channels each go through the exact same lowpass filter to acquire the certain frequencies we are interested in. Finally, the lowpass filter output is run through a decimation process of some sort to acquire digitized data that allows for analysis. Currently, we are pursuing how to go about implementing the representation of an ADC after the lowpass to get proper data samples. Currently, we have just used a quantizer for ease. With this digital data representing the I and Q channels, they can either be further processed using hardware or can be directly input into the SCT event detection algorithm.

Simulink Construct of I-Q Demodulation

The model will further be constructed to allow for random generated signals and signal distortion. The machine learning algorithm will then be trained with the randomized data under thousands of iterations to effectively train the model for a certain error rate. The algorithm will be implemented using TensorFlow-Keras, a common DSP neural network platform. From there, we know we will require a Fast Fourier Transform of the signal to analyze its frequency components and further decide peaks and differences within the frequency spectrum. It is unknown as of now what other data manipulation we may use.

Functionality

Our design will turn on a light in the hardware of a radio to tell the radio operator when simultaneous call transmission has been detected. We will have a hardware component built into the radio that will detect SCT by running the radio wave data through the ML algorithm after IQ Demodulation. It will try to detect multiple radio waves being combined in one output, which means simultaneous call transmission is happening. In the real world, when airplanes are calling the air control tower, the tower and airplanes will know that their transmission was inaudible or difficult to hear so they can try to call again.

The current design is just generating data to simulate radio transmissions so that we may create our own simultaneous call transmission events to generate massive amounts of data to train our machine learning algorithm on. So nothing functional is available yet.

Areas of Concern and Development

Concern

Some main concerns we have aside from the SCT device working properly, is will our algorithm detect an interfering signal within our desired time period of one second.

Our other main concern is the implementation of our training data. It would be widely easier to do with real world data, but we are confined to simulating the data within Simulink. This requires configuration of real world hardware scenarios and could be very difficult to model.

Development

The more accurate our training data is, the smoother/ more precise the algorithm will run, which should technically be able to detect the SCT event within that time.

We are currently working on modeling the training data and have a fairly clear objective. It is more so the learning curve associated with finding and implementing the block functions properly.

Technology Considerations

We have chosen to develop our signals in Matlab as the computer engineers are comfortable with the program. Our software engineers have chosen to use Python with Tensorflow and Keras. Our decision was after trying machine learning in Matlab and deciding that we should stick to a language we are more familiar with like Python. Our use of Python means that we will have to output data from matlab to a file and input it into our Python program. This is a just tradeoff. There are 2 major machine learning frameworks for Python, Pytorch and Tensorflow. We arbitrarily chose Tensorflow as for the depth we are using machine learning this will not affect us negatively. Keras is a library we use on top of Tensorflow to handle common machine learning modules.

Design Analysis

Our proposed design may or may not work. It is in theory correct and should accurately model radio transmissions from varying distances. We will use this to generate vast amounts of data for our machine learning model to learn from for its classification detection algorithm.

We have been constantly modifying and reiterating over this design this whole semester and are just about done. Now we will be able to modify and reiterate over how our machine learning algorithm works. We will try many different manipulations of the dataset to see what helps our model learn the best.

Design Plan

We design a very realistic simulation of radio waves and generate copious amounts of radio wave samples labeled ‘normal’ and ‘SCT’. We find the optimal way to feed in the most data possible into our Python script. In Python, we continually refine and modify our data and record what our machine learning model learns best with. We iterate through designs on the neural network layer to see if efficiency may be gained there. Data about how each ML design performs is gathered to find the best algorithm for the common environment.

CHANGES SINCE 491

- Simulation
    - Simulink simulation was modified to take in multiple audio files to act as interfering signals. Original model only had two audio sources (One message signal; one interfering signal)
    - Swapped audio sources from clean speech from Gilbert Gottfied and Josh Peck to realistic air traffic control audio from LAX Airport.
    - Created a MATLAB script for extracting the time series signal data produced from the Simulink model into a readable matrix variable.
        - Script also creates label matrixes for labeling the data.
    - Created resampling MATLAB script that up-samples audio to higher sampling rates (ex: 22.5 kHz to 48 kHz sampling rate).

- Machine Learning
    - Code Modularization
        - Created separate modules for the different aspects of pre-processing and training.
    - TensorBoard Integration
        - Allows us to better visualize the metrics that Keras is outputting instead of just looking at numbers on the command line.
    - Configuration
        - Added a configuration file that allows us to change various training parameters. This allowed us to easily switch them between training iterations instead of manually changing code.
        - A very important aspect of this is being able to specify the directory that the data is located in. This allows users to be able to put data of their own in place of our simulated data and maintain the functionality of the algorithm.
    - Documentation
        - Better documented our project (code, README, website) to allow both our client and the future team to easily ingest the functionality.
    - Saving Models
        - After a training iteration completes, a model is saved to the working directory of the user (in the Keras format).
    - Predicting
        - Added a script that allows the user to take a saved model, and predict on a given dataset.

- Overall
    - Began to use CyBox for storage, as our data exceeded the maximum allowable storage size in GitLab.
    - Consulted a broader array of outside sources and experts for advising.

IMPLEMENTATION

Before we started, we knew that this project was going to be passed on to another team. We wanted to make sure that if the model we handed off did not work, it would not be challenging for them to train a new one. Users of this project are able to take any directory of data (.csv or .bin) and configure the algorithm to train a model on that data. This to us is the most powerful part of our implementation.

The implementation of this design was done in Python because it provides the most powerful tools when it comes to data manipulation and machine learning frameworks. Our training was done using the TensorFlow Keras framework: a high-level API that allows us to call its methods without having a deep understanding of what is going on underneath. This allowed us to develop faster because we knew nothing about machine learning before this.

Some other important Python tools that we used to implement this design were NumPy and MatPlotLib. NumPy is a tool that allows us to work more closely with arrays. It is widely used in machine learning because of its ability to store data in any shape that the user specifies. Key methods were the reshape() method, which reshapes an array to the exact dimension(s) specified by the user. This was how we took our 1D array of simulation data and turned it into “windows” that were the input shape of the Keras network. MatPlotLib is a framework that can take those arrays and plot them to a graph. Although there are much more complicated uses for this, we just used it to output predictions vs ground truth. This gives users a visual on just how well the outputted model is performing.

TESTING

In order to maintain proper functionality of our design, it is important that we test at many stages in the process of development. However, we shall only be testing the software side this first year, as the hardware component and implementation will happen during Phase Two (Year Two) of this project.

Unit Testing

Software Testing

A unit of software that will require testing is the Data Shaper. Since this is the primary component of pre-processing, we had to make sure that it was doing exactly what we wanted. If it fails, our

model will not perform as we expect it to. To test this, we did unit testing on three things: the sliding window, the randomization, and the final shape.

The other unit of software that will require testing is the simulation of data that will be used to train the algorithm. The Communications Division tested many points in their I-Q Demodulation using the Frequency Spectrum Analyzer Block in Simulink.

Hardware Testing

As stated above, we will not be implementing hardware for this phase of the project, therefore there is no testing to be done at this point in time.

Integration Testing

We have the Communication Division creating a radio wave simulator to generate large amounts of data. The Machine Learning Division is building the machine learning model. We need to integrate the generated radio data into our machine learning algorithm in a simple and easy to understand format. To integrate it, we will need to format the data to be fed into the neural network by determining the input layer size. To test it, we will make sure that the data is in the correct format, and that no input layer of the network will be repeated.

The Communications Division will also be testing the effects of real-world natural interference and discrepancies on their simulation. Some examples of this include adding in white noise and recreating signal reflection and doppler effect. In order to create an accurate design that can be implemented in a real-life scenario, it is highly critical that their simulation test results will be accurate to hand over to the Machine Learning Division for training the algorithm.

Once again, they will be tested using Spectrum Analyzer Blocks in Simulink. The script that formats the data is done in Matlab, and has been tested to ensure that it properly formats all simulated data. Since this script has been tested, all data that lives in CyBox will be ready to use by the algorithm.

Regression Testing

Regression testing was done using TensorBoard. If any configuration changes made the model perform worse, we were able to see a nice graph on the UI and delete that model. This ensures that only our best model is stored in the CyBox and handed off to the next team.

Results

The results of our testing were acceptable. Since we did not implement the actual machine learning concepts (Keras did), the bulk of our testing was done on the pre-processing aspect. The combination of unit testing, and manual testing through development ensured that we were feeding the correct data into the Keras training method.

The simulation results were also acceptable. We saw no issues using the Spectrum Analyzers in Simulink, which tells us that the data being simulated was on par with the design.

APPENDIX I: OPERATION MANUAL

Data Generation Method 1

Use the Matlab Simulink .slx file located in the GitLab /simulink folder to generate a .mat file from two or more input files. Many parameters may be set such as the SCT interval, gain, and frequency offset of each input file. Run extract.m located in /tools in GitLab on the generated .mat file to create the .csv file.

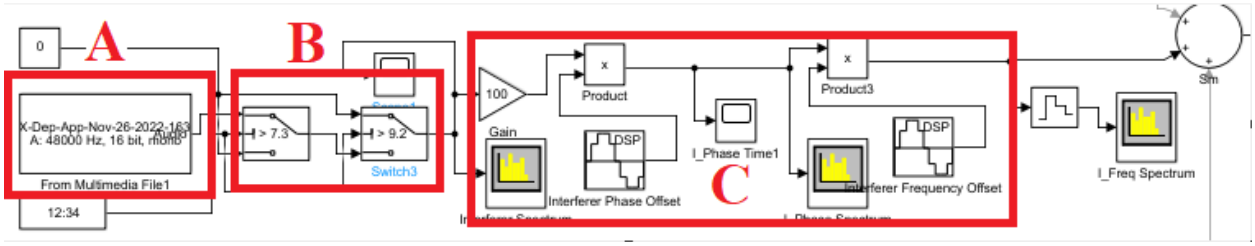
The automated data generation process using Matlab Simulink Coder is much more efficient, but much more complex. Matlab Simulink Coder must be used to generate the code on the computer that it will be run on. The generated C code must then be modified to read in a .txt file containing input parameters. These parameters will set the variables: input file 1, input file 2, output dir, gain 1, gain 2, frequency offset 1, frequency offset 2. Check out the code in /tools/exampleCSimulation to see how we modified the code to automate it. generateTrainingDataC.py can then be used to automatically run the C program, it may need tuned up or down based on your computers processing power and memory size.

This was run on MATLAB R2022a.

Data Generation Method 2

This data generation entails using Simulink to introduce hand picked audio files and apply varying signal manipulations at Complex Baseband. Signal manipulations can include frequency & phase offset, gain, time of interference, and additive noise.

Figure Complex Baseband



1.) Begin by accessing the simulation file within the GitLab/simulink folder. Use the file version that you currently have Matlab installed under.
2.) Within the Simulink file, choose your audio input using the From Multimedia File for each interferer as shown in Block A of Figure Complex Baseband shown above.
    a.) All prior simulations utilized 10 second .wav files. These .wav files were converted from .mp3 files and upsampled to step 2b) frequency using Upsampling.m within

the GitLab/Tools. The .mp3 files used were taken from LiveATC.net and downloaded as one file averaging around 30 minutes in length.

i.) Using a Sound Editor (WavePad), the 30 minute mp3 file was split into equal length files using Edit→Split→Split into Equal Parts→Duration→Choose length required→Save to appropriate folder as .wav files→Split

ii.) This will create, on average, around 190 ten second .wav files to be used as inputs for Simulink

b.) All prior simulations and training data was generated using a sampling rate of 48 kHz. This was decided as channel spacing for aviation AM is spaced at 25 kHz for and would result in no signal data loss for speech. It is expected that the sampling rate can be reduced to a much lower frequency to convey speech.

c.) All prior simulations and training data utilized .wav files.

3.) For each interferer, choose the required time of interference utilizing the switch blocks shown in Block B of Figure Complex Baseband shown above.

a.) For the top signal, deemed message signal, the switch blocks operate such that the audio file will play from the start of simulation to the value of the first block. After the first block, the audio output is 0 until it reaches the value of the second switch block in which it will continue playing again.

i.) Ex: If Switch 1 = 4 and Switch 2 = 6 and total simulation time = 10 seconds. The audio input file will play from 0-4 seconds and 6-10 seconds.

b.) For every other interfering signal, the switch blocks operate opposite to the message signal. The audio file will only play in between the values of the two switch blocks.

i.) Ex: If Switch 1 = 4 and Switch 2 =6 and total simulation time = 10 seconds. The audio input file will play from 4 - 6 seconds.

c.) Overlap segments of audio of interfering signals with the message signal to create instances of SCT (two transmissions reaching the receiver).

i.) These will be the values in time that get labeled as 1 later in time

4.) For each interferer, choose the desired gain, frequency offset, and phase offset utilizing the blocks shown in Block C of Figure Complex Baseband shown above.

a.) Gain merely multiplies the amplitude of the audio input. Can be considered as varying the strength of the signal

b.) Phase Offset incorporates the desired phase offset of the signal.

c.) Frequency Offset incorporates the desired frequency offset of the signal.

d.) For 4a), 4b), and 4c), the values were held as constant over the entire length of the simulation when generating data. We believe these values now need to be varying under time such that they are random and change with respect to simulation time.

i.) Estimated bounds for phase offset fall between 0 and 2π

ii.) Estimated bounds for frequency offset are unknown. Around 1000-2000 Hz offset results in complete distortion of the speech within the audio. We could not find sufficient enough research to conclude practical frequency

offsets present within modern receivers. It is possible that linearly time varying phase error results in a constant frequency shift. Couldn't find research to back it up.

e.) For phase and frequency offset, it is essential to utilize the DSP sine wave blocks as Output Complex→Complex. Our training data utilized Sample Mode→Discrete for all simulations. We did not vary the Sample Mode or Computation Method.

5.) Run the simulation for the desired time. Simulation time was set to match the length of time as the audio inputs. If desired, audio inputs can be split into different lengths and the simulation time can be adjusted respectively.
    a.) This will generate a .MAT file that is stored in the Matlab workspace
6.) OPEN the GEN_CSV.m or Extract.m matlab code from the GitLab/Tools. The code itself is commented to understand the process of how to label the data, but the below steps highlight the general process.
    a.) Be mindful of file naming conventions when loading and writing files at the beginning of the code.
    b.) The general process of each matlab code is to allocate a number of samples of either ones or zeros in respect the sampling time and the time in which a zero or one is desired.
        i.) One is considered to be an SCT event, Zero is considered to be NON-SCT
    c.) It is a manual process of adjusting the time in simulation where a one or zero is desired and then creating a column vector of ones and zeros that correspond to the SCT events in the simulation. This column vector is then concatenated to the simulation data to create the desired labeled data.

Notes of Importance:

- For any blocks with a sampling frequency component. Ensure all blocks utilize the same value of sampling rate, or error will occur.
- Our model does not sum complex noise into the model as we did not want to over-complicate the model. We wanted to be able to accurately predict SCT events on simpler data before moving to more advanced scenarios. Noise will eventually need implementation.
- Matlab code has options to start and stop Simulink simulations. It may prove useful to attempt automation of generating training data through Matlab code rather than manual adjustment within Simulink. This will also allow for variables to be created for gain, phase offset and frequency offset such that they can be randomized and dependent upon time.
- We only generated time series data and exported as inputs the Machine Learning Algorithm. Results may show better for other signal manipulations such as FFT, Phase & Magnitude, Cepstrum, etc.

Training a Model

First, refer to the README located in the GitLab repository. This document explains how to set up the environment in which the algorithm will operate in. Once the environment is set up, navigate to the 'default_config.yaml' file in the 'configs' folder, which is located in your working directory. You can keep this file as default, or edit it as needed. Documentation for each parameter is commented into this file. Once you have set your parameters, run the command `python train.py`. The command line will output visuals for progress of the pre-processing modules, then a model will start training. IGNORE THE WARNINGS FROM TENSORFLOW. These warnings are due to the fact that it is running on a CPU, and not a GPU.

You can more easily visualize the training results using TensorBoard by opening a separate terminal and running the command `tensorboard --logdir logs/fit`. This will start a UI that you can open by clicking on the link that this command outputs. Once again, you can ignore the warnings from TensorFlow. In order to most accurately graph the logs, turn the smoothing bar all the way to 0 and uncheck the "Ignore outliers in chart scaling" button on the top left of the UI.

```
(venv) sjahnke@DESKTOP-HFVK9IV:~/sddec22-13$ python train.py
```

train command

```
> Loading 5 files
100%|

> Sliding window over 5 input files
100%|

::==< Creating optimal window ratio from 3824 windows >==::
Window label ratio: 41.4%       ones/zeros: 1586/2238
Window label ratio: 50.0%       ones/zeros: 1586/1586
Removed 652 0 labelled windows

Randomizing 3172 windows
Setting aside 15.0%, 475/3172 windows for validation

Validation Windows: data: (475, 1024), labels: (475,)
Train Windows: data: (2697, 1024), labels: (2697,)
::=========================================================::

Model name:  default_config_partition_7_20221207-114730
```

Pre-processing progress

```
dense_6 (Dense)             (None, 32)                2080

dense_7 (Dense)             (None, 1)                 33

=================================================================
Total params: 1,007,297
Trainable params: 1,007,297
Non-trainable params: 0

_________________________________________________________________
Epoch 1/100
489/674 [====================>.........] - ETA: 1s - loss: 0.6931 - binary_accuracy: 0.4990^CTraceback (most recent call last):
```

Keras training

```
NOTE: Using experimental fast data loading logic. To disable, pass
    "--load_fast=false" and report issues on GitHub. More details:
    https://github.com/tensorflow/tensorboard/issues/4784

Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.8.0 at http://localhost:6006/ (Press CTRL+C to quit)
```

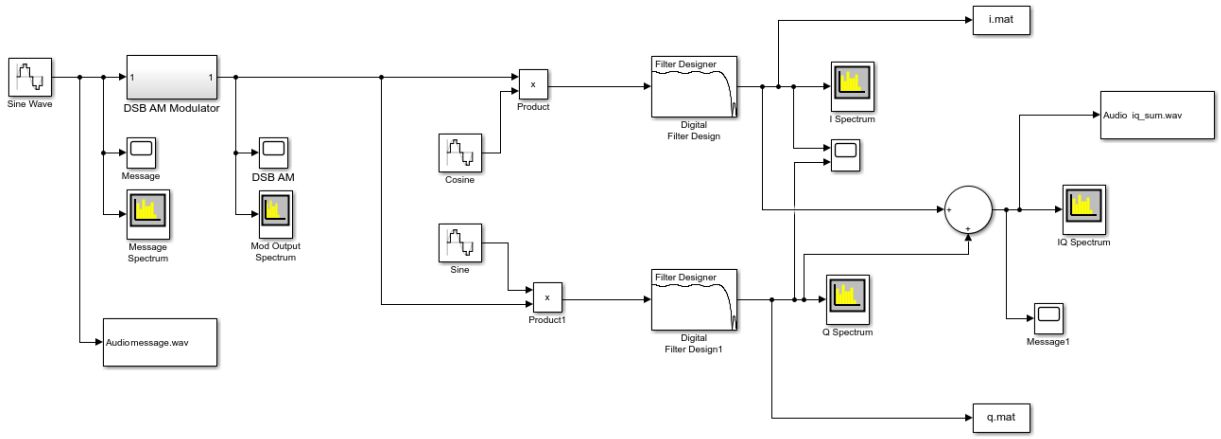
Tensorboard output

Predicting

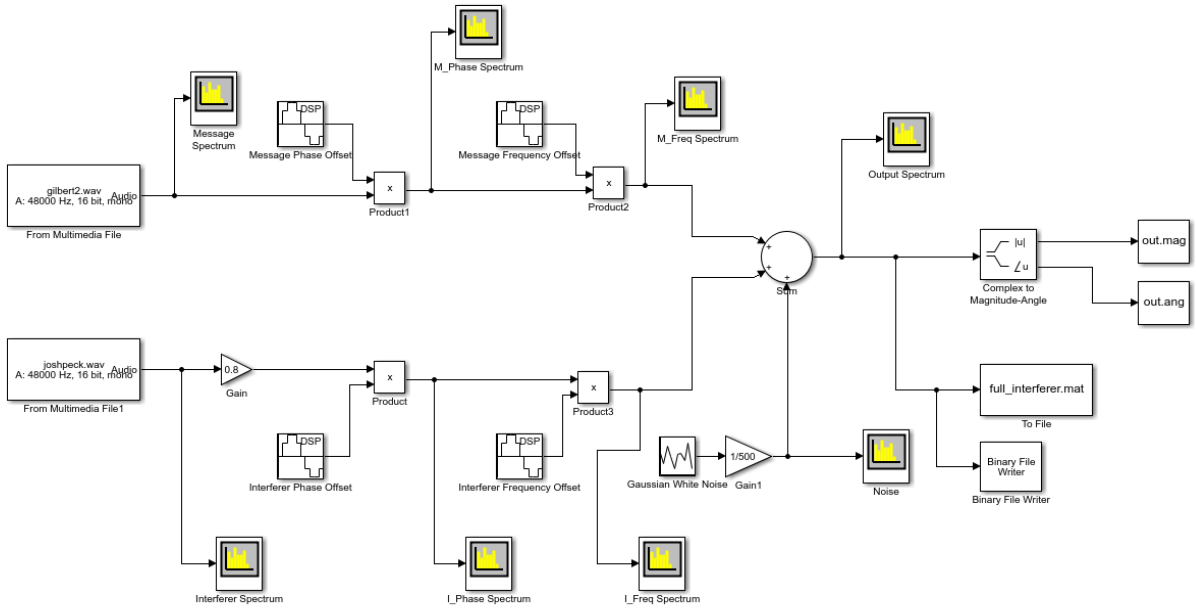
Assuming your environment is already setup, and you have a saved Keras model in your working directory, simply run the command `python predict.py <model directory> <data> (where <model directory> is the path to the saved model and <data> is the path to the .csv or .bin file you want to predict on).

APPENDIX II: ALTERNATIVE DESIGNS

SIMULATION: AM Simulink Model (Changed halfway through Semester 1, as client changed requirement from amplitude modulation to DC baseband modulation).



SIMULATION: Initial Complex DC Baseband Simulink Model



SIMULATION: Phase and Frequency offsets implemented as constant rather than time varying with simulation
SIMULATION: Audio inputs were random, not ATC specific
SIMULATION: Complex Noise never implemented due to attempts of training models on simpler simulations first.

APPENDIX III: OTHER CONSIDERATIONS

With the radio receiver hardware being unknown, certain items such as modulation and demodulation techniques, filters, or other receiver characteristics were unknown. This resulted in many of the simulation data items at complex baseband being assumed through research. This could ultimately result in our generated data being incorrect or too far extreme in estimation. We also only generated data at a sampling rate of 48 kHz. We considered this to be adequate to encompass anything present within the 25 kHz channel for voice range. Lastly, the simulation only generated training data in the form of time series equivalent data. Other manipulations could be done before exporting as inputs to the Machine Learning algorithm. Those manipulations being things such as Phase & Magnitude, Cepstrum, FFT, etc.

Another consideration on the simulation team was sending just the carrier through our simulation. This was something that Tyler had brought up, but we were unsure of how to implement this at DC baseband. We were trying to figure out the math behind this, but we ran out of time. However, later on, we did determine a way to implement it, but we were already on the final phase of our project.