Spring 2023

Folding Collision Probability in the Nyquist Folding Receiver

Hunter J. D. Miller

Folding Collision Probability in the Nyquist Folding Receiver

Hunter J. D. Miller

Honors College, Texas A&M University – Commerce

Honors Thesis Proposal

Table of Contents

Table of Figures

Abstract

As the radio-frequency spectrum becomes more crowded, there is a growing need for signal processing technology that can instantaneously interpret an ultra-wideband range of frequencies. Existing conventional systems, however, are not the most efficient solution due to the processing capability limitations of current analog-to-digital converter technology. Specifically, with regard to radars, traditional receivers are required to process large amounts of unused data, wasting time and energy. This project highlights a newer analog-to-information receiver that utilizes compressive sensing which is seeking to overcome these challenges. The innovative technology being evaluated allows users to sample far below the standard Nyquist/Shannon sample rates while still being able to detect and reconstruct the original received signal. The focus of this project is the process of calculating the probability of folding collisions using a custom generated MATLAB simulation, thus allowing for a better understanding and interpretation of the data received by the Nyquist Folding Receiver or other similar signal processing devices.

Introduction

There has been an increased demand for specialized receivers with instant broad bandwidth being utilized for specialties such as compliance testing, signals intelligence, and cognitive radio. The need for such devices has stemmed from the growing use of the radio frequency spectrum (Martin, 2018). Conventional wideband receivers typically have large data rates which make it difficult to compute and stow. The traditional process of achieving high bandwidth implemented by these receivers uses a large quantity of analog-to-digital converters (ADC) which are not economical or energy efficient.

The Nyquist/Shannon Theorem states that signals must be sampled at a frequency of at least twice that of the bandwidth of the signal to recover the full signal. The Nyquist Folding Receiver (NYFR), however, uses technology that allows for a much lower sampling rate which reduces the computational complexity, energy consumption, and cost of the system (equipment, processing, and storage). The NYFR is also capable of applying different Compressive Sensing (CS) algorithms to extract information. By implementing CS technology, the users can sample at far below the Nyquist/Shannon rates, only having to maintain an equipment operation speed slightly higher than the sampling clock while simultaneously using only one receiving channel. In addition to using relatively low-cost parts, the NYFR's structured sensing matrix allows for more accurate signal detection with iterative algorithms. Other benefits include constant bandwidth monitoring without the need for sweeping, as well as low computation time which allows users to view results faster. Using the NYFR's instantaneous bandwidth capabilities, along with refining the algorithmic CS methods, the NYFR shows itself to be an auspicious piece of equipment for this type of detection and information extraction. With additional research and time, it can be a very promising technology using low-cost and efficient systems.

The goal of a Capstone Project conducted in correlation with this project is to create a functioning Nyquist Folding Receiver. To accomplish this, the group must evaluate donated prototype boards as well as integrate other parts needed to create the physical receiver. This will be a challenging endeavor, because many of the components are used, donated parts that are not necessarily configured to integrate with each other. To interpret whether the receiver is working correctly, the project group must learn and understand the basics of CS processing. With this technology, the team should be able to significantly under-sample radio frequency (RF) signals while preserving the original signal's frequencies. The receiver sinusoidally modulates the sampling clock allowing us to maintain and recover the original data. To execute this project, the group must be able to evaluate and apply their knowledge of amplifier circuits and filters as well as learn new concepts of signal processing. Another facet of the project will be for the team to understand the functions of and implement a universal software radio peripheral (USRP) using GNU Radio software. The USRP will be used to generate a clock and modulate the sampling frequency. The researchers will also expand their knowledge of new techniques like coaxial cable termination as well as using signal and spectrum analyzers to process and analyze signals. The simplified block diagram of the NYFR, as well as the schematic for the Hyperlabs board being used, are presented in Figure 1 and Figure 2 below.

This thesis, though related to the project above, will focus on performance analysis - in particular, analyzing the probability of folding collisions. This will be done by creating a MATLAB simulation to calculate the probability of folding collisions which is a concept that pertains to the NYFR. The time component of this problem is assumed to be a Poisson Arrival Process while the folded frequency components are assumed to be uniformly distributed. This simulation will be a nice accompaniment to the NYFR, because it will help people understand

this potential problem better as well as how often it might occur. This will create a more complete picture of the data collected by the NYFR and support researchers with ways to reduce the opportunities for folding collisions to occur.

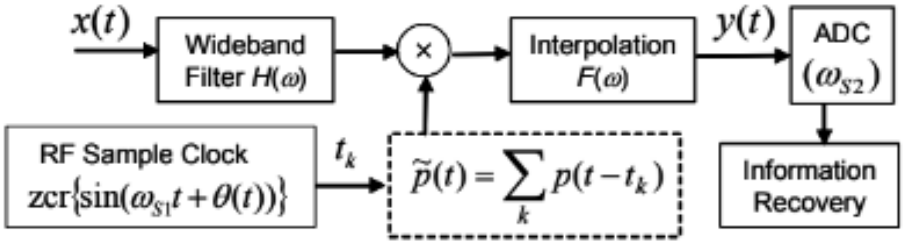


Figure 1: Block diagram describing the components used in the NYFR (Fudge, et al., 2008)

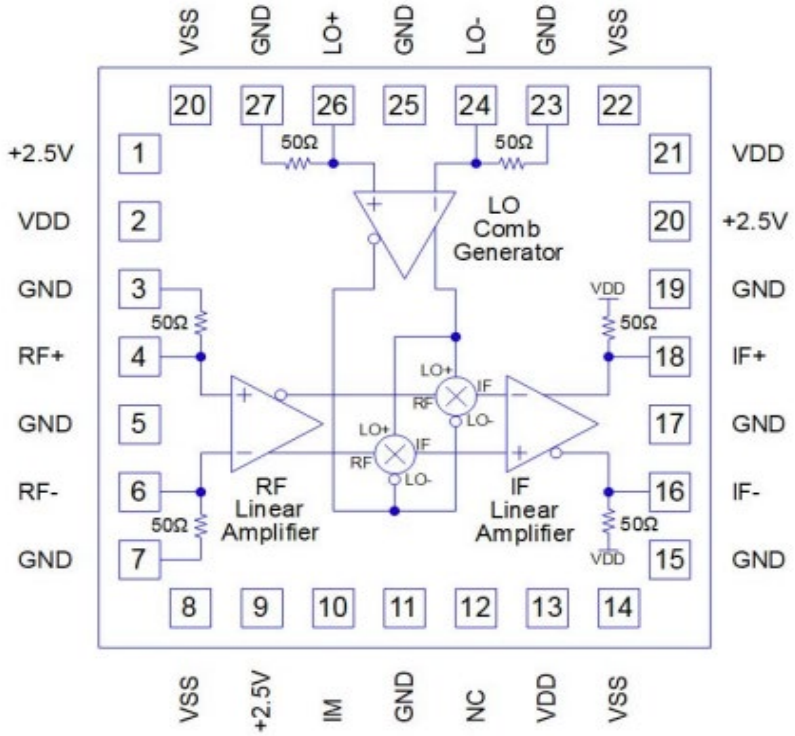


Figure 2: Schematic of Hyperlabs Evaluation Board HL933 **(Hyperlabs, 2022)**

Literature Review

Compressive Sensing

To understand the rest of this project, one must first understand compressive sensing (CS). This complex concept is the basis for the NYFR over which this project is based upon. In simple terms, CS makes it possible to capture a signal by sampling at a frequency much lower than twice the signal bandwidth that is typically required by the Nyquist-Shannon sampling theorem. Despite the advantages of this concept, there are some constraints. For instance, for CS to work effectively, the signal must be both compressible and in a sparse signal environment, which means that most of the sampled bandwidth (i.e., radar environment) is not being used. If these two conditions are met, then the signal can be compressed by eliminating the unnecessary information and maintaining only the data needed to reconstruct the signal. Ignoring the large amounts of unused data allows for a reduction in the computational complexity needed to capture and process the signals.

Nyquist Folding Receiver

Although the signal processing concepts used by the NYFR have been around since the beginning of the 21st century, the practical implementation of these concepts into a functional device did not exist until the physical creation of the NYFR was brought to life by Fudge and those he worked with (Fudge, et al., 2008). This incredible CS architecture receiver has since sparked many publications on different applications and modifications of the NYFR. Many of the articles about CS and the NYFR, such as the one by (Li & Chen, 2018), are theory based. In contrast, this project will focus on creating a software simulation to calculate and process data related to the receiver. This will partially be done using the articles by (Fudge, et al., 2008) and (Maleh, Fudge, Boyle, & Pace, 2012) as a guide. These papers were written by the original creators of the NYFR and provide valuable insight into what this complex piece of technology

can accomplish. In addition, these works provide a great overview of what happens inside the NYFR while helping readers understand the folding process. These articles also utilize visual aids to assist researchers in understanding what the data received should look like. This project is supervised and advised by Dr. Gerald Fudge who is the leading expert on the topic. Therefore, he will be an invaluable resource for conceptual questions and resolution of any problems that may be encountered.

The NYFR is a CS, analog-to-information receiver. It is capable of capturing signals from around 3 GHz all the way to 20 Hz and compressing them all down onto a 500 Hz Nyquist zone decreasing the size by around 98% (Maleh, Fudge, Boyle, & Pace, 2012). By compressing this ultra-wideband range of frequencies into one small band, the sampling frequency required can be reduced from around 40 GHz down to a mild 1 GHz. "The NYFR accomplishes this by multiplying the input signal by a pulse train (series of impulses), effectively sampling it at the pulse train's clock rate (PTCR). This has the effect of 'folding' the entirety of the original signal like an accordion into a much smaller band with the bandwidth dependent on the PTCR" (Pearson, 2023). Figure 3 below helps to illustrate this folding concept.

Although the folding process is effective for reducing the computational complexity of this process, it generates some new challenges. For instance, it is impossible to tell what the original frequencies of these signals are by just viewing their folded frequencies, because we don't know which Nyquist zone the signals originated from. To find the frequency of the original signal the PTCR must be modulated. The level of modulation is determined by the Nyquist zone in which the signal originated. Higher frequency signals will reside in a higher number Nyquist zone so they would have more induced modulation than lower frequency signals. This can be shown as the M factor in the spectrogram of Figure 4 below.

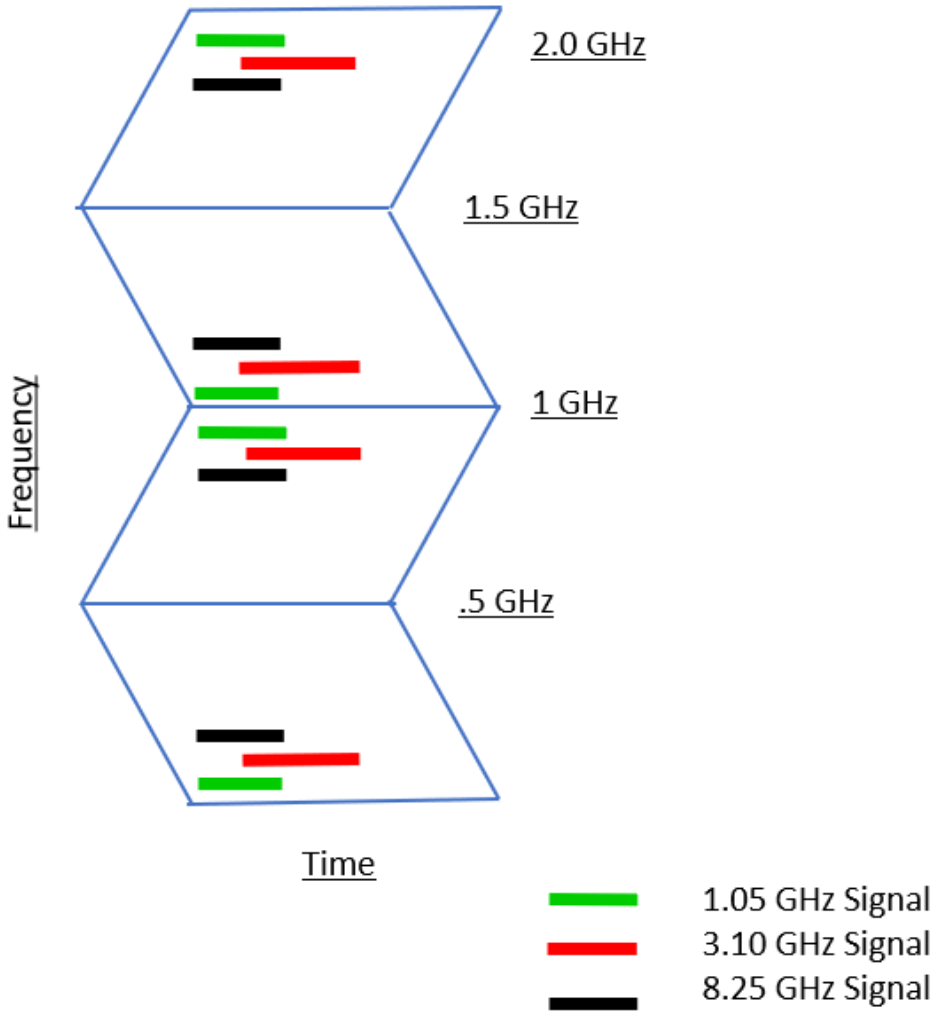


Figure 3: NYFR Folding Diagram

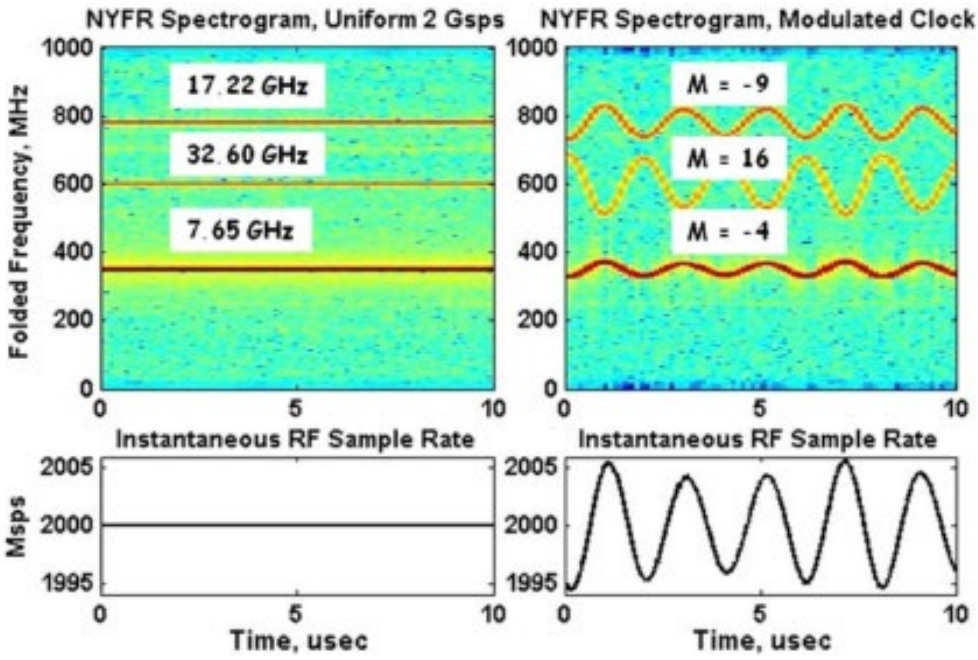


Figure 4: NYFR PTCR Modulation **(Maleh, Fudge, Boyle, & Pace, 2012)**

This induced modulation allows for the complete recreation of the original RF signals which allows for the reconstruction of the vital data needed to analyze these signals. The estimation of the RF frequencies of signals with a constant frequency is extremely fast and simple. This can be done by just measuring the center frequency, the sign, and modulation width of the signal showing the computationally simple advantages of the NYFR.

Folding Collisions

One rare but challenging issue associated with the NYFR is the possibility of folding collisions. This occurs when incoming signals overlap in both the time and frequency domain. This means that the two signal pulses arrive at the receiver at the same time, or at least overlap, giving the illusion that there is only one pulse. They also are folded to the same frequency or close to the same frequency giving the illusion of being one pulse. An example of this

phenomenon can be seen below as the pulse of the 8.11 GHz signal and the pulse of the 4.10 GHz signal overlap in Figure 5 below.

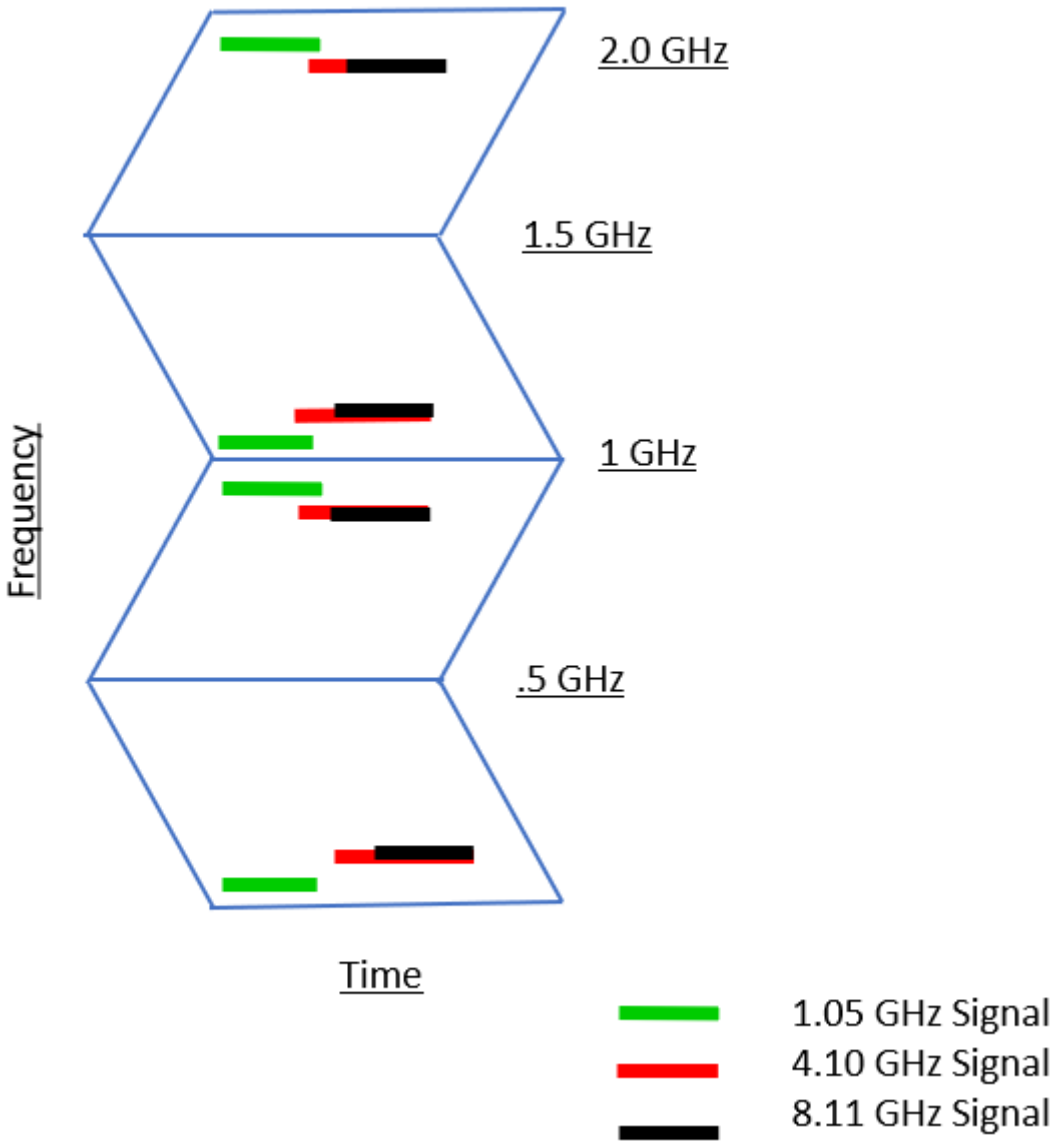
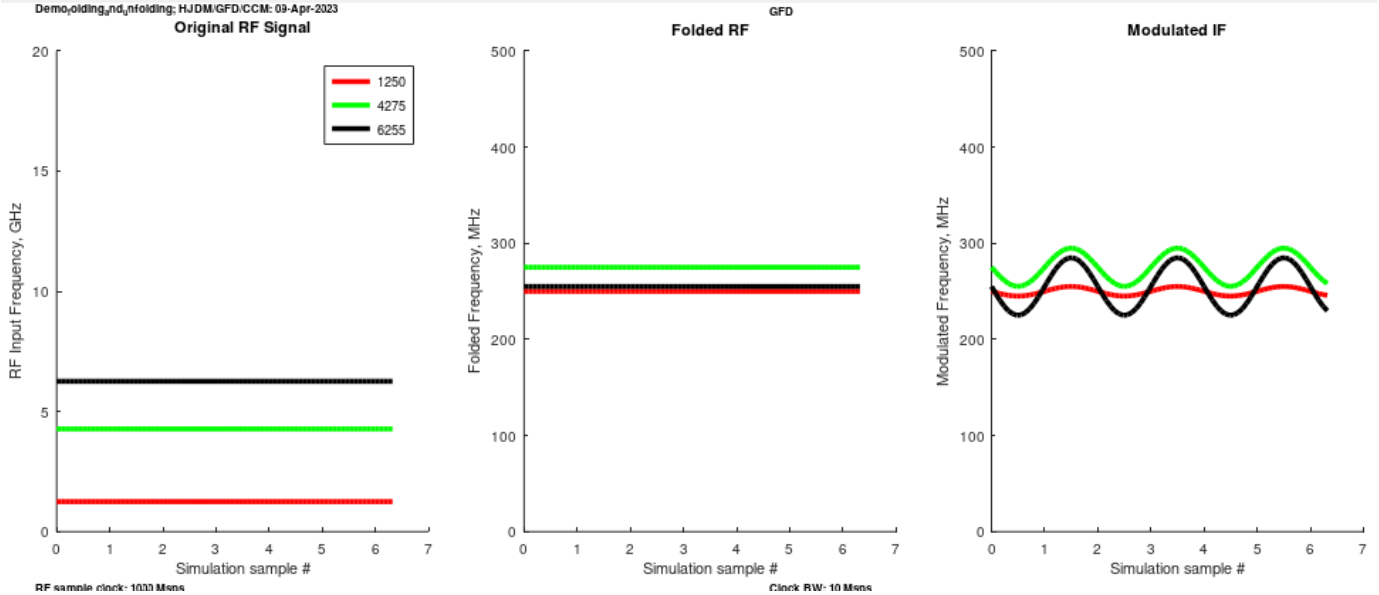


Figure 5: NYFR Folding Diagram – with folding collisions.

To be considered a folding collision the signals may not need to be the same frequency. If they are close in frequency, they may not overlap in straight folding but could overlap once modulated by the PTCR. In this situation users might be able to tell that there are two signals

being received but probably cannot calculate each signal's modulation index. This would lead to a loss of data because the original signal RF frequencies would not be able to be determined.



This situation can be seen in the figure below.

Figure 6: Folding Collision with Modulation - Frequency Coincident

Monte Carlo Simulation

A Monte Carlo simulation is a very common tool used in statistical analysis. "Monte Carlo simulation uses random sampling and statistical modeling to estimate mathematical functions and mimic the operations of complex systems" (Harrison, 2010). Monte Carlo simulations are generally used to run many simulations with random variables in order to get the probability of a particular outcome. For example, rolling two dice many times to determine the probability of rolling two ones. Generally, as more simulations are run the calculated probability gets closer to the theoretical probability. That is a simple example, but the concept remains the same. Part of the MATLAB code used in this project is a Monte Carlo simulation. It is used to run many simulations with a random number set to determine the average number of time collisions based on the number of radars in the environment.

Methodology

Simulation Process

As stated before, the main focus of this project is evaluating the possibility of folding collisions that sometimes occur between incoming signals when using the NYFR. The simulation for this project will be written and executed in MATLAB, and it will be used to calculate the probability of folding collisions associated with the NYFR. To be considered a folding collision, the signals must overlap in both their time and folded frequency components. To accomplish this, researchers will approach the time domain collision calculations as a Poisson Arrival Process. This in essence means that the pattern of the arrival of pulses is seemingly random in a given time interval, yet they appear to arrive at a specific rate. Assuming that the folding collisions have a Poisson distribution will allow for the use of the equation shown below for this probability calculation. Then the folded frequency components will be treated as being uniformly distributed within a specific range of frequencies. I will have the simulation generate random frequency pulses within this range. I will also vary the number of radars in the environment to see how that affects the probability of collisions. Then I will take the discrete Fourier transform (DFT) of the pulses to see the different frequencies in the system.

Equation 1: Poisson Distribution

$$g(k,\lambda) = P(Y = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

k is the number of successes

λ is the given rate

e is Euler's number: e = 2.71828…

$k!$ is the factorial of k

To approach the problem in this manner, some assumptions will be made for a few of the variables. This will make the results more repeatable, and the analysis of the data cleaner. For example, one might assume that the pulse width (PW) is 1 microsecond, that the duty cycle is 1 out of 1000, and that there is a pulse repetition interval (PRI) of 1 millisecond. Then a matrix can be generated to represent the duty cycle with one "1" and nine-hundred and ninety-nine "0" s. If time permits, some of the assumptions could be tossed out, varying the PW, PRI, and duty cycles of the radars. This will make the simulation more complex and realistic. Furthermore, this will be a great foundation from which to build future testing.

Data Analysis

Data gathered from this simulation will then be analyzed in MATLAB to find trends from the information. As the need for this technology continues to grow, this data could help future researchers better understand the technology and the data they receive. Folding collisions are a fairly new issue that has yet to be explored and examined in depth. In conclusion, the goal of this effort is for this project to help spark new research in this specific area of signal processing, leading to future solutions to folding collisions.

Results and Analysis

For a folding collision to occur with the NYFR at least two pulses must be received at the same time and must fold to close to the same frequency. The goal of this project is to determine the likelihood or probability of this occurring based on a few given parameters. To make this determination, multiple different MATLAB codes were written and some work in conjunction with each other. Splitting up this process into separate codes, such as one code to generate the calculations and another to plot the data calculated, allows the user more freedom to manipulate and analyze the data. This improves the overall experience as well as the efficiency of data analysis and troubleshooting of problems.

Program Breakdown

To start off this project, the simulations for time and frequency coincidence were done separately but simultaneously. However, the project advisor Dr. Fudge suggested that the time coincident simulation be focused on given the time constraints for this thesis. Dr. Fudge believed it to be fairly easy to incorporate the frequency component into the probability calculation for the time coincident. Therefore, the time coincident simulation was split into two separate MATLAB programs. One was used to complete the calculations and obtain the needed data while the other is used to calculate the probability and plot the different data sets.

The first and most complex MATLAB script that will be discussed is the "time_coincident_sim". This code contains all of the calculations except the overall probability calculation. Pictured below is the first section of code that is mostly for organizational purposes. It is used to classify the code, state the purpose and usages of the code, show who wrote it, and when it was written.

```
% script  time_coincident_sim
% purpose Generate seperate radar pulses w/ given period,duty cycle
%           and detect collisions (>=2 simultaneous pulses)
% usage     To generate data needed to calculate probability of folding collisions in the NYFR
% notes
% author  Hunter Miller
% date      04/02/2023
```

Figure 7: MATLAB Code Characterization

As you can see in the above figure this section of the code gives users valuable information to help them understand the code and know how to use it properly. This information can actually be called through the command window so that a useful, quick overview of the program can be seen easily. This is also extremely important because this project is meant to be expanded on by other students and Dr. Fudge in the future. This information could be vital in helping them understand this code in order to expand on its complexity.

The next section of the code is called the common initialization phase. This part of the code is used to set up the information needed when plotting the data from the simulation. This goes in almost every MATLAB program we write. It may or may not be used, but it keeps things organized and labeled properly should you need to plot something from this code.

```
% ================== COMMON INITIALIZATION ======================
programName_c = mfilename;              % char string of this script name
msg1_c = [programName_c,': ', date];    % program name + date plot generated

msg3_c = 'Hunter Miller';                 % name goes here (adapt for particular work requirements)

if ~exist('figNum','var')
    figNum = 1;                         % Set figNum = 1 unless already exists in workspace from earlier plots
end

plotNotes_h                             % script to set up plotting definitions
```

Figure 8: Common Initialization - For Plotting Purposes

This section just keeps things nice and organized and keeps the figure from being overwritten by incrementing the “figNum” variable. It displays the programmers name and the data on the

Figure above the plots. It also calls the "plotNotes_h" that further optimizes the set up for plotting data such as the spacing and size of plots.

The next part of the code is the Initialization phase. This phase is used to set up any variables and values needed before calculations are being done. These are usually the main variable values you want to stay consistent throughout the entirety of the simulation. The figure below shows that section.

```
% ======================== INITIALIZATION ==================

numRadar = 40; %num of signals, all identical (1us PW)
                  %varying in start time
period = 1000; %1000us = 1ms -> 0.1% Duty Cycle
%Currently duty cycle set to 1/1000 -> 1us on time

Nsims = 500;
coll_sims_m = zeros(Nsims,numRadar);
rseed_v = (1:Nsims);
k = period;
N = numRadar;
m = 2;
k_inv = (1/k);
```

Figure 9: MATLAB Initialization - Established Main Variables

This section of the code establishes the key variables used throughout the rest of the code such as "numRadar": the number of radars in the environment, "period": the period of time or number of microseconds, and "Nsims": the number of simulations run. Setting the period to 1 millisecond sets the radars to all have a duty cycle of 0.1%. This means that they all only send one pulse every millisecond. These variable values can all be changed easily to see the impact they have on the simulation results. "coll_sims_m" is a matrix of zeros that is the length of the number of simulations and the width of the number of radars. This is just setting up a blank matrix that will be indexed with simulation data later in the code. "rseed_v" is a vector of 1 to the number of simulations of seeds. This is used to make sure that each Monte Carlo trial uses a different

random number set so that the simulation generates better results. The rest of the variables are used for the probability calculation in the other MATLAB script and will be talked about later.

This section of code shown below is the heart of this script. It is the simulation phase of the program. It is basically a Monte Carlo simulation that randomly assigns a random time slot for each radar pulse. This is done by using "rng(rseed_v(ks),'twister')" to assign different seeds to each simulation and by using the "randi" function in the "sigStart_v" variable below. Using different seeds for each simulation makes sure that you have a different number set for each run of the Monte Carlo trial. The "radar_m" variable generates a matrix of zeros with dimensions of the number of radars by the number of time slots/microseconds. This is set up as an empty matrix with these specific dimensions so that it can be indexed to store the information needed from the inner "for loop." Each row represents a radar, and each column represents a microsecond time slot.

```
% ======================== SIMULATION ==================
for ks = 1:Nsims %runs simulation multiple times if wanted
    rng(rseed_v(ks),'twister')
    %actual simulation
    sigStart_v = randi(period,numRadar,1); %locks signal to lus grid, uniform distribution

    radar_m = zeros(numRadar,period);   %each radar is its own row
    for knradar = 1:numRadar
        radar_m(knradar,sigStart_v(knradar)) = 1;
        sum_t_v = sum(radar_m) ;%creates radarx1 vector of pulses per time unit
        coll_t_v = sum_t_v > 1 ;%creates radarx1 vector, = 1 if collision (>=2 simultaneous radar pulses)
        coll_sims_m(ks,knradar) = sum(coll_t_v);  %logs how many collisions in simulation
    end

    coll_pr = sum(coll_sims_m); %number of total collions across all simulations
    Ncoll_v = mean(coll_sims_m); % avg number of collisions relative to number of radars
end
```

Figure 10: Simulation Phase of MATLAB Code - Monte Carlo Simulation

This inner for loop is where the bulk of the calculations and data come from. The for loop runs from 1 radar to the number of radars specified earlier in the code. For each iteration it indexes the "radar_m" matrix with the random, uniformly distributed pulses generated in the "sigStart_v"

variable. The next line with “sum_t_v” sums up the values in those time slots to see how many pulses were received at each microsecond. The following line with variable “coll_t_v” looks at the values from “sum_t_v” to see what the values are. If “sum_t_v” has any values greater than one, it records it as a time collision because if you have two or more pulses in a time slot it is considered a collision. In the next line the variable “coll_sims_m” is indexed by the simulation number and the radar number. It then stores the sum of “coll_t_v” in that slot of the matrix. This means it is calculating how many folding collisions happen in each simulation for each number of radars. After exiting the inner for loop and going back to the outer for loop two new variables are made. The first variable “coll_pr” sums up the matrix “coll_sims_m” that records the number of collisions occurring in each simulation. “Coll_pr” now has total number of collisions from all the simulations added together and sorted by the number of radars. The variable “Ncoll_v” in the next line takes the mean of the total number of collisions in all simulations from the “coll_pr” variable. This shows how many collisions happen on average for each number of radars based on the large number of simulations.

The last section of this program, shown below, is the shortest and simplest. All it does is save the data from this program in the variables that are specified in the save function. This allows the users to import the data into other MATLAB scripts and use the same variables. In this case it was used to bring the data needed into the other MATLAB program written for this project.

```
52  % ======================== SAVE RESULTS ================================
53  fileNameData_c = [programName_c,'_data'];
54  fileNameGroundTruth_c = [programName_c,'_groundTruth'];
55
56  save(fileNameData_c,'coll_sims_m','coll_t_v','sum_t_v','coll_pr','Ncoll_v','sigStart_v',
57  'radar_m','numRadar','Nsims','k','N','m', 'k_inv')
```

Figure 11: Save Results Phase of MATLAB Code

The next MATLAB script written for this thesis was “time_coincident_plot”. As referenced in the name, the main purpose of this program was to plot the data collected from the “time_coincident_sim” program. The first two sections of this script are the same in principle to the other code. The first section shows the script name, purpose, usage, author, and date written. It can be seen in the figure below.

time_coincident_plot.m

```
% script  time_coincident_plot
% purpose Plot time coincident data and calculate probability of collision.
%
% usage Used to plot the data collected from the "time_coincident_sim" script and calculate the probability of FC.
% notes
% author  Hunter Miller
% date      04/02/2023

```

Figure 12: MATLAB Code Characterization - Plot

The next section is also the same as the previous script. It is the common initialization phase that sets up the plotting information for the figures. It can be seen in the figure below.

```
% ================== COMMON INITIALIZATION ======================
programName_c = mfilename;              % char string of this script name
msg1_c = [programName_c,': ', date];   % program name + date plot generated

msg3_c = 'Hunter Miller';              % name goes here (adapt for particular work requirements)

if ~exist('figNum','var')
    figNum = 1;                        % Set figNum = 1 unless already exists in workspace from earlier plots
end

plotNotes_h                            % script to set up plotting definitions
```

Figure 13: Common Initialization Phase – Plot Information

The next section is the initialization phase of the program. The first step in this section is loading the “time_coincident_sim_data” from the other program. This imports all the data we need to calculate the probability of collisions and plot.

```
% ======================== INITIALIZATION ==========================
load("time_coincident_sim_data.mat") % load
p_t =  zeros(1,N);
p_ind = zeros(1,N);
p = zeros(1,N);
```

Figure 14: Initialization Phase of MATLAB Plot Script – Load Data and Initialize Variables

The following three lines of code are creating vectors of zeros to be indexed in the probability calculation later in the program.

The next phase of this program is the simulation. This section of the script is fairly short in comparison to the other program. It is started off by setting up the frequency grid width. This is how far apart the folded signals need to be to not be considered a frequency collision.

```
25  % ======================== SIMULATION ============================
26  %Probability Calculations
27  freq_grid_mhz = 20 ; % 500 MHz/20 = 25 MHz grid sections
28  for kr = (1:N)
29      p_t(kr) =  get_pd_mofn_window(k,m,kr);
30      p_ind(kr) = p_t(kr) * (1/1000);
31      p(kr) = p_ind(kr) *(1/freq_grid_mhz);
32  end
```

Figure 15: Simulation Phase of Plot Script - Probability Calculation

This for loop is used to calculate the probabilities of time collisions and overall folding collision in reference to the number of radars. This calculation is based off the following formula developed by Dr. Fudge. It is used to calculate the probability of group appearances.

Equation 2: Group Appearance Probability

$$P_{M,N,K} \cong K \sum_{m=M}^{N} \binom{N}{m} (1/K)^m (1-1/K)^{N-m}$$

m is the number needed out of the group, of N, to be considered a collision. Therefore, *m* is 2.

N is the number of items in the group: in this case the number of radars

K is number of time occurrences: In this case it would be 1000 microseconds

Folding collisions are considered a group appearance because at least two pulses must arrive at the same time. This equation was derived by Dr. Fudge and is most accurate for lower probabilities. It is not proven but it has been tested and found to be accurate for low probabilities.

The purpose of the final section of this script is to plot the data from all the calculations shown previously. It sets up a figure with three subplots. The first plot displays the number of radar pulses received at each timeslot and/or microsecond as well as where the time component collisions occur. The second plot shows the average number of time component collisions relative to the number of radars in the environment. The third subplot shows the probability of folding collisions, taking the frequency component into consideration, relative to the number of radars in the environment.

```
% ========================= PLOT RESULTS ==========================
figure(figNum)
set(gcf,'position',plotPositionWide_v)
subplot(3,1,1);
plot(sum_t_v, 'r')  %plot radars
hold on
plot(coll_t_v,'b') %plot collisions
legend('Pulses Received','collisions')
%Labeling
title('Radar Timings and Collisions')
ylabel('# Radars, Collision')
xlabel('Time, microseconds')


subplot(3,1,2);
plot(Ncoll_v, 'g')
title('Number of Time Collisions per Radar')
ylabel('Collisions')
xlabel('# of Radars')



subplot(3,1,3);
plot(p, 'b')
title('Probabilty of Folding Collision')
ylabel('Folding Collision Probability')
xlabel('# of Radars')

figNum = figNum + 1;
```

Figure 16: Plot Results Section of Code – Used to Plot Important Data

Plots and Data Analysis

The program was written in multiple stages and continued to evolve as time went on. To start off, the program recorded and plotted the number of pulses received at each time slot. The time slots with two or more pulses were considered time collisions. An example of this can be seen in the figure below.

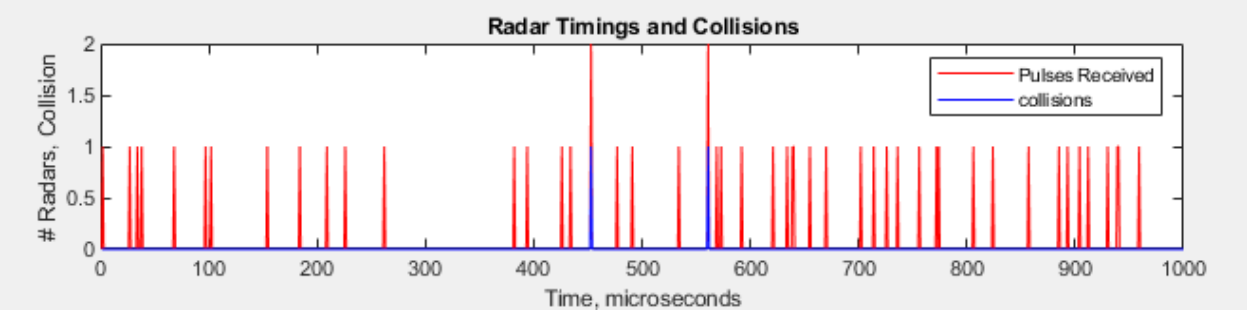


Figure 17: Radar Pulses Received Per Microsecond.

The red lines indicate a radar pulse, and the blue lines represent a collision. As you can see in Figure 16 the blue spikes are shown where there are two pulses received on a timeslot, therefore being a collision. This graph is specifically for the time collisions. Once the frequency collision is incorporated into the calculation the probability of collision goes down drastically.

The next component added to the program was a calculation of the total number of collisions across all simulations. This variable was originally assumed to be the number of time collisions per simulation, but this hypothesis was debunked after observing the graph. An example of this plot can be found in the figure below.

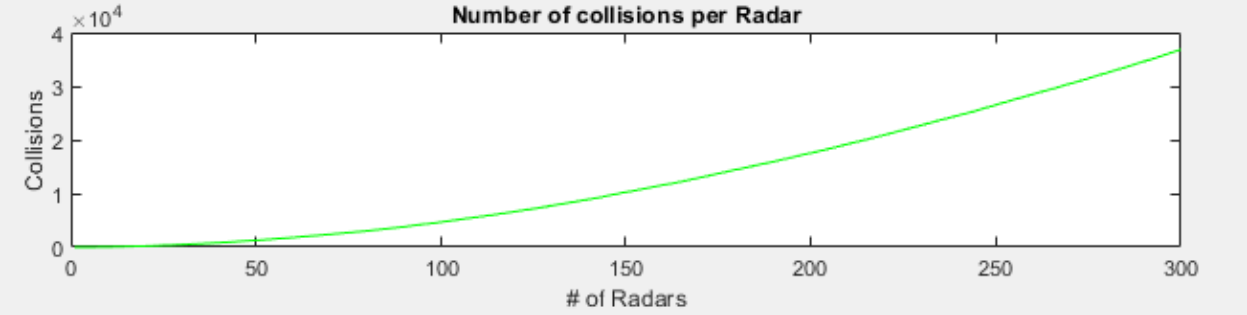


Figure 18: Plot for Sum of Time Collisions Across all Monte Carlo Simulations

After the collision value was observed to be 40000, which is far greater than the 300 radars present in the simulation, the deduction of what these values actually represented was made.

From this point the code was altered to show the average number of collisions per radar to achieve the desired values. In order to do this the mean of the number of total collisions was taken. This gave the desired and expected result seen below.

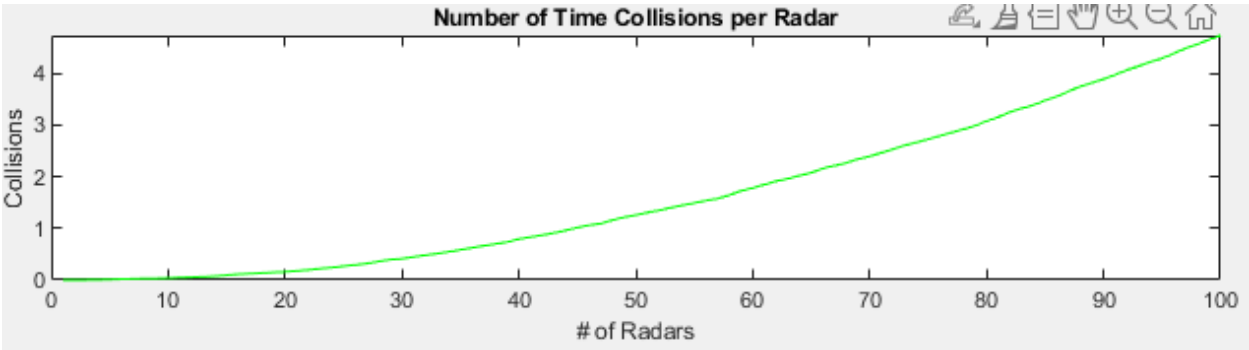


Figure 19: Average Number of Collisions to Number of Radar

Once this calculation was figured out, the final step was to convert the data into the probability of a folding collision. This was rather challenging because this is an unexplored topic. Dr. Fudge offered valuable insight into this area and provided the probability formula mentioned earlier. This formula and a MATLAB function given by Dr. Fudge, that are included

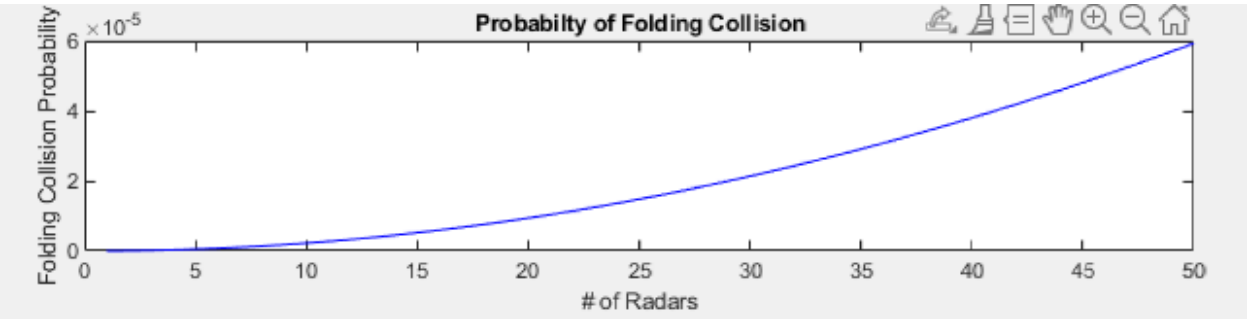


Figure 20: Folding Collision Probability Plot

in the appendix, allowed for this theoretical probability to be calculated and plotted. An example of this probability plot can be found in the figure below.

As these scripts evolved and improved over time, different variables were changed to see the impacts they would have on the data. The variables that were altered were the number of radars in the environment as well as the number of Monte Carlo simulations that were run. As expected, as the number of radars went up so did the number of collisions, and the probability of collisions occurring. As the number of Monte Carlo trials increased the plotted curve was smoothed out, giving more accurate results. Both effects can be seen in the figures below.

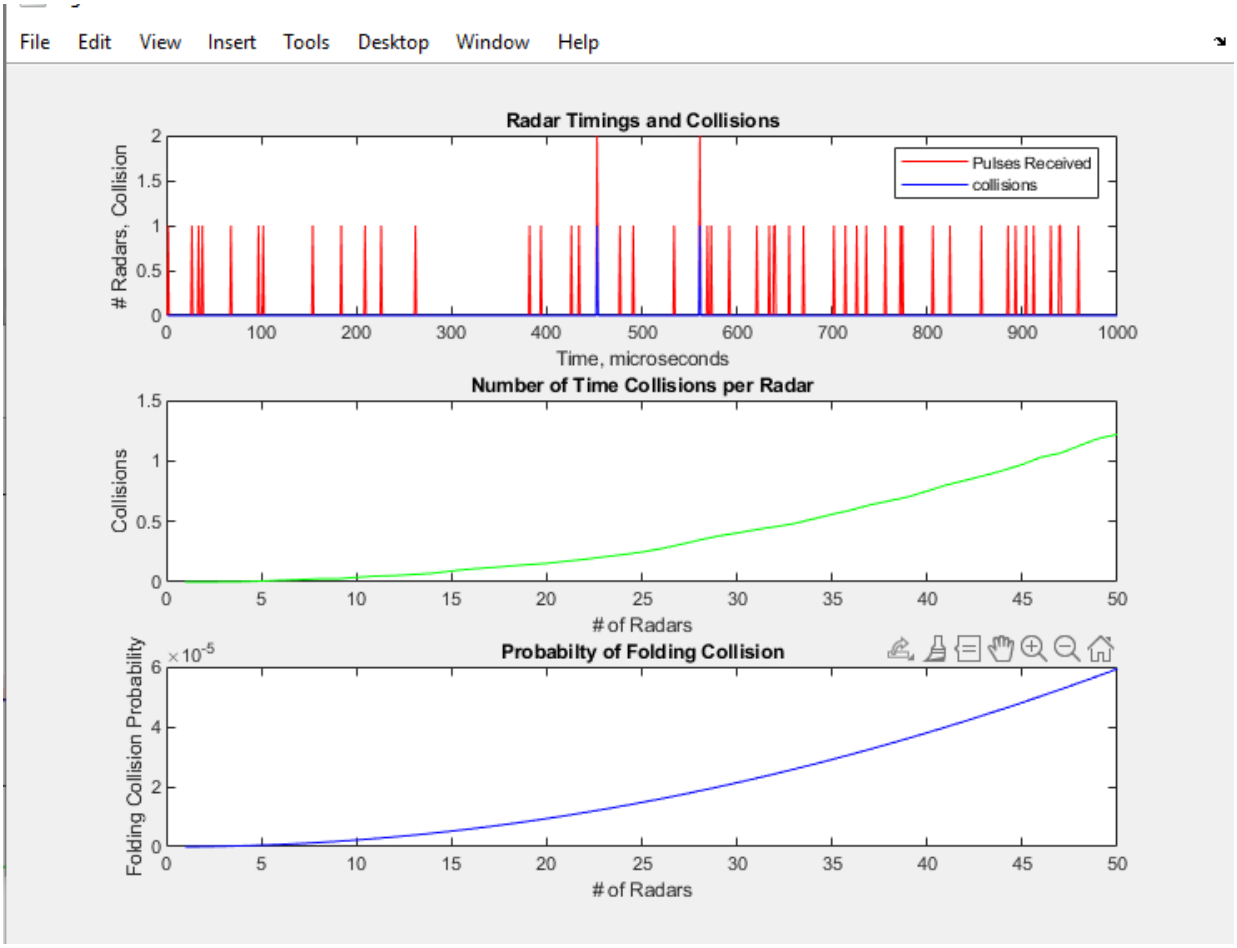


Figure 21: Plots for Folding Collision Probability - 50 Radar, 400 Simulations

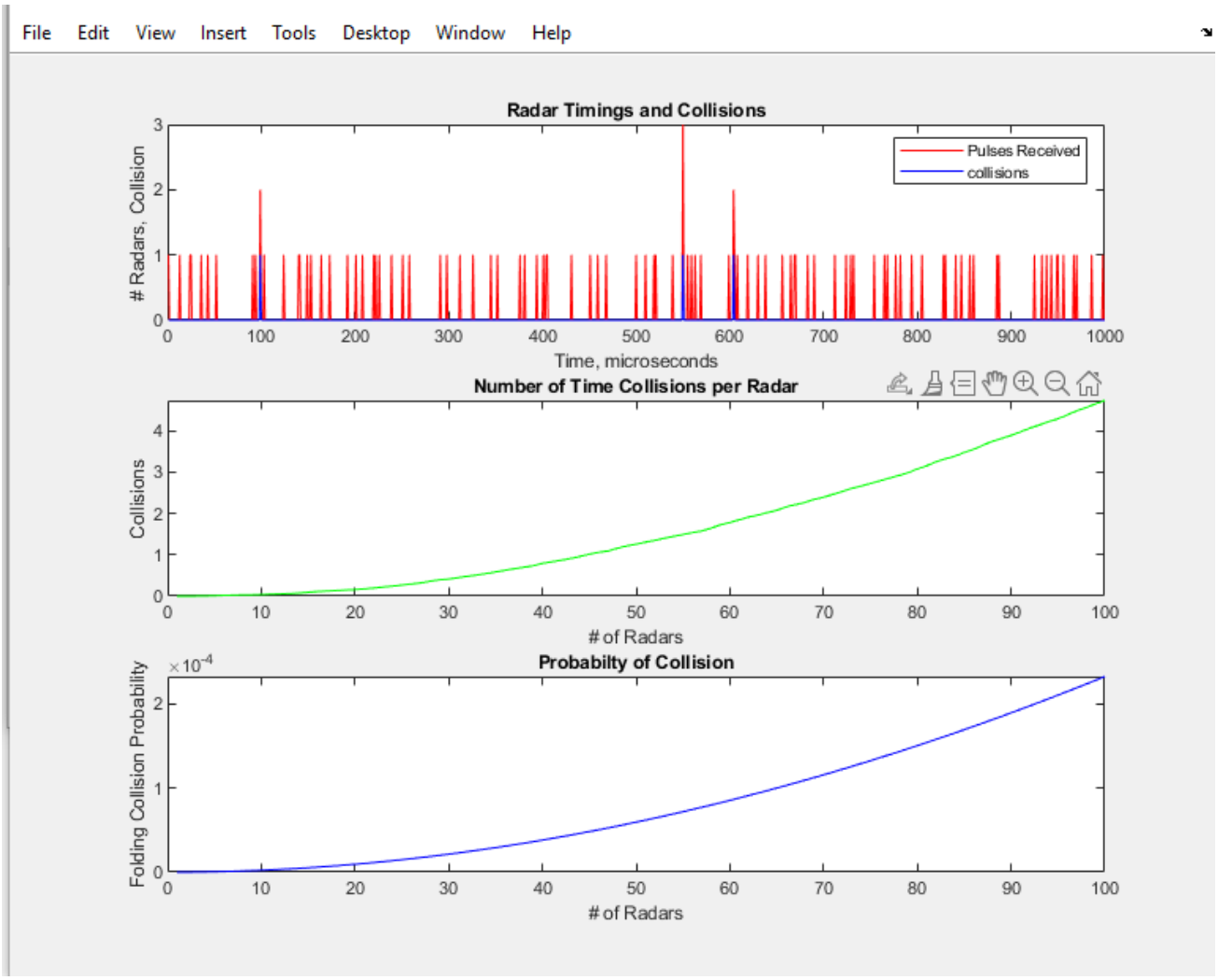


Figure 22: Plots for Folding Collision Probability - 100 Radar, 750 Simulations

It can clearly be seen in these two figures that the probability of folding collisions increases as more radars are introduced into the environment.

Conclusion

After reviewing the data and results found during this project, the probability of folding collisions can be seen to be very low as initially hypothesized. Furthermore, some of the parameters used to simplify this simulation, such as all radars having the same duty cycle and pulse width could have increased the probability of folding collisions. In a real radar environment these parameters would change from radar to radar, most likely further decreasing the probability of collisions. It is also unlikely that there would be 50 or more radars within range at a given time, further reducing the probability of collisions.

The desired results were achieved from this project. A good simulation to calculate folding collisions was made and works as designed. It will provide a great foundation for other students and Dr. Fudge to build on as they continue to pursue research in this unexplored topic. Furthermore, from this project the student was able to gain valuable knowledge that further cemented their future as an Electrical Engineer. The knowledge gained in this project included further understanding of higher-level coding in MATLAB, advanced signal processing concepts, and the value of collaborating with others to tackle high level problems. These topics will help the student by providing a strong foundation for their future in the workplace.

Future Work

Due to the time constraints and complexity of this project, not everything that was envisioned was able to be accomplished. Given more time, the complexity of the simulation would be increased, including varying duty cycles, pulse widths, and pulse repetition intervals. Also, creation of a randomized folded frequency component would be added to the simulation to make the frequency collision probability even more realistic.

Another future step would be to further test and enhance the probability equation that was used for the calculations in this project. Dr. Fudge is planning on working to prove his derived equation and possibly change the calculation for the frequency coincidence. Should this change be made, it could easily be implemented into the current MATLAB scripts.

References

Chen, H.-C., Kung, H. T., Vlah, D., & Suter, B. W. (2011). *Measurement combining and progressive reconstruction in compressive sensing*. Retrieved 3 30, 2023, from http://eecs.harvard.edu/~htk/publication/2011-milcom-chen-kung-vlah-suter.pdf

Fudge, G., Bland, R. E., Chivers, M. A., Ravindran, S., Haupt, J., & Pace, P. E. (2008). *A Nyquist folding analog-to-information receiver*. Retrieved 1 20, 2023, from http://ece.rice.edu/~jdh6/publications/asilomar08_nyfr.pdf

Harrison, R. L. (2010). Introduction To Monte Carlo Simulation. *AIP conference proceedings, 1204*, 17-21. doi:https://doi.org/10.1063/1.3295638

Hyperlabs. (2022, March). *HL9333 Sampler / Harmonic Mixer IC*. Retrieved from https://www.hyperlabs.com/wp-content/uploads/HL9333.pdf

Liu, X., Li, T., Fan, X., & Chen, Z. (2019). Nyquist Zone Index and Chirp Rate Estimation of LFM Signal Intercepted by Nyquist Folding Receiver Based on Random Sample Consensus and Fractional Fourier Transform. *Sensors, 19*(6), 1477. Retrieved 1 20, 2023, from https://mdpi.com/1424-8220/19/6/1477/htm

Maleh, R., Fudge, G., Boyle, F. A., & Pace, P. E. (2012). Analog-to-Information and the Nyquist Folding Receiver. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 2*(3), 564-578. Retrieved 1 20, 2023, from https://ieeexplore.ieee.org/document/6355638

Martin, J. C. (2018). *Analysis of the Nyquist Folding Receiver (NYFR)*. Retrieved 1 20, 2023, from https://shareok.org/handle/11244/299690

Pearson, S. (2023). Estimating Direction of Arrival For Multi-Arm Spiral Antennas Using Machine Learning.

Sharma, D. K., Singh, B., Raja, M., Regin, R., & Rajest, S. S. (2021). An Efficient Python Approach for Simulation of Poisson Distribution. *IEEE*.

Wang, J., Chen, S., Jiang, K., & Tang, B. (2016). *Frequency Estimation Using SAMP-SVD Based on Nyquist Folding Receiver*. Retrieved 1 20, 2023, from https://matec-conferences.org/articles/matecconf/ref/2016/19/matecconf_iccae2016_05012/matecconf_iccae2016_05012.html

Appendix

A. Time Coincidence Simulation

time_coincident_sim.m

```
% script  time_coincident_sim
% purpose Generate seperate radar pulses w/ given period,duty cycle
%           and detect collisions (>=2 simultaneous pulses)
% usage
% notes
% author  Hunter Miller
% date      04/02/2023
% ================== COMMON INITIALIZATION ======================
programName_c = mfilename;              % char string of this script name
msg1_c = [programName_c,': ', date];     % program name + date plot generated

msg3_c = 'Hunter Miller';                % name goes here (adapt for particular work requirements)

if ~exist('figNum','var')
    figNum = 1;                     % Set figNum = 1 unless already exists in workspace from earlier plots
end

plotNotes_h                             % script to set up plotting definitions
% ======================== INITIALIZATION ==================
numRadar = 50; %num of signals, all identical (1us PW)
                %varying in start time
period = 1000; %1000us = 1ms -> 0.1% Duty Cycle
%Currently duty cycle set to 1/1000 -> 1us on time

Nsims = 750;
coll_sims_m = zeros(Nsims,numRadar);
rseed_v = (1:Nsims);
k = period;
N = numRadar;
m = 2;
k_inv = (1/k);
% ======================== SIMULATION ==================
for ks = 1:Nsims %runs simulation multiple times if wanted
    rng(rseed_v(ks),'twister')
    %actual simulation
    sigStart_v = randi(period,numRadar,1); %locks signal to 1us grid, uniform distribution

    radar_m = zeros(numRadar,period);   %each radar is its own row
    for knradar = 1:numRadar
        radar_m(knradar,sigStart_v(knradar)) = 1;
        sum_t_v = sum(radar_m) ;%creates radarx1 vector of pulses per time unit
        coll_t_v = sum_t_v > 1 ;%creates radarx1 vector, = 1 if collision (>=2 simultaneous radar pulses)
        coll_sims_m(ks,knradar) = sum(coll_t_v);  %logs how many collisions in simulation
    end

    coll_pr = sum(coll_sims_m); %number of total collions across all simulations
    Ncoll_v = mean(coll_sims_m); % avg number of collisions relative to number of radars
end
% ========================= SAVE RESULTS ================================
fileNameData_c = [programName_c,'_data'];
fileNameGroundTruth_c = [programName_c,'_groundTruth'];
save(fileNameData_c,'coll_sims_m','coll_t_v','sum_t_v','coll_pr','Ncoll_v','sigStart_v',
'radar_m','numRadar','Nsims','k','N','m', 'k_inv')
```

B. Time Coincidence Plot Simulation

time_coincident_plot.m

E:/Matlab Sims/time_coincident_plot.m

```
% script  time_coincident_plot
%  ...t data and calculate probability of collision.
% usage Used to plot the data collected from the "time_coincident_sim" script and calculate the probability of FC.
% notes
% author  Hunter Miller
% date      04/02/2023
% ================== COMMON INITIALIZATION ======================
programName_c = mfilename;              % char string of this script name
msg1_c = [programName_c,': ', date];   % program name + date plot generated
msg3_c = 'Hunter Miller';              % name goes here (adapt for particular work requirements)

if ~exist('figNum','var')
    figNum = 1;                        % Set figNum = 1 unless already exists in workspace from earlier plots
end
plotNotes_h                            % script to set up plotting definitions
% ======================== INITIALIZATION ========================
load("time_coincident_sim_data.mat") % load
p_t =  zeros(1,N);
p_ind = zeros(1,N);
p = zeros(1,N);
% ======================== SIMULATION ============================
%Probability Calculations
freq_grid_mhz = 20 ; % 500 MHz/20 = 25 MHz grid sections
for kr = (1:N)
    p_t(kr) =  get_pd_mofn_window(k,m,kr);
    p_ind(kr) = p_t(kr) * (1/1000);
    p(kr) = p_ind(kr) *(1/freq_grid_mhz);
end
% ======================== PLOT RESULTS ==========================
figure(figNum)
set(gcf,'position',plotPositionWide_v)
subplot(3,1,1);
plot(sum_t_v, 'r')  %plot radars
hold on
plot(coll_t_v,'b') %plot collisions
legend('Pulses Received','collisions')
%Labeling
title('Radar Timings and Collisions')
ylabel('# Radars, Collision')
xlabel('Time, microseconds')

subplot(3,1,2);
plot(Ncoll_v, 'g')
title('Number of Time Collisions per Radar')
ylabel('Collisions')
xlabel('# of Radars')

subplot(3,1,3);
plot(p, 'b')
title('Probabilty of Folding Collision')
ylabel('Folding Collision Probability')
xlabel('# of Radars')
figNum = figNum + 1;
```

C. Get N Choose M MATLAB Script: From Dr. Fudge

get_n_choose_m.m

E:/Matlab Sims/get_n_choose_m.m

```
2  % purpose      Compute n choose m = n! / m!(n-m)!
3  % usage        n_choose_m = get_n_choose_m(n,m)
4  % date         5/9/2017
5  % programmer   Jerry Fudge
6
7  function n_choose_m = get_n_choose_m(n,m)
8
9  %if n > 50      % use stirling approximation
10 %  K=n-m;
11 %  n_choose_m = (sqrt(2*pi*n)*(n/exp(1))^n)/(factorial(m)*(sqrt(2*pi*K)*(K/exp(1))^K));
12 %else
13 %   n_choose_m = factorial(n)/(factorial(m)*factorial(n-m));
14 %end
15
16 if n > 100
17    n_choose_m = 1;
18    nprod = n-m;
19    nt = n;
20    nb = n-m;
21    for k = 1:nprod
22       n_choose_m = n_choose_m*nt/nb;
23       nt=nt-1;
24       nb=nb-1;
25    end
26    n_choose_m = round(n_choose_m*factorial(nt)/factorial(m));
27 else
28    n_choose_m = factorial(n)/(factorial(m)*factorial(n-m));
29 end
30
```

D. Get Probability Density of M of N MATLAB Script: From Dr. Fudge

get_pd_mofn.m

```
% function     get_pd_mofn
% purpose      Get m of n pd (at least m of n chances)
% usage        [pd_mofn,pd_exact_mofn_v] = get_pd_mofn(pd,m,n)
% notes        (1) computes exact for each possible mx >= m, then sums
% date         5/9/2017
% programmer   Jerry Fudge

function [pd_mofn,pd_exact_mofn_v] = get_pd_mofn(pd,m,n)

m_v = (m:n)';
nm_v = n - m_v;
n_cases = n-m+1;
pd_v = (pd*ones(n_cases,1)).^m_v;         % m detects
pnd_v = ((1-pd)*ones(n_cases,1)).^nm_v;   % not detects on n-m+1
ncombm_v = zeros(n_cases,1);
for kc = 1:n_cases
   ncombm_v(kc) = get_n_choose_m(n,m_v(kc));
end

pd_exact_mofn_v = pd_v.*pnd_v.*ncombm_v;

pd_mofn = sum(pd_exact_mofn_v);
```

E. Get PD M of N Window MATLAB Script: From Dr. Fudge

get_pd_mofn_window.m

```
% function      get_pd_mofn_window
% purpose       Given window of K opportunities, compute approximate probability
%               of >= m occurences from selection of n for each opportunity
% usage         [pd_mofn_joint] = get_pd_mofn_window(K,m,n)
% notes         (1) For example, suppose have a window of K minutes,
%                  and you have n people who each have one hallucination
%                  during the K minute window, what is the probability
%                  of having at least n simultaneous hallucations?
%               (2) This estimates high and can be > 1 for true high
%                  probability cases. For n = 15, once K >= 50, this
%                  is extremely accurate
%               (3) Assumes that already have 100% probability for
%                  each in group to have one event during window
% date          5/11/2017
% programmer    Jerry Fudge

function [pd_mofn_joint] = get_pd_mofn_window(K,m,n)


pd_mofn_joint = K*get_pd_mofn(1/K,m,n);
```