Naval Research Laboratory
Washington, DC 20375-5320

NRL/8110/MR—2024/1

Case-Study Analysis for Deinterleaving of X-Band RADAR Scans

WILLIAM G. WARREN

JARED ALLANIGUE

JAMES V. OUTLAW

ALAN SCHROEDER

GIL CHAPMAN

ERIN M. MANSFIELD

DR. SAMUEL LAMBRAKOS

Materials and Sensors Branch
Materials Science and Technology Division

May 15, 2024

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
15-05-2024	NRL Memorandum Report	START DATE 11/09/2024	END DATE 02/04/2024

4. TITLE AND SUBTITLE

Case-Study Analysis for Deinterleaving of X-Band RADAR Scans

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
		1G01

6. AUTHOR(S)

William G. Warren, Jared Allanigue, James V. Outlaw, Alan Schroeder, Gil Chapman, Erin M. Mansfield, and Dr. Samuel Lambrakos

7. PERFORMING ORGANIZATION / AFFILIATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
4555 Overlook Ave SW
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/8110/MR—2024/1

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR / MONITOR'S ACRONYM(S) NUMBER	11. SPONSOR / MONITOR'S REPORT NUMBER(S)
US Department of Defense 4000 Defense Pentagon, Washington, DC 20301	US DoD	

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A: Approved for public release; distribution is unlimited.

13. SUPPLEMENTAL NOTES

14. ABSTRACT

A case study analysis is presented demonstrating the procedure of deinterleaving, and its characteristics in terms of parameter sensitivity, which is for PRI-modulation recognition associated with ELINT analysis. This study examines data obtained from a RADAR collection site, consisting of various PRI-modulated characteristics, such as those occurring typically for ELINT-analysis environments. A specific aspect of this analysis is that it considers deinterleaving of RADAR pulses, which represent a precursor data format collected in a tactical maritime environment, to construct RADAR scans. Ultimately, the goal of this analysis is to construct these RADAR scans to eventually construct and identify RADAR signals, which represent a post-processed data format with respect to scans, in order to determine the geolocation of maritime targets. We present initial results of an evolving data-driven methodology based on the types of data being processed.

15. SUBJECT TERMS

Deinterleaving, ELINT analysis, RADAR scans, PRI modulation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	c. THIS PAGE		
U	U	U	SAR	46

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
William G. Warren	(202) 279-4433

This page intentionally left blank.

Abstract

A case study analysis is presented demonstrating the procedure of deinterleaving, and its characteristics in terms of parameter sensitivity, which is for PRI-modulation recognition associated with ELINT analysis. This study examines data obtained from a RADAR collection site, consisting of various PRI-modulated characteristics, such as those occurring typically for ELINT-analysis environments. A specific aspect of this analysis is that it considers deinterleaving of RADAR pulses, which represent a precursor data format collected in a tactical maritime environment, to construct RADAR scans. Ultimately, the goal of this analysis is to construct these RADAR scans to eventually construct and identify RADAR signals, which represent a post-processed data format with respect to scans, in order to determine the geolocation of maritime targets. We present initial results of an evolving data-driven methodology based on the types of data being processed.

1 Introduction

The deinterleaving of RADAR pulses is the process of separating inputted interleaved pulse streams for the purpose of electronic intelligence (ELINT) analysis [1, 2]. At this stage, there exists a large volume of studies concerning various types of deinterleaving algorithms. A comparison of their performance characteristics with respect to various types of RADAR-scan data remains an open area for further investigation. Reference [3], in its introduction, provides a fairly comprehensive survey of various deinterleaving studies and their categorization. Our analysis of data applies procedures based on histogram-binning. This category of deinterleaving procedures includes a wide range of studies that consider the influence of noise and missing-pulse artifacts on constructed histograms [3].

Presented in this study is analysis demonstrating the procedure of deinterleaving, and its characteristics in terms of parameter sensitivity, which is for modulation recognition based on pulse repetition intervals (PRI) associated with ELINT analysis. A PRI is described as the time of arrival (TOA) difference between two RADAR pulses originating from the same emitter. The TOA of a RADAR pulse is the time a collector receives the pulse in reference to some time. Our analysis uses TOA in reference to the unix time of seconds past midnight UTC on January 1, 1970. These studies examine data obtained from a RADAR collection site, consisting of various PRI-modulated characteristics, such as those occurring typically for ELINT-analysis environments. PRI-modulation refers to the adjustment of an emitter's PRI-pattern based on desired detected distances. However, error is introduced in a received PRI-modulated pattern when RADAR pulses from multiple emitters are collected at the same time and area. As a result, deinterleaving these received RADAR pulses into categorized RADAR scans gives a more efficient approach at understanding what PRI-modulated patterns are in the collected data.

In general, the methodology applied in this study combines formally those of pulse descriptor word (PDW) clustering techniques based on agglomerative hierarchic methods, discussed thoroughly in reference [4], and TOA-based histograms. A comparison of our

Manuscript approved May 14, 2024.

methodology to others is not within the scope of our analysis. In addition, not within the scope of our analysis, are any reviews of existing methodologies in terms of their performance. Further, this study does not assume a sufficiently complete list of references citing existing methodologies and results of their applications. Accordingly, references cited here are considered a reasonable sampling of existing studies. This follows in that methods and associated analyses of deinterleaving are continuously progressing. Reference [5] presents a somewhat historical overview of deinterleaving methods. Finally, the progressing volume of deinterleaving studies, which are tending to explore the interrelationship of different algorithms in terms of their formal similarities, differences, and their effectiveness for pulse-data analysis as used in combination. This has motivated our analysis framework to have a foundation on inverse-analysis theory [6,7].

For our analysis, we adopt the general perspective of inverse analysis [6,7], where our modeling framework is considered a priori as data-driven, i.e., formally motivated by characteristics of a specific class (or subset) of data, and within a "model space" of algorithms. A collection of algorithms whose different structures may be formally related, redundant with respect to certain types of data analysis, and have different advantages based on the types of data considered for analysis, and may be applied iteratively and in combination for analyses. This perspective, which is described schematically in Figure 1, is motivated by the inherent complexity of emitter data and the continuing development of deinterleaving algorithms, used separately and in combination.

The modeling framework is characterized by specific software implementations, where the numerical-computation formulation of a modeling framework has associated performance characteristics related to the software implementation. Again, by adopting the model-space perspective, we do not compare our framework to others, in terms of similarity and "general" performance, but rather to place emphasis on its structure and performance characteristics with respect to selected sets of data. General comparison of its performance and formal similarity is considered as appropriate for separate studies, in themselves.

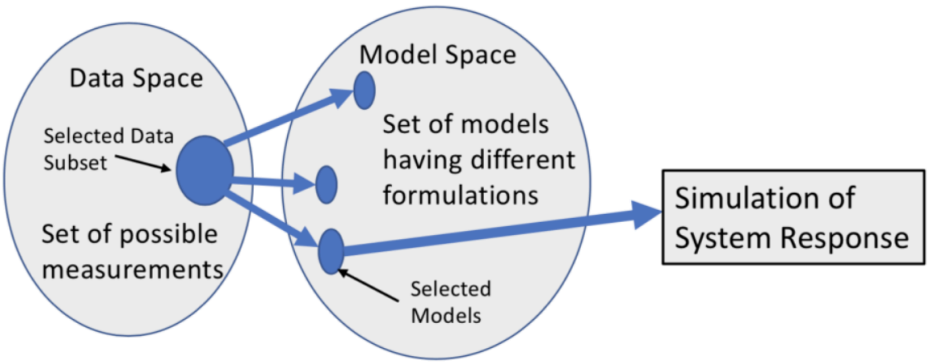

Figure 1. Mappings between data, model, and simulation spaces for inverse-analysis.

Analysis of physical processes has motivated different types of inverse-analysis problems [6,7]. Our framework is based around inverse scattering theory [8]. Inverse scattering poses specific problems for source analysis, with the purpose of identifying and estimating source

characteristics, and construction of their parameterized phenomenological representation. Following the inverse-problem approach [6,7], a system is represented by a parametric model. The particular choice of a parametric model is termed a "parameterization" of the system [7]. The choice of a particular parameterization, however, is generally not unique. In order to address the property of non-uniqueness of system parameterization, inverse problem theory has adopted the concept of "model space," where subregions of this space represent "conceivable" parametric models of the system [7].

Given a model space for PRI-modulation recognition, quantitative inverse analysis is further enhanced by isolating model-space regions associated with parameterizations that are physically consistent, conveniently adjusted with respect to given sets of measurements, and sufficiently general in term mathematical representation. A conveniently adjustable and sufficiently general parameterization of PRI-modulation is significant for the following reasons.

1. PRI-modulation features calculated by inverse methods represent a mapping from data space into parameter space. It is preferable to adopt a parametric function representation that tends to minimize any bias resulting from its mathematical form.
2. A set of parameters associated with a physically consistent representation can be used to extract relationships between parameters, which can provide further insight related to physical characteristics.
3. PRI-feature extraction for a specific application requires quantitative assessment of pulse characteristics over a sufficient range of parameter values. Fourth, a sufficiently general parametric model formulation can be adjusted to include influences due to incomplete information concerning the system.

A strict mathematical interpretation of TOA-histogram techniques is that they may be considered as formally within the category of clustering techniques, assumed by reference [9]. This interpretation appears not to be adopted by some studies. For example, reference [10] foundationally considers clustering and histogram-based deinterleaving as separate approaches. Specifically, reference [10] adopts the perspective that TOA-based histogram techniques are formally separate from clustering techniques applied to PDW feature-vectors. Reference [10] states: "Because of their varying nature PA (Pulse amplitude) and TOA cannot be used in clustering."

It has been noted that clustering techniques [10-14], as defined separately from TOA-based histogram techniques [15-19], have deficiencies that can be overcome by combining them with TOA-based techniques, such as two RADARs located in the same direction that have similar parameters and cannot be deinterleaved using clustering (as defined separately). A subtle conceptual aspect of reference [10] is that the concept of model space emerges persistently throughout that analysis, although not emphasized conceptually with respect to inverse analysis theory. In particular, the general approach described by [10] is that of a model space, where a set of algorithms are used in combination in order to compensate for their relative deficiencies.

In December 2010, the NRO approved the description of Dr. James W. Stoner's contributions to ELINT analysis, as cited in reference [20]. Among these contributions was a general approach to ELINT analysis, which emphasized that algorithms for ELINT analysis should be data-driven, i.e., as quoted, "Data Dictates Design." Our methodology adopts this approach as a paradigm for algorithm development, emphasizing that "data-driven analysis" has foundation within inverse-analysis theory, and is related to the concepts of "Data and Model Spaces."

The emphasized concepts of "Data and Model Spaces" follow from observations that given sets of algorithms are more appropriate for given sets of data, and that algorithms for specific types of analyses tend to have formal similarities, as well as possible structural differences. Further, a model space should be interpreted as containing different algorithm formulations, numerical-computation, software implementations of these algorithms. And finally, the data-driven concepts of "Data and Model Spaces," suggest that comparison of a given algorithm's performance to that of others should always be within the context of specific data sets.

This report is organized as follows. In Section 2, the RADAR-scan data is described. In Section 3, we provide a formal representation of our deinterleaving procedure in terms of mathematical structure and flowchart ontology. In Section 4, we present the results of our analysis. In Section 5, we present case study results of different sets of RADAR pulse data and compare the noise residue between each data set. This is followed-up with a discussion of the formal structure of our methodology that is related to that of references [21-23]. Reference [24] presents a method for deinterleaving signals based on the algorithms of [21-23]. It is further enhanced by the discussion and eventual implementation of the construction of RADAR signals using PRI-modulated pattern identification and a calculated clock metric. And finally, given in the Appendix, is a computer program providing software implementation of our procedure.

2 Description of RADAR-Scan Data

The data in this analysis is assumed as representative of general characteristics of pulses originating from typical emitters in tactical maritime environments. This data, however, is motivated by actual emitter pulse characteristics. The pulse-data consists of times of arrivals (TOA) in seconds after unix time, pulse frequencies (RF) in MHz, pulse widths (PW) in μs, and pulse amplitude (Amp) in dBm. A scattergram of this data's Amp vs TOA is shown in Figure 2. Our analysis considers a data set consisting of 10,001 pulses. The RF are within the designated frequency range for maritime radionavigation [25]. RADAR systems for maritime traffic, per the Federal Communications Commission, are required to operate within a specific frequency range to avoid collisions at sea and along the coast. Specifically, these vessels often operate in the 3, 5, or 9 GHz range [25].

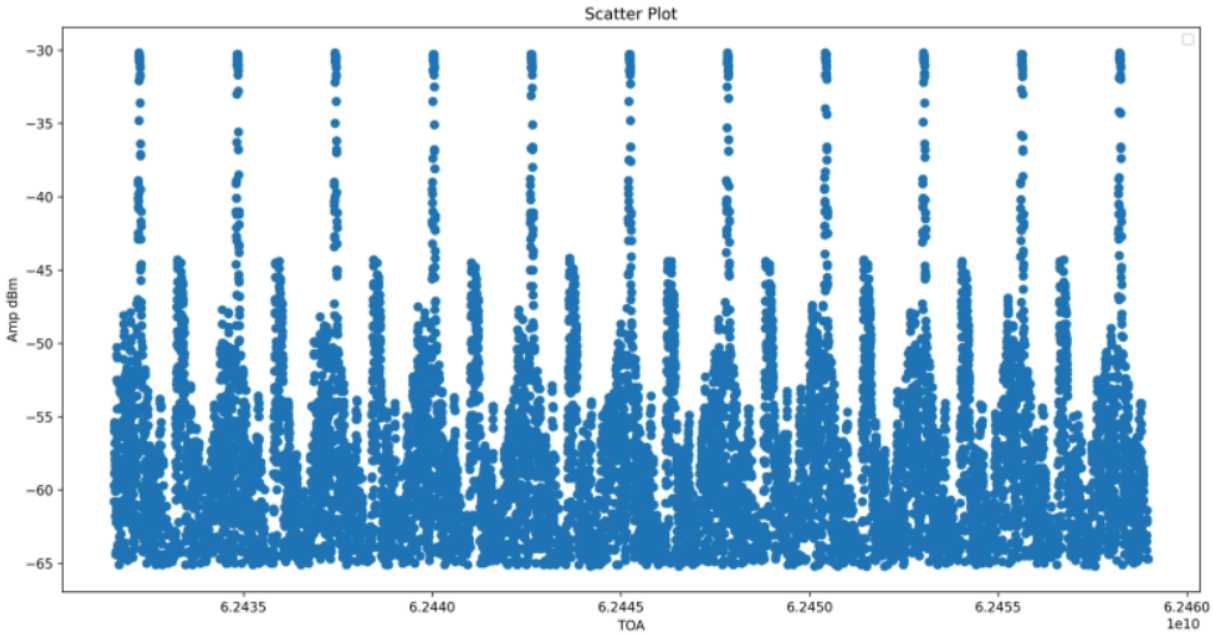

Figure 2. Scattergram plot of Amp vs TOA of pulse-data used for analysis.

Shown in Figure 3 is the visualization of a deinterleaving algorithm. When displayed as a scattergram of Amp vs TOA, RADAR scans are visually shown as peaks consisting of the associated RADAR pulses. Pulses that are not immediately recognized as a part of a peak can either be identified as associated with a RADAR scan or as noise in the data, both determined by statistical comparisons of other RADAR pulses. Figure 3 is the visualization of a deinterleaving algorithm by the human eye, and this is performed with computer code to thoroughly identify each RADAR pulse and construct RADAR scans.

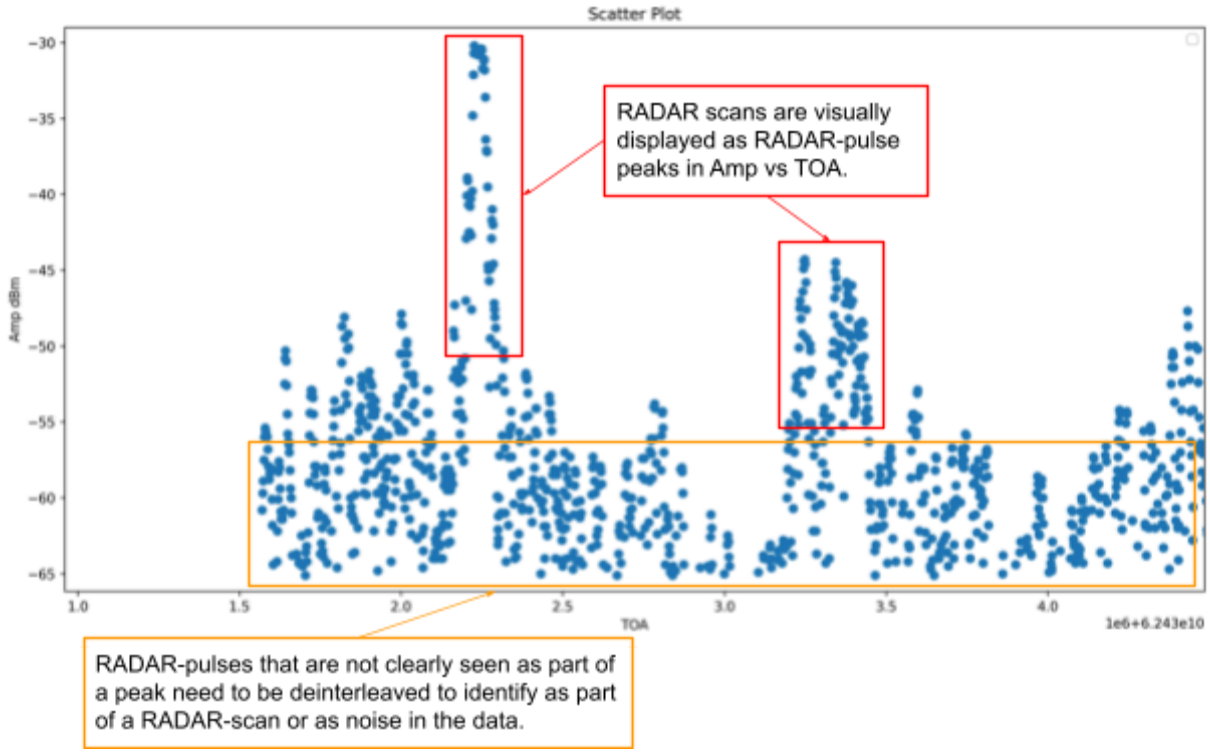

Figure 3. Visualization of deinterleaving a finite sampling of the RADAR pulse-data into RADAR scans.

Shown in Figure 4 is a scattergram of the data's RF vs PW. On this type of plot, RADAR scans are visually seen as distinct clusters made up of the associated RADAR pulses. However, this data set does not have any immediate clusters shown. As a result, the deinterleaving of the RADAR pulses is necessary to distinguish between the RADAR scans.

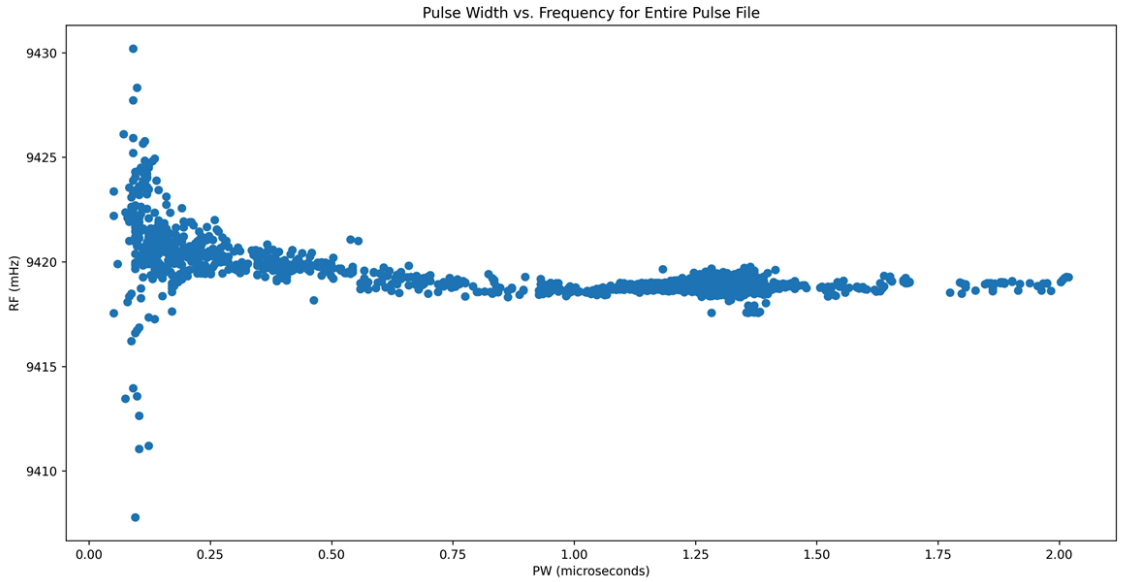

Figure 4. Scattergram plot of RF vs PW of pulse-data used for analysis.

3 Formulation of Deinterleaving Procedure

Presented in this section is a formal mathematical representation and flowchart ontology of our deinterleaving procedure. The relationship of our procedure to that used in other analyses remains for a future study. Our analysis, however, presents a specific software implementation of a deinterleaving procedure (see Appendix).

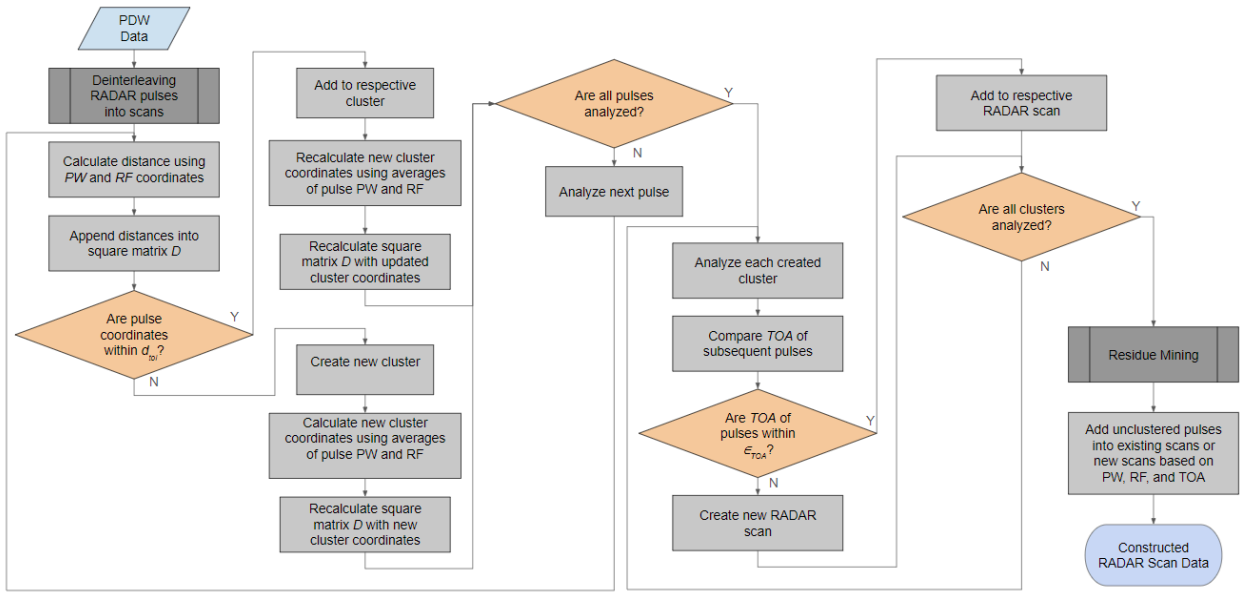

Figure 5. Flowchart ontology of deinterleaving procedure.

A general mathematical representation of our procedure is as follows. This representation is approximate, and remains for further refinement. Our procedure begins with construction of the matrix A, which contains n RADAR pulses and the corresponding data characteristics, i.e. times of arrivals (*TOA*), pulse frequencies (*RF*), pulse widths (*PW*), and pulse amplitudes (*Amp*), where n is the 10,001 RADAR pulses found in our data set. The matrix A is structured such that the RADAR pulses are ordered based on *TOA*.

$$A = [x_1 \; x_2 \; \dots \; x_n] \tag{1}$$

$$x_i = [TOA \;\; RF \;\; PW \; Amp] \tag{2}$$

Next, the distance between two RADAR pulses in the *PW*-*RF*-plane is calculated using the respective *RF* and *PW* parameters, visualized in Figure 6. The distance d_{ij} is calculated between RADAR pulses x_i and x_j, for every RADAR pulse in matrix A. As these distances d_{ij} are calculated, they populate the initial square matrix D of size n x n. For instance, the distance d_{12} is described as the distance between RADAR pulses x_1 and x_2. Elements along the matrix diagonal are equal to 0.

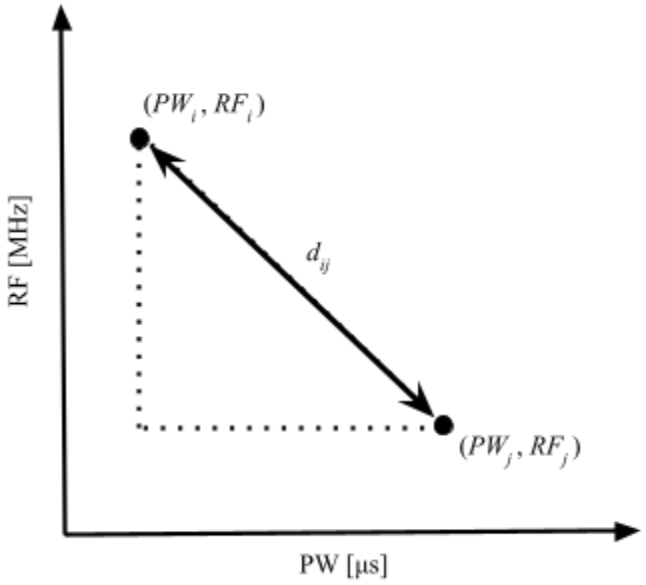

Figure 6. Visualization of distance between two RADAR pulses in the *PW*-*RF*-plane.

$$d_{ij} = \sqrt{(RF_i - RF_j)^2 + (PW_i - PW_j)^2} \tag{3}$$

$$\mathrm{D}=\begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ d_{21} & d_{22} & \cdots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & \cdots & d_{nn} \end{bmatrix} \tag{4}$$

Once an initial square matrix *D* is established, clusters can be made to begin grouping RADAR pulses of similar parameters. Clusters are defined as coordinates that are made up of the absolute averages of the *PW* and *RF* of the clustered RADAR pulses. An initial coordinate point x_{ab} is composed of two RADAR pulses x_a and x_b that have the smallest distance d_{ij}. The coordinates of x_{ab} are the absolute averages between the RADAR pulses' respective *PW* and *RF*.

$$x_{ab \mid min(D)} = \left(\frac{|PW_b - PW_a|}{2}, \frac{|RF_b - RF_a|}{2} \right) \tag{5}$$

With the seed of an initial cluster within the data established, the distance square matrix *D* is recalculated based around the initial coordinates x_{ab}. Every RADAR pulse is computationally compared to the initial coordinates x_{ab} as a distance measurement and populated into the distance square matrix *D*, where the size decreases to (*n*-1) x (*n*-1) after each iteration a RADAR pulse is appended into an existing cluster. For instance, the distance d_{iab} is described as

the distance between a RADAR pulse x_i and the RADAR pulse initial coordinates x_{ab}. Elements along the matrix diagonal are equal to 0.

$$\mathrm{D}=\begin{bmatrix} d_{abab} & d_{abj} & \cdots & d_{abn} \\ d_{iab} & d_{ij} & \cdots & d_{in} \\ \vdots & \vdots & \ddots & \vdots \\ d_{nab} & d_{nj} & \cdots & d_{nn} \end{bmatrix} \tag{6}$$

A distance tolerance d_{tol} is defined as the distance between the set tolerances of *PW* and *RF*, ϵ_{PW} and ϵ_{RF}, respectively. With seed cluster coordinates x_{ab} and the distance tolerance d_{tol}, there are two options for each calculated distance with RADAR pulse x_i.

$$d_{tol} = \sqrt{\epsilon_{RF}^2 + \epsilon_{PW}^2} \tag{7}$$

The first option is to append a new RADAR pulse into a current cluster. Using Equations (3) and (7), a RADAR pulse x_c is measured to determine it is within the distance tolerance d_{tol} of cluster coordinates x_{ab}. If the calculated distance is less than the distance tolerance d_{tol} and the minimum distance calculated in square matrix *D*, then this RADAR pulse x_c is appended into the corresponding current cluster. A new set of coordinates x_{abc} acts as the new initial seed coordinates for the cluster, and they are taken as the absolute averages between the *PW* and *RF* of x_{ab} and x_c. All subsequent RADAR pulses x_i are appended into the current cluster if the respective distance from the seed cluster coordinates is both less than the distance tolerance d_{tol} and is the minimum calculated distance within the square matrix *D* at this iteration. The distance square matrix *D* is recalculated for each additional RADAR pulse appended into a cluster such that new seed cluster coordinates are calculated, where the size of matrix *D* decreases to (*n*-1) x (*n*-1) after each iteration a RADAR pulse is appended into an existing cluster. For instance, the distance d_{iabc} is described as the distance between a RADAR pulse x_i and the RADAR pulse initial coordinates x_{abc}. Elements along the matrix diagonal are equal to 0.

$$d_{tol} > \sqrt{(RF_{ab} - RF_c)^2 + (PW_{ab} - PW_c)^2} \tag{8}$$

$$d_{min(D)} = \sqrt{(RF_{ab} - RF_c)^2 + (PW_{ab} - PW_c)^2} \tag{9}$$

$$x_{abc} = \left(\frac{|PW_c - PW_{ab}|}{2}, \frac{|RF_c - RF_{ab}|}{2} \right) \tag{10}$$

$$\mathrm{D}=\begin{bmatrix} d_{abcabc} & d_{abcj} & \cdots & d_{abcn} \\ d_{iabc} & d_{ij} & \cdots & d_{in} \\ \vdots & \vdots & \ddots & \vdots \\ d_{nabc} & d_{nj} & \cdots & d_{nn} \end{bmatrix} \tag{11}$$

The second option is if the distance between RADAR pulse x_c and the current cluster coordinates RADAR pulse x_{ab} do not satisfy both Equations (8) or (9). Using Equations (3) and (5), a new cluster is initialized with initial coordinate point x_{cd}, where it is composed of RADAR pulse x_c and another RADAR pulse x_d that have the distance d_{ij} that is less than the distance tolerance d_{tol} and the minimum calculated distance within the square matrix *D* at this iteration. The coordinates of x_{cd} are the absolute average between the RADAR pulses' respective *PW* and *RF*. The distance square matrix D is recalculated such that cluster coordinates x_{ab} and x_{cd} are now both being computationally compared with all remaining RADAR pulses x_i, where the size of matrix *D* decreases to (*n*-1) x (*n*-1) after each iteration a RADAR pulse is appended into an existing cluster. For instance, the distance d_{iabc} is described as the distance between a RADAR pulse x_i and the RADAR pulse initial coordinates x_{abc}. Elements along the matrix diagonal are equal to 0.

$$\mathrm{D}=\begin{bmatrix} d_{abab} & d_{abcd} & d_{abj} & \cdots & d_{abn} \\ d_{cdab} & d_{cdcd} & d_{cdj} & \cdots & d_{cdn} \\ d_{iab} & d_{icd} & d_{ij} & \cdots & d_{in} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{nab} & d_{ncd} & d_{nj} & \cdots & d_{nn} \end{bmatrix} \tag{12}$$

With these two options established, the distance d_{ij} for each RADAR pulse between each existing cluster and each unclustered RADAR pulse is compared to the distance tolerance d_{tol}. If the distance is less than the tolerance from a cluster and is the minimum calculated distance within the square matrix *D* at this iteration, the RADAR pulse x_i is appended into the respective cluster and the cluster coordinates are recalculated using the absolute averages between the *PW* and *RF* of the previous cluster coordinates and the RADAR pulse. If the distance is less than the tolerance from another RADAR pulse and is the minimum calculated distance within the square matrix *D* at this iteration, then a new cluster with initial coordinates are calculated using the absolute averages between the RADAR pulses' respective *PW* and *RF*. This iterative process continues until no calculated distances d_{ij} are less than the distance tolerance d_{tol} for the remaining unclustered RADAR pulses in the square matrix *D*.

Next, RADAR scans are constructed utilizing the established clusters. Within each cluster, the absolute difference between the *TOA* of subsequent RADAR pulses x_i and x_{i+1} are calculated then compared to a tolerance ϵ_{TOA}. If the calculated parameter associated with subsequent RADAR pulses is less than the tolerance ϵ_{TOA}, the following RADAR pulse x_i, made up of its TOA_i, RF_i, PW_i, and Amp_i is appended into the matrix B_j, representing a RADAR scan created within the data that is composed of *m* RADAR pulses.

$$\left|TOA_{i+1} - TOA_i\right| < \epsilon_{TOA} \tag{13}$$

$$B_j = \left[x_1 \; x_2 \; \dots \; x_m\right] \tag{14}$$

If the calculated parameter associated with subsequent RADAR pulses is greater than the tolerance ϵ_{TOA}, a new matrix B_{j+1} is constructed with this RADAR pulse appended. A minimum of 3 RADAR pulses constitute a RADAR scan; RADAR scans with less than 3 RADAR pulses are looped through to see if they can be added to an existing RADAR scan matrix B_j using the same tolerance comparison or if they should be discarded as noise in the data. This process is iterated for every existing cluster.

4 Initial Analysis

Presented in this section are the results of our analysis. These simulations consider general characteristics of pulse-data that one would find in tactical environments associated with maritime geolocation, such as the United States using ELINT to observe positions of Soviet ships in the past and of Chinese ships today [26, 27].

Shown in Figure 7 is a finite sampling of the RADAR scans generated from the deinterleaved RADAR pulses of the RADAR-pulse data. The column labeled “Final Burst ID” refers to the RADAR scan label after deinterleaving RADAR-pulse data. In this finite sampling, Scan 16, Scan 17, and Scan 18 are made up of 9, 10, and 11 RADAR pulses, respectively. Additionally, the given RADAR pulse characteristics *TOA*, *RF*, *PW*, and *Amp* are displayed for each RADAR pulse. Other data parameters are also given, but they are not utilized in any calculations toward deinterleaving the RADAR-pulse data.

	YMDHM	TOA	Freq	PW	Amp dBm	AOA	PRI	Pulse ID	OG Burst ID	Clock	Final Burst ID
1	SF	6.243236e+10	9418.93	1.187	-62.4	X-Band	0.000000	383	28	13.593772	16
2	SF	6.243236e+10	9418.89	1.211	-58.3	X-Band	1631.223991	384	28	13.593772	16
3	SF	6.243237e+10	9418.93	1.219	-55.7	X-Band	1685.600006	385	28	13.593772	16
4	SF	6.243237e+10	9418.93	1.219	-56.3	X-Band	1644.832001	386	28	13.593772	16
5	SF	6.243237e+10	9418.90	1.235	-59.7	X-Band	1699.183998	387	28	13.593772	16
6	SF	6.243237e+10	9418.92	1.227	-59.9	X-Band	1658.419998	388	28	13.593772	16
7	SF	6.243237e+10	9418.95	1.223	-60.6	X-Band	1712.792000	389	28	13.593772	16
8	SF	6.243237e+10	9418.93	1.219	-60.8	X-Band	1672.015999	390	28	13.593772	16
9	SF	6.243238e+10	9418.96	1.219	-59.7	X-Band	1726.392006	391	28	13.593772	16

	YMDHM	TOA	Freq	PW	Amp dBm	AOA	PRI	Pulse ID	OG Burst ID	Clock	Final Burst ID
1	SF	6.243238e+10	9418.92	1.215	-59.7	X-Band	0.000000	392	29	4.91454	17
2	SF	6.243238e+10	9418.88	1.215	-54.7	X-Band	1719.579994	393	29	4.91454	17
3	SF	6.243238e+10	9418.85	1.215	-52.0	X-Band	1651.624001	394	29	4.91454	17
4	SF	6.243239e+10	9418.88	1.219	-51.8	X-Band	1706.000000	395	29	4.91454	17
5	SF	6.243239e+10	9418.91	1.215	-52.2	X-Band	1638.015999	396	29	4.91454	17
6	SF	6.243239e+10	9418.92	1.215	-52.1	X-Band	1692.396004	397	29	4.91454	17
7	SF	6.243239e+10	9418.97	1.215	-53.1	X-Band	1631.219994	398	29	4.91454	17
8	SF	6.243239e+10	9419.03	1.219	-54.6	X-Band	1685.620003	399	29	4.91454	17
9	SF	6.243239e+10	9419.08	1.219	-56.1	X-Band	1644.807999	400	29	4.91454	17
10	SF	6.243240e+10	9419.09	1.259	-60.8	X-Band	1700.431999	401	29	4.91454	17

	YMDHM	TOA	Freq	PW	Amp dBm	AOA	PRI	Pulse ID	OG Burst ID	Clock	Final Burst ID
1	SF	6.243240e+10	9418.97	1.243	-61.6	X-Band	0.000000	402	30	6.796967	18
2	SF	6.243241e+10	9419.02	1.239	-59.6	X-Band	1731.995995	403	30	6.796967	18
3	SF	6.243241e+10	9419.01	1.239	-59.3	X-Band	1665.228004	404	30	6.796967	18
4	SF	6.243241e+10	9418.97	1.235	-58.8	X-Band	1719.563995	405	30	6.796967	18
5	SF	6.243241e+10	9418.90	1.223	-58.0	X-Band	1651.643997	406	30	6.796967	18
6	SF	6.243241e+10	9418.87	1.227	-57.0	X-Band	1706.000000	407	30	6.796967	18
7	SF	6.243241e+10	9418.85	1.227	-55.6	X-Band	1638.016006	408	30	6.796967	18
8	SF	6.243242e+10	9418.84	1.223	-54.1	X-Band	1692.411995	409	30	6.796967	18
9	SF	6.243242e+10	9418.86	1.223	-54.1	X-Band	1631.228004	410	30	6.796967	18
10	SF	6.243242e+10	9418.89	1.223	-55.8	X-Band	1685.584000	411	30	6.796967	18
11	SF	6.243242e+10	9418.88	1.215	-60.0	X-Band	1644.891998	412	30	6.796967	18

Figure 7. Finite sampling of RADAR scans generated from deinterleaved RADAR pulses of the data.

Shown in Figure 8 is a finite sampling plot of RADAR scans generated from the deinterleaved RADAR pulses of the RADAR-pulse data. The x-axis is TOA in seconds after unix time and the y-axis is Amp in dBm. Each data point corresponds to a RADAR pulse with its associated *TOA* and *Amp*. Based on the preceding deinterleaving analysis procedure, RADAR pulses that are within the same RADAR scan are categorized by color. Visually, the RADAR pulses form peaks in the plot; each peak constitutes a generated RADAR scan from the deinterleaved RADAR-pulse data.

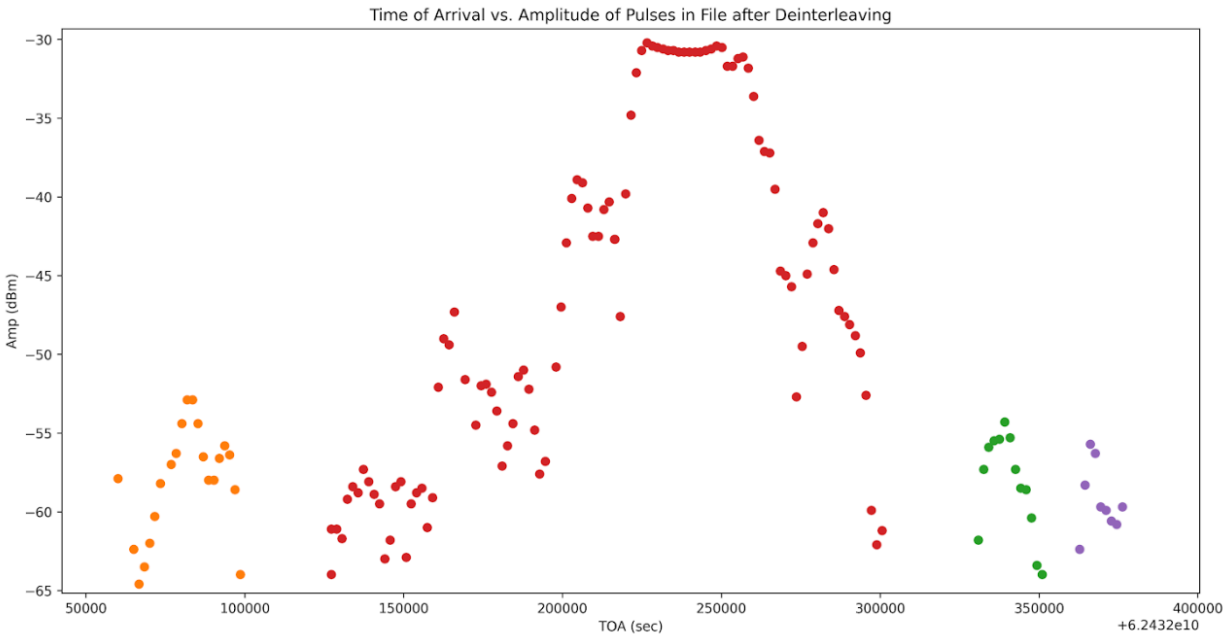

Figure 8. Finite sampling plot of RADAR scans generated from deinterleaved RADAR-pulses. Each color represents a RADAR scan, and they are visualized as peaks in the plot.

Shown in Figure 9 is a plot of the RADAR scans generated from the deinterleaved RADAR pulses of the RADAR-pulse data. The x-axis is TOA in seconds after unix time and the y-axis is Amp in dBm. Each data point corresponds to a RADAR pulse with its associated *TOA* and *Amp*. Based on the preceding deinterleaving analysis procedure, RADAR scans are categorically separated by color. Visually, the RADAR pulses form peaks in the plot that represent RADAR scans.

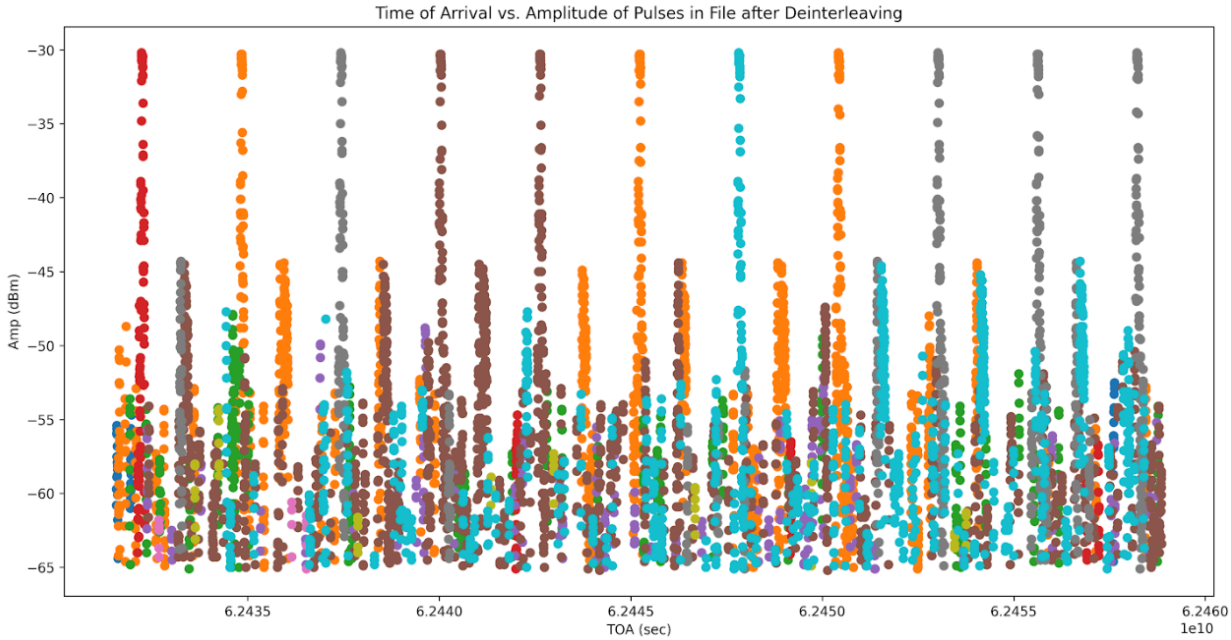

Figure 9. Plot of RADAR scans generated from deinterleaved RADAR-pulses. Each color represents a RADAR scan, and they are visualized as peaks in the plot.

Shown in Figure 10 is a residue mining summary of the RADAR-pulse data. The summary displays the total number of RADAR pulses in the data, the total number of RADAR scans generated after deinterleaving, the number of RADAR pulses used to develop these RADAR scans, and the number and proportion of RADAR scans generated with less than 4 RADAR pulses. The purpose of residue mining is to identify if any RADAR pulses in our data set is discarded as noise, due to the parameter constraints set in Equations (7) and (13) within our deinterleaving analysis procedure. Discarded RADAR pulses can be added to an existing RADAR scan matrix B_j or they can be added to a new RADAR scan matrix B_{j+1} with other previously-discarded RADAR pulses, using Equations (3), (8), (9), (13), and (14). Figure 11 is a histogram comparing the number of scans associated with constructed RADAR scans and residue noise, highlighting the approximated 5% of noise generated from the data set. Additionally, Figure 12 is a histogram visualizing the number of pulses contained in each constructed RADAR scan.

```
Total pulses: 10001
Total scans: 1016
Scans kept: 666
Pulses contained in scans: 9451
Proportion pulses kept in scans: 0.945005499450055
Scans with less than four pulses: 350
Proportion of scans with less than four pulses: 0.34448818897637795
Number of pulses in largest scan: 114
Which scan is largest: 157
Number of pulses in smallest scan: 4
Which scan is smallest: 25
```

Figure 10. Residue mining summary of RADAR-pulse data based on deinterleaving analysis procedure.

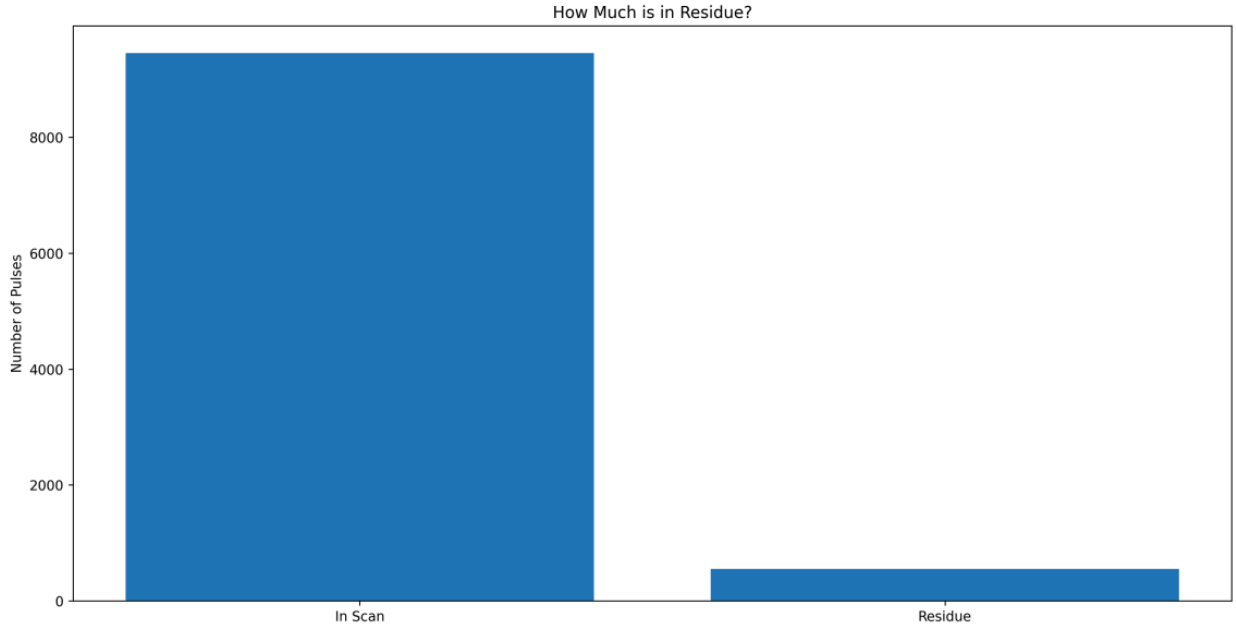

Figure 11. Histogram of RADAR Pulses in Constructed RADAR Scans and in Noise Residue.

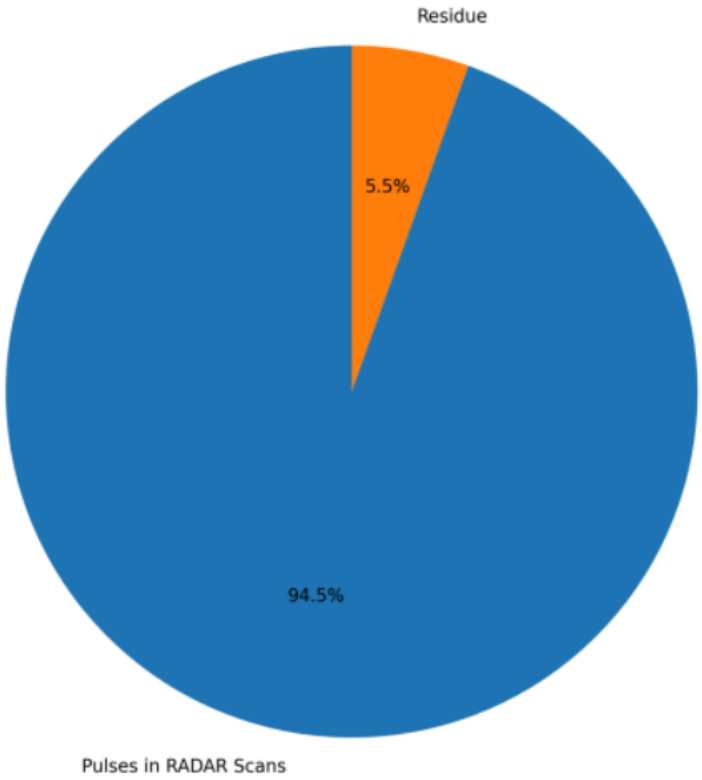

Figure 12. Pie Chart of RADAR Pulses in Constructed RADAR Scans and in Noise Residue.

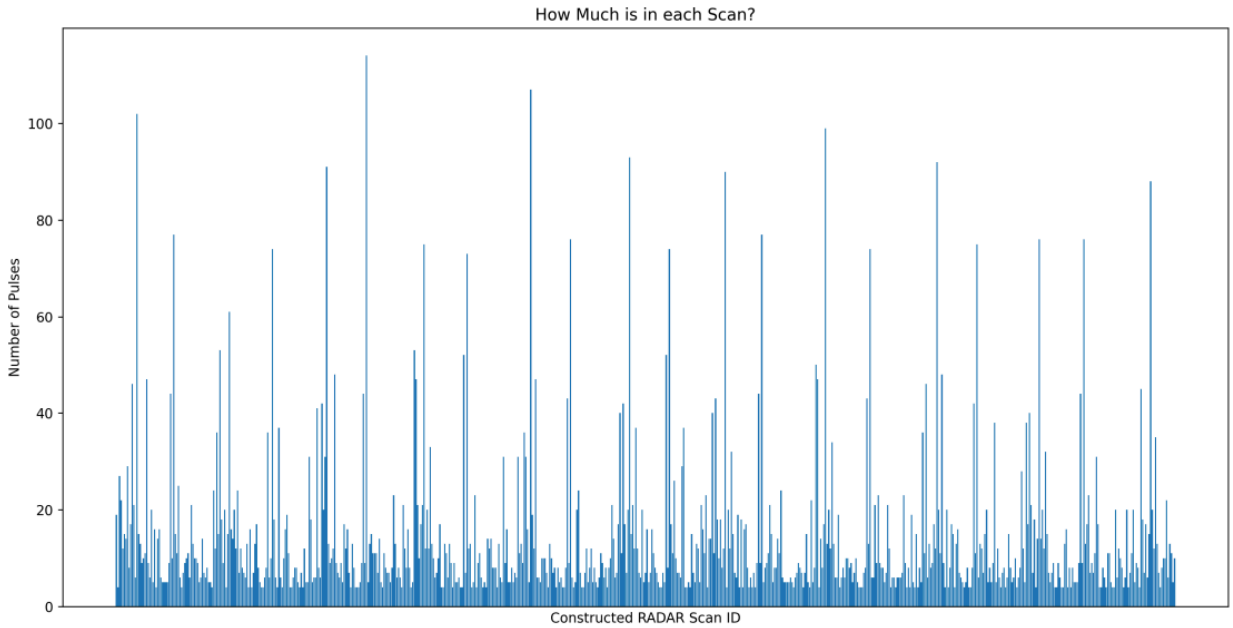

Figure 13. Histogram of RADAR Pulses Associated with Constructed RADAR Scans.

5 Case Study Analyses

Presented in this section are the results of our case study analyses. Similar to the results found in the previous section, these simulations consider general characteristics of pulse-data that one would find in tactical environments associated with maritime geolocation.

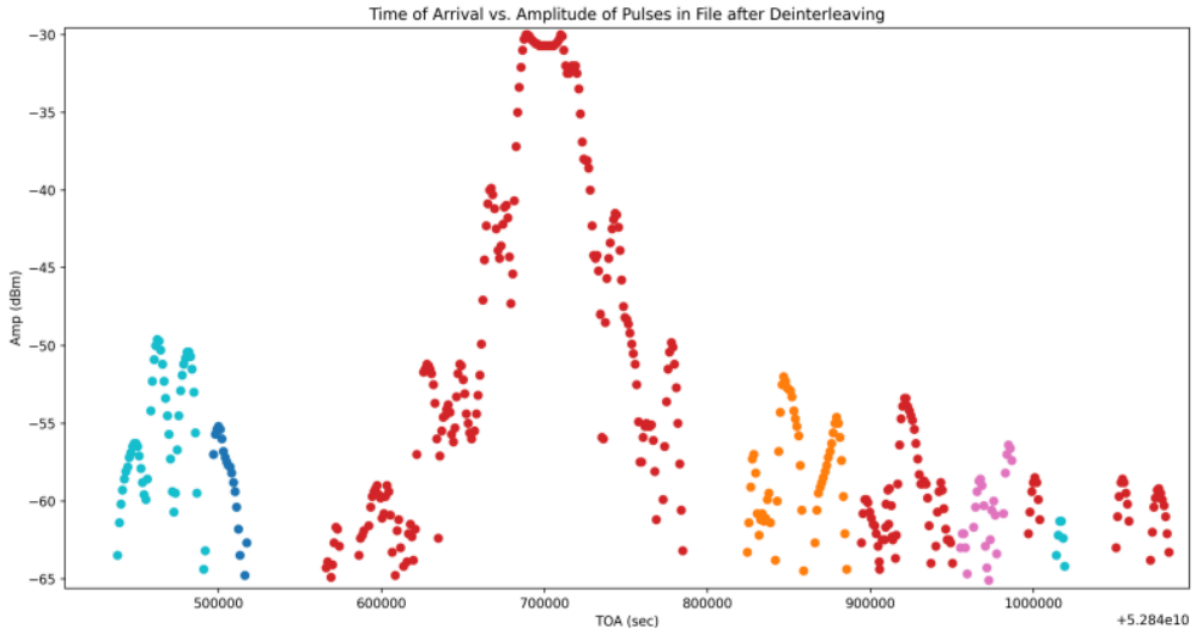

Figure 14. Finite sampling plot of RADAR scans generated from deinterleaved RADAR-pulses for Case Study 2. Each color represents a RADAR scan, and they are visualized as peaks in the plot.

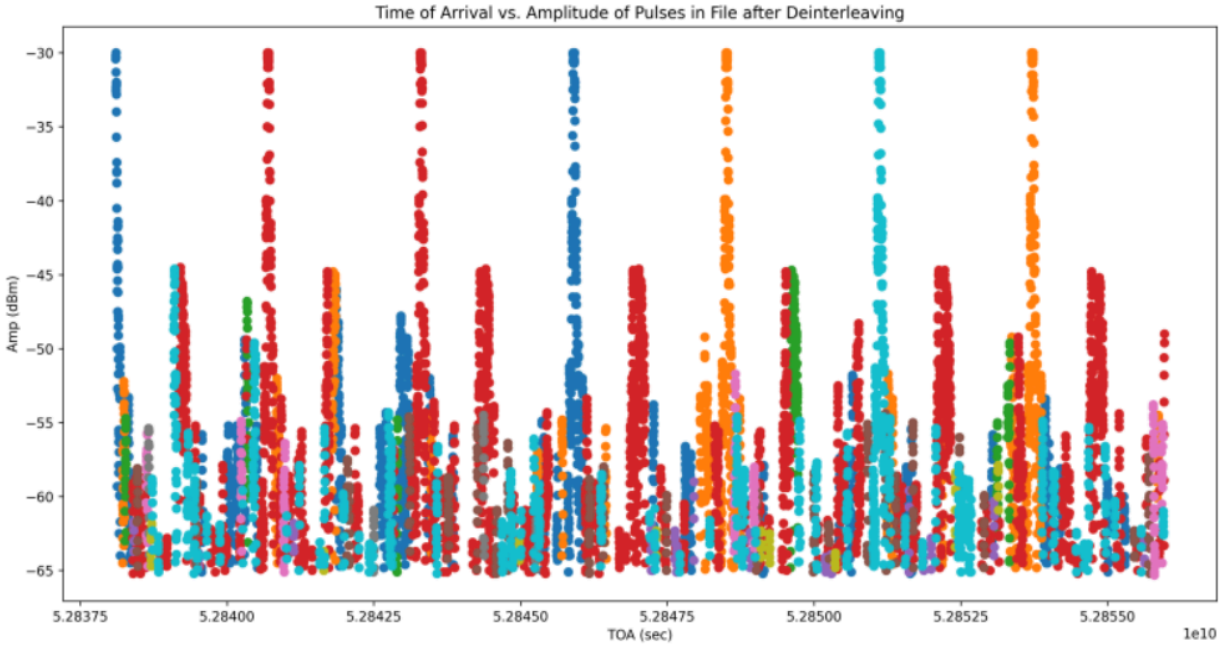

Figure 15. Plot of RADAR scans generated from deinterleaved RADAR-pulses for Case Study 2. Each color represents a RADAR scan, and they are visualized as peaks in the plot.

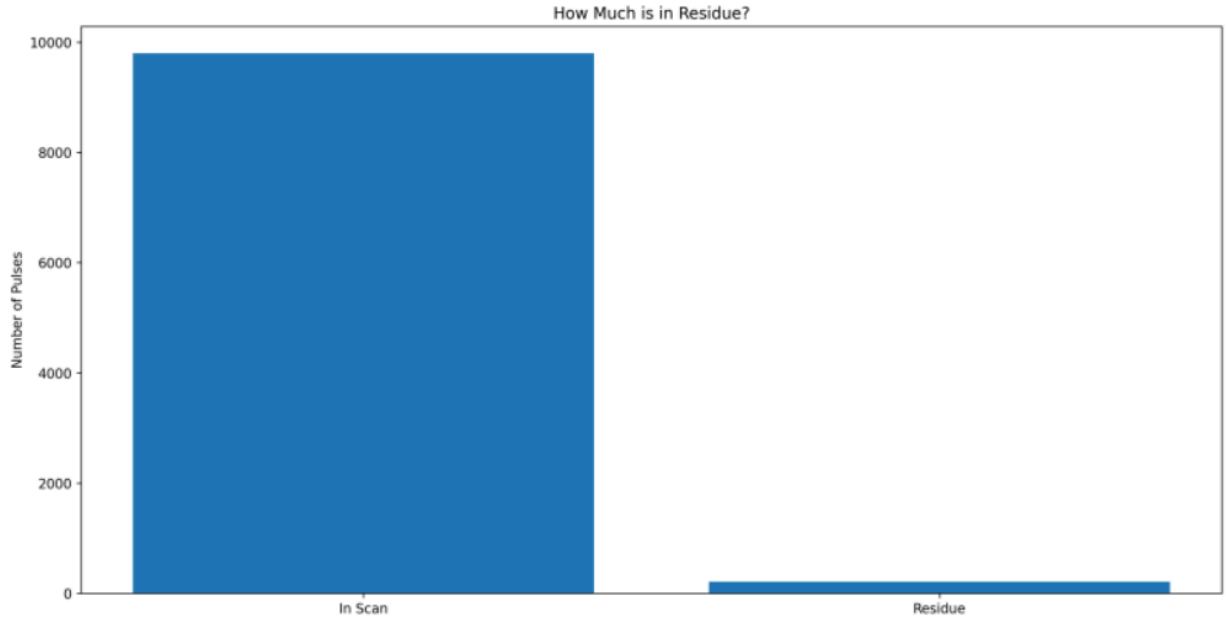

Figure 16. Histogram of RADAR Pulses in Constructed RADAR Scans and in Noise Residue for Case Study 2.

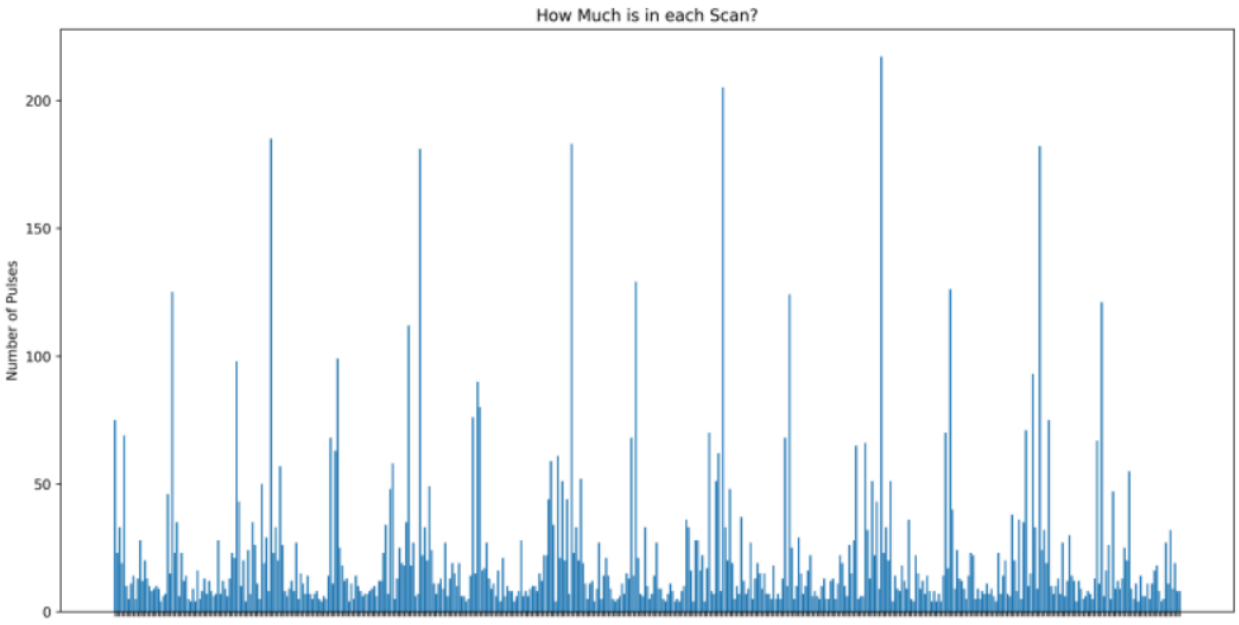

Figure 17. Histogram of RADAR Pulses Associated with Constructed RADAR Scans for Case Study 2.

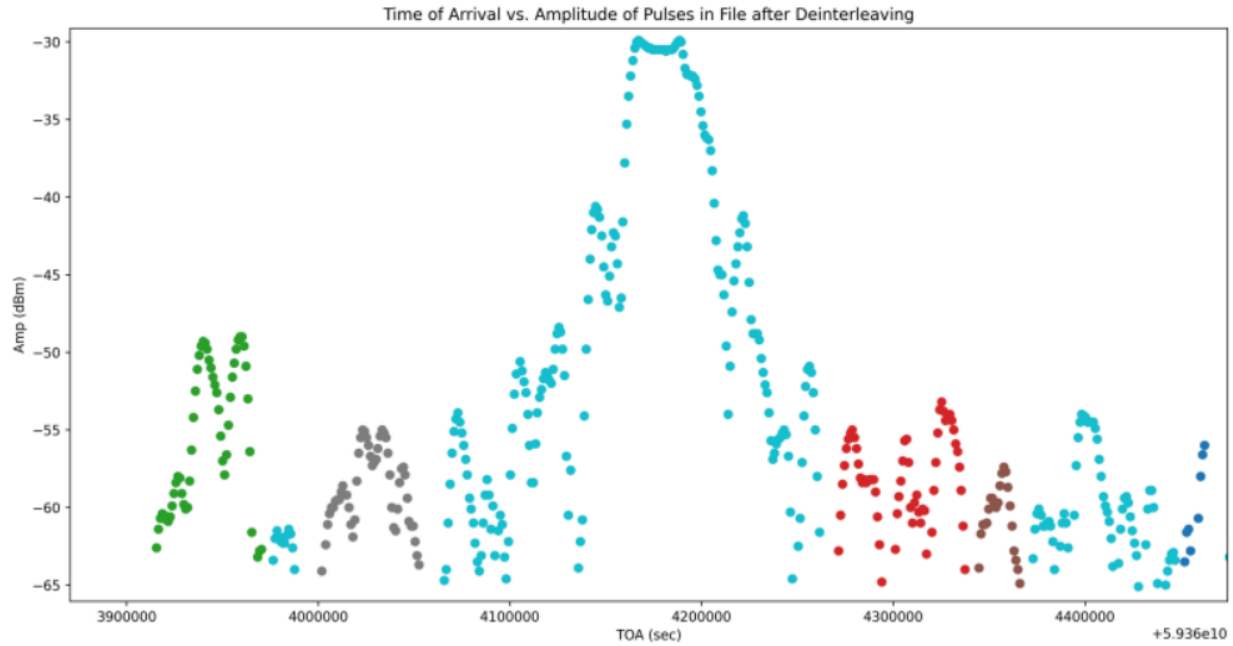

Figure 18. Finite sampling plot of RADAR scans generated from deinterleaved RADAR-pulses for Case Study 3. Each color represents a RADAR scan, and they are visualized as peaks in the plot.

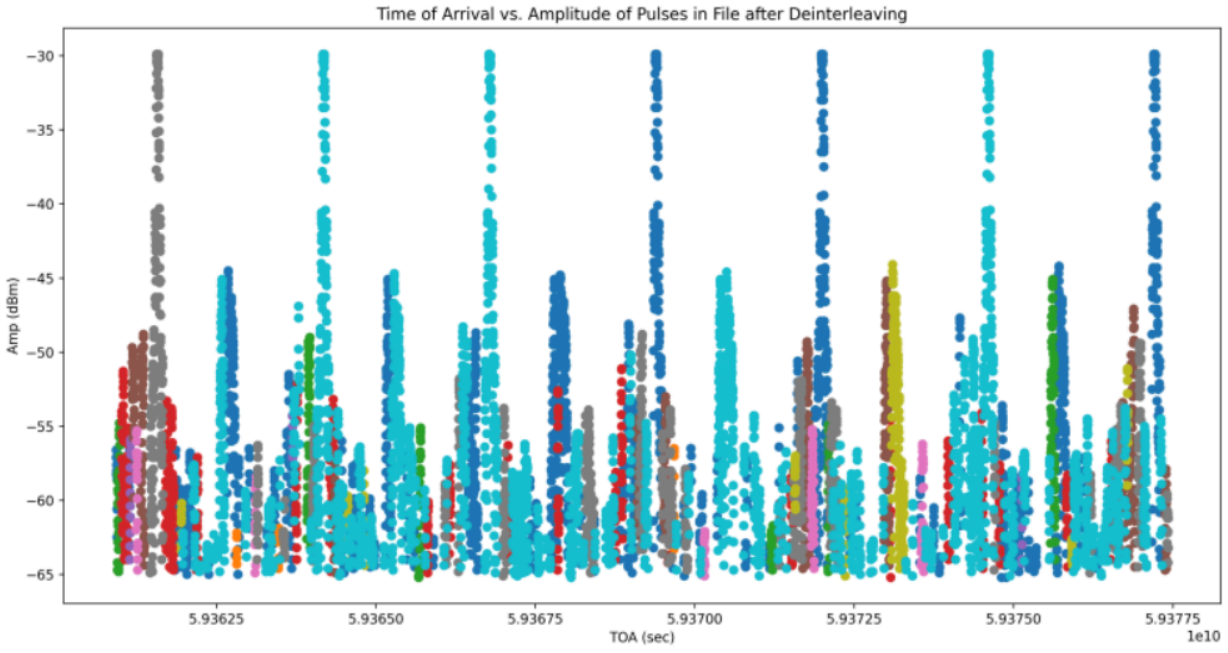

Figure 19. Plot of RADAR scans generated from deinterleaved RADAR-pulses for Case Study 3. Each color represents a RADAR scan, and they are visualized as peaks in the plot.

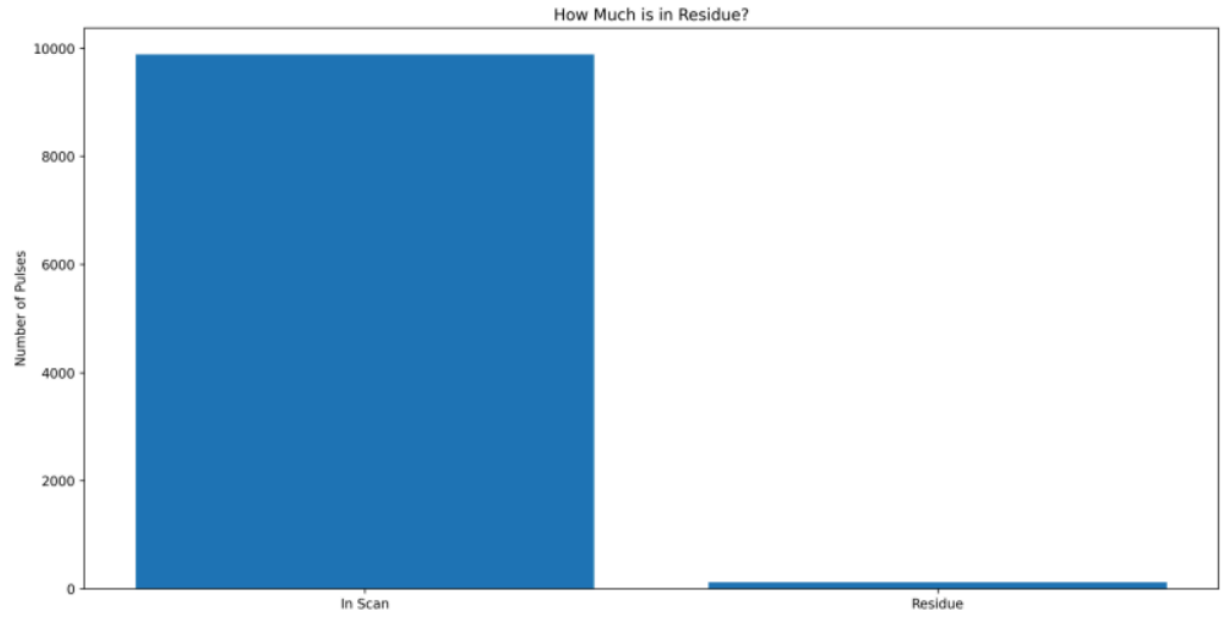

Figure 20. Histogram of RADAR Pulses in Constructed RADAR Scans and in Noise Residue for Case Study 3.

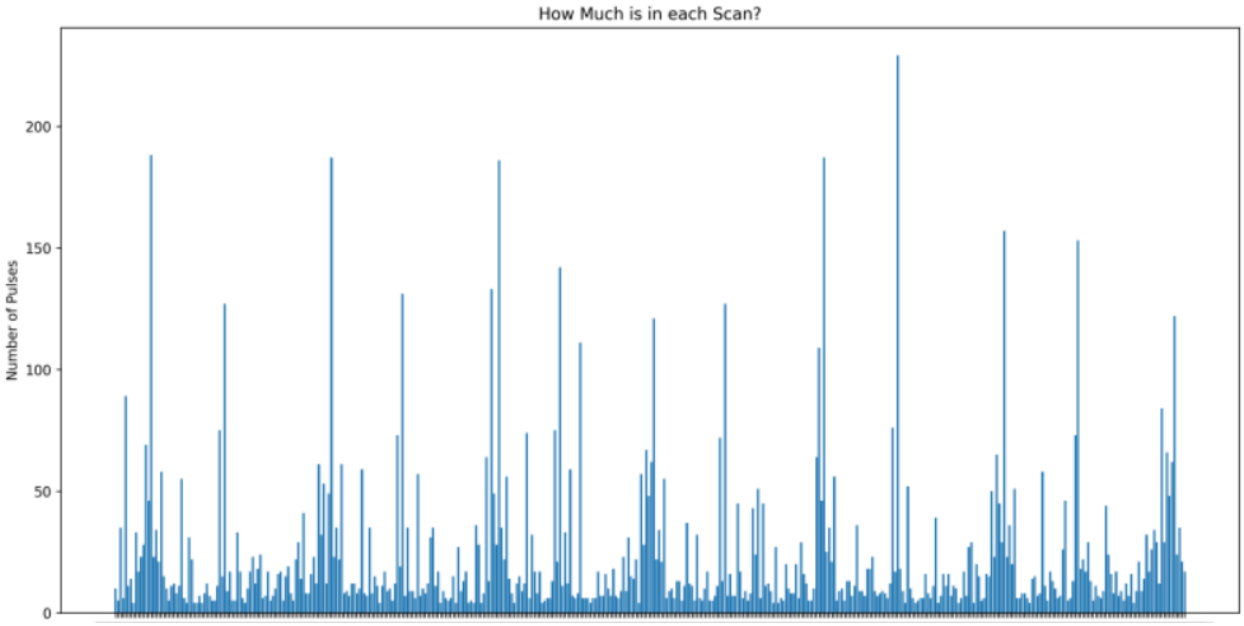

Figure 21. Histogram of RADAR Pulses Associated with Constructed RADAR Scans for Case Study 3.

6 Discussion

The focus of this report is specific software implementation of a general algorithm formulation for deinterleaving of RADAR scans, with intention to further deinterleave these

scans into RADAR signals. Appropriately, the results presented in this report should be considered within the context of software verification and validation (see references [28, 29]), where, in general, "verification" is concerned with identifying and removing errors in the model by comparing numerical solutions to analytical or highly accurate benchmark solutions, and "validation" is concerned with quantifying the accuracy of the model by comparing numerical solutions to experimental data. Accordingly, the results presented here represent an initial verification of a software implementation (see Appendix). More quantitative verification and validation of the software implementation poses a list of problems to be addressed for further analysis and software development. A general framework, based on inverse-analysis concepts, providing context for these posed problems is described in Figure 22. These problems are as follows.

- What are aspects of the algorithm's formal mathematical foundation that should be examined for further model, and associated software development (references [28, 29])?
- What are the algorithm's and associated software's performance as applied to more expansive case studies, considering "measured" RADAR scans, having wide ranging pulse characteristics?
- What are the algorithm's and associated software's performances as applied to "synthetic" RADAR scans, where pulse error and artifacts are modeled and quantified, for assessing algorithm and software robustness in terms of error-tolerance?

The problem of quantifying algorithm performance in terms of error-tolerance using synthetic RADAR poses yet another problem, which is that of constructing realistic simulators of emitter data for algorithm input (see Figure 22), e.g., see reference [30].

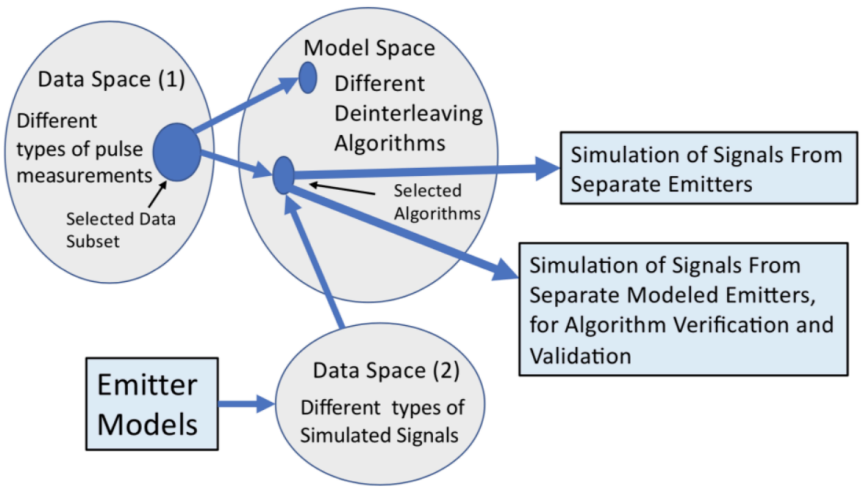

Figure 22. Mappings between data, model, simulation spaces and emitter models for inverse-analysis.

Further studies in maritime geolocation will utilize these newly constructed RADAR scan-data, where the processed data set can be further deinterleaved to construct RADAR

signals. Each identified RADAR signal is indicative of an emitter in the maritime-tactical environment. The deinterleaving of the RADAR scan-data uses PRI-modulated pattern recognition and a clock metric calculation. Recognition of a PRI-modulation pattern is critical in constructing RADAR signals because a RADAR signal is composed of PRIs that stay consistent with firing-order in the maritime-tactical environment. When there is an identified PRI-pattern within a RADAR scan, TOA-based subsequent scans can be compared to examine a continuation of the initial PRI-pattern, in addition to similarities among RF and PW among RADAR scans. The clock metric value is based on the internal clock of a RADAR emitter. Inside of an emitter is a physical clock that dictates how often a RADAR pulse is emitted, and it has a programmed parameter called a clock metric that is the greatest common denominator among an emitter's PRIs [31]. All emitted PRIs will have some consistent factor that originates from this clock metric. In combination with an identified PRI-modulated pattern in the processed-data, a clock metric can be calculated to cluster RADAR scans in order to construct RADAR signals for maritime geolocation.

7 Conclusion

This study describes a procedure for deinterleaving in terms of a specific software implementation, and initial analysis for application of this procedure and its software implementation. Refinement of this procedure and its software implementation for further analysis and its examination, in terms of formulation, with respect to other procedures remains for future study.

Acknowledgements

This work is supported by the U.S. Military and funded through the U.S. Naval Research Laboratory SSEP Training Program. We thank Dr. James W. Stoner for his contributions to this report in regard to ELINT processing and analysis.

References

[1] Wiley, R. G. (1982). *Electronic Intelligence: The Analysis of Radar Signals*. Artech House.

[2] Wiley, R. G. (1985). *Electronic Intelligence: The Interception of Radar Signals*. Artech House.

[3] Xie, M., Zhao, C., Zhao, Y., Hu, D., Wang, Z. (2022) A novel method for deinterleaving RADAR signals: First-order difference curve based on sorted TOA difference sequence. *IET Signal Processing, 17*(1), 1-13. https://doi.org/10.1049/sil2.12162

[4] Manly, B. F., & Alberto, J. N. (2016). Cluster Analysis. In *Multivariate Statistical Methods* (4th ed., pp. 58–63). Chapman & Hill.

[5] Wiley, R. G. (2006). *ELINT: The Interception and Analysis of RADAR Signals*. Artech House.

[6] Kirsch, A. (1996). An Introduction to the Mathematical Theory of Inverse Problems. *Applied Mathematical Sciences*. https://doi.org/10.1007/978-1-4612-5338-9

[7] Tarantola, A. (2005). *Inverse Problem Theory and Methods for Model Parameter Estimation*. Society for Industrial and Applied Mathematics.

[8] Colton, D., & Kress, R. (2020). *Inverse Acoustic and Electromagnetic Scattering Theory* (4th ed.). Springer.

[9] Wilson, P. (2017). ARMMS RF & Microwave Society Conference. In *It's a Complex World 'Radar Deinterleaving.'* Slipstream Design.

[10] Aslan, M. M. (2006). *Emitter Identification Techniques in Electronic Warfare* (thesis). Middle East Technical University.

[11] Mardia, H. K. (1988). IEE Colloquium on Signal Processing for ESM Systems. In *Adaptive Multi-Dimensional Clustering for ESM*. Institution of Engineering and Technology.

[12] Guven, E. (1994). *Emitter Identification by the Clustering Techniques* (thesis). Middle East Technical University.

[13] Guvenlik, A. R. (1999). *Clustering Techniques for Emitter Identification in Electronic Warfare* (thesis). Middle East Technical University.

[14] Keshavarzi, M., Amiri, D., Pezeshk, A. M., & Farzaneh, F. (2014). A Novel Method of Deinterleaving Pulse Repetition Interval Modulated Sparse Sequences in Noisy Environments. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, *E97.A*(5), 1136–1139. https://doi.org/10.1587/transfun.e97.a.1136

[15] Mardia, H. K. (1989). New Techniques for the Deinterleaving of Repetitive Sequences. *IEE Proceedings F Radar and Signal Processing*, *136*(4), 149–154. https://doi.org/10.1049/ip-f-2.1989.0025

[16] Milojević, D. J., & Popović, B. M. (1992). Improved Algorithm for the Deinterleaving of Radar Pulses. *IEE Proceedings F Radar and Signal Processing*, *139*(1), 98–104. https://doi.org/10.1049/ip-f-2.1992.0012

[17] Liu, Y., & Zhang, Q. (2018). Improved Method for Deinterleaving Radar Signals and Estimating PRI Values. *IET Radar, Sonar & Navigation*, *12*(5), 506–514. https://doi.org/10.1049/iet-rsn.2017.0516

[18] Xiao, W. H., Wu, H. C., Yang, C. Z., & Wang, M. L. (2013). An Improved SDIF RADAR Pulse Signal Main Sorting Algorithm. *Advanced Materials Research*, *710*, 637–641. https://doi.org/10.4028/www.scientific.net/amr.710.637

[19] Kuang, Y., Qingbo, S., Qianbin, C., Li, Y., & Keping, L. (2005). A Novel SDIF-based PRI Estimation Approach to Deinterleave Repetitive Pulse Sequences. *WSEAS Transactions on Mathematics*, *4(3)*, 260–265.

[20] National Reconnaissance Office Review and Redaction Guide for Automatic Declassification of 25-Years-Old Information, Version 1.0, 2008 Edition. NRO Approved for Release 16 Dec 2010. Stoner, James W. Pioneered techniques for near-real-time processing of electronic intelligence signals and whose algorithm prototyping and quality control were critical to satellite programmatic successes. Designated NRO Pioneer, 2000. Consult Center for the Study of National Reconnaissance for additional information.

[21] Campello, R. J., Moulavi, D., & Sander, J. (2013). Density-Based Clustering Based on Hierarchical Density Estimates. *Advances in Knowledge Discovery and Data Mining*, 160–172. https://doi.org/10.1007/978-3-642-37456-2_14

[22] Chakraborty, S., Paul, D., & Das, S. (2020). Hierarchical Clustering with Optimal Transport. *Statistics & Probability Letters*, *163*, 108781. https://doi.org/10.1016/j.spl.2020.108781

[23] Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1987). 2nd International Conference on Knowledge Discovery and Data Mining. In *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise* (pp. 53–65).

[24] Mottier, M., Chardon, G., & Pascal, F. (2021). IEEE RADAR Conference. In *Deinterleaving and Clustering Unknown RADAR Pulses*.

[25] FCC Office of Engineering and Technology Policy and Rules Division. (2022). *FCC Online Table of Frequency Allocations*. Federal Communications Commission. https://transition.fcc.gov/oet/spectrum/table/fcctable.pdf

[26] Kraska, J. (2022). Intelligence Collection and the International Law of the Sea. *International Law Studies*, *99*(2375–2831). https://digital-commons.usnwc.edu/cgi/viewcontent.cgi?article=3023&context=ils

[27] Valencia, M. J. (2021). *Opinion: US-China Surveillance Contest in South China Sea is Too Risky*. South China Morning Post. https://www.scmp.com/comment/opinion/article/3133329/us-china-race-surveillance-supremacy-south-china-sea-risks-needless

[28] Wikimedia Foundation. (2024, January 24). *Software verification and validation*. Wikipedia. https://en.wikipedia.org/wiki/Software_verification_and_validation

[29] Pham, H. (2000). *Software Reliability* (pp. 567). Springer-Verlag.

[30] Mutonkole, N., Samuel, E. R., de Villiers, D. I., & Dhaene, T. (2016). Parametric Modeling of Radiation Patterns and Scattering Parameters of Antennas. *IEEE Transactions on Antennas and Propagation*, *64*(3), 1023–1031. https://doi.org/10.1109/tap.2016.2521883

[31] Wolff, C. (n.d.). *Radar Basics*. Radartutorial.

https://www.radartutorial.eu/01.basics/The%20Radar%20Trigger.en.html

Appendix:

Given in this appendix is a computer program for application of deinterleaving procedure, which is written in Python.

Pulse Deinterleaver:

```
#import all of our other files for use
import deinterleave as di
#import Bins as bi
#import clock as cl
#import patternanalysis as pa
#import augment as au
#import distribution as dis
#import counts as cnt
import converter as con
#import special_pattern as patt
#import special_augment as sp
import scan_deinterleave as sc
#from fpdf import FPDF
import matplotlib.pyplot as plt
import sys
import Clustering as clu

#Hello
def main():
    file = input("Enter full file name in folder: ") #take the name of the file to analyze
    file = con.convert(file) #pass file into the converter
    value = input("Deinterleave file (Y/N): ") #decide if we want to deinterleave the file or not
    print(file)
    if value == 'Y':
        data = clu.clustering(file)
        #data = di.deinterleave(file) #If yes, deinterleave
        data = 'combined.csv'
        data = sc.scan(data)
    else:
        data = file #otherwise, just take the file as the data
    #data.to_csv('why.csv')
    #These are all of the list that we are collecting our data in
```

```
    sys.exit()

def clustering(file):
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import matplotlib.cm as cm
    import sys
    import Bins as bi
    import clock as cl
    import os

    '''
    if file.shape[0] >= 10001:
        file = file.head(10001)
    '''
    total_pulses = file.shape[0]
    data = file

    def fun(num):
        return num + 8400

    #only add this line for EMU
    #data['Freq'] = data['Freq'].apply(fun)
    RF_data = data['Freq'].tolist()
    PW_data = data['PW'].tolist()
    pulses = list(range(1, len(RF_data)+1))
    data["Pulse ID"] = pulses


    plt.scatter(file['PW'], file['Freq'])

    plt.xlabel('PW (microseconds)')
    plt.ylabel('RF (mHz)')
    plt.title('Pulse Width vs. Frequency for Entire Pulse File')


    plt.show()

    plt.scatter(file['TOA'], file['Amp dBm'])
```

```
plt.xlabel('TOA (sec)')
plt.ylabel('Amp (dBm)')
plt.title('Time of Arrival vs. Amplitude of Pulses in File before Deinterleaving')

plt.legend()

plt.show()


#make out RF-PD coordinates
pulse_points_og = []
for i in range(len(RF_data)):
    pair = [PW_data[i], RF_data[i], i]
    pulse_points_og.append(pair)


def split(list_a, chunk_size):
    for i in range(0, len(list_a), chunk_size):
        yield list_a[i:i + chunk_size]

chunks = list(split(pulse_points_og, 500))
if len(chunks[-1]) <= 3:
    for i in chunks[-1]:
        chunks[-2].append(i)
    del chunks[-1]

counter = 1
min_value = 0
clusters = []
for i in chunks:
    pulse_points = i
    counter = 1
    min_value = 0
    while min_value < 5.02494:
        arr = []
        for row in pulse_points:
            curr = []
            for point in pulse_points:
                dist = ((point[0]-row[0])**2 + (point[1]-row[1])**2)**(1/2)
```

```
            curr.append(dist)
        arr.append(curr)
    distance = np.array(arr)
    distance[distance == 0] = 10000000
    min_index = np.unravel_index(np.argmin(distance), distance.shape)

    # find our points to add in a cluster
    point1 = pulse_points[min_index[0]]
    point2 = pulse_points[min_index[1]]
    con = 0
    if len(point1) == 4:
        con += 1
    if len(point2) == 4:
        con -= 1
    if (len(point1) == 4) and (len(point2) == 4) and (con == 0):
        cluster_add_to = point1[3]
        for point in clusters[point2[3]]:
            #print(clusters[cluster_add_to])
            clusters[cluster_add_to].append(point)

        clusters[point2[3]].clear()
        #find the averages
        average_PW = (sum(point[0] for point in clusters[cluster_add_to])) /
len(clusters[cluster_add_to])
        average_RF = (sum(point[1] for point in clusters[cluster_add_to])) /
len(clusters[cluster_add_to])
        average_PID = (sum(point[2] for point in clusters[cluster_add_to])) /
len(clusters[cluster_add_to])
        delete = [min_index[0] , min_index[1]]
        #delete = [point1[3] , point2[3]]
        '''
        for i in delete:
            for j in range(len(pulse_points)):
                point = pulse_points[j]
                if (len(point) == 4) and (i == point[3]):
                    del pulse_points[j]
                    break
        '''
        for index in sorted(delete, reverse=True):
            del pulse_points[index]
```

```
            point_add = [average_PW , average_RF, average_PID, cluster_add_to]
            pulse_points.append(point_add)
            if len(pulse_points) == 1:
                min_value = 1000
            else:
                min_value = np.min(distance)
                if min_value > 5.02494:
                    for point in pulse_points:
                        if len(point) == 3:
                            to_add = [point]
                            clusters.append(to_add)

        elif (len(point1) == 4) and (con == 1):
            cluster_add_to = point1[3]
            #print(clusters[cluster_add_to])
            clusters[cluster_add_to].append(point2)
            #find the averages
            average_PW = (sum(point[0] for point in clusters[cluster_add_to])) /
len(clusters[cluster_add_to])
            average_RF = (sum(point[1] for point in clusters[cluster_add_to])) /
len(clusters[cluster_add_to])
            average_PID = (sum(point[2] for point in clusters[cluster_add_to])) /
len(clusters[cluster_add_to])
            delete = [min_index[0] , min_index[1]]
            #delete = [point1 , point2]

            '''
            for i in delete:
                for j in range(len(pulse_points)):
                    point = pulse_points[j]
                    if len(i) == 4:
                        if (len(point) == 4) and (i[3] == point[3]):
                            del pulse_points[j]
                            break
                    else:
                        if (len(point) == 3) and (i[2] == point[2]):
                            del pulse_points[j]
                            break
            '''
```

```
            for index in sorted(delete, reverse=True):
                del pulse_points[index]

            point_add = [average_PW , average_RF, average_PID, cluster_add_to]
            pulse_points.append(point_add)
            if len(pulse_points) == 1:
                min_value = 1000
            else:
                min_value = np.min(distance)
                if min_value > 5.02494:
                    for point in pulse_points:
                        if len(point) == 3:
                            to_add = [point]
                            clusters.append(to_add)
        elif (len(point2) == 4) and (con == -1):
            cluster_add_to = point2[3]
            #print(clusters[cluster_add_to])
            clusters[cluster_add_to].append(point1)
            #find the averages
            average_PW = (sum(point[0] for point in clusters[cluster_add_to])) /
len(clusters[cluster_add_to])
            average_RF = (sum(point[1] for point in clusters[cluster_add_to])) /
len(clusters[cluster_add_to])
            average_PID = (sum(point[2] for point in clusters[cluster_add_to])) /
len(clusters[cluster_add_to])
            delete = [min_index[0] , min_index[1]]
        #   delete = [point1 , point2]

            '''
            for i in delete:
                for j in range(len(pulse_points)):
                    point = pulse_points[j]
                    if len(i) == 4:
                        if (len(point) == 4) and (i[3] == point[3]):
                            del pulse_points[j]
                            break
                    else:
                        if (len(point) == 3) and (i[2] == point[2]):
                            del pulse_points[j]
                            break
```

```
    '''
    for index in sorted(delete, reverse=True):
        del pulse_points[index]
    point_add = [average_PW , average_RF, average_PID, cluster_add_to]
    pulse_points.append(point_add)
    if len(pulse_points) == 1:
        min_value = 1000
    else:
        min_value = np.min(distance)
        if min_value > 5.02494:
            for point in pulse_points:
                if len(point) == 3:
                    to_add = [point]
                    clusters.append(to_add)
else:
    cluster = [point1, point2]
    clusters.append(cluster)
    #create new point for this cluster
    average_PW = (point1[0] + point2[0]) / 2
    average_RF = (point1[1] + point2[1]) / 2
    average_PID = (point1[2] + point2[2]) / 2
    #remove previous points used
    delete = [min_index[0] , min_index[1]]
    #delete = [point1[2] , point2[2]]

    '''
    for i in delete:
        for j in range(len(pulse_points)):
            point = pulse_points[j]
            if (len(point) == 3) and (i == point[2]):
                del pulse_points[j]
                break
    '''
    for index in sorted(delete, reverse=True):
        del pulse_points[index]

    cluster_index = len(clusters) - 1
    point_add = [average_PW , average_RF, average_PID, cluster_index]
    pulse_points.append(point_add)
    if len(pulse_points) == 1:
```

```
            min_value = 1000
        else:
            min_value = np.min(distance)
            if min_value > 5.02494:
                for point in pulse_points:
                    if len(point) == 3:
                        to_add = [point]
                        clusters.append(to_add)
    '''
    if len(pulse_points) == 1:
        min_value = 1000
    else:
        min_value = np.min(distance)
    '''
    print(min_value)
    print(counter)
    counter += 1
print(clusters)

to_kill = []
inds = 0
for group in clusters:
    if len(group) == 0:
        to_kill.append(inds)
    inds += 1

for index in sorted(to_kill, reverse=True):
        del clusters[index]

clusters_final = []
pointy=[]
counts = 0
for group in clusters:
    average_PW = (sum(point[0] for point in group)) / len(group)
    average_RF = (sum(point[1] for point in group)) / len(group)
    average_PID = (sum(point[2] for point in group)) / len(group)
    cluster_index = counts
    point_add = [average_PW , average_RF, average_PID, cluster_index]
    pointy.append(point_add)
    counts+=1
```

```
    print(pointy)

    min_value = 0
    while min_value < 5.02494:
        arr = []
        for row in pointy:
            curr = []
            for point in pointy:
                dist = ((point[0]-row[0])**2 + (point[1]-row[1])**2)**(1/2)
                curr.append(dist)
            arr.append(curr)
        distance = np.array(arr)
        distance[distance == 0] = 10000000
        min_index = np.unravel_index(np.argmin(distance), distance.shape)

        # find our points to add in a cluster
        point1 = pointy[min_index[0]]
        point2 = pointy[min_index[1]]
        con = 0
        if len(point1) == 4:
            con += 1
        if len(point2) == 4:
            con -= 1
        if (len(point1) == 4) and (len(point2) == 4) and (con == 0):
            cluster_add_to = point1[3]
            for point in clusters[point2[3]]:
                #print(clusters[cluster_add_to])
                clusters[cluster_add_to].append(point)

            clusters[point2[3]].clear()
            #find the averages
            average_PW = (sum(point[0] for point in clusters[cluster_add_to])) /
len(clusters[cluster_add_to])
            average_RF = (sum(point[1] for point in clusters[cluster_add_to])) /
len(clusters[cluster_add_to])
            average_PID = (sum(point[2] for point in clusters[cluster_add_to])) /
len(clusters[cluster_add_to])
            delete = [min_index[0] , min_index[1]]
            #delete = [point1[3] , point2[3]]
            '''
```

```
        for i in delete:
            for j in range(len(pulse_points)):
                point = pulse_points[j]
                if (len(point) == 4) and (i == point[3]):
                    del pulse_points[j]
                    break
        '''
        for index in sorted(delete, reverse=True):
            del pointy[index]

        point_add = [average_PW , average_RF, average_PID, cluster_add_to]
        pointy.append(point_add)
        if len(pointy) == 1:
            min_value = 1000
        else:
            min_value = np.min(distance)



to_kill = []
inds = 0
for clust in clusters:
    if len(clust) == 0:
        to_kill.append(inds)
    inds += 1

for index in sorted(to_kill, reverse=True):
            del clusters[index]

print(clusters)

pulses_in_each = []
for cluster in clusters:
    values = []
    for point in cluster:
        values.append(point[2])
    pulses_in_each.append(values)

print(pulses_in_each)
```

```
    full_clusters = []
    for cluster in pulses_in_each:
        current = data.iloc[cluster]
        full_clusters.append(current)


    more_pulses = 0
    first_bursts = []
    for cluster in full_clusters:
        current = pd.DataFrame(columns=['YMDHM','TOA','Freq','PW','Amp dBm','AOA','PRI',
'Pulse ID'])
        cluster = pd.DataFrame(cluster, columns=['YMDHM','TOA','Freq','PW','Amp
dBm','AOA','PRI', 'Pulse ID'])
        cluster = cluster.sort_values(by=['TOA'])
        #within_cluster = []
        if cluster.shape[0] > 1:
            for i in range(len(cluster) - 1):
                time_test = cluster.iloc[i+1]['TOA'] - cluster.iloc[i]['TOA']
                if (time_test > 10) and (time_test < 5000):
                    new_row = cluster.iloc[i,]
                    new_row = new_row.to_dict()
                    current = current._append(new_row,ignore_index = True)
                    if i == (len(cluster)-2):
                        last_row = cluster.iloc[i+1,]
                        last_row = last_row.to_dict()
                        current = current._append(last_row, ignore_index = True)
                        #current = current.sort_values(by=['TOA'])s
                        PRI = [0.00]
                        for j in range(1, len(current)):
                            pri = current.iloc[j,1] - current.iloc[j-1,1]
                            PRI.append(pri)
                        current['PRI'] = PRI
                        more_pulses += current.shape[0]
                        first_bursts.append(current)
                else:
                    row = cluster.iloc[i,]
                    row = row.to_dict()
                    current = current._append(row, ignore_index = True)
                    #current = current.sort_values(by=['TOA'])
                    PRI = [0.00]
```

```
                for j in range(1, len(current)):
                    pri = current.iloc[j,1] - current.iloc[j-1,1]
                    PRI.append(pri)
                current['PRI'] = PRI
                more_pulses += current.shape[0]
                first_bursts.append(current)
                current = pd.DataFrame(columns=['YMDHM','TOA','Freq','PW','Amp
dBm','AOA','PRI', 'Pulse ID'])
        else:
            first_bursts.append(cluster)
            more_pulses += cluster.shape[0]

    number = 1
    index = 0
    for burst in first_bursts:
        burst['OG Burst ID'] = [number] * len(burst)
        first_bursts[index] = burst
        number += 1
        index += 1

    loc = os.getcwd()
    loc = loc + '\\WhoKnows.txt'
    with open(loc, 'w') as f:
        for i in first_bursts:
            df_string = i.to_string(header=True, index=True)
            f.write(df_string+'\n')

    residue = []
    drop = []
    for i in range(len(first_bursts)):
        burst = first_bursts[i]
        size = burst.shape[0]
        if size < 4:
            residue.append(burst)
            drop.append(i)

    for index in sorted(drop, reverse=True):
            del first_bursts[(index)]
```

```
    residue_data = pd.DataFrame(columns=['YMDHM','TOA','Freq','PW','Amp
dBm','AOA','PRI','Pulse ID','OG Burst ID'])
    for x in residue:
        residue_data = pd.concat([residue_data, x], ignore_index=True)

    residue_data = residue_data.sort_values(by=['Freq'])
    indy = list(range(len(residue_data)))
    residue_data.index = indy

    #residue_data.to_csv('residue.csv')

    kill = []
    changed_bursts = []
    for i in range(len(residue_data)):
        pulse = residue_data.iloc[i,]
        TOA = pulse[1]
        PW = pulse[3]
        RF = pulse[2]
        index = 0
        used = False
        for burst in first_bursts:
            for j in range(len(burst)):
                TOA_test = (burst.iloc[j]['TOA']) - TOA
                PW_test = abs((burst.iloc[j]['PW']) - PW)
                RF_test = abs((burst.iloc[j]['Freq']) - RF)
                if (TOA_test != 0) and (PW_test != 0) and (RF_test != 0):
                    if (abs(TOA_test) < 5000) and (abs(TOA_test) > 10) and (PW_test < 0.5) and
(RF_test < 5):
                        new_row = pulse.to_dict()
                        burst = burst._append(new_row, ignore_index = True)
                        burst = burst.sort_values(by=['TOA'])
                        ind = list(range(1, len(burst)+1))
                        burst.index = ind
                        first_bursts[index] = burst
                        kill.append(i)
                        changed_bursts.append(index)
                        used = True
                        pass
                if used:
                    break
```

```
        if used:
            break
        index += 1


ind = 0
for burst in first_bursts:
    PRI = [0.00]
    for i in range(1, len(burst)):
        pri = burst.iloc[i,1] - burst.iloc[i-1,1]
        PRI.append(pri)
    burst['PRI'] = PRI
    '''
    TOA_list = burst['TOA'].tolist()
    CTOA = []
    for i in TOA_list:
        length = len(TOA_list)
        test = length % 2
        if test == 1:
            index = int((length - 1) / 2)
            curr = TOA_list[index]
            CTOA.append(curr)
        else:
            avg_TOA = (sum(TOA_list)) / length
            index_lower = int((length / 2) - 1)
            index_upper = int(length / 2)
            TOA_low = TOA_list[index_lower]
            TOA_up = TOA_list[index_upper]
            diff_up = abs(avg_TOA - TOA_up)
            diff_low = abs(avg_TOA - TOA_low)
            diff = diff_up - diff_low
            if diff > 0:
                curr = TOA_low
                CTOA.append(curr)
            else:
                curr = TOA_up
                CTOA.append(curr)
    burst['CTOA'] = CTOA
    '''
    first_bursts[ind] = burst
```

```
        ind += 1

    residue_data.drop(kill, axis=0, inplace=True)

    #residue_data.to_csv('ResidueLeft.csv')

    new = []
    current = pd.DataFrame(columns=['YMDHM','TOA','Freq','PW','Amp
dBm','AOA','PRI','Pulse ID','OG Burst ID'])
    for i in range(len(residue_data)-1):
        freq_test = abs((residue_data.iloc[i+1]['Freq']) - (residue_data.iloc[i]['Freq']))
        PW_test = abs((residue_data.iloc[i+1]['PW']) - (residue_data.iloc[i]['PW']))
        if (freq_test < 5) and (PW_test < 0.5):
            new_row = residue_data.iloc[i,]
            new_row = new_row.to_dict()
            current = current._append(new_row, ignore_index = True)
            if i == (len(residue_data)-2):
                new_row = residue_data.iloc[i+1,]
                new_row = new_row.to_dict()
                current = current._append(new_row, ignore_index = True)
                new.append(current)
        else:
            row = residue_data.iloc[i,]
            row = row.to_dict()
            current = current._append(row, ignore_index = True)
            index = list(range(1,len(current)+1))
            current.index = index
            #pulse_in_scan += current.shape[0]
            new.append(current)
            current = pd.DataFrame(columns=['YMDHM','TOA','Freq','PW','Amp
dBm','AOA','PRI','Pulse ID','OG Burst ID'])
            if i == (len(residue_data)-2):
                new_row = residue_data.iloc[i+1,]
                new_row = new_row.to_dict()
                current = current._append(new_row, ignore_index = True)
                new.append(current)

    ranges = []
    for group in new:
        group = group.sort_values(by=['TOA'])
```

```
    current_range = [0.000]
    for i in range(len(group)-1):
        TOA_test = abs(group.iloc[i+1]['TOA'] - group.iloc[i]['TOA'])
        if TOA_test >= 5000:
            current_range.append(group.iloc[i+1]['TOA'])
            if i == (len(group) - 2):
                current_range.append(10000000000000000000000000000000)
    ranges.append(current_range)

residue_bursts = []
for i in range(len(new)):
    current_range = ranges[i]
    group = new[i]
    if len(current_range) != 1:
        for j in range(len(current_range)-1):
            burst = group[group['TOA'] >= current_range[j]]
            burst = burst[burst['TOA'] < current_range[j+1]]
            burst = burst.sort_values(by=['TOA'])
            print(burst)
            PRI = [0.00]
            for i in range(1, len(burst)):
                pri = burst.iloc[i,1] - burst.iloc[i-1,1]
                PRI.append(pri)
            burst['PRI'] = PRI
            '''
            TOA_list = burst['TOA'].tolist()
            CTOA = []
            for i in TOA_list:
                length = len(TOA_list)
                test = length % 2
                if test == 1:
                    index = int((length - 1) / 2)
                    curr = TOA_list[index]
                    CTOA.append(curr)
                else:
                    avg_TOA = (sum(TOA_list)) / length
                    index_lower = int((length / 2) - 1)
                    index_upper = int(length / 2)
                    TOA_low = TOA_list[index_lower]
                    TOA_up = TOA_list[index_upper]
```

```
            diff_up = abs(avg_TOA - TOA_up)
            diff_low = abs(avg_TOA - TOA_low)
            diff = diff_up - diff_low
            if diff > 0:
               curr = TOA_low
               CTOA.append(curr)
            else:
               curr = TOA_up
               CTOA.append(curr)
      burst['CTOA'] = CTOA
      '''
      index = list(range(1,len(burst)+1))
      burst.index = index
      residue_bursts.append(burst)
else:
   burst = group.sort_values(by=['TOA'])
   PRI = [0.00]
   for i in range(1, len(burst)):
      pri = burst.iloc[i,1] - burst.iloc[i-1,1]
      PRI.append(pri)
   burst['PRI'] = PRI
   '''
   TOA_list = burst['TOA'].tolist()
   CTOA = []
   for i in TOA_list:
      length = len(TOA_list)
      test = length % 2
      if test == 1:
         index = int((length - 1) / 2)
         curr = TOA_list[index]
         CTOA.append(curr)
      else:
         avg_TOA = (sum(TOA_list)) / length
         index_lower = int((length / 2) - 1)
         index_upper = int(length / 2)
         TOA_low = TOA_list[index_lower]
         TOA_up = TOA_list[index_upper]
         diff_up = abs(avg_TOA - TOA_up)
         diff_low = abs(avg_TOA - TOA_low)
         diff = diff_up - diff_low
```

```
            if diff > 0:
                curr = TOA_low
                CTOA.append(curr)
            else:
                curr = TOA_up
                CTOA.append(curr)
        burst['CTOA'] = CTOA
        '''
        residue_bursts.append(burst)

if len(residue_bursts) != 0:
    for i in residue_bursts:
        first_bursts.append(i)

'''
locations = []
if len(residue_bursts) != 0:
    for burst in residue_bursts:
        print(burst)
        test = burst["OG Burst ID"].mean()
        print(test)
        #CTOA_test = burst.iloc[0]['CTOA']
        number = 0
        used = False
        for good in first_bursts:
            test_to = good['OG Burst ID'].mean()
            print(test_to)
            #CTOA_good = good.iloc[0]['CTOA']
            compare = test - test_to
            if compare > 0:
                locations.append(number+1)
                used = True
            number += 1
            if used:
                break

print('HEHE')
locations = sorted(locations, reverse=True)
print(locations)
access = list(range(len(residue_bursts)))
```

```
access = sorted(access,reverse=True)
print(access)
if len(locations) != 0:
    for i in range(len(access)):
        access_point = access[i]
        to_add = residue_bursts[access_point]
        where = locations[i]
        first_bursts.insert(where,to_add)
'''

how_many = 0
for burst in first_bursts:
    how_many += burst.shape[0]


'''
data1 = []
for burst in data:
    burst = burst.applymap('{:.6f}'.format)
    data1.append(burst)
'''

data1 = first_bursts

print(data1)

number_scans = len(data1)
print(number_scans)
indicies = []
m=0
for i in data1:
    if (len(i.index)) < 4:
        indicies.append(m)
    m += 1
scans_less_four_pulses = len(indicies)
if number_scans != 0:
    prop_less_four = scans_less_four_pulses/number_scans
else:
    prop_less_four = 'NA'
#This loop throws away our dataframes that have less than 3 observations
```

```
for index in sorted(indicies, reverse=True):
    del data1[index]

scans_kept = len(data1)


out_of = str(len(data1))
currently_on = 1
data2 = []
indy = 1
for burst in data1:
    print("We are currently on burst "+str(currently_on)+" out of "+out_of)
    PRI = [0.00]
    for i in range(1, len(burst)):
        pri = burst.iloc[i,1] - burst.iloc[i-1,1]
        PRI.append(pri)
    burst['PRI'] = PRI
    TOA_list = burst['TOA'].tolist()
    '''

    CTOA = []
    for i in TOA_list:
        length = len(TOA_list)
        test = length % 2
        if test == 1:
            index = int((length - 1) / 2)
            curr = TOA_list[index]
            CTOA.append(curr)
        else:
            avg_TOA = (sum(TOA_list)) / length
            index_lower = int((length / 2) - 1)
            index_upper = int(length / 2)
            TOA_low = TOA_list[index_lower]
            TOA_up = TOA_list[index_upper]
            diff_up = abs(avg_TOA - TOA_up)
            diff_low = abs(avg_TOA - TOA_low)
            diff = diff_up - diff_low
            if diff > 0:
                curr = TOA_low
                CTOA.append(curr)
```

```
            else:
                curr = TOA_up
                CTOA.append(curr)
        burst['CTOA'] = CTOA
        '''

        bins_current = bi.bins(burst)
        if bins_current.shape[0] >1: #will check if clock is constant or not
            clock = cl.clock(bins_current) #more than one row in bins, we can pass bins into clock
file
        else:
            clock = 1 #otherwise, clock is constant 1
        clock_column = [clock]*(len(burst))
        burst['Clock'] = clock_column

        burst['Final Burst ID'] = [indy] * len(burst)
        index = list(range(1,len(burst)+1))
        burst.index = index
        data2.append(burst)
        indy += 1
        currently_on += 1

    data3 = []
    ind = 1
    for burst in data2:
        burst['Final Burst ID'] = [ind] * len(burst)
        index = list(range(1,len(burst)+1))
        burst.index = index
        data3.append(burst)
        ind += 1

    loc = os.getcwd()
    loc = loc + '\\pulse_deint.txt'
    with open(loc, 'w') as f:
        for i in data3:
            df_string = i.to_string(header=True, index=True)
            f.write(df_string+'\n')

    DTOA_bursts = []
    for i in range(len(data3)-1):
```

```
    current_burst = data3[i]
    next_burst = data3[i+1]
    DTOA = (float(next_burst.iloc[0]['TOA'])) - (float(current_burst['TOA'].iloc[-1]))
    DTOA_bursts.append(DTOA)

pulse_in_scan = 0
for burst in data3:
    pulses = burst.shape[0]
    pulse_in_scan += pulses

sizes = []
burst_number = []
labels = []
cue = 1
for burst in data3:
    sizes.append(burst.shape[0])
    burst_number.append(burst.iloc[0]['Final Burst ID'])
    labels.append("Scan " + str(cue))
    cue += 1

if len(sizes) != 0:
    max_burst = max(sizes)
    min_burst = min(sizes)
    burst_number_max = burst_number[sizes.index(max_burst)]
    burst_number_min = burst_number[sizes.index(min_burst)]
else:
    max_burst = 'NA'
    min_burst = "NA"
    burst_number_max = 'NA'
    burst_number_min = 'NA'

prop_pulses_kept = pulse_in_scan / total_pulses
pulses_in_residue = total_pulses - pulse_in_scan

sizes1 = [pulse_in_scan, pulses_in_residue]
labels1 = ['Pulses in RADAR Scans', 'Residue']


fig1, ax1 = plt.subplots()
ax1.pie(sizes1, labels=labels1, autopct='%1.1f%%', startangle=90)
```

```
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()

x_axis = labels
y_axis = sizes

plt.bar(x_axis, y_axis)
plt.title('How Much is in each Scan?')
plt.ylabel('Number of Pulses')
plt.show()

x_axis1 = ['In Scan', 'Residue']
y_axis1 = [pulse_in_scan, pulses_in_residue]
plt.bar(x_axis1, y_axis1)
plt.title('How Much is in Residue?')
plt.ylabel('Number of Pulses')
plt.show()

loc = os.getcwd()
loc1 = loc + '\\stats_pulse.txt'
with open(loc1, 'w') as f:
    f.write('Total pulses: ' + str(total_pulses)+'\n')
    f.write('Total scans: ' + str(number_scans)+'\n')
    f.write('Scans kept: ' + str(scans_kept)+'\n')
    f.write('Pulses contained in scans: ' +str(pulse_in_scan)+'\n')
    f.write('Proportion pulses kept in scans: ' +str(prop_pulses_kept)+'\n')
    f.write('Scans with less than four pulses: ' +str(scans_less_four_pulses)+'\n')
    f.write('Proportion of scans with less than four pulses: ' + str(prop_less_four)+'\n')
    f.write('Number of pulses in largest scan: ' + str(max_burst)+'\n')
    f.write('Which scan is largest: ' + str(burst_number_max)+'\n')
    f.write('Number of pulses in smallest scan: ' + str(min_burst)+'\n')
    f.write('Which scan is smallest: ' + str(burst_number_min)+'\n')
    f.write('List of time between scans (in microseconds): ' + str(DTOA_bursts)+'\n')

indexes = list(range(len(data2)))
combined = pd.concat(data3, keys = indexes)
combined.to_csv('combined.csv', index=False)

return data3
```

```
#deinterleave('here.csv')
if __name__ == '__main__':
    pass



    more_pulses = 0
    first_bursts = []
    for cluster in full_clusters:
        current = pd.DataFrame(columns=['YMDHM','TOA','Freq','PW','Amp dBm','AOA','PRI',
'Pulse ID'])
        cluster = pd.DataFrame(cluster, columns=['YMDHM','TOA','Freq','PW','Amp
dBm','AOA','PRI', 'Pulse ID'])
        cluster = cluster.sort_values(by=['TOA'])
        #within_cluster = []
        if cluster.shape[0] > 1:
            for i in range(len(cluster) - 1):
                time_test = cluster.iloc[i+1]['TOA'] - cluster.iloc[i]['TOA']
                if (time_test > 10) and (time_test < 5000):
                    new_row = cluster.iloc[i,]
                    new_row = new_row.to_dict()
                    current = current._append(new_row,ignore_index = True)
                    if i == (len(cluster)-2):
                        last_row = cluster.iloc[i+1,]
                        last_row = last_row.to_dict()
                        current = current._append(last_row, ignore_index = True)
                        #current = current.sort_values(by=['TOA'])s
                        PRI = [0.00]
                        for j in range(1, len(current)):
                            pri = current.iloc[j,1] - current.iloc[j-1,1]
                            PRI.append(pri)
                        current['PRI'] = PRI
                        more_pulses += current.shape[0]
                        first_bursts.append(current)
                else:
                    row = cluster.iloc[i,]
                    row = row.to_dict()
                    current = current._append(row, ignore_index = True)
                    #current = current.sort_values(by=['TOA'])
```

```
                PRI = [0.00]
                for j in range(1, len(current)):
                    pri = current.iloc[j,1] - current.iloc[j-1,1]
                    PRI.append(pri)
                current['PRI'] = PRI
                more_pulses += current.shape[0]
                first_bursts.append(current)
                current = pd.DataFrame(columns=['YMDHM','TOA','Freq','PW','Amp
dBm','AOA','PRI', 'Pulse ID'])
        else:
            first_bursts.append(cluster)
            more_pulses += cluster.shape[0]

    number = 1
    index = 0
    for burst in first_bursts:
        burst['OG Burst ID'] = [number] * len(burst)
        first_bursts[index] = burst
        number += 1
        index += 1

    loc = os.getcwd()
    loc = loc + '\\WhoKnows.txt'
    with open(loc, 'w') as f:
        for i in first_bursts:
            df_string = i.to_string(header=True, index=True)
            f.write(df_string+'\n')

    residue = []
    drop = []
    for i in range(len(first_bursts)):
        burst = first_bursts[i]
        size = burst.shape[0]
        if size < 4:
            residue.append(burst)
            drop.append(i)

    for index in sorted(drop, reverse=True):
            del first_bursts[(index)]
```

```
    residue_data = pd.DataFrame(columns=['YMDHM','TOA','Freq','PW','Amp
dBm','AOA','PRI','Pulse ID','OG Burst ID'])
    for x in residue:
        residue_data = pd.concat([residue_data, x], ignore_index=True)

    residue_data = residue_data.sort_values(by=['Freq'])
    indy = list(range(len(residue_data)))
    residue_data.index = indy

    #residue_data.to_csv('residue.csv')

    kill = []
    changed_bursts = []
    for i in range(len(residue_data)):
        pulse = residue_data.iloc[i,]
        TOA = pulse[1]
        PW = pulse[3]
        RF = pulse[2]
        index = 0
        used = False
        for burst in first_bursts:
            for j in range(len(burst)):
                TOA_test = (burst.iloc[j]['TOA']) - TOA
                PW_test = abs((burst.iloc[j]['PW']) - PW)
                RF_test = abs((burst.iloc[j]['Freq']) - RF)
                if (TOA_test != 0) and (PW_test != 0) and (RF_test != 0):
                    if (abs(TOA_test) < 5000) and (abs(TOA_test) > 10) and (PW_test < 0.5) and
(RF_test < 5):
                        new_row = pulse.to_dict()
                        burst = burst._append(new_row, ignore_index = True)
                        burst = burst.sort_values(by=['TOA'])
                        ind = list(range(1, len(burst)+1))
                        burst.index = ind
                        first_bursts[index] = burst
                        kill.append(i)
                        changed_bursts.append(index)
                        used = True
                        pass
                if used:
                    break
```

```
        if used:
            break
        index += 1


ind = 0
for burst in first_bursts:
    PRI = [0.00]
    for i in range(1, len(burst)):
        pri = burst.iloc[i,1] - burst.iloc[i-1,1]
        PRI.append(pri)
    burst['PRI'] = PRI
    '''
    TOA_list = burst['TOA'].tolist()
    CTOA = []
    for i in TOA_list:
        length = len(TOA_list)
        test = length % 2
        if test == 1:
            index = int((length - 1) / 2)
            curr = TOA_list[index]
            CTOA.append(curr)
        else:
            avg_TOA = (sum(TOA_list)) / length
            index_lower = int((length / 2) - 1)
            index_upper = int(length / 2)
            TOA_low = TOA_list[index_lower]
            TOA_up = TOA_list[index_upper]
            diff_up = abs(avg_TOA - TOA_up)
            diff_low = abs(avg_TOA - TOA_low)
            diff = diff_up - diff_low
            if diff > 0:
                curr = TOA_low
                CTOA.append(curr)
            else:
                curr = TOA_up
                CTOA.append(curr)
    burst['CTOA'] = CTOA
    '''
    first_bursts[ind] = burst
```

```
        ind += 1

    residue_data.drop(kill, axis=0, inplace=True)

    #residue_data.to_csv('ResidueLeft.csv')

    new = []
    current = pd.DataFrame(columns=['YMDHM','TOA','Freq','PW','Amp
dBm','AOA','PRI','Pulse ID','OG Burst ID'])
    for i in range(len(residue_data)-1):
        freq_test = abs((residue_data.iloc[i+1]['Freq']) - (residue_data.iloc[i]['Freq']))
        PW_test = abs((residue_data.iloc[i+1]['PW']) - (residue_data.iloc[i]['PW']))
        if (freq_test < 5) and (PW_test < 0.5):
            new_row = residue_data.iloc[i,]
            new_row = new_row.to_dict()
            current = current._append(new_row, ignore_index = True)
            if i == (len(residue_data)-2):
                new_row = residue_data.iloc[i+1,]
                new_row = new_row.to_dict()
                current = current._append(new_row, ignore_index = True)
                new.append(current)
        else:
            row = residue_data.iloc[i,]
            row = row.to_dict()
            current = current._append(row, ignore_index = True)
            index = list(range(1,len(current)+1))
            current.index = index
            #pulse_in_scan += current.shape[0]
            new.append(current)
            current = pd.DataFrame(columns=['YMDHM','TOA','Freq','PW','Amp
dBm','AOA','PRI','Pulse ID','OG Burst ID'])
            if i == (len(residue_data)-2):
                new_row = residue_data.iloc[i+1,]
                new_row = new_row.to_dict()
                current = current._append(new_row, ignore_index = True)
                new.append(current)

    ranges = []
    for group in new:
        group = group.sort_values(by=['TOA'])
```

```
    current_range = [0.000]
    for i in range(len(group)-1):
        TOA_test = abs(group.iloc[i+1]['TOA'] - group.iloc[i]['TOA'])
        if TOA_test >= 5000:
            current_range.append(group.iloc[i+1]['TOA'])
            if i == (len(group) - 2):
                current_range.append(10000000000000000000000000000000)
    ranges.append(current_range)

residue_bursts = []
for i in range(len(new)):
    current_range = ranges[i]
    group = new[i]
    if len(current_range) != 1:
        for j in range(len(current_range)-1):
            burst = group[group['TOA'] >= current_range[j]]
            burst = burst[burst['TOA'] < current_range[j+1]]
            burst = burst.sort_values(by=['TOA'])
            print(burst)
            PRI = [0.00]
            for i in range(1, len(burst)):
                pri = burst.iloc[i,1] - burst.iloc[i-1,1]
                PRI.append(pri)
            burst['PRI'] = PRI
            '''
            TOA_list = burst['TOA'].tolist()
            CTOA = []
            for i in TOA_list:
                length = len(TOA_list)
                test = length % 2
                if test == 1:
                    index = int((length - 1) / 2)
                    curr = TOA_list[index]
                    CTOA.append(curr)
                else:
                    avg_TOA = (sum(TOA_list)) / length
                    index_lower = int((length / 2) - 1)
                    index_upper = int(length / 2)
                    TOA_low = TOA_list[index_lower]
                    TOA_up = TOA_list[index_upper]
```

```
            diff_up = abs(avg_TOA - TOA_up)
            diff_low = abs(avg_TOA - TOA_low)
            diff = diff_up - diff_low
            if diff > 0:
               curr = TOA_low
               CTOA.append(curr)
            else:
               curr = TOA_up
               CTOA.append(curr)
      burst['CTOA'] = CTOA
      '''
      index = list(range(1,len(burst)+1))
      burst.index = index
      residue_bursts.append(burst)
else:
   burst = group.sort_values(by=['TOA'])
   PRI = [0.00]
   for i in range(1, len(burst)):
      pri = burst.iloc[i,1] - burst.iloc[i-1,1]
      PRI.append(pri)
   burst['PRI'] = PRI
   '''
   TOA_list = burst['TOA'].tolist()
   CTOA = []
   for i in TOA_list:
      length = len(TOA_list)
      test = length % 2
      if test == 1:
         index = int((length - 1) / 2)
         curr = TOA_list[index]
         CTOA.append(curr)
      else:
         avg_TOA = (sum(TOA_list)) / length
         index_lower = int((length / 2) - 1)
         index_upper = int(length / 2)
         TOA_low = TOA_list[index_lower]
         TOA_up = TOA_list[index_upper]
         diff_up = abs(avg_TOA - TOA_up)
         diff_low = abs(avg_TOA - TOA_low)
         diff = diff_up - diff_low
```

```
            if diff > 0:
                curr = TOA_low
                CTOA.append(curr)
            else:
                curr = TOA_up
                CTOA.append(curr)
        burst['CTOA'] = CTOA
        '''
        residue_bursts.append(burst)

locations = []
if len(residue_bursts) != 0:
    for burst in residue_bursts:
        print(burst)
        test = burst["OG Burst ID"].mean()
        print(test)
        #CTOA_test = burst.iloc[0]['CTOA']
        number = 0
        used = False
        for good in first_bursts:
            test_to = good['OG Burst ID'].mean()
            print(test_to)
            #CTOA_good = good.iloc[0]['CTOA']
            compare = test - test_to
            if compare > 0:
                locations.append(number+1)
                used = True
            number += 1
            if used:
                break

print('HEHE')
locations = sorted(locations, reverse=True)
print(locations)
access = list(range(len(residue_bursts)))
access = sorted(access,reverse=True)
print(access)
if len(locations) != 0:
    for i in range(len(access)):
        access_point = access[i]
```

```
        to_add = residue_bursts[access_point]
        where = locations[i]
        first_bursts.insert(where,to_add)

how_many = 0
for burst in first_bursts:
    how_many += burst.shape[0]


out_of = str(len(first_bursts))
currently_on = 1
data = []
indy = 1
for burst in first_bursts:
    print("We are currently on burst "+str(currently_on)+" out of "+out_of)
    PRI = [0.00]
    for i in range(1, len(burst)):
        pri = burst.iloc[i,1] - burst.iloc[i-1,1]
        PRI.append(pri)
    burst['PRI'] = PRI
    TOA_list = burst['TOA'].tolist()
    '''
    CTOA = []
    for i in TOA_list:
        length = len(TOA_list)
        test = length % 2
        if test == 1:
            index = int((length - 1) / 2)
            curr = TOA_list[index]
            CTOA.append(curr)
        else:
            avg_TOA = (sum(TOA_list)) / length
            index_lower = int((length / 2) - 1)
            index_upper = int(length / 2)
            TOA_low = TOA_list[index_lower]
            TOA_up = TOA_list[index_upper]
            diff_up = abs(avg_TOA - TOA_up)
            diff_low = abs(avg_TOA - TOA_low)
            diff = diff_up - diff_low
            if diff > 0:
```

```
                curr = TOA_low
                CTOA.append(curr)
            else:
                curr = TOA_up
                CTOA.append(curr)
        burst['CTOA'] = CTOA
        '''
        bins_current = bi.bins(burst)
        if bins_current.shape[0] >1: #will check if clock is constant or not
            clock = cl.clock(bins_current) #more than one row in bins, we can pass bins into clock
file
        else:
            clock = 1 #otherwise, clock is constant 1
        clock_column = [clock]*(len(burst))
        burst['Clock'] = clock_column

        burst['Final Burst ID'] = [indy] * len(burst)
        index = list(range(1,len(burst)+1))
        burst.index = index
        data.append(burst)
        indy += 1
        currently_on += 1

    '''
    data1 = []
    for burst in data:
        burst = burst.applymap('{:.6f}'.format)
        data1.append(burst)
    '''
    data1 = data

    DTOA_bursts = []
    for i in range(len(data1)-1):
        current_burst = data1[i]
        next_burst = data1[i+1]
        DTOA = (float(next_burst.iloc[0]['TOA'])) - (float(current_burst['TOA'].iloc[-1]))
        DTOA_bursts.append(DTOA)
```

```
number_scans = len(data1)
print(number_scans)
indicies = []
m=0
for i in data1:
    if (len(i.index)) < 4:
        indicies.append(m)
    m += 1
scans_less_four_pulses = len(indicies)
if number_scans != 0:
    prop_less_four = scans_less_four_pulses/number_scans
else:
    prop_less_four = 'NA'
#This loop throws away our dataframes that have less than 3 observations
for index in sorted(indicies, reverse=True):
    del data1[index]

scans_kept = len(data1)

data2 = []
ind = 1
for burst in data1:
    burst['Final Burst ID'] = [ind] * len(burst)
    index = list(range(1,len(burst)+1))
    burst.index = index
    data2.append(burst)
    ind += 1

loc = os.getcwd()
loc = loc + '\\pulse_deint.txt'
with open(loc, 'w') as f:
    for i in data2:
        df_string = i.to_string(header=True, index=True)
        f.write(df_string+'\n')

pulse_in_scan = 0
for burst in data2:
    pulses = burst.shape[0]
    pulse_in_scan += pulses
```

```
sizes = []
burst_number = []
labels = []
cue = 1
for burst in data2:
    sizes.append(burst.shape[0])
    burst_number.append(burst.iloc[0]['Final Burst ID'])
    labels.append("Scan " + str(cue))
    cue += 1

if len(sizes) != 0:
    max_burst = max(sizes)
    min_burst = min(sizes)
    burst_number_max = burst_number[sizes.index(max_burst)]
    burst_number_min = burst_number[sizes.index(min_burst)]
else:
    max_burst = 'NA'
    min_burst = "NA"
    burst_number_max = 'NA'
    burst_number_min = 'NA'

prop_pulses_kept = pulse_in_scan / total_pulses
pulses_in_residue = total_pulses - pulse_in_scan

x_axis = labels
y_axis = sizes

plt.bar(x_axis, y_axis)
plt.title('How Much is in each Scan?')
plt.ylabel('Number of Pulses')
plt.show()

x_axis1 = ['In Scan', 'Residue']
y_axis1 = [pulse_in_scan, pulses_in_residue]
plt.bar(x_axis1, y_axis1)
plt.title('How Much is in Residue?')
plt.ylabel('Number of Pulses')
plt.show()

loc = os.getcwd()
```

```
    loc1 = loc + '\\stats_pulse.txt'
    with open(loc1, 'w') as f:
        f.write('Total pulses: ' + str(total_pulses)+'\n')
        f.write('Total scans: ' + str(number_scans)+'\n')
        f.write('Scans kept: ' + str(scans_kept)+'\n')
        f.write('Pulses contained in scans: ' +str(pulse_in_scan)+'\n')
        f.write('Proportion pulses kept in scans: ' +str(prop_pulses_kept)+'\n')
        f.write('Scans with less than four pulses: ' +str(scans_less_four_pulses)+'\n')
        f.write('Proportion of scans with less than four pulses: ' + str(prop_less_four)+'\n')
        f.write('Number of pulses in largest scan: ' + str(max_burst)+'\n')
        f.write('Which scan is largest: ' + str(burst_number_max)+'\n')
        f.write('Number of pulses in smallest scan: ' + str(min_burst)+'\n')
        f.write('Which scan is smallest: ' + str(burst_number_min)+'\n')
        f.write('List of time between scans (in microseconds): ' + str(DTOA_bursts)+'\n')

    indexes = list(range(len(data2)))
    combined = pd.concat(data2, keys = indexes)
    combined.to_csv('combined.csv', index=False)

    return data2

#deinterleave('here.csv')
if __name__ == '__main__':
    pass
```