Bullet Background Paper

ON

Public Release of the Synthetic Radar 2021 (SynthRad-21) Dataset

1 February 2022

AFRL/RYWE is requesting public release clearance for the Synthetic Radar 21 (SynthRad-21) dataset.

- Dataset consists of 25 one-second simulated recordings of individual single function radars with various frequency, pulse width, PRI, and modulation agilities.
    - Consists of 14 search, 3 acquisition, 7 track, and 1 target illumination radars.
    - Emitters are spread across 1 GHz in the X-band (8-9 GHz).
    - Waveform parameters were selected to be distinct from any real-world radar systems.
    - A summary of each emitter's waveform parameters is provided in Appendix I.
- AFRL/RYWE is requesting public release clearance for this dataset in order to share with civilian research organizations such as universities and colleges.
    - Benefits: Provides a common dataset to facilitate collaboration across research organizations. Allows development and testing of new detection and classification algorithms on a common baseline dataset.
    - Requesting release of the following:
        - Complex-valued signal data in Midas 1000 format (25 files).
        - Pulse-by-pulse true waveform parameters in CSV format (25 files, example excerpt in Appendix II).
        - Configuration files used to create data, containing a summary of all waveforms employed by each emitter, as well as antenna type and scan rate (50 files, an emitter configuration file and a scenario configuration file for each emitter, examples in Appendix III and Appendix IV).
        - Python scripts to facilitate reading of the data from Midas format into memory (midas_tools.py, shown in Appendix V, plus the bluefile directory containing helper code).
        - A Python preprocessing script to parse, format, and shuffle the data for machine learning experimentation (data_preprocessing.py, script provided).
    - Data is provided at baseband without noise or other imperfections. The preprocessing script is responsible for upsampling the data, mixing to the correct center frequency, and simulating the effects of receiver noise and other channel impairments.
- SynthRad-21 dataset available for review in Sensors Directorate Building 620, Area B
    - POC: Chris Ebersole (937-713-4009 / christopher.ebersole.1@us.af.mil)

Appendix I Waveform Parameters

Table 1: Waveform parameters used by each emitter.

Name	Mode	Scan Type / Antenna Type	Scan Time (s)	Center Freq. (GHz)	Freq. Agility	Pulse Width (us)	Pulse Width Agility	PRI (us)	PRI Agility	Modulation	Chirp Bandwidth (MHz)	Phase Code Length	Modulation Agility
Alpaca	Search	Rotational Azimuth	0.5	8.69	N/A	32	N/A	3700	N/A	LFM	4.3	N/A	N/A
Bear	Search	Isotropic	N/A	8.195 - 8.244	Dwell & Switch	170 - 200	Dwell & Switch	4430	N/A	UNMOD	N/A	N/A	N/A
Bobcat	Search	Rotational Azimuth	0.35	8.559 - 8.658	Dwell & Switch	160	N/A	3353	N/A	UNMOD	N/A	N/A	N/A
Camel	Search	Rotational Azimuth	0.6	8.246 -8.346	Dwell & Switch	88	N/A	2330	N/A	UNMOD	N/A	N/A	N/A
Cheetah	Search	Rotational Azimuth	0.6	8.246 - 8.316	Dwell & Switch	88	N/A	2330	N/A	UNMOD	N/A	N/A	N/A
Deer	Search	Rotational Azimuth	0.6	8.64	N/A	11.6	N/A	149	Bursts, 13.5 ms gaps	UNMOD	N/A	N/A	N/A
Dingo	Acquisition	Paddle Scan	0.1	8.46	N/A	1.305 - 1.65	Dwell & Switch	8.7 - 11	Dwell and Switch	UNMOD	N/A	N/A	N/A
Goat	Search	Isotropic	N/A	8.64	N/A	13.3 - 16.9	Dwell & Switch	169 - 11400	Dwell and Switch	UNMOD	N/A	N/A	N/A
Gorilla	Track	Isotropic	N/A	8.314672	N/A	2.25	N/A	6.4	N/A	UNMOD	N/A	N/A	N/A
Hare	Search	Isotropic	N/A	8.777	N/A	97	N/A	1250	N/A	LFM	2.7	N/A	N/A
Jackal	Acquisition	Paddle Scan	0.06	8.26	N/A	1.54 - 2.2	Dwell & Switch	7 - 10	Dwell and Switch	UNMOD	N/A	N/A	N/A
Jaguar	Search	Rotational Azimuth	0.25	8.65	N/A	80 - 110	Staggered	3000 - 5000	Staggered	LFM	5.3	N/A	N/A
Kangaroo	Acquisition	Isotropic	N/A	8.78	N/A	3.125 - 4	Dwell & Switch	12.5 - 16	Dwell and Switch	UNMOD	N/A	N/A	N/A
Lion	Track	Isotropic	N/A	8.393	N/A	150 - 200	Staggered	2200 - 2400	Staggered	BARKER	N/A	5, 11	Staggered
Mandrill	Track	Isotropic	N/A	8.55	N/A	60	N/A	555 +/- 5	Jittered	UNMOD	N/A	N/A	N/A
Marmot	Track	Isotropic	N/A	8.3	N/A	10	N/A	905 +/- 5	Jittered	UNMOD	N/A	N/A	N/A
Mink	Search	Isotropic	N/A	8.18	N/A	69	N/A	510	N/A	UNMOD	N/A	N/A	N/A
Mole	Track	Isotropic	N/A	8.71	N/A	3	N/A	22 +/- 2	Jittered	UNMOD	N/A	N/A	N/A
Mouse	Track	Isotropic	N/A	8.415	N/A	75	N/A	440	N/A	BARKER	N/A	5, 7, 11	Staggered
Puma	Search	Isotropic	N/A	8.45	N/A	90 - 120	Staggered	1500 - 4000	Staggered	LFM	6	N/A	N/A
Rat	Track	Isotropic	N/A	8.018	N/A	5.84 - 7.8	Dwell & Switch	14.6 - 19.5	Dwell and Switch	UNMOD	N/A	N/A	N/A
Starling	Target Illumination	Isotropic	N/A	8.77	N/A	8400	N/A	46000	N/A	UNMOD	N/A	N/A	N/A
Weasel	Search	Isotropic	N/A	8.803	N/A	65	N/A	3700	N/A	LFM	1.9	N/A	N/A
Wolf	Search	Isotropic	N/A	8.26	N/A	67 - 90	Staggered	1750 - 5000	Staggered	UNMOD	N/A	N/A	N/A
Zebra	Search	Isotropic	N/A	8.2589	N/A	200	N/A	5000	N/A	UNMOD	N/A	N/A	N/A

Note: Rotational Azimuth and Paddle Scan use Sinc Antenna

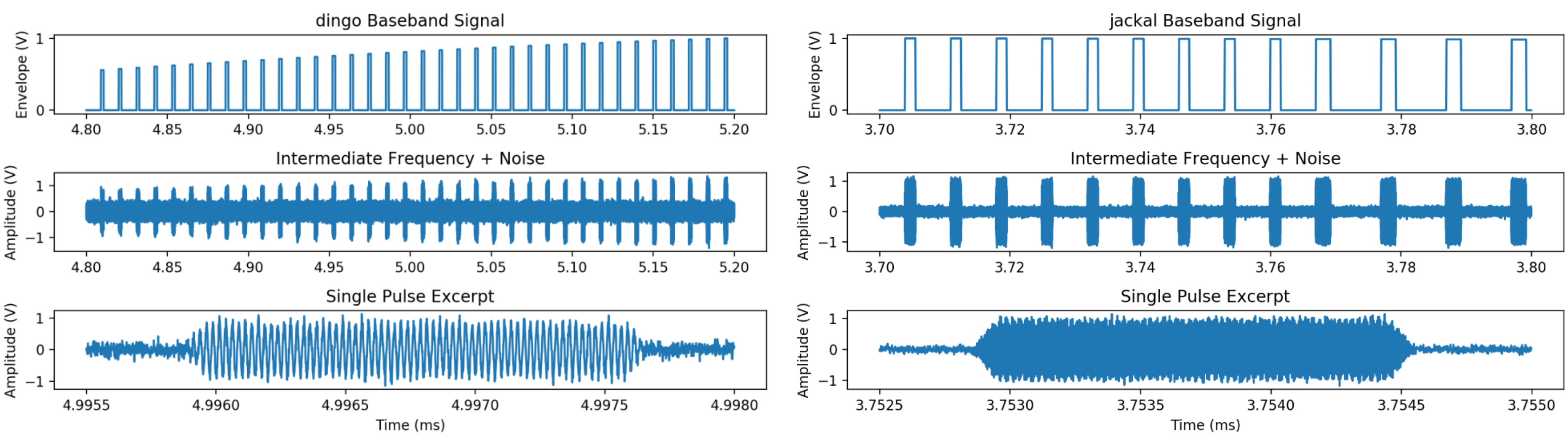

Figure 1: Examples of SynthRad-21 waveforms for two different radars. Data is distributed as noiseless baseband signals. The Python preprocessing script upsamples the signal, shifts the signal to the correct intermediate frequency, and adds noise and other artifacts.

Appendix II Example Truth File Excerpt (Dingo)

Tx Time (sec)	Rx Time (sec)	Pulse Width (sec)	Amplitude (V)	Carrier Frequency (Hz)	Leading Edge Phase (rad)	SNR (dB)	PRI (sec)	General Mode	Exact Mode	Index	Received Azimuth (deg)	Received Elevation (deg)	Tx North (km)	Tx East (km)	Tx Down (km)	Rx North (km)	Rx East (km)	Rx Down (km)	Modulation Type	Chirp Bandwidth (Hz)	Chirp Direction	Chip Count
0	0.000366921	0.00000165	6.854E-07	8460000000	-2.952556761	-12.2919545	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000011	0.000377921	0.00000165	6.334E-07	8460000000	-2.952556761	-12.9772322	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000022	0.000388921	0.00000165	5.812E-07	8460000000	-2.952556761	-13.7241578	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000033	0.000399921	0.00000165	5.288E-07	8460000000	-2.95255676	-14.5443528	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000044	0.000410921	0.00000165	4.763E-07	8460000000	-2.95255676	-15.4531299	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000055	0.000421921	0.00000165	4.236E-07	8460000000	3.330628547	-16.4712428	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000066	0.000432921	0.00000165	3.708E-07	8460000000	3.330628547	-17.6278148	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000077	0.000443921	0.00000165	3.179E-07	8460000000	3.330628547	-18.9655475	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000088	0.000454921	0.00000165	2.648E-07	8460000000	3.330628547	-20.5507426	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000099	0.000465921	0.00000165	2.117E-07	8460000000	3.330628547	-22.4947082	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.00011	0.000476921	0.00000165	1.586E-07	8460000000	3.330628546	-25.0067687	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000121	0.000487921	0.00000165	1.053E-07	8460000000	3.330628546	-28.5594159	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000132	0.000498921	0.00000165	5.21E-08	8460000000	3.330628547	-34.6789188	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000143	0.000509921	0.00000165	2.2E-09	8460000000	3.330628544	-62.2449656	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000154	0.000520921	0.00000165	5.45E-08	8460000000	3.330628547	-34.2836529	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000165	0.000531921	0.00000165	1.078E-07	8460000000	3.330628544	-28.3612313	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000176	0.000542921	0.00000165	0.000000161	8460000000	3.330628547	-24.8740075	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000187	0.000553921	0.00000165	2.142E-07	8460000000	3.330628544	-22.3944648	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000198	0.000564921	0.00000165	2.673E-07	8460000000	3.330628547	-20.4698557	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000209	0.000575921	0.00000165	3.204E-07	8460000000	3.330628543	-18.8974366	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.00022	0.000586921	0.00000165	3.733E-07	8460000000	3.330628546	-17.5687196	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000231	0.000597921	0.00000165	4.262E-07	8460000000	3.330628544	-16.4188134	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000242	0.000608921	0.00000165	4.789E-07	8460000000	3.330628547	-15.4057994	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000253	0.000619921	0.00000165	5.315E-07	8460000000	3.330628544	-14.5010246	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000264	0.000630921	0.00000165	5.839E-07	8460000000	3.330628547	-13.6840343	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000275	0.000641921	0.00000165	6.361E-07	8460000000	3.330628544	-12.9397153	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000286	0.000652921	0.00000165	6.882E-07	8460000000	3.330628541	-12.2565841	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000297	0.000663921	0.00000165	0.00000074	8460000000	3.330628544	-11.6257107	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000308	0.000674921	0.00000165	7.916E-07	8460000000	3.330628547	-11.0400132	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000319	0.000685921	0.00000165	0.000000843	8460000000	3.330628545	-10.4937811	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.00033	0.000696921	0.00000165	8.941E-07	8460000000	3.330628542	-9.98234245	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000341	0.000707921	0.00000165	0.000000945	8460000000	3.330628545	-9.50182657	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000352	0.000718921	0.00000165	9.956E-07	8460000000	3.330628548	-9.04899074	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000363	0.000729921	0.00000165	1.0458E-06	8460000000	3.330628545	-8.62109144	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000374	0.000740921	0.00000165	1.0958E-06	8460000000	3.330628543	-8.21578696	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000385	0.000751921	0.00000165	1.1454E-06	8460000000	3.330628546	-7.83106265	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000396	0.000762921	0.00000165	1.1947E-06	8460000000	3.330628549	-7.46517287	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000407	0.000773921	0.00000165	1.2436E-06	8460000000	3.330628546	-7.11659514	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0
0.000418	0.000784921	0.00000165	1.2922E-06	8460000000	3.33062854	-6.7839938	0.000011	Acquisition	Dwell_And_Switch	0	0	0	110	0	0	0	0	0	UNMODULATED	0	0	0

Appendix III Example Scenario Configuration File (Dingo)

```
BEGIN SimulationFlatEarth
    RunTime: 1 //seconds

BEGIN SimulationPlayer Player-angry_dingo_a
        BEGIN GeometryObject
            Position: 1.1e5 0.0 0.0
            Orientation: 0.0 0.0 0.0
        END

        BEGIN Transmitter angry_dingo_a
            BEGIN TransducerAssociation
                Name: Tx-angry_dingo_a
            END
            BEGIN TransmitMode
                StartTime: 0.0
                StopTime: 1
                Schedule: angry_dingo_a
            END
        END
        FILE: ../../Emitters/Chris/DAS/angry_dingo_a.cfg
    END

    BEGIN SimulationPlayer RxVehicle
        BEGIN GeometryObject
            Position: 0.0 0.0 0.0
            Orientation: 0.0 0.0 0.0
        END

        BEGIN TransducerIsotropic Rx1
            Frequency: 8.46e9
            Bandwidth: 20e6
        END
        BEGIN Receiver
            BEGIN IQDataIO
                Filename: angry_dingo_a
                Format: Midas
                DataType: complex
                ElementType: double
                Detached: false
                ProtectedFromOverwrite: true
                ADCMinimumInputVoltage: -.02
                ADCMaximumInputVoltage: .02
                ADCBits: 16
            END
            OversampleEnabled: False
            InitialReceiveTime: 0.0 // Seconds
            RepetitionInterval: 1.0e-3 // Seconds
            ExpectedRadioFrequency: 8.46e9 // Hertz
            ExpectedBandwidth: 20e6 // Hertz
            NoiseFigure: 0.0 // dB
           DisableThermalNoise: true
            RecordedPDWFilename: angry_dingo_a
            BEGIN TransducerAssociation
                Name: Rx1
            END
        END
    END
END
```

Appendix IV Example Emitter Configuration File (Dingo)

```
#Dwell then switch, PRF: 90.9 KHz, 114.9 KHz. Duty cycle 15%.

DATA GLOBAL TransmitEventEntry angry_dingo_a Acquisition Dwell_And_Switch
    100, 1.1e-5,  850, 8.46e9, 1.65e-6,  0, Pulse
    120, 8.7e-6, 850, 8.46e9, 1.305e-6, 0, Pulse
END

BEGIN TransducerSinc Tx-angry_dingo_a
    BEGIN GeometryObjectAttachment
        Offset: 0.0, 0.0, 0.0
        Orientation: 0.0, 0.0, 0.0
        BEGIN KinematicsControllerRotationalAzimuthPaddleScan
            ScanTime: 0.1
            ScanWidthDeg: 40.0
           ScanHeightDeg:40.0
            ScanStartDirection: NEGATIVE
        END
    END
    Frequency: 8.46e9
    Bandwidth: 20e6
    PeakGain: 40.0
    AzimuthBeamwidth: 1
    ElevationBeamwidth: 1
END
```

Appendix V Python Preprocessing Script

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
UNCLASSIFIED

Data Preprocessing
Ingests IQ from Midas 1000 files and produces a collection of time chunks for
ML experiments. Optionally produces spectrograms.

Enforce strict train/val/test split using the following directory structure:

output:
|---data
|   |---train
|   |   |---emitter_a
|   |   |   |---snr_1
|   |   |   |   |---0.json
|   |   |   |   |---0.png
|   |   |   |   |...
|   |   |   |---snr_2
|   |   |   |   |...
|   |   |---emitter_b
|   |   |...
|   |---validate
|   |   |...
|   |---test
|   |   |...

Chris Ebersole
AFRL/RYWE
2021
"""

import os
import time
import json
import pandas as pd
import numpy as np
from scipy import signal as sps
from scipy import io as sio
# import skimage
from matplotlib import pyplot as plt
from matplotlib import patches
import argparse
from midas_tools import MidasFile
# import torch
import imageio
from dask import dataframe as ddf

# Column labels for DEWM outputs
# Note space at beginning of labels...
EMITTER_LABEL = " Emitter ID"
TOA_LABEL = " Rx Time (sec)"
PW_LABEL = " Pulse Width (sec)"
FREQ_LABEL = " Carrier Frequency (Hz)"

UNKNOWN_LABEL = "unknowns"
```

```
RX_RF_CENTER = 8.5e9

NOISE_POWER_LIN = 1
FREQ_PAD_PX = 0  # on either side of bbox. Just do this after the fact.
TIME_PAD_PX = 0


class DatasetBuilder:

    def __init__(self):
        # Default parameters, overwritten by CLI
        in_root_path = os.path.join("data")
        out_root_path = in_root_path

        data_type = "spectrogram"
        soi_list = []  # Empty = all
        exclude_list = []  # Empty = none
        window_len_sec = 0.5e-3  # 0.5e-3
        seg_len_samp = 1024
        fft_size = 1024
        snr_list = [-40, -30, -20, -10, 0, 10, 20]
        output_sample_rate = 1e9

        num_per_class_per_snr = 1000
        pct_train = 0.7
        pct_validate = 0.15
        pct_test = 0.15

        pct_two_signal = 0.75
        pct_three_signal = 0.25

        task = "classification"
        rng_seed = None
        if (task == "classification"
                or task == "classification_detection"
                or task == "detection"):
            num_processes = 20
        elif task == "visualize" or task == "visualize_multiple":
            num_processes = 1

        if (task == "detection"
                or task == "classification_detection"
                or task =="visualize_multiple"):
            raise ValueError("Data preprocessing for multi-target detection "
                             "not included in this version.")
        ##########

        # Parse CLI arguments
        parser = argparse.ArgumentParser(description="Generate train/validate/"
                                                     "test datasets from raw "
                                                     "sampled data")
        parser.add_argument("--task",
                            type=str,
                            help="Task to perform",
                            choices=["classification",
                                     "visualize"],
                            default=task)
        parser.add_argument("--in_root_path",
                            type=str,
                            help="Path to root input data directory",
```

```
                    default=in_root_path)
parser.add_argument("--type",
                    type=str,
                    help="Dataset type (default=signal)",
                    choices=["raw", "spectrogram", "stft"],
                    default=data_type)
parser.add_argument("--out_root_path",
                    type=str,
                    help="Path to root output directory",
                    default=out_root_path)
parser.add_argument("--soi_list",
                    help="List of SOIs, corresponding to filenames of "
                    "iq/pdw files and output directories.",
                    nargs="+",
                    default=soi_list)
parser.add_argument("--exclude_list",
                    help="List of emitter filenames to be excluded.",
                    nargs="+",
                    default=exclude_list)
parser.add_argument("--window_len_sec",
                    type=float,
                    help="Window length in seconds",
                    default=window_len_sec)
parser.add_argument("--seg_len_samp",
                    type=int,
                    help="Length of each STFT segment (time) in "
                         "samples",
                    default=seg_len_samp)
parser.add_argument("--fft_length",
                    type=int,
                    help="Size of FFT to be used",
                    default=fft_size)
parser.add_argument("--snr_list",
                    help="List of desired output SNRs",
                    nargs="+",
                    default=snr_list)
parser.add_argument("--sample_rate",
                    help="Output sample rate in hertz",
                    default=output_sample_rate)
parser.add_argument("--num_examples",
                    type=int,
                    help="For classification, number of examples per "
                         "class per snr.",
                    default=num_per_class_per_snr)
parser.add_argument("--pct_train",
                    type=float,
                    help="Percent of data to use in training",
                    default=pct_train)
parser.add_argument("--pct_validate",
                    type=float,
                    help="Percent of data to use in validation",
                    default=pct_validate)
parser.add_argument("--pct_test",
                    type=float,
                    help="Percent of data to use in testing",
                    default=pct_test)
parser.add_argument("--rng_seed",
                    type=int,
                    help="RNG seed",
                    default=rng_seed)
```

```
    parser.add_argument("--num_processes",
                        type=int,
                        help="Number of parallel processes",
                        default=num_processes)
    args = parser.parse_args()

    # Extract arguments from argparser
    self.params = {"task": args.task,
                   "data_type": args.type,
                   "in_root_path": args.in_root_path,
                   "iq_path": os.path.join(args.in_root_path, "iq"),
                   "pdw_path": os.path.join(args.in_root_path, "pdw"),
                   "config_path": os.path.join(args.in_root_path,
                                               "config"),
                   "out_root_path": args.out_root_path,
                   "soi_list": args.soi_list,
                   "exclude_list": args.exclude_list,
                   "window_len_sec": args.window_len_sec,
                   "seg_len_samp": args.seg_len_samp,
                   "fft_size": args.fft_length,
                   "snr_list": args.snr_list,
                   "output_sample_rate": args.sample_rate,
                   "num_per_class_per_snr": args.num_examples,
                   "pct_train": args.pct_train,
                   "pct_validate": args.pct_validate,
                   "pct_test": args.pct_test,
                   "num_train": round(args.pct_train * args.num_examples),
                   "num_validate": round(args.pct_validate
                                         * args.num_examples),
                   "num_test": round(args.pct_test * args.num_examples),
                   "rng_seed": args.rng_seed,
                   "rng_state": np.random.RandomState(args.rng_seed),
                   "num_processes": args.num_processes}

    # Argument check
    assert args.pct_train + args.pct_validate + args.pct_test == 1, \
        "Data split percentages do not sum to one!"

def create_dataset(self):
    # Track time required to run script
    total_dataset_time = 0

    # Construct soi_list for all tasks
    # If no SOI are specified, use all tmp files in IQ directory
    if len(self.params["soi_list"]) == 0:
        soi_list = [x.rsplit(".", 1)[0]
                    for x in os.listdir(self.params["iq_path"])
                    if x.endswith(".tmp")]
        self.params["soi_list"] = soi_list
    else:
        soi_list = self.params["soi_list"]

    # If any signals are to be excluded, remove them here
    for unwanted_emitter in self.params["exclude_list"]:
        if unwanted_emitter in soi_list:
            soi_list.remove(unwanted_emitter)

    # All tasks involving single-emitter spectrograms
    if (self.params["task"] == "classification"
            or self.params["task"] == "classification_detection"
```

```
        or self.params["task"] == "visualize"):

    # Loop through IQ files and build the dataset
    for soi in soi_list:
        soi_start_time = time.time()
        print(f"Processing {soi}")

        # Open PDW file
        pdw_file = os.path.join(self.params["pdw_path"], f"{soi}.csv")
        pdws = pd.read_csv(pdw_file,
                           usecols=[TOA_LABEL, PW_LABEL, FREQ_LABEL])

        # Extract Rx frequency offset from scenario (DEWM) file
        freq_offset_hz = self.get_freq_offset(soi)

        # Oversample PDWs if necessary
        total_examples = (len(self.params["snr_list"])
                          * (self.params["num_train"]
                             + self.params["num_validate"]
                             + self.params["num_test"])
                          )
        if total_examples > len(pdws):
            replace = True
        else:
            replace = False

        # Shuffle the pdws and reset the index
        pdws = pdws.sample(n=total_examples,
                           replace=replace,
                           random_state=self.params["rng_state"]
                           ).reset_index(drop=True)

        # Assign SNR and partition to each PDW
        pdw_params = []
        for snr in self.params["snr_list"]:
            for idx in range(self.params["num_train"]):
                pdw_params.append((snr, "train", idx))
            for idx in range(self.params["num_validate"]):
                pdw_params.append((snr, "validate", idx))
            for idx in range(self.params["num_test"]):
                pdw_params.append((snr, "test", idx))

        # Loop through the selected pdws, gathering signals and
        # annotations, and writing them to the directory
        # for idx, this_pdw in pdws.iterrows():
        # For debugging, use scheduler="single-threaded"
        df_dask = ddf.from_pandas(
            pdws,
            npartitions=self.params["num_processes"]
            )
        if (self.params["task"] == "classification"
                or self.params["task"] == "classification_detection"):
            df_dask.apply(self.generate_single_signal_data,
                          axis=1,
                          meta=str,
                          args=(pdw_params,
                                freq_offset_hz,
                                soi)
                          ).compute(scheduler="multiprocessing")
                        # ).compute(scheduler="single-threaded")
```

```
            elif self.params["task"] == "visualize":
                df_dask.apply(self.generate_single_signal_data,
                              axis=1,
                              meta=str,
                              args=(pdw_params,
                                    freq_offset_hz,
                                    soi)
                              ).compute(scheduler="single-threaded")

            soi_time = time.time() - soi_start_time
            total_dataset_time += soi_time
            print(f"soi_time = {soi_time / 60} min")

    print("Dataset complete! Total time = "
          f"{total_dataset_time / 60 / 60 :.4} hrs")

def generate_single_signal_data(self,
                                this_pdw,
                                pdw_params,
                                freq_offset_hz,
                                soi):
    # Extract arguments from primary signal
    idx = this_pdw.name
    snr_desired_db, partition, file_idx = pdw_params[idx]
    rx_time = this_pdw[TOA_LABEL]
    pulse_width = this_pdw[PW_LABEL]

    # Prepare output file path
    outpath = os.path.join(self.params["out_root_path"],
                           partition,
                           soi,
                           f"snr_{snr_desired_db}")
    os.makedirs(outpath, exist_ok=True)
    outpath = os.path.join(outpath, f"{file_idx}")

    # Use snr_desired_db to select sig_power_lin
    snr_desired_lin = 10**(snr_desired_db / 10)
    sig_power_lin = snr_desired_lin * NOISE_POWER_LIN

    # Preprocess signal (upsample, shift, noise)
    if self.params["data_type"] == "raw":
        meta, signal = self.process_signal(soi,
                                           rx_time,
                                           pulse_width,
                                           freq_offset_hz,
                                           sig_power_lin,
                                           snr_desired_db)

        # Scale to desired SNR
        siglen = signal.size
        noise = np.sqrt(NOISE_POWER_LIN / 2) * (
            np.random.randn(siglen) + 1j * np.random.randn(siglen)
            )
        sig_power_meas = np.linalg.norm(signal)**2 / siglen
        noise_power_meas = np.linalg.norm(noise)**2 / siglen
        snr_meas_lin = sig_power_meas / noise_power_meas
        err = 10*np.log10(
            abs(sig_power_lin / noise_power_meas - snr_desired_lin)
            )
        assert err < -6, f"Incorrect SNR! Error = {err} dB"
```

```
        signal = signal + noise

        # Dear god, find a format with compression. Midas?
        # Don't remove this unless you have lots of disk space...
        raise NotImplementedError("Compression badly needed...")
        np.save(outpath + ".npy", signal)

    elif (self.params["data_type"] == "spectrogram"
          or self.params["data_type"] == "stft"):
        (meta,
         freq,
         time,
         stft) = self.process_signal(soi,
                                     rx_time,
                                     pulse_width,
                                     freq_offset_hz,
                                     sig_power_lin,
                                     snr_desired_db)

        # Scale to desired SNR
        siglen = meta["metadata"]["if_len"]
        noise = np.sqrt(NOISE_POWER_LIN / 2) * (
            np.random.randn(siglen) + 1j * np.random.randn(siglen)
            )
        _, _, noise = sps.stft(noise,
                               fs=self.params["output_sample_rate"],
                               window="hann",
                               nperseg=self.params["seg_len_samp"],
                               nfft=self.params["fft_size"],
                               return_onesided=False,
                               noverlap=self.params["seg_len_samp"]//2,
                               )

        stft = stft + noise
        if self.params["data_type"] == "spectrogram":
            Sxx = np.abs(stft)**2
            Sxx = Sxx / np.max(Sxx)
            Sxx = (Sxx*(2**8 - 1)).astype(np.uint8)
            Sxx = np.fft.fftshift(Sxx, axes=0)

            if self.params["task"] != "visualize":
                # After some testing, I don't believe saving as a png does
                # any other significant compression or quantization...
                # torch.save(Sxx, outpath)
                imageio.imwrite(outpath + ".png", Sxx)

        else:
            stft = np.fft.fftshift(stft, axes=0)

            if self.params["task"] != "visualize":
                sio.savemat(outpath + ".mat",
                            {"stft_i":np.real(stft).astype(np.float16),
                             "stft_q":np.imag(stft).astype(np.float16)})

                # tmp = sio.loadmat(outpath)


        freq = np.fft.fftshift(freq)
```

```
    if self.params["task"] != "visualize":
        # Either type of signal, write annotations to file
        with open(outpath + ".json", "w") as f:
            json.dump(meta, f)

    if self.params["task"] == "visualize":
        if (self.params["data_type"] == "raw"
                or self.params["data_type"] == "stft"):
            raise NotImplementedError("Add this!")

        plt.figure()
        plt.pcolormesh(1e3 * time,
                       1e-6 * freq,
                       Sxx,
                       vmin=0,
                       vmax=2**8 - 1
                       )
        bbox_sec_hz = meta["annotation"]["bbox_sec_hz"]
        rect = patches.Rectangle(xy=(1e3*bbox_sec_hz[0],
                                     1e-6*bbox_sec_hz[2]),
                                 width=1e3*(bbox_sec_hz[1]
                                            - bbox_sec_hz[0]),
                                 height=1e-6*(bbox_sec_hz[3]
                                              - bbox_sec_hz[2]),
                                 alpha=0.2,
                                 linewidth=2,
                                 facecolor="white",
                                 edgecolor="red")
        # print(rect)
        plt.gca().add_patch(rect)
        plt.xlabel("Time (ms)")
        plt.ylabel("Frequency (MHz)")
        plt.colorbar()
        plt.title("Spectrogram")
        plt.show()

        # Note: factor of 2 hard coded for overlapping segments
        print("Spectrogram properties:")
        print(f"Sxx shape\t\t= {Sxx.shape[0]} px (freq), "
              f"{Sxx.shape[1]} px (time)")
        print("Time resolution\t\t= "
              f"{1e6 * meta['metadata']['time_res']} us")
        print("Frequency resolution\t= "
              f"{1e-6 * meta['metadata']['freq_res'] :.6} MHz")
        print(f"Time extent\t\t= {1e3 * self.params['window_len_sec']} ms")

        print("Signal time range\t= "
              f"{1e3*bbox_sec_hz[0] :.6}, {1e3*bbox_sec_hz[1] :.6} ms")
        print("Signal freq range\t= "
              f"{1e-6*bbox_sec_hz[2] :.6}, {1e-6*bbox_sec_hz[3] :.6} MHz")

        print(meta["metadata"]["if_len"])
        raise Exception("Stop here!")

    # To appease Dask
    return("")

def get_freq_offset(self, soi):
    # The simulated receiver frequency is given in the scenario file.
    # Here we assume the scenario file is in a subdirectory of config_path
```

```python
    if "_" in soi:
        soi_root = soi.split("_")[1]
    else:
        soi_root = soi

    # Open scenario file (search through all subfolders of config
    # directory)
    file_matches = []
    for dirpath, dirnames, filenames in os.walk(
            self.params["config_path"]):
        for file in filenames:
            if file == f"{soi_root}.cfg":
                file_matches.append(os.path.join(dirpath, file))
    assert len(file_matches) != 0, ("No scenario file found for "
                                    f"{soi_root}!")
    assert len(file_matches) < 2, ("Conflicting scenario files for "
                                   f"{soi_root}!")

    # Would be much better to make a robust parser... This works for now.
    with open(file_matches[0], "rt") as f:
        center_frequencies = []
        for line in f:
            if any(x in line for x in ("BEGIN", "END")):
                pass

            elif line == "\n":
                pass

            elif "frequency" in line.lower():
                line_components = line.strip().split()
                for x in line_components:
                    if x[0].isnumeric():
                        center_frequencies.append(float(x))

    # Check to make sure Tx transducer and Rx are tuned to same frequency
    assert np.unique(center_frequencies).size == 1, \
        f"{soi_root}.cfg uses inconsistent Tx/Rx frequencies"

    return(center_frequencies[0])

def process_signal(self,
                   soi,
                   rx_time,
                   pulse_width,
                   freq_offset_hz,
                   sig_power_lin,
                   snr_desired_db):

    # To keep things compact, grab needed parameters
    soi = soi.strip()
    iq_path = self.params["iq_path"]
    window_len_sec = self.params["window_len_sec"]
    rng_state = self.params["rng_state"]
    output_sample_rate = self.params["output_sample_rate"]
    seg_len_samp = self.params["seg_len_samp"]
    fft_size = self.params["fft_size"]
    data_type = self.params["data_type"]
    eff_fft_size = min((fft_size, seg_len_samp))
    time_res = seg_len_samp / output_sample_rate / 2
    freq_res = output_sample_rate / eff_fft_size
```

```
# Open Midas file, grab signal information, create time array
iq_file = os.path.join(iq_path, f"{soi}.tmp")
mf = MidasFile(iq_file)
sample_rate_hz = mf.sample_rate
time = np.arange(0, window_len_sec, 1/sample_rate_hz)

# Load sampled data for this pulse
# Randomly shift so the pulse could appear anywhere in the window
# Require at least half the pulse is in the window
t0 = rx_time - (rng_state.choice(time) - pulse_width / 2)
t0 = max(t0,
         0,  # Should be no less than start of file
         rx_time + pulse_width / 2 - window_len_sec)  # Right edge
t0 = min(t0,
         mf.data_duration-window_len_sec)  # Don't exceed file length

bb_signal = mf.read_at_time(t0, window_len_sec, reset_time=True)

# Close Midas file
mf.fp.close()

# Assuming signal is noiseless. Scale so we don't deal with
# floating point issues.
bb_signal = bb_signal / np.max(np.abs(bb_signal))

# Interpolate to IF.
up_rate = round(output_sample_rate / sample_rate_hz)
# oversample_rate = 4
# if_len = up_rate * len(time)
if_len = round(window_len_sec * output_sample_rate)

# Could use zero-order hold for DC signals. For now let's stick
# with windowed FFT method.
dc_only = False  # Could get this from config files
if dc_only:
    if_signal = bb_signal.repeat(up_rate)
else:
    if_signal = sps.resample(bb_signal,
                             if_len,
                             window="hamming")

# Experimented with interpolation filter. Sticking with FFT method.

# Design an FIR filter for upsampling. Windowing method is far
# preferable here.
# bands = [0, 20e6, 20.1e6, oversample_rate * output_sample_rate/2]
# filt_ls = sps.firls(9999,
#                         bands=bands,
#                         desired=[1, 1, 0, 0],
#                         weight=[.7, .3],
#                         fs=oversample_rate * output_sample_rate)
# filt_win = sps.firwin2(9999,
#                            freq=bands,
#                            gain=[1, 1, 0, 0],
#                            nfreqs=2**15,
#                            window="hamming",
#                            fs=oversample_rate * output_sample_rate)
# freq, response_ls = sps.freqz(filt_ls,
#                                   fs=(oversample_rate
```

```
#                                  * output_sample_rate),
#                              worN=2**15)
# _, response_win = sps.freqz(filt_win,
#                              fs=(oversample_rate
#                                  * output_sample_rate),
#                              worN=2**15)
# plt.figure()
# plt.plot(freq, 10*np.log10(np.abs(response_ls)), label="FIRLS")
# plt.plot(freq, 10*np.log10(np.abs(response_win)), label="FIRWIN2")
# plt.legend()

# Interpolate!
# if_signal = sps.upfirdn(filt_win,
#                            bb_signal,
#                            up=up_rate * oversample_rate,
#                            down=oversample_rate)

# Update time indices, mix signal to correct center frequency
# time = np.arange(0, window_len_sec, 1/output_sample_rate)
time = np.arange(if_len) * 1/output_sample_rate
if_signal = if_signal * np.exp(
    2j * np.pi * (freq_offset_hz - RX_RF_CENTER) * time
    )

# Use noiseless signal to measure ground truth bandwidth and time.
threshold = 0.05

if data_type == "raw":
    time_range_s = time[np.abs(if_signal) / np.max(np.abs(if_signal))
                        > threshold]
    time_range_s = np.array([time_range_s[0], time_range_s[-1]])

    if_fft = np.fft.fftshift(np.fft.fft(if_signal))
    freq = np.fft.fftshift(
        np.fft.fftfreq(if_signal.size, 1/output_sample_rate))
    freq_range_hz = freq[np.abs(if_fft) / np.max(np.abs(if_fft))
                         > threshold]
    freq_range_hz = np.array([freq_range_hz[0], freq_range_hz[-1]])

# plt.figure()
# plt.plot(1e3*time, np.real(if_signal), label="Noiseless IF", zorder=2)
# plt.plot(1e3*time_range_s, [threshold, threshold])

elif data_type == "spectrogram" or data_type == "stft":
    # Instead, use STFT. As inefficient as it is to repeat this
    # calculation twice, this will give us far better bounding boxes
    freq, time, Sxx = sps.stft(if_signal,
                               fs=output_sample_rate,
                               window="hann",
                               nperseg=seg_len_samp,
                               nfft=fft_size,
                               return_onesided=False,
                               noverlap=seg_len_samp//2,
                               )
    # NOTE HERE Sxx IS USED FOR BBOXES ONLY
    Sxx = np.abs(Sxx)**2
    Sxx = Sxx / np.max(np.abs(Sxx))
    Sxx = (Sxx*(2**8 - 1)).astype(np.uint8)
    Sxx = np.fft.fftshift(Sxx, axes=0)
```

```
    max_time = np.max(np.abs(Sxx), axis=0)
    time_range_s = time[max_time / np.max(np.abs(Sxx))
                        > threshold]
    time_range_s = np.array([time_range_s[0], time_range_s[-1]])
    time_range_s[0] -= TIME_PAD_PX * time_res
    time_range_s[1] += (TIME_PAD_PX + 1) * time_res

    freq = np.fft.fftshift(freq)
    max_freq = np.max(np.abs(Sxx), axis=1)
    freq_range_hz = freq[max_freq / np.max(max_freq) > threshold]
    freq_range_hz = np.array([freq_range_hz[0], freq_range_hz[-1]])
    freq_range_hz[0] -= FREQ_PAD_PX * freq_res
    freq_range_hz[1] += (FREQ_PAD_PX + 1) * freq_res

# Scale to desired signal power (will add noise externally)
sig_power_meas = np.linalg.norm(if_signal)**2 / if_len
scale = np.sqrt(sig_power_lin / sig_power_meas)
if_signal = scale * if_signal

# plt.plot(1e3*time, np.real(if_signal), label="Noisy IF", zorder=1)
# plt.xlabel("Time (ms)")
# plt.legend()
# plt.title("Noisy IF")

# plt.figure()
# bb_fft = np.fft.fftshift(np.fft.fft(bb_signal))
# if_fft = np.fft.fftshift(np.fft.fft(if_signal))
# freq_bb = np.fft.fftshift(
#     np.fft.fftfreq(bb_signal.size, 1/sample_rate_hz))# + RX_RF_CENTER
# freq_if = np.fft.fftshift(
#     np.fft.fftfreq(if_signal.size, 1/output_sample_rate))# + RX_RF_CENTER
# plt.plot(1e-6*freq_if,
#          np.abs(if_fft)/np.max(np.abs(if_fft)),
#          label="IF")
# plt.plot(1e-6*freq_bb,
#          np.abs(bb_fft)/np.max(np.abs(bb_fft)),
#          label="BB")
# plt.xlabel("Frequency (MHz)")
# plt.legend()
# plt.title("Frequency shifted signal")

# JSON annotations for detection task
# If plotting with pcolormesh and correct time/freq axes, use
# bbox_sec_hz, else use bbox_px
json_dict = {"annotation": {},
             "metadata": {},
             "emitter_name": soi}
bbox_sec_hz = [time_range_s[0],
               time_range_s[1],
               freq_range_hz[0],
               freq_range_hz[1]]
json_dict["annotation"]["bbox_sec_hz"] = bbox_sec_hz
bbox_px = [int(np.argmin(np.abs(time - time_range_s[0]))),
           int(np.argmin(np.abs(time - time_range_s[1]))),
           int(np.argmin(np.abs(freq - freq_range_hz[0]))),
           int(np.argmin(np.abs(freq - freq_range_hz[1])))]
json_dict["annotation"]["bbox_px"] = bbox_px

# Also, save some useful metadata!
# NOTE: Factor of two hard coded in time_res for segment overlap!!
```

```
        #       Fix that at some point...
        json_dict["metadata"]["snr_db"] = snr_desired_db
        json_dict["metadata"]["sample_rate_hz"] = output_sample_rate
        json_dict["metadata"]["window_length_sec"] = window_len_sec
        json_dict["metadata"]["seg_len_samp"] = seg_len_samp
        json_dict["metadata"]["fft_size"] = fft_size
        json_dict["metadata"]["eff_fft_size"] = eff_fft_size
        json_dict["metadata"]["time_res"] = time_res
        json_dict["metadata"]["freq_res"] = freq_res
        json_dict["metadata"]["if_len"] = if_len

        if data_type == "raw":
            return(json_dict, if_signal)

        elif data_type == "spectrogram" or data_type == "stft":
            # Todo: Make overlap accessible?
            freq, time, stft = sps.stft(if_signal,
                                        fs=output_sample_rate,
                                        window="hann",
                                        nperseg=seg_len_samp,
                                        nfft=fft_size,
                                        return_onesided=False,
                                        noverlap=seg_len_samp//2,
                                        )

            return(json_dict, freq, time, stft)



if __name__ == "__main__":

    builder = DatasetBuilder()
    builder.create_dataset()

"""
UNCLASSIFIED
"""
```