U.S. AIR FORCE

USSF

AFRL

OTFS over AFRL Terahertz Testbed

Claire Parisi, Electronics Engineer

AFRL/RITGA

Looking Ahead to Next Gen Comms

Early Generations	3G, 4G LTE	5G
800 – 1900 MHz	850 MHz - 2.5 GHz	FR1: 450 MHz – 6 GHz FR2: 24.25 GHz – 52.6 GHz
Analog → Digital	CDMA → OFDM	OFDM→ OTFS**?**

6G and Beyond!

https://venturebeat.com/mobile/fcc-opens-95ghz-to-3thz-spectrum-for-6g-7g-or-whatever-is-next/

Could OTFS work at terahertz band frequencies?

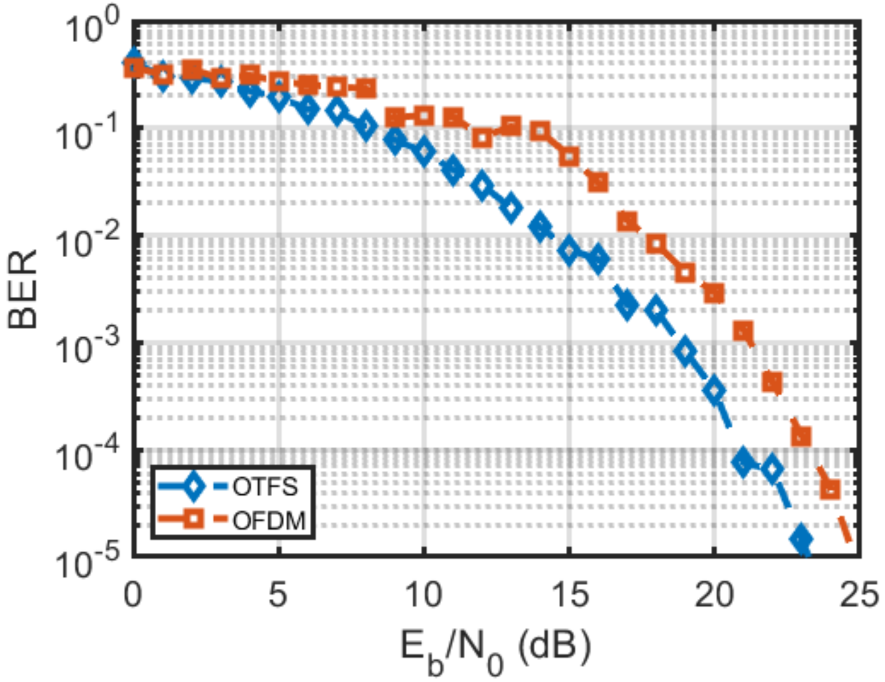
Previous Work: Modulations for Terahertz Band Communications

Context: Characterize the performance of modulation schemes in presence of PAPR and phase noise at terahertz-band frequencies

In OTFS vs OFDM power backoff from PAPR is main distinguisher of performance

$$s(t)_{\text{penalized}} = A_{\text{sat}} \frac{s(t)}{\max|s(t)|}$$

OTFS vs OFDM at 225 GHz in AWGN with PN and PAPR backoff applied



OTFS performs better in context of terahertz-band hardware impairments

C. T. Parisi, S. Badran, P. Sen, V. Petrov and J. M. Jornet, "Modulations for Terahertz Band Communications: Joint Analysis of Phase Noise Impact and PAPR Effects," in IEEE Open Journal of the Communications Society, vol. 5, pp. 412-429, 2024, doi: 10.1109/OJCOMS.2023.3344411

Outline

- AFRL Terahertz Testbed Overview
- Implementing OTFS at Terahertz Frequencies
  - Original OTFS Python Code
  - Modifying OTFS Python Code for Terahertz Testbed
  - Instrument Control & Workflow
  - Testing Set Up
- Results
- Summary
- Opportunities for Extension

AFRL Terahertz Testbed Overview

AFRL Terahertz Testbed

Instrument Control Messages

← Digital Tx Signal

Computer with MATLAB

- Instrument control for automated data collection

Analog Tx Signal

Keysight M8194A Arbitrary Waveform Generator (AWG)

- Converts input into electrical signals
- High-speed DAC: 120GSa/s

IF Port

LO Port

Wireless Tx Signal

Mixer Amplifier Multiplier Chain (MixAMC) Upconverter

- Mixes AWG output with up-converted LO tone and amplifies the signal for transmission

LO Signal

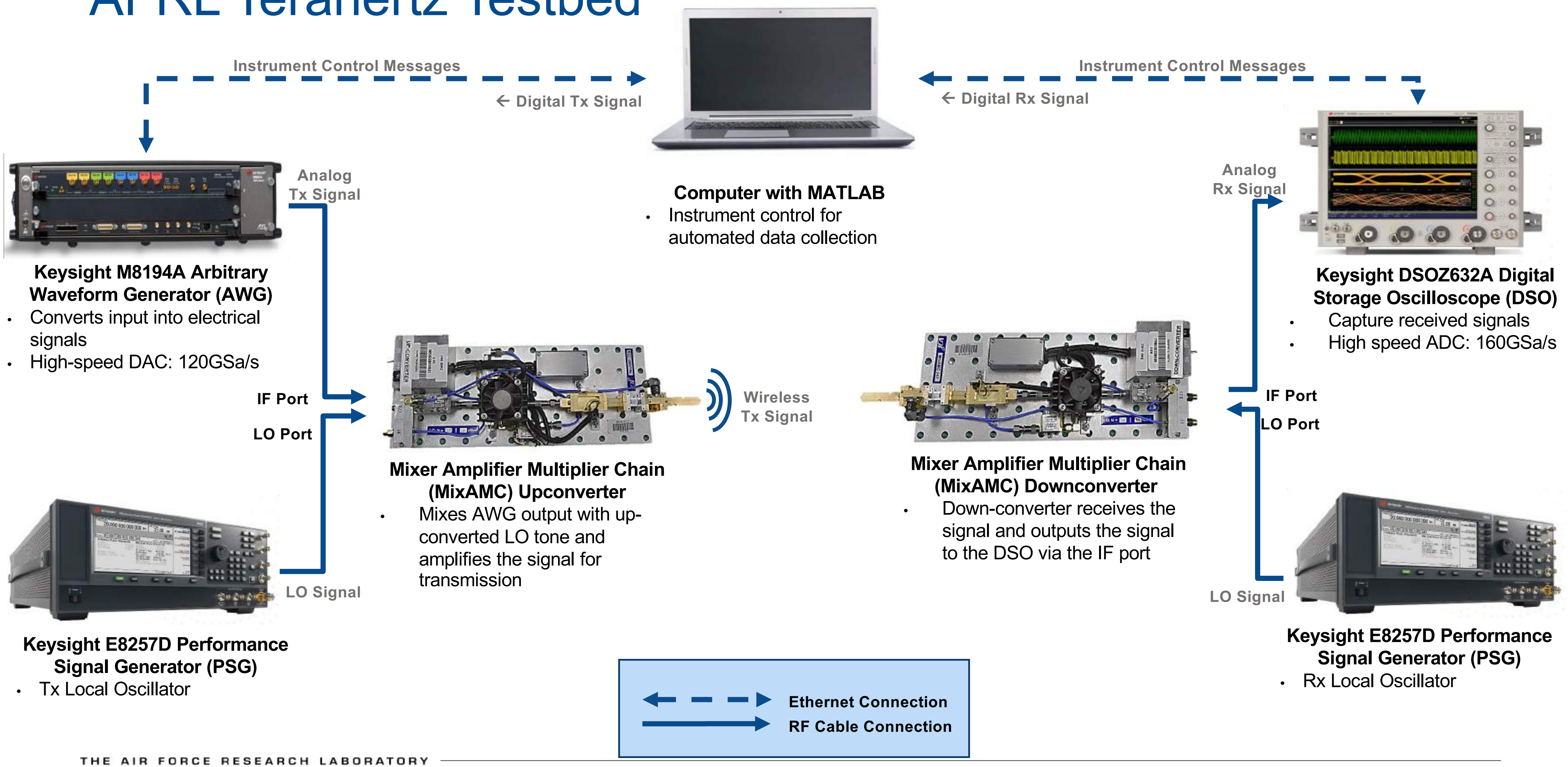
Keysight E8257D Performance Signal Generator (PSG)

- Tx Local Oscillator

Ethernet Connection

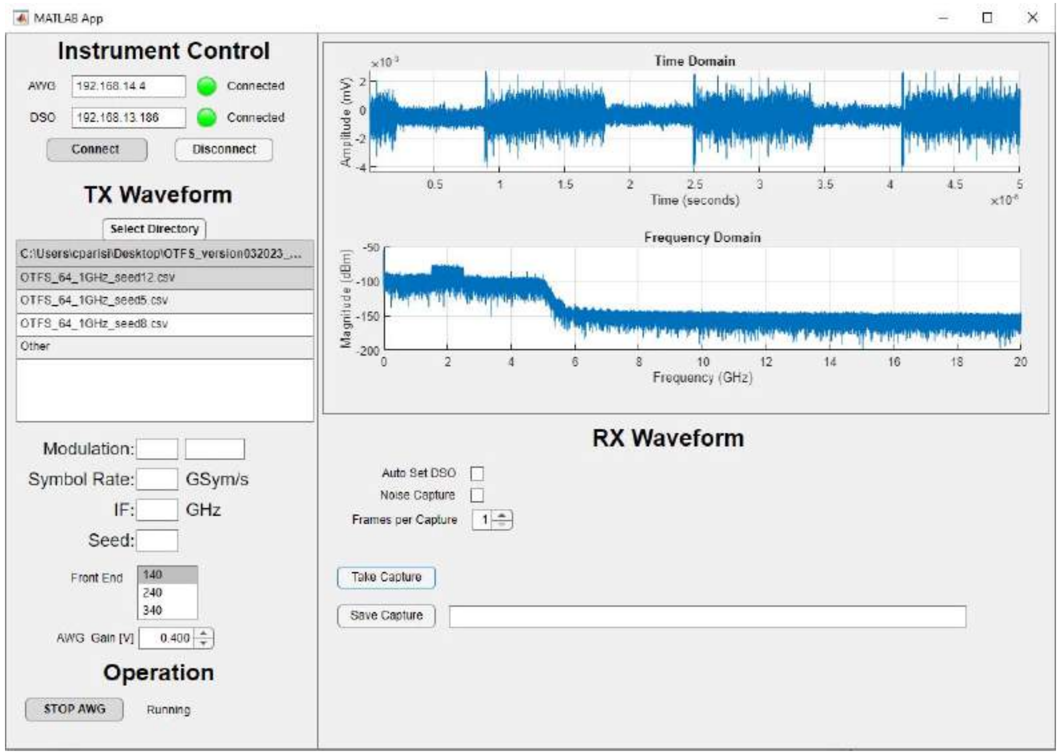
RF Cable Connection

AFRL Terahertz Testbed

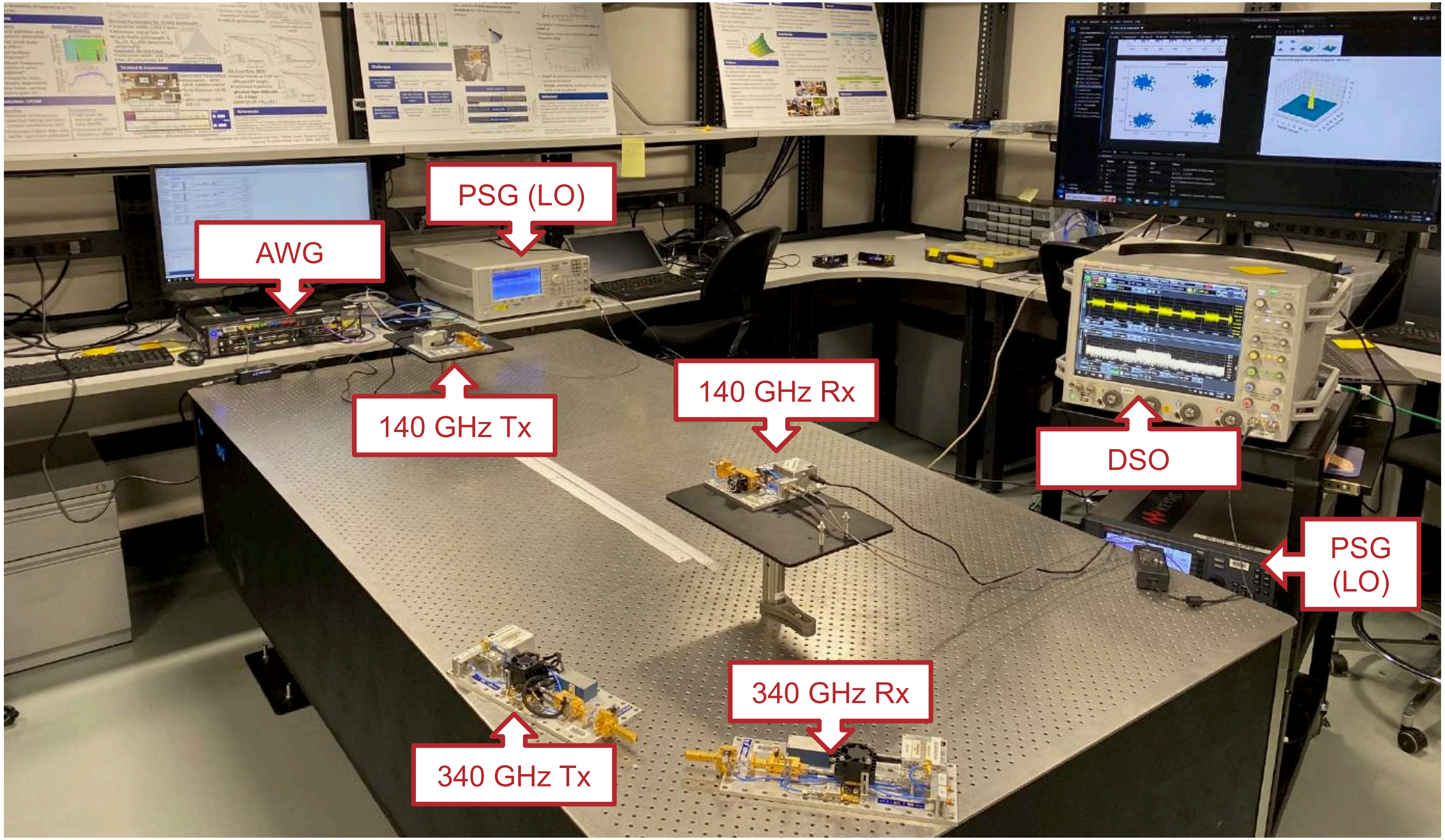


Instrument Control Software

- Designed by AFRL team for control and automation of test equipment
  - Uses MATLAB Instrument Control Toolbox to connect and communicate through VISA comms architecture
  - Custom GUI for easy user interaction
- Key Capabilities:
  - Upload user-designed signals or build signal with preset options
  - Control equipment from single device with ethernet connection
  - Design experiments and automate testing processes
  - Rapidly transmit, receive, and save wireless signals
  - Built-in file naming option for capture data



AFRL Terahertz Testbed



Implementing OTFS at Terahertz Frequencies

Original OTFS Python Code

Set Parameters

Generate OTFS, Send Through Channel, and Recover

Plotting

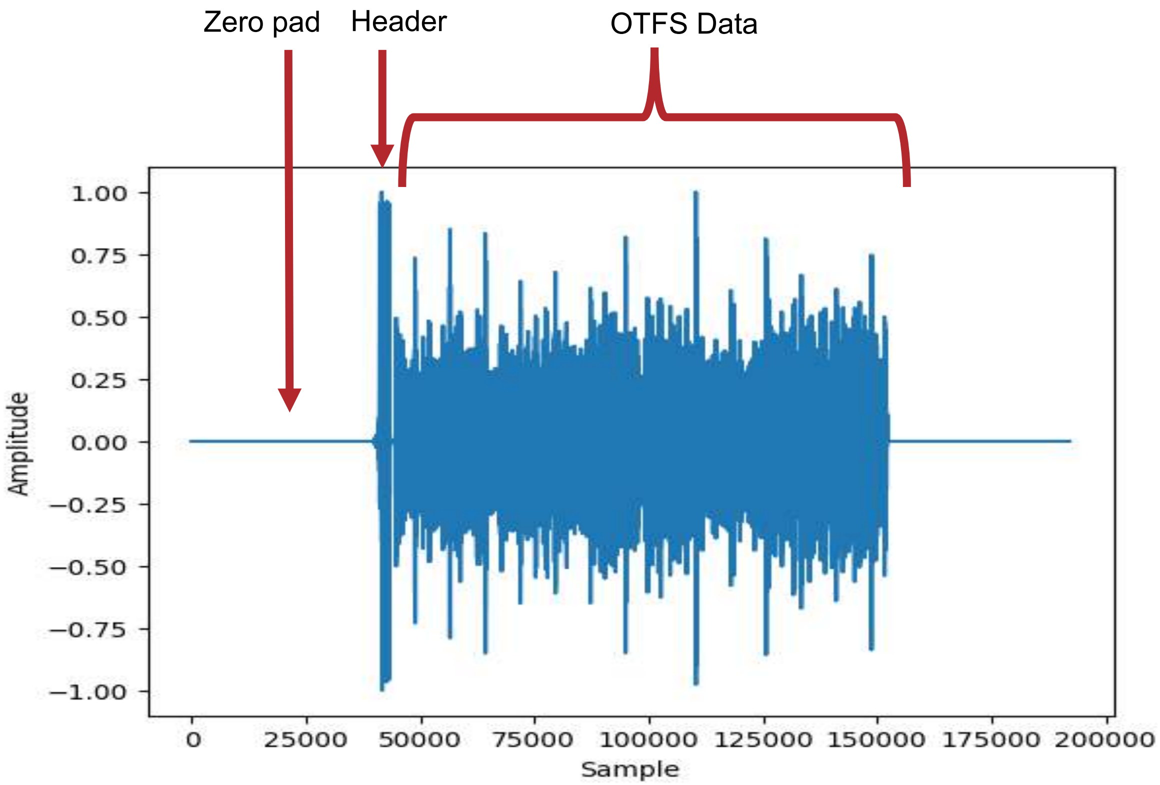
```python
import numpy as np
from Data_set import Data_set
import matplotlib.pyplot as plt

OTFS_parameter = \
{
    'N_c': 1024, # number of subcarriers
    'N_slot': 14, # number of time slot per sub_frame
    'm': 2, # modulation order, 2^m - QAM
    'N_t': 1, # only works in SISO case
    'N_r': 1, # only works in SISO case
    'mobility_speed': 0, # in km/h
    'kernel_size': (14, 48),
    'kernel_index': np.array([[7, 520]]),  # antenna, doppler, delay
    'pilot_pattern': 'spike', # spike, block
    'SNR': 30,
    'car_fre': 4 * 10**9, # carrier frequency
    'delta_f': 15.0 * 10 ** 3, # subcarrier spacing
}

DS = Data_set(OTFS_parameter)

DS.Generate()

def ISFFT(x):
    y = np.fft.fft(np.fft.ifft(x, norm='ortho', axis=-2), norm='ortho', axis=-1)
    return y

Tx_symbols = DS.Tx_symbols_4D[0,0,:,:]
Rx_symbols = DS.Rx_symbols_2D

x = np.arange(0, DS.N_slot)
y = np.arange(0, DS.N_c)
Y, X = np.meshgrid(y, x)

# show transmitted subframe in delay-Doppler domain
fig2 = plt.figure()
ax = fig2.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, np.abs(Tx_symbols), rstride=1, cstride=1,cmap='viridis', edgecolor='none')
ax.set_xlabel('Doppler Spread')
ax.set_ylabel('Delay Spread')
ax.set_zlabel('$|x|$')
plt.title('Transmitted signal in delay-Doppler domain')
plt.savefig("trans_dd_eva_snr30.png")

# show received subframe in delay-Doppler domain
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, np.abs(Rx_symbols), rstride=1, cstride=1,cmap='viridis', edgecolor='none')
```

This code is limited to simulation only and not suited for implementation on hardware

We can adapt it for hardware by splitting into transmit and receive codes and making additional modifications

Modifying OTFS Python Code THz: Transmit

1. Generate OTFS signal
2. Up-sample for AWG (120 Gsa/s)
3. Apply IF
4. Add header
5. Fit to AWG requirements
   - Maximum 512,000 samples/ transmission
   - Multiple of 128 samples, requires zero padding
6. Write to file



Modifying OTFS Python Code THz: Parameters

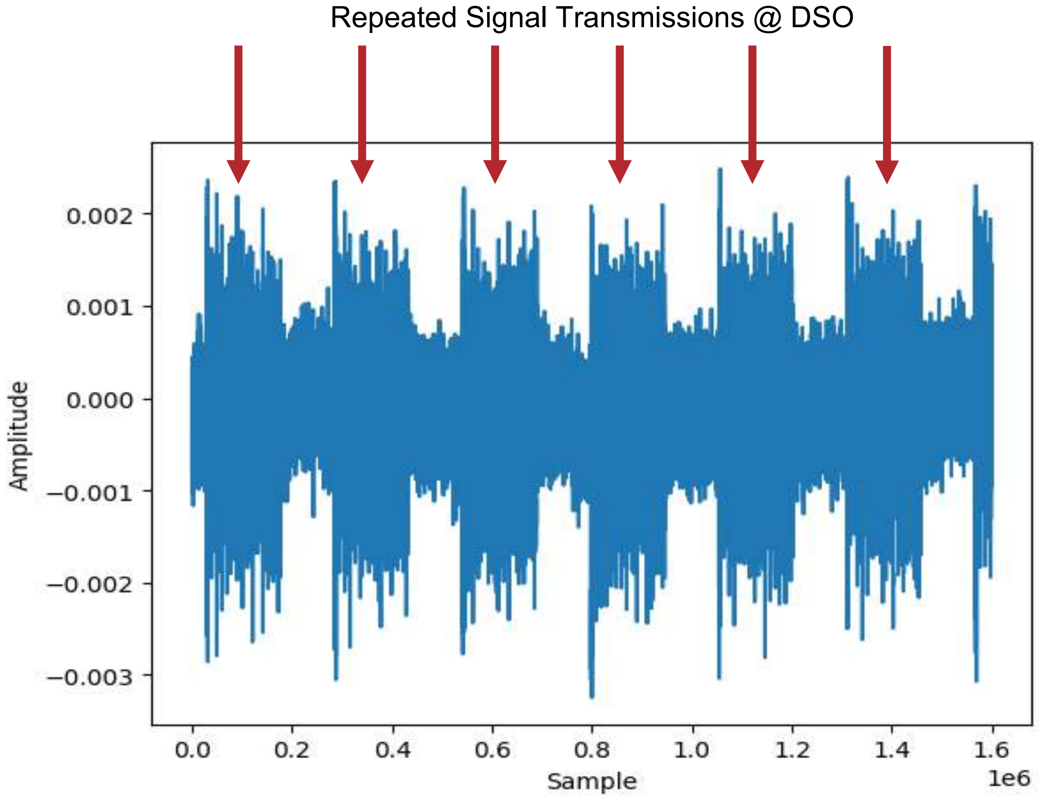
- AWG requirements
  - Maximum 512,000 samples/ transmission
  - Multiple of 128 samples, requires zero padding
  - 120 GSa/s DAC rate
- Choosing Parameters:
  - Leave time slots as is at 14
  - Low modulation order
  - Adjust subcarrier # and spacing

	Low # Subcarriers	**High # Subcarriers**
kHz Spacing	Resample factor too high leads to timeout or >512k samples	Slow generation, >512k samples
MHz Spacing	★ Meets system requirements	Requires more bandwidth, more than front-end can handle

We choose parameters such that the code can quickly generate the signal and that the signal meets the equipment specifications.

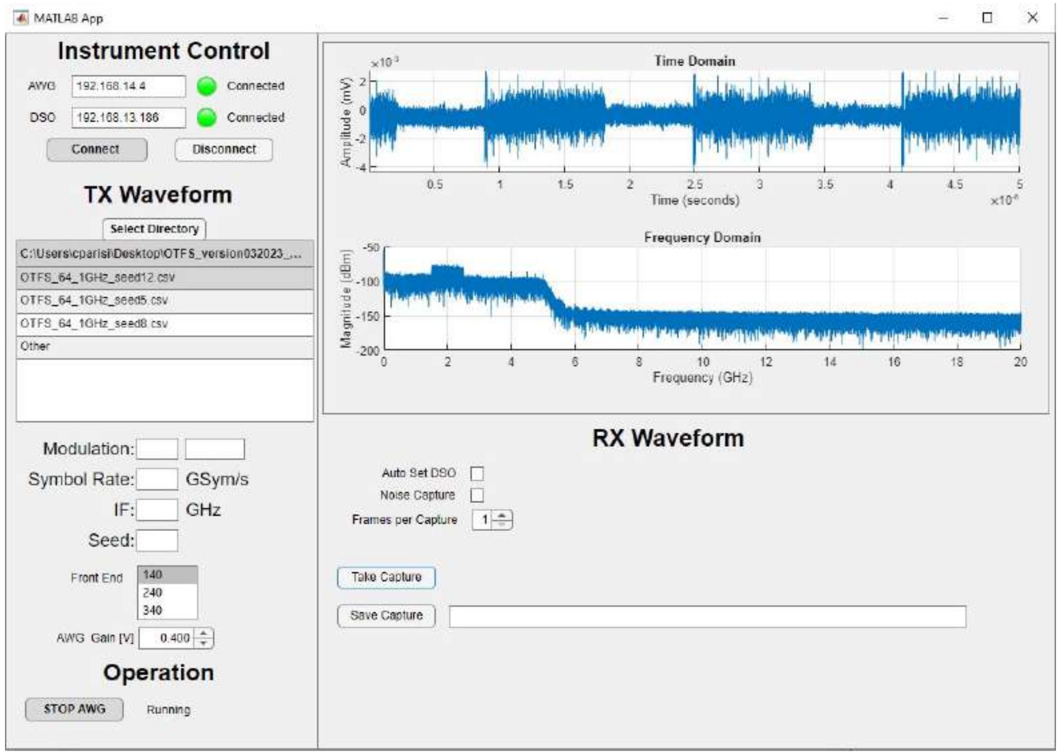
Modifying OTFS Python Code THz: Receive

1. Read in capture file
2. Down-sample from DSO
3. Remove IF
4. Locate signal
5. Process received OTFS signal
6. Plot results

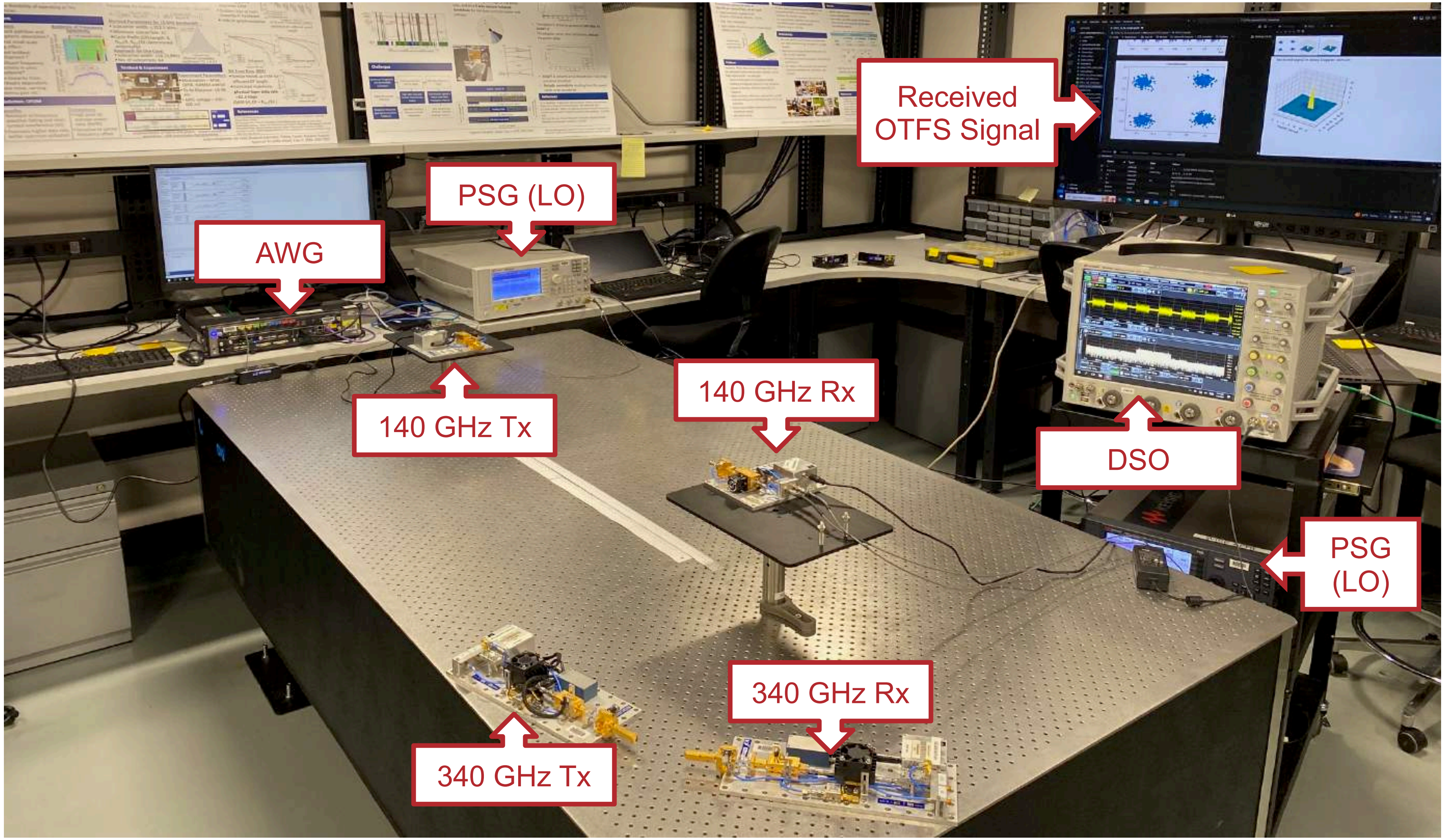


Instrument Control & Workflow

- Outputs from python (.csv) loaded into testbed instrument control software
- Instrument control software connects to instruments via ethernet, sends signal over-the-air, and saves data (.mat)
- Data read back into python for post-processing



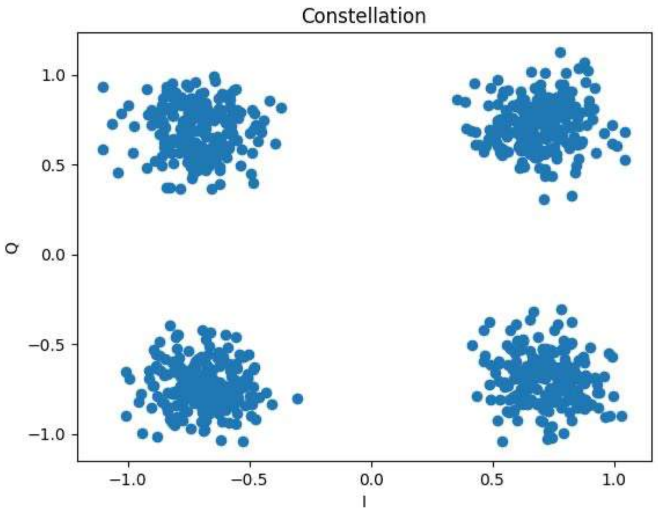
Testing Set Up



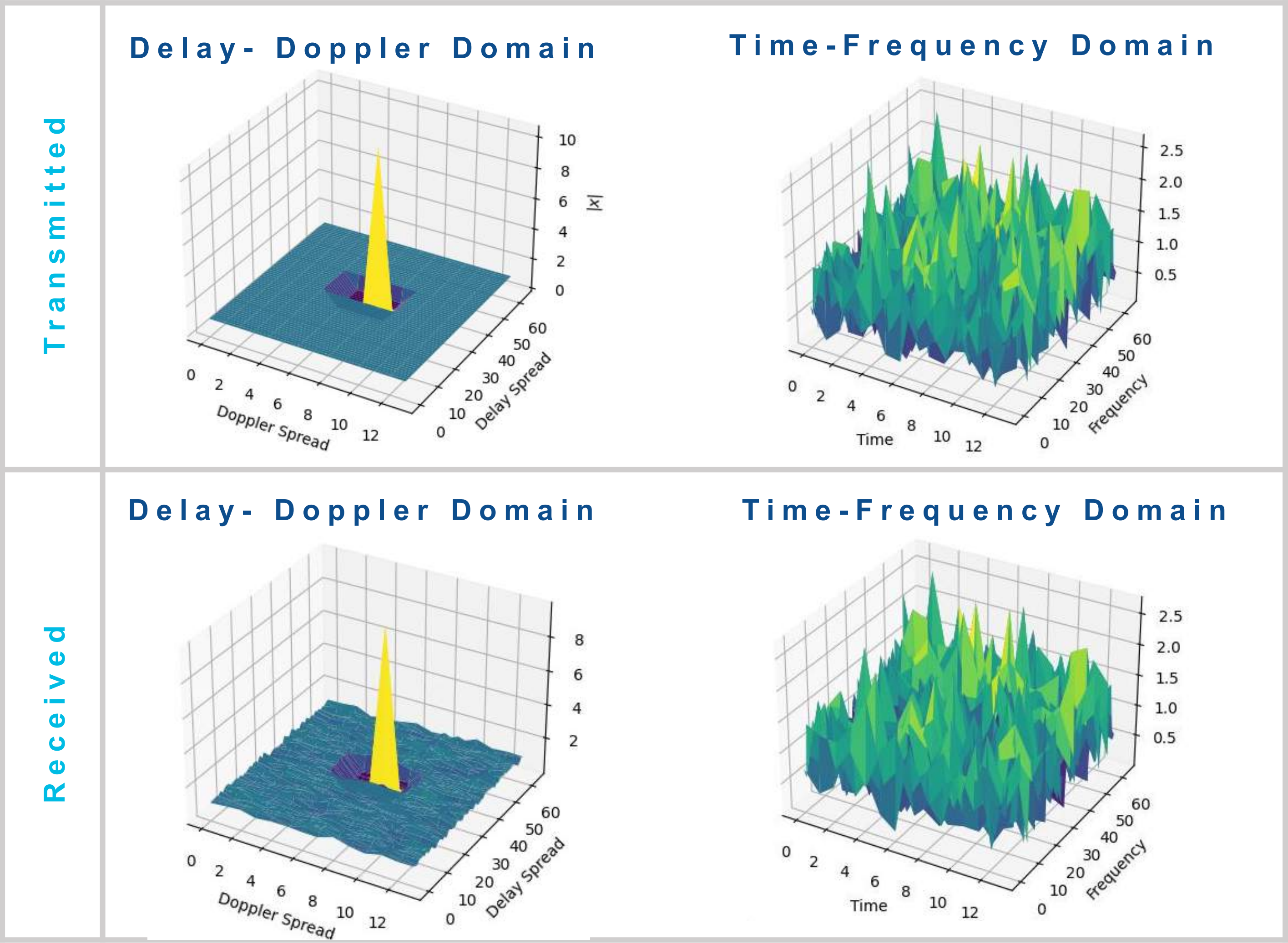
Results

140 GHz Sample Capture ~10dB SNR

Parameters	Values
Modulation	QPSK
# of subcarriers	64
Subcarrier spacing	~15 MHz
Time slots	14
Bandwidth	~1 GHz
Distance	1 m

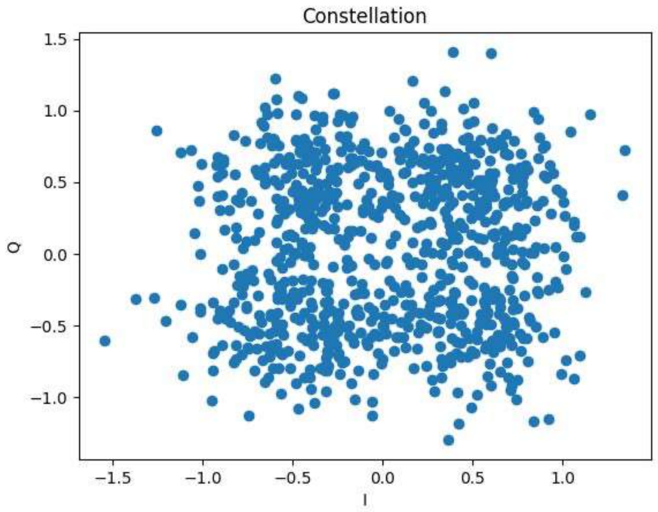


Error Free Transmission!

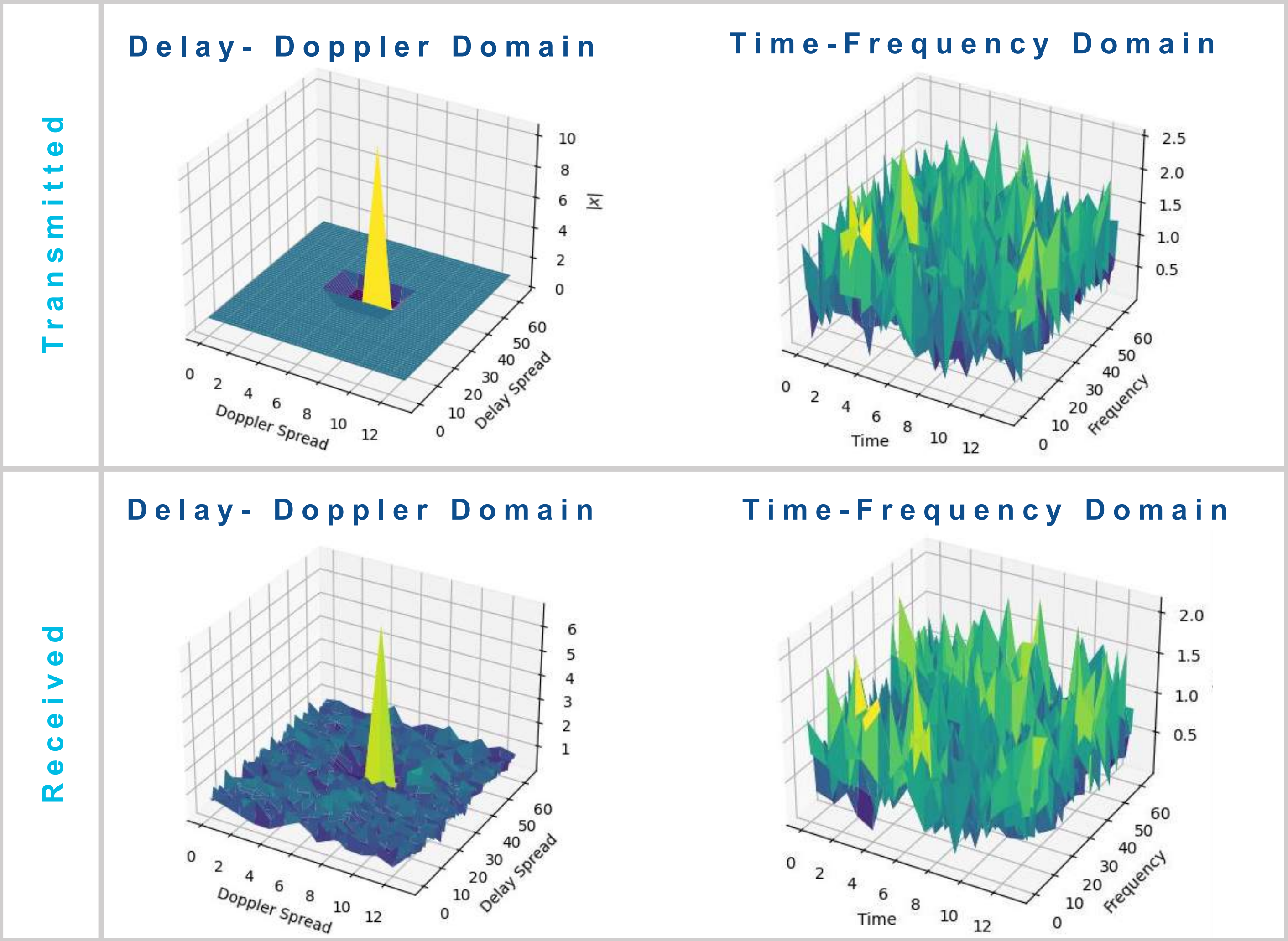


140 GHz Sample Capture ~2dB SNR

Parameters	Values
Modulation	QPSK
# of subcarriers	64
Subcarrier spacing	~15 MHz
Time slots	14
Bandwidth	~1 GHz
Distance	1 m

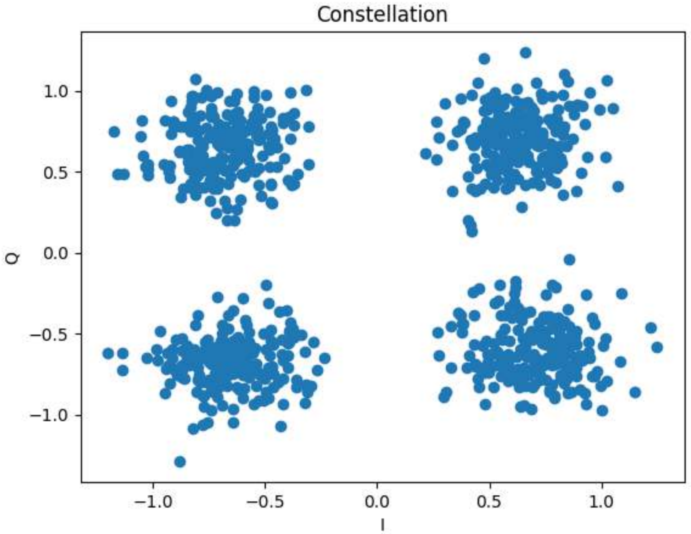


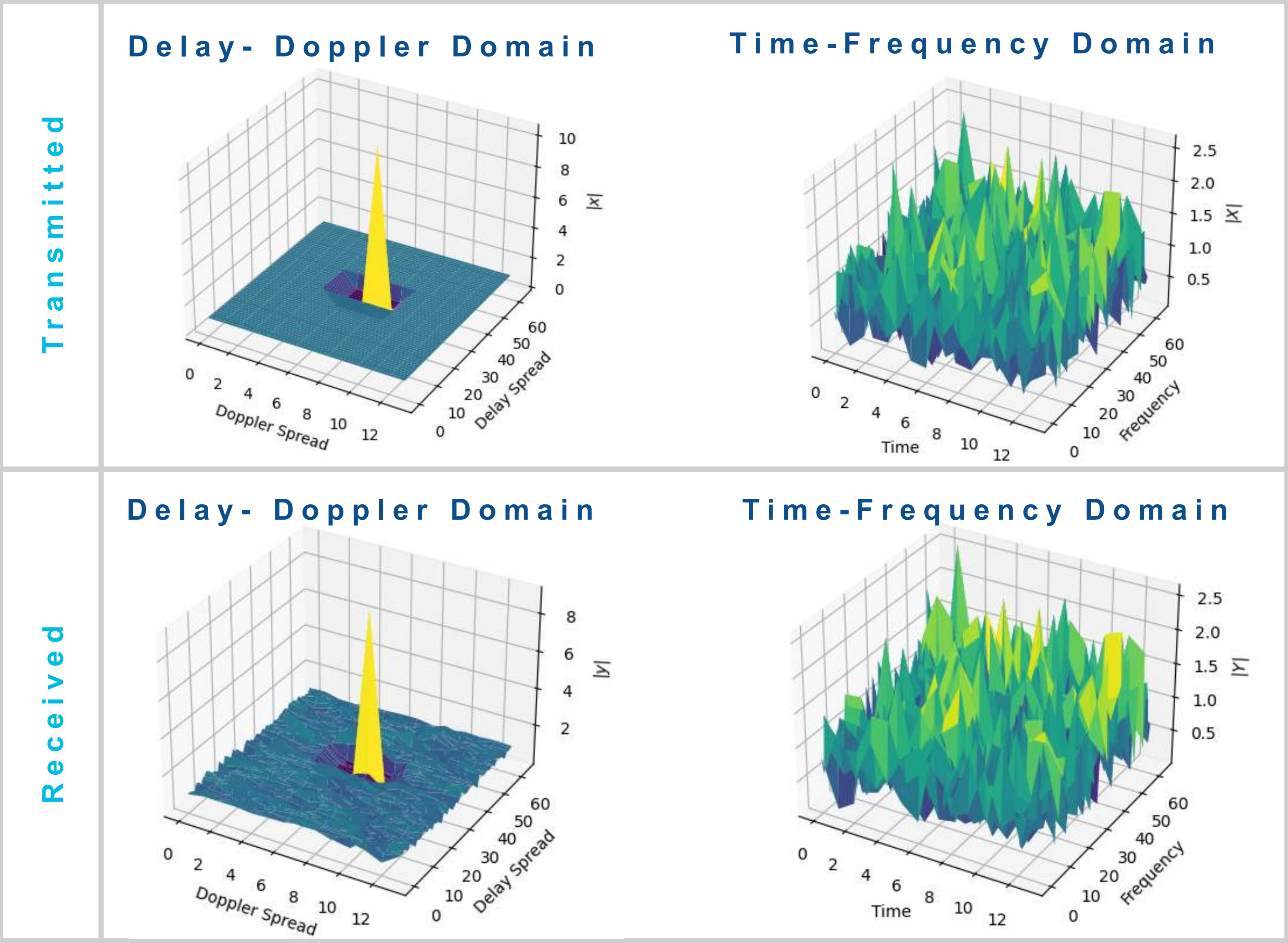
BER ~ 0.05



340 GHz Sample Capture ~7dB SNR

Parameters	Values
Modulation	QPSK
# of subcarriers	64
Subcarrier spacing	~15 MHz
Time slots	14
Bandwidth	~1 GHz
Distance	.5 m

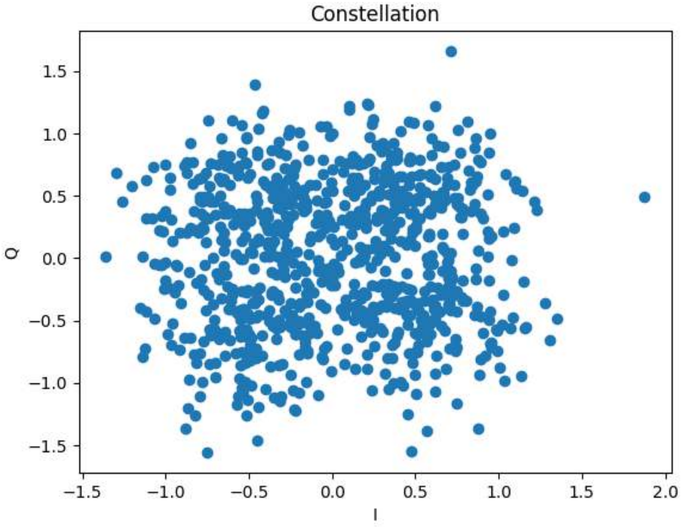




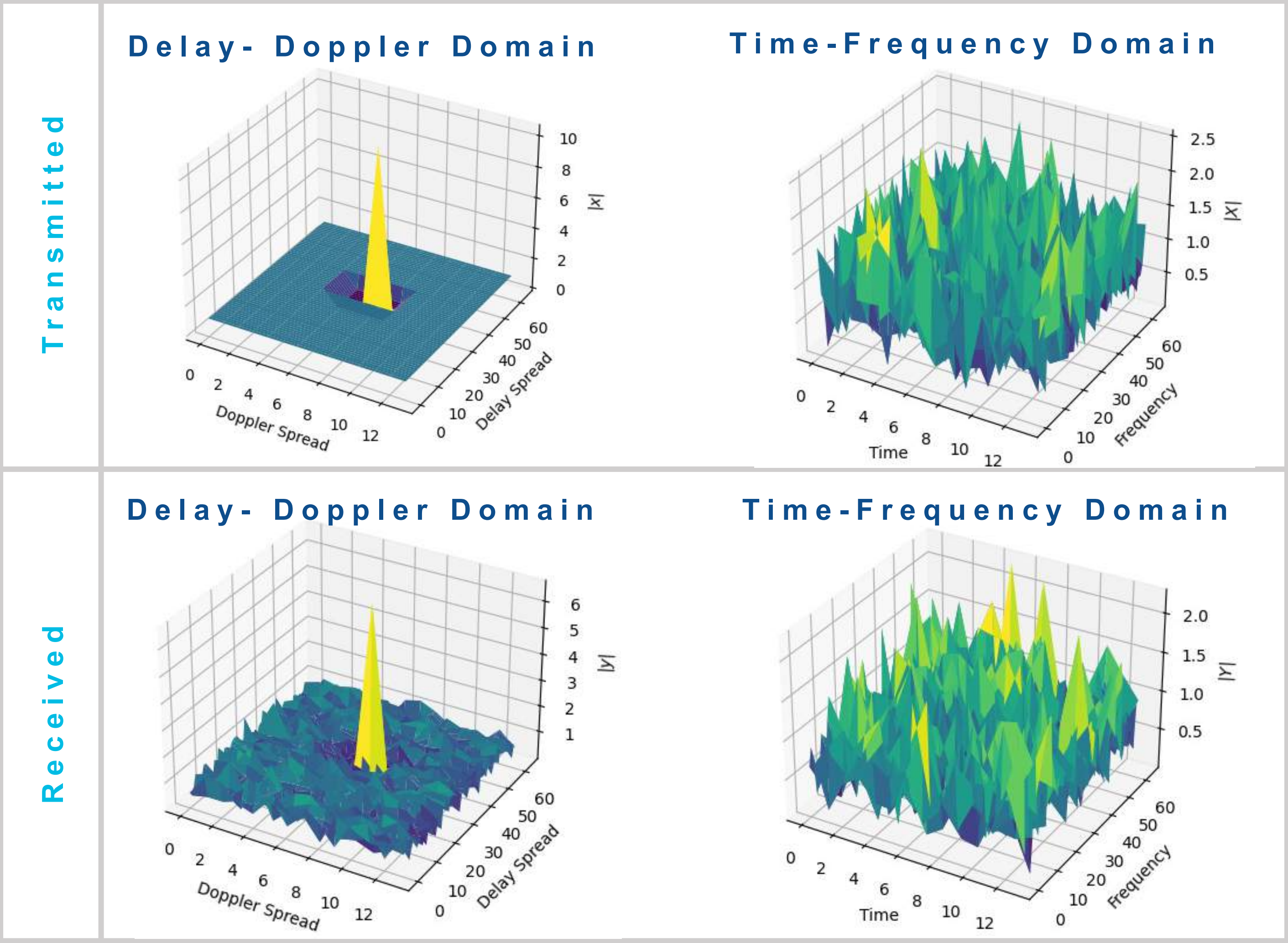
Error Free Transmission!

340 GHz Sample Capture ~2dB SNR

Parameters	Values
Modulation	QPSK
# of subcarriers	64
Subcarrier spacing	~15 MHz
Time slots	14
Bandwidth	~1 GHz
Distance	.5 m



BER ~ 0.08



Summary

Summary

- OTFS shows promise for terahertz band communications
- Tailored existing OTFS simulation code for successful operation with AFRL terahertz testbed system
- Demonstrated successful transmission and reception of OTFS at 140 GHz and 340 GHz
    - Show successful recovery low and high SNR scenarios

Opportunities for Extension

Opportunities for Extension

- Improved workflow and interoperability between Python processing and MATLAB instrument control
- Further design of OTFS
  - Better parameter choices for terahertz system?
- Explore different implementations
  - PAPR-enhanced OTFS