

~\Documents\RF Datasets\F25 Radar Spoofing\radar spoofing code.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 class RadarSimulator:
5     def __init__(self, target_range=90, target_velocity=0,
6                 enable_spoofing=False, spoof_range=None, spoof_velocity=None):
7         """
8         Initialize FMCW Radar Simulator
9
10        Parameters:
11        - target_range: Distance to target in meters
12        - target_velocity: Target velocity in m/s (positive = moving away)
13        - enable_spoofing: Enable spoofing attack (default: False)
14        - spoof_range: Spoofed range in meters (None = no range spoofing)
15        - spoof_velocity: Spoofed velocity in m/s (None = no velocity spoofing)
16        """
17        # Radar parameters
18        self.c = 3e8 # Speed of light (m/s)
19        self.fc = 77e9 # Carrier frequency (77 GHz - automotive radar)
20        self.B = 150e6 # Bandwidth (150 MHz)
21        self.T_chirp = 50e-6 # Chirp duration (50 microseconds)
22        self.fs = 10e6 # Sampling frequency (10 MHz)
23
24        # Target parameters
25        self.target_range = target_range
26        self.target_velocity = target_velocity
27
28        # Spoofing parameters - ADDED
29        self.enable_spoofing = enable_spoofing
30        self.spoof_range = spoof_range if spoof_range is not None else target_range
31        self.spoof_velocity = spoof_velocity if spoof_velocity is not None else
target_velocity
32
33        # Derived parameters
34        self.slope = self.B / self.T_chirp # Chirp slope (Hz/s)
35        self.max_range = (self.c * self.T_chirp) / 2
36        self.range_resolution = self.c / (2 * self.B)
37
38        # Calculate maximum unambiguous velocity
39        self.lambda_wavelength = self.c / self.fc
40        self.v_max = self.lambda_wavelength / (4 * self.T_chirp)
41
42        # Time vector
43        self.t = np.arange(0, self.T_chirp, 1/self.fs)
44
45    def generate_chirp(self):
46        """Generate transmitted FMCW chirp signal"""
47        # Linear frequency modulation

```

```

48     freq_inst = self.fc + self.slope * self.t
49     phase = 2 * np.pi * (self.fc * self.t + 0.5 * self.slope * self.t**2)
50     chirp = np.cos(phase)
51     return chirp, freq_inst
52
53     def generate_echo(self):
54         """Generate received echo from target"""
55         # Calculate time delay due to range
56         tau = 2 * self.target_range / self.c
57
58         # Calculate Doppler frequency shift
59         f_doppler = 2 * self.target_velocity * self.fc / self.c
60
61         # Delayed time vector
62         t_delayed = self.t - tau
63         t_delayed = np.maximum(t_delayed, 0) # Causal signal
64
65         # Generate echo with delay and Doppler
66         phase_echo = 2 * np.pi * ((self.fc + f_doppler) * t_delayed +
67                                     0.5 * self.slope * t_delayed**2)
68
69         # Add propagation loss
70         wavelength = self.c / self.fc
71         path_loss = (4 * np.pi * self.target_range / wavelength)**2
72         amplitude = 0.1 / np.sqrt(path_loss) # Scaling factor for visibility
73
74         echo = amplitude * np.cos(phase_echo)
75         freq_inst_echo = self.fc + f_doppler + self.slope * t_delayed
76
77         return echo, freq_inst_echo
78
79     def generate_range_doppler_map(self, num_chirps=128):
80         """
81         Generate Range-Doppler map using multiple chirps
82         Includes spoofing signals if enabled
83
84         Parameters:
85         - num_chirps: Number of chirps to simulate (for Doppler processing)
86         """
87         # Initialize data matrix
88         N_samples = len(self.t)
89         data_matrix = np.zeros((N_samples, num_chirps), dtype=complex)
90
91         # True target parameters
92         tau_true = 2 * self.target_range / self.c
93         f_beat_true = self.slope * tau_true # Beat frequency due to range
94         f_doppler_true = 2 * self.target_velocity * self.fc / self.c
95
96         # Spoofed target parameters (when enabled)
97         if self.enable_spoofing:

```

```

98     tau_spoof = 2 * self.spoof_range / self.c
99     f_beat_spoof = self.slope * tau_spoof
100    f_doppler_spoof = 2 * self.spoof_velocity * self.fc / self.c
101
102    for chirp_idx in range(num_chirps):
103        # Time offset for this chirp (simulates motion)
104        t_chirp_start = chirp_idx * self.T_chirp
105
106        # True target signal
107        doppler_phase_true = 2 * np.pi * f_doppler_true * t_chirp_start
108        signal_true = 1.0 * np.exp(1j * 2 * np.pi * f_beat_true * self.t + 1j *
doppler_phase_true)
109
110        # Spoofed signal (when enabled)
111        if self.enable_spoofing:
112            doppler_phase_spoof = 2 * np.pi * f_doppler_spoof * t_chirp_start
113            signal_spoof = 1.0 * np.exp(1j * 2 * np.pi * f_beat_spoof * self.t + 1j *
doppler_phase_spoof)
114            total_signal = signal_true + signal_spoof
115        else:
116            total_signal = signal_true
117
118        # Adding noise
119        noise = 0.01 * (np.random.randn(N_samples) + 1j * np.random.randn(N_samples))
120        data_matrix[:, chirp_idx] = total_signal + noise
121
122    # Range FFT (across fast-time samples within each chirp)
123    range_fft = np.fft.fft(data_matrix, axis=0, n=2048) # Zero-pad for better resolution
124
125    # Doppler FFT (across slow-time chirps)
126    range_doppler = np.fft.fft(range_fft, axis=1)
127    range_doppler = np.fft.fftshift(range_doppler, axes=1)
128
129    # Magnitude in dB
130    rd_map = 20 * np.log10(np.abs(range_doppler) + 1e-10)
131
132    # Create range axis
133    freq_range = np.fft.fftfreq(range_fft.shape[0], 1/self.fs)
134    range_axis = (freq_range * self.c / (2 * self.slope))
135
136    # Create velocity axis
137    doppler_freq = np.fft.fftshift(np.fft.fftfreq(num_chirps, self.T_chirp))
138    velocity_axis = doppler_freq * self.c / (2 * self.fc)
139
140    return rd_map, range_axis, velocity_axis
141
142    def generate_beat_signal_with_noise(self):
143        """
144        Generate beat signal from target echo with added noise
145        """

```

```
146     demo_range = 10
147     tau = 2 * demo_range / self.c
148     f_beat = self.slope * tau
149
150     # Generate beat signal (oscillating at beat frequency)
151     beat_signal = np.cos(2 * np.pi * f_beat * self.t)
152
153     # Add noise
154     noise = 0.4 * np.random.randn(len(self.t))
155     noisy_signal = beat_signal + noise
156
157     return noisy_signal, beat_signal, noise, demo_range
158
159 def smooth_with_convolution(self, noisy_signal, window_size=25):
160     """
161     Applies smoothing convolution to reduce noise
162     """
163     # Create smoothing kernel (moving average)
164     kernel = np.ones(window_size) / window_size
165
166     # Convolution
167     smoothed_signal = np.convolve(noisy_signal, kernel, mode='same')
168
169     return smoothed_signal, kernel
170
171 def plot_convolution_demo(self):
172     """
173     Simple 3-plot demonstration: Noisy → True Signal → Smoothed (via convolution)
174     """
175     # Generate signals
176     noisy_signal, true_signal, noise, demo_range = self.generate_beat_signal_with_noise()
177     smoothed_signal, kernel = self.smooth_with_convolution(noisy_signal, window_size=25)
178
179     # Create figure with 3 subplots
180     fig, axes = plt.subplots(3, 1, figsize=(14, 10))
181
182     # Calculate SNR
183     signal_power = np.mean(true_signal**2)
184     noise_power = np.mean(noise**2)
185     snr_before = 10 * np.log10(signal_power / noise_power)
186
187     residual_noise = smoothed_signal - true_signal
188     residual_power = np.mean(residual_noise**2)
189     snr_after = 10 * np.log10(signal_power / residual_power)
190
191     # Plot 1: Noisy Signal (Raw received signal)
192     axes[0].plot(self.t * 1e6, noisy_signal, 'r', linewidth=1, alpha=0.8)
193     axes[0].set_xlabel('Time (μs)', fontsize=11, fontweight='bold')
194     axes[0].set_ylabel('Amplitude', fontsize=11, fontweight='bold')
195     axes[0].set_title('Noisy Received Signal', fontsize=13, fontweight='bold')
```

```

196     axes[0].grid(True, alpha=0.3)
197     axes[0].set_xlim([0, 30])
198     axes[0].set_ylim([-2, 2])
199
200     info_text = f"SNR: {snr_before:.1f} dB"
201     axes[0].text(0.02, 0.98, info_text, transform=axes[0].transAxes,
202                 fontsize=11, verticalalignment='top',
203                 bbox=dict(boxstyle='round', facecolor='lightcoral', alpha=0.9))
204
205     # Plot 2: True Signal (What we want to recover)
206     axes[1].plot(self.t * 1e6, true_signal, 'b', linewidth=2)
207     axes[1].set_xlabel('Time (μs)', fontsize=11, fontweight='bold')
208     axes[1].set_ylabel('Amplitude', fontsize=11, fontweight='bold')
209     axes[1].set_title('True Signal', fontsize=13, fontweight='bold')
210     axes[1].grid(True, alpha=0.3)
211     axes[1].set_xlim([0, 30])
212     axes[1].set_ylim([-2, 2])
213
214     beat_freq = self.slope * (2 * demo_range / self.c)
215     info_text = f"Frequency: {beat_freq/1e3:.1f} kHz\nDemo target at {demo_range}m"
216     axes[1].text(0.02, 0.98, info_text, transform=axes[1].transAxes,
217                 fontsize=11, verticalalignment='top',
218                 bbox=dict(boxstyle='round', facecolor='lightblue', alpha=0.9))
219
220     # Plot 3: After Convolution (Smoothed signal)
221     axes[2].plot(self.t * 1e6, smoothed_signal, 'g', linewidth=2, label='After
Convolution')
222     axes[2].plot(self.t * 1e6, true_signal, 'b--', linewidth=1.5, alpha=0.6, label='True
Signal')
223     axes[2].set_xlabel('Time (μs)', fontsize=11, fontweight='bold')
224     axes[2].set_ylabel('Amplitude', fontsize=11, fontweight='bold')
225     axes[2].set_title('After Convolution', fontsize=13, fontweight='bold', color='green')
226     axes[2].grid(True, alpha=0.3)
227     axes[2].set_xlim([0, 30])
228     axes[2].set_ylim([-2, 2])
229     axes[2].legend(loc='upper right', fontsize=11)
230
231     improvement = snr_after - snr_before
232     info_text = (f"SNR: {snr_after:.1f} dB\n"
233                 f"Improvement: +{improvement:.1f} dB\n")
234     axes[2].text(0.02, 0.98, info_text, transform=axes[2].transAxes,
235                 fontsize=11, verticalalignment='top',
236                 bbox=dict(boxstyle='round', facecolor='lightgreen', alpha=0.9))
237
238     # Add convolution formula
239     formula_text = "Convolution"
240     axes[2].text(0.98, 0.02, formula_text, transform=axes[2].transAxes,
241                 fontsize=10, verticalalignment='bottom', horizontalalignment='right',
242                 family='monospace',
243                 bbox=dict(boxstyle='round', facecolor='yellow', alpha=0.8))

```

```

244
245     plt.suptitle('Convolution Demonstration',
246                  fontsize=14, fontweight='bold', y=0.995)
247
248     plt.tight_layout(rect=[0, 0, 1, 0.99])
249     plt.show()
250
251     # Print summary
252     print("\n" + "="*60)
253     print("CONVOLUTION DEMONSTRATION SUMMARY")
254     print("="*60)
255     print(f"Demo Target Range: {demo_range} m (for visualization)")
256     print(f"Beat Frequency: {beat_freq/1e3:.1f} kHz")
257     print(f"\nNoise Reduction Results:")
258     print(f"   SNR Before: {snr_before:.1f} dB (noisy)")
259     print(f"   SNR After:   {snr_after:.1f} dB (smoothed)")
260     print(f"   Improvement: +{improvement:.1f} dB")
261     print(f"\nConvolution successfully recovered the signal pattern!")
262     print("="*60)
263
264
265
266
267     def plot_chirp_time_domain(self):
268         """Plot transmitted and received chirps in time domain"""
269         tx_chirp, _ = self.generate_chirp()
270         rx_echo, _ = self.generate_echo()
271
272         plt.figure(figsize=(14, 5))
273
274         # Transmitted chirp
275         plt.subplot(1, 2, 1)
276         plt.plot(self.t * 1e6, tx_chirp, 'b', linewidth=0.8)
277         plt.xlabel('Time (μs)', fontsize=12)
278         plt.ylabel('Amplitude', fontsize=12)
279         plt.title('Transmitted Chirp - Time Domain', fontsize=13, fontweight='bold')
280         plt.grid(True, alpha=0.3)
281         plt.xlim([0, 5])
282
283         info_text = ("This shows the transmitted radar signal.\n"
284                     "It's a high-frequency cosine wave whose\n"
285                     "frequency increases linearly over time\n"
286                     "(FMCW chirp).")
287         plt.text(0.02, 0.98, info_text, transform=plt.gca().transAxes,
288                 fontsize=9, verticalalignment='top',
289                 bbox=dict(boxstyle='round', facecolor='lightblue', alpha=0.7))
290
291         # Received echo
292         plt.subplot(1, 2, 2)
293         plt.plot(self.t * 1e6, rx_echo, 'r', linewidth=0.8)

```

```

294     plt.xlabel('Time ( $\mu$ s)', fontsize=12)
295     plt.ylabel('Amplitude', fontsize=12)
296     plt.title(f'Received Echo - Time Domain\n(Target: {self.target_range}m,\n{self.target_velocity}m/s)',
297             fontsize=13, fontweight='bold')
298     plt.grid(True, alpha=0.3)
299     plt.xlim([0, 5])
300
301     tau = 2 * self.target_range / self.c
302     info_text = (f"This is the reflected signal from the target.\n"
303                f"It's delayed by {tau*1e9:.2f} ns due to\n"
304                f"the round-trip travel time to {self.target_range}m.\n"
305                f"The frequency is also shifted by Doppler effect.")
306     plt.text(0.02, 0.98, info_text, transform=plt.gca().transAxes,
307            fontsize=9, verticalalignment='top',
308            bbox=dict(boxstyle='round', facecolor='lightcoral', alpha=0.7))
309
310     plt.tight_layout()
311     plt.show()
312
313     def plot_chirp_frequency(self):
314         """Plot instantaneous frequency vs time"""
315         tx_chirp, freq_tx = self.generate_chirp()
316         rx_echo, freq_rx = self.generate_echo()
317
318         plt.figure(figsize=(14, 5))
319
320         # Transmitted chirp frequency
321         plt.subplot(1, 2, 1)
322         plt.plot(self.t * 1e6, (freq_tx - self.fc) / 1e6, 'b', linewidth=2)
323         plt.xlabel('Time ( $\mu$ s)', fontsize=12)
324         plt.ylabel('Frequency Offset from Carrier (MHz)', fontsize=12)
325         plt.title('Transmitted Chirp - Frequency vs Time', fontsize=13, fontweight='bold')
326         plt.grid(True, alpha=0.3)
327
328         info_text = (f"This shows how the chirp frequency sweeps\n"
329                    f"linearly from 0 to {self.B/1e6:.0f} MHz over {self.T_chirp*1e6:.0f}
330                    f" $\mu$ s.\n"
331                    f"Carrier freq: {self.fc/1e9:.1f} GHz\n"
332                    f"This linear sweep is what makes a radar FMCW.")
333         plt.text(0.02, 0.98, info_text, transform=plt.gca().transAxes,
334                fontsize=9, verticalalignment='top',
335                bbox=dict(boxstyle='round', facecolor='lightblue', alpha=0.7))
336
337         # Received echo frequency
338         plt.subplot(1, 2, 2)
339         plt.plot(self.t * 1e6, (freq_rx - self.fc) / 1e6, 'r', linewidth=2)
340         plt.xlabel('Time ( $\mu$ s)', fontsize=12)
341         plt.ylabel('Frequency Offset from Carrier (MHz)', fontsize=12)
342         plt.title('Received Echo - Frequency vs Time', fontsize=13, fontweight='bold')

```

```

342     plt.grid(True, alpha=0.3)
343
344     f_doppler = 2 * self.target_velocity * self.fc / self.c
345     info_text = (f"The echo has the same linear sweep but:\n"
346                 f"1. Delayed in time\n"
347                 f"2. Shifted by Doppler: {f_doppler/1e3:.2f} kHz\n"
348                 f"This frequency difference between TX and RX\n"
349                 f"reveals the target's range and velocity.")
350     plt.text(0.02, 0.98, info_text, transform=plt.gca().transAxes,
351             fontsize=9, verticalalignment='top',
352             bbox=dict(boxstyle='round', facecolor='lightcoral', alpha=0.7))
353
354     plt.tight_layout()
355     plt.show()
356
357 def plot_range_doppler_map(self):
358     """Plot the Range-Doppler map (what the radar 'sees')"""
359     rd_map, range_axis, velocity_axis = self.generate_range_doppler_map()
360
361     plt.figure(figsize=(12, 8))
362
363     # Limiting range and velocity values
364     range_mask = (range_axis >= 0) & (range_axis <= 150)
365     velocity_mask = (velocity_axis >= -50) & (velocity_axis <= 50)
366
367     rd_map_plot = rd_map[np.ix_(range_mask, velocity_mask)]
368     range_axis_plot = range_axis[range_mask]
369     velocity_axis_plot = velocity_axis[velocity_mask]
370
371     # Normalize for better visualization
372     rd_map_norm = rd_map_plot - np.min(rd_map_plot)
373     threshold = np.max(rd_map_norm) - 40 # Show 40 dB dynamic range
374
375     im = plt.imshow(rd_map_norm.T, aspect='auto',
376                    extent=[range_axis_plot[0], range_axis_plot[-1],
377                           velocity_axis_plot[0], velocity_axis_plot[-1]],
378                    origin='lower', cmap='hot', vmin=threshold,
379                    interpolation='bilinear')
380
381     plt.xlabel('Range (m)', fontsize=13, fontweight='bold')
382     plt.ylabel('Velocity (m/s)', fontsize=13, fontweight='bold')
383
384     # Title changes based on spoofing
385     if self.enable_spoofing:
386         plt.title('Range-Doppler Map',
387                  fontsize=14, fontweight='bold', color='red')
388     else:
389         plt.title('Range-Doppler Map',
390                  fontsize=14, fontweight='bold')
391

```

```

392     plt.grid(True, alpha=0.3, color='cyan', linewidth=0.5, linestyle='--')
393
394     # Mark true target position
395     '''
396     plt.plot(self.target_range, self.target_velocity, 'go',
397              markersize=15, markeredgewidth=3, markeredgewidth=3, markeredgewidth=3, markeredgewidth=3,
398              label=f'True Target: {self.target_range}m, {self.target_velocity}m/s',
399              zorder=10)
400
401     # Mark spoofed target if enabled
402     if self.enable_spoofing:
403         plt.plot(self.spoof_range, self.spoof_velocity, 'r^',
404                  markersize=18, markeredgewidth=3, markeredgewidth=3, markeredgewidth=3, markeredgewidth=3,
405                  label=f'SPOOFED: {self.spoof_range}m, {self.spoof_velocity}m/s',
406                  zorder=11)
407     '''
408
409     # Check for velocity aliasing
410     if abs(self.target_velocity) > self.v_max:
411         v_aliased = self.target_velocity % (2 * self.v_max)
412         if v_aliased > self.v_max:
413             v_aliased = v_aliased - 2 * self.v_max
414         plt.plot(self.target_range, v_aliased, 'yo',
415                  markersize=12, alpha=0.6,
416                  label=f'Aliased: {v_aliased:.1f}m/s', zorder=9)
417
418     plt.legend(loc='upper right', fontsize=11)
419
420     plt.colorbar(im, label='Signal Strength (dB)', pad=0.02)
421
422     # Info text with spoofing details
423     aliasing_warning = ""
424     if abs(self.target_velocity) > self.v_max:
425         aliasing_warning = f"\n Velocity aliasing on true target"
426
427     spoof_info = ""
428     if self.enable_spoofing:
429         spoof_type = []
430         if self.spoof_range != self.target_range:
431             spoof_type.append("RANGE")
432         if self.spoof_velocity != self.target_velocity:
433             spoof_type.append("VELOCITY")
434         spoof_info = f"\n\n SPOOFING: {' + '.join(spoof_type)}\n Spoofers injects false
435 signal"
436
437     box_color = 'red' if self.enable_spoofing else 'black'
438     text_color = 'yellow' if self.enable_spoofing else 'white'
439
440     info_text = (f"Bright spots = detected objects\n"
441                  f"Range resolution: {self.range_resolution:.2f} m\n"
442                  f"Max unambiguous velocity: ±{self.v_max:.2f} m/s"
443                  f"{aliasing_warning}{spoof_info}")

```

```

440     '''
441     plt.text(0.02, 0.98, info_text, transform=plt.gca().transAxes,
442             fontsize=9, verticalalignment='top',
443             bbox=dict(boxstyle='round', facecolor=box_color, alpha=0.8),
444             color=text_color)
445     '''
446     plt.tight_layout()
447     plt.show()
448
449     def plot_all(self):
450         """Generate all plots"""
451         print("\n" + "="*60)
452         print("FMCW RADAR SIMULATION")
453         if self.enable_spoofing:
454             print("SPOOFING ATTACK ACTIVE")
455         print("="*60)
456         print(f"\nRadar Configuration:")
457         print(f"  Carrier Frequency: {self.fc/1e9:.1f} GHz")
458         print(f"  Bandwidth: {self.B/1e6:.0f} MHz")
459         print(f"  Chirp Duration: {self.T_chirp*1e6:.0f} μs")
460         print(f"  Range Resolution: {self.range_resolution:.2f} m")
461         print(f"  Max Unambiguous Velocity: ±{self.v_max:.2f} m/s")
462         print(f"\nTrue Target:")
463         print(f"  Range: {self.target_range} m")
464         print(f"  Velocity: {self.target_velocity} m/s")
465         print(f"  Time Delay: {2*self.target_range/self.c*1e9:.2f} ns")
466         print(f"  Doppler Shift: {2*self.target_velocity*self.fc/self.c/1e3:.2f} kHz")
467
468         # Spoofing info
469         if self.enable_spoofing:
470             print(f"\nSpoofed Target:")
471             print(f"  Range: {self.spoof_range} m", end="")
472             if self.spoof_range != self.target_range:
473                 print(f" (RANGE SPOOFING: Δ = {self.spoof_range - self.target_range:+.1f}m)")
474             else:
475                 print(" (no range spoofing)")
476             print(f"  Velocity: {self.spoof_velocity} m/s", end="")
477             if self.spoof_velocity != self.target_velocity:
478                 print(f" (VELOCITY SPOOFING: Δ = {self.spoof_velocity -
self.target_velocity:+.1f}m/s)")
479             else:
480                 print(" (no velocity spoofing)")
481
482         # Velocity aliasing check
483         if abs(self.target_velocity) > self.v_max:
484             v_aliased = self.target_velocity % (2 * self.v_max)
485             if v_aliased > self.v_max:
486                 v_aliased = v_aliased - 2 * self.v_max
487             print(f"\n WARNING: VELOCITY ALIASING on true target!")
488             print(f"  Target velocity ({self.target_velocity} m/s) exceeds v_max")

```

```
489         print(f" Will appear at: {v_aliased:.2f} m/s")
490
491     print("="*60 + "\n")
492
493     #self.plot_chirp_time_domain()
494     #self.plot_chirp_frequency()
495     self.plot_range_doppler_map()
496     #self.plot_convolution_demo()
497
498
499 # Run simulation
500 if __name__ == "__main__":
501     # True target parameters
502     TARGET_RANGE = 100      # meters
503     TARGET_VELOCITY = 10    # m/s (positive is moving away, negative is moving closer)
504
505     # Spoofing controls - ADDED
506     ENABLE_SPOOFING = True  # Set to False for no spoofing
507     SPOOF_RANGE = 80        # meters (None = same as true target)
508     SPOOF_VELOCITY = 7.5    #m/s (None = same as true target)
509     # =====
510
511     # Creates radar simulation
512     radar = RadarSimulator(
513         target_range=TARGET_RANGE,
514         target_velocity=TARGET_VELOCITY,
515         enable_spoofing=ENABLE_SPOOFING,
516         spoof_range=SPOOF_RANGE,
517         spoof_velocity=SPOOF_VELOCITY
518     )
519
520     # Generate all plots
521     radar.plot_all()
522
523     print("\nSimulation complete!")
```