itle[CFAR-STFT]Analiza semnalelor radar în prezența ecourilor marine (sea clutter)

Abordare CFAR-STFT, experimente pe date sintetice și IPIX

Ingrid Corobana Teodora Nae

Universitatea din București
Facultatea de Matematică și Informatică

2 februarie 2026

Agenda

Problema Detecției Radar

- **Provocare**: Detecția fiabilă de semnale radar în sea-clutter
- **Dificulți tradiționale**:
  1. Metode CFAR clasice - doar domeniu frecvență
  2. Pierdere informații temporale
  3. Rate înalte de fals alarm în clutter
- **Soluție**: Exploatare structură timp-frecvență cu CFAR-STFT
- **Referință**: Abratkiewicz (2022) - Sensors

Obiectivele Proiectului

1. Implementare completă algoritm CFAR-STFT în Python
2. Validare pe date sintetice (chirp nonliniar)
3. Testare pe date reale (radar IPIX X-band)
4. Analiză Doppler pentru viteza obiectelor
5. Documentație detaliată și reproducibilitate

Rezultate așteptate:

- Detecție 100% pe semnale sintetice
- RQF (Reconstruction Quality Factor) ¿ 25 dB
- Timp execuție ¡ 100 ms/segment

Pipeline Algoritm - Pasul 1: STFT

- **Transformata Fourier cu Timp Scurt**:

$$X(k,n) = \sum_{m=0}^{N-1} x(m) \cdot w(m - nH) \cdot e^{-j2\pi km/N}$$

- **Parametri STFT**:
  - FFT length: $N = 512$
  - Hop: $H = 256$ (50% suprapunere)
  - Fereastră: Gaussiană cu $\sigma = 8$ bin-uri
- **Avantaj**: Reprezentare timp-frecvență localizată

Pipeline Algoritm - Pasul 2: CFAR 2D

- **Constant False-Alarm Rate** - adaptat la 2D:

$$H(k,n) = \begin{cases} 1 & \text{dacă } |X(k,n)|^2 > P_f \cdot \mathcal{N}(k,n) \\ 0 & \text{altfel} \end{cases}$$

- **GOCA-CFAR (Guard-Cell Order-Statistic)**:

$$\mathcal{N}(k,n) = \frac{1}{4N_G N_T} \sum_{TC} |X(i,j)|^2$$

- **Parametri**:
  - $P_f = 0.4$ (probabilitate falsă alarmă)
  - $N_G = N_T = 16$ (guard + training cells)

Pipeline Algoritm - Pasul 3: DBSCAN

- **Density-Based Spatial Clustering**:
  - Grupare puncte detectate în planul timp-frecvență
  - Rază clustering: $\varepsilon = 4$ Hz/s
  - Minim puncte: minSamples $= 5$
- **Avantaj**: Elimină zgomot și izolate (false alarme)
- **Rezultat**: Clustere coerente = semnale reale

Pipeline Algoritm - Pasul 4-5: Dilatare + iSTFT

Pasul 4: Dilatare Geodezică

- Expandare mască cu 3-5 iterații
- Recuperare energie pierdută în mascare
- Kernel: cruce 3×3 binar

Pasul 5: iSTFT cu Mască

- Reconstrucție inversă: $\hat{x}(n) = \text{iFFT}(X_{masked})$
- Overlap-add cu fereastră identică
- Normalizare finală

Metrica Evaluare: Reconstruction Quality Factor (RQF)

$$\text{RQF} = 10 \log_{10} \left(\frac{\sum_m |x(m)|^2}{\sum_m |x(m) - \hat{x}(m)|^2} \right) \text{ [dB]}$$

Rezultate - Semnale Sintetice

SNR [dB]	RQF_mean [dB]	RQF_std	P_detect [%]
5	7.28	0.47	100
10	16.81	0.60	100
15	22.95	0.56	100
20	26.40	0.51	100
25	28.43	0.39	100
30	29.17	0.25	100

Observații:

- 100 rulări Monte Carlo per SNR level
- Detecție perfectă (100%) la toate nivelurile
- RQF crește monoton cu SNR
- Variabilitate scade la SNR înalt

Rezultate - Date Reale IPIX

Test pe 50 segmente X-band Radar (9.39 GHz)

- Sursă: IPIX database (Canadian institute research)
- Tip semnal: Complex I/Q sea-clutter
- Rezultate:
  - **Detecții**: 3.2 obiecte/segment (mediu)
  - **Viteza Doppler**: 2.4 m/s $\pm$ 0.8 m/s
  - **False alarme**: ¡ 1%
  - **Timp execuție**: 75 ms/segment

Concluzie: Algoritmul generalizează bine de la date sintetice la reale.

Performanță Computațională

- **STFT** (FFT optimizat): 10 ms
- **CFAR 2D** (cu convoluție): 50 ms
- **DBSCAN**: 5 ms
- **iSTFT**: 8 ms
- **Dilatare**: 2 ms
- **Total**: **75 ms/segment** $\Rightarrow$ **13 FPS**

Interpretare: Suficient pentru aplicații offline și semi-real-time.

Structura Cod

src/cfar_stft_detector.py (1110 linii):

- Clasa `CFAR2D`: Detecție adaptivă 2D
- Clasa `DBSCAN`: Clustering custom
- Clasa `CFARSTFTDetector`: Pipeline complet
- Clasa `AcousticCFARDetector`: Variantă acustică

simulations/paper_replication.py (1087 linii):

- `run_paper_experiment()`: Reproducere articol
- `run_ipix_experiment()`: Date reale
- `compute_rqf()`: Metrica evaluare

Dependințe:

- NumPy, SciPy, matplotlib
- Fără framework-uri heavy (TensorFlow, PyTorch)

Exemplu Cod - STFT

```
def compute_stft(x, N_fft=512, hop=256, sigma=8):
    N_t = (len(x) - N_fft) // hop + 1
    X_stft = np.zeros((N_fft//2 + 1, N_t), dtype=complex)

    # Fereastr  Gaussian
    window = np.exp(-np.arange(N_fft)**2 / (2*sigma**2))
    window /= np.sqrt(np.sum(window**2))

    for n in range(N_t):
        x_n = x[n*hop : n*hop + N_fft] * window
        X_n = np.fft.rfft(x_n, N_fft)
        X_stft[:, n] = X_n

    return X_stft
```

Exemplu Cod - CFAR 2D

```
def detect_cfar2d(X_stft, P_f=0.4, N_G=16, N_T=16):
    H = np.zeros(X_stft.shape, dtype=bool)
    N_f, N_t = X_stft.shape

    for k in range(N_G + N_T, N_f - N_G - N_T):
        for n in range(N_G + N_T, N_t - N_G - N_T):
            # Training cells
            X_tc = X_stft[k-N_G-N_T:k-N_G, ...]
            N_local = np.mean(np.abs(X_tc)**2)

            lambda_th = P_f * N_local
            if np.abs(X_stft[k,n])**2 > lambda_th:
                H[k, n] = True

    return H
```

Contribuții Principale

1. **Implementare completă**: CFAR-STFT end-to-end în Python
2. **Validare**:
   - Semnale sintetice: 100% detecție, RQF = 29.17 dB @ SNR=30dB
   - Semnale reale: Funcționare pe date IPIX complexe
3. **Reproducibilitate**: Cod open-source, rezultate verificabile
4. **Performanță**: 75 ms/segment (13 FPS)

Perspective Viitoare

- Accelerare GPU (CUDA/OpenCL)
- Optimizare parametri adaptivi
- Integrare sisteme radar real-time
- Multi-target tracking și predictie traiectorii
- Machine learning pentru calibrare automată

Repository: `https://github.com/dirgnic/Radar_Detection_STFT`
Status: COMPLET - Gata pentru implementare și desfășurare

Mulțumiri

- Recunoștiință: Abratkiewicz (2022) - paper original
- Mulțumiri: Baza IPIX - date experimentale
- Cod disponibil: GitHub public repository

Întrebări?