Blind LoRa Signal Detection Leveraging Chirp Spread Spectrum Characteristics: Algorithm, Analysis, and Performance Evaluation

Matan Leilien

July 22, 2025

Preface

This report is submitted as a supplementary component of my application to the Elisra performance research group. While not a formal requirement of the recruitment process, this work is intended to demonstrate my proactive engagement, entrepreneurial spirit, and an assertive approach to problem-solving. Its primary objective is to showcase my analytical capabilities, creative thinking, problem-solving methodologies, and adherence to high delivery standards.

Abstract

This paper presents an algorithm for the blind detection and parameter estimation of LoRa signals. Unlike conventional methods that rely on known preambles, this approach operates without prior knowledge of key transmission parameters such as spreading factor (SF) or bandwidth (BW). The proposed algorithm exploits the inherent characteristics of LoRa's chirped spread spectrum (CSS) modulation, specifically the behavior of its dechirped and Fast Fourier Transformed (FFT) signal, to enable robust detection. Performance is evaluated through simulation, demonstrating effective detection at low signal-to-noise ratios (SNRs).

1 Introduction

LoRa, a proprietary low-power wide-area network (LPWAN) modulation technology, has gained significant traction in Internet of Things (IoT) applications due to its extended range and energy efficiency. LoRa utilizes chirp spread spectrum (CSS) modulation, which provides robustness against noise and interference. Traditional LoRa receivers typically rely on detecting a known preamble sequence to synchronize and identify signal parameters. However, scenarios exist where such a preamble is unavailable or intentionally omitted, necessitating "blind" detection capabilities.

This work addresses the challenge of blind LoRa signal detection, aiming to identify the presence of a LoRa signal and potentially infer its parameters (SF, BW) without any prior information. The motivation for this research stems from its potential applications in spectrum sensing, unauthorized signal detection, and advanced cognitive radio systems. This paper details a practical algorithm for blind LoRa detection, highlighting its underlying principles and simulated performance.

2 Background on LoRa CSS Modulation

LoRa's CSS modulation encodes information by varying the frequency of a chirp signal linearly over time. A LoRa symbol is a wideband chirp, sweeping across the entire allocated bandwidth within a symbol period. The rate of this frequency sweep determines the spreading factor (SF), with higher SFs corresponding to slower chirp rates, longer symbol durations, and improved robustness against noise.

A fundamental characteristic of CSS is that multiplying (dechirping) a received LoRa symbol by a conjugate chirp (a chirp with the opposite sweep direction, "down chirp") transforms the symbol into a constant frequency tone. The frequency of this tone, after a Fast Fourier Transform (FFT), corresponds to the symbol's payload value. If the received signal is a LoRa chirp with unknown parameters, a correct dechirping operation (using the correct BW and SF) will manifest as a distinct, strong peak in the FFT spectrum.

The mathematical representation of a base LoRa upchirp (ignoring phase offsets) can be given by:

$$s(t) = e^{j2\pi(f_0 t + \frac{K}{2}t^2)}$$

where f_0 is the starting frequency, and $K = \frac{BW}{T_s}$ is the chirp rate, with BW as the bandwidth and T_s as the symbol duration. The symbol duration $T_s = \frac{2^{SF}}{BW}$.

The dechirping operation involves multiplying the received signal $r(t)$ by a downchirp reference:

$$r_{dechirped}(t) = r(t) \cdot e^{-j2\pi\frac{K}{2}t^2}$$

If $r(t)$ is a perfect LoRa upchirp, $r_{dechirped}(t)$ becomes a complex exponential at a constant frequency. The subsequent FFT of this dechirped signal will exhibit a sharp peak at this constant frequency.

3 System Model and Assumptions

The proposed blind detector operates under the following assumptions:

- **Blind Reception:** The receiver cannot rely on the presence of the LoRa preamble for synchronization or parameter estimation.
  Remark: If the preamble were available, the detection algorithm would follow a process similar to that of a conventional LoRa receiver. This would involve segmenting the input signal into $M = 2^{SF}$ non-overlapping segments, de-chirping them and searching for the initial 8 or more downchirp symbols, which would manifest as a constant peak location across several consecutive symbols in the FFT domain.

- **Wideband Operation:** The receiver observes a wide frequency band, significantly larger than the target LoRa signal's bandwidth. The LoRa signal is assumed to reside within a subchannel of this wider band.

- **Channel Impairments:** The wireless channel introduces additive white Gaussian noise (AWGN), integer and fractional symbol timing offsets (STO), and integer carrier frequency offsets (CFO).

- **Ignored Impairments:** Fractional CFO is not explicitly addressed in this work, as it's mitigation can be achieved through standard techniques (e.g., oversampling or fine frequency tracking) once a signal is detected. additionally, sample clock offset (SCO) is not addressed in this scope of work.

In this work, we make the following clarifications regarding system parameters and mission requirements:

1. **Receiver and Environment:** Parameters such as receiver bandwidth, noise figure, and available computing resources are not defined.
2. **Reconnaissance Information:** No prior knowledge is assumed regarding:
   - the exact target LoRa parameters (e.g., bandwidth, spreading factor ranges),
   - LoRaWAN compatibility,
   - the use of frequency hopping,
   - frequency band(s)/carrier search space,
   - minimum number of symbols per transmission packet.
3. **Mission Requirements:** Constraints such as strict detection time or minimum SNR are considered flexible for the purposes of this initial algorithm development and are not strictly enforced.

4 Proposed Blind Detection Algorithm

4.1 general strategy and pre-processing

The proposed algorithm incorporates the following key strategies:

1. **Brute-Force Parameter Search:** The algorithm iteratively attempts to dechirp the received signal using all possible standard LoRa bandwidths (125 kHz, 250 kHz, 500 kHz) and spreading factors (SF 7 to SF 12). This results in 15 distinct dechirping attempts for each segment of the received signal.
2. **Aliasing for CFO/STO Equivalence:** Inspired by "BW-rate" LoRa receivers, the received signal is downsampled without explicit anti-aliasing filtering. If the sampling frequency (F_s) is set equal to the LoRa signal's bandwidth (BW), linear frequency offset (FO) becomes equivalent to cyclic FO due to aliasing. This property facilitates a connection between STO and CFO, simplifying certain aspects of parameter estimation. However, this method introduces an increase in noise power, leading to SNR degradation.
3. **Wideband Decimation and Spectrogram Analysis:** For the purpose of analysis, consider a 5 MHz sub-band that may contain a CSS signal, subject to unknown Symbol Timing Offset (STO) and Carrier Frequency Offset (CFO). While the initial spectrogram of this wideband signal might present a complex representation, a crucial pre-processing step involves decimation.

   Specifically, if the 5 MHz band is decimated by a factor of 40 to yield a 125 kHz bandwidth, the CSS signal remains detectable. It is important to note that this decimation process, particularly without anti-aliasing filtering (as discussed in Section 4.1.2), results in the CSS signal undergoing a cyclic shift in the frequency domain (FD), but its fundamental chirping characteristic is preserved. This property allows for the continuation of the detection algorithm on the downsampled signal.

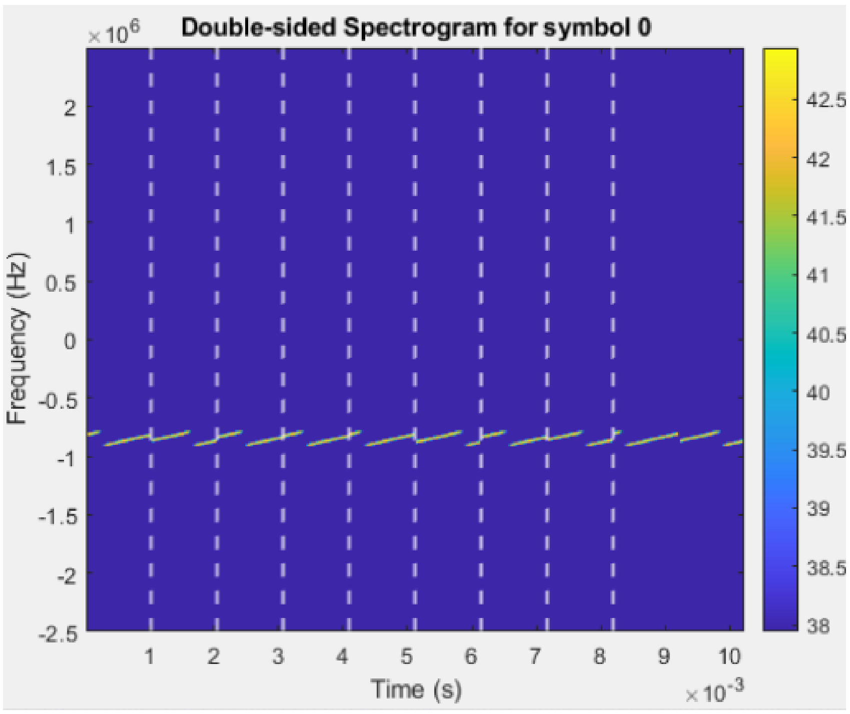


Figure 1: Spectrogram of the wideband (5 MHz) input signal. The CSS signal is present but visually complex due to the wide frequency span.

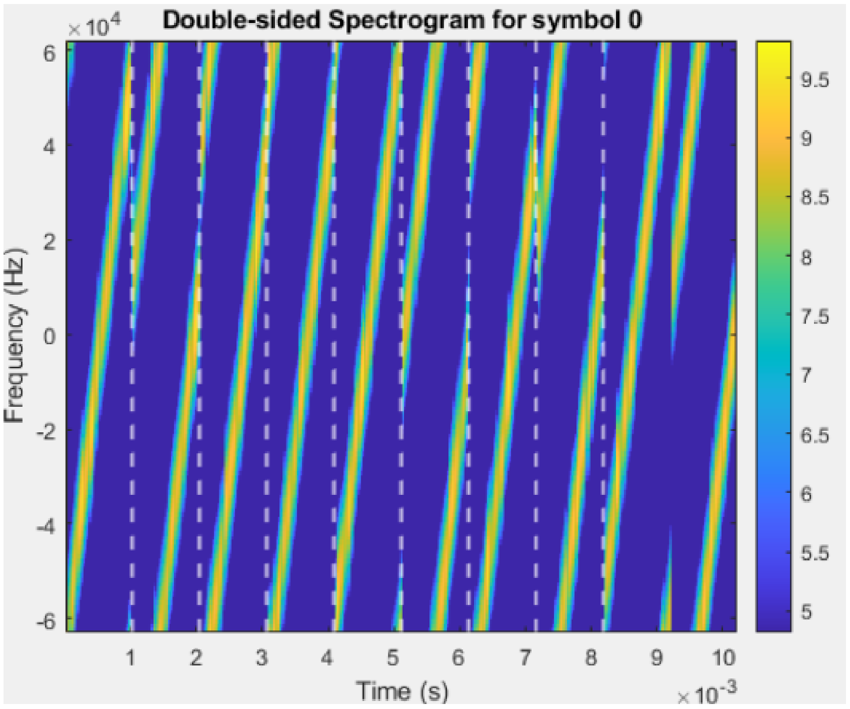


Figure 2: Spectrogram of the decimated (125 kHz) signal. The CSS chirp structure is preserved and more visually apparent after decimation.

4. **Processing Gain Compensation:** To counteract the SNR loss incurred by aliasing, the algorithm processes multiple LoRa symbols together. By coherently or non-coherently combining information across several symbols, a "processing gain" is achieved, aiming to restore overall detection performance comparable to a tradi-

tional LoRa receiver.

5. **Brute-Force Symbol Timing Offset (STO):** The algorithm accounts for unknown STO by evaluating the dechirped FFT output at various timing offsets. A LoRa symbol is sensitive to timing, and a perfectly synchronized symbol will produce a sharp, singular peak. Deviations from perfect timing spread this peak across two dominant bins, demonstrating ISI (as illustrated in Fig. 1-3).

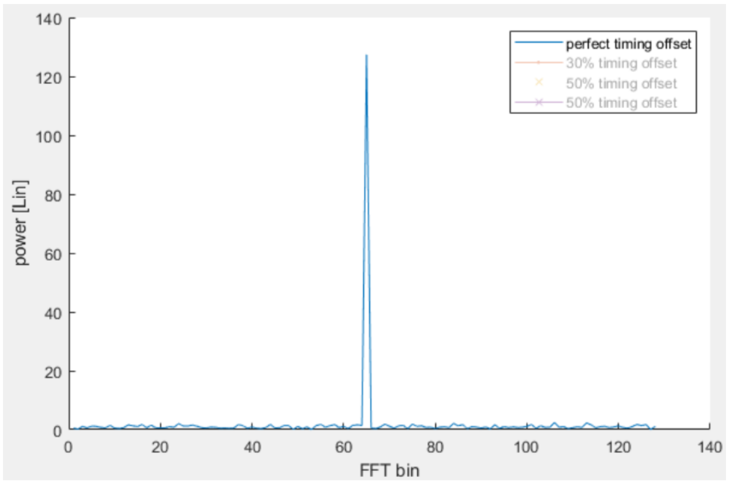


Figure 3: FFT output for a perfectly synchronized LoRa symbol (STO = 0). A single sharp peak is observed.

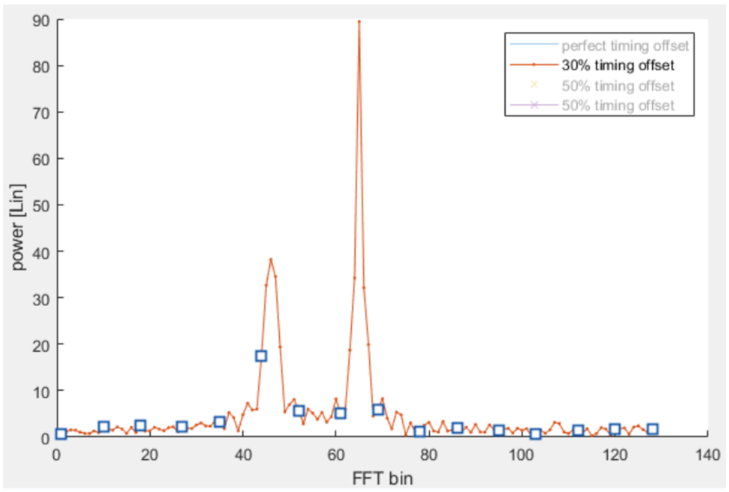


Figure 4: FFT output for a LoRa symbol with a symbol timing offset of 30 samples. The peak is spread across two prominent bins.

4.2 Detection Mechanism: Post-FFT Peak

The primary detection approach involves the following steps:

1. **Segmentation:** The received signal is divided into non-overlapping segments, each of length $M = 2^{SF}$ samples, corresponding to the duration of a single LoRa symbol for a given SF.

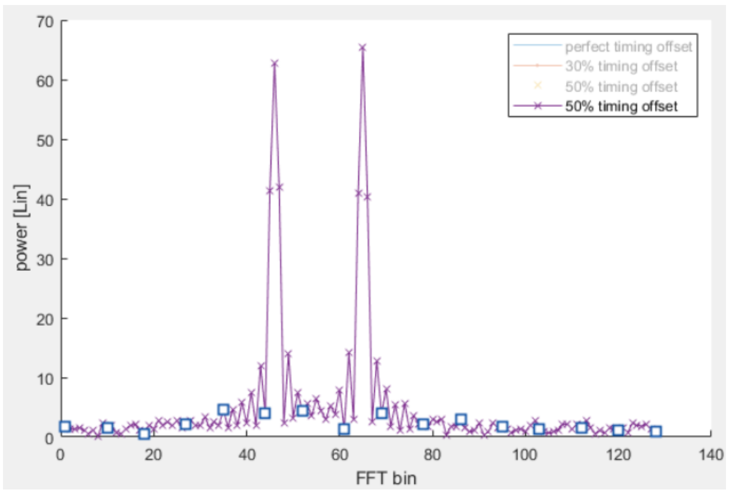


Figure 5: FFT output for a LoRa symbol with a symbol timing offset of 50 samples. The peak is further spread and less distinct.

2. **Dechirping and FFT:** For each segment, the signal is dechirped using a reference downchirp matched to the hypothesized SF and BW. A Fast Fourier Transform (FFT) is then applied to the dechirped segment. If the chosen SF and BW are correct and the timing offset is sufficiently compensated, a strong, distinct peak is expected in the FFT magnitude spectrum.

3. **Peak Centering and Combination:** The individual FFT outputs are analyzed. For each segment, the strongest peak in the FFT magnitude is identified, and the entire FFT spectrum is circularly shifted so that this peak is centered at a predefined reference bin (e.g., $M/2 + 1$). This alignment enables coherent or non-coherent combination of information from multiple symbols, enhancing the detectability of the LoRa signal under low SNR conditions.

   Figure 6 demonstrates this process: multiple centered FFT magnitudes from different segments are overlaid (grey), and their mean is shown in black, highlighting the enhancement of the central peak after alignment and averaging.

The combined FFT magnitude, $FFT_{\text{combined}}[k]$, is calculated as the mean of the magnitudes of the centered FFT outputs from all processed segments:

$$FFT_{\text{combined}}[k] = \frac{1}{N_{\text{symbols}}} \sum_{i=1}^{N_{\text{symbols}}} |FFT^{(i)}_{\text{centered}}[k]| \tag{1}$$

where N_{symbols} is the number of symbols processed.

Post-Processing SNR (PPSNR) Metric: A detection metric, termed Post-Processing SNR (PPSNR), is calculated to quantify the prominence of the detected peak against the noise floor. This metric is defined as the ratio of the peak value in the combined FFT to the mean noise level:

$$PPSNR_{dB} = 10 \log_{10} \left(\frac{\text{Peak Value}}{\text{Mean Noise Value}} \right) \tag{2}$$

The peak value is taken from the centered bin in the combined FFT ($FFT_{\text{combined}}[M/2 + 1]$). The mean noise value is estimated from the average of the combined FFT values outside a small exclusion window around the central peak (e.g., bins $M/2-2$ to $M/2+4$).

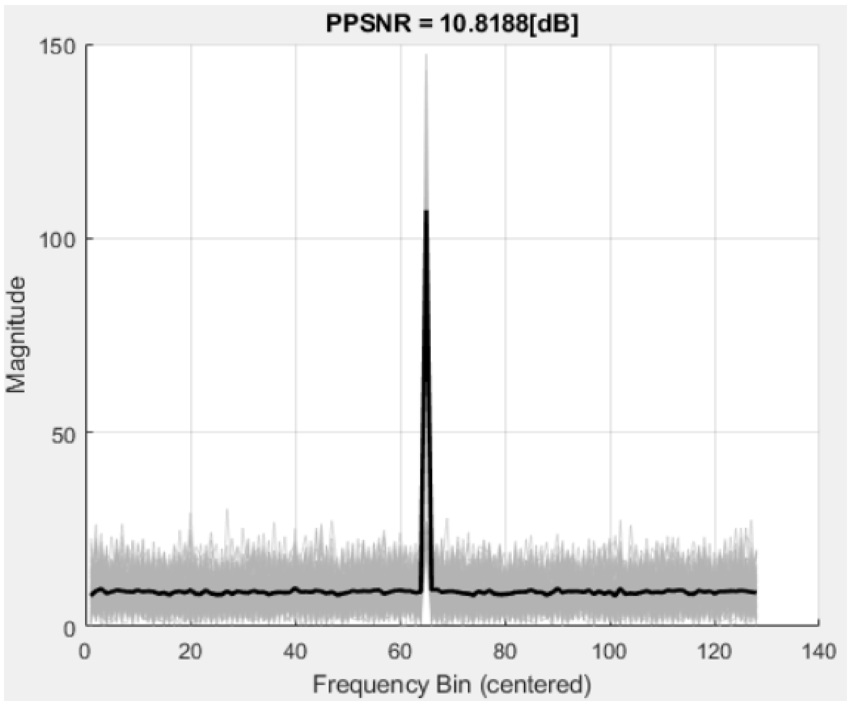


Figure 6: Overlay of centered FFT magnitudes from multiple segments (grey) and their mean (black). Centering and averaging enhances the detectability of the LoRa peak.

Furthermore, it is important to note that a coherent combination of post-FFT vectors is deemed invalid within this specific operational context. This limitation is primarily attributed to the absence of a preamble, which would typically facilitate the identification of precise symbol boundaries.

In an idealized scenario involving a perfectly time-aligned signal, and assuming the successful correction of any fractional frequency offset, the initial phase of each symbol would be precisely identical (under the assumption of a coherent transmitter architecture). As illustrated in Figure 7 (referencing the figure depicting the phase of LoRa symbols vs time), there exists only a single temporal instance where the signal phases undergo a "reset," thereby becoming suitable for coherent addition.

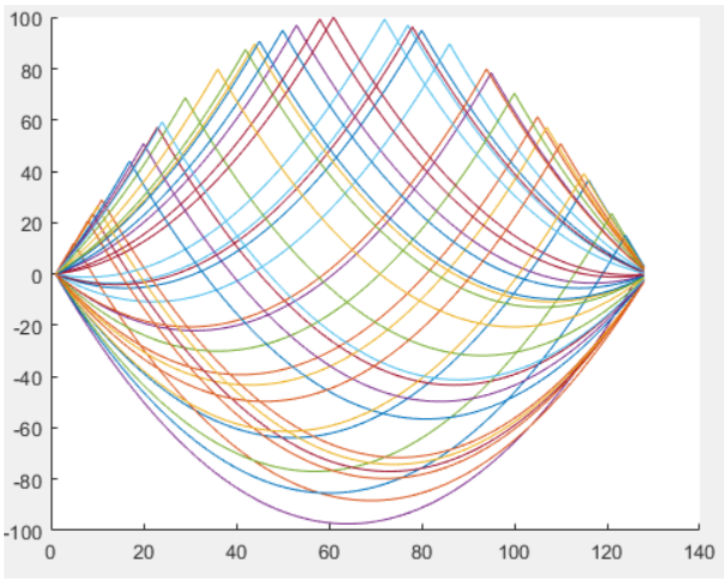


Figure 7: Phase of LoRa symbols versus time for a perfectly synchronized signal. The phase "reset" occurs at the start of each symbol, enabling coherent addition under ideal conditions.

Under such favorable conditions, the phase associated with the peak of the FFT bin could be subjected to compensation, analogous to the frequency offset correction applied during the symbol-by-symbol centering process. Consequently, the summation of the post-FFT bins would be performed on the entirety of the complex FFT output, rather than being restricted solely to the magnitude. This approach would inherently facilitate further improvements in sensitivity, potentially enabling performance levels that surpass those of conventional LoRa receivers.

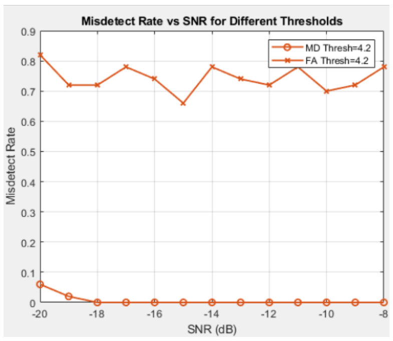
Decision Threshold: A signal is declared detected if the calculated PPSNR exceeds a pre-defined threshold.

4.3 Threshold Selection Process

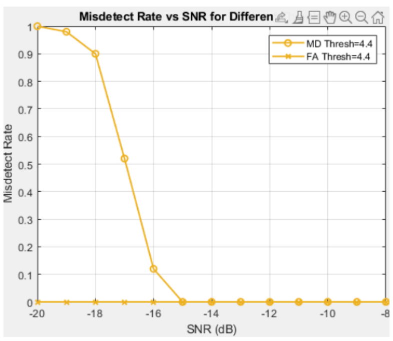
The selection of an appropriate Post-Processing SNR (PPSNR) threshold is critical for balancing detection sensitivity and false alarm rates. To determine an effective threshold, comprehensive False Alarm (FA) and Misdetection (MD) rate curves were generated by testing a valid signal against a noise-only reference, across a range of potential PPSNR threshold values.

Empirical analysis indicated that a PPSNR threshold of 4.4 dB resulted in 0% false alarm and 0% misdetection rates at an input SNR of -15 dB.

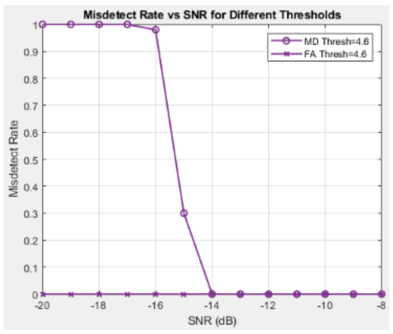
It is important to note that the Monte Carlo simulation conducted for this preliminary analysis utilized a limited number of iterations ($n = 50$) for expediency. For the final, rigorous parameter tuning and validation, a significantly larger test set would be employed. This would enable the establishment of more statistically robust performance metrics, typically aiming for a maximum false alarm rate of 1% or as stipulated by specific product requirements.



(a) Threshold = 4.2 dB



(b) Threshold = 4.4 dB



(c) Threshold = 4.6 dB

Figure 8: False Alarm and Misdetection rates for different PPSNR thresholds.

5 Final Simulation Results and Discussion

6 Simulation Parameters

The following parameters were used during the final simulation campaign to evaluate the blind LoRa detection algorithm:

- **Bandwidth (BW):** 125 kHz
- **Spreading Factor (SF):** 7
- **Number of Monte Carlo Iterations:** 1000 per hypothesis
- **Symbol Timing Offset (STO):** Brute-force correction over 8 equally spaced offsets ($M/8$)
- **Detection Threshold:** PPSNR = 4.4 dB (unless otherwise specified)
- **Channel Model:** AWGN, with random integer and fractional STO, and integer carrier frequency offset
- **Number of Symbols per Packet:** 100

These values reflect typical LoRa system settings and are chosen to demonstrate the algorithm's robustness under realistic and challenging conditions.

It should be noted that for each combination of spreading factor (SF) and bandwidth (BW), a dedicated minimum SNR threshold should ideally be determined by performing additional simulations. This ensures that the detection threshold is appropriately matched to the specific signal characteristics and noise resilience associated with each parameter set.

The preliminary minimum SNR threshold selection detailed in Section 4.2 was conducted under idealized conditions, specifically assuming zero symbol timing offset (STO) and utilizing a limited number of simulation iterations. To provide a more robust assessment of the algorithm's performance in realistic scenarios, a comprehensive final simulation campaign was executed.

This final assessment involved running the Monte Carlo simulation 1000 times per hypothesis. A key aspect of this simulation was the incorporation of brute-force STO "correction," where the algorithm tested 8 distinct symbol boundary starting positions, spaced equally at $M/8$ samples. This strategy aims to mitigate the effects of unknown STO.

For visualization of the impact of fractional STO, Figure X (referencing the figure displaying the worst-case sample offset) illustrates the impact of fractional STO. Specifically, Figure Y (referencing the figure showing the centered FFT at high SNR) presents a centered FFT at a high SNR of 20 dB, clearly demonstrating the consequences of a worst-case fractional STO. This scenario manifests as a noticeable peak power loss and a broadening of the strongest peak, often extending across two adjacent FFT bins with comparable power levels.

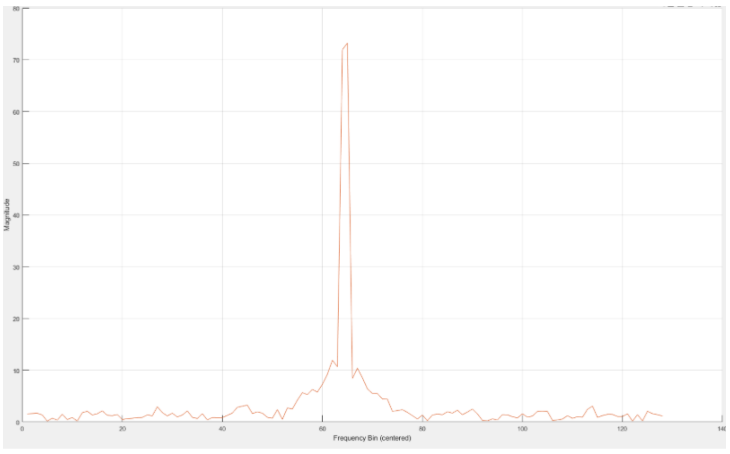


Figure 9: Impact of worst-case fractional symbol timing offset (STO) on the FFT output. The peak is broadened and split across adjacent bins, resulting in a loss of detection sensitivity.

The inclusion of these realistic STO variations in the simulation revealed an approximate 2 dB loss in detection sensitivity compared to the idealized scenario. Consequently, the final achieved sensitivity figure for the proposed blind LoRa detection algorithm is -13 dB SNR.

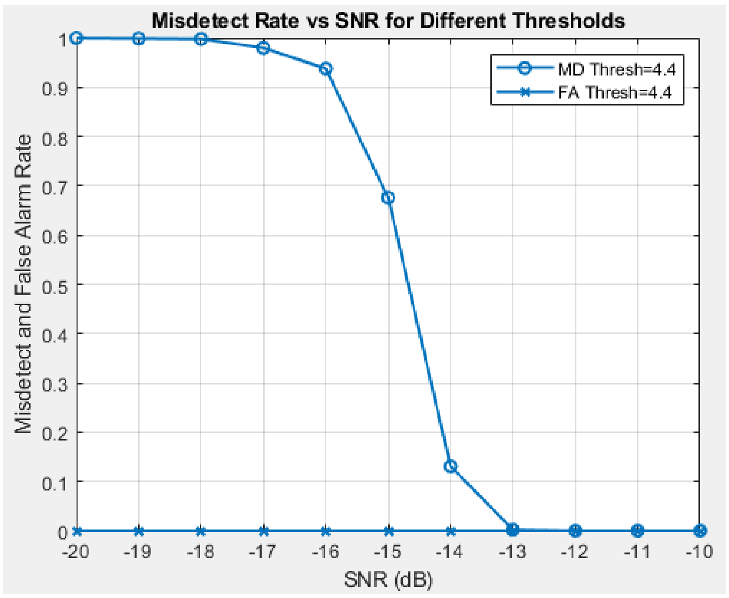


Figure 10: Centered FFT output at high SNR (20 dB) under worst-case fractional STO. The main peak is broadened and split, illustrating the impact of timing misalignment on detection performance.

The results demonstrate the robustness of the detection algorithm to timing offsets. The trade-off between misdetection and false alarm rates is controlled by the detection threshold. The algorithm maintains reliable detection even with a fractional sample delay.

7 Conclusion

A robust LoRa detection algorithm based on FFT segment analysis was developed and tested under challenging timing offset conditions and unknown carrier frequency. The method achieves low misdetection and false alarm rates at very low SNR.

8 Further Development

The current algorithm leverages the inherent correlation of the LoRa signal in contrast to the uncorrelated nature of noise samples. A promising avenue for further enhancing detection sensitivity involves exploiting this characteristic by introducing sample-offset processing.

A proposed improvement entails offsetting each received signal segment by ± 1 sample before dechirping and performing the FFT. This strategic offset is hypothesized to result in the peaks of the FFT outputs from these shifted segments being precisely one FFT bin apart. Crucially, a predictable, and ideally correctable, linear phase difference is anticipated between these offset peaks. By coherently combining these sample-offset post-FFT outputs, it is expected that a further improvement in the overall signal-to-noise ratio (SNR) can be achieved, thereby increasing detection performance in challenging low-SNR environments.

Appendix A: Analysis of Suboptimal Detection Approaches

This appendix discusses alternative analytical approaches to LoRa signal detection that were considered and subsequently deemed suboptimal for robust blind detection due to their inherent limitations.

A.1 Phase-Based Detection Methods

Initial analytical approaches explored the detection of LoRa signals by examining their unique phase characteristics:

- **Phase Reset per Symbol:** An ideal LoRa symbol, transmitted with coherent continuous phase, is theoretically expected to exhibit no net phase shift from its starting to ending sample. This implies a "phase reset" at the beginning of each symbol.
- **Constant Second Derivative of Phase:** Given the linear frequency modulation intrinsic to CSS, the second derivative of the signal's phase should be constant and directly proportional to the chirp rate ($K = \frac{BW}{T_s}$).

A.2 Limitations of Phase-Based Approaches

Despite their theoretical appeal, these phase-based methods suffer from significant practical drawbacks that limit their utility in real-world, low-SNR scenarios:

- **Sensitivity to Noise:** Phase information is highly susceptible to corruption by additive noise. Even at moderate SNRs, noise can introduce significant perturbations, obscuring the subtle phase characteristics that these methods attempt to exploit.
- **Dominance by Interferers:** The observed phase of a composite signal is primarily dominated by the most powerful component within the observed bandwidth. Consequently, if a narrowband (NB) interferer or any other strong signal is present in the band, its phase characteristics will overshadow those of the desired LoRa signal, rendering these detection techniques ineffective unless the target LoRa signal is unequivocally the strongest signal in the band.

A.3 Spectrogram-Based Feature Extraction

Instead of simple energy detection, this approach treats the initial detection as a feature extraction problem applied to a spectrogram. A spectrogram provides a time-frequency representation of the signal, visually highlighting variations in frequency over time. For a signal $x(t)$, its spectrogram $S_x(t,f)$ is typically computed as the squared magnitude of the short-time Fourier transform (STFT):

$$S_x(t,f) = |STFT(t,f)|^2 = \left| \int_{-\infty}^{\infty} x(\tau) w(\tau - t) e^{-j2\pi f \tau} d\tau \right|^2$$

where $w(t)$ is a window function.

Limitations Regarding SNR Requirements:

While spectrograms can visually reveal the characteristic linear frequency sweep of LoRa chirps (often appearing as diagonal lines), the efficacy of extracting such features is fundamentally constrained by the signal-to-noise ratio (SNR). At very low SNRs, the visual and quantitative distinctness of these characteristic "chirp lines" on the spectrogram becomes significantly degraded. The energy of the chirp, spread across both time and frequency, may not sufficiently stand out against the background noise, even when visualized. This makes it challenging for automated feature extraction algorithms (e.g., image processing techniques or pattern recognition) to reliably identify the LoRa signal's signature. Consequently, while spectrograms offer a richer representation than simple energy summation, their practical application for initial signal presence detection in extremely low-SNR conditions remains limited as the required features become indistinguishable from noise, failing to provide a robust initial trigger for a more detailed, LoRa-specific analysis stage.

Appendix B: Full MATLAB Code

Main Detection Script

```
% LoRaDetector_FFTsegments.m
% ————————————————————————————————————————
% This script generates a LoRa signal, passes it through a
    ↪ wireless channel,
% and visualizes the FFT output of each segment of length M (no
    ↪ preamble used).
% ————————————————————————————————————————

close all; clear all; clc;

plotDuringRun = false; % Set to true to enable plotting during
    ↪ SNR sweep

% SNR sweep parameters
SNRsweep = -20:1:-10; % dB

minSNRs = 4.4;%4:0.2:6; % Set your desired minSNR threshold(s)
    ↪ here
testVals = minSNRs;
misdetectRate = zeros(length(testVals), length(SNRsweep));
falseAlarmRate = zeros(length(testVals), length(SNRsweep));
numTrials = 1000;

% Parameters
SF = 7;              % Spreading Factor
BW = 125e3;          % Bandwidth (Hz)
Fs = BW;             % Original sampling frequency
M = 2^SF;            % Number of frequency bins (segment length)
```

```
Nsymbols = 100;    % Number of symbols to transmit
Nbits = Nsymbols * SF;

% Channel params
timingOffset = M/16 + 0.5; % Test at this specific timing
   ↪ offset (fractional sample)
maxDelay = 0;%timingOffset;     % For clarity, use this as the
   ↪ delay in the channel
max_Fo = BW;                    % No frequency offset
dF = BW/M;

for testIdx = 1:length(testVals)
    for snrIdx = 1:length(SNRsweep)
        SNR = SNRsweep(snrIdx);
        misdetects = zeros(1, numTrials);
        falseAlarms = zeros(1, numTrials);
        for trial = 1:numTrials

            % Generate random data bits
            TxDataBits = randi([0 1], Nbits, 1);

            % Modulate data (no preamble)
            modulatedSignal = CSSmod(reshape(TxDataBits, SF,
               ↪ []).', SF, BW, 1, true, false);
            modulatedSignal = modulatedSignal(:); % Ensure
               ↪ column vector


            % Apply timing offset (fractional delay) before
               ↪ channel
            delaySamples = timingOffset;
            %% Use linear interpolation for fractional delay
            t = (0:length(modulatedSignal)-1)';
            t_delayed = t - delaySamples;
            modulatedSignalDelayed = interp1(t,
               ↪ modulatedSignal, t_delayed, 'spline', 0);

            % Pass through wireless channel (no additional
               ↪ delay)
            [RxSignal, ~, ~] =
               ↪ wireless_channel(modulatedSignalDelayed, Fs,
               ↪ 0, max_Fo, SNR, dF);

            detected = lora_detect_signal(RxSignal, SF, BW,
               ↪ testVals(testIdx), M, plotDuringRun);
            misdetects(trial) = ~detected;

            % Noise-only test for false alarm
            noiseOnly = randn(length(RxSignal),1) +
               ↪ 1j*randn(length(RxSignal),1);

            detectedNoise = lora_detect_signal(noiseOnly, SF,
               ↪ BW, testVals(testIdx), M, plotDuringRun);
            falseAlarms(trial) = detectedNoise;
        end
        misdetectRate(testIdx, snrIdx) = mean(misdetects);
        falseAlarmRate(testIdx, snrIdx) = mean(falseAlarms);
        fprintf('TestVal = %.2f, SNR = %d dB, Misdetect Rate =
           ↪ %.2f, False Alarm = %.2f\n', testVals(testIdx),
           ↪ SNR, misdetectRate(testIdx, snrIdx),
           ↪ falseAlarmRate(testIdx, snrIdx));
    end
end

% Plot FAMD vs SNR

figure;
hold on;
colors = lines(length(testVals));
for testIdx = 1:length(testVals)
    figure;
    plot(SNRsweep, misdetectRate(testIdx,:), '-o', 'Color',
       ↪ colors(testIdx,:), 'LineWidth', 1.5, 'DisplayName',
       ↪ ['MD Thresh=' num2str(testVals(testIdx))]);
    hold on;
    plot(SNRsweep, falseAlarmRate(testIdx,:), '-x', 'Color',
       ↪ colors(testIdx,:), 'LineWidth', 1.5, 'DisplayName',
       ↪ ['FA Thresh=' num2str(testVals(testIdx))]);
    xlabel('SNR (dB)'); ylabel('Misdetect and False Alarm
       ↪ Rate');
    title('Misdetect Rate vs SNR for Different Thresholds');
    legend('show');
    grid on;
end



% --- Helper function for LoRa signal detection over all
   ↪ segments ---

function detected = lora_detect_signal(RxSignal, SF, BW,
   ↪ threshold, M, plotDuringRun)
    numSegments = floor(length(RxSignal)/M);
    fft_centered_all = zeros(M, numSegments);
    bitWord = de2bi(0, SF, 'left-msb');
    downchirp = CSSmod(bitWord, SF, BW, -1, true, false);
    for segIdx = 1:numSegments
        segment = RxSignal((segIdx-1)*M+1 : segIdx*M);
        dechirped = segment .* downchirp;
        fft_raw = fftshift(fft(dechirped));
        [~, peakIdx] = max(abs(fft_raw));
        shift_amt = (M/2 + 1) - peakIdx;
        fft_centered = circshift(fft_raw, shift_amt);
        fft_centered_all(:, segIdx) = fft_centered;
        if plotDuringRun
            figure(1001); hold on;
            plot(abs(fft_centered), 'Color', [0.7 0.7 0.7 0.5]);
        end
    end

    fft_combined = mean(abs(fft_centered_all),2);

    if plotDuringRun
        figure(1001); hold on;
        plot(fft_combined, 'k', 'LineWidth', 2, 'DisplayName',
            ↪ 'Mean Combined');
        title('All Centered FFTs (Overlayed) and Mean');
        xlabel('Frequency Bin (centered)'); ylabel('Magnitude');
        grid on;
    end

    peakVal = fft_combined(M/2+1);
    wrapIdx = mod((M/2-2:M/2+4)-1, M) + 1;
    noiseVals = fft_combined;
    noiseVals(wrapIdx) = [];
    noiseMean = mean(noiseVals);
    metric = 10*log10(peakVal / noiseMean);
    detected = metric > threshold;
    if plotDuringRun
        fprintf('Combined FFT: Peak=%.2f, NoiseMean=%.2f,
            ↪ Metric=%.2f, Threshold=%.2f, Detected=%d\n',
            ↪ peakVal, noiseMean, metric, threshold, detected);
        title(['PPSNR = ' num2str(metric) ' [dB]']);
    end
end

% --- Helper function for segment metric calculation ---
% (Optional) Segment-level metric calculation function
% Not used in main detection, but useful for debugging or
    ↪ alternate logic
function [metric, detectedSeg, fft_out, peakIdx] =
    ↪ lora_segment_detection(segment, SF, BW, threshold)
    bitWord = de2bi(0, SF, 'left-msb');
    downchirp = CSSmod(bitWord, SF, BW, -1, true, false);
    dechirped = segment .* downchirp;
    fft_out = abs(fftshift(fft(dechirped)));
    [peakVal, peakIdx] = max(fft_out);
    Nfft = length(fft_out);
    wrapIdx = mod((peakIdx-4:peakIdx+3)-1, Nfft) + 1;
    noiseVals = fft_out;
    noiseVals(wrapIdx) = [];
    noiseMean = mean(noiseVals);
    metric = 10*log10(peakVal / noiseMean);
    detectedSeg = metric > threshold;
end
```

Accessory Functions and Scripts

```
function [y,delay,Foffset] = wireless_channel(x, fs, max_delay,
   ↪ max_freq_offset, snr_dB,dF)
  % x: input signal asusmed to be column
  % fs: sampling frequency (Hz)
  % max_delay: maximum delay in samples
  % max_freq_offset: maximum frequency offset in Hz
  % snr_dB: signal-to-noise ratio in dB

  if size(x,1) == 1 %make x column if it's not
    x = x.';
  end

  % Random delay
  delay = randi([0, max_delay]);
  x_delayed = [zeros(delay,1);x;zeros(floor(length(x)/4),1)];
  N = length(x_delayed);

  % Random frequency offset
  Foffset = (2*rand - 1) * max_freq_offset;  % Uniform in
     ↪ [-max, max]

  FoInt = floor(Foffset / dF) * dF;
  FoFrac = Foffset - FoInt;

  % Enable fractional freq offset
  Foffset = FoInt + FoFrac;
  % disp(['freq offset is : ' num2str(Foffset/dF) ' times dF,
     ↪ frac offset = ' num2str(FoFrac) ' Hz']);

  t = (0:N-1)' / fs;
  freq_shift = exp(1j * 2 * pi * Foffset * t);
  x_freq_shifted = x_delayed .* freq_shift;

  % Add AWGN
```

```
    signal_power = mean(abs(x_freq_shifted).^2);
    noise_power = signal_power / (10^(snr_dB/10));
    noise = sqrt(noise_power/2) * (randn(size(x_freq_shifted)) +
        ↪ 1j*randn(size(x_freq_shifted)));
    y = x_freq_shifted + noise;
end

function cssSamples = CSSmod(dataBits, SF, BW, chirpSign,
    ↪ doFM, verbose)
% CSSmod - Chirp Spread Spectrum (CSS) modulation
%
%   cssSamples = CSSmod(dataBits, SF, BW, chirpSign, verbose,
    ↪ doFM)
%
%   Inputs:
%     dataBits   : Vector of input bits to modulate
%     SF         : Spreading Factor (number of bits per symbol)
%     BW         : Bandwidth (Hz)
%     chirpSign  : 1 for up-chirp, -1 for down-chirp (default is
    ↪ 1)
%     verbose    : (optional) true to plot, false to suppress
    ↪ plots (default: false)
%     doFM       : (optional) true to output FM modulated signal
    ↪ (default: true)
%
%   Output:
%     cssSamples: Vector of complex CSS modulated samples (if
    ↪ doFM true)
%                 or matrix of instantaneous frequency values
    ↪ (if doFM false)
%
%   The function pads dataBits if needed, splits into symbols,
    ↪ and generates
%   a CSS modulated signal for each symbol. It can also plot
    ↪ the instantaneous
%   frequency and the spectrogram for each symbol if verbose is
    ↪ true.

if nargin < 4
    chirpSign = 1;
end
if nargin < 5
    doFM = true;
end
if nargin < 6
    verbose = false;
end
```

```
Fs = BW; % Sampling frequency, set to bandwidth for simplicity
bitsPerSymbol = SF; % Number of bits per symbol
M = 2^SF;
sps = M; % Samples per symbol

dataBits = dataBits(:)'; % Ensure row vector for bit grouping

% Pad dataBits with zeros if not a whole multiple of
    ↪ bitsPerSymbol
numBits = length(dataBits);
remainder = mod(numBits, bitsPerSymbol);
if remainder ~= 0
    padLength = bitsPerSymbol - remainder;
    dataBits = [dataBits, zeros(1, padLength)];
end
Nsymbols = length(dataBits) / bitsPerSymbol; % Number of symbols

%       k = (0:M-1);                    % k vector
t = (0:1/Fs:(M - 1)/Fs); % Time vector for one symbol

df = BW / M; % Frequency step

cssSamples = [];


for curSymbol = 1:Nsymbols
    bits = dataBits((curSymbol-1)*bitsPerSymbol+1 :
        ↪ curSymbol*bitsPerSymbol);
    % Convert bits to integer symbol (MSB first)
    m = bi2de(bits, 'left-msb');
    fm = -BW/2 + m*df;
    fi =fm + chirpSign*t*BW*df;



    if verbose
        % Plot instantaneous frequency
        figure;
        plot(t, fi);
        title([" fi for symbol ", num2str(curSymbol), "
            ↪ (chirpSign = ", num2str(chirpSign), ")"]);
        xlabel("Time (s)"); ylabel(" fi ");
    end
    if doFM
        % FM Modulation
        phase = 2*pi*cumtrapz(t, fi);

        fmSignal = (exp(1i*phase)).'; % Complex baseband CSS
            ↪ signal
        cssSamples = [cssSamples; fmSignal]; % Concatenate as
            ↪ column vector
        if verbose
            plotCSSSpectrogram(fmSignal, Fs, M, curSymbol,-20);
        end
    else
        cssSamples = [cssSamples; fi(:)];
    end
end

end

function plotCSSSpectrogram(signal, Fs, M, curSymbol,
    ↪ minThreshold)
% plotCSSSpectrogram - Plots double-sided spectrogram for a CSS
    ↪ signalsignal
%   plotCSSSpectrogram(signal, Fs, M, curSymbol, minThreshold)
%   signal: complex baseband CSS signal
%   Fs: sampling frequency
%   M: FFT length (number of frequency bins)
%   curSymbol: symbol index (for title)
%   minThreshold: (optional) minimum dB threshold for display

    figure;
    winLen = max(floor(length(signal)/200),4); % 50 windows per
        ↪ symbol
    [S,F,T] = spectrogram(signal, winLen, floor(winLen/2), M,
        ↪ Fs,'yaxis');
    S = fftshift(S,1);
    S_dB = 20*log10(abs(S));
    % Set min threshold to 3 dB below max peak
    minThreshold = max(S_dB(:)) - 5;
    S_dB(S_dB < minThreshold) = minThreshold;
    imagesc(T, F-Fs/2, S_dB);
    axis xy;
    xlabel('Time (s)'); ylabel('Frequency (Hz)');
    title(['Double-sided Spectrogram for symbol '
        ↪ num2str(curSymbol)]);
    colorbar;
    % Draw vertical lines at symbol boundaries if signal is
        ↪ longer than one symbol
    numSymbols = floor(length(signal)/M);
    if numSymbols > 1
        hold on;
        for k = 1:numSymbols-1
            % Symbol boundary in samples
            sampleIdx = k*M;
            % Convert sampleIdx to time (seconds)
            tBoundary = sampleIdx/Fs;
            xline(tBoundary, 'w--', 'LineWidth', 1.5);
        end
        hold off;
    end
end
```