GSM Signal Monitor

Real-time Spectrum Analysis Tool

Harea Teodor-Adrian

January 11, 2026

Contents

1 Introduction

1.1 Project Overview

GSM Signal Monitor is a real-time spectrum analysis application designed to monitor and visualize GSM network signals in Romania. The application uses an RTL-SDR dongle to scan radio frequencies and identify signals from major Romanian mobile network providers: Orange, Vodafone, Telekom, and Digi.

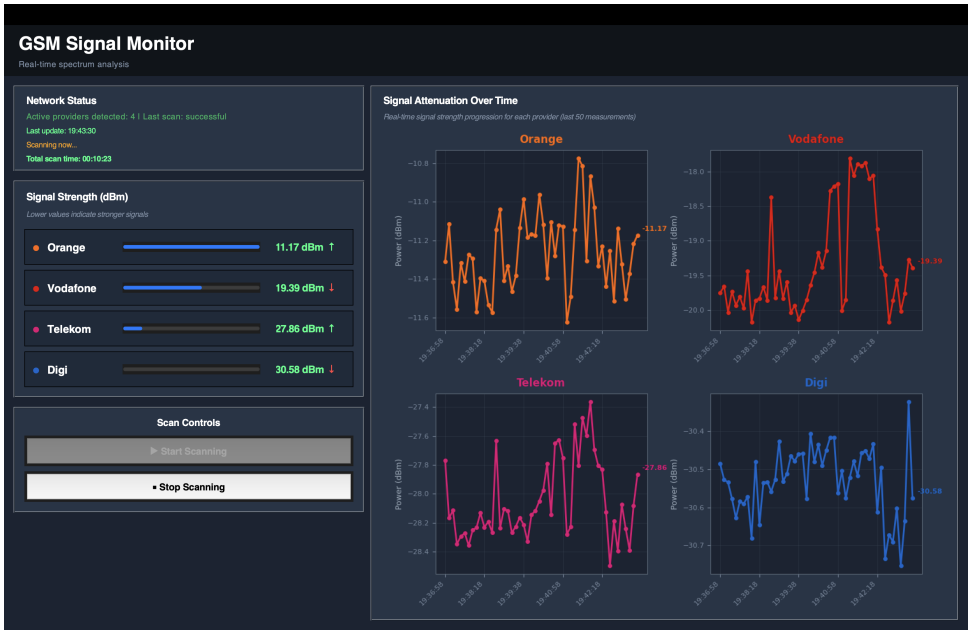

Figure 1: GSM Signal Monitor Main Interface

1.2 Key Features

- Real-time signal strength monitoring for all major providers
- Visual signal strength bars with color-coded indicators
- Historical signal data graphs (last 50 measurements)
- Automatic provider identification
- Signal trend indicators (improving/degrading)
- Scan session timer and statistics

2 System Requirements

2.1 Hardware Requirements

- RTL-SDR USB dongle (RTL2832U chipset recommended)
- Computer with USB 2.0 or higher port
- Antenna suitable for 900 MHz range

2.2 Software Requirements

- Python 3.8 or higher
- macOS, Linux, or Windows operating system
- RTL-SDR drivers installed

3 Installation Guide

3.1 Step 1: Install RTL-SDR Drivers

For macOS:

Install via Homebrew

```
brew install librtlsdr
```

For Linux (Ubuntu/Debian):

Install via apt

```
sudo apt-get update
sudo apt-get install rtl-sdr librtlsdr-dev
```

3.2 Step 2: Install Python Dependencies

Install Required Packages

```
pip install numpy scipy matplotlib tkinter pyrtlsdr
```

3.3 Step 3: Run the Application

Launch GSM Monitor

```
cd /path/to/project
python gsm_monitor_gui.py
```

4 User Guide

4.1 Starting a Scan

1. Connect your RTL-SDR dongle to a USB port
2. Launch the application
3. Click the **"Start Scanning"** button

4. Wait for the system to initialize (3-5 seconds)
5. Signal data will begin appearing in real-time

4.2 Understanding the Interface

The interface is divided into two main sections:

4.2.1 Left Panel - Signal Information

- **Network Status**: Shows scan status and detected providers
- **Last Update**: Timestamp of most recent scan
- **Next Update**: Countdown to next scan cycle
- **Total Scan Time**: Elapsed time since scan started
- **Signal Strength Bars**: Visual representation for each provider

4.2.2 Right Panel - Signal Graphs

- Four graphs (one per provider)
- Shows signal strength progression over time
- Displays last 50 measurements
- Color-coded by provider

4.3 Interpreting Signal Values

Signal strength is measured in **dBm** (decibel-milliwatts):

Signal Strength (dBm)	Quality
-50 to -70	Excellent
-70 to -85	Good
-85 to -100	Fair
Below -100	Poor

Table 1: Signal Quality Guide

Important: Lower (more negative) values indicate weaker signals.

4.4 Signal Trend Indicators

- ↑ **Green Arrow**: Signal is improving (getting stronger)
- ↓ **Red Arrow**: Signal is degrading (getting weaker)
- **No Arrow**: Signal strength unchanged

4.5 Stopping a Scan

1. Click the "**Stop Scanning**" button
2. Wait for the system to safely close the SDR connection
3. Total scan time will be displayed

5 GSM Frequencies in Romania

5.1 Frequency Bands Overview

Romanian mobile networks operate on the following GSM frequency bands:

Band	Downlink (MHz)	Uplink (MHz)
GSM 900	935.0 - 960.0	890.0 - 915.0
GSM 1800	1805.0 - 1880.0	1710.0 - 1785.0

Table 2: GSM Frequency Bands in Romania

5.2 Provider Frequency Allocations

This application monitors the **GSM 900 band** (935-960 MHz range) which is the primary band used by all Romanian providers.

5.2.1 Orange Romania

- Primary band: GSM 900 (935-945 MHz)
- Network code: 226-10
- Color indicator: Orange (#FF6600)

5.2.2 Vodafone Romania

- Primary band: GSM 900 (945-950 MHz)
- Network code: 226-01
- Color indicator: Red (#E60000)

5.2.3 Telekom Romania

- Primary band: GSM 900 (950-955 MHz)
- Network code: 226-03
- Color indicator: Magenta (#E20074)

5.2.4 Digi Romania

- Primary band: GSM 900 (955-960 MHz)
- Network code: 226-05
- Color indicator: Blue (#0066CC)

5.3 Scanning Method

The application uses the following scanning approach:

1. **Frequency Range**: 935-960 MHz (GSM 900 downlink)
2. **Sample Rate**: 2.4 MHz
3. **Gain**: 40 dB (configurable)
4. **Scan Interval**: 5 seconds between scans
5. **Processing**: FFT-based power spectrum analysis

6 Technical Details

6.1 Signal Processing

The application performs the following steps:

1. **Data Acquisition**: Captures IQ samples from RTL-SDR
2. **FFT Analysis**: Converts time-domain to frequency-domain
3. **Power Calculation**: Computes signal power in dBm
4. **Peak Detection**: Identifies strongest signals per provider
5. **Provider Mapping**: Associates frequencies with providers

6.2 Provider Identification

Providers are identified by their frequency ranges:

Frequency Mapping

```
PROVIDERS = {
    "Orange": (935e6, 945e6),    # 935-945 MHz
    "Vodafone": (945e6, 950e6),  # 945-950 MHz
    "Telekom": (950e6, 955e6),   # 950-955 MHz
    "Digi": (955e6, 960e6)       # 955-960 MHz
}
```

7 Code Implementation

This section presents key code snippets that demonstrate the core functionality of the GSM Signal Monitor application.

7.1 SDR Configuration and Initialization

The RTL-SDR device must be properly configured to receive GSM signals. This code initializes the hardware with appropriate parameters.

SDR Setup and Configuration

```
from rtlsdr import RtlSdr
import numpy as np

def configure_sdr(self):
    """Configure RTL-SDR device for GSM monitoring"""
    try:
        self.sdr = RtlSdr()

        # Center frequency: middle of GSM 900 band
        self.sdr.center_freq = 947.5e6  # 947.5 MHz

        # Sample rate: 2.4 MHz bandwidth
        self.sdr.sample_rate = 2.4e6

        # Gain: 40 dB for optimal reception
        self.sdr.gain = 40

        print("SDR configured successfully")
        return True

    except Exception as e:
        print(f"SDR configuration failed: {e}")
        return False
```

Explanation: This function initializes the RTL-SDR with a center frequency of 947.5 MHz, positioned in the middle of the GSM 900 band (935-960 MHz). The 2.4 MHz sample rate provides sufficient bandwidth to capture all provider frequencies simultaneously. A gain of 40 dB ensures strong signal reception while maintaining a good signal-to-noise ratio.

7.2 FFT-Based Signal Processing

The core of the application uses Fast Fourier Transform to analyze the frequency spectrum and extract signal strength information.

Frequency Spectrum Analysis

```python
def scan_frequencies(self):
    """Scan GSM frequencies and calculate signal strengths"""
    try:
        # Capture 256K IQ samples from SDR
        samples = self.sdr.read_samples(256 * 1024)

        # Apply FFT to convert time-domain to frequency-domain
        fft_result = np.fft.fft(samples)
        fft_magnitude = np.abs(fft_result)

        # Convert magnitude to power in dBm
        power_db = 20 * np.log10(fft_magnitude + 1e-10)

        # Calculate frequency bins
        freqs = np.fft.fftfreq(len(samples),
                               1/self.sdr.sample_rate)
        freqs = freqs + self.sdr.center_freq

        # Extract peak signal for each provider
        results = {}
        for provider, (freq_min, freq_max) in PROVIDERS.items():
            mask = (freqs >= freq_min) & (freqs <= freq_max)
            if np.any(mask):
                provider_power = power_db[mask]
                results[provider] = np.max(provider_power)
            else:
                results[provider] = -120

        return results

    except Exception as e:
        print(f"Scan error: {e}")
        return None
```

Explanation: This function captures 256,000 IQ samples and applies FFT to analyze the frequency content. The FFT output is converted to power values in dBm. For each provider's frequency range, the function identifies the peak signal strength, representing the strongest detected signal in that band. A small constant (1e-10) prevents logarithm errors when signal is absent.

7.3 Real-Time Data Visualization

The application uses matplotlib to create dynamic graphs showing signal history for each provider.

Graph Update Function

```
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

def update_graphs(self):
    """Update signal history graphs for all providers"""
    self.fig.clear()

    for i, (provider, color) in enumerate(PROVIDER_COLORS.items()):
        ax = self.fig.add_subplot(2, 2, i+1)

        history = self.signal_history[provider]
        if len(history) > 0:
            x_values = range(len(history))

            # Plot line graph with markers
            ax.plot(x_values, history, color=color,
                    linewidth=2, marker='o', markersize=3)

            # Fill area under curve
            ax.fill_between(x_values, history, -120,
                            alpha=0.3, color=color)

        # Graph formatting
        ax.set_title(f"{provider} Signal Strength",
                     fontsize=11, fontweight='bold')
        ax.set_xlabel("Measurement #")
        ax.set_ylabel("Power (dBm)")
        ax.set_ylim(-120, -40)
        ax.grid(True, alpha=0.3)

        # Reference line at -85 dBm (good/fair threshold)
        ax.axhline(y=-85, color='orange',
                   linestyle='--', alpha=0.5)

    self.fig.tight_layout()
    self.canvas.draw()
```

Explanation: This function creates a 2×2 grid of graphs, one for each provider. Each graph displays up to 50 historical measurements as a line plot with filled area underneath. The orange dashed line at -85 dBm marks the threshold between good and fair signal quality. Graphs are color-coded using each provider's brand colors for easy identification.

7.4 Threading for Non-Blocking Operation

To maintain a responsive GUI, scanning operations run in a separate background thread.

Asynchronous Scanning Thread

```
import threading
import time

def start_scanning(self):
    """Initialize and start the scanning thread"""
    if self.configure_sdr():
        self.scanning = True
        self.scan_thread = threading.Thread(
            target=self.scan_loop,
            daemon=True
        )
        self.scan_thread.start()
        self.start_button.config(state='disabled')
        self.stop_button.config(state='normal')

def scan_loop(self):
    """Main scanning loop (runs in background thread)"""
    while self.scanning:
        # Perform frequency scan
        results = self.scan_frequencies()

        if results:
            # Schedule GUI update on main thread
            self.root.after(0,
                lambda r=results: self.update_display(r))

        # Wait 5 seconds before next scan
        time.sleep(5)

def stop_scanning(self):
    """Stop scanning and cleanup"""
    self.scanning = False
    if hasattr(self, 'sdr'):
        self.sdr.close()
    self.start_button.config(state='normal')
    self.stop_button.config(state='disabled')
```

Explanation: The scanning process runs in a daemon thread to prevent blocking the GUI. The `scan_loop` continuously scans frequencies every 5 seconds while `self.scanning` is True. After each scan, it uses `root.after()` to schedule GUI updates on the main thread, which is required for thread-safe Tkinter operations. The daemon thread automatically terminates when the application closes.

8 Limitations

- Only monitors GSM 900 band (not 1800/2100 MHz)
- Does not decrypt or decode GSM traffic
- Requires line-of-sight to cell towers for best results
- Signal strength affected by building materials and obstacles

- Provider identification based on frequency ranges only

9 Safety and Legal Notice

9.1 Legal Compliance

This tool is designed for **educational and monitoring purposes only**. It operates in receive-only mode and does not transmit any signals.

9.2 Important

- Only receives publicly broadcast signals
- Does not intercept or decode communications
- Complies with Romanian telecommunications regulations
- Intended for signal quality monitoring