Real-Time FM Demodulation Using RTL-SDR and MATLAB - A Practical SDR and DSP Experiment

Electrical Engineering Department
DSP Laboratory (250704)
Instructor: Mohammadreza Bagheri Jazi

May 17, 2025

Contents

0.1 Introduction

In the last session, we focused on **offline FM signal demodulation** by recording complex I-Q samples from an FM broadcast and processing them in MATLAB after acquisition. This allowed us to explore the fundamental steps of FM demodulation such as filtering, phase differentiation, and decimation without real-time constraints.

In this session, however, we aim to transition from offline to **real-time signal processing**. That means our system must acquire radio signals continuously using an RTL-SDR dongle, process the incoming samples *on the fly*, and immediately play the demodulated audio through the computer's speakers all in a serial pipeline.

Achieving this requires the use of **frame processing**, a common approach that we have previously worked with. In frame processing, incoming data samples are stored in a temporary buffer (or "frame"), and each frame is processed and sent to the output device in a loop. In our case:

- The input is a frame of I-Q samples from the RTL-SDR
- The processing block is our FM demodulator
- The output is real-time audio played through your laptop speakers

Compared to offline analysis, real-time processing presents unique challenges:

- Ensuring timely data flow (low latency)
- Maintaining synchronization between acquisition and playback
- Avoiding buffer overflows or underflows

Consult with your group members: What might be the potential bottlenecks or sources of latency in this pipeline? How can we optimize performance or handle errors such as audio glitches or dropped frames? Thinking ahead will prepare you for debugging and extending the system with advanced features.

1 System Implementation: Step-by-Step Guide

Step 1: Define Parameters

Dedicate a separate section in your code for defining all relevant **system parameters**, such as:

- Sampling rate
- Carrier frequency
- Audio sampling rate
- Any other configurable constants or settings

Tip: Use clear and descriptive variable names, and group these definitions at the beginning of your script.

```
    centerFreq_MHz = 93.5;
    sampleRate = 240e3;
    audioFs = 48e3;
```

Step 2: Configure RTL-SDR Receiver

In this part of, you must define a System object that interfaces with an RTL-SDR (Software Defined Radio) hardware device. MATLAB provides the `comm.SDRRTLReceiver` object for this purpose, which allows you to receive real-time IQ signals.

```
    radio = comm.SDRRTLReceiver( ...
    'CenterFrequency', centerFreq_MHz*1e6, ...
    'SampleRate', sampleRate, ...
    'OutputDataType', 'double', ...
    'EnableTunerAGC', true, ...
    'SamplesPerFrame', 1024*10);
```

Note: Do not copy the full code. Refer to this explanation to build your own configuration line by line.

The object has several important parameters:

- `CenterFrequency`: The radio frequency (in Hz) that the SDR will tune to. For example, if you want to receive a signal at 100, you must convert this to `100e6`.
- `SampleRate`: The number of samples per second captured from the RF signal. Choose a value supported by your SDR device (commonly between 225 and 3.2 for RTL-SDR).
- `OutputDataType`: The data type of the samples. You can use `'double'` to get double-precision values for signal processing.
- `EnableTunerAGC`: A logical value that enables (true) or disables (false) Automatic Gain Control (AGC). If enabled, the receiver will automatically adjust the gain to maintain a stable signal level.
- `SamplesPerFrame`: Number of samples acquired in each frame (each call to the receiver). Larger values increase latency but reduce overhead.

Note: You are free to adjust the values to match your signal of interest and hardware limitations.

Step 3: Spectrum Analyzer (Optional)

```
    specAnalyzer = dsp.SpectrumAnalyzer( ...
    'SampleRate', sampleRate, ...
    'SpectrumType', 'Power', ...
    'Title', 'RTL-SDR␣Real-Time␣Spectrum', ...
    'ShowLegend', true);
```

Step 4: FM Demodulator Setup

After receiving each frame from the SDR module, the next step is to extract the audio content from the modulated signal. For this, we use the `comm.FMBroadcastDemodulator` System object, which performs demodulation of a wideband FM broadcast signal.

```
fmDemod = comm.FMBroadcastDemodulator( ...
'SampleRate', sampleRate, ...
'FrequencyDeviation', 75e3, ...
'AudioSampleRate', audioFs);
```

This object takes in a complex baseband signal (typically output from the SDR receiver) and outputs the demodulated audio signal.

Parameter Descriptions

- `SampleRate`: This should match the sample rate of the incoming signal, i.e., the rate at which the SDR receiver provides samples. It ensures the demodulator is synchronized with the incoming data stream.
- `FrequencyDeviation`: This specifies the peak frequency deviation of the FM signal. In standard FM broadcasting, this is typically 75.
- `AudioSampleRate`: The rate at which the demodulated audio will be sampled. A typical value is 48000, which is standard for audio playback.

Hint: Choose `AudioSampleRate` based on the playback capabilities of your system, and make sure it matches the configuration of any downstream audio player object.

Step 5: Audio Output

To play the audio signal, several MATLAB functions are available, such as `sound()` and `soundsc()`. However, for real-time streaming applications, the most appropriate option is to define an object of type `audioDeviceWriter`.

The most important feature of this object in our application is its ability to buffer the audio frames it receives and play them in the correct order. This internal buffering mechanism helps manage playback timing automatically, eliminating the need for manual timing control in your code.

```
player = audioDeviceWriter('SampleRate', audioFs);
```

Parameter Description

- `SampleRate`: This sets the sampling rate (in Hz) for audio playback. It must match the audio signal's sampling rate (e.g., from the FM demodulator) and be supported by your systems audio hardware.

  Typical values: `44100`, `48000`, or any value between `8000` and `192000` Hz, depending on your system.

Note: If you choose a rate not supported by your audio hardware, playback may be distorted or fail to initialize.

Step 6: Main Processing Loop

```
while true
iq = double(radio());
iq = iq / max(abs(iq)); % Normalize
specAnalyzer(iq);
audio = fmDemod(double(iq));
player(audio);
end
```

Step 0: Releasing System Objects Before Reinitialization

When working with **System objects** in MATLAB (such as radios, demodulators, audio players, etc.), it's important to `release` and `clear` these objects before re-running your script. This practice ensures:

- Hardware or system resources used by the object are freed.
- The object can be safely reconfigured or recreated.
- Runtime errors such as "object is locked" are avoided.

Before creating a new instance of a System object (e.g., `radio`, `fmDemod`, `player`, or `specAnalyzer`), write a short block of code that:

1. Checks if the object already exists in the workspace.
2. Releases it if necessary.
3. Clears it from memory.

Hint: Use the MATLAB functions `exist`, `release`, and `clear`.

Write the code yourself. Do not copy-paste from the lecture slides.

Example

```
if exist('radio','var'), release(radio); clear radio; end
```

Releases any previous SDR or audio objects to avoid conflicts.

1.1 Your Task

In this task, you are required to design and implement your own FM demodulator function, replacing the use of MATLAB's built-in `comm.FMBroadcastDemodulator` object.

Instructions:

- Define a custom FM demodulation function using signal processing techniques (e.g., phase differentiation).
- Replace the predefined FM demodulator in your code with your custom function and run the complete FM reception chain.

- Compare the quality of the audio output from your custom demodulator with the output from MATLABs built-in demodulator.
- Analyze and explain the source of any differences in output quality. Are they related to filter design, frequency deviation handling, or sampling rate mismatches?
- Try to improve the performance of your custom demodulator by adjusting the design of your filters (e.g., changing filter type, order, or cutoff frequency). Document the improvements and justify the design changes.