天文无线电技术基础 期末报告

使用 RTL-SDR 接收并解码 TETRA 信号

滕一鸣 [a]

[a] *2000011380*

E-mail: tengyiming@stu.pku.edu.cn

1 导言

TETRA (Terrestrial Trunked Radio，地面中继式无线电) 是世界范围内被广泛应用于政府部门、紧急服务部门以及军队的中继式无线电系统。一般而言，民用行业在使用这一系统时不会进行加密，因此我们对它的解码将不会出现法律问题。我们考虑使用 RTL-SDR 解码 TETRA 的原因有二：一是北京地铁的通信方案使用了这一系统，因而我们在毗邻地铁站的校园内使用非常一般的天线也能接收到信噪比可观的信号；二是目前已有业余无线电爱好者给出了解码 TETRA 的技术资料，对这些资料的复现势必可以加深我们对软件无线电、信号处理以及数字通信的认识，并让一些~~跟笔者一样的~~人尽快熟悉 Linux 系统的使用。

2 原理与实现方法

2.1 配置 RTL-SDR 驱动

我们使用的系统环境是 Windows 11 WSL 中的 Ubuntu 20.04.5 LTS。对于 Linux 环境下 RTL-SDR 的配置，我们采用的方法如下：

1. 使用`sudo apt-get update`更新后执行`sudo apt-get install build-essential`以安装必需的 build 工具。
2. 执行`sudo apt-get install libusb-1.0-0-dev`以安装与 USB 设备通信所必需的 libusb-1.0-0-dev 库。
3. 安装 RTL2832U Osmocom 驱动：

```
git clone git://git.osmocom.org/rtl-sdr.git
cd rtl-sdr/
mkdir build
cd build
cmake ../ -DINSTALL_UDEV_RULES=ON
make
sudo make install
sudo ldconfig
sudo cp ../rtl-sdr.rules /etc/udev/rules.d/
```

4. 将电视棒接收电视信号所使用的驱动拉入黑名单以避免与 Osmocom 驱动冲突：进入`~/etc/modprobe.d`后创建`blacklist-rtl.conf`，写入“blacklist dvb_usb_rtl28xxu”，之后保存文件并重启系统。

2.2 建立 WSL 子系统与 USB 器件的通信

WSL 并不能直接访问插入的 USB 设备，因此我们还需要配置 usbipd：

1. 在 powershell 中执行 `winget install --interactive --exact dorssel.usbipd-win`
2. 在 Ubuntu 中执行

```
sudo apt install linux-tools-5.4.0-77-generic hwdata
sudo update-alternatives --install /usr/local/bin/usbip usbip
/usr/lib/linux-tools/5.4.0-77-generic/usbip 20
```

3. 以管理员身份运动 powershell 并执行`usbipd list` 以查看电视棒的 BUSID，然后执行 `usbipd wsl attach --busid <BUSID>`

2.3 配置环境

1. `sudo apt-get install gnuradio` 安装最新版的 GNURadio。
2. 使用 https://slackbuilds.org/repository/15.0/libraries/rapidjson/?search=rapidjson 提供的资源与教程安装安装 rapidjson-1.1.0。
3. `sudo apt-get install -y gr-osmosdr` 安装为 GNURadio 提供 RTL-SDR 支持的 `gr-osmosdr` 模块。
4. `sudo apt-get install libx11-dev`
5. 执行 `sudo apt remove soapysdr0.7-module-remote`。这是因为安装 `gr-osmosdr` 时会自动安装 Soapy，而这会安装 Soapy Remote 模块，但是这个模块必须在同一个 Avahi 服务器建立链接后才能运行。如果不移除 Soapy Remote 的话，我们在执行 GNURadio Companion 流程图时会出现报错信息“avahi_client_new() failed: Daemon not running”并使编译失败。

2.4 解调

TETRA 信号使用的调制方式为 $\pi/4$ DQPSK。我们使用的解调模块基于 https://github.com/Tim---/tetra-toolkit.git 。将其 `git clone` 到本地后进入 `grc` 目录，其中应该包含 `tetra_demod.grc` 和 `tetra_rx.grc` 两个 GNURadio Companion 文件。如果是使用的最新版本 GNURadio，那么 `tetra_demod.grc` 中的 fractional resampler 会显示模块丢失，此时将其替换成在程序中搜索得到的 fractional resampler 即可。此后执行这一流程图，这会在 GNURadio Companion 中生成一个名为 TETRA Demod 的 hier block。

打开 `tetra_rx.grc`，将流程图中的 osmocom Source 替换成 RTL-SDR Source 后执行这一流程图。对于笔者使用的 GNURadio，其在编译流程图时存在使用 python 3 但是调用了 python 2 的 ConfigParser 库的问题，这会导致报错“ModuleNotFoundError: No module named 'ConfigParser'”。为此我们需要打开编译得到的 `tetra_rx.py` 文件并将 `import ConfigParser` 替换为 `import configparser as ConfigParser` 以将之替换为正确的 python 3 库。此后执行 `sudo python3 tetra_rx.py` 即可正常运行解调程序。虽说我们在配置 RTL-SDR 驱动时对 `cmake` 的设置应该可以使得运行电视棒不需要 root 权限，但是笔者发现这一解调程序还是需要 `sudo` 才能正常运行。

接下来我们将介绍这一解调模块的工作原理：我们在 `tetra_rx.grc` 中使用 RTL-SDR Source 接收了电视棒对射频信号采样得到的数据；这一数据一边被送入了 QT GUI Frequency Sink 中生成了频谱图、一边被送入了 QT GUI Waterfall Sink 里生成了数据图、一边被送入了由 `tetra_demod.grc` 定义的 hier block 中以实现解调。信号输入 TETRA Demod 模块后会首先经过 Frequency Xlating FIR Filter，这可以先将我们感兴趣的频率分量

置于中心，然后滤除其它频率分量并下采样以将信号带宽降低到合适值并尽量减少后续处理消耗的计算资源。接着，Fractional resampler 对信号的采样率进行了调整并将调整后的信号在送到 channel 管脚输出的同时送到了 Feed Forward AGC 以让接下来待处理的信号有一个合适的幅值。由于 TETRA 的信息传递是通过信号的相位变化而非频率变化实现的，因此我们要先以这个信号作为 FLL Band-Edge 的输入参考信号使得输出信号的频率被锁定。为了最大化利用信号的功率并减少传输的字符间的干扰，我们希望在接收时能够在信号发射时的采样点处对信号进行采样，而这会要求接受端的时钟与发射端的同步。这在一般情况下是不可能的，并会使得我们最后解调出在星座图上散乱无章的采样点。为了解决这个问题，我们需要将信号输入 Polyphase Clock Sync 模块。最后，由于 TETRA 调制使用的 **D**QPSK 中携带信息的是两个相邻采样信号间的相位差，我们需要使用 Differential Phasor 以实现这一点。在此之后，Differential Phasor 输出的信号一边被输入了“constellation”输出管脚以绘制星座图、一边被输入了 Constellation Decoder 模块以根据我们定义的采样点与符号的映射规则将输入信号转换成符号。由于转换出的符号为 0, 1, 2, 3 这四个由 2 bit 刻画的数，我们使用 Unpack K Bits 并设置 K=2 以将输出数据便为 bit 流并送往 out 输出管脚。此后，我们在 `tetra_rx.grc` 中将 TETRA Demod 的 channel 管脚连接到另一个 QT GUI Frequency Sink 中以监视被解调的信号、将 constellation 输出的信号输入到 QT GUI Constellation Sink 中以通过星座图监视解调的信号质量、将 out 管脚输出的 bit 流通过 UDP Sink 使用用户数据报协议（UDP）传输出去。设置 UDP 端口为 42000 以方便后续的解码。

2.5 解码

我们使用 https://gitlab.com/larryth/tetra-kit.git 中的解码程序[1]对解调出的数据进行解码。在完成对 tetra-kit 的安装后，我们先使用 `sudo python3 tetra_rx.py` 运行接收与解调程序。保持程序运行，另开启一个 WSL 进程，进入 `tetra-kit/decoder` 目录后执行 `./decoder` 即可运行解码程序。这一解码程序可以通过 UDP 端口接收解调程序输出的 bit 数据、重建 TETRA 数据包并将其以 Json 文件的形式通过 42100 端口。

保持以上两个程序的运行，再开启一个 WSL 进程。进入 `tetra-kit/recorder` 目录后执行 `sudo ./recorder` 即可从 42100 端口接收 decoder 发送的 Json 文件（对于笔者的情况 `sudo` 是必须的，否则程序将无法打开 Json 文件）。Recorder 程序会根据 Json 文件记录我们所监听的信道的 caller id、subscriber identites 等信息。同时 recorder 会将音频文件以 `.raw` 的格式记录在该目录下的 `raw` 文件夹中，为了播放这些文件，我们需要先在目录 `tetra-kit/recorder/wav` 下执行 `./raw2wav.sh`，这会将 `.raw` 文件转换为 `.wav` 文件并保存在本目录中。这些 `.wav` 文件便是最终可供播放的音频文件。

3 讨论

我们在图1中展示了按照笔者方案构造的解码系统运行时的样子以供对此项目感兴趣的读者在复现时参考。虽然我们这里的方案的确能够实现对北京地铁 TETRA 信号的解调解码、监控信道并输出音频，但是它也存在可待改进之处：

[1]虽然这个 tetra-kit 项目里也有物理层的 GNURadio 解调程序，但是这一项目的开发者大量使用了基于较老版本 GNURadio 的 WX GUI 模块，而在我们安装的最新版本中这一模块已被 QT GUI 所取代。笔者在将这里面过时的模块进行替换后发现 tetra-kit 中提供的物理层程序虽能工作，但是使用体验远不如我们修改后的https://github.com/Tim---/tetra-toolkit.git。

- 程序的集成度不够高：tetra-kit 里的 decoder 和 recorder 程序是使用 C++ 编写的，我们使用 GNURadio Companion 生成的则是基于 Python 3 的解调程序，这三个程序几乎完全独立。这使得最终的系统略显臃肿。同时正如 tetra-kit 的原开发者所指出的，我们在三个程序间传递数据使用的 UDP 协议数据包大小被限制在了 2048 比特，这在由 decoder 向 recorder 传输 Json 数据时可能会造成一部分数据丢失。直观上 decoder 和 recorder 里的 C++ 程序都应该能改写为 Python 程序并整合到解调程序中。

- 无法实现实时的音频监控，同时现行系统解调出的音频信噪比较低并且会生成众多时长极短的无效音频文件。

信噪比的问题或许可以通过选用更优质的天馈系统得到解决；而解决其余问题需要对 tetra-kit 中的程序进行重构，这在目前大大超出了笔者的能力范围。笔者计划在业余时间学习更多关于通信系统的知识并在理解了 TETRA 编码的技术文档后再考虑对现有系统进行这些优化工作。

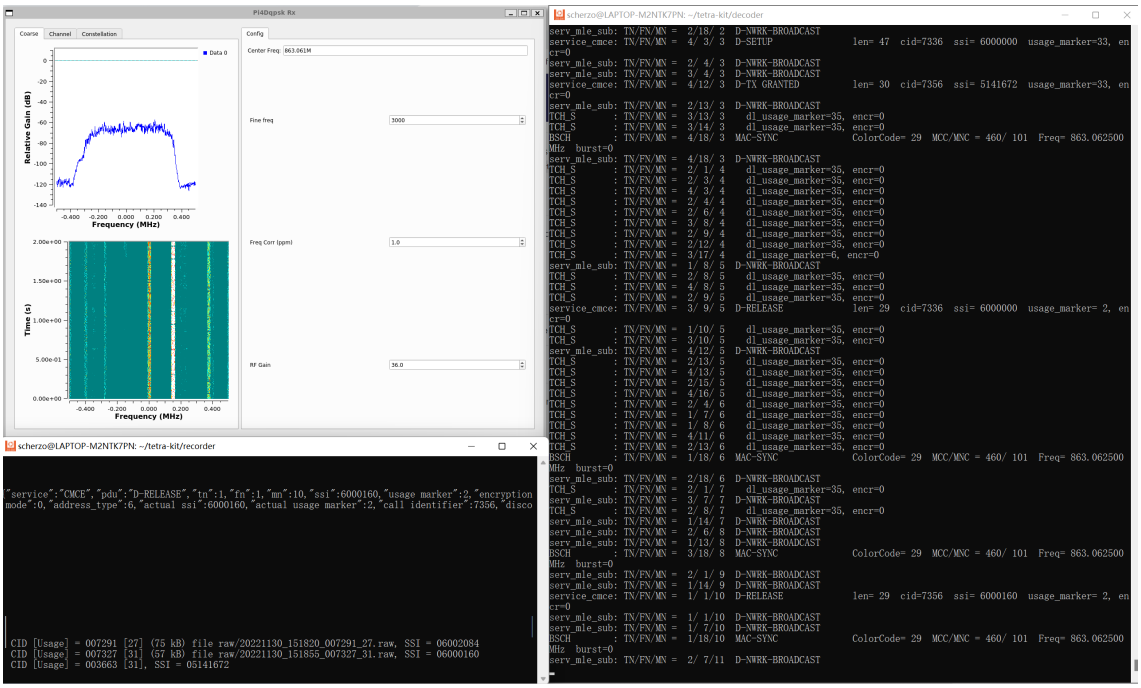

图 1. 解码系统运行时的情况