Frequency Modulation and Demodulation

1st Aidan Sharpe
Electrical and Computer Engineering
Rowan University
Glassboro, United States
sharpe23@students.rowan.edu

2nd Elise Heim
Electrical and Computer Engineering
Rowan University
Glassboro, United States
heimel27@students.rowan.edu

3rd John Leahy
Electrical and Computer Engineering
Rowan University
Glassboro, United States
leahyj92@students.rowan.edu

I. Introduction

No matter your location on Earth, as long as NOAA 15, 18, or 19 are in orbit, you will be able to receive their transmissions multiple times everyday. These weather satellites are in polar orbits around the Earth and constantly broadcast a real-time picture feed of the atmosphere to receivers below. Our research focused on how these pictures are broadcast, how they may be received, and how to reconstruct the images.

NOAA 15, 18, and 19 are part of a series of weather satellites used for atmospheric imaging. NOAA 15, also called NOAA K, was launched on May 13th, 1998, and became operational on December 12th, 1998. NOAA 18, also called NOAA N, started its mission in 2005 . NOAA 19, also called NOAA N' (read N-prime), became operational in 2009, and is intended to replace NOAA 18 sometime in the future.

II. Polar Orbiting Satellites

Polar orbits are a type of satellite orbit, where the satellite travels over Earth's poles, typically at an inclination at or close to 90°. This orbit allows for the satellite to pass over nearly every point of the Earth's surface as the planet rotates underneath. In the context of satellite communication, polar orbits are important because of their ability to provide global coverage, making them suitable for applications such as Earth observation, weather monitoring, and reconnaissance. Satellites in polar orbit can capture data over regions that are challenging to access with other orbital configurations, such as remote polar areas. In addition, their predictable coverage patterns make them particularly valuable for environmental monitoring, disaster management, and scientific research, while ensuring consistent and comprehensive data collection.

III. NOAA POES

The National Oceanic and Atmospheric Administration (NOAA) has many environmental satellites that it uses to monitor the Earth's environment, weather, and climate. One such type is the Geostationary Operational Environmental Satellites (GOES). The GOES satellites, as their name suggests, are stationary satellites relative to Earth's surface. This is because they move in sync with the rotation of the Earth, so their position remains in place over fixed points on the planet's surface. GOES delivers rapid, high-resolution, color imagery of the Earth's atmosphere. Polar Operational Environmental Satellites (POES), on the other hand, operate in polar orbits. These offer the advantage of global coverage daily, because the satellites orbit multiple times a day, every day. This makes POES great for environmental monitoring applications including weather analysis and forecasting, climate research and prediction, forest fire detection, global vegetation analysis, search and rescue, and many more applications. [4]

In our case, the GOES satellites are sending too much data for us to receive with our inexpensive setup. However, the POES series transmits greyscale images, which is much less data than color images. This is a huge benefit for us, which makes POES satellites more attractive for us to listen to.

IV. Automatic Picture Transmission

The POES satellites broadcast images using the Automatic Picture Transmission (APT) system. Our best resource for the APT system was the NOAA KLM User's Guide [1]. This user guide is official NOAA documentation for the NOAA KLM satellites as well as NOAA N and N'.

The data pipeline for the APT system is as follows. First, the Advanced Very High Resolution Radiometer (AVHRR) sends a ten-bit word to the digital processor. The eight most significant bits of the processed data are then passed through a digital-to-analog converter and filtered. Then, the data is modulated onto the 2400 Hz sub-carrier using amplitude modulation. Finally, the signal is modulated for a second time, this time using frequency modulation, at the satellite's broadcast frequency [1].

The image is broadcast pixel-by-pixel in lines of 2080 pixels. Two of the AVHRR sensor channels are broadcast at any given time, and 909 pixels per line are allotted to each channel. Lines of pixels are broadcast at a rate of two lines per second, for a total of 4160 words per second [1].

It is important to note that not all data transmitted is video data, as 909 pixels per channel times two channels is less than the total 2080 pixels per line. In addition to the 909 words of video data, each channel is also allotted 39 words of sync data, 47 words of space and minute marker data, and 45 words of telemetry data [1].

V. Downlink Setup

Our receiver, just like any communication link, has multiple layers of communication. Our setup included a dipole antenna, a 50Ω RF coaxial cable, an RTL-SDR v3, and several software tools.

A. Antenna

Choosing an antenna was one of the more difficult design decisions we had to make. For our initial tests, we used a simple dipole setup, but we also explored some other designs such as the double crossed dipole [3], and the quadrifilar helix (QFH) [5]. With our limited resources, however, these more advanced antenna designs proved to be too difficult to construct.

The satellites that we listened to broadcast between about 137 [MHz] and 138 [MHz]. The optimal length for our setup was a "quarter-wave" antenna, where each of the two elements is set to $\lambda/4$, or one quarter the wavelength. With the transmission frequency in mind, the optimal length can be found by

$$\frac{\lambda}{4} = \frac{c}{4f_c} \tag{1}$$

where c is the speed of light (3×10^8 [m/s]) and f_c is the carrier frequency. Using 137 [MHz] as the carrier frequency gives an optimal length of about 54.7 [cm].

Since the signal we are listening for is circularly polarized, if the dipole elements are along the same axis, the signal will be in a null for at least half of the time. This effect was minimized by setting the elements at 120° to each other as seen in figure 1.

B. Radio Receiver

The remaining portions of our physical layer were contained within the RTL-SDR, a simple, affordable software defined radio device. It has an SMA connector to interface with our 50Ω coaxial cable, and a USB connector to plug into a computer.

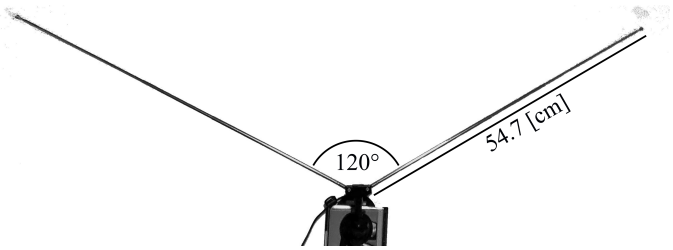

Fig. 1. Dipole antenna dimensions

C. Low-Level SDR Software

Before we could begin to write software for the RTL-SDR, we explored some pre-existing low-level tools. These included librtlsdr, a common driver for the device, and pyrtlsdr, a Python wrapper for the driver. Together, these tools helped us perform basic tasks such as setting the center frequency, sample rate, gain, and frequency correction, as well as reading complex samples from the tuner.

A more useful way to read samples is through the driver's support of asynchronous streaming. This technique allows for a constant stream of samples provided in chunks of a given size. For our tests, we used sizes of 512, 1024, and 65536. It is important that the size of the chunk is a power of two, as it drastically speeds up the fast Fourier transform (FFT).

D. High-Level SDR Software

Our software contribution to this project involved taking the samples from the SDR, then re-creating and interpreting the message signal. Since APT modulates its message using both AM and FM, we wrote demodulation code for both.

We utilized SDRangel, which is an open-source software designed for Software Defined Radio (SDR). It also includes a satellite tracker, which was helpful for finding the locations of the POES satellites. Most importantly, it includes a pre-made demodulator and display for APT signals. We were able to utilize this feature to test that our physical setup worked, without having to debug any of our own code.

VI. Demodulating FM

The first step to recovering the images from the satellites is to demodulate the FM signals they broadcast. If we had to wait for one of the three satellites to pass every time we wanted to test or debug our code, the development process would be slowed to a crawl. Therefore, we focused our efforts on recovering the message from another FM source on a frequency band no too far away from that of the satellites': broadcast FM radio.

The signal emmitted by an FM transmitter is typically of the form

$$s(t) = A_c \cos(2\pi f_c t + \beta_f m(t)) \tag{2}$$

where A_c is the transmitted amplitude, f_c is the carrier frequency, β_f is the frequency modulation index, and $m(t)$ is the original message signal.

When we receive the signal from the RTL-SDR, we are given a series of IQ samples [2]. To recover the message we used the demodulation function seen in listing 1. To demodulate an FM signal, we must know the instantaneous frequency. Our demodulation works by taking the product of the current sample and the complex conjugate of the next sample. The current sample can be written as $e^{j2\pi(fn+\phi)}$, and the conjugate of the next sample can be written as $e^{-j2\pi(f(n-1)+\phi)}$. The product will be $e^{j2\pi f}$, and we are left with f when taking the angle of this phasor [2].

While the broadcast FM does not transmit images, it is a constant source of FM transmission. We determined that the best way to know if our setup worked was if we could listen to the radio in real time. Doing so required one more tool, a library to play the demodulated message using computer's audio device. We opted to use the sounddevice library for Python due to our prior experience with it, and its simple, yet comprehensive functionality. Specifically, the sounddevice library allowed us to send a constant stream of samples to the audio device. This functionality proved to be critical for real time audio playback.

Our broadcast FM demodulation pipeline worked as follows. First, an asynchronous sample stream would be set up to read 512 samples at a time. Then, we passed the samples to a demodulation function, which would return the samples of

the message and a time axis for the message samples. Finally, we normalized the message for sound card compatibility and added the message samples to the audio output stream. By running both streams simultaneously, we were able to receive, process, and play back broadcast FM in real time.

VII. Results & Discussion

Our broadcast FM demodulator worked wonderfully. Just by changing the center frequency of the SDR (a single line of code), we were able to tune to a wide array of stations. Unfortunately, we also discovered a flaw with our setup. It was possible for adjacent stations with a stronger signal to overpower the signal from the station to which we were tuned. The best way to fix this is by applying a bandpass filter to the incoming signal, we would eliminate other stations.

Additionally, when testing alternative antennas, we used the broadcast FM demodulator to get a rough idea for where the nulls of our antennas were. This process helped us eliminate some designs.

Getting the code to work for the broadcast FM demodulator was tricky, getting a real-time image receiver to work was even harder. No longer were we able to test our code whenever we wanted to. Instead, we would have to wait for a satellite transit. This obstacle made further development very difficult. To maximize the number of debugging attempts, we utilized a satellite tracker tool that provided us with a ten-day outlook of transit predictions.

To get an initial idea of how well our physical setup would work, we used a the SDRangel software during a transit on a clear night. We were able to recover the image seen in figure 2. At first glance, the image is very noisy and one could easily be convinced that there was no image at all. There do appear to be some Earthly formations in the darker regions of image on the right.

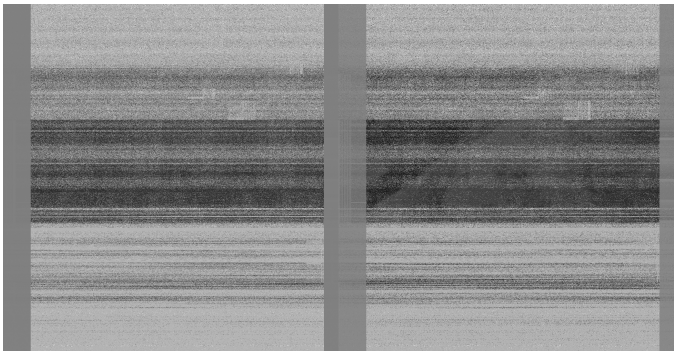

Fig. 2. NOAA 15

Compared to the image received duirng a transit of NOAA 18 on an overcast day, seen in figure 3, the NOAA 15 image appears quite clear.

We believe that our reduced signal quality during this NOAA 18 transit is due to several environmental factors including the overcast weather and our view of the satellite being obstructed by buildings. Additionally, the NOAA 18 satellite transmits at a very precise 137.9125 [MHz], which is not easily tuned to using the SDRangel software.

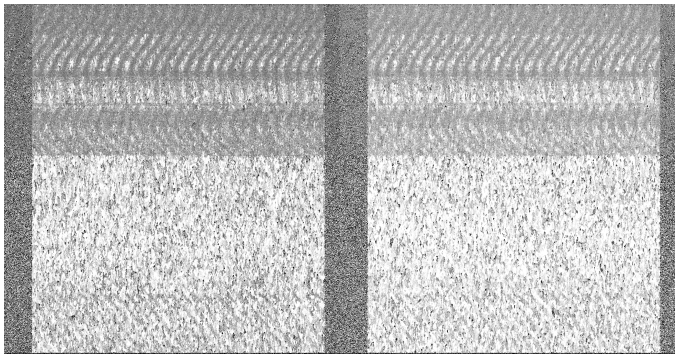

Fig. 3. NOAA 18

To better improve signal quality for future tests we aim to do several things. First, we would like to invest in an antenna better suited for receiving circularly polarized signals. Second, we would like to look into filters such as surface acoustic wave (SAW) filters that are purpose-built for the 137[MHz] range. With these tools, we would likely be able to improve the quality of our received images.

VIII. Conclusion

After extensive research on polar orbiting satellites, NOAA's POES series, and automatic picture transmission, we were ready to try out our own experiment. We made substantial progress on our program that decodes APT signals from satellites, as well as building a physical setup to receive these signals. We tested our setup using SDRangel, and confirmed that our setup was able to function as intended. However, after successfully receiving a POES signal from NOAA 15, we gained more insight into what would make our setup better suited for APT. Firstly, the small, cheap dipole antenna is good for receiving linearly polarized signals, such as FM radio stations. However, the signals from POES are circularly polarized, so we would require a circularly polarized antenna to best receive them. In addition to a new antenna, better amplifiers and filters would allow for the signals to be more isolated when we received them. We also learned that decoding the POES signals is a bit difficult, especially when attempting to time-synchronize the transmission to our receiving end.

APPENDIX

Listing 1. Quadrature demodulation function

```
def quad_demod(samples):
    return 0.5*np.angle(samples[:-1] * np.conj(samples[1:]))
```

Listing 2. Broadcast FM demodulation and streaming

```
async def start(self):
    self.stream.start()

    async for samples in self.sdr.stream(self.num_samples):
        message = dsp.quad_demod(samples)
        message = dsp.lowpass(message, 16E3, self.sdr.sample_rate)
        message /= np.max(np.abs(message))

        time = len(message)/self.sdr.sample_rate
        num_samples = int(time*self.audio_sample_rate)
        message = sp.signal.resample(message, num_samples)

        message = message.astype(np.float32)
        self.stream.write(message)
```

REFERENCES

[1] Kathy Kidwell (NOAA) et al. *NOAA KLM User's Guide with NOAA-N, N Prime, and MetOp Supplements*. 2014. URL: https://www.star.nesdis.noaa.gov/mirs/documents/0.0_NOAA_KLM_Users_Guide.pdf.

[2] Dr. Marc Lichtman. *PySDR: A Guide to SDR and DSP using Python*. 2024. URL: https://pysdr.org/index.html#.

[3] Gerald Martes. "Double Cross — A NOAA Satellite Downlink Antenna". In: (2008). URL: https://www.qsl.net/py4zbz/DCA.pdf.

[4] *Polar Operational Environmental Satellites (POES) Overview*. URL: https://www.noaasis.noaa.gov/POLAR/poes_overview.html.

[5] Romi Wiryadinata et al. "Image Data Acquisition for NOAA 18 and NOAA 19 Weather Satellites Using QFH Antenna and RTL-SDR". In: *MATEC Web of Conferences*. 2018.