2022 CSE 490 W Final Project Report

Project Title: Multi-channel Walkie-Talkie Receiver

Author: Tom Lou

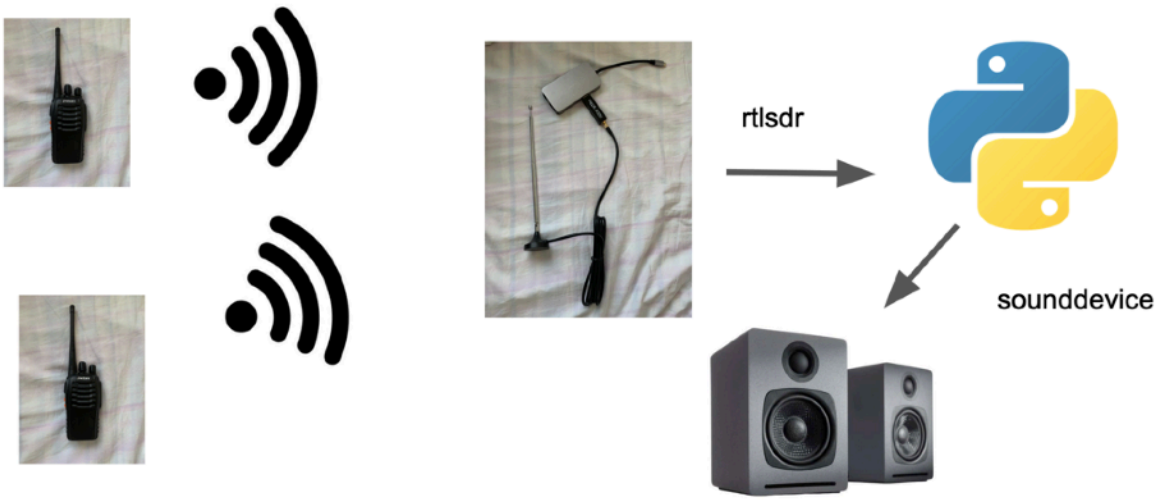
Date: 06/11/2022

Goal: What were you trying to do?

Sometime, in a large event, we might want to split people into multiple channels on the walkie-talkie. At the same time, the coordinator might want to monitor all of the channels. This is a common situation for stage managers, and there are some hardware solutions.

In this project, I tried to implement such a functionality with Software Defined Radio (SDR). My software can monitor multiple channels at the same time on the Family Radio Service (FRS) channels, which is a range of unlicensed channels that walkie-talkies can use. I also added one improvement. If multiple channels have voice at the same time, I will record and delay the voice from some of the channels so that the voice won't overlap with each other.

Approach: How did you (attempt to) do it? This should be a detailed description of what you did. Include details such as description of all hardware, what frequency you were operating at, what modulation schemes were involved, etc.



Project Structure

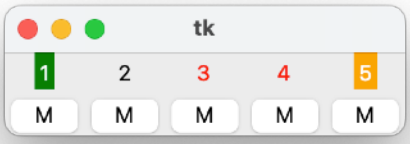
The basic project structure is above. My program receives the signal stream from the SDR by using the rtlsdr package and then play back the sound by using the sounddevice package.

The frequencies for FRS is shown below. In particular, I used 462.600 MHz (channel 16) and 462.675 MHz (channel 20) for testing. Each channel is 12 kHz wide using Frequency Modulation (FM).

List of FRS channels compared to GMRS [edit]

Channel	Frequency (MHz)	FRS EIRP Restriction	GMRS EIRP Restriction
1	462.5625	Up to 2 watt	Up to 5 watts
2	462.5875	Up to 2 watt	Up to 5 watts
3	462.6125	Up to 2 watt	Up to 5 watts
4	462.6375	Up to 2 watt	Up to 5 watts
5	462.6625	Up to 2 watt	Up to 5 watts
6	462.6875	Up to 2 watt	Up to 5 watts
7	462.7125	Up to 2 watt	Up to 5 watts
8	467.5625	Up to 0.5 watt	Up to 0.5 watt[Note 1]
9	467.5875	Up to 0.5 watt	Up to 0.5 watt[Note 1]
10	467.6125	Up to 0.5 watt	Up to 0.5 watt[Note 1]
11	467.6375	Up to 0.5 watt	Up to 0.5 watt[Note 1]
12	467.6625	Up to 0.5 watt	Up to 0.5 watt[Note 1]
13	467.6875	Up to 0.5 watt	Up to 0.5 watt[Note 1]
14	467.7125	Up to 0.5 watt	Up to 0.5 watt[Note 1]
15	462.5500	Up to 2 watt	Up to 50 watts
16	462.5750	Up to 2 watt	Up to 50 watts
17	462.6000	Up to 2 watt	Up to 50 watts
18	462.6250	Up to 2 watt	Up to 50 watts
19	462.6500	Up to 2 watt	Up to 50 watts
20	462.6750	Up to 2 watt	Up to 50 watts
21	462.7000	Up to 2 watt	Up to 50 watts
22	462.7250	Up to 2 watt	Up to 50 watts

https://en.wikipedia.org/wiki/Family_Radio_Service



Screenshot of My Program

I designed my UI as follows. Each channel has 4 types of state:

- Active: the audio is playing in the speaker. (Shown as green background)
- Waiting: the audio is delayed to avoid overlapping with the active channel. The sound will play back later. (Shown as yellow background)
- Nothing: there's no audio detected on this channel. (Shown as no background)
- Muted: the voice on this channel should be ignored. (Shown as red foreground)

Results: Did it work? What exactly did and didn't work?

Yes. I can successfully monitor multiple channels and the voice won't overlap.

Evaluation: How well did it work? Can you measure its performance?

On a Macbook Pro with M1 chip, my program can monitor 5 channels at the same time. However, if I add more channels, the program can't catch up with the speed and can't play the audio in real time. The CPU usage didn't reach 100%, so I guess the inefficiency is caused by IO or multi-processing communication.

Problems/solutions to highlight to future students working on similar projects: were there things you figured out (tips/tricks/hacks) that would be helpful for students working on similar things in the future? Were there problems you didn't anticipate and couldn't solve, that others should be warned about?

- I found that the program doesn't work if the buffer time is too long. It seems like MacOS will clip the audio buffer and only play the beginning of each buffer. This won't happen in Linux. I guess this is because MacOS has a small default maximum buffer size. Thus, I decreased the buffer time from 2.5 seconds to 0.1 second.
- When there's no signal on a channel, we will get random voice from the channel instead of silent voice. Thus, we should use the signal strength for silence detection.

Future work: what should be done in the future?

Do you have suggestions to future students or future you for additional cool project ideas that could extend what you've done here? Or, are there labs for future classes that could be based on this?

When two channels have a frequency that are too close to each other (e.g. channel 17, 462.600Mhz, and channel 3, 462.6125Mhz), my program will think that there are two channels speaking at the same time even if only one channel has signal. This can be improved if we can detect interference and only pick the voice from the strongest channel with similar frequencies.

Appendix: Detailed documentation

Please include an accessible link to your code, documentation, and other detailed information, or a pdf of your code etc.

My code is based on Kavel Rao's project (https://github.com/kavelrao/fm-streaming)
Here's the link to my code: https://github.com/LouYu2015/py-family-radio-service-station

Most of the logic is in the `next_audio_block` function. Basically, I first try to get the audio from the active channel first. If the channel is silent for a long time, then I will loop through all channels to find the next active channel.