CSIRO Summer Vacation Project Presentation Script – Faisal Umar

Title Slide:

Background and Purpose:

Previous work on the LOCUST module included 2 separate software modules for the detection of galactic hydrogen and local radio signals respectively. The newest project had a goal of creating sound out of the radio waves instead of graphs and plots like prior work. This was for the purpose of allowing students another method to participate in science classes and engage with Radio Astronomy as a gateway to STEM and increase the accessibility of this gateway into STEM.

Set-up:

Don’t worry too much about the wires, here’s the basics of the hardware I used. We start at the antenna (animation). Now you’re all working in a radio physics building so ill assume you know what it is, but for those who forgot it’s the instrument that converts EM into current. Next the current is passed to the amplifier here (animation) with a gain of <find out this number in the morning>. After the amplified signal is digitised here (animation) and fed into the raspberry Pi for processing via a USB interface. Final bit of hardware the processor, in this case a raspberry Pi (animation). This is where the digitised signal if processed you know your FFTs, domain transforms and so on, including the topic today: sonification. Before continuing I had to know a few hardware limits such as bandwidth and receiver range. The bandwidth turned out to be approx. 2 Mhz with a range from 25MHz till 1765 Mhz. Lastly the human audible frequency range I used the average adult range as children are typically able to pick up on more range than adults so should have no issues with a range from 20Hz to 20 kHz. At this point I noted down that I could divide the radio frequency by 1e6 (a million) to get sound frequencies audible to humans.

Result and Completed Features:

From the projects approximate goal, I did achieve the sonification of radio waves. Here’s a sample (play sample). The sample you just heard had a central frequency of 761 MHz, right on this peak here (triple animation) in the Optus range. The tone itself was produced by synthesising together the audio frequencies from each bin of sampled data the distribution of which can be seen here (histogram). This plot was made from 1 sample of data taken over a second, the audio just now is from multiple samples. The reason its impossible to differentiate any frequencies from the synthesised audio is due to the frequencies being less than a Hz apart. One last result is that due to the narrow band it is difficult if not impossible to audibly differentiate white noise from a wide band signal, which has been mitigated by raising amplitude values to a power before other transform function normalises then re-raises them. Also, while more than the 129 bins heard and seen here can be processed, they take too long to process.

Program design:

I initially abstracted away tasks in this planning phase and ended up with this control flow for the program. Start at appStart.py (animation) which opens up the connection to the radio to get data as well as handling user input and controls for how many bins, duration of each audio clip etc. Next it hands off control to sonifiWaves.py (animation) which will get the data samples before entering a while loop awaiting the user to press a designated quit key to change the central frequency (animation). However, while the user hasn’t quit the program will loop over getting samples and turning them into sound arrays then playing them. Todo this it asynchronously hands off control to pyGameAudioOutput.py (animation), which serves the purpose of returning sound objects as well as being the synthesiser (animation). To create the sound objects, it hands off control to freqArrayMaker.py (animation) which will return the scaled and transformed amplitude array and frequency bins array. Then control is returned to pyGameAudioOutput.py (animation) where it converts these values into pyGame sound objects. Lastly the sound is played by pyGameAudioOutput.py and control is returned to sonifiWaves.py so it can control the loop and exit if needed.

Data Processing:

Had to figure out a vague combination of libraries that would produce the amplitude and frequency arrays. So initially got an array of complex numbers from RTL-SDR function (animation), then fed that into matplotlibs’ PSD function (animation) to get binned frequencies and amplitudes. After that I could do the interpolation of the amplitudes (animation) and other transformations to get useful data for sonification. Learnt some better abstraction skills since I didn’t have to worry about this once it was completed, also learnt how to import libraries and their functions in python as well as giving things alternate names when importing them using the “as” keyword. I also learnt how to interpolate the amplitudes.

Sonification:

Initially had the system set to generate a second’s worth of audio for each frequency, however that was too slow. I realised I could speed it up by using the cyclic nature of sound waves and only computing a singular period of sound data rather than hundreds (pause), during the New Years break (animation). This was done by modifying the highlighted for loops end condition to be the seconds worth of samples divided by the frequency it was being calculated for to give a period worth of samples. This combined with some asynchronous programming allowed for the much faster and non-pausing playback of audio samples. The next challenge was getting the pygame mixer to synthesis these audios rather than playing just a singular one at a time. This required some reading however it turned out that the mixer had 2 types of channel widths, the first a mono vs stereo audio and the second was an amount of simultaneous audio clips (animation). Using this code here the amount of parallel audio clip channels was set to the amount of frequency bins and each was played for the duration set in appStart.py.

UI and Saving:

The UI for this ended up being a simple set of terminal print statements with instructions and some diagnostic information. Mostly during the processing, a series of statements showing the frequencies and amplitudes the radio waves were being sonified as. In the terminal the user can either input an integer central frequency to scan or scroll some pre-set options using the arrow keys. Also the amplitude and frequency bin arrays are saved to an output .txt file for record purposes in-case the user wishes to keep that data.

Future Work:

* Add base, had initially planned to have shakers that could be passed audio to create a tactile experience as well as the auditory one need to revisit and implement that once electronics issues are overcome (low voltage and/or current)
* Figure out how to get a histogram of amplitude being y-axis for frequency bins being x-axis to run without freezing the whole program and update in live time with sound data processing completing
* Speed up each sound loop after removing the click between each sample audio playing
* Add a functionality to save the generated audio clips
* Shift processing over to raspberry Pi’s onboard GPU to parallelise the workload and further increase speed
* Add Bluetooth functionality, assuming pyGame won’t work by default with raspberry Pi onboard Bluetooth module