Astro Sonification

Akshat Odiya

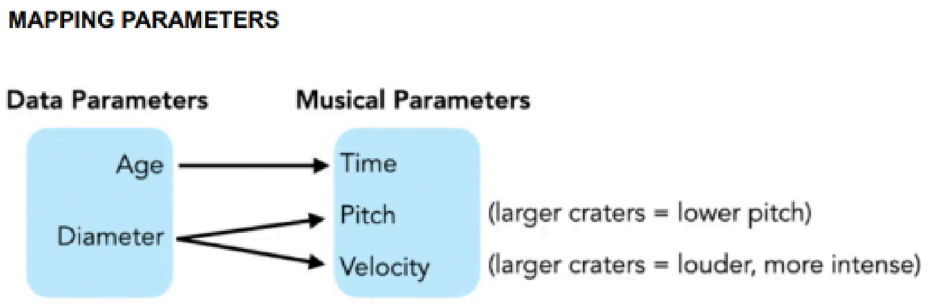
December 2024

1 Introduction

This report outlines a series of assignments focused on sonifying astronomical data, specifically the crashing of craters, light curves, and image sonification. Students will develop skills in transforming complex datasets into auditory experiences. This project enhances programming skills and demonstrates how data can be represented audibly.

2 Week 1

In this assignment, you will convert the provided dataset (lunar craters with their age and diameter) into sound by linking data points to sound properties such as pitch and volume. Using Python libraries such as `MIDITime` or `Pyo` in Jupyter Notebook or Google Colab, you will analyze the dataset, apply mapping parameters, and document your methodology. This will include creating a Jupyter notebook that outputs an audio file.



References:

- `https://www.youtube.com/watch?v=DUdLRy8i9qI&lc=UgzCz7WLdzWHg-2fesF4AaABAg`
- `https://medium.com/@astromattrusso/sonification-101-how-to-convert-data-into-mus`

3 Week 2

This assignment explores celestial data by converting light curves, which represent brightness variations over time, into sound using Python's `lightkurve` library. You will download light curves from the Kepler or TESS missions, analyze and smooth the data, map flux to sound frequencies, and interpret patterns such as transits or stellar variability.

1. **Data Acquisition:**
    - Download the Target Pixel File (TPF) of the exoplanet HAT-P-7 b using the `lightkurve` library (Kepler mission data).
2. **Light Curve Extraction and Processing:**
    - Extract the light curve from the TPF using the pipeline mask.
    - Flatten the light curve to remove long-term trends.
    - Fold the flattened light curve to observe periodic variations (orbital period: 3.5225 days).
    - Bin the folded light curve to reduce noise and enhance the signal.
3. **Normalization:**
    - Normalize time and flux values to a range of [0, 1] for consistent mapping to sound parameters.
4. **Sonification (Frequency Mapping):**
    - Map normalized flux values to frequencies (200 Hz to 1000 Hz).
    - Generate a sine wave for each time step and concatenate the waves to form a complete waveform.
    - Save the waveform as a `.wav` audio file (`lightcurve_sound.wav`).
5. **Sonification (Amplitude Modulation):**
    - Use a fixed frequency (440 Hz) and modulate amplitude based on normalized flux values.
    - Generate amplitude-scaled sine waves and save as a `.wav` file (`amplitude_modulated_sound.w`
6. **Audio Export:**
    - Convert the waveform to 16-bit audio and write it to `.wav` files using the `wave` module.

This process transforms the light curve into sound, enabling auditory exploration of celestial phenomena.

4 Week 3

The objective is to learn the basics of image sonification and extract pixel data from astronomical images. Student will explore the concepts and tools related to image sonification, such as sonoUno and Photosounder. Involves selecting a high-resolution galaxy image and using Python libraries like OpenCV or PIL to extract pixel intensity or RGB values, storing them in a structured format such as a 2D array or CSV file and visualizing the pixel data using Matplotlib to better understand its structure. This week lays the groundwork for developing a sonification tool.

5 Week 4

The objective is to map pixel data to sound properties like pitch, volume, and duration. Students will create Python functions to convert pixel data into sound parameters, such as mapping brightness to pitch or RGB values to sound effects. They will also generate short sound files using libraries like Pydub or Pygame to demonstrate their mappings.

5.1 Brightness based

This process converts an image into sound by mapping pixel brightness to audio frequencies. The following steps outline the methodology:

1. **Image Preprocessing:**
   - Load the image using OpenCV and convert it to the RGB color space for consistent pixel value interpretation.
   - Resize the image to 100×100 pixels to reduce processing time and maintain a manageable sound duration.
2. **Sound Initialization:**
   - Start with an empty, silent audio segment using the `pydub` library.
3. **Block-Based Pixel Processing:**
   - Split the image into smaller blocks to reduce resolution further:
     - Use a vertical step size (`row_step`) based on the maximum duration and tone length to ensure uniform coverage of image rows.
     - Process horizontal blocks of a fixed size (`block_size`) across each row.
   - For each block:
     - Compute the average RGB values of the pixels.
     - Calculate brightness as the mean of the RGB values.
     - Map brightness to a sound frequency between 200 Hz and 2000 Hz (linearly scaled).
4. **Sound Generation:**
   - Generate a sine wave tone for each block using the computed frequency. The tone duration is fixed at 50 milliseconds.
   - Concatenate the tone to the overall sound sequence.
5. **Duration Limit:**
   - Stop processing when the cumulative sound length exceeds the defined `max_duration` (15 seconds).
   - Trim the final sound to ensure it does not exceed this duration.
6. **Sound Export:**
   - Save the generated audio as a `.wav` file for playback and further analysis.

This approach sonifies the visual brightness and structure of an image, enabling auditory representation of visual data.

5.2 Pixel values based

This methodology maps the raw Red (R), Green (G), and Blue (B) pixel values from an image to sound parameters: frequency, volume, and duration. The steps are as follows:

1. **Image Preprocessing:**
   - Load the image using OpenCV and convert it to the RGB color space.
   - Resize the image to 100×100 pixels to reduce processing time and sound duration.
2. **Sound Initialization:**
   - Start with a silent audio segment using the `pydub` library.
3. **Pixel-Based Sound Mapping:**
   - Process the image row by row in steps of `row_step`, calculated to fit the maximum duration (`max_duration`) within the 50 ms tone duration.
   - For each pixel or block:
     - Extract the Red (R), Green (G), and Blue (B) components of the pixel.
     - Map the Red (R) value to frequency:

       $$\text{Frequency} = \text{base\_frequency} + \frac{R}{255} \times (\text{max\_frequency} - \text{base\_frequency})$$

       where `base_frequency` is 200 Hz, and `max_frequency` is 2000 Hz.
     - Map the Green (G) value to volume:

       $$\text{Volume (dB)} = \left(\frac{G}{255} \times 10\right) - 5$$

       resulting in a volume range of $[-5\,\text{dB}, 5\,\text{dB}]$.
     - Map the Blue (B) value to duration:

       $$\text{Tone Duration (ms)} = 20 + \frac{B}{255} \times 80$$

       producing durations between 20 and 100 milliseconds.
   - Generate a sine wave tone with the mapped frequency, duration, and volume using `pydub`.
   - Concatenate the generated tone to the overall audio sequence.
4. **Duration Constraint:**
   - Stop processing when the cumulative sound duration exceeds `max_duration` (15 seconds).
   - Trim the final audio segment to ensure it does not exceed this limit.
5. **Sound Export:**

- Save the generated audio as a `.wav` file (`image_sound_raw_rgb.wav`) for playback and analysis.

This methodology translates pixel-level RGB data into sound, enabling auditory exploration of visual information.
References:

- `https://jythonmusic.me/sonifying-images/`

6 Week 5

The objective is to integrate previous work to create a complete tool for sonifying astronomical images. The student will combine their code from the past two weeks into a single application that allows users to upload an image and select from at least two sonification modes, such as brightness-based pitch modulation and color-based sound effects. Additionally, developing a simple command-line interface (CLI) for user convenience, ensuring the tool outputs a complete sound file based on the uploaded image. This can be done by combining the two functions that were written in week 4 assignment and use according to the input given by the user.