**Proposal and Implementation Plan for Final Project, “Implementing a pitch recognition and note segmentation system for singing transcription”**

**Problem description**

1. Singing transcription – the conversion of sung audio input into digitally stored musical information, e.g. a MIDI file – is a well-known problem in audio processing.
2. Singing transcription systems have many applications. For example:
   1. Humming or singing to query a database of music and select a song
   2. Computer-aided music composition, in which a musician can sing a melody instead of manually inputting music notation
   3. Computer-aided music learning, in which a system could take audio input and guide an inexperienced musician to produce more correct pitches.
3. Generally speaking, there are two subproblems:
   1. How to extract pitch from singing input
      1. This is actually a very hard audio processing problem and I am not confident in my ability to do it from scratch in the time I have
      2. I will use existing pitch extraction libraries for the first implementation (Praat, Crepe)
      3. If I have time, I will implement the auditory model described in the paper for educational value and compare my results with the premade libraries
   2. How to recognize boundaries between notes, i.e. “note segmentation” – various different methods, parallels to Miniproject 1
4. Many approaches currently exist, although there is no standard methodology currently. These include:
   1. Handcrafted pitch-based and audio-based separation of notes
   2. Hidden Markov models to detect note events
   3. Probabilistic approach
5. For this project, I will be implementing the approach described in Mulder et al., “Recent improvements of an auditory model based front-end for the transcription of vocal queries” ([link](https://ieeexplore.ieee.org/abstract/document/1326812/footnotes#footnotes)).
   1. I chose this approach because review articles indicated that it had the highest accuracy among state of the art singing transcription systems, and it seems simple to implement relative to others.
   2. This satisfies expectations for one of the final project options in the assignment handout, “a single mode recognizer for a modality not explored in the miniprojects”

**Deliverable**

A program, meeting expectations as follows

* V1. A command line program that takes as input a single audio file, and outputs a list of notes with musical pitches (e.g. A5, C#3) and timestamps of their starts and ends.
* V2. The above, but with the following features, added in order of priority
  + A simple graphical user interface
  + Ability to visualize the notes in a human readable format
  + Ability to save and export a MIDI track
  + Integrated audio recorder so the user can record a clip and immediately transcribe it without switching between programs
* V3. The above, but also allow the user to sing while watching transcription occur in real time

And a report evaluating performance, as described below.

**Evaluation**:

Using a wider dataset of sung recordings, measure rates of note error. The main types of error would be:

* + 1. Notes deleted
    2. Notes inserted
    3. Notes moved (same number but segmented at wrong place)
    4. Pitch recognition error (a note was recorded but at the wrong pitch)

The data and error measures are taken from this review article comparing three singing transcription systems. The dataset they use is publicly available and quite large – I am using only the set of recordings that contains singing with no lyrics (there is also singing with lyrics, humming, whistling, etc.): <https://riuma.uma.es/xmlui/bitstream/handle/10630/8372/298_Paper.pdf?sequence=1>

**Detailed plan**

1. ~~Create a simple “V1” dataset consisting of two 10 second clips of myself singing, no lyrics.~~ 
   1. ~~Also download the dataset from the Molina article~~
   2. ~~Manually write down the correct note data for these clips by ear~~
      1. A3, B3, C#4, D4, E4
   3. ~~Load files onto computer~~
2. ~~Download Crepe and Praat with Python wrapper, figure out how to use it/whether you need voicing evidence data~~
3. ~~Write code to import a single audio file and output pitch data using the existing libraries~~
4. ~~Write code to generate data produced by the auditory model: for each frame generate a vector containing auditory spectrum, voiced/unvoiced decision, voicing evidence and pitch~~
5. ~~Implement the segmentation algorithm (taken from paper)~~
   1. ~~Primarily uses loudness data, but has to be able to distinguish between note boundaries and loudness fluctuations that are not note boundaries~~
   2. ~~The searching algorithm described in the paper is:~~
      1. ~~Go from left to right, searching for minimums and maximums~~
      2. ~~To search for a maximum: keep track of the largest loudness found so far (the “potential maximum”), move towards the right and if the current loudness is sufficiently lower, store the position and loudness of the potential maximum as an actual maximum. Then switch to looking for a minimum.~~
      3. ~~To search for a minimum: keep track of the position and value of smallest loudness (potential minimum), and consider a minimum found as soon as the current loudness is sufficiently greater than the potential minimum. When a minimum is found, generate a new note segment, then store the current frame as a potential maximum and switch to looking for a maximum~~
      4. ~~In order to detect white spaces:~~
         1. ~~When the loudness goes under some white space threshold for more than 2 successive frames, generate a note segment and stop the search until 2 successive frames above white-space threshold are found. At this moment, generate a white-space segment and switch to searching for a maximum.~~
6. Implement post processing
   1. Relabel low energy segments (breaths and noises) as white-spaces if the satisfying one or of the following conditions:
      1. Voicing level is too low
      2. Loudness too small
7. Detect pitches of segments (not the same as detecting pitch of voice data)
   1. Average over the frames of the segment
   2. How to cope with octave errors? go back to paper
8. Convert to scale notes for MIDI output