The Journal of Open Source Software

# pytch v2: A Real-Time Monitoring Tool For Polyphonic Singing Performances

**Sebastian Rosenzweig** [1*¶], **Marius Kriegerowski** [2*], **and Frank Scherbaum** [3]

**1** Independent Researcher, Barcelona, Spain **2** Independent Researcher, Berlin, Germany **3** University of Potsdam, Potsdam, Germany ¶ Corresponding author * These authors contributed equally.

**DOI:** 10.xxxxxx/draft

**Software**

- Review
- Repository
- Archive

**Editor:**

**Submitted:** 01 July 2025
**Published:** unpublished





## Summary

Polyphonic singing is one of the most widespread forms of music-making. During a performance, singers must constantly adjust their pitch to stay in tune with one another — a complex skill that requires extensive practice. Research has shown that pitch monitoring tools can assist singers in fine-tuning their intonation during a performance (Berglin et al., 2022). Specifically, real-time visualizations of the fundamental frequency (F0), which represents the pitch of the singing voice, help singers assess their pitch relative to a fixed reference or other voices. To support the monitoring of polyphonic singing performances, we developed `pytch`, an interactive Python tool with a graphical user interface (GUI) designed to record, process, and visualize multiple voices in real time. The GUI displays vocal spectra and estimated F0 trajectories for all singers, as well as the harmonic intervals between them. Additionally, users can adjust visual and algorithmic parameters interactively to accommodate different input devices, microphone signals, singing styles, and use cases. Written in Python, `pytch` utilizes the `libf0` library (Rosenzweig et al., 2022) for real-time F0 estimation and `pyqtgraph`[1] for efficient visualizations of the analysis results. Our tool builds upon a late-breaking demo in (Kriegerowski & Scherbaum, 2017), which we refer to as version 1. Since then, the tool has been significantly extended with a new real-time graphics engine, a modular audio processing backend that facilitates the integration of additional algorithms, and improved support for a wider range of platforms and recording hardware, which we refer to as version 2. Over its seven years of development, `pytch` has been tested and refined through use in several rehearsals, workshops, and field studies — including Sardinian quartet singing (see demo video[2]) and traditional Georgian singing (see demo video[3]).

## Statement of Need

Software that assesses the pitch of a singing voice in real time is best known from Karaoke singing applications, such as Let's Sing[4], Rock Band[5], or Cantamus[6]. These tools typically compare the singer's pitch to a score reference to judge whether notes are 'correct' or 'incorrect'. However, such applications face several limitations when applied to polyphonic or group singing contexts. Most notably, many Karaoke systems can only process one or two singing voices at a time, which is problematic for monitoring group performances. Additionally, software that

[1] https://www.pyqtgraph.org
[2] https://www.uni-potsdam.de/de/soundscapelab/computational-ethnomusicology/the-benefit-of-body-vibration-recordings/real-time-analysis-of-larynx-microphone-recordings
[3] https://youtu.be/LPt83Wqf2e4
[4] https://www.uni-potsdam.de/de/soundscapelab/computational-ethnomusicology/the-benefit-of-body-vibration-recordings/real-time-analysis-of-larynx-microphone-recordings
[5] https://youtu.be/LPt83Wqf2e4
[6] https://en.wikipedia.org/wiki/Let%27s_Sing

relies on a score as a reference poses challenges for a cappella performances, where singers may drift together in pitch over time while maintaining relative harmony, or in orally-transmitted traditions that may lack a formal score altogether. Finally, existing open-source research software for singing voice processing, like Praat (Boersma, 2001), Sonic Visualiser (Cannam et al., 2010), and Tarsos (Six et al., 2013), lack real-time feedback, preventing an effective feedback loop between singers and their tool.

To address these challenges, we developed pytch. Our tool is currently the only software that enables singers and conductors to monitor and train harmonic interval singing in real time — a skill that is essential in many vocal traditions. This includes not only polyphonic genres such as traditional Georgian vocal music (Scherbaum et al., 2019) or Barbershop singing (Hagerman & Sundberg, 1980), where precise tuning between voices is stylistically central, but also the practice of non-tempered tuning systems found in various oral traditions. In more detail, the vocal spectra can help singers fine-tune the expression of formant frequencies, while melodic and harmonic issues become visible through F0 trajectories and harmonic intervals. Unlike many existing tools, pytch does not require a musical score, making it well-suited for rehearsals, ethnomusicological research and pedagogical contexts focused on intonation and harmonic listening.

In addition to its practical applications, pytch also provides a flexible platform for music information retrieval (MIR) research on real-time audio processing. Working with real-time data introduces challenges such as a limited audio context for analysis and strict timing constraints to ensure low-latency processing. Researchers can use pytch to develop, test, and compare algorithms for F0 estimation and other music information retrieval tasks (Goto, 2004; Meier et al., 2024; Stefani & Turchet, 2022).

# Multitrack Singing Recordings

To fully leverage the capabilities of pytch, it is essential to record each singer with an individual microphone. While there is no hard limit on the number of input channels, we recommend recording up to four individual singers to ensure visibility of the charts and responsiveness of the GUI. Stereo recordings—-such as those captured by a room microphone placed in front of the ensemble–are not suitable for the analysis with pytch, because contributions of individual voices are difficult to identify from polyphonic mixtures (Cuesta, 2022). Suitable multitrack recordings can be obtained using handheld dynamic microphones or headset microphones. However, these setups are prone to cross-talk, especially when singers are positioned close together.

One way to reduce cross-talk is to increase the physical distance between singers or to record them in isolation. However, this is not always feasible, as singers need to hear one another to maintain accurate tuning. An effective workaround is the use of contact microphones, such as throat microphones, which capture vocal fold vibrations directly from the skin of the throat. This method offers a significant advantage: the recorded signals are largely immune to interference from other singers, resulting in much cleaner, more isolated recordings. Throat microphones have successfully been used to record vocal ensembles in several past studies (Scherbaum, 2016).

In addition to live monitoring, pytch can also be used to analyze pre-recorded multitrack singing performances. By playing back individual vocal tracks in a digital audio workstation (DAW) and using virtual audio routing tools such as Loopback[7] (macOS) or BlackHole[8], these tracks can be streamed into pytch as if they were live microphone inputs. This setup, which was also used in the demo video[9], allows users to benefit from pytch's real-time visualization and analysis features during evaluation of rehearsals, performances, or field recordings.

[7] https://rogueamoeba.com/loopback/
[8] https://existential.audio/blackhole/
[9] https://youtu.be/LPt83Wqf2e4

## Audio Processing

The real-time audio processing pipeline implemented in the file `audio.py` is the heart of `pytch` and consists of two main stages: recording and analysis. The recording stage captures multichannel audio waveforms from the soundcard or an external audio interface using the `sounddevice` library. The library is based on PortAudio and supports a wide range of operating systems, audio devices, and sampling rates. The recorded audio is received in chunks via a recording callback and fed into a ring buffer shared with the analysis process. When the buffer is sufficiently filled with audio chunks, the analysis process reads the recorded audio to compute several audio features.

For each channel, the analysis stage computes the audio level in dBFS, a time–frequency representation of the audio signal via the Short-Time Fourier Transform (see (Müller, 2021) for fundamentals of music processing), and an estimate of the F0 along with a confidence value, using the `libf0` library (Rosenzweig et al., 2022). The library includes several implementations of well-known F0 estimation algorithms. We make use of YIN (Cheveigné & Kawahara, 2002), which is a time-domain algorithm that computes the F0 based on a tweaked auto-correlation function. It is computationally efficient and well-suited for low-latency applications, but it tends to suffer from estimation errors, particularly confusions with higher harmonics such as the octave. The obtained F0 estimates, which are natively computed in the unit Hz, are converted to the unit cents using a user-specified reference frequency. Depending on the audio quality and vocal characteristics, F0 estimates may exhibit artifacts such as discontinuities or pitch slides, which can make the resulting trajectories difficult to interpret (Rosenzweig et al., 2019). Previous research has shown that using throat microphones can improve the isolation of individual voices in group singing contexts, resulting in cleaner signals and more accurate F0 estimates (Scherbaum, 2016). To further enhance interpretability, `pytch` includes several optional post-processing steps: a confidence threshold to discard estimates with low confidence score, a median filter to smooth the trajectories, and a gradient filter to suppress abrupt pitch slides. As a final step in the audio analysis, the harmonic intervals between the F0 trajectories are computed. Every audio feature is stored separately in a dedicated ring buffer. After processing, the pipeline sets a flag that notifies the GUI that new data is ready for visualization.

## Graphical User Interface (GUI)

In this section, we provide a step-by-step explanation of the `pytch` GUI implemented in the file `gui.py`. Right after the program start, a startup menu opens in which the user is asked to specify the soundcard, input channels, sampling rate, and window size for processing (see Figure Figure 1). Furthermore, the user can choose to store the recorded audio and the F0 trajectories on disk.

The Journal of Open Source Software

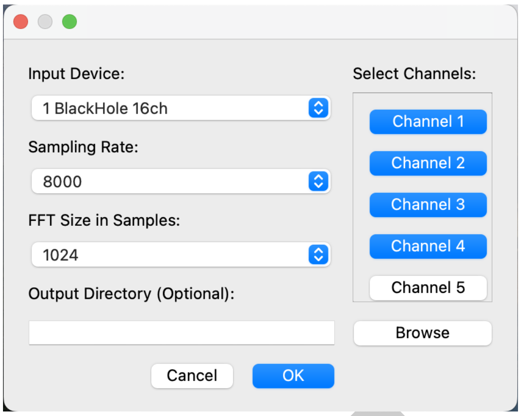


**Figure 1:** `pytch` startup menu.

These configuration choices are required to initialize the audio processing module and the main GUI, which is loaded when the user clicks "ok". A screenshot of the main GUI which opens after successful initialization is shown in Figure [Figure 2](#).

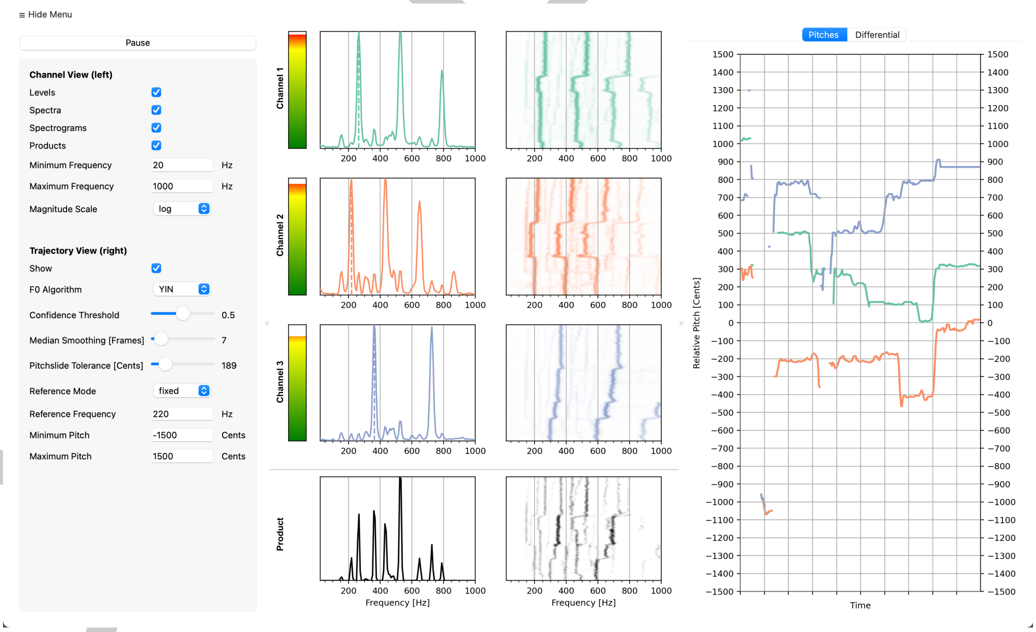


**Figure 2:** `pytch` GUI monitoring three singing voices.

The main GUI is organized into three horizontal sections. On the left, a control panel provides a start/stop button and allows users to adjust both the visual layout and algorithmic parameters. The central section displays "channel views"–one for each input channel–color-coded for clarity. Each view includes a microphone level meter, a real-time spectrum display with a vertical line marking the current F0 estimate, and a scrolling spectrogram with a 5 second time context. Channels are listed from top to bottom in the order they were selected during setup. Optionally, the bottommost view can display a product signal from all channels.

The right section, referred to as the "trajectory view," provides time-based visualizations of either the F0 trajectories ("pitches" tab) or the harmonic intervals between voices ("differential" tab) with a 10 second time context. Using the controls in the left-side menu, the user can select the F0 estimation algorithm and improve the real-time visualization by adjusting the confidence threshold, the median filter length for smoothing, and the tolerance of the gradient filter. F0 and interval trajectories can be displayed with respect to a fixed reference frequency or a dynamic one derived from a selected channel, the lowest, or highest detected voice. Axis limits for this section can also be manually set.


# Acknowledgements

We would like to thank Lukas Dietz for his help with the implementation, Peter Meier and Sebastian Strahl for the collaboration on real-time implementations, and all the singers who contributed to testing `pytch` during its development.


# References


Berglin, J., Pfordresher, P. Q., & Demorest, S. (2022). The effect of visual and auditory feedback on adult poor-pitch remediation. *Psychology of Music*, *50*(4), 1077–1090. https://doi.org/10.1177/03057356211026730

Boersma, P. (2001). Praat, a system for doing phonetics by computer. *Glot International*, *5*(9/10), 341–345.

Cannam, C., Landone, C., & Sandler, M. B. (2010). Sonic Visualiser: An open source application for viewing, analysing, and annotating music audio files. *Proceedings of the International Conference on Multimedia*, 1467–1468.

Cheveigné, A. de, & Kawahara, H. (2002). YIN, a fundamental frequency estimator for speech and music. *Journal of the Acoustical Society of America (JASA)*, *111*(4), 1917–1930.

Cuesta, H. (2022). *Data-driven pitch content description of choral singing recordings* [PhD thesis]. Universitat Pompeu Fabra, Barcelona, Spain.

Goto, M. (2004). A real-time music-scene-description system: Predominant-F0 estimation for detecting melody and bass lines in real-world audio signals. *Speech Communication*, *43*(4), 311–329.

Hagerman, B., & Sundberg, J. (1980). *Fundamental frequency adjustment in barbershop singing*. Citeseer.

Kriegerowski, M., & Scherbaum, F. (2017). Pytch - simultane mehrkanalige audioanalyse von gesangstimmen. *Late-Breaking Demos of the Workshop: Musik Trifft Informatik at 47. Jahrestagung Der Gesellschaft Für Informatik*.

Meier, P., Chiu, C.-Y., & Müller, M. (2024). A real-time beat tracking system with zero latency and enhanced controllability. *Transactions of the International Society for Music Information Retrieval (TISMIR)*, *7*(1), 213–227. https://doi.org/10.5334/tismir.189

Müller, M. (2021). *Fundamentals of music processing – using python and jupyter notebooks* (2nd ed., pp. 1–495) [Monograph]. Springer Verlag. https://doi.org/10.1007/978-3-030-69808-9

Rosenzweig, S., Scherbaum, F., & Müller, M. (2019). Detecting stable regions in frequency trajectories for tonal analysis of traditional Georgian vocal music. *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 352–359. https://doi.org/10.5281/zenodo.3527816

Rosenzweig, S., Schwär, S., & Müller, M. (2022). libf0: A python library for fundamental frequency estimation. *Late Breaking Demos of the International Society for Music Information Retrieval Conference (ISMIR)*.

Scherbaum, F. (2016). On the benefit of larynx-microphone field recordings for the documentation and analysis of polyphonic vocal music. *Proceedings of the International Workshop Folk Music Analysis*, 80–87.

Scherbaum, F., Mzhavanadze, N., Rosenzweig, S., & Müller, M. (2019). Multi-media recordings of traditional Georgian vocal music for computational analysis. *Proceedings of the International Workshop on Folk Music Analysis (FMA)*, 1–6.

Six, J., Cornelis, O., & Leman, M. (2013). Tarsos, a modular platform for precise pitch analysis of Western and non-Western music. *Journal of New Music Research*, *42*(2), 113–129. https://doi.org/10.1080/09298215.2013.797999

Stefani, D., & Turchet, L. (2022). On the challenges of embedded real-time music information retrieval. *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, *3*, 177–184.