FFTrack documentation

release 0.1.4

Team L2B1
Nicolas Schuldt, Jennifer Zahora,
Ismael Bouarfa, Nicolas Fontaine

Supervisors
Hugo Demaret, Ioana Ileana

May 03, 2024

CONTENTS

CHAPTER

ONE

FFTRACK PACKAGE

1.1 Package contents

DEVELOPMENT.md
Development guide.

docs
Documentation and related files.

fftrack
Program files.

INSTALLATION.md

README.md

requirements.txt

setup.py

tests
Folder containing the tests.

1.2 Modules

1.2.1 fftrack.config module

`fftrack.config.`**`load_config`**(*config_file_path: str*)

Load configuration data from a JSON file.

Args:
config_file_path (str): Path to the JSON config file.

Returns:
config_data (dict): Configuration data.

`fftrack.config.`**`save_config`**(*config_data: dict*, *config_file_path: str*)

Save configuration data to a JSON file.

Args:
config_data (dict): Configuration data to save.
config_file_path (str): Path to the JSON config file. If None, use the default path.

1.2.2 fftrack.main module

fftrack.main.**add_song**(*file_path*, *yt_link*)

Add a song to the database, either from a local file or by downloading it from YouTube.

Args:
song_path (str): Path to the audio file.
yt_link (str): YouTube link to download the song.

fftrack.main.**delete_song**(*song_id*)

Delete a song from the database.

Args:
song_id (int): ID of the song to delete.

fftrack.main.**get_config**()

Display the current configuration settings.

fftrack.main.**get_setting**(*setting: str*)

Display the value of a specific configuration setting.

fftrack.main.**identify**(*file_path: str*)

Identify an audio by giving its file path.

fftrack.main.**listen**()

Identify a song using a recorded sample, by the microphone of the device.

fftrack.main.**list_songs**()

Display all songs in the database.

fftrack.main.**menu**()

Displays the main menu options to the user, which are:

listen (to record a piece of audio),
identify (to find a match through an uploaded file),
add-song (to add the fingerprints of a file to the database).

1.3 Subpackages

1.3.1 fftrack.audio package

Contents

AudioProcessing
AudioReader
main_audio

Modules

fftrack.audio.audio_processing module

class fftrack.audio.audio_processing.**AudioProcessing**(*fs=44100*, *window_size=4096*, *overlap_ratio=0.5*, *fan_value=15*, *amp_min=10*, *peak_neighborhood_size=20*, *fingerprint_reduction=20*, *max_hash_time_delta=200*, *min_hash_time_delta=0*, *peak_sort=True*, *plot=False*)

Bases: `object`

Class for processing audio files and generating fingerprints.

Used by: Matching, main.py, Scripts/populate_database.py
Uses: hashlib, librose, logging, matplotlib.pyplot, numpy, scipy.nbimage, pydub

Args / constants of the module:
fs (int): Sampling rate.
window_size (int): Size of the FFT window.
overlap_ratio (float): Overlap ratio for FFT.
fan_value (int): Degree for pairing peaks in fingerprinting.
amp_min (int): Minimum amplitude in spectrogram for considering a peak.
peak_neighborhood_size (int): Size of the neighborhood around a peak.
fingerprint_reduction (int): Reduction in fingerprint to trim hash value size.
max_hash_time_delta (int): Max time delta between peaks in a hash.
min_hash_time_delta (int): Min time delta between peaks in a hash.
peak_sort (bool): Whether to sort peaks for hashing.

crop_samples(*samples*, *start_time*, *end_time*)

Crop audio samples to a specified time range.

Args:
samples (np.ndarray): Audio samples.
start_time (float): Start time in seconds.
end_time (float): End time in seconds.

Returns:
np.ndarray: Cropped audio samples.

find_peaks(*spectrogram_2d*)

Find peaks in the 2D array of the spectrogram.

Args:
spectrogram_2d (np.ndarray): 2D array of the spectrogram.

Returns:
list: List of peak indices in the format [(frequency, time), ...].

generate_fingerprints_from_file(*file_path*)

Full audio processing pipeline: load, generate spectrogram, find peaks, and generate hashes.

Args:
file_path (str): Path to the audio file.

Returns:
list: A list of hashes representing the audio fingerprint.

`generate_fingerprints_from_peaks`(*peaks*)

Generate hashes from the peaks.

Args:
peaks (list): Peaks in the format [(frequency, time), ...].

Returns:
hashes (list): A list of hashes representing the audio fingerprint.

`generate_fingerprints_from_samples`(*samples*)

Full audio processing pipeline: generate spectrogram, find peaks, and generate hashes.

Args:
samples (np.ndarray): Audio samples as a 1D numpy array.

Returns:
fingerprints (list): A list of hashes representing the audio fingerprint.

`generate_spectrogram`(*samples*)

Generate a spectrogram from the audio samples.

`load_audio_file`(*file_path*)

Load an audio file as a floating point time series.

Args:
file_path (str): Path to the audio file.

Returns:
samples (np.ndarray): Audio samples as a 1D numpy array.
fs (int): Sampling rate of the audio file.

`offset_to_seconds`(*offset*)

Transforms offset into seconds.

`static plot_peaks`(*peaks*)

Plot the peaks on the spectrogram.

fftrack.audio.audio_reader module

`class fftrack.audio.audio_reader.`**AudioReader**(*chunk=1024*, *frmt=8*, *channels=1*, *rate=44100*)

Bases: `object`

Handles the process of recording/retrieving an audio file and convert it into the right format (.wav)

Used by: main.py
Uses: pyaudio, wave, pydub, threading

Args / constants of the module:
chunk (int): Number of frames per buffer
format (pyaudio.paInt16): Sample format
channels (int): Number of channels (mono)
rate (int): Sampling rate

`audio_to_wav`(*filename*)

Convert an audio file into .wav format.

Args:
filename (str): The filename of the audio file to be converted.

record_audio()

Record an audio file from the user's microphone.

save_audio(*frames*)

Save the audio to the output_filename.

Args:
frames (list): List of audio frames.

start_recording()

Start recording audio.

stop_recording()

Stop the audio recording.

fftrack.audio.main_audio module

fftrack.audio.main_audio.**main**()

Demo on how to use the audio unit.

1.3.2 fftrack.database package

Contents

DatabaseManager
main_database
Models

Modules

fftrack.database.db_manager module

class fftrack.database.db_manager.**DatabaseManager**(*session=None*)

Bases: object

Handles interactions with the database, including adding and retrieving songs and fingerprints.

Used by: main.py, Matcher, populate_database.py
Uses: models, datetime, sqlalchemy

add_fingerprint(*song_id*, *hex_fingerprint*, *offset*)

Adds a new fingerprint to the database associated with a song.

Args:
song_id (int): The ID of the song the fingerprint belongs to.
hex_fingerprint (str): The fingerprint data as a 20-character hexadecimal string.
offset (int): The offset of the fingerprint within the song.

Returns:
bool: True if the fingerprint was added successfully, False otherwise.

add_song(*title*, *artist*, *album=None*, *release_date=None*, *youtube_link=None*)

Adds a new song to the database.

Args:
title (str): The title of the song.
artist (str): The artist of the song.
album (str, optional): The album of the song. Defaults to None.
release_date (str, optional): The release date of the song in 'YYYY-MM-DD' format. Defaults to None.
youtube_link (str, optional): A YouTube link of the song. Defaults to None.

Returns:
song_id (int): The ID of the newly added song, or None if an error occurred.

close_session()

Closes the database session.

delete_song(*song_id*)

Delete a song from the database.

Args:
song_id (int): ID of the song to delete.

get_all_songs()

Gets all songs from the database.

Returns:
list: A list of Song objects.

get_fingerprint_by_hash(*hex_fingerprint*)

Fetches fingerprints by their hash, returning offsets and song IDs.

Args:
hex_fingerprint (str): The 20-character hexadecimal hash of the fingerprint to search for.

Returns:
list of tuples: A list where each tuple contains (song_id, offset) for each matching fingerprint.

get_song_by_id(*song_id*)

Gets a song by its ID.

Args:
song_id (int): The ID of the song to retrieve.

Returns:
Song: The Song object if found, None otherwise.

get_song_by_title_artist(*title*, *artist*)

Gets a song by its title, and its artist.

Args:
title (string): The title of the song to retrieve.
artist (string): The artist of the song to retrieve.

Returns:
Song: The Song object if found, None otherwise.

reset_database()

Resets the database by dropping all tables and recreating them.

fftrack.database.main_db module

fftrack.database.main_db.**main**()

Demo on how to use the database manager.

fftrack.database.models module

class fftrack.database.models.**Fingerprint**(***kwargs*)

Bases: Base

Represents an audio fingerprint in the database.

Used by: DatabaseManager
Uses: sqlalchemy

fingerprint_id

A unique identifier for the fingerprint.

hash

The hash of one fingerprint of the song.

offset

The offset of one fingerprint of the song.

song_id

The ID of the song this fingerprint belongs to.

class fftrack.database.models.**Song**(***kwargs*)

Bases: Base

Represents a song in the database.

Used by: DatabaseManager
Uses: sqlalchemy

album

The album on which the song appears.

artist

The artist of the song.

release_date

The release date of the song.

song_id

A unique identifier for the song.

title

The title of the song.

youtube_link

The YouTube link of the song.

fftrack.database.models.**create_database**()

Creates the database tables based on the models.

1.3.3 fftrack.matching package

Contents

Modules

fftrack.matching.main_matching module

`fftrack.matching.main_matching.`**`database`**`()`

Setting up a mock database for the demo.

`fftrack.matching.main_matching.`**`main`**`()`

Demo on how to use the database manager.

fftrack.matching.matcher module

class `fftrack.matching.matcher.`**`Matcher`**(*database_manager* , *plot=False* , *top_n=5* , *top_list=0* , *confidence_calculator=1*, *confidence_threshold=0.5* , *match_count_benchmark=0*)

Bases: `object`

Matches the fingerprint of the query with the fingerprints of the database.

Used by: main.py
Uses: AudioProcessing, logging, defaultdic, Counter, matplotlib

Args / constants of the module:

db_manager (DatabaseManager): The database manager to access the database
plot (bool): Plot the results or not
top_n (int): Number of top matches to return
top_list (int): List of top matches is constructed by; 0: number of matches, 1: confidence level
confidence_calculator (int): Choose the confidence calculator; 0: dividing with the length of offsets, 1: dividing with the sum of all matches, 2: counting score confidence_threshold (int/float): Confidence threshold for a match; for calculator 0 and 1: <1, for 2: >1
match_count_benchmark (int): Minimum number of fingerprint matches needed to count as a match

`align_matches`(*matches*, *matches_per_song*)

Aligns the time difference of matches to find the most probable song match.

Args:

matches (list): List of matches in the format [(song_id, offset_difference), ...].
matches_per_song (dict): Dictionary of song and the number of hash matches.

Returns:

aligned_results (dict): A dictionary of aligned match results for each song.

`confidence_by_matches`(*aligned_results*, *sum_matches*)

Calculates how confident the algorithm is in the correctness of the match, and applies confidence_threshold.

Args:
aligned_results (dict): A dictionary of aligned match results for each song.
sum_matches (int): Sum of all the aligned matches.

Returns:
aligned_result: Updated results.

`confidence_by_score`(*aligned_results*, *matches_per_song*)

Calculates how confident the algorithm is in the correctness of the match, which is the sum of hash and offset matches in each song, and applies confidence_threshold.

Args:
aligned_results (dict): A dictionary of aligned match results for each song.
matches_per_song (dict): Dictionary of songs and the number of their hash matches.

Returns:
aligned_result: Updated results.

`find_best_match`(*top_matches*)

Returns the best match from the top matches.

Args:
top_matches (list): A list of tuples with the top matches.

Returns:
A tuple of the best matching song ID and its match details

`find_matches`(*sample_hashes*)

Find matches between sample hashes and the database.

Args:
sample_hashes (list): List of hashes in the format [(hash, offset), ...].

Returns:
possible_matches (list): A list of tuples of the match results, in the form of (song_id, offset_difference)
matches_per_song (dict): A dictionary of the song IDs, and the number of matches each song has

`find_top_n_matches`(*aligned_results* , *n*)

Find the top matches (max top n) from aligned results based on the highest count.

Args:
aligned_results (dict): A dictionary of aligned match results for each song.
n (int): Number of top matches to be returned.

Returns:
top_matches (list): A list of tuples of the best matching song IDs and their match details.

`get_best_match`(*sample_fingerprint*)

Matches the sample fingerprint with the database.

Args:
sample_fingerprint (list): List of hashes in the format [(hash, offset), ...].

Returns:
top_matches (list): List of the song IDs and their match details of the top 5 matches
best_match (tuple): A tuple of the best matching song ID and its match details.

1.3.4 fftrack.ui package

Contents

Modules

fftrack.ui.cli module

Contains the functions that display to and communicate with the user.

Uses: DatabaseManager, AudioReader, AudioProcessing, Matcher, typer, Console, Table, os, time
Used by: main.py

`fftrack.ui.cli.`**`display_best_match`**(*best_match*)

Display song metadata of the best match to the user.

Args:
best_match (tuple): A tuple with the best match in the following format: (song_ID, {matching_details}).

`fftrack.ui.cli.`**`display_top_matches`**(*top_matches*)

Display song metadata of the top matches to the user.

Args:
top_matches (list): A list of tuples of the top matching song IDs and their match details.

fftrack.ui.main_cli module

Demo of how the CLI is going to be used in the main

PYTHON MODULE INDEX

f