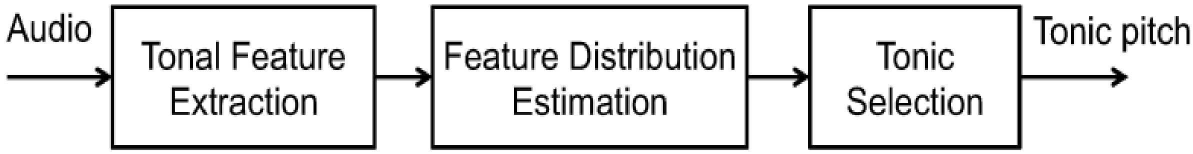
TONIC ESTIMATION



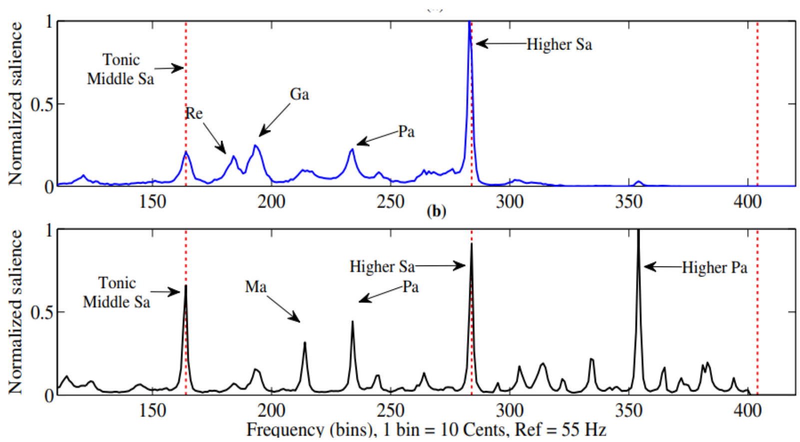
The features extracted in the first block are always pitch related. In the second block, an estimate of the distribution of these features is obtained using either Parzen window based density estimation or by constructing a histogram. Pitch-related features are extracted from the audio signal for further processing.

Method	Features	Feature Distribution	Tonic Selection
RS (Sengupta et al., 2005)	Pitch (Datta, 1996)	N/A	Error minimization
RH1/2 (Ranjani et al., 2011)	Pitch (Boersma & Weenink, 2001)	Parzen-window-based PDE[1]	GMM fitting
JS (Salamon et al., 2012)	Multi-pitch salience (Salamon, Gómez, & Bonada, 2011)	Multi-pitch histogram	Decision tree
SG (Gulati et al., 2012)	Multi-pitch salience (Salamon et al., 2011)	Multi-pitch histogram	Decision tree
	Predominant melody (Salamon & Gómez, 2012)	Pitch histogram	Decision tree
AB1 (Bellur et al., 2012)	Pitch (De Cheveigné & Kawahara, 2002)	GD[2] histogram	Highest peak
AB2 (Bellur et al., 2012)	Pitch (De Cheveigné & Kawahara, 2002)	GD histogram	Template matching
AB3 (Bellur et al., 2012)	Pitch (De Cheveigné & Kawahara, 2002)	GD histogram	Highest peak

Pipeline

1. Feature Extraction:

Pitch Extraction: Multiple methods, including Autocorrelation, AMDF, and Phase-Space Analysis, will be used to extract pitch information from the audio signal.
Predominant Melody Estimation: Different techniques, such as Salience-Based and Predominant F0 Estimation, will be employed to estimate the predominant melody.



2. Feature Distribution Estimation:

Parzen Window-Based Density Estimation: Probability density functions are estimated using a Parzen window function.
Histogram Construction: Feature values are binned, and a histogram of the feature values is created.

3. Tonic Selection:

Decision Trees: The feature space is split into branches based on feature values, and tonics are assigned to each leaf.
Template Matching: A template representing the typical frequency distribution is compared to the feature
distribution to identify the tonic pitch.

$$T(i) = \sum_{k=-\Delta}^{\Delta} G(i/2+k) + G(3i/4+k) + G(i) + G(3i/2+k) + G(2i+k)$$

G represent a vector with the magnitude of the candidate peaks at corresponding frequency values and zero in all other bins. I = frequency

Semi-Continuous GMM Fitting: This approach leverages the fixed means representing svar ratios and uses statistical inference through the EM algorithm to determine the weights and variances of the Gaussian components.

$$\begin{aligned}
\theta_1 &= \arg\min_{S_0(j)} \left\{ \frac{\sigma_{S_0}}{\alpha_{S_0}} \middle| S_0(j) \right\} \; ; j \in [1:J] \\
\theta_2 &= \arg\min_{S_0(j)} \left\{ \frac{\sigma_{S_0} + \sigma_{P_0} + \sigma_{S_+}}{\alpha_{S_0} + \alpha_{P_0} + \alpha_{S_+}} \middle| S_0(j) \right\} \; ; j \in [1:J]
\end{aligned}$$

Basic code (no result till now)

```
import numpy as np
from scipy.io import wavfile
from scipy import signal
from scipy.stats import norm
from scipy.signal import find_peaks
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

Feature Extraction

```
def extract_pitch(audio_signal, sampling_rate):
    # Pitch extraction using Autocorrelation
    autocorrelation = np.correlate(audio_signal, audio_signal, mode='full')
    pitch_period = np.argmax(autocorrelation[len(autocorrelation)//2:]) / sampling_rate
    pitch_values = 1 / pitch_period
    return pitch_values

def estimate_predominant_melody(audio_signal, sampling_rate):
    # Predominant Melody Estimation using spectral peaks
    _, _, Sxx = signal.spectrogram(audio_signal, fs=sampling_rate)
    peaks, _ = find_peaks(Sxx.max(axis=0), height=0.5)
    predominant_melody = peaks * (sampling_rate / len(Sxx))
    return predominant_melody
```

Feature Distribution Estimation

```
def parzen_window_density_estimation(features):
    # Parzen Window-Based Density Estimation
    return norm.pdf(features, loc=np.mean(features), scale=np.std(features))

def construct_histogram(features):
    # Histogram Construction
    histogram, _ = np.histogram(features, bins='auto', density=True)
    return histogram
```

Tonic Selection

```
def decision_tree_tonic_selection(features, tonics):
    # Decision Trees for Tonic Selection
    model = DecisionTreeClassifier()
    model.fit(features.reshape(-1, 1), tonics)
    return model

def template_matching_tonic_selection(feature_distribution, template):
    #Template Matching for Tonic Selection
    cross_corr = np.correlate(feature_distribution, template, mode='same')
    identified_tonic = np.argmax(cross_corr)
    return identified_tonic

file_path = 'audio_file.wav'
sampling_rate, audio_signal = wavfile.read(file_path)

# Feature Extraction
pitch_values = extract_pitch(audio_signal, sampling_rate)
predominant_melody = estimate_predominant_melody(audio_signal, sampling_rate)

# Feature Distribution Estimation
features = np.concatenate([pitch_values, predominant_melody], axis=0)
density_estimation = parzen_window_density_estimation(features)
histogram = construct_histogram(features)

# Tonic Selection
tonics = y (ground truth)
features_train, features_test, tonics_train, tonics_test = train_test_split(features, tonics,
test_size=0.2, random_state=42)

# Training Decision Tree model
model = decision_tree_tonic_selection(features_train, tonics_train)

# Prediction
predictions = model.predict(features_test.reshape(-1, 1))
```