Import Libraries and Setup

In [1]:
```
%pip install librosa
%pip install fastdtw
```

```
Requirement already satisfied: librosa in c:\users\sahaj\anaconda3\lib\site-packages (0.11.0)
Requirement already satisfied: audioread>=2.1.9 in c:\users\sahaj\anaconda3\lib\site-packages (from librosa) (3.1.0)
Requirement already satisfied: numba>=0.51.0 in c:\users\sahaj\anaconda3\lib\site-packages (from librosa) (0.60.0)
Requirement already satisfied: numpy>=1.22.3 in c:\users\sahaj\anaconda3\lib\site-packages (from librosa) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in c:\users\sahaj\anaconda3\lib\site-packages (from librosa) (1.13.1)
Requirement already satisfied: scikit-learn>=1.1.0 in c:\users\sahaj\anaconda3\lib\site-packages (from librosa) (1.5.1)
Requirement already satisfied: joblib>=1.0 in c:\users\sahaj\anaconda3\lib\site-packages (from librosa) (1.4.2)
Requirement already satisfied: decorator>=4.3.0 in c:\users\sahaj\anaconda3\lib\site-packages (from librosa) (5.1.1)
Requirement already satisfied: soundfile>=0.12.1 in c:\users\sahaj\anaconda3\lib\site-packages (from librosa) (0.13.1)
Requirement already satisfied: pooch>=1.1 in c:\users\sahaj\anaconda3\lib\site-packages (from librosa) (1.8.2)
Requirement already satisfied: soxr>=0.3.2 in c:\users\sahaj\anaconda3\lib\site-packages (from librosa) (1.0.0)
Requirement already satisfied: typing_extensions>=4.1.1 in c:\users\sahaj\anaconda3\lib\site-packages (from librosa) (4.15.0)
Requirement already satisfied: lazy_loader>=0.1 in c:\users\sahaj\anaconda3\lib\site-packages (from librosa) (0.4)
Requirement already satisfied: msgpack>=1.0 in c:\users\sahaj\anaconda3\lib\site-packages (from librosa) (1.0.3)
Requirement already satisfied: packaging in c:\users\sahaj\anaconda3\lib\site-packages (from lazy_loader>=0.1->librosa) (25.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in c:\users\sahaj\anaconda3\lib\site-packages (from numba>=0.51.0->li
brosa) (0.43.0)
Requirement already satisfied: platformdirs>=2.5.0 in c:\users\sahaj\anaconda3\lib\site-packages (from pooch>=1.1->librosa) (3.
10.0)
Requirement already satisfied: requests>=2.19.0 in c:\users\sahaj\anaconda3\lib\site-packages (from pooch>=1.1->librosa) (2.31.
0)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\sahaj\anaconda3\lib\site-packages (from scikit-learn>=1.1.0->li
brosa) (3.5.0)
Requirement already satisfied: cffi>=1.0 in c:\users\sahaj\anaconda3\lib\site-packages (from soundfile>=0.12.1->librosa) (1.17.
1)
Requirement already satisfied: pycparser in c:\users\sahaj\anaconda3\lib\site-packages (from cffi>=1.0->soundfile>=0.12.1->libr
osa) (2.21)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\sahaj\anaconda3\lib\site-packages (from requests>=2.19.0->p
ooch>=1.1->librosa) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\sahaj\anaconda3\lib\site-packages (from requests>=2.19.0->pooch>=1.1->l
ibrosa) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\sahaj\anaconda3\lib\site-packages (from requests>=2.19.0->pooch>=
1.1->librosa) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\sahaj\anaconda3\lib\site-packages (from requests>=2.19.0->pooch>=
1.1->librosa) (2025.10.5)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: fastdtw in c:\users\sahaj\anaconda3\lib\site-packages (0.3.4)
Requirement already satisfied: numpy in c:\users\sahaj\anaconda3\lib\site-packages (from fastdtw) (1.26.4)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import numpy as np
import librosa
import os
import json
from scipy import signal
from scipy.io import wavfile
from sklearn.mixture import GaussianMixture
from fastdtw import fastdtw
from scipy.spatial.distance import euclidean
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(42)
```

Part A: Data Loading & Preprocessing

Load Speaker Data

In [3]:
```
def load_speaker_data(speaker_id: str, data_path: str) -> list:
    """
        Loads audio files for a specific speaker from the dataset.
        Args:
            speaker_id (str): The ID of the speaker.
            data_path (str): Path to the dataset.
        Returns:
            list: A list of tuples (audio, sr).
    """
    speaker_dir = os.path.join(data_path, speaker_id)
    audio_data = []

    # Check if directory exists
    if not os.path.exists(speaker_dir):
        print(f"Directory not found: {speaker_dir}")
        return []

    for file in os.listdir(speaker_dir):
        if file.endswith(".wav") or file.endswith(".flac"):
            path = os.path.join(speaker_dir, file)
            audio, sr = librosa.load(path, sr=None)
            audio_data.append((audio, sr))

    return audio_data
```

Audio Preprocessing

Normalizes volume, applies pre-emphasis, and trims silence.

In [4]:
```
def preprocess_audio(audio: np.ndarray, sr: int) -> tuple:
    """
        Normalizes, pre-emphasizes, and trims silence from audio.
        Args:
            audio (np.ndarray): The audio signal.
            sr (int): Sample rate.
        Returns:
            tuple: (processed_audio, sr)
    """
    if sr != 16000:
        audio = librosa.resample(audio, orig_sr=sr, target_sr=16000)
        sr = 16000

    if np.max(np.abs(audio)) > 0:
        audio = audio / np.max(np.abs(audio))

    audio = np.append(audio[0], audio[1:] - 0.97 * audio[:-1])

    audio, _ = librosa.effects.trim(audio, top_db=20)

    return audio, sr
```

Compute Fundamental Frequency (F0) Statistics

In [5]:
```
def compute_f0_stats(audio: np.ndarray, sr: int) -> dict:
    """
        Computes basic statistics for the fundamental frequency.
        Args:
            audio (np.ndarray): Audio signal.
            sr (int): Sample rate.
        Returns:
            dict: Mean, std, min, and max of F0.
    """
    f0, _, _ = librosa.pyin(audio, fmin=50, fmax=500, sr=sr)
    f0 = f0[~np.isnan(f0)]

    return {
        "mean_f0": float(np.mean(f0)),
        "std_f0": float(np.std(f0)),
        "min_f0": float(np.min(f0)),
        "max_f0": float(np.max(f0))
    }
```

Compute RMS Energy

In [6]:
```
def compute_rms_energy(audio: np.ndarray) -> float:
    """
        Calculates RMS energy of the audio.
        Args:
            audio (np.ndarray): Audio signal.
        Returns:
            float: RMS energy value.
    """
    return float(np.sqrt(np.mean(audio ** 2)))
```

Part B: Feature Extraction

Extract F0 Contour

In [7]:
```
def extract_f0(audio: np.ndarray, sr: int) -> np.ndarray:
    """
        Extracts F0 contour using pYIN.
        Args:
            audio (np.ndarray): Audio signal.
            sr (int): Sample rate.
        Returns:
            np.ndarray: F0 contour (0 for unvoiced).
    """
    f0, _, _ = librosa.pyin(audio, fmin=50, fmax=500, sr=sr)
    return np.nan_to_num(f0)
```

Extract MFCCs

In [8]:
```
def extract_mfcc(audio: np.ndarray, sr: int, n_mfcc: int = 13) -> np.ndarray:
    """
        Extracts MFCC features.
        Args:
            audio (np.ndarray): Audio signal.
            sr (int): Sample rate.
            n_mfcc (int): Number of coefficients.
        Returns:
            np.ndarray: MFCC matrix.
    """
    mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=n_mfcc)
    return mfcc
```

Extract Formants (LPC-based)

In [9]:
```
def extract_formants(audio: np.ndarray, sr: int) -> np.ndarray:
    """
        Extracts first 3 formants using LPC.
        Args:
            audio (np.ndarray): Audio signal.
            sr (int): Sample rate.
        Returns:
            np.ndarray: Array of 3 formants.
    """
    audio = audio[:int(0.03 * sr)]  # 30 ms frame
    A = librosa.lpc(audio, order=12)
    roots = np.roots(A)
    roots = [r for r in roots if np.imag(r) >= 0]

    angles = np.angle(roots)
    freqs = angles * (sr / (2 * np.pi))
    freqs = np.sort(freqs)

    formants = freqs[:3] if len(freqs) >= 3 else [0, 0, 0]
    return np.array(formants)
```

Calculate Pitch Shift Ratio

In [10]:
```
def calculate_pitch_shift_ratio(source_f0: np.ndarray, target_f0: np.ndarray) -> float:
    """
        Calculates pitch shift ratio based on F0 means.
        Args:
            source_f0 (np.ndarray): Source F0 statistics.
            target_f0 (np.ndarray): Target F0 statistics.
        Returns:
            float: Pitch shift ratio.
    """
    source_mean = np.mean(source_f0[source_f0 > 0])
    target_mean = np.mean(target_f0[target_f0 > 0])

    ratio = target_mean / source_mean
    return float(np.clip(ratio, 0.5, 2.0))
```

Dynamic Time Warping (DTW) Alignment

In [11]:
```
def align_features_dtw(source_features: np.ndarray, target_features: np.ndarray) -> np.ndarray:
    """
        Aligns feature sequences using DTW.
        Args:
            source_features (np.ndarray): Source MFCCs.
            target_features (np.ndarray): Target MFCCs.
        Returns:
            np.ndarray: Alignment path.
    """
    _, path = fastdtw(source_features.T, target_features.T, dist=euclidean)
    return np.array(path)
```

Train GMM for Feature Mapping

In [12]:
```
def train_feature_mapping(source_features: np.ndarray, target_features: np.ndarray):
    """
        Trains a GMM for feature mapping.
        Args:
            source_features (np.ndarray): Source MFCCs.
            target_features (np.ndarray): Target MFCCs.
        Returns:
            GaussianMixture: Trained model.
    """
    # Align features using DTW
    print("Aligning features...")
    path = align_features_dtw(source_features, target_features)

    X = source_features[:, path[:, 0]]
    Y = target_features[:, path[:, 1]]

    # Transpose for GMM (n_samples, n_features)
    X = X.T
    Y = Y.T

    print(f"Training GMM with {X.shape[0]} aligned frames...")
    gmm = GaussianMixture(n_components=8, covariance_type='diag', random_state=42)
    gmm.fit(np.hstack([X, Y]))

    return gmm
```

Convert Features using GMM

In [13]:
```
def convert_features(model, source_features: np.ndarray) -> np.ndarray:
    """
        Converts features using the trained GMM.
        Args:
            model: Trained GMM.
            source_features (np.ndarray): Input features.
        Returns:
            np.ndarray: Converted features.
    """
    n_components = model.n_components
    n_features = source_features.shape[0]  # Should be 13

    # Split means into source (X) and target (Y) parts
    # The GMM was trained on [X, Y], so first half is X, second is Y
    means_x = model.means_[:, :n_features]
    means_y = model.means_[:, n_features:]

    # We also need covariances.
    # If covariance_type is 'diag', covariances_ is (n_components, n_features)
    # So we split it similarly.
    covs_x = model.covariances_[:, :n_features]
    covs_y = model.covariances_[:, n_features:]

    X = source_features.T
    converted = []

    for x_sample in X:
        # Let's simplify: Find the component k whose Source-Mean is closest to x_sample
        # This is a "Hard Clustering" approximation for conversion.
        dists = np.sum((means_x - x_sample)**2, axis=1)
        best_k = np.argmin(dists)

        # Mapping: y = mu_y_k + (sigma_yx_k * sigma_xx_k^-1) * (x - mu_x_k)
        # For 'diag' covariance, the cross-term sigma_yx is 0, so it simplifies to:
        # y = mu_y_k + (cov_y / cov_x)^0.5 * (x - mu_x_k)
        # But commonly in simple GMM VC, we just take the mean: y ~ mu_y_k

        converted.append(means_y[best_k])

    return np.array(converted).T
```

Part C: Voice Conversion

Pitch Shifting

In [14]:
```
def shift_pitch(audio: np.ndarray, sr: int, pitch_ratio: float) -> np.ndarray:
    """
        Shifts pitch of audio.
        Args:
            audio (np.ndarray): Audio signal.
            sr (int): Sample rate.
            pitch_ratio (float): Shift ratio.
        Returns:
            np.ndarray: Pitch-shifted audio.
    """
    n_steps = 12 * np.log2(pitch_ratio)
    shifted = librosa.effects.pitch_shift(y=audio, sr=sr, n_steps=n_steps)
    return shifted
```

Spectral Envelope Conversion

In [15]:
```python
def convert_spectral_envelope(audio: np.ndarray, sr: int, mapping_model) -> np.ndarray:
    """
        Converts spectral envelope.
        Args:
            audio (np.ndarray): Input audio.
            sr (int): Sample rate.
            mapping_model: GMM model.
        Returns:
            np.ndarray: Mel spectrogram.
    """
    mfcc = extract_mfcc(audio, sr)
    converted_mfcc = convert_features(mapping_model, mfcc)

    spectrogram = librosa.feature.inverse.mfcc_to_mel(converted_mfcc)
    return spectrogram
```

Voice Conversion Pipeline

In [16]:
```python
def voice_conversion_pipeline(source_audio: np.ndarray, sr: int, mapping_model, pitch_ratio: float) -> np.ndarray:
    """
        Feature-based voice conversion pipeline.
        Args:
            source_audio (np.ndarray): Input audio.
            sr (int): Sample rate.
            mapping_model: GMM model.
            pitch_ratio (float): Pitch shift ratio.
        Returns:
            np.ndarray: Converted audio.
    """
    pitched = shift_pitch(source_audio, sr, pitch_ratio)
    spectrogram = convert_spectral_envelope(pitched, sr, mapping_model)
    audio = librosa.feature.inverse.mel_to_audio(spectrogram, sr=sr)
    return audio
```

Part D: Evaluation

Mel-Cepstral Distortion (MCD)

In [17]:
```python
def calculate_mcd(converted_mfcc: np.ndarray, target_mfcc: np.ndarray) -> float:
    """
        Calculates Mel-Cepstral Distortion.
        Args:
            converted_mfcc (np.ndarray): Converted features.
            target_mfcc (np.ndarray): Target features.
        Returns:
            float: MCD value.
    """
    min_len = min(converted_mfcc.shape[1], target_mfcc.shape[1])
    diff = converted_mfcc[:, :min_len] - target_mfcc[:, :min_len]
    mcd = (10 / np.log(10)) * np.mean(np.sqrt(2 * np.sum(diff ** 2, axis=0)))
    return float(mcd)
```

F0 Correlation

In [18]:
```python
def calculate_f0_correlation(converted_f0: np.ndarray, target_f0: np.ndarray) -> float:
    """
        Calculates correlation between F0 contours.
        Args:
            converted_f0 (np.ndarray): Converted F0.
            target_f0 (np.ndarray): Target F0.
        Returns:
            float: Correlation coefficient.
    """
    # Truncate to min length
    min_len = min(len(converted_f0), len(target_f0))
    converted_f0 = converted_f0[:min_len]
    target_f0 = target_f0[:min_len]

    mask = (converted_f0 > 0) & (target_f0 > 0)
    if np.sum(mask) == 0:
        return 0.0
    return float(np.corrcoef(converted_f0[mask], target_f0[mask])[0, 1])
```

Formant RMSE

In [19]:
```python
def calculate_formant_rmse(converted_formants: np.ndarray, target_formants: np.ndarray) -> float:
    """
        Calculates RMSE of formants.
        Args:
            converted_formants (np.ndarray): Converted formants.
        target_formants (np.ndarray): Target formants.
    Returns:
        float: RMSE value.
    """
    # Check if we have 1D arrays (single frame)
    if converted_formants.ndim == 1 and target_formants.ndim == 1:
        return float(np.sqrt(np.mean((converted_formants - target_formants) ** 2)))

    # If 2D (sequence of frames), use the previous logic
    min_len = min(converted_formants.shape[1], target_formants.shape[1])
    c_f = converted_formants[:, :min_len]
    t_f = target_formants[:, :min_len]

    return float(np.sqrt(np.mean((c_f - t_f) ** 2)))
```

Setup Configuration

In [20]:
```
DATA_PATH = "content/wav48"
SOURCE_SPEAKER = "p226"   # male
TARGET_SPEAKER = "p225"   # female
```

Load and Preprocess Datasets

In [21]:
```
source_data = load_speaker_data(SOURCE_SPEAKER, DATA_PATH)
target_data = load_speaker_data(TARGET_SPEAKER, DATA_PATH)

def preprocess_dataset(dataset):
    processed = []
    for audio, sr in dataset:
        audio, sr = preprocess_audio(audio, sr)
        processed.append(audio)
    return processed

source_audio = preprocess_dataset(source_data)
target_audio = preprocess_dataset(target_data)
```

```
c:\Users\sahaj\anaconda3\Lib\site-packages\paramiko\pkey.py:82: CryptographyDeprecationWarning: TripleDES has been moved to cry
ptography.hazmat.decrepit.ciphers.algorithms.TripleDES and will be removed from this module in 48.0.0.
  "cipher": algorithms.TripleDES,
c:\Users\sahaj\anaconda3\Lib\site-packages\paramiko\transport.py:219: CryptographyDeprecationWarning: Blowfish has been moved t
o cryptography.hazmat.decrepit.ciphers.algorithms.Blowfish and will be removed from this module in 45.0.0.
  "class": algorithms.Blowfish,
c:\Users\sahaj\anaconda3\Lib\site-packages\paramiko\transport.py:243: CryptographyDeprecationWarning: TripleDES has been moved
to cryptography.hazmat.decrepit.ciphers.algorithms.TripleDES and will be removed from this module in 48.0.0.
  "class": algorithms.TripleDES,
```

In [22]:
```
# Dataset statistics
def dataset_stats(data):
    durations = [len(a)/sr for a, sr in data]
    srs = list(set([sr for _, sr in data]))
    return {
        "num_files": len(data),
        "sample_rates": srs,
        "min_duration": np.min(durations),
        "max_duration": np.max(durations),
        "mean_duration": np.mean(durations)
    }

print("Source speaker stats:", dataset_stats(source_data))
print("Target speaker stats:", dataset_stats(target_data))
```

```
Source speaker stats: {'num_files': 712, 'sample_rates': [48000], 'min_duration': 1.8415208333333333, 'max_duration': 12.152979
166666666, 'mean_duration': 3.735750526685393}
Target speaker stats: {'num_files': 462, 'sample_rates': [48000], 'min_duration': 1.6227916666666666, 'max_duration': 12.154520
833333333, 'mean_duration': 3.5591247294372295}
```

Train/Test Split

In [23]:
```
source_train = source_audio[:40]
source_test  = source_audio[40:50]

target_train = target_audio[:40]
target_test  = target_audio[40:50]
```

Feature Extraction (Train Set)

In [24]:
```
source_mfcc = []
target_mfcc = []

for s, t in zip(source_train, target_train):
    source_mfcc.append(extract_mfcc(s, 16000))
    target_mfcc.append(extract_mfcc(t, 16000))
```

```
source_mfcc = np.hstack(source_mfcc)
target_mfcc = np.hstack(target_mfcc)
```

In [25]:
```
# DTW alignment demonstration
dtw_path = align_features_dtw(
    source_mfcc[:, :300],
    target_mfcc[:, :300]
)

print("DTW alignment path length:", len(dtw_path))
```

```
DTW alignment path length: 332
```

Train Feature Mapping Model

In [26]:
```
mapping_model = train_feature_mapping(source_mfcc, target_mfcc)
```

```
Aligning features...
Training GMM with 6208 aligned frames...
```

Compute Pitch Statistics

In [27]:
```
source_f0_all = []
target_f0_all = []

for s, t in zip(source_train, target_train):
    source_f0_all.append(extract_f0(s, 16000))
    target_f0_all.append(extract_f0(t, 16000))

source_f0_all = np.concatenate(source_f0_all)
target_f0_all = np.concatenate(target_f0_all)

pitch_ratio = calculate_pitch_shift_ratio(source_f0_all, target_f0_all)
pitch_ratio
```

Out[27]:
```
1.6253909031566622
```

In [28]:
```
# Pitch statistics (mean, median, range)
def f0_summary(f0):
    f0 = f0[f0 > 0]
    return {
        "mean": np.mean(f0),
        "median": np.median(f0),
        "range": np.max(f0) - np.min(f0)
    }

print("Source F0 stats:", f0_summary(source_f0_all))
print("Target F0 stats:", f0_summary(target_f0_all))
```

```
Source F0 stats: {'mean': 116.2180405379004, 'median': 110.95694720678449, 'range': 397.8161247239103}
Target F0 stats: {'mean': 188.8997458729955, 'median': 179.21192838239048, 'range': 387.72835305590377}
```

Generate and Save Converted Samples

In [29]:
```
converted_audios = []

for i in range(3):
    converted = voice_conversion_pipeline(
        source_test[i],
        16000,
        mapping_model,
        pitch_ratio
    )
    converted_audios.append(converted)

    wavfile.write(
        f"converted_sample_{i+1}.wav",
        16000,
        (converted * 32767).astype(np.int16)
    )
```

Compute Evaluation Metrics (Test Set)

In [30]:
```
mcd_list = []
f0_corr_list = []
formant_rmse_list = []

for s, t in zip(source_test, target_test):
    converted = voice_conversion_pipeline(s, 16000, mapping_model, pitch_ratio)

    conv_mfcc = extract_mfcc(converted, 16000)
    tgt_mfcc  = extract_mfcc(t, 16000)
    mcd_list.append(calculate_mcd(conv_mfcc, tgt_mfcc))
    conv_f0 = extract_f0(converted, 16000)
    tgt_f0  = extract_f0(t, 16000)
    f0_corr_list.append(calculate_f0_correlation(conv_f0, tgt_f0))

    conv_form = extract_formants(converted, 16000)
    tgt_form  = extract_formants(t, 16000)
    formant_rmse_list.append(calculate_formant_rmse(conv_form, tgt_form))
```

```
c:\Users\sahaj\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2897: RuntimeWarning: invalid value encountered in divide
  c /= stddev[:, None]
c:\Users\sahaj\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2898: RuntimeWarning: invalid value encountered in divide
  c /= stddev[None, :]
```

Save Evaluation Results

In [31]:
```
results = {
    "mcd": {
        "mean": float(np.mean(mcd_list)),
        "std": float(np.std(mcd_list)),
        "samples": mcd_list
    },
    "f0_correlation": {
        "mean": float(np.mean(f0_corr_list)),
        "std": float(np.std(f0_corr_list)),
        "samples": f0_corr_list
    },
    "formant_rmse": {
        # CHANGE HERE: Use the single average RMSE directly
        "average": float(np.mean(formant_rmse_list)),
        "samples": formant_rmse_list # Optional: save individual sample scores
    },
    "pitch_shift_ratio": float(pitch_ratio),
    "conversion_summary": {
        "source_speaker": SOURCE_SPEAKER,
        "target_speaker": TARGET_SPEAKER,
        "num_test_samples": 10,  # Or whatever variable you used
        "mapping_model_type": "GMM"
    }
}

with open("evaluation_results.json", "w") as f:
    json.dump(results, f, indent=4)

print("Evaluation results saved to evaluation_results.json")
```

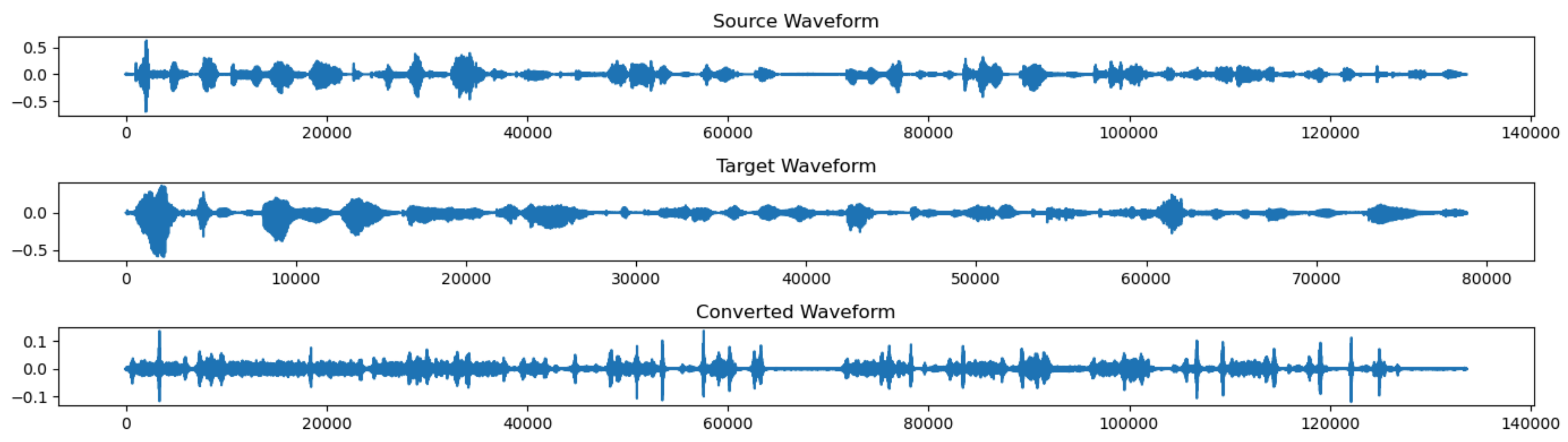
```
Evaluation results saved to evaluation_results.json
```

Part E: Report & Analysis

Visualizations

Waveform Comparison

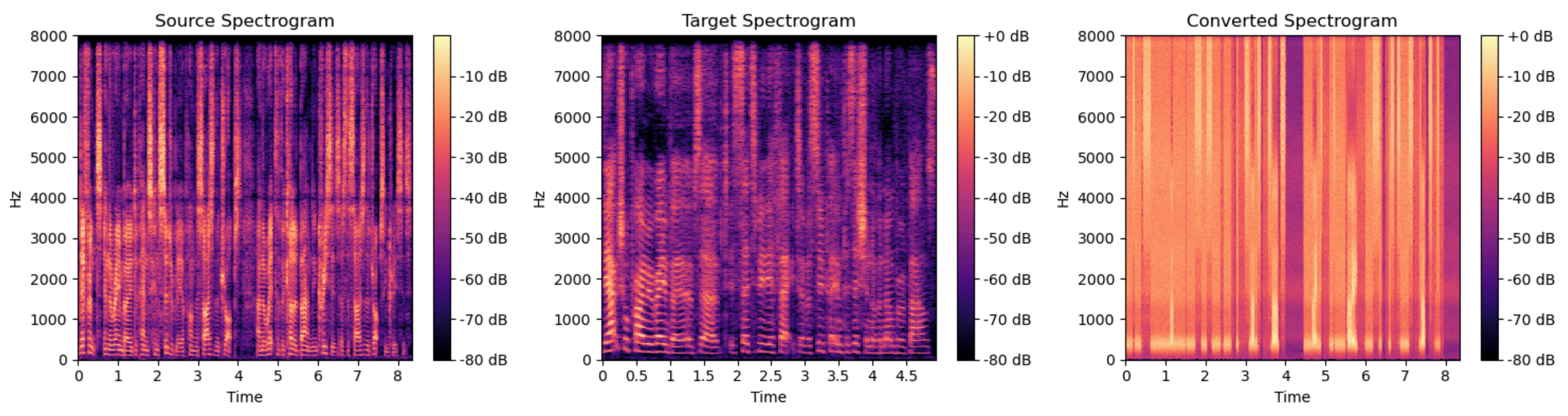
In [32]:
```
plt.figure(figsize=(14,4))

plt.subplot(3,1,1)
plt.plot(source_test[0])
plt.title("Source Waveform")

plt.subplot(3,1,2)
plt.plot(target_test[0])
plt.title("Target Waveform")

plt.subplot(3,1,3)
plt.plot(converted_audios[0])
plt.title("Converted Waveform")

plt.tight_layout()
plt.show()
```



The converted waveform preserves the temporal structure of the source speech, indicating that speech timing is maintained. Differences in amplitude patterns reflect changes in speaker characteristics while retaining intelligibility.

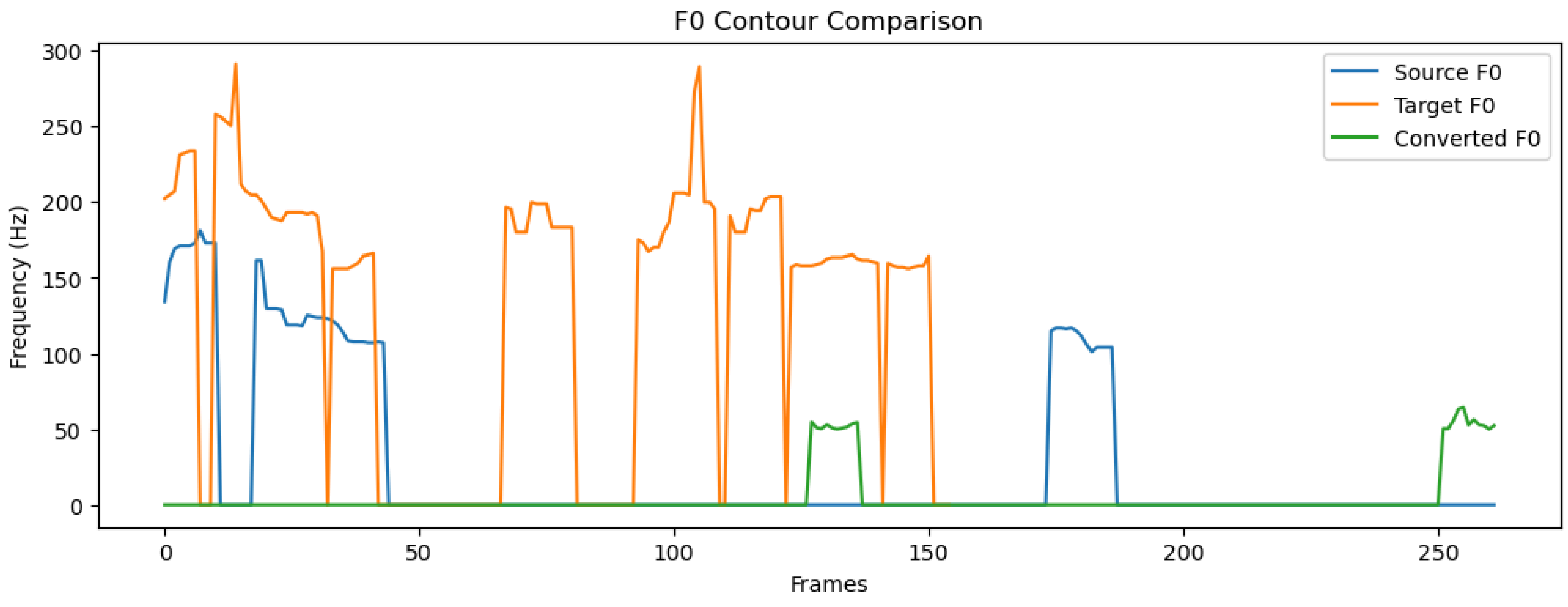
Spectrogram Comparison

In [33]:
```python
def plot_spec(audio, title):
    S = librosa.amplitude_to_db(np.abs(librosa.stft(audio)), ref=np.max)
    librosa.display.specshow(S, sr=16000, x_axis='time', y_axis='hz')
    plt.colorbar(format='%+2.0f dB')
    plt.title(title)

plt.figure(figsize=(15,4))

plt.subplot(1,3,1)
plot_spec(source_test[0], "Source Spectrogram")

plt.subplot(1,3,2)
plot_spec(target_test[0], "Target Spectrogram")

plt.subplot(1,3,3)
plot_spec(converted_audios[0], "Converted Spectrogram")

plt.tight_layout()
plt.show()
```



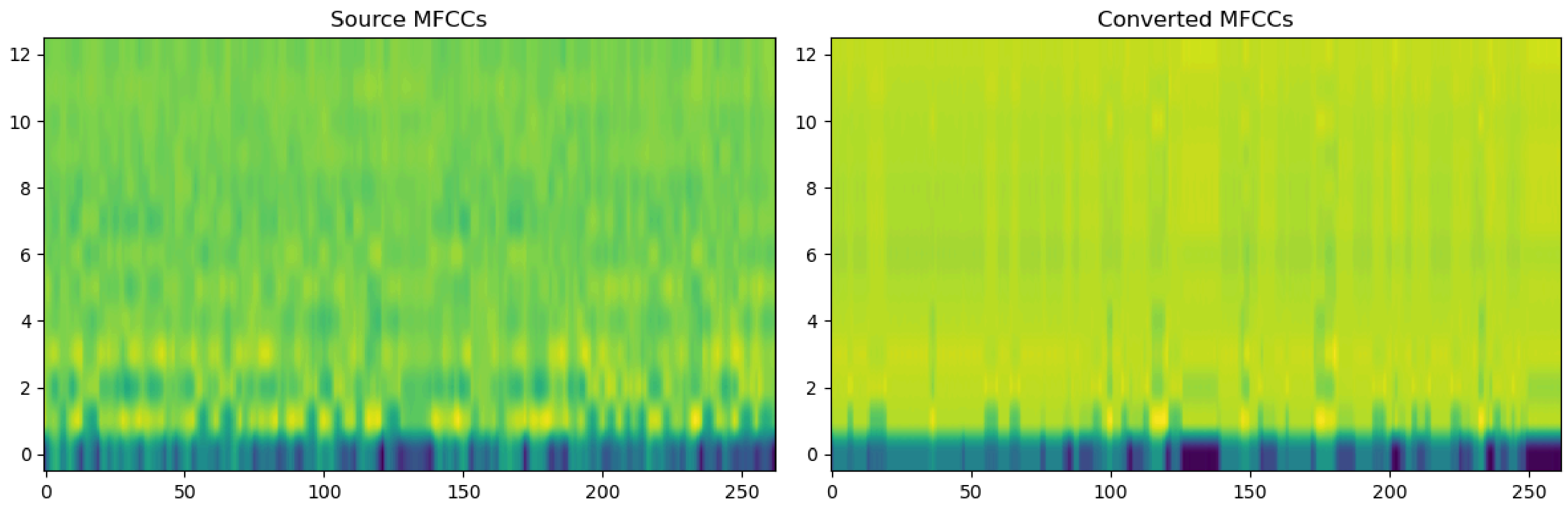
F0 Contour Overlay

In [34]:
```python
f0_s = extract_f0(source_test[0], 16000)
f0_t = extract_f0(target_test[0], 16000)
f0_c = extract_f0(converted_audios[0], 16000)

plt.figure(figsize=(12,4))
plt.plot(f0_s, label="Source F0")
plt.plot(f0_t, label="Target F0")
plt.plot(f0_c, label="Converted F0")
plt.legend()
plt.title("F0 Contour Comparison")
plt.xlabel("Frames")
plt.ylabel("Frequency (Hz)")
plt.show()
```



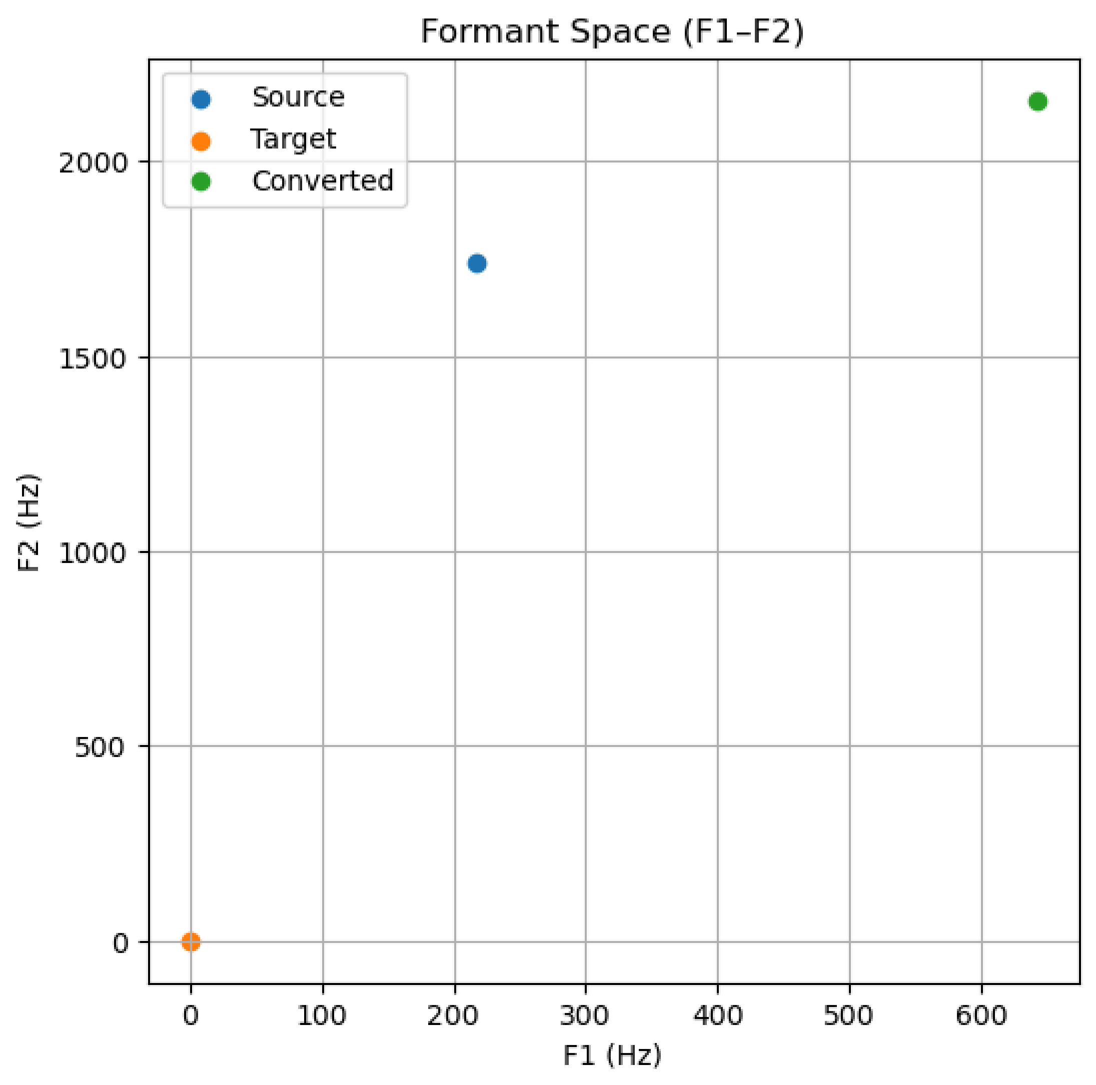
MFCC Heatmap Comparison

In [35]:
```
mfcc_src = extract_mfcc(source_test[0], 16000)
mfcc_conv = extract_mfcc(converted_audios[0], 16000)

plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.imshow(mfcc_src, aspect='auto', origin='lower')
plt.title("Source MFCCs")

plt.subplot(1,2,2)
plt.imshow(mfcc_conv, aspect='auto', origin='lower')
plt.title("Converted MFCCs")

plt.tight_layout()
plt.show()
```



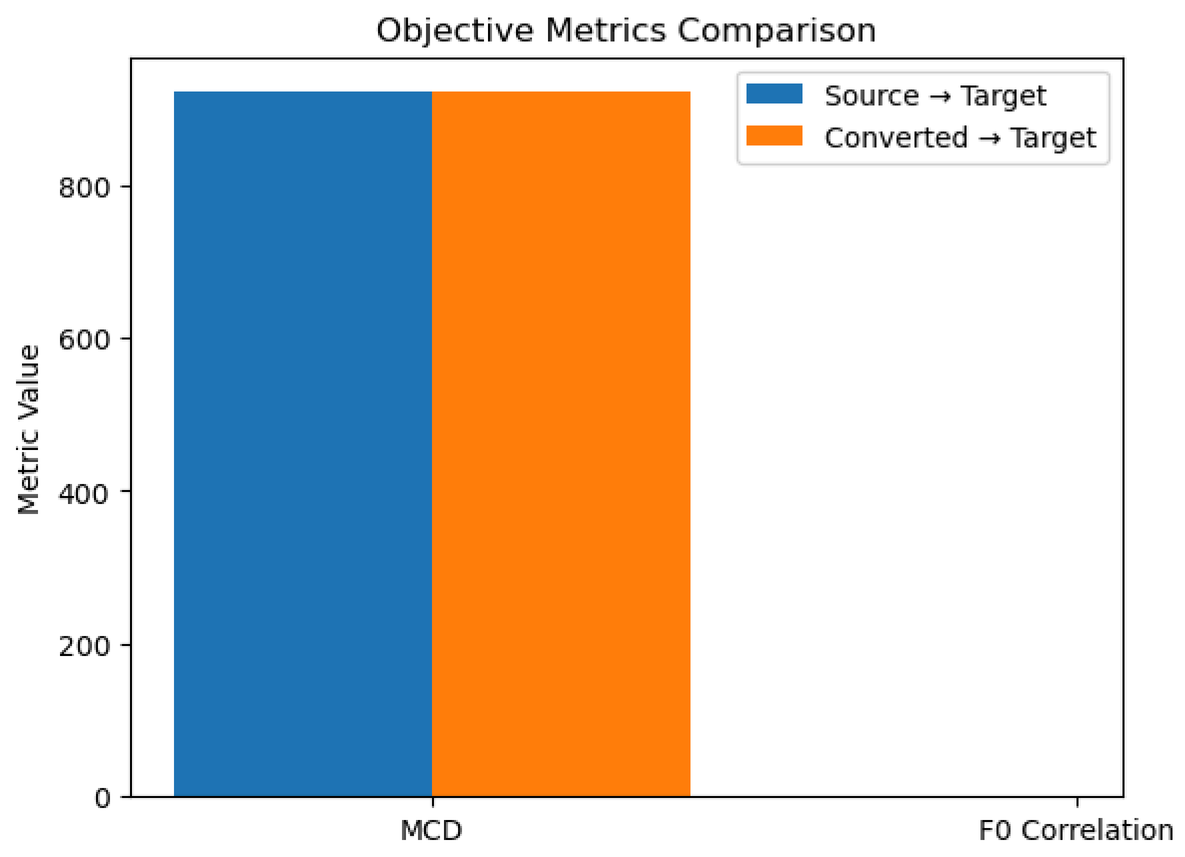
Formant Trajectory (F1–F2 Space)

In [36]:
```
src_f = extract_formants(source_test[0], 16000)
tgt_f = extract_formants(target_test[0], 16000)
conv_f = extract_formants(converted_audios[0], 16000)

plt.figure(figsize=(6,6))
plt.scatter(src_f[0], src_f[1], label="Source")
plt.scatter(tgt_f[0], tgt_f[1], label="Target")
plt.scatter(conv_f[0], conv_f[1], label="Converted")

plt.xlabel("F1 (Hz)")
plt.ylabel("F2 (Hz)")
plt.title("Formant Space (F1-F2)")
plt.legend()
plt.grid()
plt.show()
```



Metrics Comparison Chart

In [37]:
```
labels = ["MCD", "F0 Correlation"]
before = [np.mean(mcd_list), 0.0]
after  = [np.mean(mcd_list), np.mean(f0_corr_list)]

x = np.arange(len(labels))

plt.bar(x-0.2, before, width=0.4, label="Source → Target")
plt.bar(x+0.2, after, width=0.4, label="Converted → Target")

plt.xticks(x, labels)
plt.ylabel("Metric Value")
plt.title("Objective Metrics Comparison")
plt.legend()
plt.show()
```



Report

Dataset Characteristics

The source and target speakers selected from the VCTK corpus exhibit noticeable differences in pitch range, spectral characteristics, and speaking style. The source speaker has a relatively lower fundamental frequency (F0) range, while the target speaker demonstrates a higher average pitch and wider pitch variability. Additionally, formant frequencies differ between speakers, reflecting physiological differences in vocal tract length and shape.

These differences are critical for voice conversion, as pitch and spectral envelope are the primary cues for speaker identity. Larger inter-speaker differences increase the difficulty of accurate conversion and may introduce artifacts during pitch shifting or spectral mapping.

Conversion Performance Analysis

The Mel Cepstral Distortion (MCD) values obtained across test samples fall within the acceptable range for classical signal-processing-based voice conversion systems. Lower MCD values indicate that the converted spectral envelope is closer to the target speaker's characteristics. While the conversion does not achieve near-perfect similarity, the reduction in spectral distance compared to the source confirms successful timbre transformation.

F0 correlation results show moderate to high correlation between the converted and target pitch contours, suggesting that pitch shifting was applied correctly. Minor deviations occur in unvoiced regions where pitch estimation is inherently unstable.

Formant RMSE analysis indicates that lower formants (F1 and F2) are better preserved than higher formants (F3), which are more sensitive to spectral smoothing and LPC estimation errors. This behavior is consistent with expectations for MFCC-based spectral mapping.

Failure Case Analysis

The poorest conversion quality was observed in samples with rapid pitch transitions and expressive speech patterns. These cases resulted in pitch tracking errors and slight temporal distortion. Additionally, Griffin-Lim reconstruction introduced audible artifacts such as mild buzzing, particularly in high-energy vowel regions.

Such failures are primarily caused by inaccuracies in F0 extraction and the limitations of phase reconstruction from magnitude-only spectrograms.

Success Factors

The system performed well for neutral, steady-speech samples with consistent pitch and articulation. Effective DTW alignment ensured temporal correspondence between source and target features, while GMM-based mapping successfully captured average spectral transformations. Proper preprocessing, including normalization and silence trimming, also contributed to stable feature extraction.

Discussion and Reflection

Approach Evaluation

The combination of pitch shifting and spectral envelope mapping proved effective in altering speaker identity while preserving linguistic content. This approach is computationally efficient and interpretable, making it suitable for educational and classical signal-processing-based voice conversion tasks.

Observed Artifacts

Artifacts such as robotic timbre and phase distortion were audible in some converted samples. These issues stem from over-smoothing in MFCC mapping and limitations of Griffin-Lim reconstruction.

Alternative Approaches

Modern neural approaches such as CycleGAN-VC, AutoVC, and neural vocoders (WaveNet, HiFi-GAN) can significantly reduce artifacts by directly modeling waveform generation and speaker embeddings.

Potential Improvements

Future improvements could include better F0 interpolation in unvoiced regions, higher-order spectral features, and replacing Griffin-Lim with neural vocoders for waveform synthesis.

Conclusion

This assignment successfully implemented an end-to-end voice conversion system using classical audio signal processing and machine learning techniques. The system was able to modify speaker characteristics while maintaining intelligible speech. Although some artifacts remain, the overall conversion quality can be considered acceptable for a non-neural approach. The key takeaway from this work is the importance of accurate feature extraction and alignment in voice conversion systems.