複数人の自発音を考慮したトランスオーラルシステムの開発

関西大学　環境都市工学部　建築学科
建築環境工学第Ⅰ研究室
建 15-35　小関　勇太
指導教員　豊田　政弘　准教授

目次

図表目次

1. 序論

1.1 研究背景

音を発する行為とその周辺の音環境や空間を伝搬して自分自身の耳に聞こえてくる聴覚フィードバックには深い結びつきがあるとされており、歌唱者は舞台において、自身の発した反射音などを通して知覚することで、声の大きさや速さ、歌い方などを調整することが知られている。その為、聴衆だけでなく発話者や演奏者に対して最適な音環境を構築する必要がある。現在の講堂やホールの音響設計においては、主に音響障害を防止しつつ、座席位置において最適な残響時間を得ることを目的として、設計やシミュレーションが行われている[1]。このように、コンサートホールなどの音響設計においては、主に観客側を聴取の主体として物理的な指標に基づいて設計されることが多く、発話者や演奏者を聴取の主体とした研究は少ない。

1.2 既往研究

大谷らによる「実時間バイノーラル合成システムによる自発音の仮想聴空間での再現」の研究では、設計段階において発話者や演奏者による主観的な評価を可能にする可聴化システムの実現を目標とし、ダミーヘッド測定で得られるインパルス応答（IR: Impulse Response）を用いるシステム（従来システム）と 64 チャンネルのマイクロホンアレイ測定で得られた IR を用いるシステム（アレイシステム）の 2 つのシステムを作成し、再現度の高い仮想的な自発音可聴化システムを検討しており、再現性についてはどちらも高い精度を示したがアレイシステムの方がより高い再現性があったという結果であった。また、頭部運動による空間知覚精度に有意差が見られないという結果も得られている[2]。

1.3 目的

本研究では、既存の講堂やコンサートホールを仮想聴空間内で再現し、発話者や演奏者による主観的な評価を可能にするシステムの実現を目標とし、聴取者自身が発した音声を聴取する環境を仮想的に再現することができる自発音可聴化システムを構築する。具体的には、再生方法について従来システムではヘッドホンを用いているが、それをスピーカに変えたシステム（提案システム）を提案し、また、デュエット、カルテット等複数人での演奏がなされることもあることから提案システムを複数人に対応させる。従来システム、一人用提案システム、複数人用提案システムの 3 つのシステムを比較し、従来システムと同程度の再現性を有しているか検討するとともに、

頭部運動による空間知覚精度が変化するかを検討する。本システムの開発によって、発話者や演奏者にとって最適な音環境の設計を可能にする足がかりとする。

2. 方法

2.1 バイノーラルとトランスオーラル

バイノーラル信号とは、実頭やダミーヘッドの両耳に装着した 2 つのマイクロホンで収音した 2 チャンネル音響信号のことを言う（Fig. 1）。このバイノーラル信号を用いてヘッドホンから再生する方式をバイノーラル再生法という（Fig. 2）。また、スピーカを用いてバイノーラル再生を実現する方式をトランスオーラル再生法という。バイノーラル再生法では、ヘッドホンを用いるため右耳の音は右耳へ左耳の音は左耳へ伝えることが可能であるが、スピーカを使った再生法では左右スピーカからの音がそれぞれ反対側の耳に入ってしまうクロストークという問題が生じてしまう（Fig. 3）。このクロストークを消す処理として逆フィルタを作成し、音の出力前にこの逆フィルタをかけることでクロストークの成分を打ち消す方法がトランスオーラル再生法である。

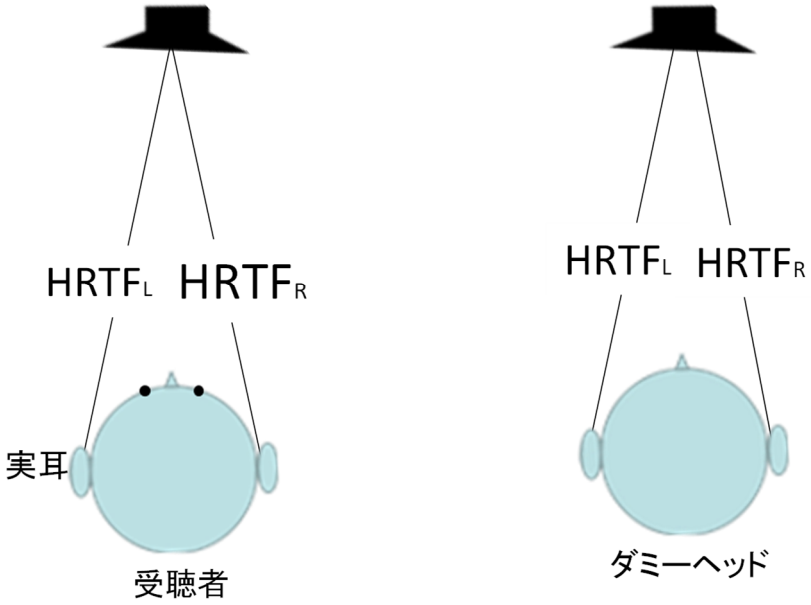

Fig. 1　バイノーラル信号取得方法

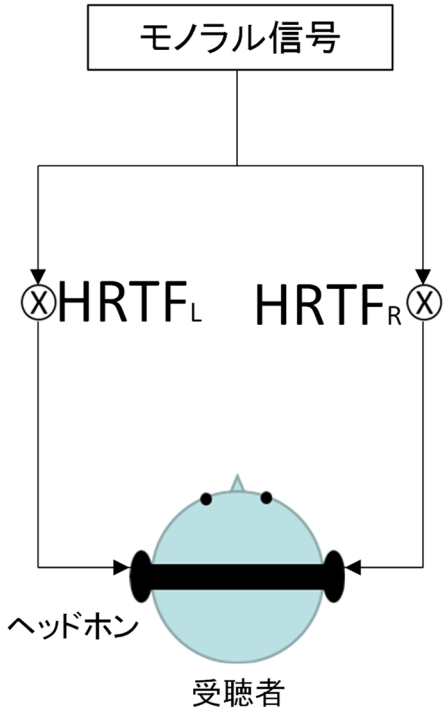

Fig. 2　バイノーラル再生法

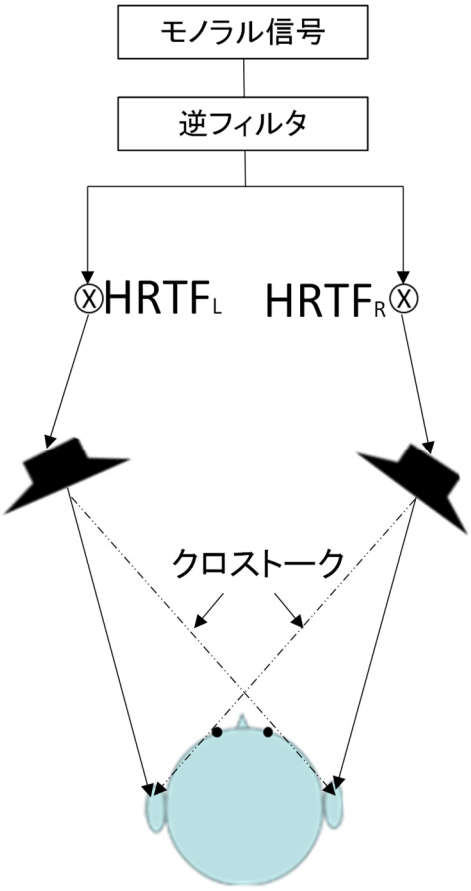

Fig. 3　トランスオーラル再生法

2.2 逆フィルタ

あるシステムがあったときに、その入力から出力を求める問題を順問題、出力から入力を推定する問題を逆問題という。音場再現における逆問題は、原音場の観測点で音圧などの物理量を観測し、再生音場の制御点で同じ物理量が観測されるように二次音源（スピーカ）信号を計算する問題のことを指す。また、この問題を実現するフィルタを逆フィルタと呼ぶ[3]。

Fig. 4 が示すように人が聞く音をそれぞれ Y_L、Y_R、スピーカから耳への伝達関数をそれぞれ H_{LL}、H_{LR}、H_{RL}、H_{RR}、スピーカからの出力を X_L、X_R とすると Y_L と Y_R は次のように表すことができる。

$$Y_L = H_{LL}X_L + H_{RL}X_R \tag{1}$$

$$Y_R = H_{LR}X_L + H_{RR}X_R \tag{2}$$

これを行列で表すと

$$\begin{pmatrix} Y_L \\ Y_R \end{pmatrix} = \begin{pmatrix} H_{LL} & H_{RL} \\ H_{LR} & H_{RR} \end{pmatrix} \begin{pmatrix} X_L \\ X_R \end{pmatrix} \tag{3}$$

となる。出力する X_L、X_R と人が聞く音 Y_L、Y_R がそれぞれ $X_L = Y_L$ と $X_R = Y_R$ になればクロストークがキャンセルされることになるので伝達関数の部分にフィルタを掛け合わせて単位行列となるようにすればよい。つまり、

$$\begin{pmatrix} H_{LL} & H_{RL} \\ H_{LR} & H_{RR} \end{pmatrix} \begin{pmatrix} G_{LL} & G_{RL} \\ G_{LR} & G_{RR} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{4}$$

となる行列 $\mathbf{G}$ をかければよいので、

$$\begin{pmatrix} G_{LL} & G_{RL} \\ G_{LR} & G_{RR} \end{pmatrix} = \begin{pmatrix} H_{LL} & H_{RL} \\ H_{LR} & H_{RR} \end{pmatrix}^{-1} \tag{5}$$

とする。ここで、人が聞く音を $\mathbf{Y}$、伝達関数を $\mathbf{H}$、フィルタを $\mathbf{G}$、スピーカからの出力を $\mathbf{X}$ と置くと、

$$\mathbf{Y} = \mathbf{HGX} \tag{6}$$

となる。このフィルタ $\mathbf{G}$ は逆フィルタと呼ばれる。

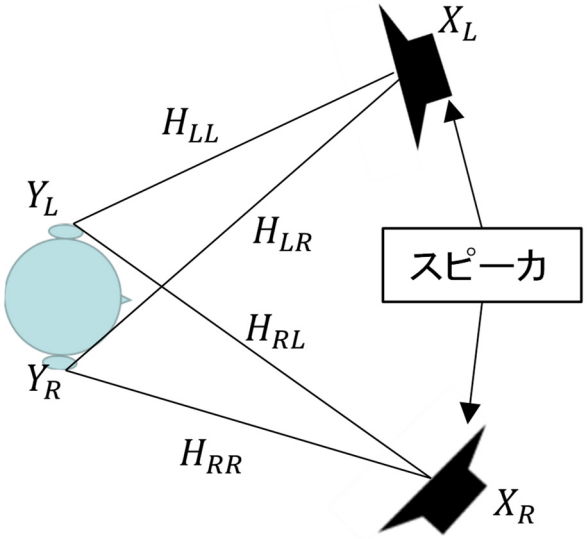

Fig. 4　スピーカから両耳までの伝達系

2.3 実装方法

プログラミング言語 Python とビジュアル・プログラミング・ソフトウェア MAX8 で提案システムを実装する。

Python とは、オランダの Guido van Rossum によって開発されたオープンソースのプログラミング言語であり、特徴として、

・必要最小限のシンプルな文法
・誰が書いても大筋は同じ
・様々な環境に対応
・本格的なオブジェクト指向

などが挙げられる[4]。

MAX8 とは、バーチャル・パッチ・コードでオブジェクトを繋げることで、インタラクティブ・サウンド、グラフィック、カスタム・エフェクトなどの開発が可能なソフトウェアである（Fig.5）。特徴として、

・MIDI/MPE および OSC プロトコルを完全にサポートする独自のコントロール・インターフェースを開発可能
・オリジナルのシンセサイザーを開発可能
・オーディオ⇔ビデオ制御を使用し独自のビデオ処理ルーティングおよびフィードバック・システムを開発可能
・リアルタイムのコード生成とコンパイル能力を統合している

等が挙げられる[5]。

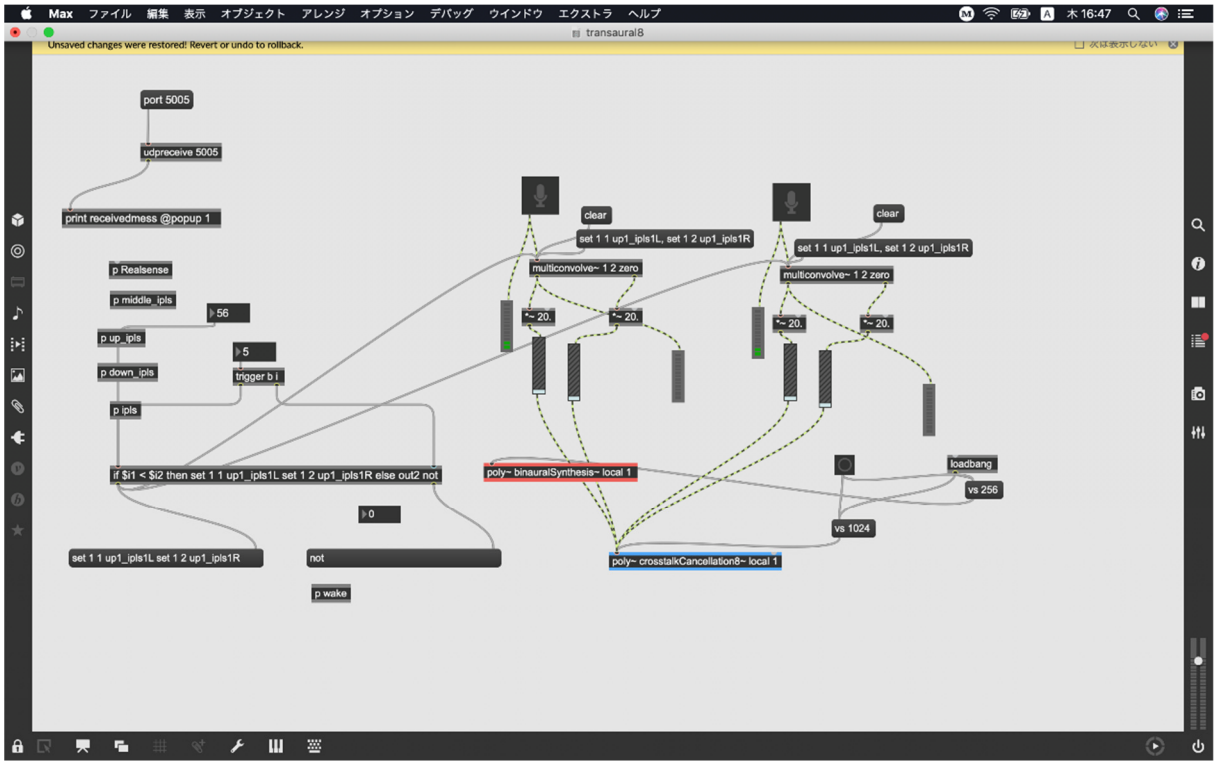

Fig. 5　MAX8 のプログラミング画面

2.4 使用機材

ヘッドトラッキングを可能とし、顔の動きまでセンシングできるようにインテルの RealSense を用いる（Fig. 6、Table 1）。従来用いられてきた Kinect よりもより性能になっている[6]。

Fig. 6　RealSense

Table 1　RealSense の仕様

動作仕様	
動作範囲(最小-最大)	0.11m～10m
Depth 解像度と FPS	1280×720
Depth 視野	85.2×58

スピーカは TIME DOMAIN mini（Fig. 7）を用いることとする。このスピーカの特徴は

・遠くからでもはっきりと聞き取れる
・小さい音でも、また騒音の中でもはっきりと聞き取れる
・日本語も英語も発音がはっきりと聞き取れる

等が挙げられる[7]。

Fig. 7　TIME DOMAIN mini

2.5 インパルス応答の測定

本研究で使用する IR は関西大学第二学舎 BIG ホール（Fig. 8）にて測定したものであり、実頭の両耳位置を受音点とし、TSP（Time Stretched Pulse）を音源とした。

受音応答に時間反転した TSP 信号を畳み込むことでインパルス応答を得ることが可能である。TSP 信号の再生、IR の計算は Python で行った。Fig. 9 に測定の様子を示す。この IR をフーリエ変換し、周波数領域にしたものを頭部伝達関数（HRTF: Head Related Transfer Function）と言う。Fig. 10 が示すように、顔の向きによって IR が違うため、顔の角度を変えた場合に対応するよう顔の角度を yaw 方向に 9 分割、pitch 方向に 3 分割した 27 分割して計測した。IR のサンプリング周波数は 44100 Hz であり、サンプル数は 262144 とした。顔の方向の分割を Fig. 11 と Fig. 12 に示す。

自ら音を発した際の直接音がそのまま耳へ届くものと考えれば、測定した IR をそのまま用いると直接音成分が 2 重になってしまう。そのため、測定した IR から直接音成分を取り除く必要がある。床からの反射音を第一反射音とすると耳へ届くまでに約 6.8ms であった。測定した IR から約 6.8ms 分の成分を全て 0 にすることで直接音成分を取り除いた。直接音成分を取り除く前後の IR を Fig. 13 に示す。

Fig. 8　関西大学第 2 学舎 BIG ホール

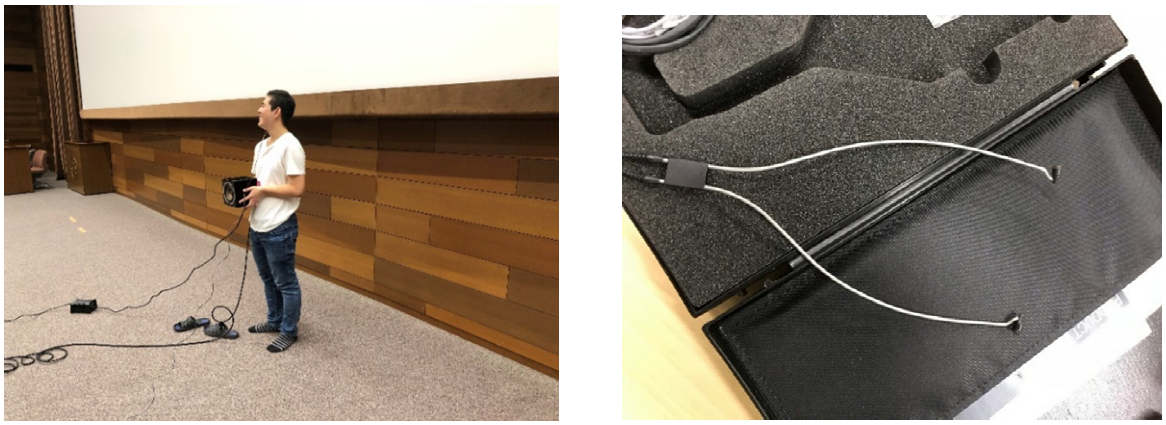

Fig. 9　インパルス応答の測定
（左：測定の様子，　右：耳に装着したマイク）

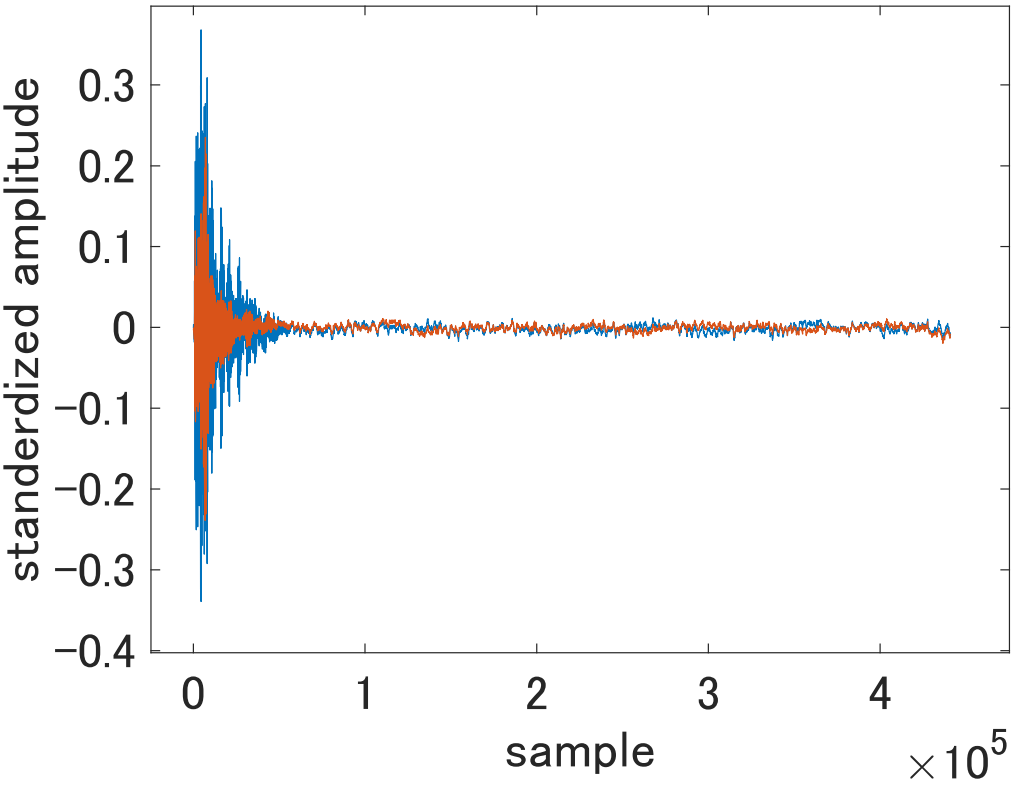

Fig. 10　顔の向きによる音の違い

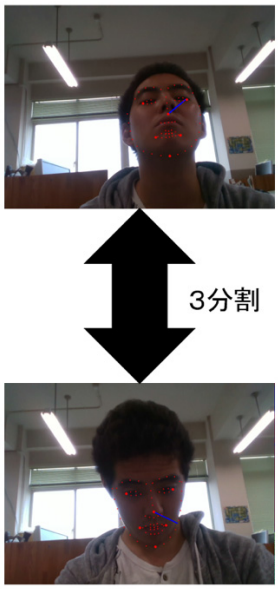

Fig. 11　pitch 方向の分割

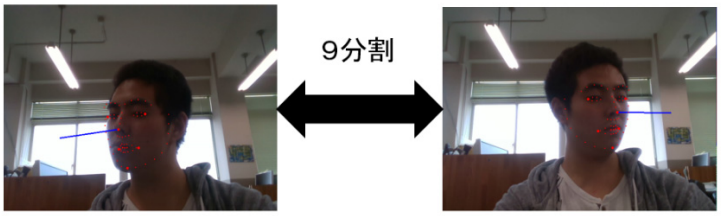

Fig. 12　yaw 方向の分割

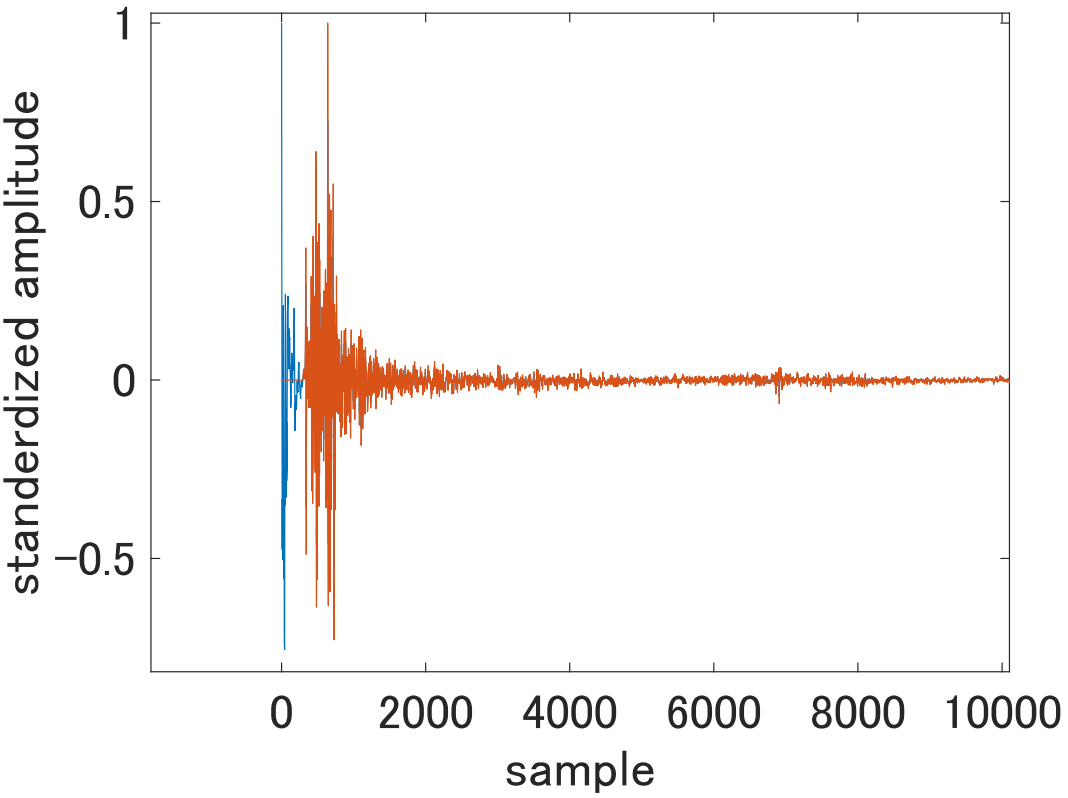

Fig. 13　測定した IR 拡大図
（青：元の IR,　オレンジ：直接音成分を取り除いた IR）

2.6 評価方法

高い再現精度があるとされている従来システムと比較して、提案システムの再現精度がどのように変化するのか、頭部運動による空間知覚精度が変化するのかを検討する。そのため、実験前に実空間での響きを確認させ、その後従来システム、一人用提案システム、複数人用提案システムの順で再現精度の評価を行う。音を発する際、室内にいると反射音が届くため、聞きたい音以外の音が耳に到来することになることから、再現音場の再現性や頭部運動による空間知覚精度が正確に評価できない。したがって、評価は反射音がない無響室で行う。従来システムでは耳にヘッドホンを付け、声を発し、実験前に確認した実空間での響く音の大きさや響く音の長さ等を確認させ、再現空間の再現精度、空間知覚精度を被験者の主観的な評価を得る。その後、提案システムの一人用、複数人用の順で、主観的な再現精度、空間知覚精度に関して評価をさせる。

評価方法としては、提案システムが一人用、複数人用ともに従来システムよりも再現精度が向上するのか低下するのかを検討する。また、頭部運動を許容した場合に空間知覚精度について有意差がみられるかを検討する。

条件について、まずは複数人として 2 人を想定する。2 人で高い評価を得られるの

であれば同じ処理を増やすことで 3 人、4 人に対応することが出来ると考えられる。また、ここでは一人当たり 4 つのスピーカを使用することとする。

2.7 システム概要

本研究で用いた従来システムは Python のみで、提案システムは Python だけで実装したものと Python と MAX8 の 2 つを用いて実装した 2 パターン用意した。従来システムと提案システムの Python だけで実装したものは実時間畳み込み、頭部運動検出の処理の両方を Python で行った。MAX8 を使用した提案システムでは頭部運動検出は Python で行い、実時間畳み込みを MAX8 上で行った。

従来システムと提案システムの 2 つのシステム概要を順に説明する。2 つのシステムに共通して行われている処理として、IR の実時間畳み込みが挙げられる。RealSense を用いて顔の角度と位置（yaw, roll, pitch, x, y, z）を取得し、その値から両耳の IR を収録したデータベースより選択する。時間領域の入力音源、IR（R, L）を高速フーリエ変換（FFT：Fast Fourier Transform）し、周波数領域において両者の乗算を行った結果に高速フーリエ逆変換（IFFT : Inverse Fast Fourier Transform）を行い、時間領域に変換する。その後、Overlap Add 法を用いて出力音とする[8]。模式図を Fig. 14 に示す。

従来システムにおいて、まず無響室内で、受聴者が発した音を受聴者の襟元に設置したマイクロホンによって収録する。収録した音をオーディオインターフェース（MOTU Pro オーディオ A16）を介し PC に渡す。PC で前述の処理を行いヘッドホンから音を再生する（Fig. 15）。

提案システムにおいて、従来システムと音の出力直前までは同じ処理をするがスピーカからのクロストークを消すために逆フィルタの計算をリアルタイムで行い、フィルタを変える処理を追加する（Fig. 16）。

処理中にある Overlap Add 法とは信号長の大きい信号を一定の区画長で区切り、区間毎にフィルタとの畳み込みを行うことで、信号長の大きい信号に対する畳み込み演算を高速に行う手法である。畳み込み後の出力信号の区画長の後半部分と次の区間の畳み込み後の出力信号の前半部分を重ね合わせ加算したものが各区間の出力となる。各区間の出力結果を連結して繋ぎ合わせた波形が最終的な出力となる。1 回で出力する音のバッファを 512 とした模式図を Fig. 17 に示す。

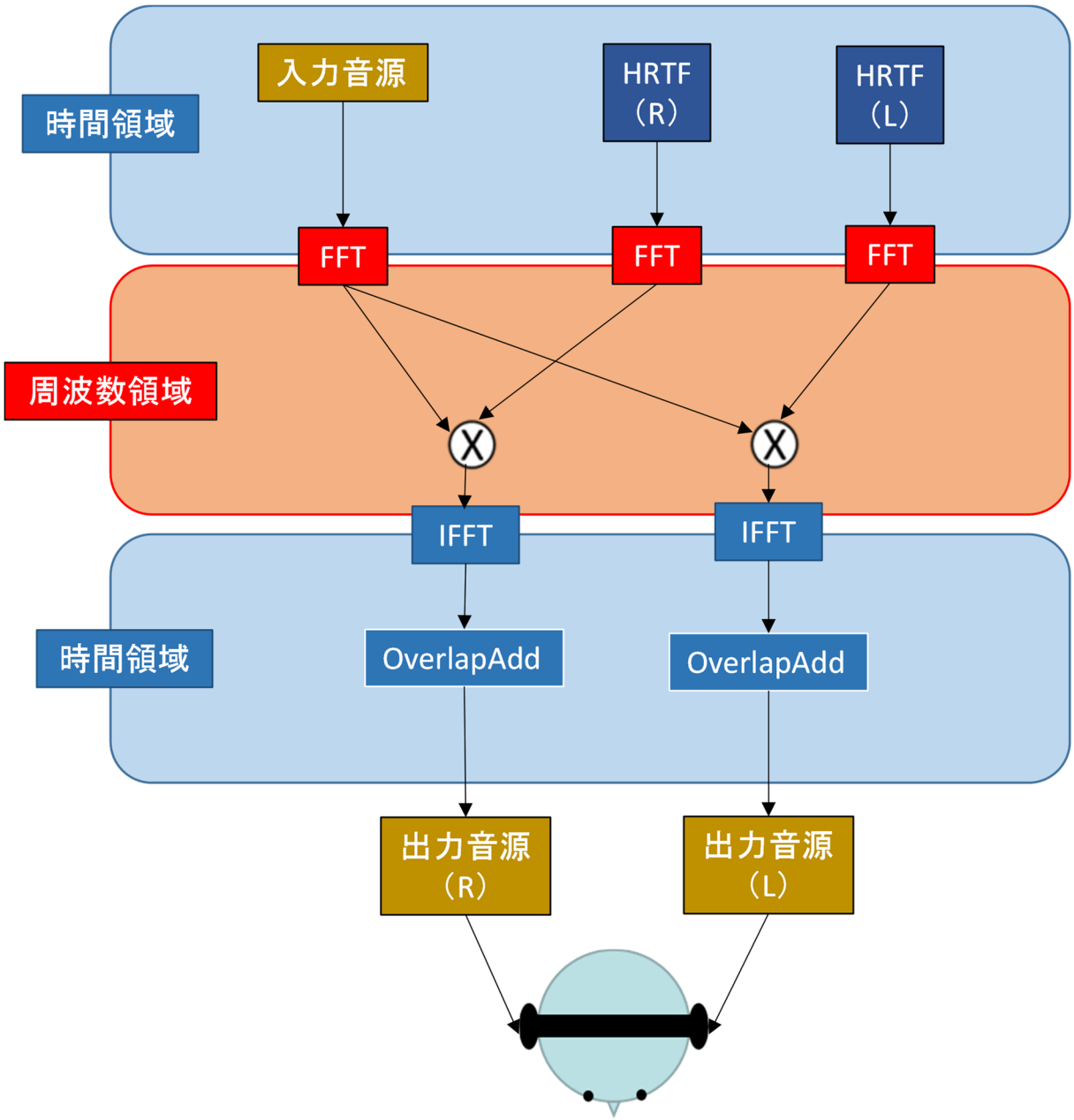

Fig. 14　従来システムと提案システムに共通する処理模式図

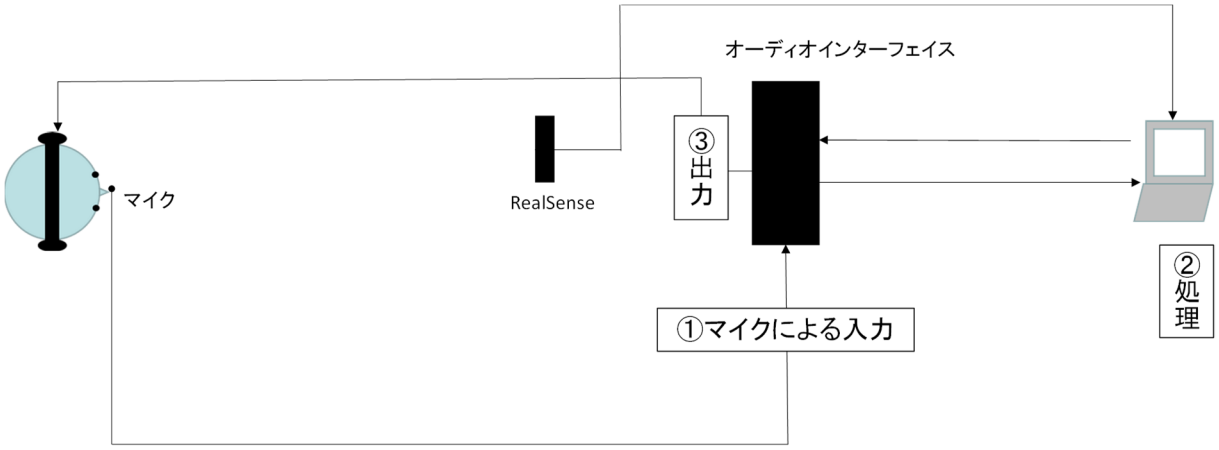

Fig. 15　従来システム模式図

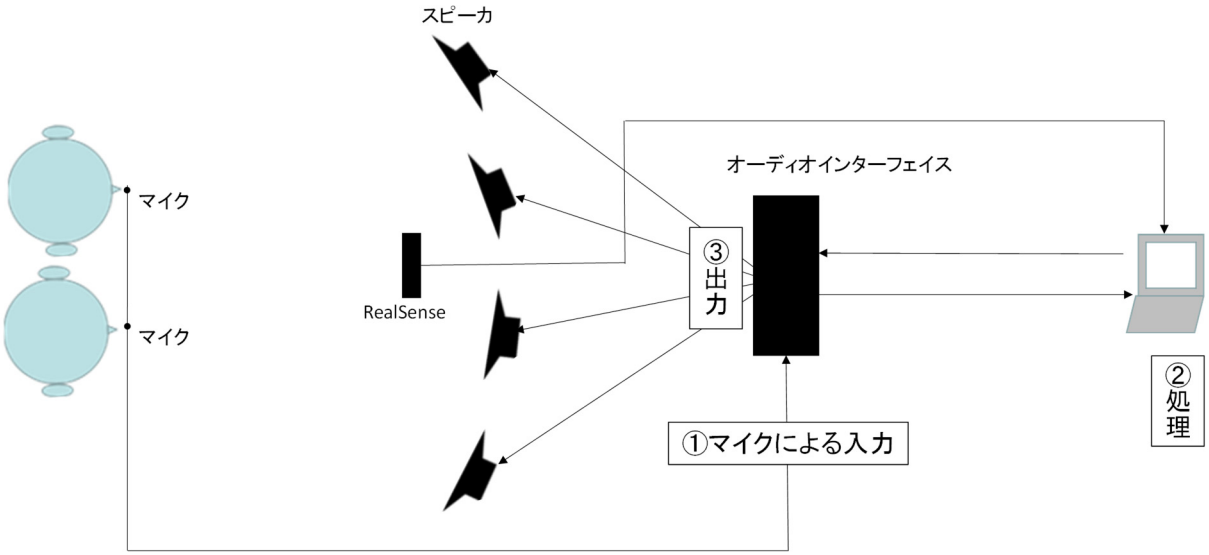

Fig. 16　提案システム模式図

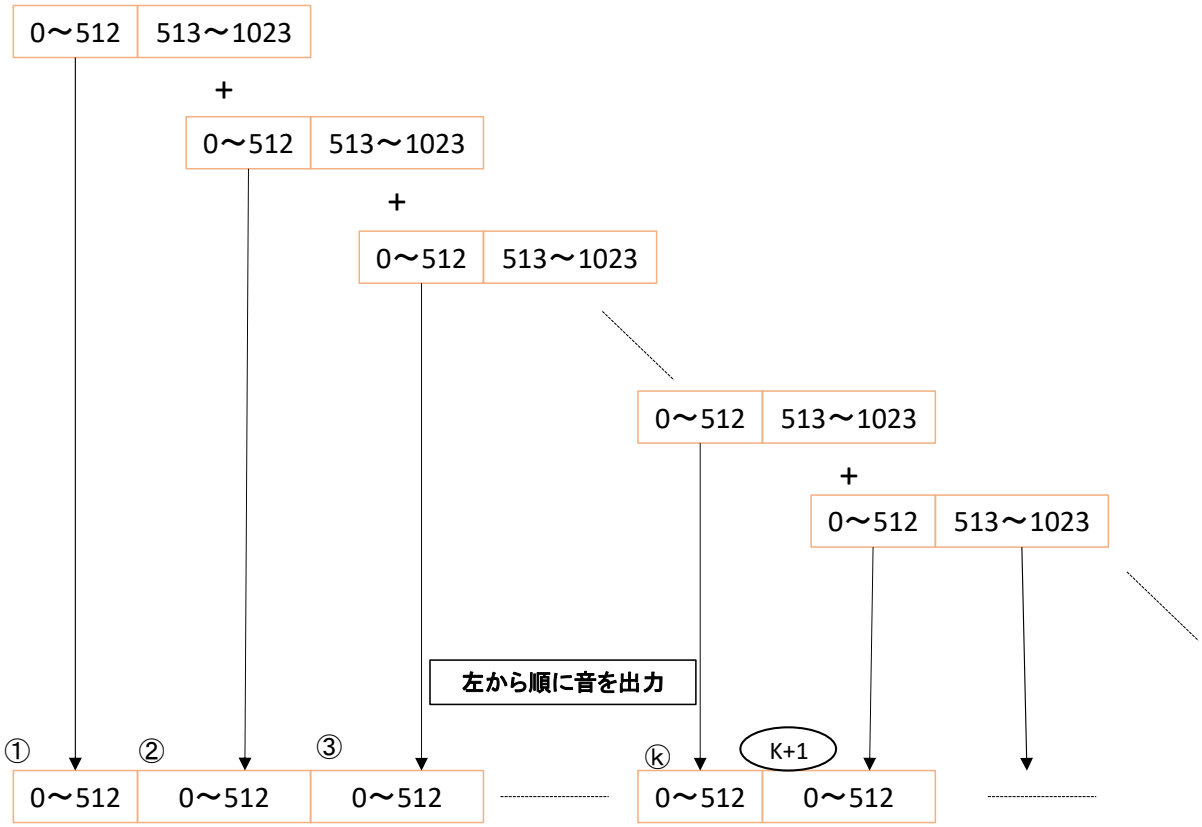

Fig. 17　Overlap Add 法模式図

2.8　プログラム内容

従来システムと提案システムの Python を用いたプログラムの内容について述べる。

① IR を読み込み、その後それぞれの IR を高速フーリエ変換して HRTF とし、一時保存する。

② マイクからの入力音のバッファを 1024 サンプルとし、入力音を 1024 サンプルずつに分け、1024 サンプルずつ高速フーリエ変換する。この際に、畳み込み処理を行うためとして 1024 サンプルのバッファの最後の 1025 サンプル目から 0 を追加し IR と同じサンプル数 262144 とする。

③ 262144 サンプルの IR と 262144 サンプルの入力音とで畳み込みを行い、高速フーリエ逆変換し、時間領域に変える。

④ ③で出来た 262144 サンプルから出力として 1024 サンプルだけ前から抜き取り出力する。残りの 261120 サンプルは一時保存する。

⑤ ④の 261120 のサンプルの後ろに 0 を 1024 サンプル追加し 262144 サンプル（前サンプル）とする。

⑥ 次の入力音のバッファ 1024 サンプルに対しても②～③の処理を再度行うと、また 262144 サンプル（後サンプル）が出来上がるので前サンプルと後サンプルとを足し算する（Overlap Add）。

⑦ 足し算した 262144 サンプルから出力として 1024 サンプルを前から抜き取り出力する。残りの 261120 サンプルの後ろに 0 を 1024 サンプル追加し、262144 サンプル（前サンプル）にする。

⑧ ⑥に戻る。

①～⑧までの処理は従来システムと提案システムに共通している処理内容である。模式図を Fig. 18 に示す。

提案システムでは①に追加の処理としてスピーカから耳までの IR を読み込み高速フーリエ変換し、一時保存しておく処理が加えられている。また、出力する 1024 サンプルに対して⑨、⑩、⑪の処理を行っている。

⑨ 一時保存しておいた伝達関数を用いて、(5)式に従い、逆行列を求める。

⑩ 出力音を高速フーリエ変換し、⑨の逆行列と畳み込みをする。

⑪ ⑩でできたものを高速フーリエ逆変換し出力する。

の 3 つの処理を加えたものが提案システムである。

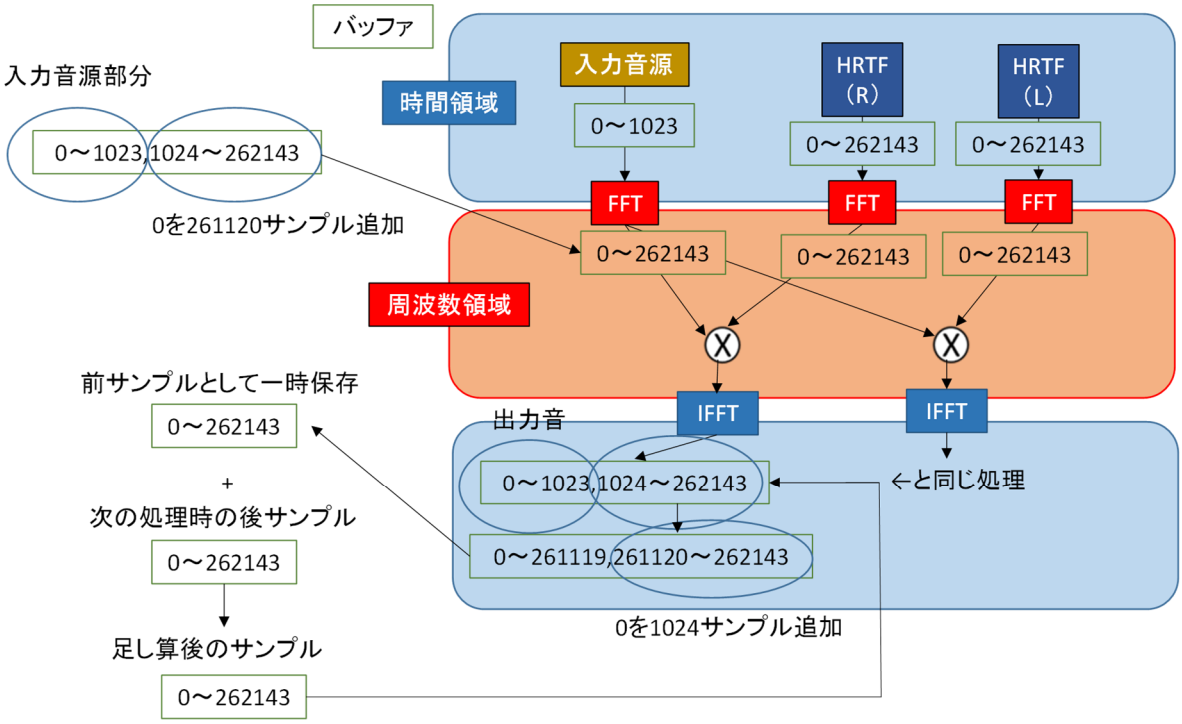

Fig. 18　アルゴリズム模式図

2.9　RealSense による顔の角度と位置取得

RealSense を用いた顔の角度と位置の取得方法について述べる。ワールド座標を RealSense の RGB カメラと深度カメラを用いて取得する。その座標と前もって与えている鼻の先、顎、左目の左端、右目の右端、口の左端、口の右端のカメラ座標を一致させ、PnP 問題を解き、顔の角度を求めている。位置に関しては RGB カメラと深度カメラで取得している。PnP 問題とは、ワールド座標系における n 点の 3 次元座標とそれらの点が観測された画像座標から、カメラの位置姿勢を推定する問題である。コンピュータビジョンにおける重要な問題の 1 つであり、ロボットナビゲーション、AR などで利用されている[9]。本研究では Python で上記の計算を行った。また、Python での並列処理の実装が困難であるため音を出すプログラムと顔の角度と位置の取得の 2 つのプログラムを別々に動作させた。その際のデータの受け渡し方法として OSC（Open Sound Control）を用いた。OSC とは電子機器やコンピュータなどの機器において音楽演奏データをネットワーク経由でリアルタイムに共有するための通信プロトコルである[10]。顔の角度と位置のプログラムでは顔の角度である yaw, pitch の情報を送り出す。音を出力プログラムでそれを受け取り、27 分割した顔の角度に合う IR をデータベースから取得する。また、提案システムでは、伝達関数もこの受け取ったデータを基に決める。1 人用提案システムでは Kurabayashi らが作成した伝達関数データベースを用いている[11]。複数人用提案システムでは関西大学第 4 学舎建築環境工学第 1 研究室無響室で測定したものを使用している。プログラムを

付録に記載しておく。RealSense による顔の角度と位置の取得の際の様子を Fig. 19 に示す。

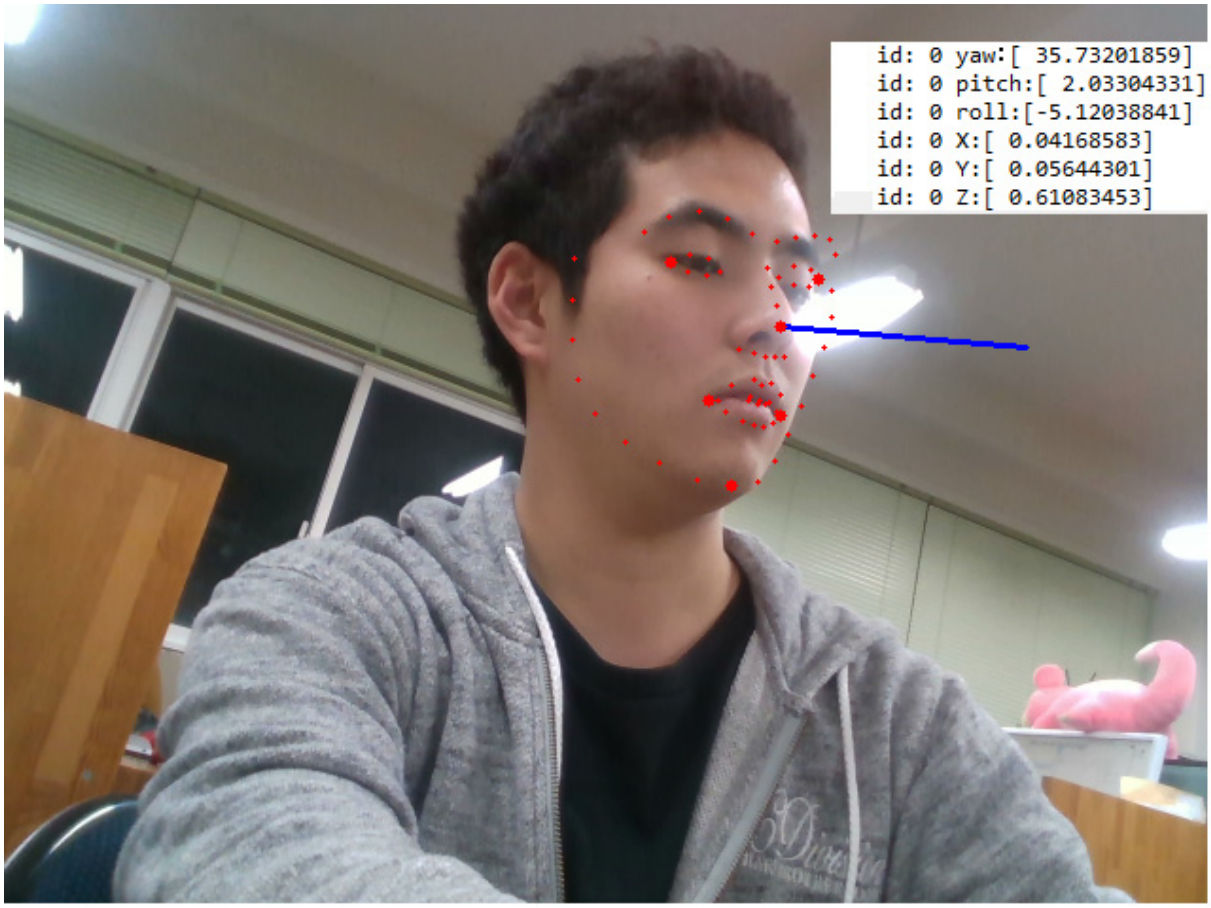

Fig. 19：RealSense による顔の角度と位置取得

3. 結果

Python により従来システム、一人用提案システム、複数人用提案システムと 3 つのシステムを実装した。従来システムにおいては、ややリアルタイム性が欠けるが実装出来ている事が分かった。提案システムにおいては、プログラム上では実装出来ていると思われるがプログラムが動かず、動作の確認ができていない。
MAX8 による実装は Python で従来システムが動作していた為、提案システムのみ実装を目指したが、途中までしか完成させることが出来なかった。したがって、評価を行うことが出来なかった。

4.考察

Python による従来システムのリアルタイム性がやや欠ける事についてはリアルタイム性を上げると音への処理が間に合わず動作が途中で停止してしまうからである。そのため、動作するギリギリのバッファに設定しており、ややリアルタイム性が欠けている。これはパソコンのスペックを上げる、IR のバッファ数を少なくするなどで対策は可能だが現実的ではなく、Python を用いた処理に対する新しいアルゴリズムを考えることが最適であると思われる。

Python だけで実装した提案システムに関しては、伝達関数や IR の読み込みと処理が音の出力に間に合わず動作しなかった。リアルタイム性を欠如させ、音の出力を遅くし、動作だけでも確認しようと試みたが音の出力を遅くすると一度の逆フィルタの計算量が増え、動作しなかった。解決策としては、Python のようなインタープリタ方式を使用している言語ではリアルタイムでの処理が間に合わなくなるのでコンパイル方式が利用出来る C++や C 言語のようなプログラミング言語を使用することが良いと考えられる。

MAX8 は、Python では提案システムが動作しないことが分かってから取り組み始めたため、扱いに慣れる途中で実装を終えてしまった。

5. 結論

本研究では実在する空間における自発音の再現を目標として、可聴化システムを 2 つ構築した。しかし、Python のみを用いた場合、従来システムは動作したが、提案システムは動作しなかった。リアルタイム性を求めるならインタープリタ方式言語ではなくコンパイル方式言語を使用すべきだと分かった。MAX8 を使用すれば動作しそうだったが、基本的な使い方や動作方法を理解するだけで時間がなくなってしまった。

今後の課題としては、新しいアルゴリズムの構築、使用言語の変更が挙げられ、動作を確認したのちにシステムの評価をする必要がある。また、スピーカから音を発することによりマイクにその音が入力されることでハウリングを起こす可能性があり、ハウリングを消去するハウリングキャンセラの作成も視野に入れる必要がある。

付録　OSC を用いた RealSense での頭部運動検出データ送信プログラム

```
import pyrealsense2 as rs
import numpy as np
import cv2
import dlib
from imutils import face_utils
import math
import time
from pythonosc import udp_client
from pythonosc.osc_message_builder import OscMessageBuilder
class FaceDetector():
def __init__(self):
self.xsize = 640
self.ysize = 480
self.pipeline = rs.pipeline()
self.config = rs.config()
self.config.enable_stream(rs.stream.color, self.xsize, self.ysize, rs.format.bgr8,
30)
self.config.enable_stream(rs.stream.depth, self.xsize, self.ysize, rs.format.z16,
30)
face_landmark_path = './shape_predictor_68_face_landmarks.dat'
self.detector = dlib.get_frontal_face_detector()
self.predictor = dlib.shape_predictor(face_landmark_path)
self.profile = self.pipeline.start(self.config)
self.align_to = rs.stream.color
self.align = rs.align(self.align_to)
self.depth_sensor = self.profile.get_device().first_depth_sensor()
self.dist_coeffs = np.zeros((4,1))          # Assuming no lens distortion
#Center は Nose Tip から 330 だけ w 方向（奥方向）
self.model_points = np.array([
(0.0, 0.0, 330.0),              # Nose tip
(0.0, -330.0, 275.0),           # Chin
(-225.0, 170.0, 195.0),         # Left eye left corner
```

```
(225.0, 170.0, 195.0),          # Right eye right corne
(-150.0, -150.0, 205.0),       # Left Mouth corner
(150.0, -150.0, 205.0)])          # Right mouth corner
self.camera_matrix = np.array(
[[self.xsize, 0, self.xsize/2],
[0, self.xsize, self.ysize/2],
[0, 0, 1]], dtype = "double")
self.Aorg = 0
self.IP = '127.0.0.1'
self.PORT = 5005
# UDP のクライアントを作る
self.client = udp_client.UDPClient(self.IP, self.PORT)
self.yaw_empty = np.array([])
self.pitch_empty = np.array([])
def detectFace(self):
self.images = self.pipeline.wait_for_frames()
self.color_frame = self.images.get_color_frame()
self.aligned_frames = self.align.process(self.images)
self.depth_frame = self.aligned_frames.get_depth_frame() # aligned でカラーフレームに深度フレームを合わせている
self.color_image = np.asanyarray(self.color_frame.get_data())
self.depth_image = np.asanyarray(self.depth_frame.get_data())
self.depth_colormap = cv2.applyColorMap(cv2.convertScaleAbs(self.depth_image, alpha=0.1), cv2.COLORMAP_JET)
self.face_rects = self.detector(self.color_image, 0)
if len(self.face_rects) > 0:
if self.yaw_empty.size == 0:
for self.i, d in enumerate(self.face_rects):
self.shape = self.predictor(self.color_image, self.face_rects[self.i])
self.shape = face_utils.shape_to_np(self.shape)
self.image_points = np.float32([self.shape[30], self.shape[8], self.shape[36],
self.shape[45], self.shape[48], self.shape[54]])
(self.success, self.rotation_vector, self.translation_vector) = cv2.solvePnP(
self.model_points, self.image_points, self.camera_matrix, self.dist_coeffs,
flags=cv2.SOLVEPNP_ITERATIVE)
(self.nose_end_point2D, self.jacobian) = cv2.projectPoints(np.array([(0.0, 0.0, 1000.0)]),
```

```
    self.rotation_vector, self.translation_vector,
    self.camera_matrix, self.dist_coeffs)
    p1 = (int(self.image_points[0][0]), int(self.image_points[0][1]))
    p2 = (int(self.nose_end_point2D[0][0][0]), int(self.nose_end_point2D[0][0][1]))
    cv2.line(self.color_image, p1, p2, (255,0,0), 2)
    cv2.line(self.depth_colormap, p1, p2, (255,0,0), 2)
    #顔器官点(68 点)を frame に描画
    for (x, y) in self.shape:
    cv2.circle(self.color_image, (x, y), 1, (0, 0, 255), -1)
    cv2.circle(self.depth_colormap, (x , y), 1, (0, 0, 255), -1)
    #顔器官点(68 点の内 6 点)を frema 濃くに描画
    for p in self.image_points:
    cv2.circle(self.color_image, (int(p[0]), int(p[1])), 3, (0,0,255), -1)
    cv2.circle(self.depth_colormap, (int(p[0]), int(p[1]) ), 3, (0,0,255), -1)
    self.rvec_matrix = cv2.Rodrigues(self.rotation_vector)[0]
    self.proj_matrix = np.hstack((self.rvec_matrix, self.translation_vector))
    self.eulerAngles = -cv2.decomposeProjectionMatrix(self.proj_matrix)[6]
    yaw      = self.eulerAngles[1]
    pitch = self.eulerAngles[0]
    roll    = self.eulerAngles[2]
    if pitch > 0:
    pitch = 180 - pitch
    elif pitch < 0:
    pitch = -180 - pitch
    yaw = -yaw
    print('id: ' + str(self.i) + ' yaw : ' + str(yaw))
    print('id: ' + str(self.i) + ' pitch:' + str(pitch))
    print('id: ' + str(self.i) + ' roll:' + str(roll))
    cx=int(self.shape[30][0])
    cy=int(self.shape[30][1])
    self.kyori = self.depth_frame.get_distance(cx ,cy)
    #depth と tranlation_vector の距離を合わせている
    ZZ    =    (self.translation_vector[2]**2    +    self.translation_vector[1]**2    +
self.translation_vector[0]**2) - 330.0
    self.A = self.kyori / (math.sqrt(ZZ))
    if self.kyori == 0:
    self.A = self.Aorg
    else:
```

```
    self.Aorg = self.A
    print('id: ' + str(self.i) + ' X:' + str(self.translation_vector[0] * self.A))
    print('id: ' + str(self.i) + ' Y:' + str(-self.translation_vector[1] * self.A))
    print('id: ' + str(self.i) + ' Z:' + str(self.translation_vector[2] * self.A))
    print('========================================')
    self.msg = OscMessageBuilder(address='/data')
    self.msg.add_arg(int(yaw))
    self.msg.add_arg(int(pitch))
    self.m = self.msg.build()
    self.client.send(self.m)
    self.yaw_empty = np.append(self.yaw_empty, yaw)
    self.pitch_empty = np.append(self.pitch_empty, pitch)
    else:
    self.yaw_empty = np.array([])
    self.pitch_empty = np.array([])
    for self.i, d in enumerate(self.face_rects):
    self.shape = self.predictor(self.color_image, self.face_rects[self.i])
    self.shape = face_utils.shape_to_np(self.shape)
    self.image_points = np.float32([self.shape[30], self.shape[8], self.shape[36],
    self.shape[45], self.shape[48], self.shape[54]])
    (self.success, self.rotation_vector, self.translation_vector) = cv2.solvePnP(
    self.model_points, self.image_points, self.camera_matrix, self.dist_coeffs,
    flags=cv2.SOLVEPNP_ITERATIVE)
    (self.nose_end_point2D, self.jacobian) = cv2.projectPoints(np.array([(0.0, 0.0,
1000.0)]),
    self.rotation_vector, self.translation_vector,
    self.camera_matrix, self.dist_coeffs)
    p1 = (int(self.image_points[0][0]), int(self.image_points[0][1]))
    p2 = (int(self.nose_end_point2D[0][0][0]), int(self.nose_end_point2D[0][0][1]))
    cv2.line(self.color_image, p1, p2, (255,0,0), 2)
    cv2.line(self.depth_colormap, p1, p2, (255,0,0), 2)
    for (x, y) in self.shape:
    cv2.circle(self.color_image, (x, y), 1, (0, 0, 255), -1)
    cv2.circle(self.depth_colormap, (x , y), 1, (0, 0, 255), -1)
    #顔器官点(68 点の内 6 点)を frema 濃くに描画
    for p in self.image_points:
    cv2.circle(self.color_image, (int(p[0]), int(p[1])), 3, (0,0,255), -1)
    cv2.circle(self.depth_colormap, (int(p[0]), int(p[1]) ), 3, (0,0,255), -1)
```

```
    self.rvec_matrix = cv2.Rodrigues(self.rotation_vector)[0]
    self.proj_matrix = np.hstack((self.rvec_matrix, self.translation_vector))
    self.eulerAngles = -cv2.decomposeProjectionMatrix(self.proj_matrix)[6]
    yaw     = self.eulerAngles[1]
    pitch = self.eulerAngles[0]
    roll    = self.eulerAngles[2]
    if pitch > 0:
    pitch = 180 - pitch
    elif pitch < 0:
    pitch = -180 - pitch
    yaw = -yaw
    print('id: ' + str(self.i) + ' yaw : ' + str(yaw))
    print('id: ' + str(self.i) + ' pitch:' + str(pitch))
    print('id: ' + str(self.i) + ' roll:' + str(roll))
    cx=int(self.shape[30][0])
    cy=int(self.shape[30][1])
    self.kyori = self.depth_frame.get_distance(cx ,cy)
    #depth と tranlation_vector の距離を合わせている
    ZZ    =    (self.translation_vector[2]**2    +    self.translation_vector[1]**2    +
self.translation_vector[0]**2) - 330.0
    self.A = self.kyori / (math.sqrt(ZZ))
    if self.kyori == 0:
    self.A = self.Aorg
    else:
    self.Aorg = self.A
    print('id: ' + str(self.i) + ' X:' + str(self.translation_vector[0] * self.A))
    print('id: ' + str(self.i) + ' Y:' + str(-self.translation_vector[1] * self.A))
    print('id: ' + str(self.i) + ' Z:' + str(self.translation_vector[2] * self.A))
    print('========================================')
    self.msg = OscMessageBuilder(address='/data')
    self.msg.add_arg(int(yaw))
    self.msg.add_arg(int(pitch))
    self.m = self.msg.build()
    self.client.send(self.m)
    self.yaw_empty = np.append(self.yaw_empty, yaw)
    self.pitch_empty = np.append(self.pitch_empty, pitch)
    else:
    self.msg = OscMessageBuilder(address='/data')
```

```
self.msg.add_arg(int(self.yaw_empty[0]))
self.msg.add_arg(int(self.pitch_empty[0]))
self.m = self.msg.build()
self.client.send(self.m)
def show(self):
self.images = np.hstack((self.color_image, self.depth_colormap))
cv2.imshow("Output", self.images)
def stop(self):
self.pipeline.stop()
cv2.destroyAllWindows()
def main():
myCap = FaceDetector()
while(True):
myCap.detectFace()
myCap.show()
if cv2.waitKey(1) & 0xFF == ord('q'):
break
myCap.stop()
if __name__ == "__main__":
    main()
```

参考文献

[1] 上野佳奈子, “コンサートホールの科学：形と音のハーモニー”, コロナ社, 2012

[2] 光石将隆, 京都大学卒業論文, 2018

[3] 安藤彰男, “音場再現”, コロナ社, 2014

[4] 掌田津那乃, “かんたん Python”, 技術評論社, 2018

[5] クリエイターの共通言語 Max, https://www.mi7.co.jp/products/cycling74/ ,2019 年 1 月

[6] インテル ®RealSense ™ デプスカメラ D435, https://ark.intel.com/ja/products/128255/Intel-RealSense-Depth-Camera-D435 ,2019 年 1 月

[7] TIME DOMAIN mini, http://www.timedomain.co.jp/product/mini.html , 2019 年 1 月

[8] 井上裕翔, 信州大学大学院卒業論文, 2015

[9] 中野学, 一般カメラモデルの PnP 問題に対するグレブナー基底を用いた統一的解法, 画像の認識・理解シンポジウム（MIRU2011）, 2011

[10] Open Sound Control, https://ja.wikipedia.org/wiki/OpenSound_Control, 2019 年 1 月

[11] Kurabayashi Hiroaki, Sound Image Localization Using Dynamic Transaural Reproduction with Non-contact Head Tracking, 電子通信学会 VOL.E97, 2014