RealSense と音響信号を用いた視覚障害者支援デバイスの開発

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 15－93　萩原洋平
指導教官　豊田政弘

目次

1　序論

1.1　研究背景

現在、視覚障害者が外出した場合、自転車や自動車、段差、用水路などの障害が存在する。こうした障害を回避し、歩行するために、白杖や盲導犬の利用、点字ブロック、音響装置付信号機の設置などの工夫がなされている。これらは視覚障害者が周囲の状況を把握するのに必要不可欠なものとなっているが、白杖や盲導犬を利用するためには訓練が必要であり、点字ブロックは自転車などの障害物が置かれてしまうといった問題点が存在する。また、白杖や盲導犬の利用時には片手はふさがった状態である。一方で、音響装置付信号機は、両手が空いた状態での誘導が可能であると言えるが、押ボタンを押さないと音が鳴らないものがあり、また、指向性が低いといった問題点が存在する。そこで、視覚障害者自身が音の再生装置を装着し、「音」によって常に、感覚的に、障害物の距離や位置を把握することが出来れば、両手が空いた状態で活動することができ、それらの問題点が解決できると考えられる。
本研究では、上記実現のために RealSense と音響信号を用いた視覚障害者支援のためのデバイスを開発し、その有用性を検証する。

1.2　既往研究

以下に深度カメラを用いた視覚障害者向けの支援装置に関する既往研究を紹介する。

・コンスタンツ大学での研究[１]

ヘルメットに取り付けた写真１の Kinect によって、壁や障害物への距離を測定し、腰に巻きつけた振動装置で着用者に警告するというシステム。また、壁やドアに貼られたシンボルマークを検出し、距離などを音声で伝えることも可能である。

・高知大学での白状型歩行支援デバイスの開発[２]

腰に取り付けた写真２の Xtion によって、壁や障害物への距離を測定し、両手首に装着した振動装置で着用者に警告するというシステム。また、白状に右、左、中央の 3 つのボタンを取り付け、ボタンが押されると、割りあてられた領域の探索を行い、障害物までの有無と障害物までの歩数を音声によって通知する。

写真１Kinect

写真２Xtion

・久保の研究[3]、古川の研究[4]

○目的

Kinect と音響信号を用い、「音」を使って視覚障害者が物や人の位置を認識し、行動できるようになるためのデバイスを開発する。

○開発されたシステムの概要

Kinect のドライバ、MATLAB、puredata、音の出力ドライバを用い、図１のように障害物を検出し音を提示する。

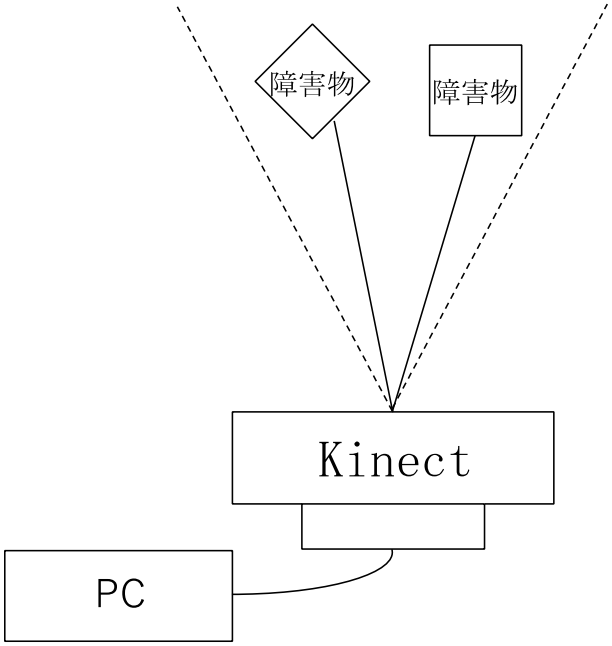

図１システムの概要図

○デバイスの処理手順

・Kinect で 640×480 の深度データを取得
・8×6 のサブセットに分割
・最も深度の小さいサブセットの番号と深度を検出
・図２のサブセットの番号と深度に応じて音源を作成
・骨伝導ヘッドフォンで音を提示

0	6	１２	１８	２４	３０	３６	４２
1	7	１３	１９	２５	３１	３７	４３
2	8	１４	２０	２６	３２	３８	４４
3	9	１５	２１	２７	３３	３９	４５
4	１０	１６	２２	２８	３４	４０	４６
5	１１	１７	２３	２９	３５	４１	４７

図２サブセット番号

○音の鳴り方

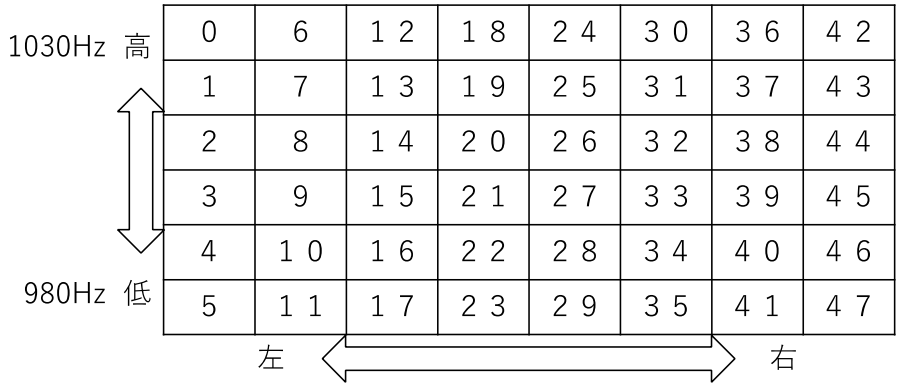

図３サブセット番号による提示音

・図３のようにサブセットの位置の方向からの HRTF を畳み込んだ音を提示する。
・音量は深度が大きくなるにつれて、小さくなる。
・距離によって提示する音のテンポが変わる。
近い(0.4～0.5m)ピピピピピ
中間(0.6～0.7m)ピピピ
遠い(0.8m～)ピピ
・図３のようにサブセットの高さによって提示する音の周波数を変化させる。
0 番の行 1030Hz
1 番の行 1020Hz
2 番の行 1010Hz
3 番の行 1000Hz
4 番の行 990Hz
5 番の行 980Hz

○結果

障害物の位置認識は、左右の判別は比較的容易で、正解率が高いが、上下方向の判別は難しく正解率はかなり低い結果となった。また、初めて使う状態では難しいが、少しデバイスに②慣れる練習を重ねると、ゆっくりと音が鳴る方向を確認しながら進むことで、目隠しをしてデバイスを装着した状態で簡易な迷路を接触せずに通り抜けることも可能であった。

1.3　研究目的

[3][4]の既往研究で作成された Kinect によるデバイスから、新しい機器である RealSense によるデバイスへとシステムを移行することを目的とする。また、既存のシステムを見直し、デバイスの性能を向上させる。最終的には RealSense と音響信号を用い、「音」によって視覚障害者が、より直感的に、また正確に障害物の位置を認識し、行動できるようになるためのデバイスを開発する。次節以降、[4]の既往研究で確認された問題点を挙げ、それに対する改善案を提案する。

1.4　問題点と改善案

1.4.1 問題点

[4]の既往研究では、図４のように片側に障害物がある場合は問題なく通行できるが、図５のように両側に障害物がある場合、通行可能か判断できないという結果が出た。[4]の既往研究で使用しているシステムは、最短の一点を計測し、使用者にその点の位置を音響信号によって知らせるといったものであるため、音が鳴り障害物を避けようとしても音が鳴り続ける、正面方向が空いていても壁だと思い込み通行不可だと判断してしまう。

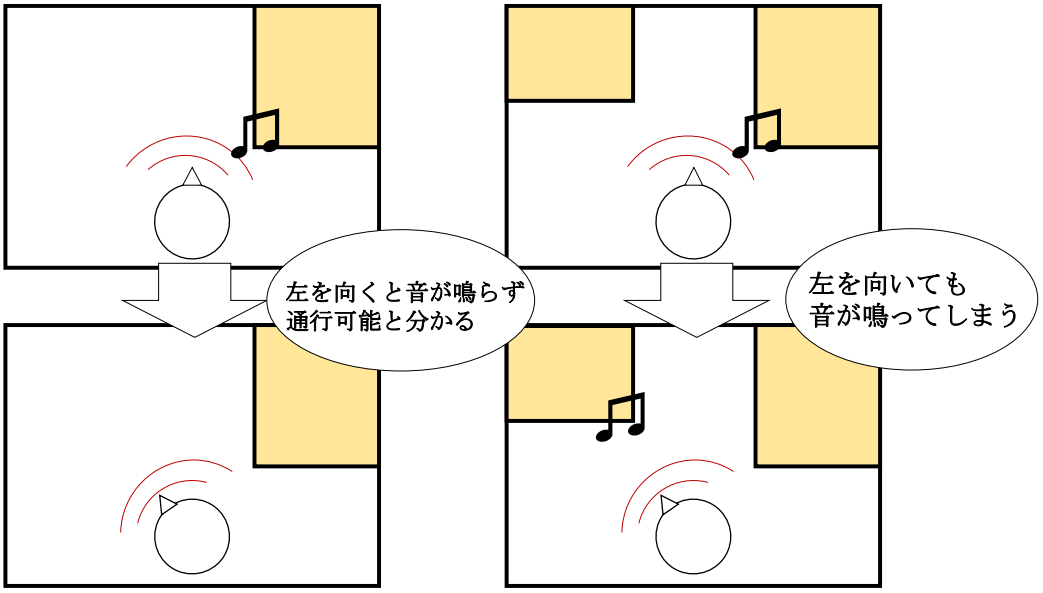

図４片側に障害物がある場合　　図５両側に障害物がある場合

1.4.2 提案

両側に障害物がある場合に通行可能か判断できないのは、デバイスの使用者に、最も近い点の位置のみを通知するため。つまり、使用者は通知された方向に障害物があるということしか分からないためである。そこで、システムを根本的に見直し、正面方向に障害物があるかどうか判断するシステムを作成する。一点ではなく、右、左、中央の３方向を同時に計測し、使用者がより正確に、障害物の位置を把握できるシステムを作成する。

2　研究方法

2.1　開発環境

本研究では、システムの処理には PC を用いる。PC に RealSense、骨伝導ヘッドフォンを接続し、障害物の検知、使用者への通知を行う。また、RealSense との互換性とプログラム作成の簡易性を考慮して、デバイスのためのシステム作成には python を使用する。RealSense と python の接続には Intel 社が公式で配布している「pyrealsense2」というモジュールを使用する。図６に各機器の接続を示し、また以下に使用機器を示す。

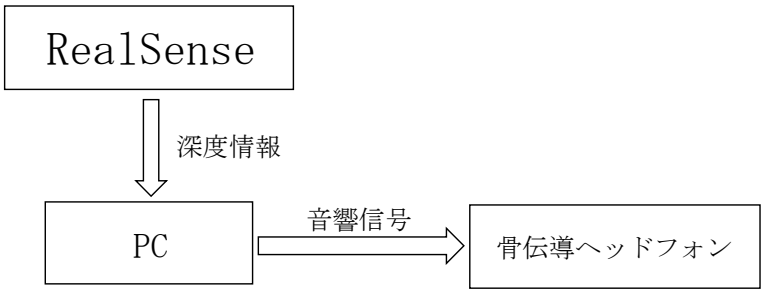

図６各機器の接続

写真３RealSense

写真３が Intel 社の RealSense モデルD435 である。ＲＧＢセンサと赤外線を用いた深度センサが２つ搭載されており、障害物との距離を測ることができる。尚、深度センサの解像度は以下に最高値を記すが、本研究では 480×270 の解像度でシステムの作成を行う。これは、作成したシステムを動かすときの負荷をできるだけ減らすためである。

仕様
・解像度深度センサ／1280×720pixelＲＧＢセンサ／1920×1080pixel
・角度横／85° 縦／58°
・距離計測範囲 0.2～10m
・サイズ 90×25×25mm

写真４骨伝導ヘッドフォン

デバイスの使用者が装着する再生装置は、なるべく使用者の聴覚の妨げにならないように、写真４のような骨伝導ヘッドフォンを使用する。

2.2　デバイスの装着方法

デバイスの装着方法は既往研究の場合腰に装着していたが、RealSense は Kinect と比較してサイズが小さくなっているため、検討の余地があると考えた。以下に装着方法の案を示す。

写真５装着方法案１

写真５はサスペンダーを利用して胸の位置にデバイスを装着した案である。計測位置の高さ、安定感はあるが、左右を計測したい場合、体全体で向きを変えなくてはいけないといった問題点があった。

写真６装着方法案２

写真６は帽子の鍔の部分にデバイスを装着した案である。帽子をかぶるだけのため、装着は簡単だが、計測位置が少し高く、左右を計測したい場合、目の位置より少し高いためか感覚のズレを感じ、使いにくかった。

写真７装着方法案３

写真７はアイマスクにデバイスを装着した案である。帽子の案と使用感はあまり変わらなかったが、装着位置が目の位置と同じため、感覚のズレはなく使いやすかった。
上記に示した三つの案はデバイスを使用する上では問題なかったが、立ち止まって左右を計測したい場合、顔の向きを変えるだけで計測が可能なので、アイマスクにデバイスを装着し、目の位置から計測する方法を採用した。デバイスの使用時も、RealSense はサイズ、重量が小さいため、目の位置に装着しても気にならなかった。

2.3　システム

2.3.1障害物検知

RealSenseによって読み取った深度情報から障害物を検知する。図７のように480×270の解像度の画像の480×270のすべての点の距離を測定し、測定した点のうち、2mより大きい値のものを排除する。

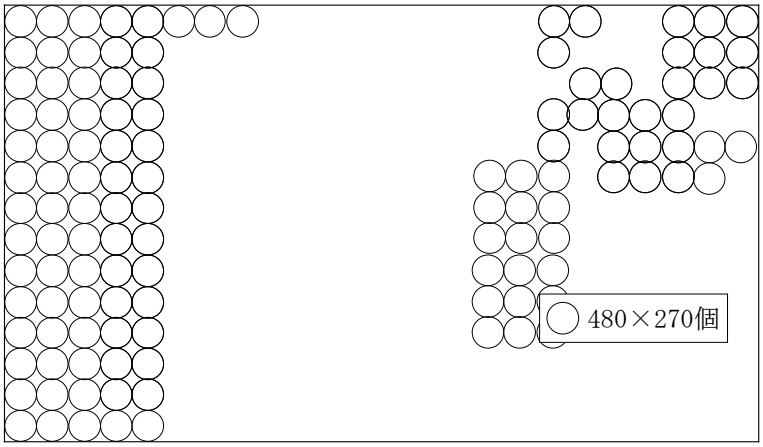

図７深度情報から測定した点

図８のようにRealSenseの撮影した画像を2×3の領域に分け、各領域で残った点が、５割以上の場合、障害物と判定する。

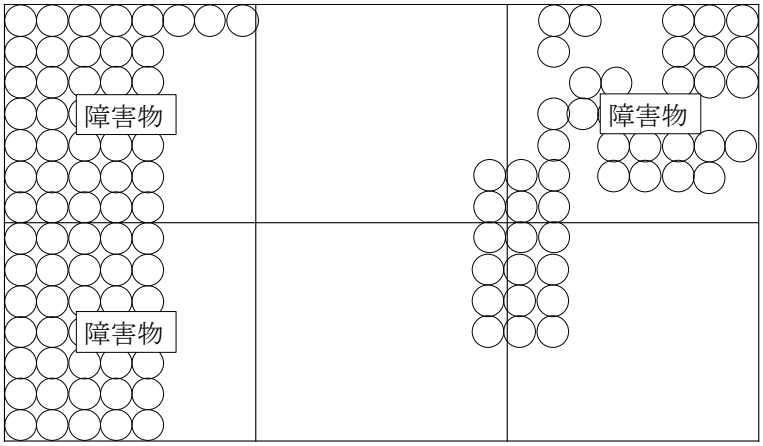

図８各領域での処理

2.3.2 使用者への通知

使用する音源は、左、中央、右の 3 方向と各方向を上下に分割した 2×3 の 6 つの領域分と、障害物と使用者までの距離が近いときに使用する警告音の７種類を用意する。音源は使用者が長時間聴くことになるため、使用者が不快に感じない「ピー」という音を採用した。音源の縦方向は音の高さで表現し、歩行時には上下の判別はそれほど重要でないと判断したため、上と下の 2 方向のみを、上ならばドレミファソラシドのラの音を鳴らし、したならばミの音を鳴らして上のほうが高い音が鳴るようにした。また、使用者への通知は三方向を同時に通知するため、各方向の音の高さが近すぎると判断しづらくなるのを避けるために、右、中央、左の順に 1 オクターブずつ音の高さが下がるようにした。音源の横方向はパニングと呼ばれる、スピーカー間の音量差によって音像定位を表現する方法で通知し、中央方向であれば、左右の音量差を０にし、左端方向であれば、左のみから音源が聞こえるようにし、右端方向であれば、右のみから音源が聞こえるようにした。

使用者への通知の際は、各領域をさらに縦に３分割し、各領域内の 3 つの領域のうち、最も点を占めている領域の中心部に向けて、音源をパニングして通知する。各方向のパニングの割合は、左端を-１、中央を０、右端を+１として図９に領域と併せて示す。右上部と右下部などの、縦が同じ領域の両方で音源を再生する場合、使用者が混乱してしまうのを防ぐために、上部側の音源のみ再生する。

最も近い点の距離が 0.2m 以下の場合、画像を縦に９分割し、点が位置している範囲の中心部に向けて、警告音をパニングして使用者に通知する。使用する音源は 1760 Hzの音を 2 音ピピッといったように鳴らしたもので、パニングの割合は上記に述べたものと同様にする。

表１に使用する音源の種類と周波数を示す。

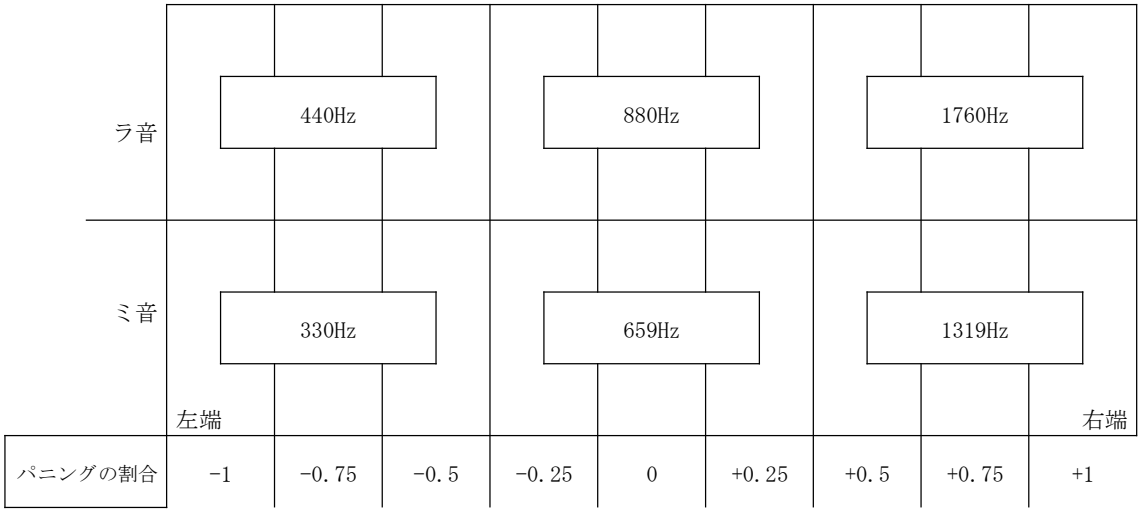

図 9 各領域に対応する音源

表１．音源の種類と周波数

音源の種類	周波数（Hz）
右上部	1760
右下部	1319
中央上部	880
中央下部	659
左上部	440
左下部	330
警告	1760 1760（２音）

2.3.3 処理の分岐

障害物が 0.7mより離れると、近くにある場合と比較して画像に占める割合が減ってしまい、上記のシステムでは障害物の検知の正確性に欠けると感じたため、最短の点が 0.7mより大きい場合、最も近い点の距離が 0.2m以下の場合と同様の処理を行い、1760Hz の音源で使用者に通知するようにし、最も近い点の距離が近づくほど、音源の再生間隔を短くし、使用者に障害物の距離を伝えるようにした。また、最短の点が 0.7m以下の場合でも、全ての領域で、点が５割に満たない場合、音源を一つも再生しなくなるので、その場合は 0.7mより大きい場合と同様の処理を行うようにした。

3　　評価実験とデバイスの改良

3.1　実験内容

開発したデバイスを装着し、下記の実験から有用性を評価する。また、下記の実験において、障害物を正確に検知できるか、使用者への通知は適切か、歩行するのに問題はないかという点に着目する。

実験 1 ホワイトボードに向かって前進し、接触する前に止まる実験。

実験 2 角を曲がる実験。

実験 3 問題に対する改善策についての評価実験を行う。両側に障害物がある通路を通過する実験。

尚、デバイスは、目の高さに装着して使用するため、実験では写真８のような疑似壁を使用する。

写真８疑似壁

3.2　実験結果

実験１ホワイトボードに向かって前進し、接触する前に止まる実験。

写真９のように、スタート時は距離が遠かったので、正面方向から距離が遠い場合の 1076 Hz のピー音が鳴っていた。ホワイトボードに少し近づくと、距離が近い場合の処理に切り替わり、3 方向から左上部、中央上部、右上部の音がなりだした。さらに近づくと、警告音が鳴りだしたので障害物が近いことが分かり、止まることができた。念のため、止まった後左右を見渡してみると、正面方向から音が鳴らなかったため、直進可能であることが分かった。

システムの処理の速度の問題と、警告音が鳴ってから止まると判断して止まるまでの、使用者が判断する時間があることから、速く歩くと障害物に衝突してしまった。ゆっくり歩いた場合は、余裕をもって止まることができた。

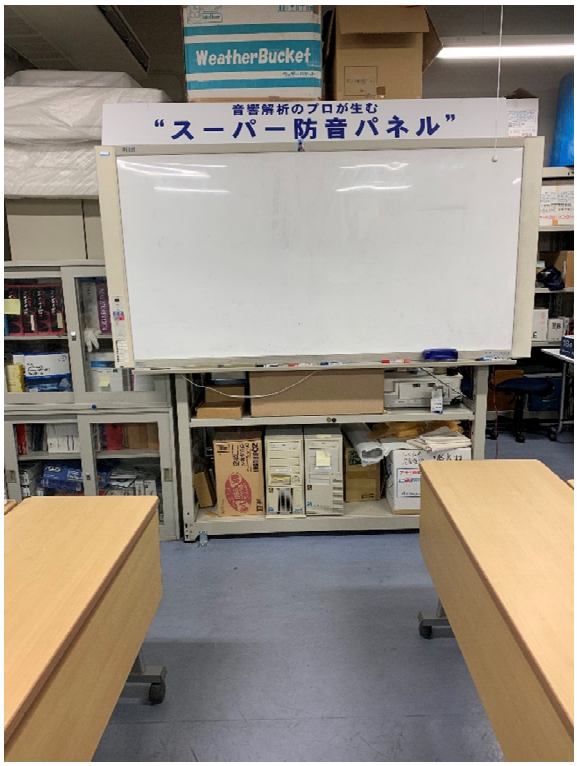

写真９　実験１

実験２角を曲がる実験

　写真１０のようにスタート時から左側に壁があったため、左側から音が鳴っていた。少し歩くと中央からも音が鳴りだし、曲がる段階で、中央から警告音が鳴り、右を向いた場合中央から音が鳴らず、左側から音が鳴ったため、右方向は通行可能であると判断し、曲がることができた。

写真１０　実験２の疑似壁配置

実験 3 両側に障害物がある通路を通過する実験

写真１１のように、スタート時には左右に壁があったため、左右からの音と警告音が鳴っており、中央から音は鳴っていなかった。通行可能か不安であったため、左右を向いてみると、右を向くと左は鳴らず、左を向くと右は鳴らなかったため、中央方向に障害物はないと判断でき、通過することができた。通行可能かの判断はできたが、止まるときや曲がる時と比べて通行するまでに時間がかかった。

写真１１　実験 3 の疑似壁配置

3.3　課題点

2 章で述べたように、最も近い点の距離によってシステムの処理を変えているが、使用者への通知に右上部の音源を使用していたため、障害物の距離が近いか遠いか分からず、混乱してしまう時があった。また、実験 1 において、かなりゆっくり歩かないと警告音が鳴らず、壁にぶつかってしまった。

3.4　デバイスの改良

障害物が遠い場合の通知音のために、新たに、1568Hz と 1319Hz の 2 音を短い間隔で再生する音源を作成した。また、警告音が鳴るシステムを、最も近い点が 0.2m以内の場合になるよう設定していたが、これを 0.4m以内に変え、より早く使用者に警告するようにした。

この改良に関しても同様の評価実験を行った。評価として、障害物が遠くにある場合と近くにある場合の違いが明確になり、障害物の距離がより正確につかめるようになった。また、速い歩行はまだ無理だったが、ある程度の速さでの歩行は許容できるようになった。

4　　総括

4.1　まとめ

RealSense と音響信号を用いた視覚障害者支援デバイスを実用化させていくには、障害物の認識の制度をより向上させていくことが求められており、まだまだ改良すべき点が多々ある。

本研究では[3][4]の研究で開発された Kinect を用いたデバイスを、RealSense を用いたデバイスへとシステムを移行し、またデバイスの性能の向上、2 物体間の通行の問題を解決することができた。

4.2　今後の展開

本研究では、平面上の簡単な通行のみを想定しているため、そのほかに、段差や自分以外の通行人を検知して使用者に通知するシステムを検討するべきである。また、使用者に通知する音声の種類の増加、音声による伝達なども検討すべきである。

参考文献

［１］“ウェアラブル Kinect ナビ“、
URL(https://japanese.engadget.com/2011/03/28/kinect/)2019 年 1 月 29 日閲覧

［２］篠原克麻、森雄一郎、“視覚障害者のための白杖型歩行支援デバイスの開発”、高知大学卒業論文、2016 年、URL(http://trick.is.kochi-u.ac.jp/Vol08/article01.html)2019 年 1 月 29 日閲覧

［３］久保卓真、“kinect と音響信号を用いた視覚支援デバイスの開発”、関西大学卒業論文、2013 年

［４］古川晃司、“kinect と音響信号を用いた視覚支援デバイスの性能向上”、関西大学卒業論文、2014 年

```
Import pyrealsense2 as rs       #モジュールの読み込み
Import time
Import cv2
Import numpy as np
From pydub import AudioSegment
From pydub.playback import play

Class sound_devise():
Def __init__(self):     #realsense の設定

self.pipeline=rs.pipeline()
self.config=rs.config()
self.config.enable_stream(rs.stream.depth,480,270,rs.format.z16,30)

self.profile=self.pipeline.start(self.config)

def setting(self):      #frame の取得
self.frames=self.pipeline.wait_for_frames()
self.depth_frame=self.frames.get_depth_frame()
self.depth_image=np.asanyarray(self.depth_frame.get_data())

self.depth_colormap=cv2.applyColorMap(cv2.convertScaleAbs(self.depth_image,alpha=0.03)
,cv2.COLORMAP_JET)

def devise(self):       #カメラの処理
mute=AudioSegment.from_wav('mute.wav')          #音源の読み込み
alert=AudioSegment.from_wav('alert.wav')
dst=AudioSegment.from_wav("distant.wav")
dl=AudioSegment.from_wav("down_low.wav")
dh=AudioSegment.from_wav("down_high.wav")
ul=AudioSegment.from_wav("up_low.wav")
uh=AudioSegment.from_wav("up_high.wav")
dm=AudioSegment.from_wav("down_middle.wav")
um=AudioSegment.from_wav("up_middle.wav")
cov1a=[]
cov1b=[]
cov1c=[]
cov2a=[]
cov2b=[]
cov2c=[]
cov3a=[]
cov3b=[]
cov3c=[]
cov4a=[]
cov4b=[]
cov4c=[]
cov5a=[]
cov5b=[]
cov5c=[]
```

```
cov6a=[]
cov6b=[]
cov6c=[]
self.minimum=10.0**10.0
def rg_devise(covup1,covup2,covup3,maxa,maxb,maxc,sound1,deg1,deg2,deg3,
covdown1,covdown2,covdown3,sound2,deg4,deg5,deg6):        #範囲でのパニング処理
if len(covup1)+len(covup2)+len(covup3)>=maxa*maxb/maxc: #上判定用プログラム
if covup1>=covup2andcovup1>=covup3:
panned=sound1.pan(deg1)
elif covup2>=covup1andcovup2>=covup3:
panned=sound1.pan(deg2)
elif covup3>=covup2andcovup3>=covup1:
panned=sound1.pan(deg3)
else:
if len(covdown1)+len(covdown2)+len(covdown3)>=maxa*maxb/maxc:    #下判定用プログラム
if covdown1>=covdown2andcovdown1>=covdown3:
panned=sound2.pan(deg4)
elif covdown2>=covdown1andcovdown2>=covdown3:
panned=sound2.pan(deg5)
elif covdown3>=covdown2andcovdown3>=covdown1:
panned=sound2.pan(deg6)
else:
panned=mute
play(panned)

def ps_devise(sound):    #一点でのパニング処理
if self.xm<=50:
deg=-1.0
elif 50<=self.xm<=100:
deg=-0.75
elif 100<=self.xm<=150:
deg=-0.5
elif 150<=self.xm<=200:
deg=-0.25
elif 200<=self.xm<=280:
deg=0.0
elif 280<=self.xm<=330:
deg=+0.25
elif 330<=self.xm<=380:
deg=+0.5
elif 380<=self.xm<=430:
deg=+0.75
elif 430<=self.xm<=480:
deg=+1.0

panned=sound.pan(deg)
play(panned)

if 1.7<self.minimum<=2: #再生間隔調整
time.sleep(2.0)
elif 1.4<self.minimum<=1.7:
```

```
time.sleep(1.5)
elif 1.1<self.minimum<=1.4:
time.sleep(1.0)
elif 0.7<self.minimum<=1.1:
time.sleep(0.5)

def camera(x1,x2,y1,y2,cov):     #最短距離測定
for xinrange(x1,x2):
for yinrange(y1,y2):
dist=self.depth_frame.get_distance(x,y)
if dist==0ordist>2:
continue
elif dist<self.minimum:
self.minimum=dist
self.xm=x
self.ym=y
if dist<=0.7:
cov.append(dist)

camera(1,53,1,135,cov1a)
camera(53,106,1,135,cov1b)
camera(106,159,1,135,cov1c)
camera(159,213,1,135,cov2a)
camera(213,267,1,135,cov2b)
camera(267,321,1,135,cov2c)
camera(321,374,1,135,cov3a)
camera(374,427,1,135,cov3b)
camera(427,480,1,135,cov3c)
camera(1,53,135,270,cov4a)
camera(53,106,135,270,cov4b)
camera(106,159,135,270,cov4c)
camera(159,213,135,270,cov5a)
camera(213,267,135,270,cov5b)
camera(267,321,135,270,cov5c)
camera(321,374,135,270,cov6a)
camera(374,427,135,270,cov6b)
camera(427,480,135,270,cov6c)

print(round(self.minimum,2),self.xm,self.ym)
cv2.circle(self.depth_colormap,(self.xm,self.ym),5,(255,255,255),1)       #最短距離に赤
点表示
if self.minimum<=0.7:
if
len(cov1a)+len(cov1b)+len(cov1c)<=159*135/2andlen(cov2a)+len(cov2b)+len(cov2c)<=162*13
5/2andlen(cov3a)+len(cov3b)+len(cov3c)<=159*135/2andlen(cov4a)+len(cov4b)+len(cov4c)<=
159*135/2andlen(cov5a)+len(cov5b)+len(cov5c)<=162*135/2andlen(cov6a)+len(cov6b)+len(co
v6c)<=159*135/2:
ps_devise(dst)
rg_devise(cov1a,cov1b,cov1c,159,135,2,ul,-1.0,-0.75,-0.5,cov4a,cov4b,cov4c,dl,-1.0,-
0.75,-0.5)
rg_devise(cov2a,cov2b,cov2c,162,135,2,um,-0.25,0.0,+0.25,cov5a,cov5b,cov5c,dm,-
```

```
0.25,0.0,+0.25)
rg_devise(cov3a,cov3b,cov3c,159,135,2,uh,+0.5,+0.75,+1.0,cov6a,cov6b,cov6c,dh,+0.5,+0.
75,+1.0)

elif 0.7<self.minimum<=2:
ps_devise(dst)

if self.minimum<0.4:
ps_devise(alert)

cov1a.clear
cov1b.clear
cov1c.clear
cov2a.clear
cov2b.clear
cov2c.clear
cov3a.clear
cov3b.clear
cov3c.clear
cov4a.clear
cov4b.clear
cov4c.clear
cov5a.clear
cov5b.clear
cov5c.clear
cov6a.clear
cov6b.clear
cov6c.clear

def border(self):        #画像に線を描画
cv2.line(self.depth_colormap,(53,1),(53,480),(255,255,255),thickness=1,lineType=cv2.LI
NE_4)
cv2.line(self.depth_colormap,(106,1),(106,480),(255,255,255),thickness=1,lineType=cv2.
LINE_4)
cv2.line(self.depth_colormap,(159,1),(159,480),(255,255,255),thickness=1,lineType=cv2.
LINE_4)
cv2.line(self.depth_colormap,(213,1),(213,480),(255,255,255),thickness=1,lineType=cv2.
LINE_4)
cv2.line(self.depth_colormap,(267,1),(267,480),(255,255,255),thickness=1,lineType=cv2.
LINE_4)
cv2.line(self.depth_colormap,(321,1),(321,480),(255,255,255),thickness=1,lineType=cv2.
LINE_4)
cv2.line(self.depth_colormap,(374,1),(374,480),(255,255,255),thickness=1,lineType=cv2.
LINE_4)
cv2.line(self.depth_colormap,(427,1),(427,480),(255,255,255),thickness=1,lineType=cv2.
LINE_4)
cv2.line(self.depth_colormap,(1,135),(480,135),(255,255,255),thickness=1,lineype=cv2.L
INE_4)
def window(self):
cv2.imshow("Output2",self.depth_colormap)
```

```
def stop(self):          #デバイスの終了プログラム
self.pipeline.stop()
cv2.destroyAllWindows()

def main():     #関数をまとめて実行

myCap=sound_devise()

while(True):
myCap.setting()
myCap.devise()
myCap.border()
myCap.window()
if cv2.waitKey(1)&0xFF==ord('q'):        #「Qキー」を押したらプログラムを終了
break

myCap.stop()
if __name__=="__main__":
main()
```