2021 年度（令和 3 年度）

創造工学セミナーII Final Report

音声認識で操作する
先導型案内ロボットの開発

指導教員　：　金丸　隆志　教授

チームメンバー

S5-17045 細川 森平

S5-18026 佐藤 涼平

S5-18033 関本 崇文

S5-18056 松本 幸寛

目次

第 1 章　緒言（松本担当）

1.1　研究背景

本節では、2020 年より蔓延している新型コロナウイルスの感染状況についてはじめに説明する。そして、新型コロナウイルスの感染拡大が与えるロボット市場への影響について示す。

1.1.1　新型コロナウイルスの蔓延

新型コロナウイルスは、2020 年の蔓延をきっかけに新規感染者が後を絶たない状況となっている。下記の図 1.1 は、2020 年初頭から 2022 年初頭までの全国における一日あたりの感染者数の合計を示したグラフである。グラフに注目すると、国内の感染者数は全体を通して増減を繰り返しているが、ピーク時の一日あたりの感染者数は年月が経つごと増えていることが分かる。また、このグラフの傾向から今後も感染の規模が大きくなっていくことが予想される。

図 1.1　一日あたりの全国の感染者数の推移［1］

1.1.2　感染拡大とロボット市場の関係性

新型コロナウイルスの蔓延は、ロボット市場にも影響している。実際、国内におけるサービスロボット市場は、コロナウイルスが蔓延する 2019 年度から蔓延後の 2021 年の 2 年間で 2 倍以上に増加している(図 1.2 参照)。その要因として、病院や商業施設での消毒・殺菌ロボットの需要急増や、飲食店での配膳ロボットの急速な普及がある。このことから、今まで人が行なっていた業務を、感染対策として代わりにロボットに行わせる動きが強まっていることが分かる。また、現在も感染が拡大していることから、今後もさらに市場規模が大きくなることが予想されている。

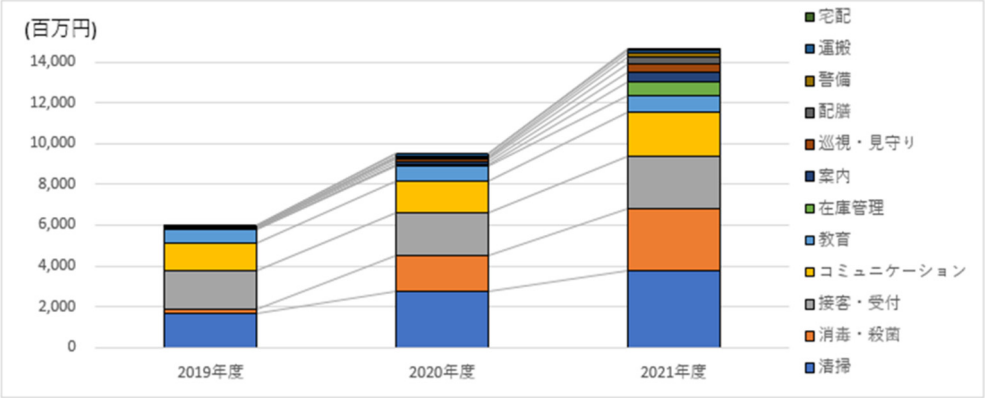

図 1.2　国内におけるサービスロボットの市場推移 [2]

1.2　研究目的

需要が年々増加しているサービスロボットの中で、私たちは屋内施設の案内業務を行うロボットに着目した。一例として、図 1.3 はコロナ蔓延後に駅構内に試験導入された案内ロボットの写真である。

図 1.3　駅案内ロボット(奈良市: 近鉄大和西大寺駅)[3]

図 1.3 の案内ロボットは、2020 年にオムロングループが接客業務の負荷軽減を目的として開発したものである。このロボットの特徴は、高い音声処理技術を用い、音声対話による案内業務を行えるところである。この機能によって、人同士の接触を避けながら案内業務を行えるため、感染対策として注目が集まった。

しかし、この案内ロボットは自律移動ロボットではないため、案内する場所まで直接誘導することができない。そのため、地図が読めない人や音声案内だけでは場所を把握できない人にとっては、目的地まで案内をすることができない可能性がある。

そこで、自律移動ロボットに音声認識機能を搭載させることを考案した。これにより、目的地まで先導して案内を行える機能も両立させることで、案内ロボットとしての利便性の向上に繋がると考えた。

以上の理由から、音声認識で操作する先導型案内ロボットの開発を本研究の研究目的とする。

1.3　研究概要

1.3.1　開発する案内ロボットについて

本研究で開発する案内ロボットの条件は下記の通りである。

・使用する環境は、駅構内や博物館、デパートなどの屋内施設とする。
・自律移動ロボットが直接目的地まで移動できるものとする。
・案内できる場所を複数設定できるものとする。
・音声認識機能を搭載し、音声で行先(目的地)を選択できるものとする。

使用環境として屋内施設を想定したのは、コロナウイルスの影響で屋内では人同士の接触が避けられているためである。また、音声認識の操作のみで案内を行えるようにするのは、ロボットに直接触れずに操作できるようにするためである。

1.3.2　ナビゲーションの流れ

1.3.1 項の条件を基に、以下の流れで案内(ナビゲーション)を行えるロボットの開発を目指す。

まず、ロボットにあらかじめマップデータを登録させる。この際、図 1.4 のようにマップデータ上に複数の目的地を登録し、選択できるようにする。

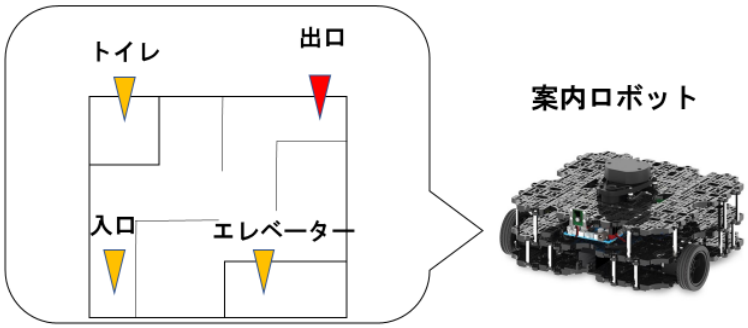

図 1.4　マップデータと目的地の登録

次に、登録した目的地を音声で選択できるようにする。その後、ロボットは選択された目的地まで、マップデータを用いて移動する。これにより、案内ロボットを扱うユーザーは音声で目的地を伝えた後、移動するロボットを追従することで目的地まで行くことが可能となる(図 1.5 参照)。

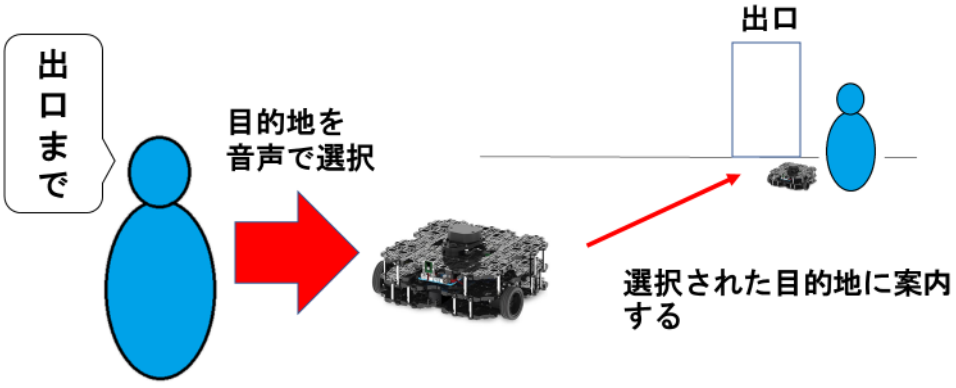

図 1.5　音声入力を用いた目的地までのナビゲーション

第２章　使用機材とソフトウェア（細川担当）

2.1　ROS について

本実験では、ロボットを制御するために Robot Operating System(以下 ROS)を使用する。ROS とは、ロボットを動かすための処理システムの 1 つであり、大学や研究機関といった学術分野をはじめ、産業界や趣味分野まで幅広く利用されているところが特徴である。用いることのできるパッケージの数が多く、Rviz などのシミュレーターツールによるロボット開発が可能という利点がある。Rviz とは ROS 上の 3 次元データの可視化を行うツールであり、ロボットの位置や姿勢、センサが認識した周囲の障害物を PC 画面上に表示することが可能である。また ROS ではロボットの自律制御だけではなくマウスとキーボードによるロボットの手動操作も可能である。

2.2　使用機材

本研究において主要となる機材は以下の通りである。

2.2.1　TurtleBot3 waffle pi

TurtleBot3 waffle pi は ROS をインストールした PC で操作することが可能な移動ロボットである（以下 TurtleBot と呼称する)。レーザーセンサやカメラによって周囲の状況を把握することが出来る。

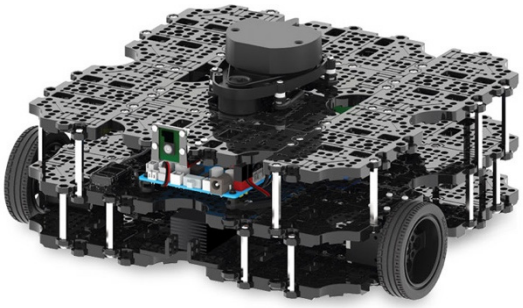

図 2.1　TurtleBot3 waffle pi [4]

2.2.2　ノート PC

Ubuntu を導入した PC を図 2.2 で示す。これは TurtleBot の制御に必要となる。Ubuntu とは Linux を母体としたオペレーティングシステムであり、フリーソフトウェアとして提供されている。先述の ROS はこの Ubuntu 上で動作するソフトウェアである。

図 2.2　Ubuntu を導入した PC

2.2.3　バッテリー(LB-012)

TurtleBot に附属のバッテリーを図 2.3 に示す。型番は LB-012 である。容量は 1800mAh あり、過充電防止、過放電防止の保護回路も搭載されている。本体から取り外し、附属のアダプターで充電する。

図 2.3　LB-012 LIPO Battery 1800mAh [5]

2.2.4　Raspberry Pi

Raspberry Pi はシングルボードコンピュータと呼ばれるハードウェアであり、TurtleBot に搭載される（図 2.4)。Ubuntu と ROS をインストールすることで、ノート PC と通信しロボットの制御を行なえる。

図 2.4　Raspberry Pi [6]

2.2.5　OpenCR

OpenCR とは TurtleBot に搭載されているマイクロコントローラ搭載ボードである（図 2.5)。OpenCR はオープンハードウェアとソフトウェアから構成されており、ROS を使用した組み込みシステム向けに作られている。
ロボットの基本的な動作はこのボードを介して行われる。

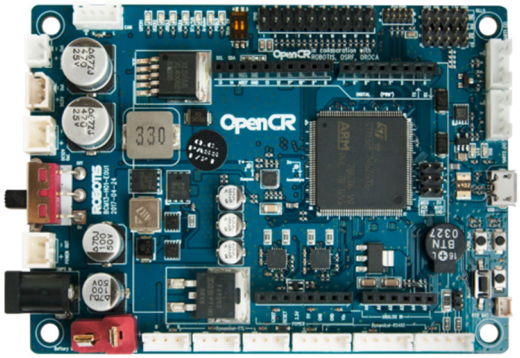

図 2.5　OpenCR [7]

2.2.6　レーザースキャナー（LDS-01）

TurtleBot に附属する 2D レーザースキャナーを図 2.6 に示す。型番は LDS-01 である。スキャナーの周囲 360 度を検知可能である。これを利用することで、自己位置推定やマッピングが可能になり、さらにナビゲーションを行うことも出来る。

図 2.6　レーザースキャナー LDS-01 [8]

2.2.7　RealSense

RealSense は、通称デプスカメラと呼ばれるカメラベースのセンサである。今回はそのシリーズの中でも下記の 2 機種を併用する。

- **RealSense D435**

RealSense D435 は通常のカメラ同様の RGB 映像の取得に加えて、ステレオビジョンによる距離計測も可能なカメラである（図 2.7)。本研究では先述のレーザースキャナーと比較する目的で用いる。

図 2.7　RealSense D435 [9]

- **RealSense T265**

RealSense T265 は､2 対の魚眼カメラと IMU(加速度センサ)を搭載したセンサである（図 2.8)。RealSense D435 と併用することで、ロボットの自己位置推定が可能になる。

図 2.8　RealSense T265 [10]

なお、この項ではレーザースキャナーと RealSense の 2 機種を記載したが、最終的に案内ロボットに搭載したものはレーザースキャナーのみである。その理由や、2 機種の性能比較に関しては次の 3 章で述べる。

第 3 章　レーザースキャナーと RealSense の比較（細川担当）

3.1　概要

3.1.1　自己位置推定について

今回製作する案内ロボットを動作させるために最も重要な要素が自己位置推定と呼ばれる機能である。自己位置推定とは、ロボットが使用する環境内においてどの位置に居るのかを把握する機能である。ロボットによるナビゲーションや地図作成を行うにはこの機能が必須であり、その精度が高ければ高いほど、正確なナビゲーションを行うことができる。この移動ロボットのために用意したセンサは、先述のレーザースキャナーと RealSense の 2 種類である。通常、このような移動ロボットにはレーザースキャナーか、RealSense のようなステレオカメラ(デプスカメラ)のどちらかがが使用されることが多い。この章では今回作成する案内ロボットにおいて、どちらのセンサを搭載するべきか、その精度を比較検証して決める。

3.1.2　一般的なレーザースキャナーの特徴

レーザースキャナーとは、スキャナーから照射されたレーザー光によって、対象物の空間位置情報を取得する計測センサである。機種によって認識範囲は異なるが、概ね広範囲の認識が可能であることが特徴である。3 次元スキャナーと 2 次元スキャナーが存在するが、今回使用するのは 2 次元スキャナーであり、周囲 360 度認識可能なものである。図 3.1 に示したのは実用化されているレーザースキャナーの例である。

図 3.1　様々なレーザースキャナー

3.1.3　一般的なステレオカメラの特徴

ステレオカメラとは、２つのレンズにより、対象までの距離計測が可能なセンサのことである。カメラの映像の範囲内が認識範囲である。大きな特徴としては、人間の距離認識と近い原理で動作しているという点である。なお今回利用する RealSense はこのステレオカメラよりもより高性能なデプスカメラと呼ばれるもので、ステレオビジョンによる、より高度な深度測定に加え、赤外線プロジェクタや、ジャイロセンサなどが搭載されている。
図 3.2 で示すものは実用化されているステレオカメラの例である。

図 3.2　様々なステレオカメラ

3.2　実験に使用したセンサ

実験に使用したセンサは以下の通りである。

- レーザースキャナー　LDS-01　　（2 章　使用機材　参照）
- RealSense D435 および T265　（2 章　使用機材　参照）

3.3　比較実験に使用したロボット

3.3.1　レーザースキャナー搭載ロボット

レーザースキャナーを搭載したロボットを図 3.3 に示す。赤丸部分がレーザースキャナーである。レーザーの認識範囲は図 3.4 で示した通り周囲 360 度である。また、レーザースキャナー搭載ロボットは TurtleBot 本体の車輪の回転数や回転角からその移動量を読み取ることで自己位置推定を行う「オドメトリ法」も併用している。このオドメトリ法は、RealSense では T265 と同等の役割を果たすものである。

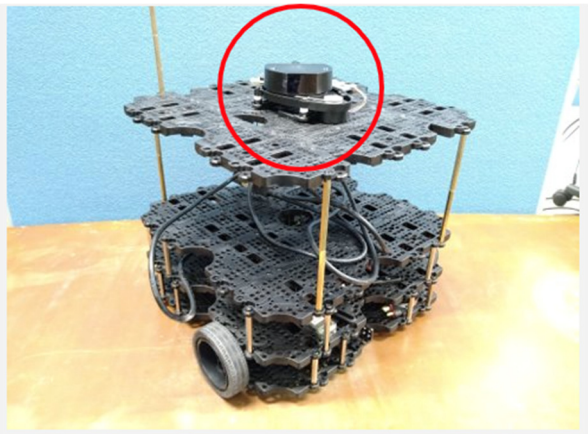

図 3.3 レーザースキャナーを搭載した TurtleBot

図 3.4　レーザースキャナーの認識範囲

3.3.2　RealSense 搭載ロボット

RealSense を搭載したロボットを図 3.5 に示す。赤丸で示した部分が RealSense である。今回は D435 と T265 を併用する。RealSense の認識範囲は図 3.6 で示した通りカメラの映像範囲内である。今回の実験ではレーザースキャナーと高さをそろえるために固定位置を高くしている。

図 3.5　RealSense を搭載したロボット

図 3.6　RealSense の認識範囲

3.4　実験環境

レーザースキャナー搭載ロボットと RealSense 搭載ロボットを比較するにあたり、２台のロボットにそれぞれ簡易的な地図を作成させることにした。それらを比較し、より正確な地図を作成できたものを自己位置推定の精度が高いと考えるものとする。

今回は研究室内で簡易的な５つのパターンの環境を用意し、レーザースキャナー搭載ロボットと、RealSense 搭載ロボットの２つを使用してそれぞれ地図を作成していく。

地図作成には、レーザースキャナーでは ROS の gmapping パッケージを用いる。一方、RealSense では下記の occupancy パッケージを用いる

https://github.com/IntelRealSense/realsense-ros/tree/occupancy-mapping/occupancy

図 3.7 は用意した環境のイメージ図であり、図 3.8 はそれぞれに対応する実験環境の実際の写真である。黄色い丸はロボット本体を示しており、これを矢印の通りに移動させて地図を作成する。ロボットの操作は ROS（2.1 項目参照）を介してキーボード操作にて行う。また、レーザースキャナーと RealSense で認識範囲に大きな差があるため、同じ環境でそれぞれ３往復（または３周）ずつ移動させて比較する。比較には１往復目と３往復目の結果を比較する。

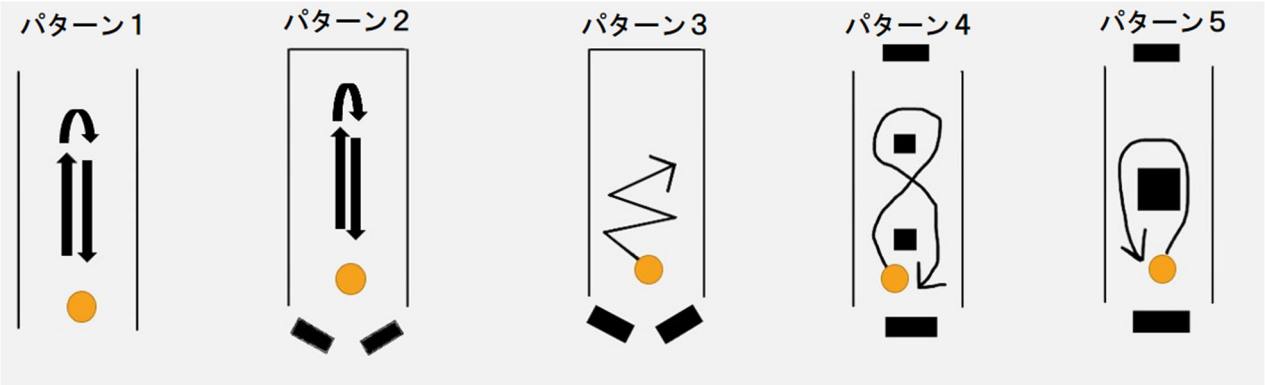

図 3.7　実験環境イメージ図

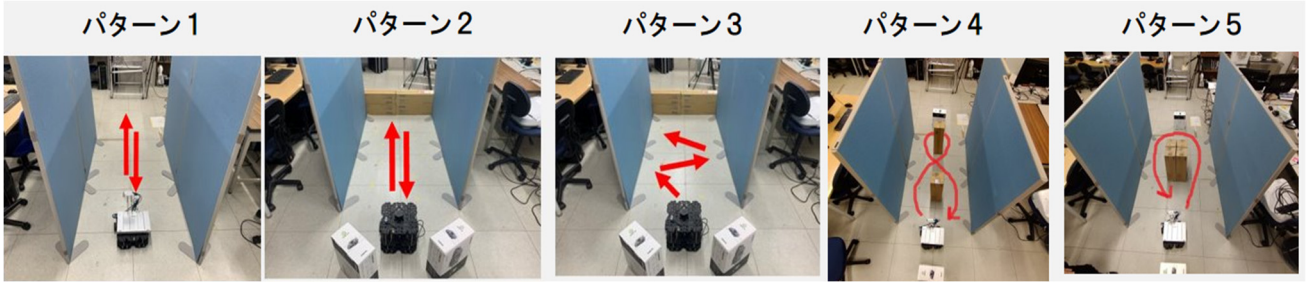

図 3.8　実際に用意した実験環境

3.5　実験結果

3.5.1　パターン 1

図 3.9 に示したパターン 1 は、衝立を 2 枚使って細い通路を再現したものである。ロボットは図 3.9 の写真手前から奥に進み、U ターンしてまた手前に戻ってくる動作をさせた。

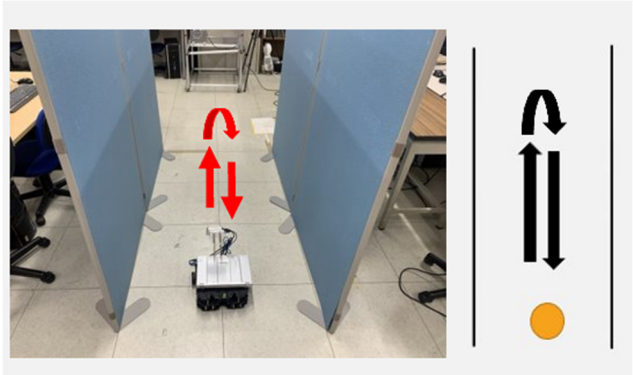

図 3.9　パターン 1

下記の図 3.10 がロボットにより作成された地図である。上段がレーザースキャナー搭載ロボットで作成した地図、下段が RealSense 搭載ロボットで作成した地図であり、左側が 1 往復目で作成された地図、右が三往復目で作成された地図である。　RealSense で作成した図に注目してみると、図の中の赤い丸で囲った部分（RealSense で作成した 3 往復目の画像）は、壁が二重に認識されてしまっている。考えられる原因としては、レーザースキャナーと違い認識範囲が狭く、正面以外（ロボットの側面）の認識の精度が悪くなってしまったためと考えられる。

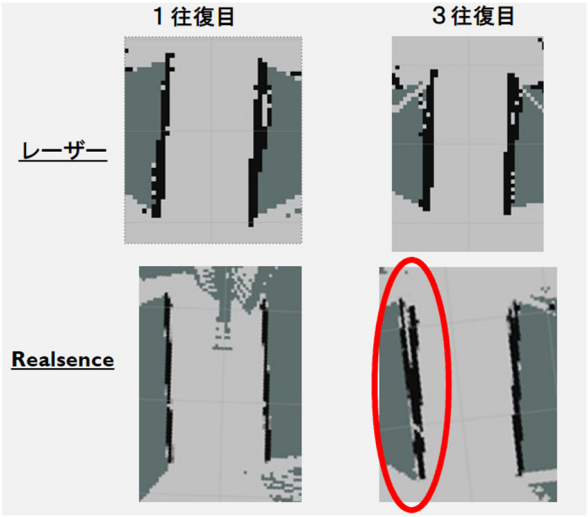

図 3.10　ロボットが作成した地図（パターン 1）

3.5.2　パターン２

図 3.11 に示したパターン２は先述のパターン１の衝立の奥側をふさぎ、手前側に障害物として段ボールを２個置いたものである。これはパターン１よりもロボットが認識できる障害物を増やし、それによって壁の認識精度が向上するかどうかを調べる意図で設定したパターンである。

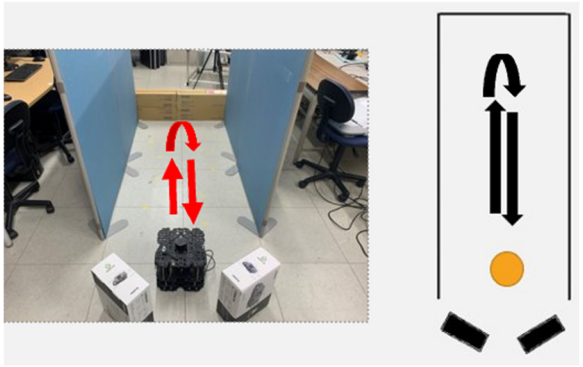

図 3.11　パターン２

図 3.12 で示した図が作成した地図である。こちらも赤丸部分（RealSense で作成した３往復目の地図）に注目すると、やはり壁がずれてしまっていることがわかる。しかし、ロボットに対して正面に位置する壁や、障害物は正しく認識できている。このことから RealSense は、側面の認識精度は悪いものの、正面の認識精度は悪くないということがわかる。

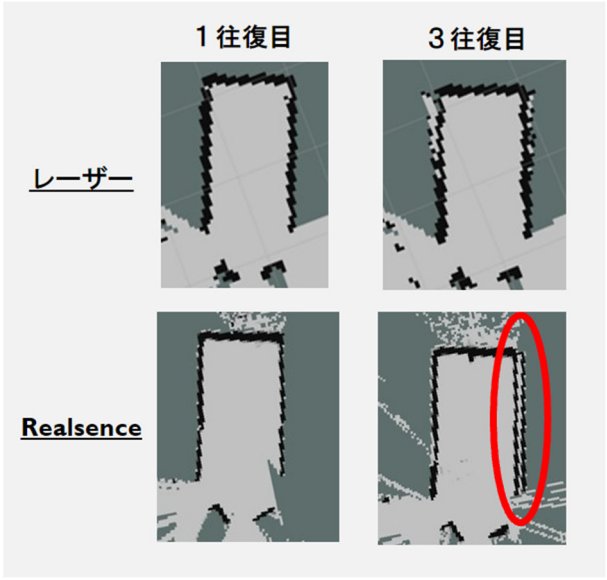

図 3.12　ロボットが作成した地図（パターン２）

3.5.3　パターン 3

図 3.13 に示したパターン 3 は、パターン 2 と同様の配置で、ロボット自体の動きを細かく屈折させたものである。これにより、細かいロボットの動きにセンサが追従できるかどうかを調べる。さらに RealSense では、パターン 2 で側面に位置していた壁を正面に捉えることで精度が向上するのかも調べる。

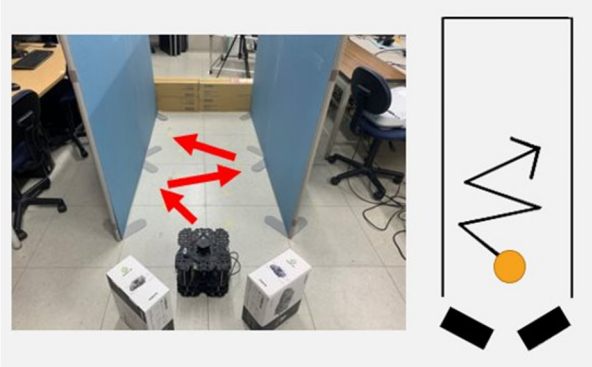

図 3.13　パターン 3

図 3.14 で示した図が作成した地図である。RealSense で作成した地図はずれがなく、正確に壁が地図上に生成されている。逆に、レーザースキャナーで作成した地図は、少量ではあるが、壁にずれが生じている。　レーザースキャナーの精度が悪化した原因としては、細かな動きが多く、オドメトリが不正確になってしまったためと考えられる。逆に、RealSense の精度が向上した原因としては、側面に位置していた壁を正面に捉えることが出来るようになったためと考えられる。

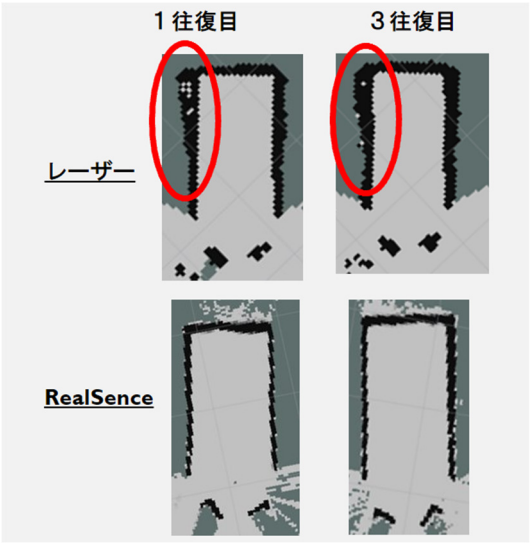

図 3.14　ロボットが作成した地図（パターン 3）

3.5.4　パターン４

図 3.15 に示したパターン４は、衝立の間に角柱状の障害物を２個設置したパターンである。その間を８の字にロボットを動かしていく。

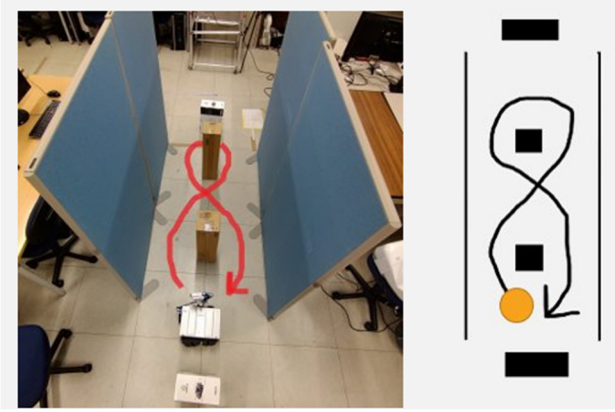

図 3.15　パターン４

図 3.16 で示した図が作成した地図である。赤丸部分に注目してみると、RealSense で作成した地図は、衝立間にある障害物がうまく認識できていないという事がわかる。これもやはり RealSense のロボット側面の認識精度の悪さが原因となっていることが考えられる。

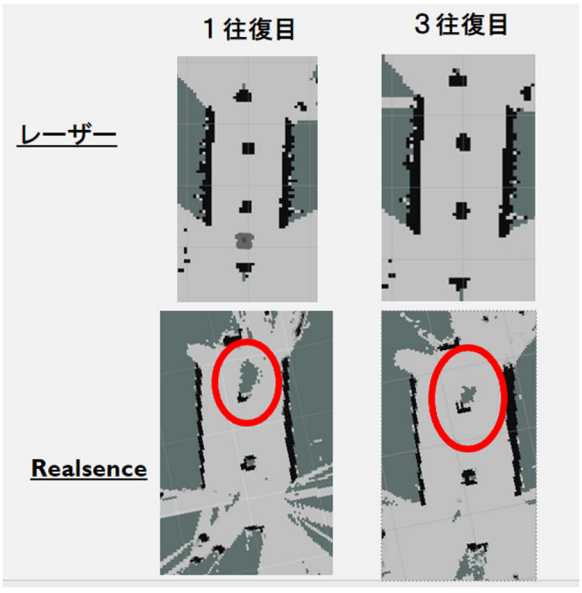

図 3.16　ロボットが作成した地図（パターン４）

3.5.5　パターン 5

図 3.17 に示したパターン 5 は、一個の大きな障害物を衝立間に置いたパターンである。その障害物の周りを一周するようにロボットを動かした。

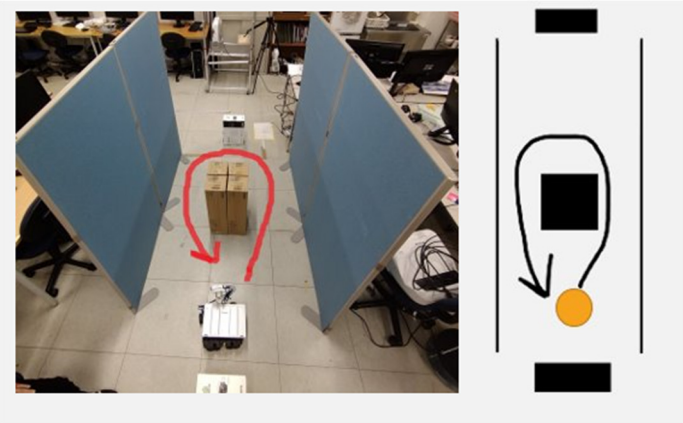

図 3.17　パターン 6

図 3.18 に示したのが作成した地図である。こちらも先のパターン 4 と同様に、RealSense では障害物を正確に認識出来ていない事がわかる。

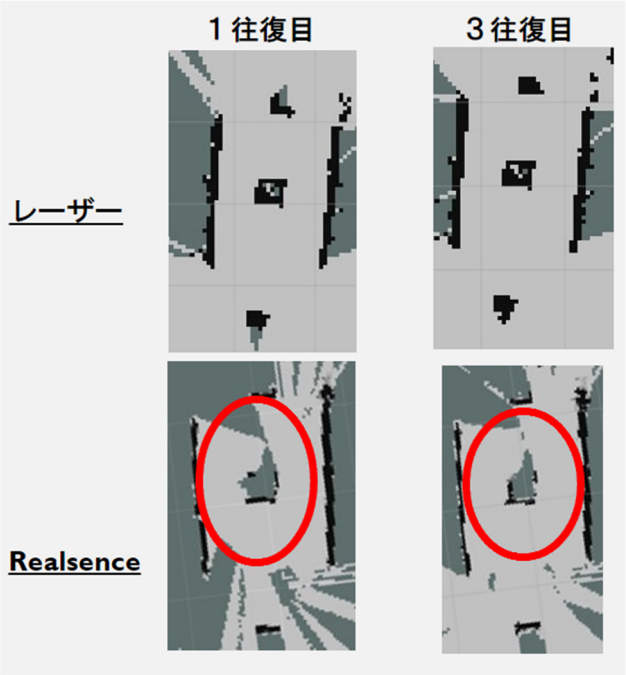

図 3.18　ロボットが作成した地図（パターン 5）

3.6　比較結果

今回５つのパターンを用意して比較検証を行ったが、総合的に見ると、レーザースキャナーで作成した地図の方が、RealSense で作成した地図よりも安定して正確に描画できていることが分かった。大きな要因としては、レーザースキャナーの認識範囲の広さにあり、周囲360 度認識可能であるという利点が生かされた結果と言えるだろう。逆に、レーザースキャナーと比べて認識範囲の狭い RealSense は、ロボットに対して側面に位置する障害物の認識精度が悪く、実用的とは言い難いことがわかった。また、パターン３においてはレーザースキャナーの方にずれが生じてしまっているが、案内ロボットの動作時の状況を考えると、そこまで細かな動作が要求されるとは考えにくい。よってこの点については問題ないものとした。以上の理由から、今回製作する案内ロボットにはレーザースキャナーを採用することにした。

第 4 章　音声認識（佐藤担当）

4.1　概要

4.1.1　目的

移動ロボットを音声で操作するためには入力された音声を PC で認識する必要がある。ロボットはその後、認識した文章に従って案内を開始する。そこで、本章では案内で使用する PC で音声認識を行い、ロボットを目的地まで移動させるプログラムを起動することを目的とする。

4.1.2　要求仕様

本研究において案内にて使用する PC には OS として Ubuntu が使用されている（図 2.2）。音声認識も同じ PC で行う為、Ubuntu 上で動くことが必要条件である。加えて ROS と同時に起動するため、可能な限り処理が軽量である必要がある。本研究では以上の条件を満たす音声認識ソフトとして Julius を用いる。

4.1.3　Julius

Julius とは音声認識用のフリーソフトウェアであり、数万語の語彙を一般的なスペックの PC 上で認識可能である。Julius は単語辞書・言語モデル・音響モデルの三つから構成されており、各構成要素を編集することで幅広い用途に対応することができる[11]。

4.2　音声認識の方法

4.2.1　音声認識の準備

今回の実験では PC に搭載された内蔵マイクを使用した。また、既存の Julius では認識候補が多く、誤認識が多いため単語辞書を作成することで候補を減らし、認識精度を向上させる。本研究では下記の 4 つの単語を単語辞書に登録する。

- 研究室
- 作業室
- 保管室
- 実験室

単語辞書への登録は次の手順で行う。なお単語辞書への登録手順は以下のサイトを参考にした[12]。

https://raspibb2.blogspot.com/2017/03/raspberry-PI-julius-lirc.html

1. テキストエディタを用いて、登録する単語の表記及び読みを記したテキストファイルを作成する。本実験ではファイル名は remocon.yomi とした（図 4.1）。
2. 作成されたテキストファイルを元に単語辞書を作成する。

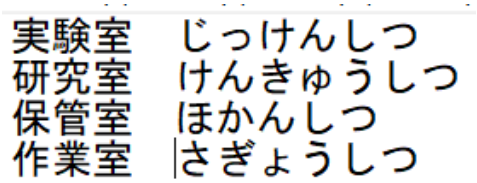

図 4.1　作成したテキストファイル

単語辞書への登録手順の詳細を説明する。

図 4.1 は実際に本研究で作成したテキストファイルである。左側が単語の表記であり右側が単語の読みである。この際表記と読みの間には半角空白または Tab キーを入力して区切りとする必要がある。

続いて作成したテキストファイルから単語辞書を作成するが、その準備段階として下記のコマンドを実行し補助コマンドをダウンロードする。

```
wget https://raw.githubusercontent.com/neuralassembly/raspi/master/addsil.pl
```

さらにこの補助コマンドに実行権限を付与するために下記コマンドを実行する。

```
chmod a+x addsil.pl
```

そして下記のコマンドを実行することでテキストファイルから単語辞書への変換が行われる。

```
iconv -f utf-8 -t euc-jp remocon.yomi | ./julius-4.4.2/gramtools/yomi2voca/yomi2voca.pl  |
iconv -f euc-jp  -t utf-8 | ./addsil.pl > remocon.dic
```

このコマンドでは以下のような順序で処理を行う。

1. 作成したテキストファイルの文字コードを utf-8 から euc-jp に変換する
2. yomi2voca.pl コマンドにより読みをローマ字に変換する
3. 文字コードを utf-8 に変換する
4. テキストファイルの先頭に休止状態の情報を追加する
5. 追加した状態のテキストファイルを remocon.dic として保存する

図 4.2 は上記のコマンドで保存された単語辞書ファイルである。

```
<s>     []      silB
</s>    [。]    silE
実験室  j i q k e N sh i ts u
研究室  k e N ky u u sh i ts u
保管室  h o k a N sh i ts u
作業室  s a gy o u sh i ts u
```

図 4.2　作成された単語辞書

4.2.2　音声認識の実験

本項では実際に音声認識の実験を行う。実験の際には下記のコマンドを実行した。

```
julius -C dictation-kit-v4.4/main.jconf -C dictation-kit-v4.4/am-gmm.jconf -demo
```

図 4.3 は実際に案内にて使用する PC で音声認識を行った際のターミナルの画像である。実験した結果、4.2.1 に示した単語辞書に登録した 4 つの単語の音声認識に成功していることがわかる。

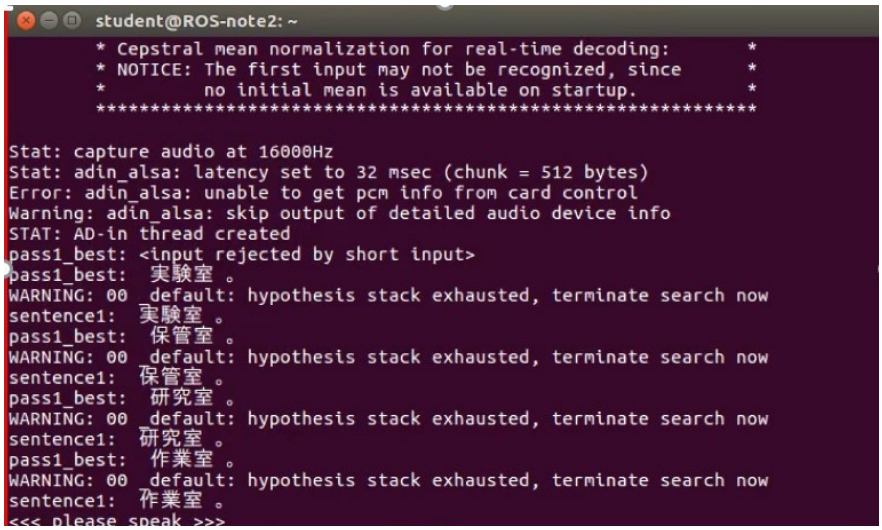

図 4.3　音声認識の際のターミナルの画面

4.2.3　Julius からプログラムを起動する

続いて Julius での音声認識からロボットを動かすためのプログラムを起動するためのプログラムを作成する。プログラムの作成にはインターネット上の既に完成したプログラムを参考にした。参考にしたプログラムの URL を以下に示す[13]。

https://raw.githubusercontent.com/neuralassembly/raspi/master/recog-TV.py

以下の図 4.4 はこのプログラムの実行部分である。実験室という単語が認識されたときに「rosrun test zikken.py」というコマンドが実行されるコードとなっている。プログラム全体を付録 1 に掲載する。

```
if u('実験室') in word:
    print(word)
    args = ['rosrun', 'test', 'zikken.py']
    try:
        subprocess.Popen(args)
    except OSError:
        print('command not found.')
```

図 4.4　Julius からプログラムを起動するためのプログラムの実行部分

このプログラムから起動する「ロボットを目的地まで移動させるプログラム」については 5 章 2 節で解説する。

Julius からプログラムを起動する際には Julius を module モードで起動する必要がある。ここで使用する module モードとは Julius を音声認識サーバーとして動かす事が出来るモードである。Julius を用いた自作のプログラムを作成する際には必ず module モードにする。Julius を module モードで起動する際のコマンドは以下の通りである。

```
julius -C dictation-kit-v4.4/main.jconf -C dictation-kit-v4.4/am-gmm.jconf -demo -module
```

第 5 章　プログラム（佐藤担当）

5.1　概要

本章ではロボットを目的地まで移動させるプログラムと、案内ロボットの状態を周囲に知らせるプログラムを実装する。ロボットを目的地まで移動させるプログラムは Julius から起動されるものであり、ロボットに目的地の座標を与えることを目的としている。一方、案内ロボットの状態を周囲に知らせるプログラムは、ロボットの状態（待機中か移動中かなど）を読み取り、周囲の人間に音声で知らせることを目的とする。

5.2　案内ロボットを目的地まで移動させるプログラム

本研究で使用する案内ロボットを目的地まで移動させるプログラムはインターネット上に公開されている中部大学ロボット理工学科演習の授業資料を参考にした。以下に URL を示す[14]。

https://robot.isc.chubu.A.C.　JP/？p＝1626

このプログラムは目的地の座標をロボットに入力することでロボットを目的地まで移動させることができる。目的地の座標および目的地でのロボットの向きは下記の 4 つの数値部分で指定している。プログラム全体を付録 2 に掲載する。

```
goal_ob.setGoal(-20.0 , -21.0 , 0.0 , 1.0)
```

プログラムの実装にあたり、ROS ではプログラムを配置する場所が決まっており正しく配置しなければプログラムを起動する事が出来ないことに注意する必要がある。そのためこのプログラムを実装する手順を解説する。なおここで説明する手順は Python を用いたプログラムを作成する場合に限った説明となる。この手順は以下の URL に示すページを参考にした[15]。

https://brain.cc.kogakuin.ac.jp/~kanamaru/lecture/ROS/index3.html

はじめにパッケージを作成し、作成したパッケージ内にある scripts というフォルダ内にてプログラムを作成する。では最初に下記のコマンドをターミナルで打ち込みパッケージを作成するフォルダに移動する。

```
cd ~/catkin_ws/src
```

なおこのコマンド内の「catkin_ws」は ROS をインストールした際に作成されたフォルダである。続いて下記のコマンドでパッケージを作成する。末尾の単語がフォルダの名称となり、本研究においては test とした。

```
catkin_create_pkg test
```

最後に下記のコマンドで scripts フォルダを作成する。

```
mkdir test/scripts
```

これでプログラムを作成する準備が終了する。その後 Ubuntu で使用できるテキストエディタを使用してプログラムを作成する。我々の環境の場合 leafpad というアプリケーションを使用した。作成したファイルは保存する際に、準備段階で作成した scripts フォルダに保存する。なおこのプログラムは目的地毎に新しく作成する必要がある。

5.3 案内ロボットの状態を周囲に知らせるプログラム

5.3.1 概要

本節では音声出力により周囲の人物に案内ロボットの状態を知らせるプログラムについて概要を説明する。図 5.1 はプログラムのフローチャートである。図 5.1 で使用している time というのはプログラムを実行した回数であり、本研究においてはロボットの状態に変化がない限りはプログラムが 40 回繰り返された場合、すなわち time という変数が 40 となった時のみ音声出力を行う。また、ROS ではロボットの状態を変数として発信しておりその変数によってそれぞれ音声出力を行う。このプログラムはインターネット上で公開されているものを参考にした[16]。プログラム全体を付録 3 に掲載する。

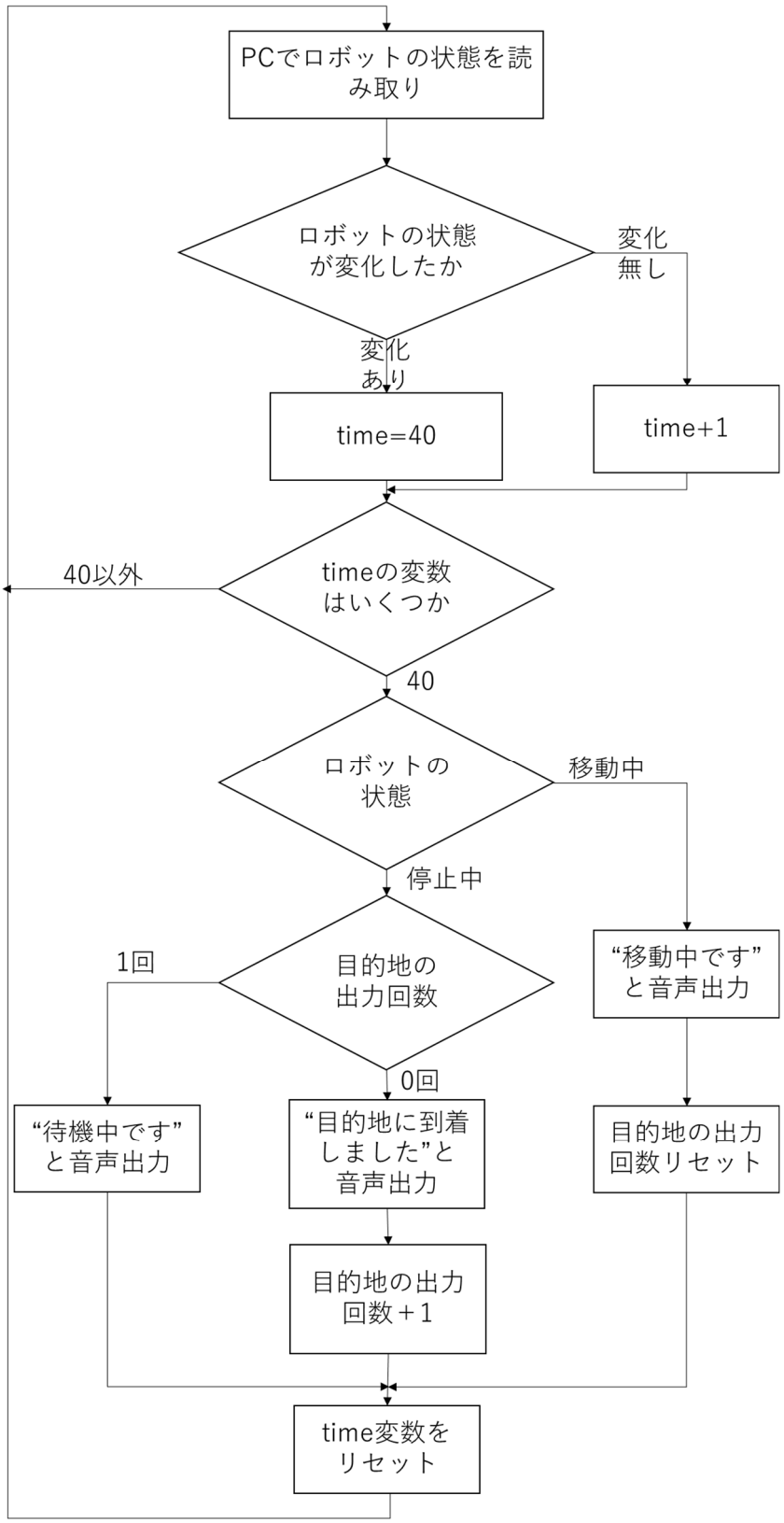

図 5.1　案内ロボットの状態を周囲に知らせるプログラムのフローチャート

5.3.2　プログラムの詳細

本節では案内ロボットの状態を周囲の人間に知らせるプログラム（図 5.1）をより詳しく説明する。

最初に ROS ではロボットの状態が navigation スタックの move_base の機能で status_id という変数として配信されている。status_id と内容との対応は下記の URL から確認することが出来る[17]。

http://docs.ros.org/en/jade/api/actionlib_msgs/html/msg/GoalStatus.html

この変数を下記の 1 行で読み取る。

```
status_id = msg.status_list[msg_len-1].status
```

ロボットが待機中であれば status_id は 0 となり、移動中であれば status_id は 1 、目的地に到着した際には 3 となる。ただし、status_id は 3 となったのち、時間経過で自動的に 0 になることはなかった。

status_id を読み取った際に繰り返し前と比較して変化がなかった場合、プログラムの繰り返し回数をカウントする time という変数を 1 増やし、time の変数が 40 となったときにロボットの状態を音声出力する。もし status_id を読み取った際に繰り返し前と比較して変化があった場合、time の変数を 40 とし、ロボットの変化後の状態をすぐさま音声出力する。

status_id は 3 から時間経過で 0 となることはないため、status_id の読み取りのみでは待機中と目的地に到着した状態の区別を付けることが出来ない。そのため「目的地に到着しました」と音声出力を行った回数を goal_count という変数でカウントし、goal_count が 1 であった場合、「目的地に到着しました」ではなく「待機中です」と音声出力をおこなう。この goal_count という変数はロボットが移動を開始した際にリセットされる。

第 6 章　案内ロボットの設計（松本担当）

6.1　概要

本章では、案内ロボットの設計及び、それに伴う実験検証について述べる。今回、案内ロボットを設計するに当たり、以下の条件を設定した。

・他の機材を経由せずに、案内ロボット 1 台で案内が行えるようにすること。
・案内ロボットがユーザーの音声を認識しやすい設計にすること。

1 つ目の条件は、案内開始から案内終了までの流れを案内ロボット 1 台で完結させることで操作を減らし、ユーザーが扱いやすい設計にするために設定した。

なお、今回の案内ロボットは目的地の選択をユーザーの音声で行う。そのため、ナビゲーションの精度を向上する上でも案内ロボットが音声を認識しやすい設計にする必要があると考え、2 つ目の条件を設定した。

6.2　案内ロボットのハードウェア設計

本節では、案内ロボットのハードウェアの設計について述べる。

6.2.1　使用機材

使用機材は、以下の通りである。第 2 章で紹介したものと同様の機材を多く使用するため、説明は一部省略する。

・TurtleBot3 waffle pi (2 章　図 2.1 参照)
・ノート PC(2 章　図 2.2 参照)
・バッテリー(LB-012 LIPO Battery)　(2 章　図 2.3 参照)
・Raspberry Pi　(2 章　図 2.4 参照)
・レーザースキャナー LDS-01　(2 章　図 2.6 参照)

・マイク(MICROPHONE ELM-2500)
本研究の案内ロボットは目的地の選択をユーザーの音声で行うため、音声入力にはマイクを使用する。比較的性能が良く、小さいものを選択した。

図 6.1　MICROPHONE ELM-2500

・マイクミュートスイッチアダプタ(ST35-SWAF)

会話などの案内に不必要な音を認識させないために、マイクミュートスイッチアダプタを使用する。音声を入力する際のみ off、基本的には on で使用する。

図 6.2　マイクミュートスイッチアダプタ ST35-SWAF

・ヘッドセット用変換アダプタケーブル

マイクとミュートスイッチアダプタを繋げる際に、ヘッドセット用変換アダプタケーブルを使用する。

図 6.3　ヘッドセット用変換アダプタケーブル

6.2.2　使用部品

・六角スペーサー（60mm x 66mm M3 オス - メスネジ）

TurtleBot を改造する際に、六角スペーサーを柱として使用する。

図 6.4　六角スペーサー(60mm x 66mm M3 オス - メスネジ)

・ TurtleBot3 Waffle Plate-IPL-01

TurtleBot に使われるプレートの部品をさらに追加で使用する。 TurtleBot を改造する際に、もう 1 枚プレートを追加するためである。

図 6.5　TurtleBot3 Waffle Plate-IPL-01

6.2.3　TurtleBot の改造

6.1 節で説明した条件を満たす案内ロボットの設計について検討した結果、基盤となる TurtleBot の形は図 6.6 のようになった（改造前の TurtleBot は図 2.1 参照）。

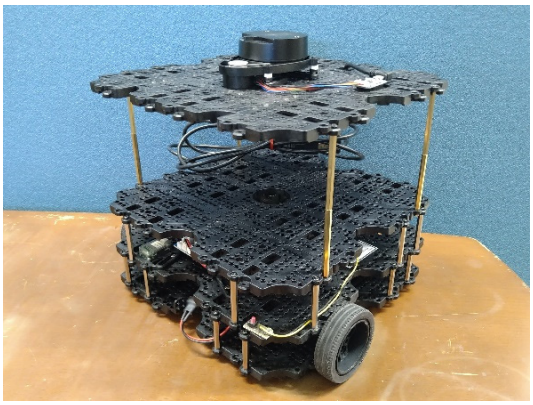

図 6.6　改造後の TurtleBot

これは、他の機材を経由せずに案内が行える構造にする上で、案内ロボットに PC を含めた全ての使用機材を乗せる必要があるからである。まず、本研究に使用する TurtleBot には PC を置けるスペースを用意する。その際、レーザースキャナーの認識範囲の広さを維持するために、中段にノート PC を置き、レーザースキャナー を一番上段に設置できるようにした。TurtleBot を動かす上での主要機材の設置場所は図 6.7 の通りである。　また、改造には 6.2.2 項で紹介した六角スペーサーとプレートを使用した。

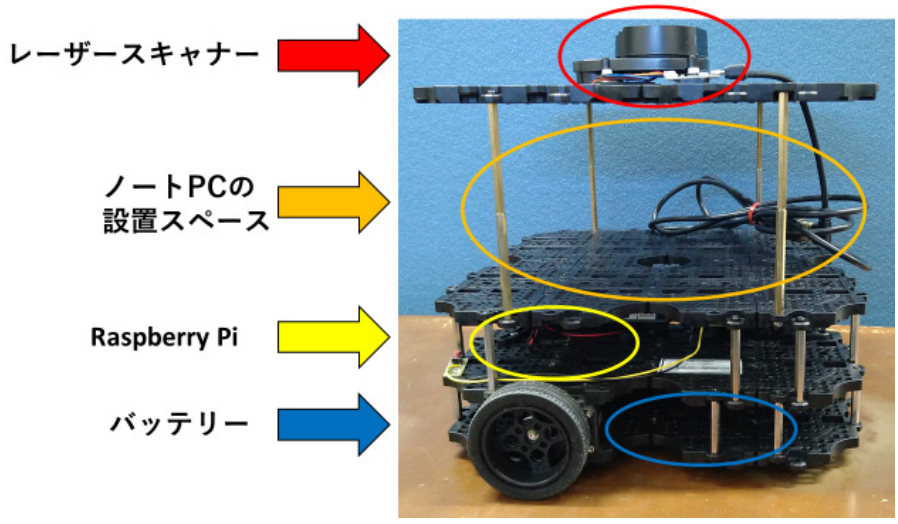

図 6.7　TurtleBot の詳細

6.2.4　マイク用の柱の設置

マイクは、扱うユーザーが起立した状態でも音声を認識できるように、できる限り高い位置に設置することを目標とした。ミュートスイッチアダプタも同様に、起立した状態で操作できるように高い位置に設置する。そのために、六角スペーサーを用いて図 6.8 のような柱を設けた。

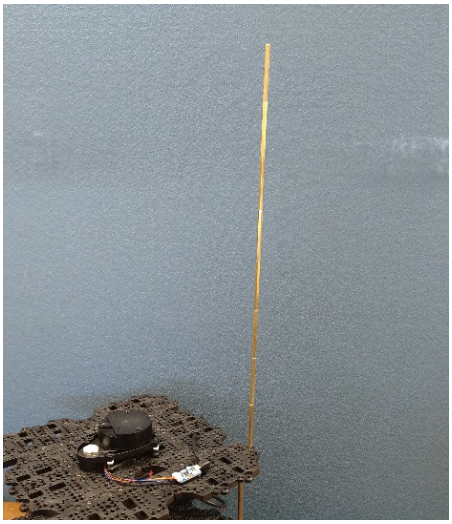

図 6.8　六角スペーサーの柱

6.2.5　マイクの設置

マイクの接続方法は図 6.9 の通りである。ミュートスイッチアダプタとマイクを柱の先端に設置し、延長コードを用いてノート PC と接続する。実際にマイクを柱の先端に設置した様子が図 6.10 である。

図 6.9　ノート PC からマイクまでの接続方法

図 6.10　柱の先端の様子

6.2.6　使用機材の設置

6.2.5 項までの内容を踏まえ、使用機材を全て設置した様子を示したのが図 6.11 である。

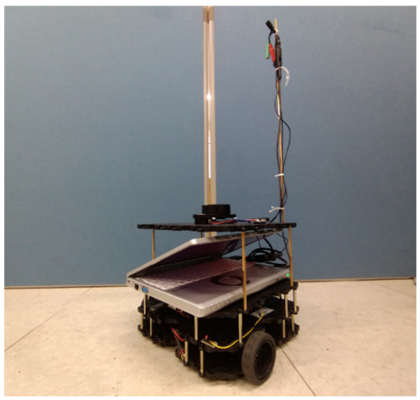

図 6.11　使用機材を全て設置した案内ロボット

主要機材の設置場所は図 6.12 の通りである。レーザースキャナーの下にノート PC を設置することで、案内ロボット全体の機材を 1 台で完結させることができた。また、マイクとミュートスイッチアダプタを柱の先端に設置することで、案内ロボットを扱うユーザーが起立した状態で音声入力が行えるようにした。これにより、6.1 節の条件を満たせる構造にすることに成功した。

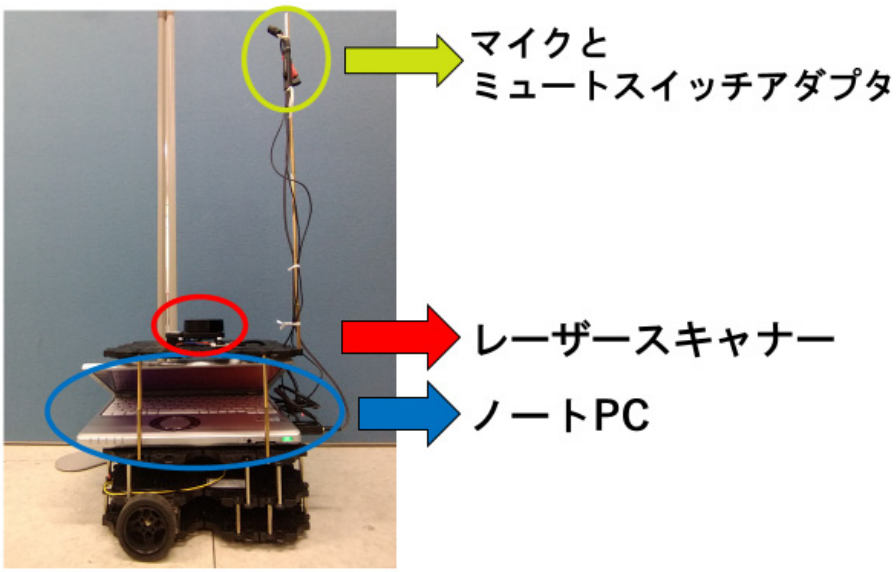

図 6.12　設置した主要機材の場所

6.3　柱がレーザースキャナーに与える影響について

本節では、6.2 節で出来上がった案内ロボットの構造上の問題点を挙げ、それがナビゲーションの精度にどの程度影響を与えるかについて、実験とその結果を示す。

6.3.1　柱による問題点

6.1 節より、マイクとミュートスイッチアダプタを高い位置に設置するために、TurtleBot の上段に六角スペーサーの柱を設置した。しかし、この構造は同じ段に搭載されているレーザースキャナーの認識範囲に柱が入ってしまうという問題点がある。図 6.13 はレーザースキャナーが柱を認識している実際の画像である。

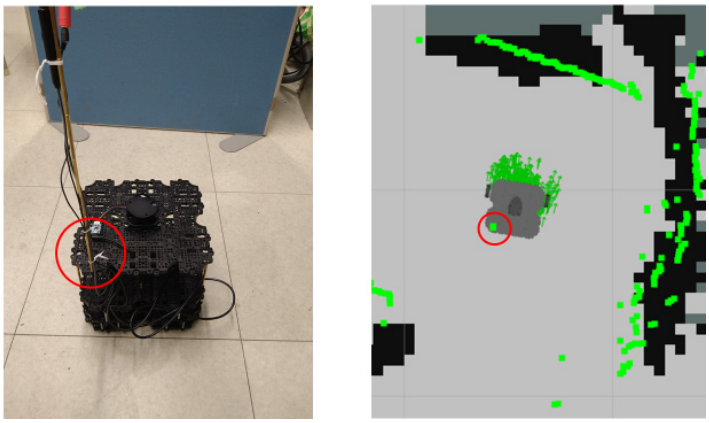

図 6.13　実際の写真(左)と地図作成時の PC 画面(右)

図 6.13（右）の画像は地図作成時の PC 画面の様子であり、レーザースキャナーが認識した障害物の位置が緑色で示されている。赤丸で示した部分に注目すると、図 6.13（左）の柱が緑色の点で表示されていることが分かる。すなわち、6.2 節で搭載した柱がレーザースキャナーにより障害物として認識されている。

6.3.2　柱の影響に関する実験

6.3.1 項より、レーザースキャナーが柱を認識していることが分かった。そのため、6.2 節で説明した案内ロボットの構造では、ナビゲーションの精度が下がることが予想される。そこで、柱の有無でナビゲーションの精度にどの程度影響が出るのかを調べる必要がある。本項では、柱の有無によるナビゲーションの精度の比較実験について説明する。

実験は次の手順で行う。

1. 目的地(Point)を 4 つ決め、その中から Main Point、Point1、Point2、Point3 を 1 つずつ選ぶ。
2. Main Point から Point1 までをナビゲーションで往復させる。
3. Main Point に戻れた場合をナビゲーション成功と見なし、記録する。
4. Point2、Point3 も同様にナビゲーションで往復させる。
5. Point1 から Point3 まで順番に往復し、ナビゲーションの成功率を測る。

今回、上記の手順を案内ロボットに柱を搭載した場合としない場合でそれぞれ行い、ナビゲーションの精度の比較を行う。下記の図 6.14 は、実際にナビゲーションを行う実験環境の写真である。

図 6.14　実験環境の写真

今回の実験で扱う 4 つの目的地(Point)の場所は、下記の図 6.15 の通りである。それぞれの目的地の座標を数値で設定し、ROS で指定した座標まで移動させるプログラム(5.2 節、付録 2 参照)を起動させることでナビゲーションを行う。

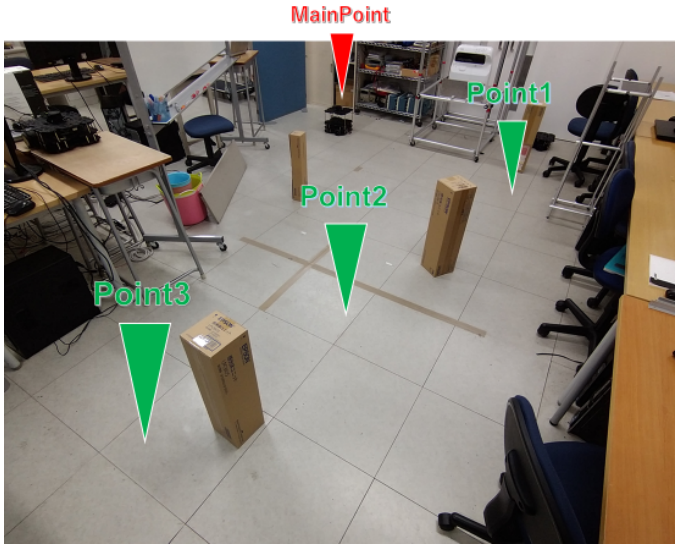

図 6.15　それぞれの目的地の場所

なお、ロボットが目的地に到着した際には、ROS のターミナルに「Goal reached」という文章が表示される。今回の実験では、「Goal reached」という文章が確認できた場合のみをナビゲーション成功と見なす。下記の図 6.16 は、実際にナビゲーションが完了したときのターミナルの様子である。

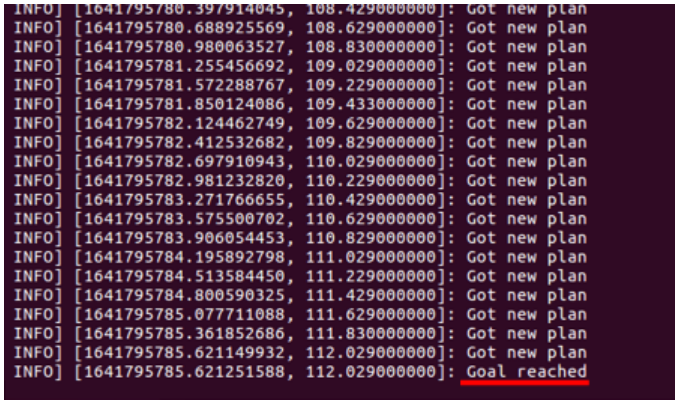

図 6.16　ナビゲーション時のターミナルの様子

6.3.3　実験結果

今回、案内ロボットに柱を設置した場合と設置しない場合でそれぞれ 100 回ずつナビゲーションを行った。その結果、成功率は下記の表 6.1 のようになった。

表 6.1　ナビゲーション成功率の測定結果

	柱有り	柱無し
ナビゲーション成功	81 回	84 回
ナビゲーション失敗	19 回	16 回
ナビゲーション成功率	81%	84%

今回の実験より、柱を設置した場合は設置しなかった場合に比べて、ナビゲーションの成功率が 3%下がるという結果が得られた。実験前に予想した通りナビゲーションの精度は下がってしまったが、それほど大きな差はでなかった。

ナビゲーションが失敗した原因の多くは、案内ロボットが実際の障害物に近づきすぎて、方向転換ができなくなってしまう場合だった。その現象は柱の有無に関わらず発生しており、明確に柱が原因となる失敗は見られなかった。

以上のことを踏まえ、今回の実験では柱がナビゲーションに与える影響は少ないと考える。

6.4　案内ロボットの完成

6.1 節から 6.3 節より、条件を満たす案内ロボットを完成させることができた。図 6.17 は完成した案内ロボットの写真である。

図 6.17　完成した案内ロボット

6.2 節と 6.3 節で述べたように、レーザースキャナーの認識範囲にマイク用の柱を設置する形となった。ナビゲーションの精度にできる限り影響を与えずに音声認識がしやすい設計にすることができたといえる。

第 7 章　案内ロボットの起動および実験（関本担当）

7.1　概要

本節では、完成した案内ロボットに案内を行わせ、第 1 章にて掲げた要求条件を達成しているのか検証する。

7.2　実験方法

実験環境は以下のとおりである

・実験環境は工学院大学八王子キャンパス 4 号館の 8 階の廊下である。実験環境の写真は図 7.1 であり、作成された地図は図 7.2 である。

・案内ロボットには「研究室」と「実験室」の 2 か所を登録した。図 7.1 および図 7.2 の赤い目印が、それぞれ登録されている目的地である。

本実験では最初に研究室へ案内をさせ、続けて実験室へ案内をさせることとする。

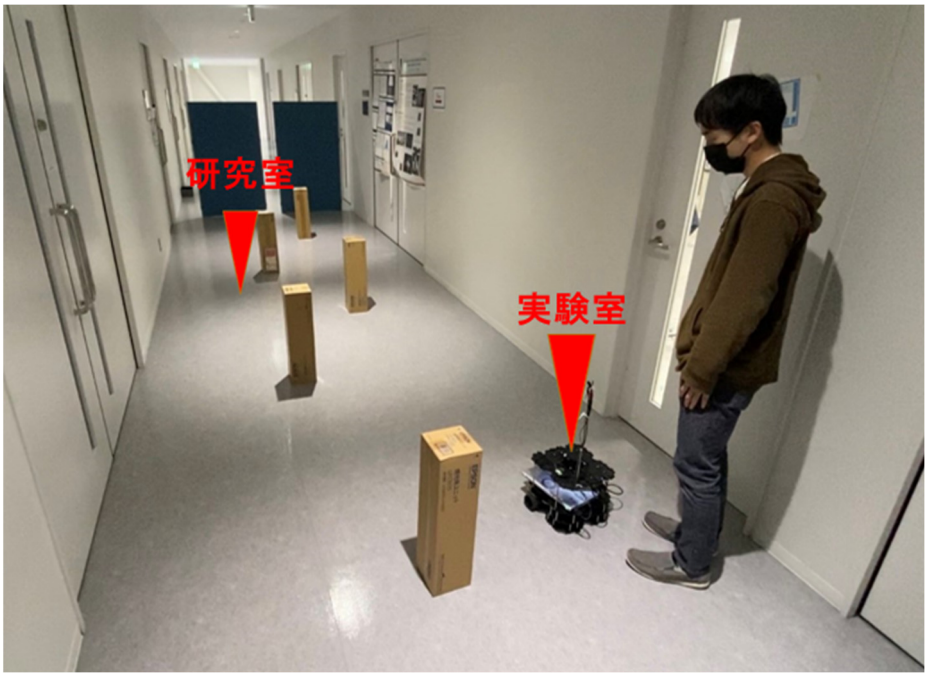

図 7.1　実験環境

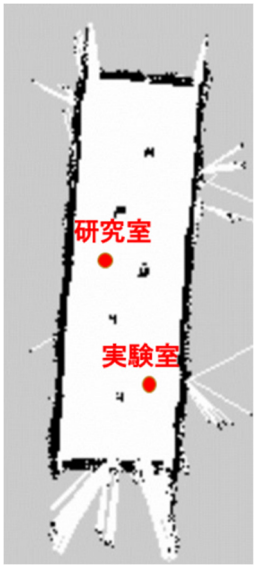

図 7.2　作成された地図

7.3　実験の様子

まず、案内ロボットに搭載した音声認識により、音声入力をする。マイクに付いているミュートスイッチを切り替え、「研究室」と音声入力をした。

図 7.3　音声入力をする様子

音声入力後、ロボットは「案内を開始します」と音声出力をした上で移動を開始した。案内ロボットは、案内をしている際には「移動中です」と音声出力を繰り返していた。また、移動の途中に図 7.4 のように進行方向の柱に衝突しそうな軌道であったが、図 7.5 にて手前で方向転換をした後、衝突を回避した。

図 7.4　案内ロボットが障害物の手前まで移動している様子

図 7.5　案内ロボットが方向転換をしている様子

目的地に到着した後、案内ロボットは停止し「目的地に到着しました」と音声出力をした。その後、数秒間を空けて「待機中です」と音声出力をした。

図 7.6　案内ロボットが研究室に到着した様子

さらに案内ロボットが目的地を複数設定できているかの検証をするために、続けて案内を行わせる。1 回目と同様にミュートスイッチを切り替え、今回は「実験室」と音声入力をした。

図 7.7　二回目の音声入力をする様子

案内ロボットは「案内を開始します」と音声を出力し移動を開始した。一回目と同様に、案内ロボットは「移動中です」と音声出力をしながら移動をした。

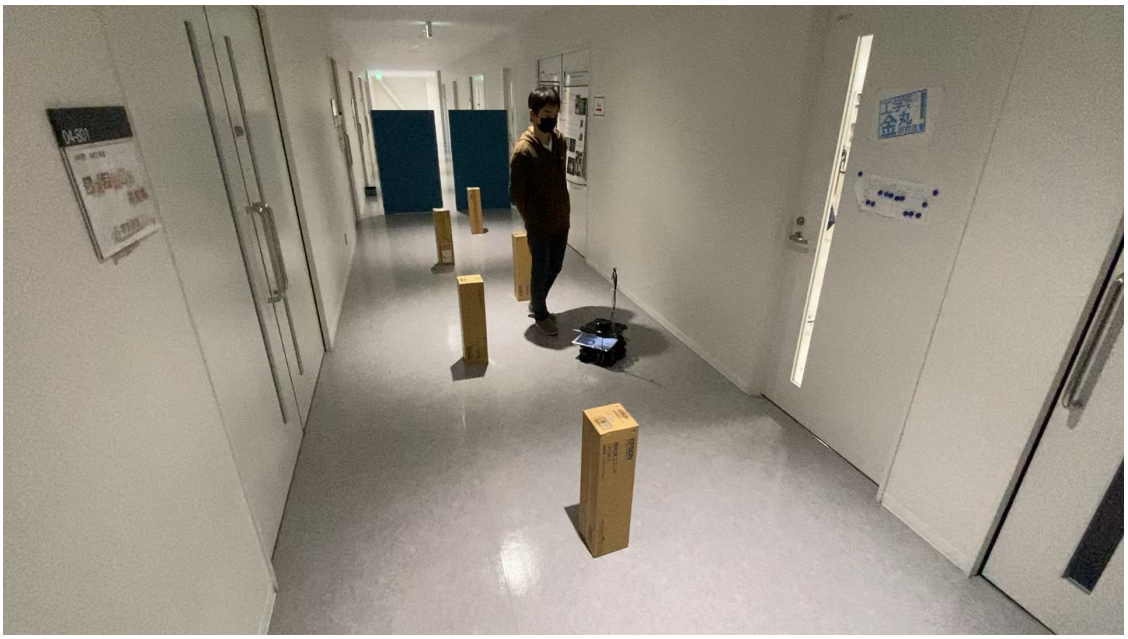

図 7.8　二回目の案内をする様子

目的地で停止した後に「目的地に到着しました」と音声出力をした。このように、二回目のナビゲーションも問題なく行うことが出来た。

図 7.9　案内ロボットが実験室に到着した様子

7.4　要求条件への達成度

以上の実験結果をもとに、第一章にて掲げた「本研究で開発する案内ロボットの条件」を満たせているのか確認をする。

まず今回の実験より、案内ロボットが被案内者を目的地まで直接移動し、案内をすることが出来ることが確認できた。よって、自律移動ロボットを主体とし、直接目的地まで移動できるようにするという条件を満たすことができた。

次に、Julius による音声認識機能を搭載したことにより、今回の実験のように音声入力で目的地を選択出来ることが確認できた。よって、音声認識機能を搭載し、音声で行先を選択できるようにするという条件を満たすことができた。

また、続けて二回案内を行うことで、目的地を複数設定できていることが確認できた。よって、案内できる場所は、複数設定できるようにするという条件を満たしている。

さらに、以上のような案内を大学校内の廊下という屋内施設で実行出来ることが確認できた。よって、使用する環境は、駅構内や博物館、デパートなどの屋内施設という条件を達成できていることがわかる。

このように、本研究で開発した案内ロボットは、第一章にて掲げた要求条件をすべて十分に達成していることが確認できた。

第８章　ナビゲーション（関本担当）

8.1　概要

第 7 章にて、完成した案内ロボットは当初要求していた条件を十分に満たしていることが分かった。しかし、案内ロボットに挙げられる問題点として、案内の際に目的地の座標に正確に止まろうと微調整をすることにより、停止するまでに時間がかかってしまうことがある。また、さらに案内を開始する際に一定の確率でエラーが起きてしまうことがある。実際に案内ロボットを使用する場面において、同様の事態が多発することは実用的とは言えない。そこで、案内ロボットに何回か案内を行わせ、案内にかかった時間およびエラーが発生した回数を記録し、案内ロボットの性能を評価することにした。

8.2　実験方法

実験環境は以下のとおりである。

・実験環境は工学院大学八王子キャンパス 4 号館の 8 階の廊下である。実際の実験環境の写真は図 8.1 であり、作成された地図は図 8.2 である。
・目的地を図 8.2 の赤点に記す二か所に設定し、その間隔は 5m とした。
・障害物として、二か所の目的地の間に 4 本の柱を設置した。

図 8.1　実験環境

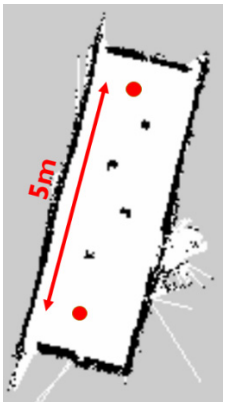

図 8.2　作成された地図

なお、時間の測定にはスマートフォンのタイマーを用いる。案内ロボットに音声入力を終えた瞬間にタイマーを開始する。そして案内ロボットによる音声出力の「目的地に到着しました」と発声した瞬間にタイマーを止めることとした。また、実験回数はバッテリーの充電が切れるまでの 96 回とした。

8.3　実験結果

下記の図 8.3 が実験結果である。

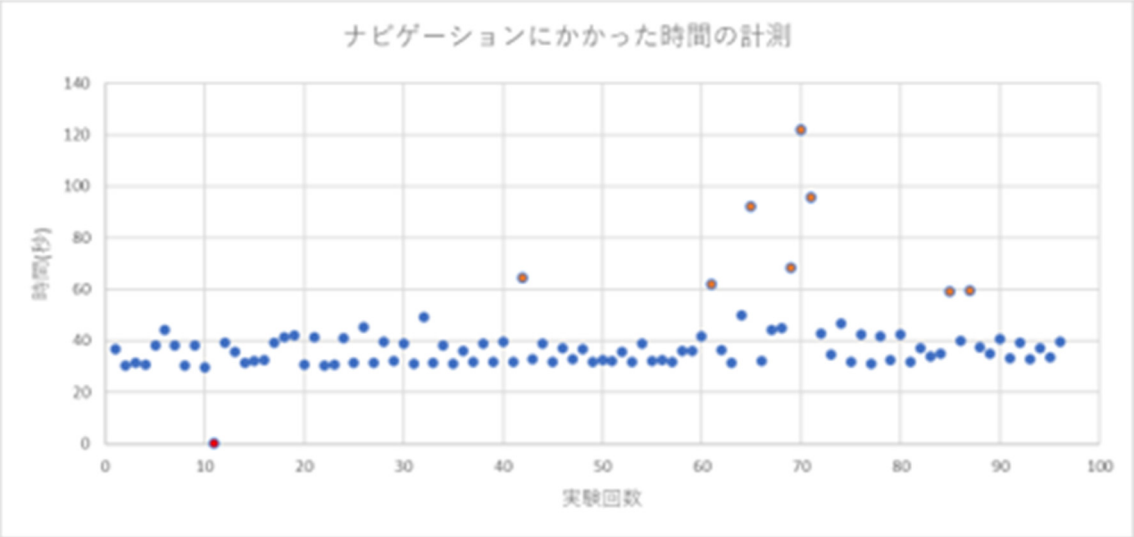

図 8.3　ナビゲーションにかかった時間の計測

96 回の案内の間、1 回エラーが発生した。エラーが発生した記録は、図 8.3 において赤点で着色した。

8.4　実験結果の分析

最初に、この図 8.3 から案内にかかった最短時間は 29.55 秒であることがわかった。一方で、同じ実験環境にもかかわらず 1 分以上時間がかかっている記録もいくつか見られた。この結果より、案内に生じる誤差を考慮し、最短時間よりも 2 倍以上時間がかかっている場合は良い案内とは言えないと考えた。そこで性能評価では 29.55 秒の 2 倍である 59.1 秒よりも時間がかかった記録を失敗とみなすものとする。また、エラーが起きた場合も失敗とする。図 8.3 にて、橙点に着色した記録が時間がかかった記録を表す。

結果、96 回中 7 回 案内に失敗した。これは全体の 9.37%であった。

第 9 章　結論（関本担当）

9.1　性能評価と考察

時間を考慮しないのであれば、案内ロボットの案内の成功率は 98.9%とほぼ成功するため、機能面に問題はないと評価した。しかし、この実験結果から大きく時間がかかる場合およびエラーが発生する確率は 9.73%であったため、安定性や実用面ではまだ改善の余地があると考えられる。今回の性能評価において、失敗と判断した記録を除いた平均の時間は 36.1 秒なので、改善することが出来れば理想的な案内ロボットと評価できると思われる。

9.2　改善案

上記の問題の改善案としては、目的地の座標に到着した判定を緩めることで改善することが出来ると考える。また図 8.3 より、バッテリーが消耗するとナビゲーションに失敗するケースが増えていることがわかる。そのため、定期的にバッテリーを交換する必要がある。

参考文献・URL

[1]厚生労働省「国内の発生状況など」(閲覧日:2022/01/25)
https://www.mhlw.go.jp/stf/covid-19/kokunainohasseijoukyou.html

[2] デロイト トーマツ ミック経済研究所(2021)「with コロナ時代で普及が加速する法人向けサービスロボット市場展望 2021 年版」(閲覧日:2022/01/25)
https://mic-r.co.jp/mr/01990/

[3]SankeiBiz(2020)「ウィズコロナへロボット活用　感染リスクの低減に」(閲覧日:2022/01/26)
https://www.sankeibiz.jp/business/news/200717/cpc2007170650001-n1.htm

[4] ROS COMPOMENTS「TurtleBot3 Waffle pi」
https://www.roscomponents.com/en/mobile-robots/215-TurtleBot-3-waffle.html#/courses-no/TurtleBot3_waffle_pi_model-waffle_pi

[5] RT ROBOT SHOP「LIPO Battery 11.1V 1800mAh LB-012」
https://www.rt-shop.jp/index.php?main_page=product_info&products_id=2595

[6]RS 「Raspberry pi B+ 150 枚入りバルクボックス」
https://jp.rs-online.com/web/p/raspberry-pi/8300608

[7] ROBOTIS e-Manual「OpenCR 1.0」
https://emanual.robotis.com/docs/en/parts/controller/opencr10_jp/

[8] ROBOTIS「360 Laser Distance Sensor LDS-01」
https://e-shop.robotis.co.jp/product.php?id=11

[9] SWITCHSCIENCE「Intel RealSense Depth Camera D435」
https://www.switch-science.com/catalog/3633/

[10]SWITCHSCIENCE「Intel RealSense Tracking Camera T265--在庫限り」
https://www.switch-science.com/catalog/5424/

[11] Julius development team「Julius now on GitHub」

https://julius.osdn.jp/index.php?q=whatis.html

[12] 実例で学ぶ Raspberry Pi 電子工作　補足情報「Raspberry Pi + Julius + LIRC により家電製品を音声認識で操作する」
https://raspibb2.blogspot.com/2017/03/raspberry-pi-julius-lirc.html

[13] 実例で学ぶ Raspberry Pi 電子工作　補足情報「Raspberry Pi + Julius + LIRC により家電製品を音声認識で操作する」より Julius からプログラムを起動するプログラム
https://raw.githubusercontent.com/neuralassembly/raspi/master/recog-TV.py

[14]中部大学　ロボット理工学科 演習「プログラムからナビゲーションを実行する方法」
https://robot.isc.chubu.ac.jp/?p=1626

[15] Windows で ROS のシミュレーションを行う「Turtlesim で ROS を体験する」
https://brain.cc.kogakuin.ac.jp/~kanamaru/lecture/ROS/index3.html

[16]Qiita「move_base でゴールについたことを検知する」
https://qiita.com/nnn112358/items/d159204d565469f647bb

[17] ros.org「actionlib_msgs/GoalStatus Message」
http://docs.ros.org/en/jade/api/actionlib_msgs/html/msg/GoalStatus.html

付録

付録 1　Julius からプログラムを起動するためのプログラム

```
# -*- coding: utf-8 -*-
import socket
from io import StringIO
import re
import subprocess

try:
    unicode # python2
    def u(str): return str.decode('utf-8')
    pass
except: # python3
    def u(str): return str
    pass

host = '127.0.0.1'
port = 10500
bufsize = 1024

buff = StringIO(u(''))
pattern = r'WHYPO WORD=¥"(.*)¥" CLASSID'
try:
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((host, port))
    while True:
        data = sock.recv(bufsize)
        buff.write(data.decode('utf-8'))
        data = buff.getvalue().replace('> ', '>¥n ')
        if '¥n' in data:
            lines = data.splitlines()
            for i in range(len(lines)-1):
                if lines[i] != '.':
                    #print(lines[i])
```

```
                m = re.search(pattern, lines[i])
                if m:
                    word = m.group(1)



                    if u('実験室') in word:
                        print(word)
                        args = ['rosrun', 'test', 'zikken.py']
                        try:
                            subprocess.Popen(args)
                        except OSError:
                            print('command not found.')
                    elif u('研究室') in word:
                        print(word)
                        args = ['rosrun', 'test', 'kenkyu.py']
                        try:
                            subprocess.Popen(args)
                        except OSError:
                            print('command not found.')
                    elif u('保管室') in word:
                        print(word)
                        args = ['rosrun', 'test', 'hokan.py']
                        try:
                            subprocess.Popen(args)
                        except OSError:
                            print('command not found.')
                    elif u('作業室') in word:
                        print(word)
                        args = ['rosrun', 'test', 'sagyou.py']
                        try:
                            subprocess.Popen(args)
                        except OSError:
                            print('command not found.')

            buff.close()
            buff = StringIO(u(''))
```

```
                if lines[len(lines)-1] != '.':
                    buff.write(lines[len(lines)-1])

except socket.error:
    print('socket error')
except KeyboardInterrupt:
    pass

sock.close()
```

付録２　案内ロボットを目的地まで移動させるプログラム

```
#! /usr/bin/env python3

import rospy
from geometry_msgs.msg import PoseStamped

class Goal:
    def __init__(self):
        rospy.init_node('set_goal', anonymous=True)
        self.ps_pub = rospy.Publisher('move_base_simple/goal', PoseStamped,
queue_size=1)

    def setGoal(self, px, py, pz, ow):
        rospy.sleep(1.0)

        now = rospy.Time.now()
        goal_point = PoseStamped()

        goal_point.pose.position.x = px
        goal_point.pose.position.y = py
        goal_point.pose.position.z = pz
        goal_point.pose.orientation.w = ow
        goal_point.header.stamp = now
        goal_point.header.frame_id = 'map'
        self.ps_pub.publish(goal_point)

if __name__ == '__main__':

    try:
        goal_ob = Goal()
        goal_ob.setGoal(-20.0, 21.0, 0.0, 1.0)
        rospy.spin()
    except rospy.ROSInterruptException: pass
```

付録３　案内ロボットの状態を周囲に知らせるプログラム

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import rospy
from move_base_msgs.msg import MoveBaseAction
from actionlib_msgs.msg import GoalStatusArray
import subprocess

goal_count = 0
time = 0

def callback_navstatus(msg):
    msg_len = len(msg.status_list)
    time_max = 40
    global goal_count
    global time
    if msg_len < 1:
        return
    status_id = msg.status_list[msg_len-1].status
    # if unstable, try the line below.
    #status_id = msg.status_list[0].status

    if status_id == status_id:
        time = time + 1

    else:
        time = time_max

    if status_id == 1 and time == time_max:
        print('Moving')
        x = '移動中です'
        args = ['speech.sh',x]
        try:
            subprocess.Popen(args)
```

```
            except OSError:
                        print('no speech.sh')
            goal_count = 0
            time = 0

      elif (status_id == 3 or status_id == 0) and time == time_max:
            if goal_count < 1 :
                        print('Goal reached')
                        x = '目的地に到着しました'
                        args = ['speech.sh',x]
                        try:
                                    subprocess.Popen(args)
                        except OSError:
                                    print('no speech.sh')
                        goal_count = goal_count + 1
                        time = 0
            elif goal_count == 1 :
                        print('staying')
                        x = '待機中です'
                        args = ['speech.sh',x]
                        try:
                                    subprocess.Popen(args)
                        except OSError:
                                    print('no speech.sh')
                        time = 0
rospy.init_node('move_base_goal_state')
rospy.Subscriber('/move_base/status', GoalStatusArray, callback_navstatus)
rospy.spin()
```

音声認識で操作する先導型案内ロボットの開発

指導教員　金丸 隆志　教授
S5-17045 細川 森平　S5-18026 佐藤 涼平
S5-18033 関本 崇文　S5-18056 松本 幸寛

1. 緒言

2020 年に新型コロナウィルスが世界中に蔓延した影響で、ロボットの需要が高まっている。一例として、駅や博物館などの一部の屋内施設では案内係をロボットに置き換えることで感染対策を行っている。しかし、案内（ナビゲーション）を行うロボットの多くは音声対話による案内であり、目的地まで直接案内を行うことはできない。

そこで、音声認識機能と目的地まで先導して案内を行える機能を両方搭載させることで、案内ロボットとしての利便性の向上に繋がると考えた。このことから、本研究では音声認識で操作する先導型案内ロボットの開発をする。

2. 原理

案内ロボットを開発する上での要求条件は以下の通りである。

・使用する環境は、駅構内や博物館などの屋内施設とする。
・自律移動ロボットを主体とし、直接目的地まで移動できるものとする。
・案内できる場所は、複数設定できるものとする。
・音声認識機能を搭載し、音声で行先(目的地)を選択できるものとする。

上記の条件を踏まえ、目的地まで案内する過程を図 1 に示す。案内ロボットには、あらかじめマップデータを登録させ、指定した座標まで移動できるプログラムを組んだ。その後、目的地ごとに違う座標のプログラムを用意し、認識した音声によって対応するプログラムを起動できるものとした。これにより、案内ロボットを扱うユーザーは音声で目的地を伝えた後、移動するロボットを追従することで目的地まで行くことが可能となる。本研究ではこれらの機能を、システム開発には ROS(Robot Operating System)、自律移動ロボットには TurtleBot3 waffle pi を採用して開発を行った。

図 1. 目的地までナビゲーションを行う過程

3. 案内ロボットの設計

設計した案内ロボットを図 2 に示す。本研究では、精度の高いナビゲーションやマップデータの登録を行うために、センサーにはレーザースキャナーを採用した。また、マイクを高い位置に設置することで、案内ロボットを扱うユーザーが起立した状態でも音声の認識が行える設計にした。

図 2. 設計した案内ロボット

4. 完成した案内ロボットの実験

完成した案内ロボットが要求条件を満たしているかを実験により確認をする。

案内ロボットに目的地として研究室と実験室の座標を登録し、実際に案内をさせる。図 3 が実験環境であり、図 4 が作成された地図である。本実験では目的地として研究室と実験室を登録した。この 2 つの目的地への案内を行わせることで、要求条件を満たしているか確認をする。

図 3. 実験環境

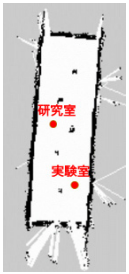

図 4. 作成された地図

以上の実験の結果、問題なく案内が出来ることが確認できた。そのため、要求条件の全てを十分に満たしていることがわかった。

5. 結言

(1)案内ロボットに使用する部品を選定し、制作した.
(2)案内ロボットに使用するプログラムを作成した.
(3)完成した案内ロボットは要求条件を十分に満たしていることが確認できた。