2020 年度（令和 2 年度）

創造工学セミナーII Final Report

移動ロボットの地図作成と
ナビゲーションに関する研究

チームメンバー

S5-17020　君塚　瑛

S5-17051　宮前　直樹

S5-17055　山下　創

指導教員　：　金丸　隆志　教授

目次

第１章　緒言（宮前担当）

1.1　研究背景

本節では、新型コロナウィルスが 2020 年から 2021 年にかけてどれほどの勢いで蔓延しているかをはじめに示し、続いて、その感染拡大がロボット市場に与えている影響について解説する。

1.1.1　新型コロナウィルスの蔓延

2020 年より新型コロナウィルスが世界中に蔓延している。図 1-1 は日本国内の新型コロナウィルス感染者数の推移であり、第一波から第三波にかけて感染規模が増大していることが示されている。その影響で、感染拡大を防ぐために密閉・密集・密接（三密）を避けて行動することが求められるなど人々の生活が大きく制限されている。外出を極力避けるためネットショッピングの利用者が増加したり、会社に出勤せずに仕事を行うリモートワークが普及するなど、人と人との距離がなるべく近づかないことに注意が向けられている。

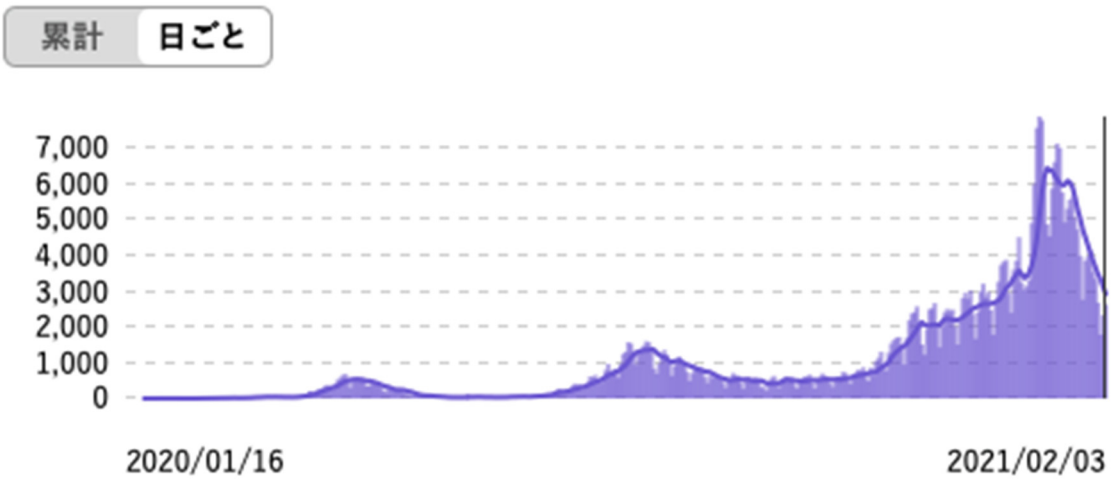

図 1-1　日本国内の新型コロナウィルス感染者数の推移 [1]

1.1.2　自律移動ロボットの需要増加

その一方で、ロボット市場は産業用ロボットから非産業用ロボット中心へとシフトし、AI技術などの成果も取り入れながら大きく成長することが見込まれている。新型コロナウィルスの蔓延によって、2020 年から数年は部品の調達や工場の生産などへの影響が見込まれるものの、その後は世界経済の回復とともにロボットの需要が増える可能性があることが予測されている(図 1-2) [2]。

それに伴い、これまで工場内の荷物運搬など産業用に主に使われてきた自律移動ロボットが、より身近な場面で普及していくことが期待される。

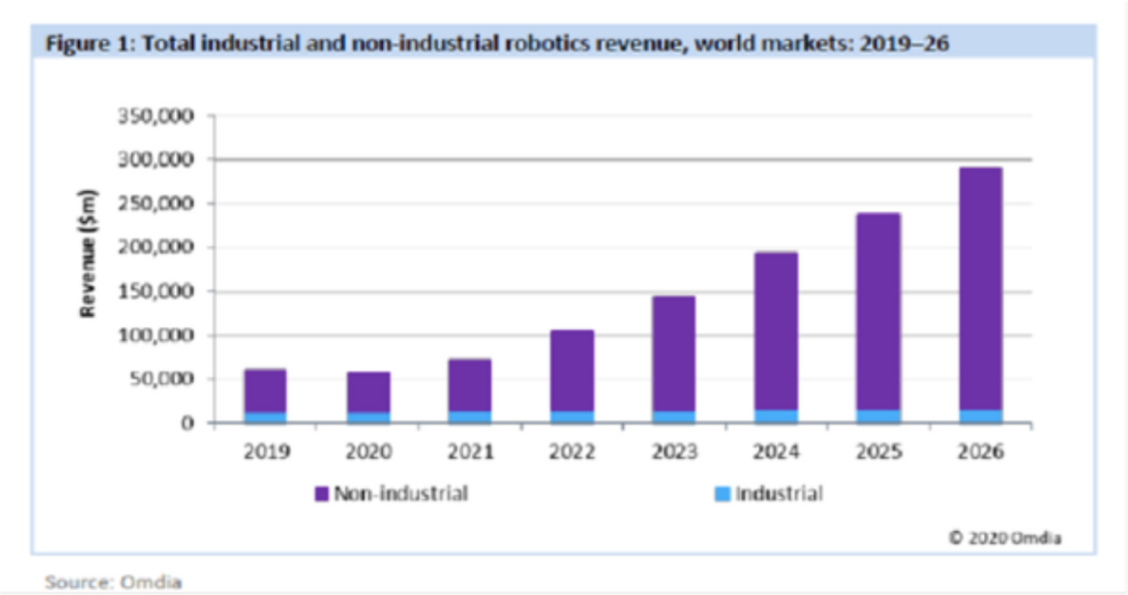

図 1-2　ロボットの市場予測 [2]

そこで、我々は自律移動ロボットに関連した地図作成技術とそれを用いたナビゲーションについて研究する。

1.2　先行研究

1.2.1　画像処理に基づく自動書類運搬装置の開発

自律移動ロボットに関連し、先行研究では自動で書類の運搬を行うロボットの開発を行っている[3]。具体的な使用状況は以下の通りである。

- 使用場所　　会社のオフィス
- 使用目的　　デスク間の書類の受け渡しの簡略化のため、書類等の荷物を運搬すること
- 運搬する荷物　　書類・軽い荷物

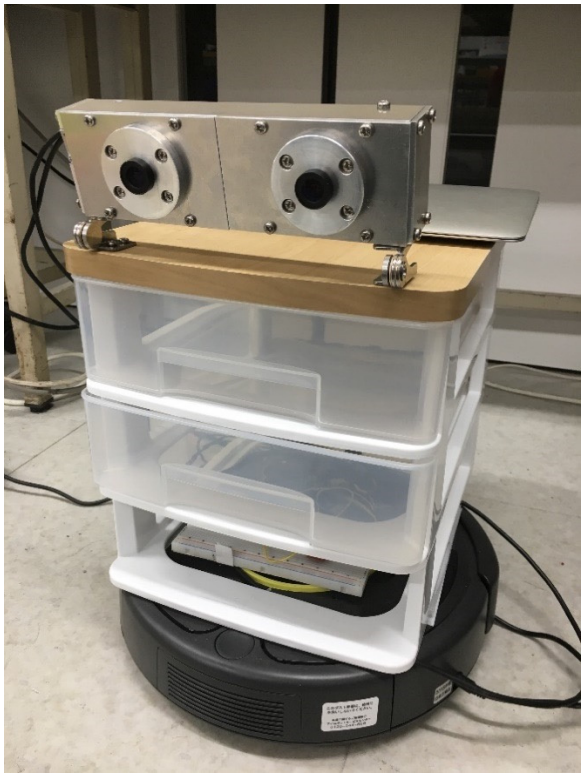

図 1-3　自動書類運搬装置［3］

先行研究では自律ロボットとして iRobot 社による掃除ロボットである Roomba 622 を利用し、その上に自作ステレオカメラと PC を載せた構造になっている。ステレオカメラとは 2 個のレンズを左右に並べることで奥行き方向の情報を取得できるカメラである。図 1-4 は、スバルのアイサイトという運転支援システムで用いられているステレオカメラである。

図 1-4　スバルのアイサイトで用いられているステレオカメラ［4］

この先行研究では、書類の受け渡し場所(目的地)にあらかじめ赤や青や緑などの色がついた立方体の箱を設置し、ロボットに赤→青→緑の箱の位置に順に移動させることで自律移動を実現している。

その際、ステレオカメラは対象物の色の取得と、対象物までの距離を測定する目的で利用されている（図 1-5）。

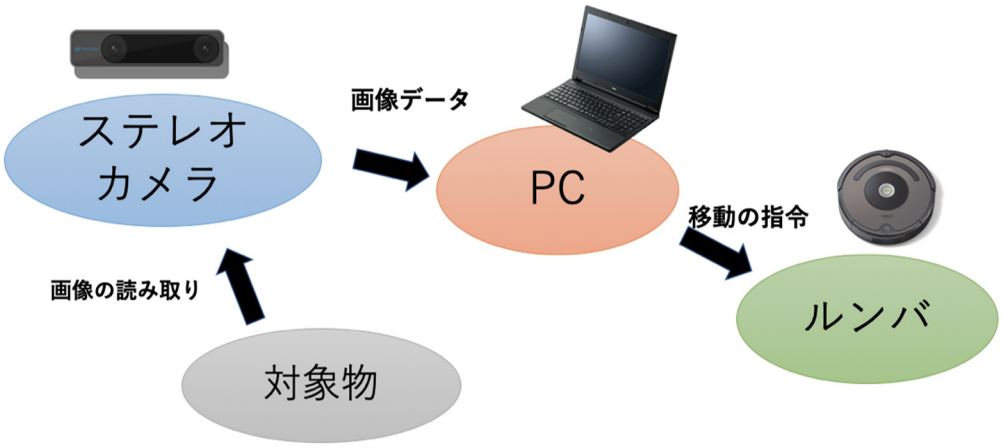

図 1-5　先行研究の機器の構成

1.2.2　先行研究の問題点

先行研究には以下のようにいくつかの問題点があった。

i. 自作のステレオカメラの機械的な精度が低いため、対象物を誤認識したり、対象物までの距離測定を正しく行えないことがある
ii. 目的地を色によって検出するため、オフィスに目的地と似た色のものがあった場合に誤認識する可能性がある
iii. あらかじめ決められた順番にしか移動ができないため、実際にオフィスで使えるほど実用的でない

これらの問題のうち ii.や iii.は、認識プログラムや制御プログラムの改善により解決できる可能性があるものの、i.はセンサとして自作ステレオカメラを用いること自体に問題があり、容易に解決できるものではなかった。

1.3　研究目的

本節では、本研究の目的について述べる。

1.3.1　自己位置推定

自己位置推定とはロボットが環境内のどの位置にいるのかをロボット自身が推定することである。自律移動ロボットが目的地まで移動する際、自分の位置と目的地までの経路を得るためには自己位置推定の技術が必要不可欠である。

先行研究では自己位置を推定するために自作ステレオカメラを利用していたが、精度の問題のため、対象物の誤認識や不正確な距離測定など、多くの問題があった。

そこで、本研究では自作ステレオカメラではなく LiDAR（Light Detection and Ranging または Laser Imaging Detection and Ranging）と呼ばれる赤外線レーザー距離センサ(図 1-6)を用いる。LiDAR による距離計測とロボットの車輪の回転角の計測とを合わせて自己位置推定を行う。

センサの変更により、1.2.2 項で述べた先行研究の問題点の i.および ii.を解決できる。

図 1-6　360 Laser Distance Sensor LDS-01 [5]

1.3.2　地図作成とナビゲーションの精度

本研究では、自律移動ロボットに必要な機能として、部屋の構造や障害物の位置を把握する地図作成機能と、壁や障害物などを避けて指示された目的地に進むナビゲーション機能の二つを考える。

地図作成は走行時にセンサから得た情報をもとに自己位置を推定しながら行う。このように、自己位置推定と地図作成を同時に行うことを SLAM（Simultaneous Localization and Mapping）という。

一方ナビゲーションは、事前に作成した地図と走行時のロボットの自己位置をもとに、目的地までの経路を適宜生成することで実現できる。

この地図作成とナビゲーションによって 1.2.2 項で述べた先行研究の問題点の iii.を解決することができる。すなわち、自律移動ロボットが決められた順番で部屋の中を移動するのではなく、目的地を指定される度にロボットが経路を算出して自律移動するのである。

この地図作成とナビゲーションの機能は ROS（Robot Operating System）というミドルウェアにより提供されており、利用者がプログラムを書くことなく容易に利用可能となっている。

この ROS では地図作成を行うための様々な手法が提供されているので、それらの精度を第 3 章で比較する。得られた地図を用いたナビゲーションについては第 4 章で解説する。

第 2 章　使用機器（宮前担当）

2.1　ROS(Robot Operating System)

ROS（Robot Operating System）とは、ロボット用のソフトウェアプラットフォームである。歩行ロボットや移動ロボットなどで多くの利用例があり、ロボット開発において非常に役立つシステムとなっている。本研究では ROS の地図作成とナビゲーションの機能を主に利用する。

図 2-1 は、ROS をインストールした PC（Panasonic 製 Let's note CF-SZ5）であり、OS としては Linux ディストリビューションの一つである Ubuntu 16.04 LTS を用いる。

図 2-1　本研究で使用した PC（Panasonic 製 Let's note CF-SZ5）

2.2　使用するロボット

自律移動ロボットとして我々は二種類のロボットを用いる。一つ目は、先行研究で使用してきた iRobot 製の Roomba 622 である（図 2-2)。そして二つ目は、TurtleBot3 Waffle Pi である(図 2-3）。TurtleBot3 は ROS の公式ロボットであり、ROS との相性が良いと考えられるため使用する。

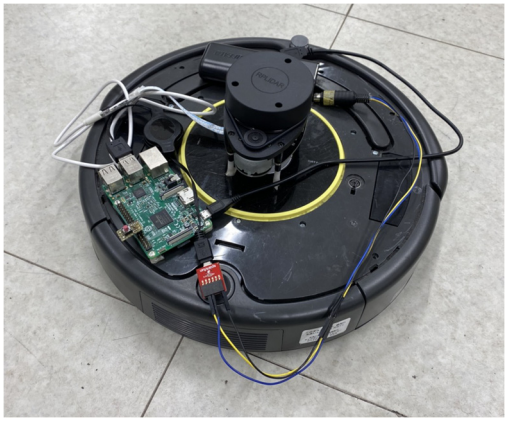

図 2-2　iRobot 製 Roomba 622

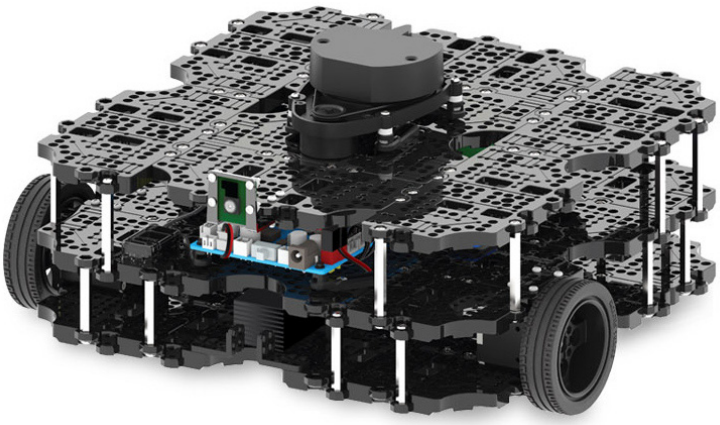

図 2-3　TurtleBot3 Waffle Pi [6]

2.3　ロボットに搭載する機器

Roomba には Raspberry Pi、LiDAR、バッテリーなどが搭載されており、TurtleBot3 にはそれらに加えてマイコンボード OpenCR が搭載されている。以下で解説する。

2.3.1　Raspberry Pi

図 2-4 の Raspberry Pi はシングルボードコンピュータと呼ばれる超小型コンピュータである。Roomba の場合も TurtleBot3 の場合も、この Raspberry Pi がロボットを制御するためのコンピュータとなる。一方、図 2-1 のノート PC は、この Raspberry Pi と通信し命令を与えるための PC という位置づけである。

図 2-4　Raspberry Pi 3 Model B [7]

ノート PC には OS として Ubuntu 16.04 LTS を用いるため、Raspberry Pi の OS も同系統の Ubuntu MATE 16.04 LTS を用いる。なお、Ubuntu MATE 16.04 は 2016 年 2 月に発表された Raspberry Pi 3 Model B までしかサポートしておらず、その後に発表された Raspberry Pi 3 Model B+や Raspberry Pi 4 では動作しないので注意が必要である。

2.3.2　OpenCR

図 2-5 に示したのが OpenCR と呼ばれるマイコンボードである。TurtleBot3 にはこの OpenCR がロボット制御用に搭載されている。ROS がインストールされた Raspberry Pi が OpenCR に命令を送信し、OpenCR がロボットを直接制御するという仕組みである。

Roomba の場合は ROI（Roomba Open Interface）という仕組みにより Raspberry Pi が直接 Roomba を制御できるためマイコンボードは不要となっている。

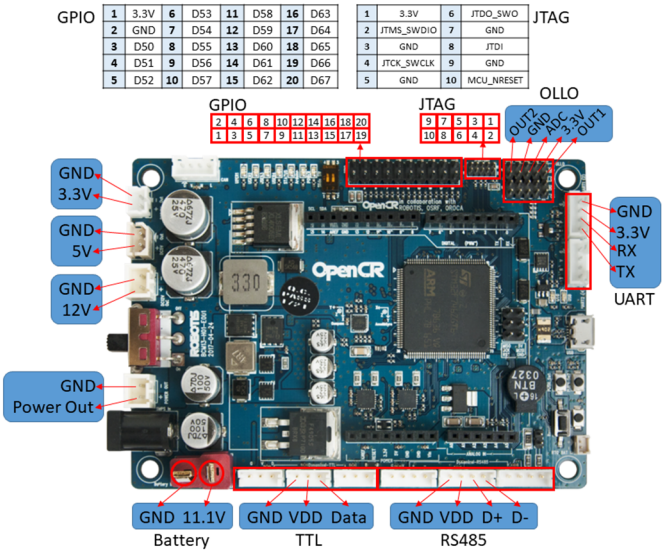

図 2-5　OpenCR [6]

2.3.3　LiDAR

1.3.1 で述べたように LiDAR はレーザーを用いて周囲の障害物までの距離を測るセンサであり、測域センサとも呼ばれる。

Roomba の場合、図 2-6 の RPLIDAR A1 と呼ばれる LiDAR を用いる。

TurtleBot3 の場合は図 1-6 に示した LDS-01 を用いる。

図 2-6　RPLIDAR A1M8 [8]

2.3.4　バッテリー

ロボットに電力を供給するためにバッテリーが必要である。

Roomba の場合、ロボット動作用バッテリーは Roomba に内蔵されており、Raspberry Pi 用のバッテリーは市販の USB モバイルバッテリーを利用する。

一方 TurtleBot3 の場合、図 2-7 の Li-Po バッテリーが、ロボット・Raspberry Pi・OpenCR の全てに電力を供給する。

図 2-7　Li-Po バッテリー LB-012 [9]

2.3.5　ロボットに用いられる機器のまとめ

以上で解説した、ロボットに用いられる機器をまとめたのが表 2-1 である。

表 2-1 ロボットに用いられる機器のまとめ

項目	Roomba	TurtleBot3
命令送信用 PC	Let's note CF-SZ5 (Ubuntu 16.04 LTS)	
ロボット制御用 PC	Raspberry Pi 3 Model B (Ubuntu MATE 16.04 LTS)	
マイコンボード	なし	OpenCR
LiDAR	RPLIDAR A1	LDS-01
ロボット用バッテリー	Roomba 内蔵バッテリー	Li-Po バッテリー LB-012
Raspberry Pi 用バッテリー	USB モバイルバッテリー	

2.4　ROS のためのネットワーク設定

2.1 節および 2.3 節では、命令送信用 PC である Let's note CF-SZ5 とロボット制御用 PC の Raspberry Pi 3 Model B にそれぞれ Linux 系 OS である Ubuntu 16.04 を搭載することを紹介した。さらにそれらには、ロボット制御用のミドルウェアある ROS をインストールする。

ロボットを ROS で制御するためには、命令送信用 PC とロボット制御用 PC が同一のネットワークに属し、適切に設定されていなければならない。本節ではその設定について簡単に解説する。

図 2-8 は、ROS を利用するために構築した研究室のネットワークの概要を表している。命令送信用 PC である Let's note CF-SZ5 が中央に、Roomba および TurtleBot3 が左右に配置されており、それぞれの IP アドレスやマシン名が記されている。IP アドレスはすべて固定としている。

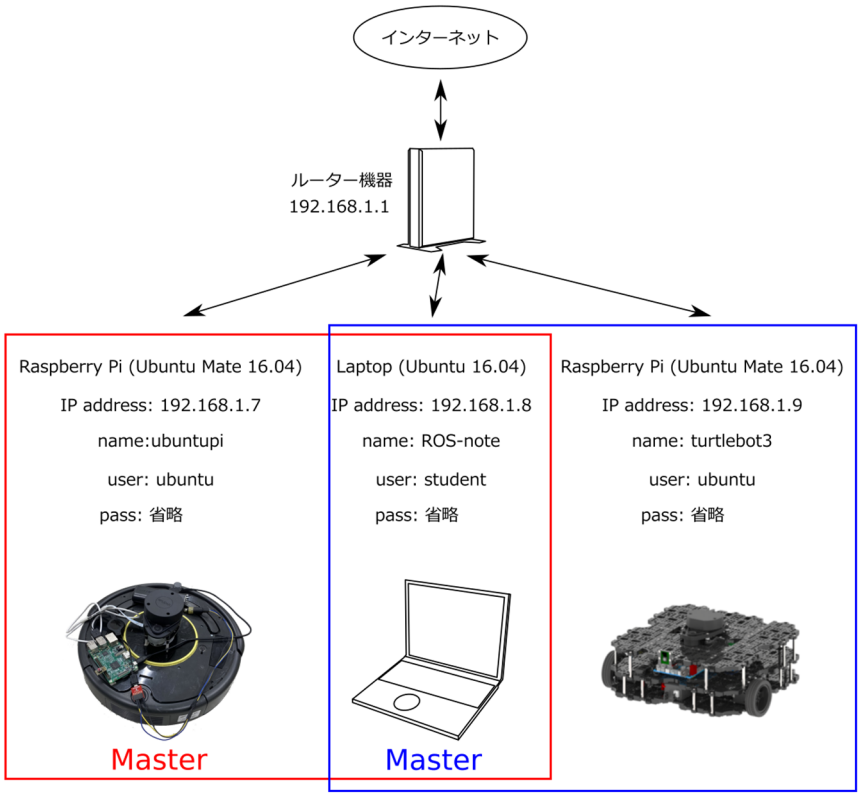

図 2-8 ROS を利用するための研究室のネットワーク

ROS には Master/Slave の概念があるが、Roomba を制御する場合は Roomba 上の

Raspberry Pi が Master に、TurtleBot3 を制御する場合は Let's note CF-SZ5 が Master になることも図に示されている。これは、それぞれのロボットを制御するために用いるプログラム（後述）のデフォルトの設定を踏襲している。

なお、ROS でロボットを制御する場合、Master と Slave（すなわち、Let's note CF-SZ5 とロボット上の Raspberry Pi）とで時刻が揃っている必要がある。本研究では Ubuntu 16.04 の時刻合わせに timesyncd を利用した。大学内などのプロキシ環境ではファイル /etc/systemd/timesyncd.conf 内に時刻合わせ用の学内サーバーを記述する必要がある。

このとき、ROS の更新などを行うと、chrony という時刻合わせ用のパッケージが導入され、timesyncd による時刻合わせに失敗しその結果 ROS が正常動作しなくなることがある。その場合「sudo apt remove chrony」などのコマンドで chrony を削除することで timesyncd による時刻合わせが再開される。

さて、ROS により Roomba を制御するため、下記で解説されている Roomblock というプログラムを Let's note CF-SZ5 および Roomba 上の Raspberry Pi に導入する **[10]**。

- Roomblock: a Platform for Learning ROS Navigation With Roomba, Raspberry Pi and RPLIDAR
- https://www.instructables.com/Roomblock-a-Platform-for-Learning-ROS-Navigation-W/

このとき、Master である Roomba 上の Raspberry Pi では、下記の環境変数を設定ファイルの.bashrc に記して設定する。どちらも、Roomba 上の Raspberry Pi の IP アドレスが指定されている（図 2-8 参照）。

```
export ROS_HOSTNAME=192.168.1.7
export ROS_MASTER_URI=http://192.168.1.7:11311
```

一方、Slave である Let's note CF-SZ5 では、下記のように設定する。ROS_HOSTNAME には自分自身の IP アドレスが、ROS_MASTER_URI には Master の IP アドレスが指定されている。

```
export ROS_HOSTNAME=192.168.1.8
export ROS_MASTER_URI=http://192.168.1.7:11311
```

一方、TurtleBot3 を制御するためには、TurtleBot3 用の ROS の公式パッケージを導入する必要がある[11]。

- TurtleBot3: 3. Quick Start Guide
- https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/

このとき、Master である Let's note CF-SZ5 では、下記の環境変数を設定ファイルの.bashrc に記して設定する。どちらも、Let's note CF-SZ5 の IP アドレスが指定されている（図 2-8 参照)。

```
export ROS_HOSTNAME=192.168.1.8
export ROS_MASTER_URI=http://192.168.1.8:11311
```

一方、Slave である TurtleBot3 上の Raspberry Pi では、下記のように設定する。ROS_HOSTNAME には自分自身の IP アドレスが、ROS_MASTER_URI には Master の IP アドレスが指定されている。

```
export ROS_HOSTNAME=192.168.1.9
export ROS_MASTER_URI=http://192.168.1.8:11311
```

以上のように、Let's note CF-SZ5 は Roomba と TurtleBot3 のどちらを制御するかで設定が異なる。実際には.bashrc に下記のように記し、状況に応じて有効な ROS_MASTER_URI を変更するようにしている。

```
export ROS_HOSTNAME=192.168.1.8
### for roomblock
export ROS_MASTER_URI=http://192.168.1.7:11311

### for turtlebot3
export ROS_MASTER_URI=http://192.168.1.8:11311
```

2.5　TurtleBot3 での Cartographer の利用

本研究で用いる ROS は、OS である Ubuntu のバージョンと対応付けられており、本研究で用いる Ubuntu 16.04 LTS に対しては ROS Kinetic というバージョンを利用することになる。

ROS Kinetic で TurtleBot3 を制御し、さらに Google が開発した SLAM ツールである Cartographer を利用する場合、Cartographer のバージョンを ROS Kinetic が必要とするバージョンに合わせてインストールしなければならなかった。

具体的には、Cartographer のダウンロード時に下記のようにバージョン指定を行った。ダウンロード後の Cartographer のインストール法は、TurtleBo3 公式ドキュメントの「4.1. Run SLAM Node」に記されている。

```
cd ~/catkin_ws/src
git clone https://github.com/googlecartographer/cartographer.git
git clone https://github.com/googlecartographer/cartographer_ros.git
cd cartographer
git checkout f83ba3db2f336c147500e6874cebaaf6252e8711
cd ../cartographer_ros
git checkout c4ac76a579522989e00be85d11cf4646403a65fa
```

第３章　地図作成（山下担当）

3.1　概要

3.1.1　目的

1.3.2 項で述べたように、我々は自律移動ロボットには地図作成とナビゲーションの機能が必要であると考えている。本章ではそのうち地図作成について解説する。

3.1.2　要求仕様

Roomba と TurtleBot3 の２つのロボットを用いて地図作成を行う場合、その手法として下記の 5 つが ROS により提供されている。これらの精度を比較する。なお、frontier_exploration と cartographer は TurtleBot3 でのみ実験を行った。

- gmapping
- hector
- karto
- frontier_exploration
- cartographer

3.2　地図作成の手順

3.2.1　概要

本節では、地図作成の手順を解説する。

3.2.2 Roomba での地図作成の手順

Roomba での地図作成の手順は以下の通りである。

1. Let's note CF-SZ5 のターミナルで以下のコマンドを実行する。このコマンドにより、設定ファイル「.bashrc」をテキストエディタ「gedit」で開くことができる。

```
gedit .bashrc
```

2. 開いたファイルの文末付近に下記のように表示される。先頭に「#」ついている行はコメントとして無効化されていること表す。下記 5 行目の先頭に「#」を付けることで、TurtleBot3 用の設定を無効化し、Roomba 用の設定のみを有効化する。

```
### for roomblock
export ROS_MASTER_URI=http://192.168.1.7:11311


### for turtlebot3
#export ROS_MASTER_URI=http://192.168.1.8:11311
```

3. Roomba を起動する。Roomba 上の Raspberry Pi では、下記のコマンドが自動で実行されるよう設定されている。このプログラムが、Let's note CF-SZ5 からの命令を受け取り、Roomba を制御することになる。

```
roslaunch roomblock_bringup roomblock.launch
```

4. Let's note CF-SZ5 で以下のコマンドを実行し、キーボード操作アプリケーションを起動する。

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py _speed:=0.2 _turn:=0.2
```

5. キーボード操作アプリケーションを実行したまま、Let's note CF-SZ5 の別ターミナルまたは別タブで以下のコマンドを実行し、地図作成用アプリケーションを起動する。gmapping.launch の部分は地図作成の手法に対応したファイル名を表しており、他の手法のファイル（hector.launch、karto.launch）に変更することが出来る。

```
roslaunch roomblock_mapping gmapping.launch launch_joy:=false
```

6. Let's note CF-SZ5 上でキーボード操作アプリケーションが実行されているターミナルまたはタブを選択し、操作用キーを押すことでロボットを操作する。ロボットの動作に応じて、地図が作成されていく。キーボードでの操作方法は以下の表 3-1 の通りである。

表 3-1　キーボードでの操作方法

進行方向	操作キー
前進	I
後退	<
右折	L
左折	J
停止	K

7. ある程度地図が描けたと思ったら、Let's note CF-SZ5 のターミナルで以下のコマンドを実行する。このコマンドにより、ユーザのホームディレクトリに二つのファイル map.pgm と map.yaml が保存される。これが作成した地図となる。

```
rosrun map_server map_saver -f ~/map
```

3.2.3　TurtleBot3 での地図作成の方法

TurtleBot3 での地図作成の手順は以下の通りである。

1. 3.2.2 項と同様に Let's note CF-SZ5 のターミナルで以下のコマンドを実行する。

```
gedit .bashrc
```

2. 開いたファイルの文末付近に下記のように表示される。3.2.2 項と同様に、先頭に「#」ついている行はコメントとして無効化されていること表す。2 行目の先頭に「#」を付けることで、Roomba を無効化し、TurtleBot3 のみを有効化したことになる。

```
### for roomblock
#export ROS_MASTER_URI=http://192.168.1.7:11311


### for turtlebot3
export ROS_MASTER_URI=http://192.168.1.8:11311
```

3. Let's note CF-SZ5 のターミナルで以下のコマンドを実行する。これは、ROS の Master で実行されるべきコマンドである。

```
roscore
```

4. TurtleBot3 を起動する。TurtleBot3 上の Raspberry Pi では、下記のコマンドが自動で実行されるよう設定されている。このプログラムが、Let's note CF-SZ5 からの命令を受け取り、TurtleBot3 を制御することになる。

```
roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

5. TurtleBot3 の起動後、Let's note CF-SZ5 の別ターミナルまたは別タブで以下のコマンドを実行する。

```
roslaunch turtlebot3_bringup turtlebot3_remote.launch
```

6. Let's note CF-SZ5 の別ターミナルまたは別タブで以下のコマンドを実行し、キーボード操作アプリケーションを起動する。キーボードによる TurtleBot3 の操作方法は表 3-1 のとおりである。

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py    _speed:=0.2 _turn:=0.2
```

7. キーボード操作アプリケーションを実行したまま、Let's note CF-SZ5 の別ターミナル

または別タブで以下を実行し、地図アプリケーションを起動する。「gmapping」の部分は地図作成の手法を表しており、他の手法に変更することが出来る。

```
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

8. ある程度地図が描けたと思ったら、Let's note CF-SZ5 のターミナルで以下のコマンドを実行する。このコマンドにより、ユーザのホームディレクトリに二つのファイル map.pgm と map.yaml が保存される。これが作成した地図となる。

```
rosrun map_server map_saver -f ~/map
```

3.3　距離測定

3.3.1　概要

3.1 で紹介した 5 つの地図作成の手法の精度を比べるため、ロボットにより作成された地図をもとに計測した距離と、実際の距離との間にどの程度誤差があるのかを測定する。

3.3.2　測定方法

実験は工学院大学八王子キャンパスの廊下で行う。図 3-1 のようにロボットのスタート位置から 5 m および 10m の位置にデスクパーティションを設置した環境を考える。この実験環境内でロボットを Let's note CF-SZ5 のキーボードにより手動制御し、地図作成を行う。

図 3-1　距離測定での研究環境

距離測定の手順は以下の通りである。

1. Roomba では 3.2.2 項に記した方法で、TurtleBot3 では 3.2.3 項に記した方法で地図作成を行う。得られた地図の例が図 3-2 である。
2. 作成された地図が表示されている rViz というアプリケーション上にある、Measure ボタンをクリックする。
3. 計測地点を選択し垂直に線を伸ばしデスクパーティションまでの距離を計測する
4. 図 3-4 のように、画面の左下に計測した数値が表される。
5. 上記の手順を各 3 回行い、それぞれ平均を求める。

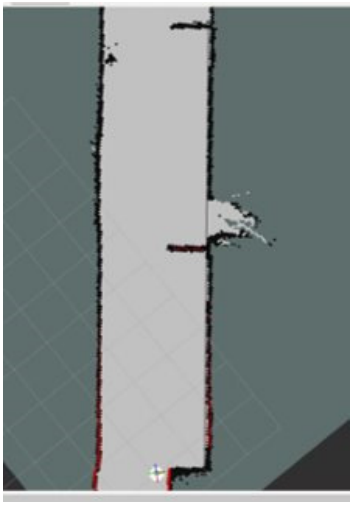

図 3-2　ロボットによって作成された地図

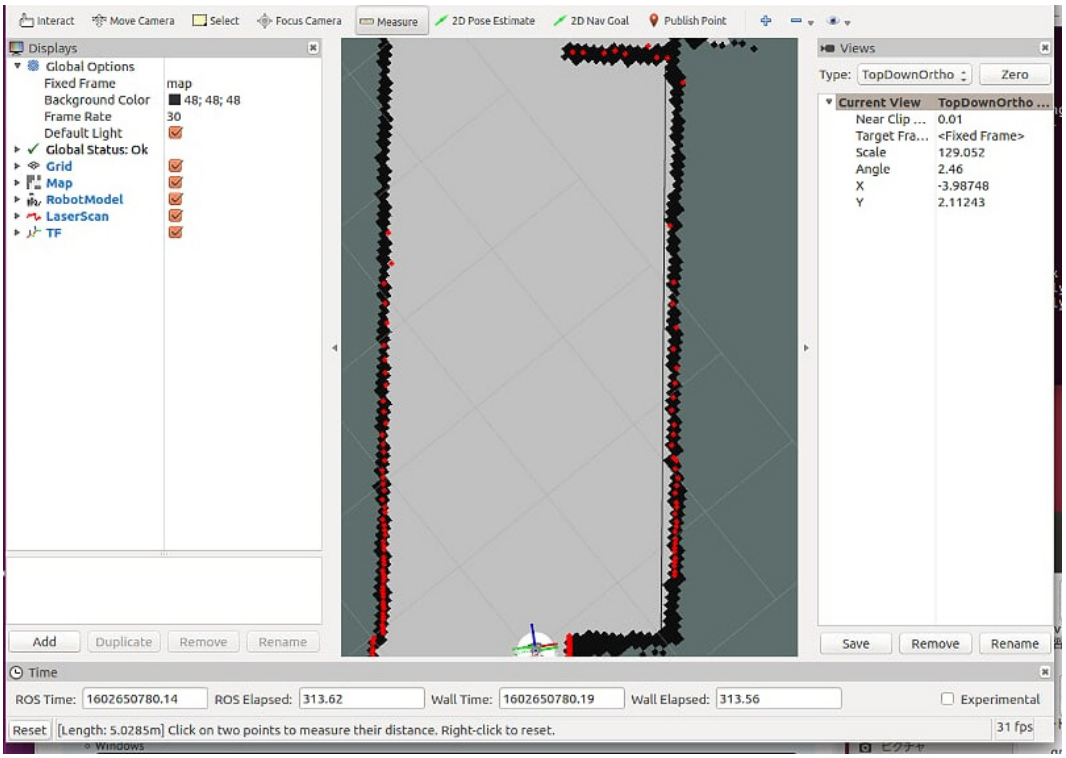

図 3-3　PC 上での距離測定時の画面

図 3-4　測定値が表されている画面の拡大図

3.3.3　測定結果

測定結果を表とグラフで表していく。

まず、Roomba により 5m および 10m のデスクパーティションまでの距離を測定した結果を表 3-2 および表 3-3 に示した。さらに、実際の距離との誤差を図 3-5 と図 3-6 にグラフで示した。なお、Roomba を制御するために用いた Roomblock では gmapping、hector、karto の 3 つの手法のみを取り扱うことができる。

5m の場合も 10m の場合も、gmapping の精度が最も良いことがわかる。

表 3-2　Roomba での測定結果（5 m）

手法	測定結果(m)			
	1 回目	2 回目	3 回目	平均
gmapping	5.00889	5.00394	5.00139	5.00474
hector	5.08747	5.09266	5.1178	5.09931
karto	5.02281	5.01143	5.05126	5.0285

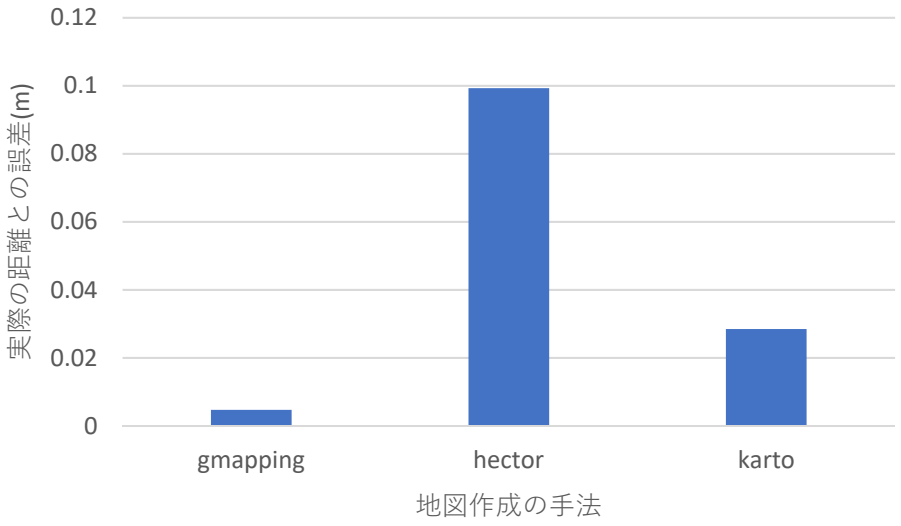

図 3-5　Roomba で測定した距離実際の距離との誤差（5 m）

表 3-3　Roomba での測定結果(10m)

手法	測定結果(m)			
	1 回目	2 回目	3 回目	平均
gmapping	10.3247	10.2534	10.0696	10.2159
hector	10.353	10.2495	10.1535	10.252
karto	10.2873	10.2006	10.3605	10.2828

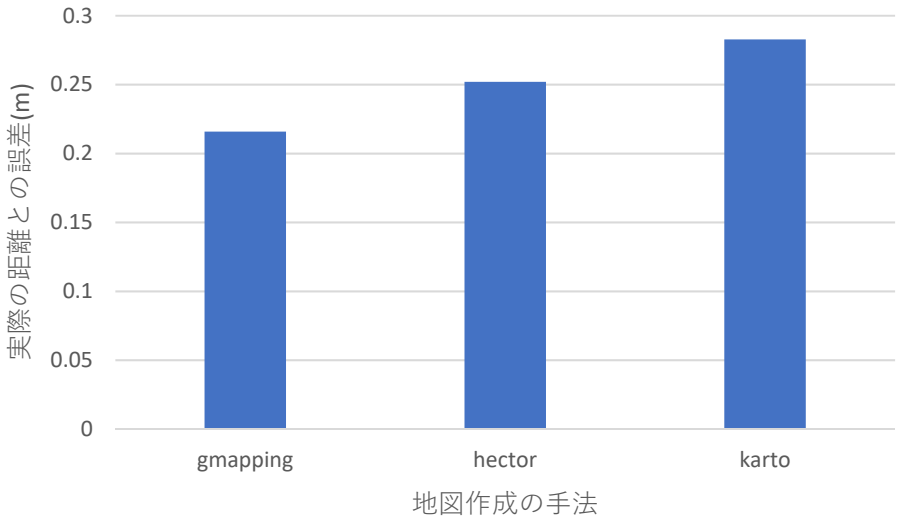

図 3-6　Roomba で測定した距離と実際の距離との誤差(10m)

一方、TurtleBot3 により 5m および 10m のデスクパーティションまでの距離を測定した結果を表 3-4 および表 3-5 に示した。さらに、実際の距離との誤差を図 3-7 と図 3-8 にグラフで示した。

5m の場合も 10m の場合も、hector の精度が最も良いことがわかる。

表 3-4　TurtleBot3 での測定結果(5 m)

手法	測定結果(m)			
	1 回目	2 回目	3 回目	平均
gmapping	4.7791	4.7001	4.71554	4.73158
hector	4.9683	4.7961	4.90303	4.94481
karto	4.6981	4.7961	4.63148	4.70856
frontier_exploration	4.61382	4.96187	4.37539	4.65036
cartographer	4.83167	4.06783	4.33587	4.41179

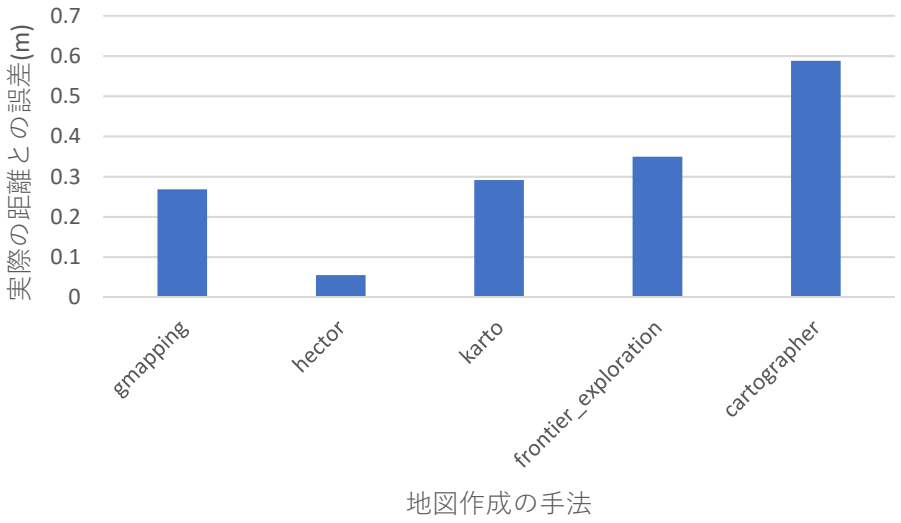

図 3-7　TurtleBot3 で計測した距離と実際の距離との誤差(5 m)

表 3-5　TurtleBot3 での測定結果(10m)

手法	測定結果(m)			
	1 回目	2 回目	3 回目	平均
gmapping	9.70289	9.75441	9.77725	9.74485
hector	9.95641	10.0089	10.04519	10.0035
karto	8.49674	8.98784	8.97458	8.81972
frontier_exploration	9.66574	9.61422	9.55505	9.61167
cartographer	9.71331	9.58776	9.52227	9.60778

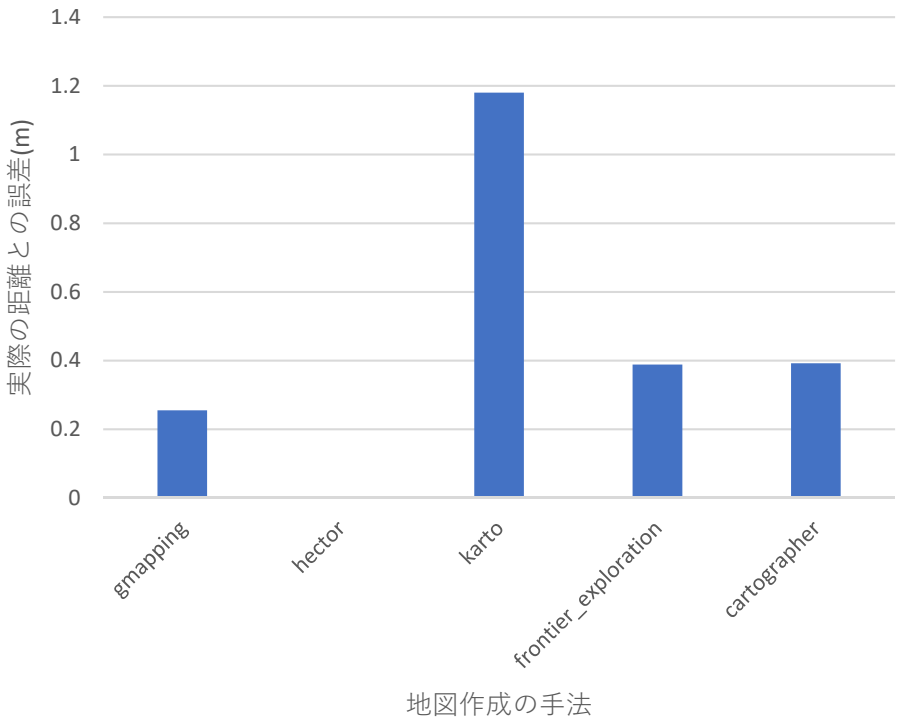

図 3-8　TurtleBot3 で計測した距離と実際の距離との誤差(10m)

3.4　ロボットの速度を変えた距離測定

3.4.1　概要

ロボットの速度を 2 倍に速めた場合、速度を速める前と比べて、測定精度にどのような影響があるのか実験する。

3.4.　測定方法

実験環境は、距離測定と同じ環境を使用する（3.3.2 項）。

実験手順も距離測定と同じ手順（3.3.2 項）で行うが、キーボード操作アプリケーションを起動するコマンドにおいて、速度の数値を以下のように 2 倍の 0.4 と変更する（Roomba では 3.2.2 項、TurtleBot3 では 3.2.3 項）

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py  _speed:=0.4 _turn:=0.2
```

3.4.3　測定結果

測定結果を表とグラフで表していく。

まず、Roomba により 5m および 10m のデスクパーティションまでの距離を測定した結果を表 3-6 および表 3-7 に示した。さらに、実際の距離との誤差を図 3-9 と図 3-10 にグラフで示した。

5m の場合は gmapping と karto の精度が良いことがわかる。

表 3-6　Roomba での測定結果（5 m）

手法	測定結果(m)			
	1 回目	2 回目	3 回目	平均
gmapping	5.0512	5.0399	5.0544	5.0485
hector	5.0641	5.0564	5.0751	5.0652
karto	4.9518	4.9901	4.91443	4.95211

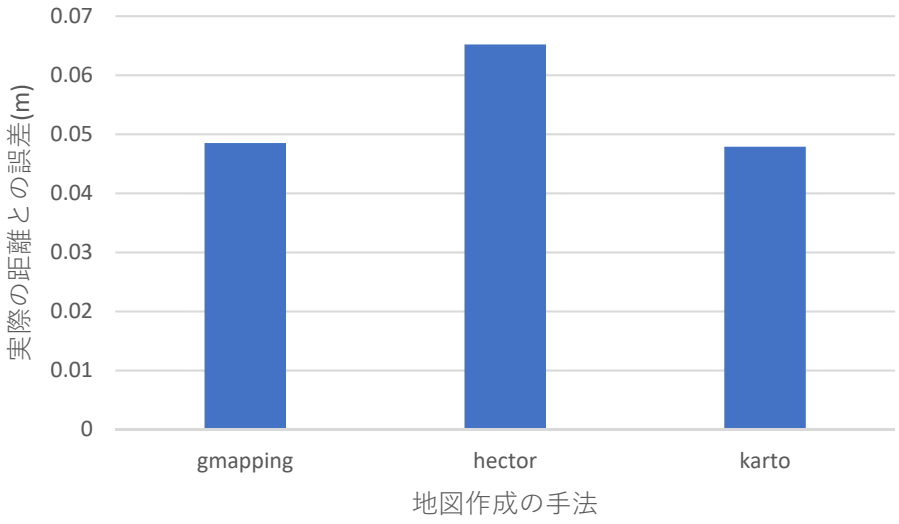

図 3-9　Roomba で測定した距離と実際の距離との誤差(5m)

一方、10m の場合は karto の精度が良いことがわかる。

表 3-7　Roomba での測定結果(10m)

手法	測定結果(m)			
	1 回目	2 回目	3 回目	平均
gmapping	10.3104	10.1019	10.1628	10.1917
hector	10.0879	10.1518	10.3402	10.1933
karto	9.765	10.8012	9.5589	10.0417

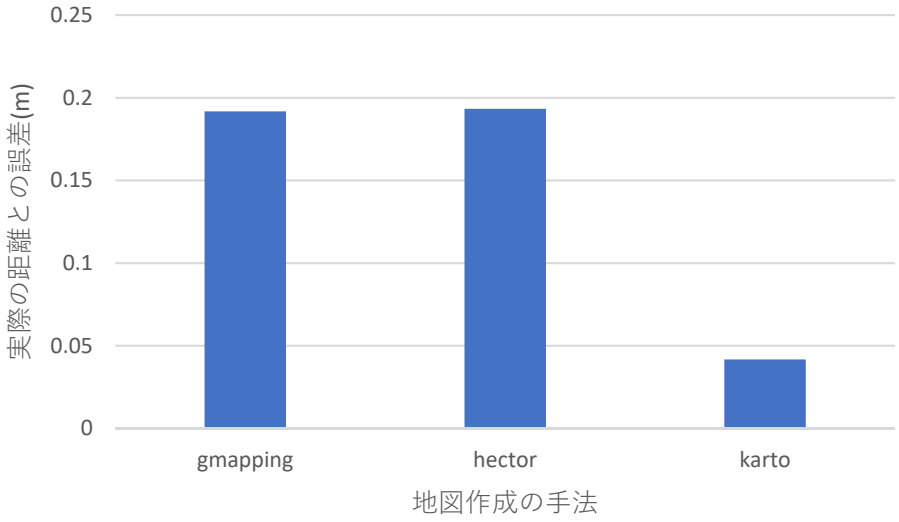

図 3-10　Roomba で測定した距離と実際の距離との誤差(10m)

TurtleBot3 により 5m および 10m のデスクパーティションまでの距離を測定した結果を表 3-8 および表 3-9 に示した。さらに、実際の距離との誤差を図 3-11 と図 3-12 にグラフで示した。

まず、karto では地図が作成できなかったことに注意しなければならない。

それ以外は、5m の場合も 10m の場合も、hector の精度がおおむね良いことがわかる。

表 3-8　TurtleBot3 での測定結果(5 m)

手法	測定結果(m)			
	1 回目	2 回目	3 回目	平均
gmapping	4.5167	4.8352	4.605	4.6523
hector	4.9091	4.91342	4.88054	4.90102
karto	×	×	×	×
frontier_exploration	4.8135	4.79135	4.86097	4.82194
cartographer	4.73641	4.98271	4.70056	4.80656

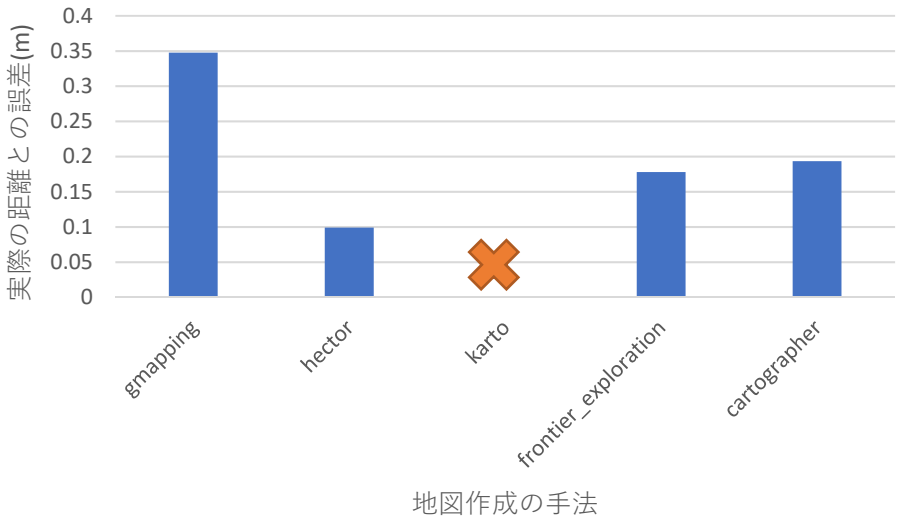

図 3-11　TurtleBot3 で計測した距離と実際の距離との誤差(5 m)

表 3-9　TurtleBot3 での測定結果(10m)

手法	測定結果(m)			
	1 回目	2 回目	3 回目	平均
gmapping	9.67125	9.70127	9.67567	9.68273
hector	9.95041	9.90952	10.0282	9.96271
karto	×	×	×	×
frontier_exploration	9.73234	10.04239	10.147159	9.973963
cartographer	9.80744	9.91883	9.95315	9.89314

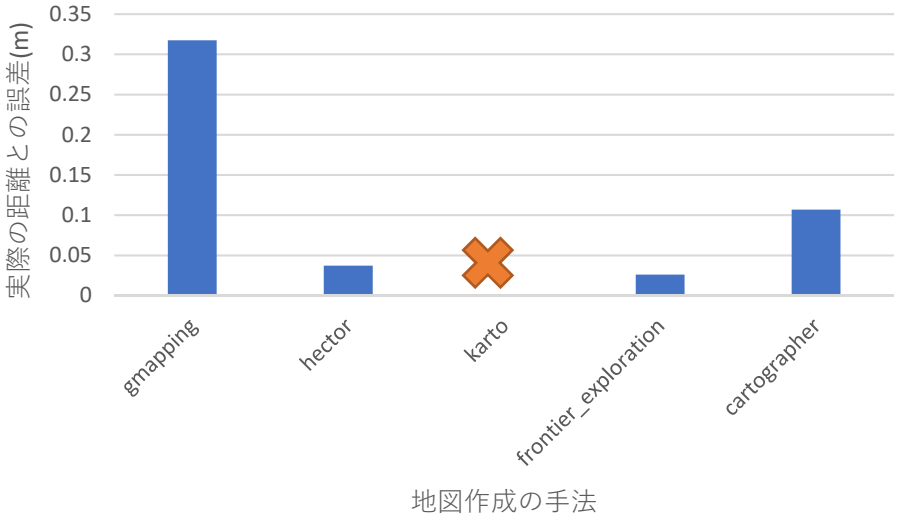

図 3-12　TurtleBot3 で計測した距離と実際の距離との誤差(10m)

次に、Roomba と TurtleBot3 のそれぞれのロボットで一つの手法に着目したとき、速度変化による誤差の変化をグラフ化する。

図 3-13 は、Roomba で gmapping を用いた場合の速度変化による誤差の変化を示している。まず、5m と 10m の測定結果を比較すると、10m の方が誤差が大きいという結果が得られており、これは直観に合う結果である。

また、5m における測定誤差は、速度が 2 倍になったときに大きくなっており、これも直観に合う結果である。一方、10mにおける測定誤差は、速度が 2 倍になったときに小さくなっており、これは直観に反する。これは、図 3-3 のように手動で計測したことによる誤差や、平均回数が少ないことなどが影響していると考えられる。再実験を行うなどより詳細な検討が必要であろう。

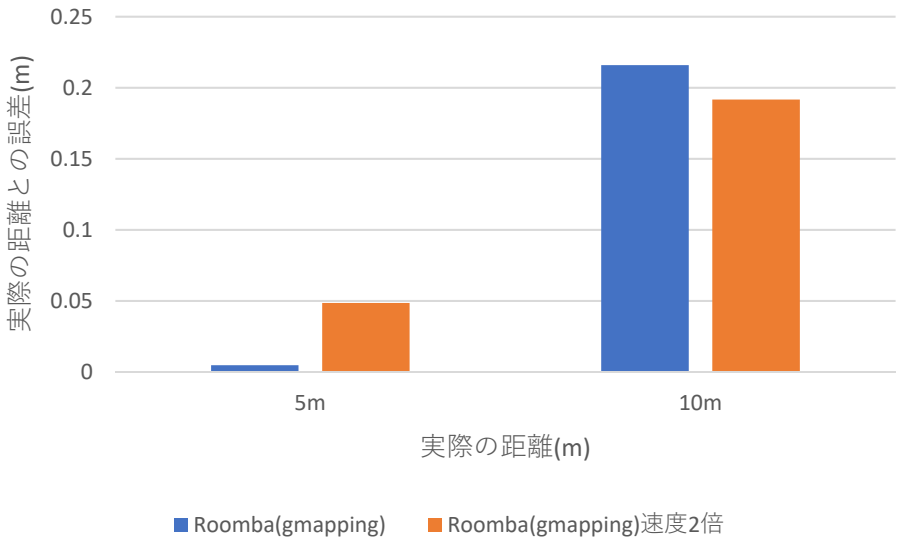

図 3-13　Roomba(gmapping)での速度変化前後の測定誤差

図 3-14 は、TurtleBot3 で hector を用いた場合の速度変化による誤差の変化を示している。

こちらの場合、速度が 2 倍になることで誤差が大きくなっており、これは直観に合う結果である。

しかし、5m よりも 10m の方が誤差が小さいことは直観に反する結果であり、やはり再実験を行うなどより詳細な検討が必要である。

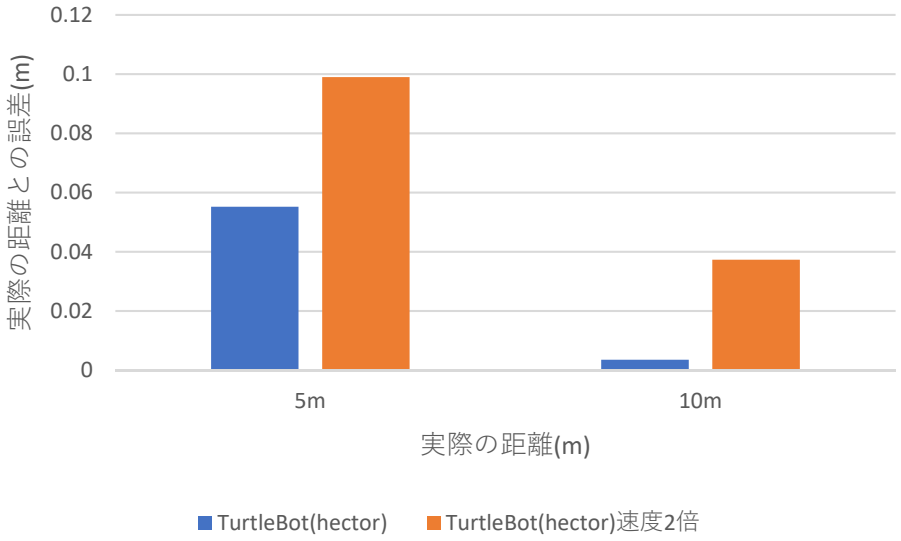

図 3-14　TurtleBot3(hector)での速度変化前後の測定誤差

3.5　障害物が多い環境での距離測定

3.5.1　概要

多数の障害物がある環境の中で地図作成を行い、地図から計測した距離と実際の距離との誤差を測定し、5 つの地図作成の手法の精度を比べる。

3.5.2　測定方法

図 3-15 のように、研究室内で障害物が多数ある領域を実験環境とする。

この環境下でロボットを Let's note CF-SZ5 のキーボードにより手動制御し、冷蔵庫・デスクパーティション・机の 3 つの物体の長さを測定する。

図 3-15　障害物が多い実験環境

冷蔵庫・デスクパーティション・机の長さは以下の**表 3-10** の通りである。また、それぞれの写真を図 3-16 から図 3-18 に示した。

表 3-10　3 つの障害物の長さ

障害物	長さ(m)
冷蔵庫	0.475
デスクパーティション	0.905
机	1.125

図 3-16　実験で使用した冷蔵庫

図 3-17　実験で使用したデスクパーティション

図 3-18　実験で使用した机

距離測定の手順は以下の通りである。

1. Roomba では 3.2.2 項に記した方法で、TurtleBot3 では 3.2.3 項に記した方法で地図作成を行う。得られた地図の例が図 3-19 である。赤で囲んでいる部分が冷蔵庫、青で囲んでいる部分がデスクパーティション、緑で囲んでいる部分が机を表している。
2. 作成された地図が表示されている rViz というアプリケーション上にある、Measure ボタンをクリックする。
3. 基準となる障害物の端から端を選択する。
4. 画面の左下に計測した数値が表される。
5. 上記の手順を各 3 回行い、平均を求める。

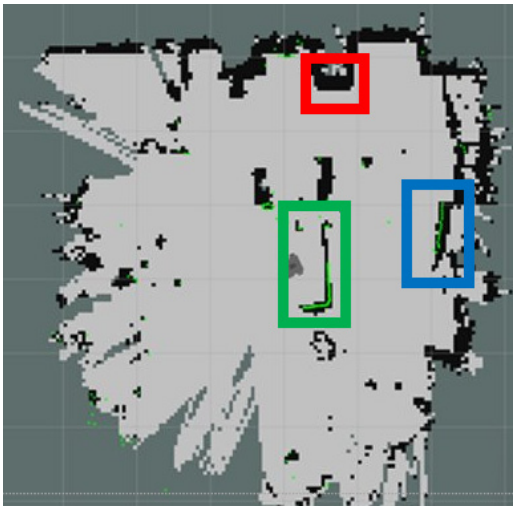

図 3-19　障害物が多い実験環境で作成した地図

3.5.3　測定結果

測定結果は以下の表とグラフで表す。

Roomba で作成した地図により、3 つの対象物の長さを測定した結果が表 3-11 から表 3-13 に示されている。

表 3-11　Roomba での測定結果(冷蔵庫)

Roomba(冷蔵庫)				
手法	測定結果(m)			
	1 回目	2 回目	3 回目	平均
gmapping	0.53117	0.55158	0.52672	0.53649
hector	0.50796	0.50372	0.49186	0.50796
karto	0.55643	0.59772	0.596257	0.583469

表 3-12　Roomba での測定結果(デスクパーティション)

Roomba(デスクパーティション)				
手法	測定結果(m)			
	1 回目	2 回目	3 回目	平均
gmapping	1.11107	1.00735	0.95988	1.0261
hector	0.95742	0.98113	0.983276	0.973942
karto	0.9464	1.02745	0.989598	0.987816

表 3-13　Roomba での測定結果(机)

Roomba(机)				
手法	測定結果(m)			
	1 回目	2 回目	3 回目	平均
gmapping	1.06384	1.2537	0.98966	1.1024
hector	1.15377	1.1549	1.28518	1.19795
karto	1.45271	1.28496	1.2034	1.31369

全ての結果をまとめたグラフが図 3-20 である。hector により地図を作成した場合に最も誤差が小さくなることがわかる。

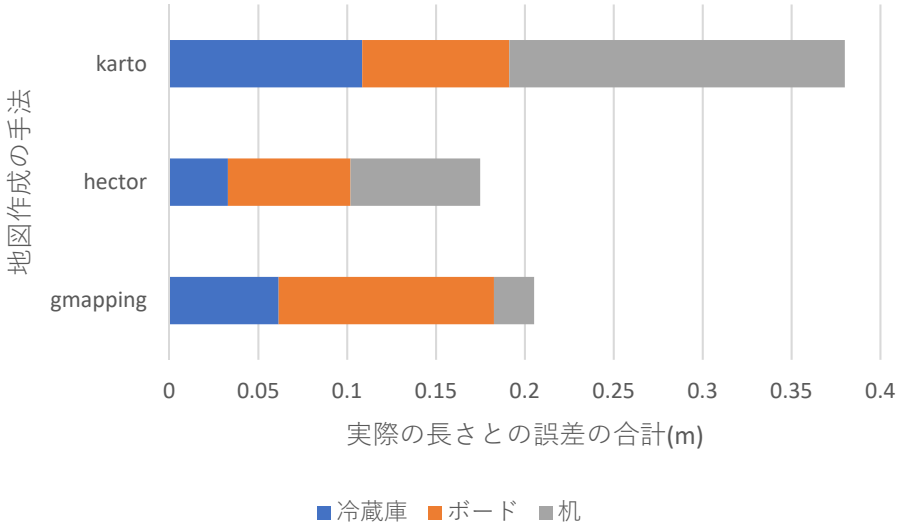

図 3-20　それぞれの地図作成の手法と 3 つの障害物の実際の長さとの誤差の合計(Roomba)

一方、TurtleBot3 での結果を示したのが表 3-14 から表 3-16 である。

表 3-14　TurtleBot3 での測定結果(冷蔵庫)

TurtleBot3(冷蔵庫)				
手法	測定結果(m)			
	1 回目	2 回目	3 回目	平均
gmapping	0.44882	0.45726	0.474268	0.460116
hector	0.48101	0.50277	0.515593	0.499791
karto	0.37644	0.35971	0.322502	0.352884
frontier_exploration	0.59004	0.58146	0.581142	0.584214
cartographer	0.50648	0.51477	0.508525	0.509925

表 3-15　TurtleBot3 での測定結果(デスクパーティション)

TurtleBot3(デスクパーティション)				
手法	測定結果(m)			
	1 回目	2 回目	3 回目	平均
gmapping	1.02587	1.0236	1.03501	1.02816
hector	0.98722	0.94714	0.920059	0.951473
karto	0.97835	0.98114	0.944588	0.968026
frontier_exploration	0.95274	0.96003	0.976599	0.963123
cartographer	1.08045	1.01743	0.96227	1.02005

表 3-16　TurtleBot3 での測定結果(机)

TurtleBot3(机)				
手法	測定結果(m)			
	1 回目	2 回目	3 回目	平均
gmapping	1.14667	1.15486	1.15225	1.15126
hector	1.27455	1.21004	1.12246	1.20235
karto	0.6385	0.57333	0.585191	0.599007
frontier_exploration	1.160803	1.17562	1.164067	1.16683
cartographer	1.28649	1.33756	1.30316	1.30907

全ての結果をまとめたグラフが図 3-21 である。hector により地図を作成した場合に最も誤差が小さくなることがわかる。

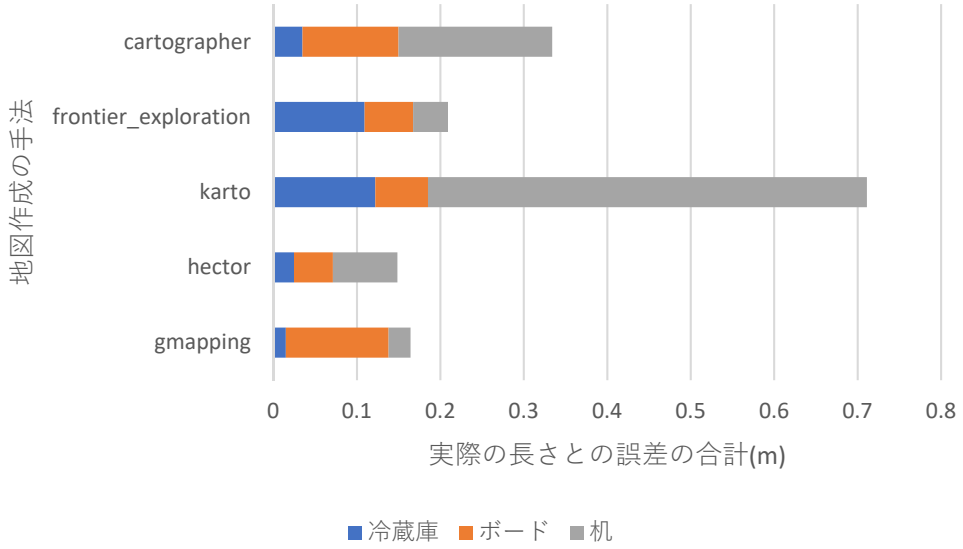

図 3-21　それぞれの地図作成の手法と 3 つの障害物の実際の長さとの誤差の合計 (TurtleBot3)

3.6　認識可能な高さの調査

3.6.1　概要

Roomba と TurtleBot3 の 2 つのロボットは、LiDAR から出力したレーザーが対象物に反射されることで距離を算出することができる。そのため、対象物の高さが低すぎるとレーザーが反射されず、対象物と認識されない。

本実験では、高さ何 cm の物体を認識することが出来るのかを調査する。

3.6.2　調査方法

図 3-22 のようにロボットから 2m の位置に 10cm 四方の箱を設置した環境を考える。箱を設置する高さを少しずつ変えたときに、何 cm から箱がロボットから認識されるかを調べる。

実験の手順は以下の通りである。

1. Roomba または TurtleBot3 を起動し、ROS で制御できる状態にする。
2. ロボットから 2 m の位置に高さ 10cm の箱を設置する
3. rViz 上で箱が認識されているか確認する。
4. 認識するまで箱を設置する位置を高くしていく。
5. 上記の手順を各 3 回行う。

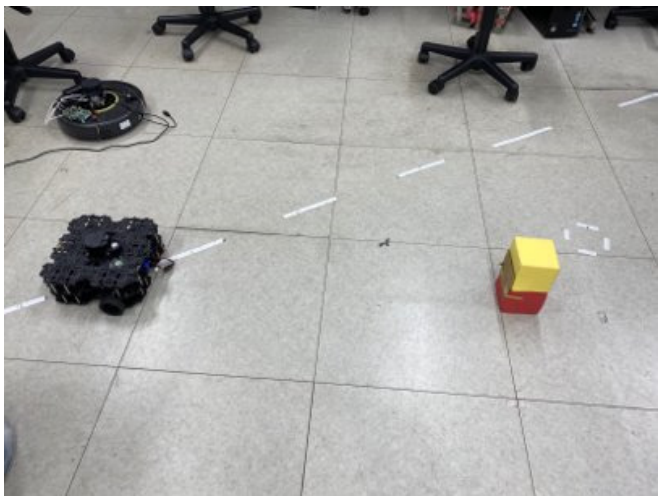

図 3-22　認識可能な高さを調査する実験の様子

図 3-23 は、実験を行っている際の rViz 上の画面である。青い四角で囲んでいる部分がロボットを示しており、赤い四角で囲んでいる部分が認識している箱を示している。

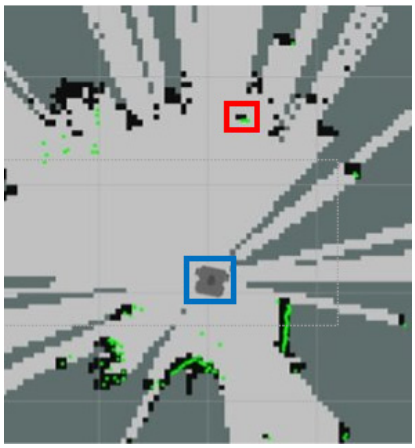

図 3-23　認識可能な高さを調査している時の PC 上の地図

3.6.3　調査結果

箱の高さを変えたとき、箱が認識できたか（○）できなかったか（×）を表 3-17 および表 3-18 に示した。表から、Roomba では高さ 16cm から、TurtleBot3 では高さ 30cm から認識出来ることが分かる。地図作成手法ごとの比較を行ったものの、認識可能な高さは LiDAR を設置する高さや、LiDAR の性能のみによって決まると考えられる。

表 3-17　認識可能な高さの調査結果(Roomba)

手法	高さ		
	10cm	15cm	16cm
gmapping	×	×	○
hector	×	×	○
karto	×	×	○

表 3-18　認識可能な高さの調査結果(TurtleBot3)

手法	高さ				
	10cm	15cm	20cm	25cm	30cm
gmapping	×	×	×	×	○
hector	×	×	×	×	○
karto	×	×	×	×	○
frontier_exploration	×	×	×	×	○
cartographer	×	×	×	×	○

第 4 章　ナビゲーション（君塚担当）

4.1　概要

4.1.1　目的

我々は自律移動ロボットには地図作成機能とナビゲーション機能が必要であると考えている。第 3 章にて作成された地図の精度を調べたので、本章では 5 つの手法で作成した地図の中でナビゲーションに適しているものはどれかを調べる。

4.1.2　要求仕様

本章では、図 4-1 のような作成済の地図をもとにナビゲーションを行う。ナビゲーションの方法を次節で解説する。

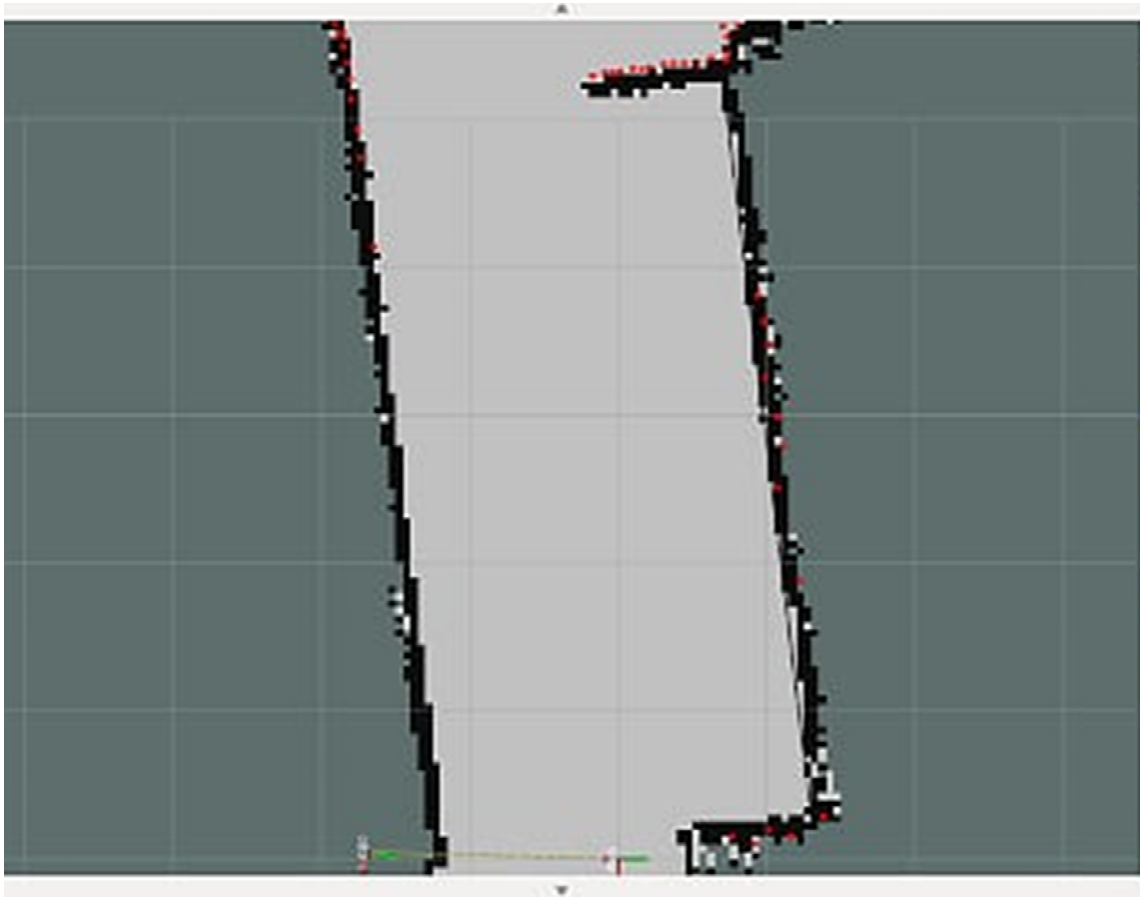

図 4-1　作成した地図の例

4.2　ナビゲーションの方法

4.2.1　概要

本節ではナビゲーションを行う方法を解説する。

4.2.2　ナビゲーションの手順

本実験は図 4-2 のように工学院大学八王子キャンパスの廊下にデスクパーティションを立てた環境で行う。

図 4-2　ナビゲーションの測定環境

作成された地図は図 4-3 の通りである。

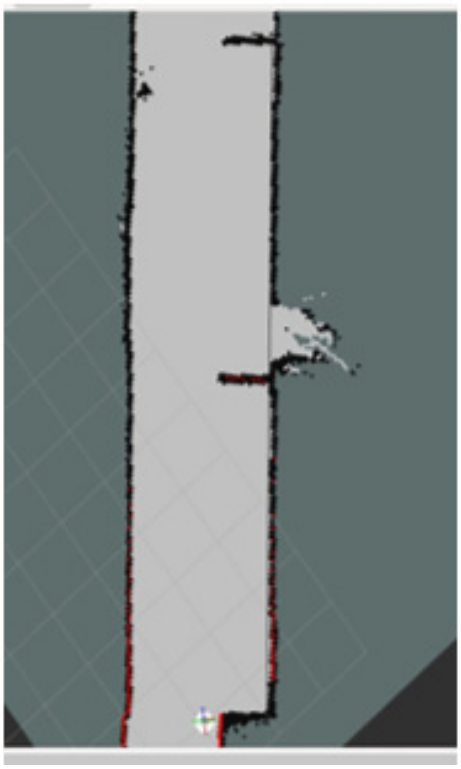

図 4-3　作成された地図

このような環境下でナビゲーションを行うためには、Let's note CF-SZ5 のターミナル上でプログラムを実行する必要がある。

Roomba の場合のコマンドは以下である。途中で折り返されているが、一行のコマンドである。なお、これは 3.2.3 項において map という名称で地図を保存した場合の例ある。別名で地図を保存した場合、「map.yaml」の部分を適宜変更する必要がある。

```
roslaunch roomblock_navigation amcl.launch map_file:=$HOME/map.yaml
with_camera:=false
```

同様に、TurtleBot3 の場合のコマンドは以下である。途中で折り返されているが、一行のコマンドである。

```
roslaunch turtlebot3_navigation turtlebot3_navigation.launch
map_file:=$HOME/map.yaml
```

起動した rViz 上には、図 4-4（左）のように作成された地図（白黒表示）と現在のロボット周囲の環境（RGB 表示）とが重ねて描かれている。白黒表示の地図と RGB 表示の地図は一致すべきものであるが、初期状態では一致していない。

rViz 上で「2D Pose Estimation」をクリックし、白黒表示の地図上でロボットの実際の位置と向きをマウスのドラッグにより指定すると、これらがおおまかに揃う。さらに、地図作成時のようにロボットをキー操作により移動させると、さらにこの 2 つの地図が揃うようになる。2 つの地図がほぼ一致したところを示したのが図 4-4（右）である。

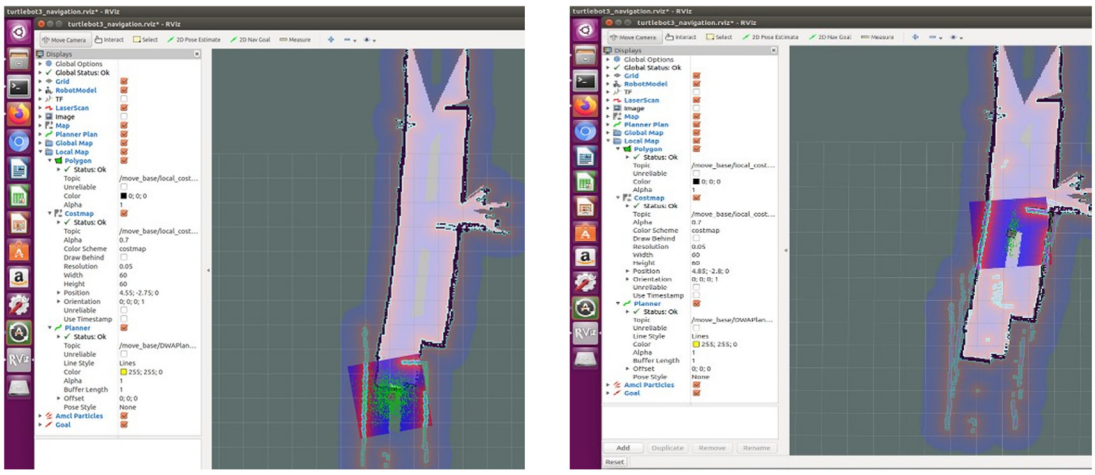

図 4-4　2D Pose Estimation で地図を合わせる画像

次に、Let's note CF-SZ5 のターミナルでキーボード操作アプリケーションを終了し、rViz 上の「2D Nav Goal」ボタンをクリックし、ロボットが到達すべき位置と向きをマウスのドラッグにより設定する。

rViz 上でロボットの位置と目的地を表示したのが図 4-5 である。

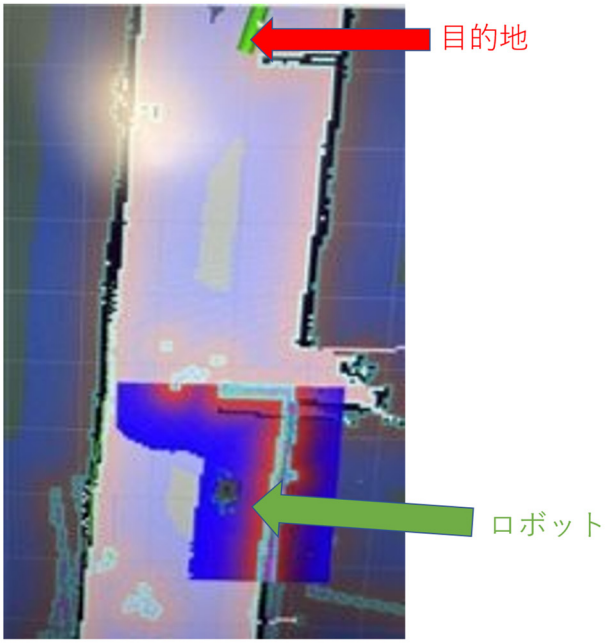

図 4-5　目的地が設定された様子

目的地が設定されると、最適な経路が計算され、図 4-6 のように rViz 上に表示される。それと同時にロボットは移動を開始し、図 4-7 のように rViz 上でもロボットの移動の様子が示される。

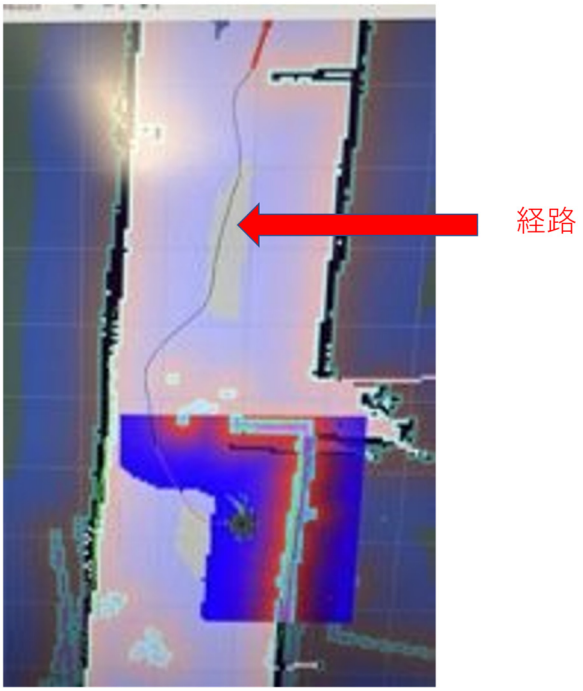

図 4-6　最適経路が決定された様子

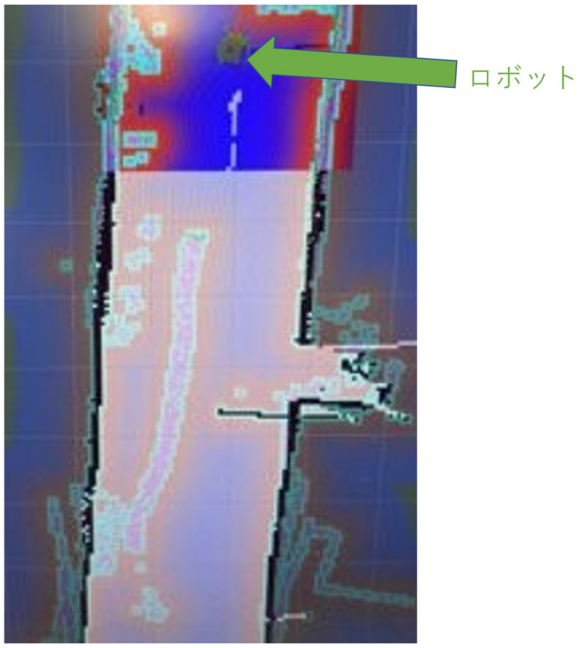

図 4-7　目的地に移動した様子

4.3　時間測定

4.3.1　概要

5 つの手法で作成した地図すべてにおいて、ナビゲーションで障害物を避けて目的地に到達することができた。本節では、地図作成をした際に存在しなかった、動く障害物を避けるのにかかる時間を調べる。

4.3.2　測定方法

測定の手順は以下の通りである。

1　距離測定をした際に作成した地図でナビゲーションをおこなうものとする。

2　10m 地点を目的地とし、5m 地点で人が移動型ロボットの経路上に歩き出す。

3　10m 地点に到着するのにかかった時間を調べる。

4　5 つの地図作成用の手法とも上記の手順を各 3 回行い、それぞれ平均を出し比較する

4.3.3　測定結果

測定結果を以下の表とグラフに示す。

測定結果から、Roomba の gmapping で作成した地図でのナビゲーションが平均 40.61 秒と最も到着時間が早いということが分かる。

なお、TurtleBot3 の cartographer で作成した地図でのナビゲーションでは、障害物を認識することが出来たが、新しい経路を見つけることが出来ず、目的地から遠ざかるという問題が起こったため、データは存在しない。

表 4-1　時間測定の結果のデータ

手法	測定結果（S）			
	1 回目	2 回目	3 回目	平均
TurtleBot3(cartograher)	×	×	×	×
TurtleBot3(frontier_exploration)	75.13	117.16	53.29	81.86
TurtleBot3(karto)	65.58	53.82	×	59.7
TurtleBot3(hector)	65.04	49.93	53.3	56.09
TurtleBot3(gmapping)	45.1	46.61	41.82	44.51
Roomba(karto)	44.39	47.62	43.56	45.19
Roomba(hector)	50.16	45.68	44.83	46.89
Roomba(gmapping)	39.32	44.21	38.3	40.61

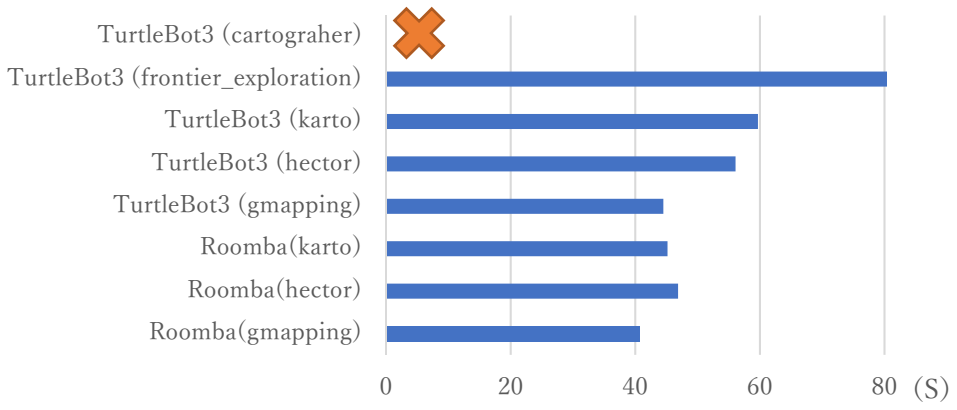

図 4-8　時間測定の平均を比べたグラフ

4.4　侵入可能な幅の調査

4.4.1　概要

本節では、移動ロボットが通ることが可能である幅について調べる。移動ロボットの横幅は以下の図 4-9 の通り、Roomba と TurtleBot3 とでそれぞれ 34cm および 28.1cm であった。

図 4-9　移動ロボットの横幅の画像

実験は、図 4-10 のように 2 枚のデスクパーティションの間をナビゲーションで移動ロボットが通ることができるかを調べることによって行う。

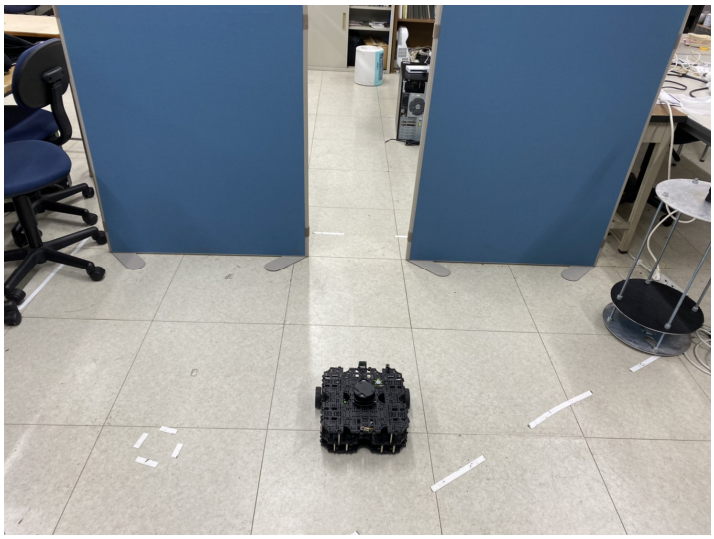

図 4-10　調査環境

その実験のため事前に作成した地図が図 4-11 である。

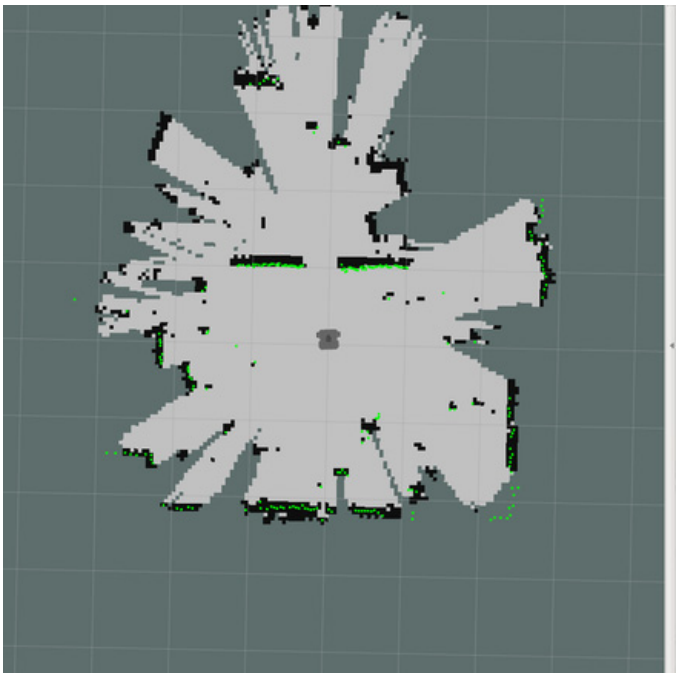

図 4-11　事前に作成した地図

4.4.2　測定方法

実験手順は以下の通りである。

1. ２枚のデスクパーティションをあらかじめ設定した間隔で設置する。
2. Let’s note CF-SZ5 のターミナル上でのキーボード操作で移動ロボットを動かし、デスクパーティションの間を通ることが可能であるかを確認する。
3. 実験環境の地図（図 4-11）を作成する。
4. 作成した地図を用いてナビゲーションを行う。目的地はデスクパーティションの間とする。
5. 5 つの地図作成用の手法で上記の手順を各 3 回行い、3 回とも通れたら侵入可能な幅であるとする。

4.4.3　実験結果

実験結果を表 4-2 に示す。

表 4-2　侵入可能な幅の調査結果

手法	測定結果(cm)		
	45	44	43
TurtleBot3(cartograher)	○	×	×
TurtleBot3(frontier_exploration)	○	×	×
TurtleBot3(karto)	○	×	×
TurtleBot3(hector)	○	×	×
TurtleBot3(gmapping)	○	×	×
Roomba(karto)	○	×	×
Roomba(hector)	○	○	○
Roomba(gmapping)	○	×	×

この表から、Roomba の hector で地図を作成した場合のみ幅 43cm から通ることが可能であり、幅 45cm から全ての方法で通ることが可能であるということがわかる。

4.5　ナビゲーションの考察

表 4-1 の時間測定の結果のデータから、最も到着時間が早かったのは、Roomba の gmapping で地図を作成した場合であり、次に到着時間が早かったのは TurtleBot3 の gmapping で地図を作成した場合であるという結果がわかる。この結果から、人が多い環境でのナビゲーションには gmapping で作成した地図が適していると考えられる。

また、侵入可能な幅の調査では、TurtleBot3 より Roomba の方がサイズが大きいにも関わらず、Roomba の hector で地図を作成した場合のみ、43cm から侵入が可能であった。これは Roomba と TurtleBot3 とで使われているソフトウェアにおいて、障害物をどれだけ避けるべきかを決めるパラメータが異なっていることが原因と考えられる。

例えば、occdist_scale というパラメータは、障害物をどの程度避けるべきかを決めるものであり、その値を大きくすると、移動ロボットは障害物を大きく避けるようになる。

このように障害回避の性能を変えるパラメータを変更して実験し、結果の比較を行うべきであったと考えられる。

第 5 章　結論（君塚担当）

5.1　概要

第 3 章と第 4 章の測定結果から、Roomba では gmapping を用いるのが適しており、TurtleBot3 では hector を用いるのが適していると我々は考えた。この結果から、本節では、この 2 つの測定結果を比較する。

5.2　測定結果の比較

5.2.1　距離測定の誤差の比較

距離測定の誤差の比較結果を以下の表とグラフに示す。距離が長い方が誤差が小さくなるという TurtleBot3 の結果は、より詳細な実験による検証が必要であると思われる。

表 5-1　距離測定の誤差のデータ（平均）

手法	誤差結果(cm)	
	5m（青）	10m（オレンジ）
Roomba(gmapping)	0.474	21.59
TurtleBot3(hector)	5.519	0.35

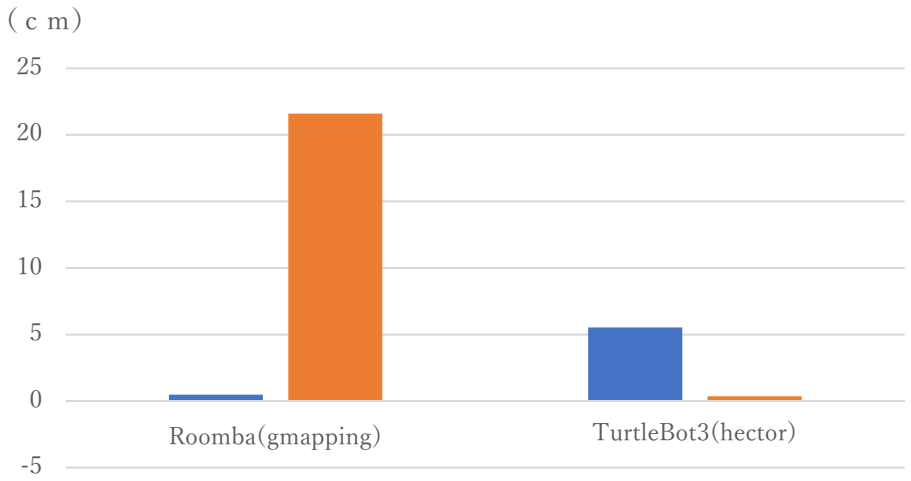

図 5-1　距離測定の誤差の比較グラフ（平均）

5.2.2　障害物の多い環境における距離測定の誤差の比較

障害物の多い環境における距離測定の誤差の比較結果を以下の表とグラフに示す。

表 5-2　障害物の多い環境における距離測定のデータ（平均）

手法	誤差結果(cm)		
	冷蔵庫（青）	デスクパーティション（オレンジ）	机（緑）
Roomba(gmapping)	6.149	12.11	2.26
TurtleBot3(hector)	2.4791	4.6473	7.735

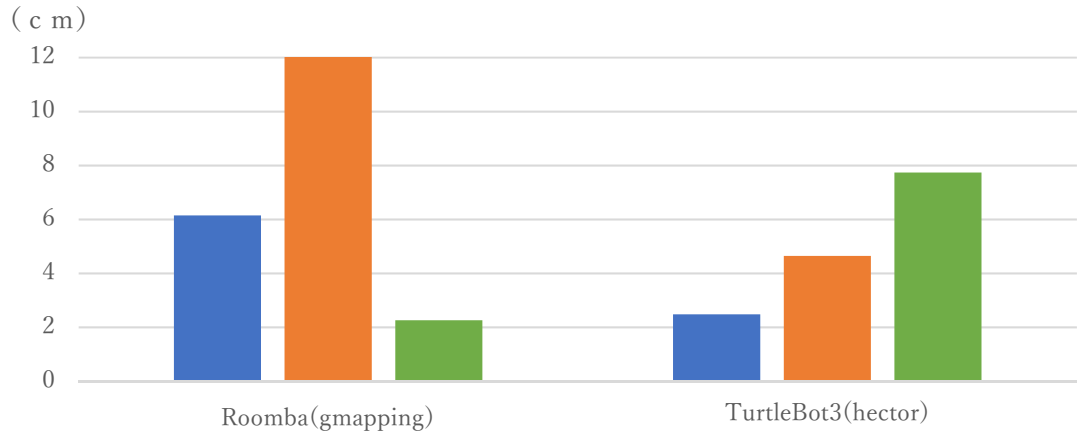

図 5-2　障害物の多い環境における距離測定の誤差の比較グラフ（平均）

5.2.3　認識可能な高さの比較

認識可能な高さの比較結果を以下の表に示す。

表 5-3　認識可能な高さの比較

手法	高さ(cm)	
	16	30
Roomba(gmapping)	○	○
TurtleBot3(hector)	×	○

5.2.4　時間測定の比較

目的地到達までにかかる時間の比較結果を以下の表に示す。

表 5-4　目的地到達までにかかる時間の比較（平均）

手法	平均（s）
Roomba(gmapping)	40.61
TurtleBot3(hector)	56.09

5.2.5　侵入可能な幅の比較

侵入可能な幅の比較結果を以下の表に示す。

表 5-5　侵入可能な幅の比較

手法	幅（cm）
Roomba(gmapping)	45
TurtleBot3(hector)	45

5.3　比較と考察

5.2.1 項と 5.2.2 項の結果から、TurtleBot3 の hector は、Roomba の gmapping と比べて、地図作成の精度が高いと考えられる。

また 5.2.4 項の結果より、Roomba はナビゲーションで目的地に到達するまでにかかる時間が TurtleBot3 より速いという結果が得られた。ただし、この実験は地図作成の性能に依存するというよりは、2 種類のロボットの特性の違いや、ナビゲーションのパラメータの設定の違いが大きいと考えられる。これらを変更して比較するなど、より詳細な実験を行うべきであった。

5.4　問題点とその解決方法

今回の研究で得た問題点は以下の通りである。

1.　一定の高さ以下の物体を認識出来ない。

Roomba では高さ 16cm、TurtleBot3 では高さ 30cm より低い障害物を認識することができない。また、これは実験結果には含まれていないが、机の脚は認識できるものの机の天板は認識できない、ということも当然起こる。これらは障害物の認識に、レーザーの直進性を利用する LiDAR を用いる以上やむをえないことである。

この問題を解決するには、ステレオカメラなど LiDAR 以外のセンサを用いる方法が考えられる。

2.　動く障害物がある際のナビゲーションにスムーズさがない。

ナビゲーションは Roomba と TurtleBot3 ともにおおむねうまくいったが、障害物が多い家庭内で荷物を運べるほどの実用的なスムーズさはなかった。

ROS のナビゲーションのパラメータを変更することで、この問題がどの程度改善されるか、より詳細な実験をすべきであった。

第 6 章　参考文献・URL

[1] 新型コロナウィルス感染者数の推移
https://www.asahi.com/special/corona/

[2] ロボティクス・自律型マシンの市場規模 2026 年には 2860 億米ドルに到達予測
https://www.value-press.com/pressrelease/256222

[3] 工学院大学卒業論文「ステレオカメラによる距離計測を用いた移動ロボットの製作」(2019 年 3 月)

[4] アイサイトのしくみ
https://www.subaru.jp/eyesightowner/about/

[5] 360 Laser Distance Sensor LDS-01
https://e-shop.robotis.co.jp/product.php?id=11

[6] TurtleBot3
https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/#notices

[7] Raspberry Pi 3 Model B
https://www.switch-science.com/catalog/3050/

[8] RPLidar A1M8 - 360 度レーザスキャナ開発キット
https://www.robotshop.com/jp/ja/rplidar-a1m8-360-degree-laser-scanner-development-kit.html

[9] LIPO 11.1V 1800mAh LB-012
https://www.rt-shop.jp/index.php?main_page=product_info&products_id=2595

[10] Roomblock: a Platform for Learning ROS Navigation With Roomba, Raspberry Pi and

RPLIDAR
https://www.instructables.com/Roomblock-a-Platform-for-Learning-ROS-Navigation-W/

[11] TurtleBot3: 3. Quick Start Guide
https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/

謝辞

最後に、この場をお借りして本研究を進めるにあたり、2 年間多大なご指導を頂きました金丸隆志教授、研究において親身になり多くの助言を頂いた先輩方、施設の利用において快く対応をして頂いた 4 号館の皆様方、協力して頂いた皆様に心より感謝いたします。これをもって謝辞とかえさせて頂きます。

移動ロボットの地図作成とナビゲーションに関する研究

指導教員　金丸 隆志　教授

S5-17020　君塚 瑛　　S5-17051　宮前 直樹　　S5-17055　山下 創

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、感染拡大を防ぐために密閉・密集・密接（三密）を避けた行動が求められるなど、人々の生活が大きく制限されるようになった。その影響で、今までは人同士でコミュニケーションをとって行なっていた公共施設の案内や工場内の荷物運搬などの作業を自律移動ロボットに任せる動きが今後さらに強まることが考えられる。

そこで我々は、自律移動ロボットの関連技術である地図作成とナビゲーションに関する研究を行なった。

2. 原理

自律移動ロボットには部屋の構造・物体の位置を把握する地図作成機能と壁や障害物などを避けて指示された目的地に進むナビゲーション機能が必要だと我々は考えた。地図作成は自己位置推定によって得た情報をもとに行うことができ、ナビゲーションは作成した地図情報と走行時のセンサからの情報をもとに目的地までの経路を計画することで実現できる。なお、自己位置推定と地図作成を同時に行う技術のことを SLAM（Simultaneous Localization and Mapping）という。

3. 地図作成

本研究では、Roomba と TurtleBot3 の２つのロボットを用い、ROS（Robot Operating System）が提供する gmapping・hector・karto・frontier_exploration・cartographer の５つの地図作成の手法の測定精度の違いを調べる。

作成した地図上で計測した距離と実際の距離の誤差を測定することにより、測定精度を比較する。

地図の作成は、コントロール用 PC のキーボードでロボットを操作することで行う。図 1（左）が地図を作成した環境であり、作成された地図が図 1（右）である。

地図上で対象物の距離を測定した結果、開けた環境だけではなく、障害物が多い環境でも十分な精度が得られた。その中でも、Roomba では gmapping で作成した地図、TurtleBot3 では hector で作成した地図の精度が良い傾向があることもわかった。

また、ロボットの速度を速くすると、測定精度が低下する傾向があることも分かった。

ただし、遠い対象物に対する計測精度が良いなど直観に反する結果もあったため、試行回数を増やすなど、より詳細な実験が必要であることも明らかになった。

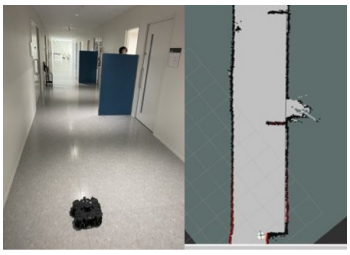

図 1　測定環境(左)と作成した地図(右)

4. ナビゲーション

移動ロボットで作成した地図をもとにナビゲーションを行った。ナビゲーション時のコントロール用 PC の画面を図 2 に示す。地図通りのナビゲーションは問題なく実現できたので、地図作成時には存在しなかった動く障害物を追加して実験を行った。ナビゲーションはおおむね成功したが、動作のスムーズさに欠ける、目的地に到達できない、などの問題も起こった。ナビゲーションのパラメータを変更して改善するか調べる必要がある。

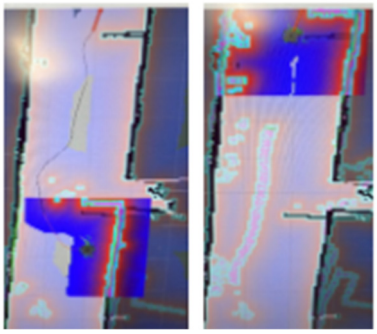

図 2　ナビゲーション

（左図：移動前、右図：移動後）

5. 結言

Roomba と TurtleBot3 の２つのロボットを用いて、gmapping・hector・karto・frontier_exploration・cartographer の５つの地図作成の手法の測定精度を調べることが出来た。また、作成した地図をもとにナビゲーションを行うことが出来た。

パラメータの変更による精度の変化を調べるなど、ROS の仕組みを理解した上で実験を行えば、より詳細な結果が得られるであろう。