2024 年度（令和 6 年度）
創造工学セミナーII Final Report

人の存在を考慮した頑強な
自己位置推定に関する研究

研究メンバー
S521009　大島陸駆
S521049　波山正陽
S521058　松山夢希
S521063　山下祥太

指導教員
金丸隆志　教授

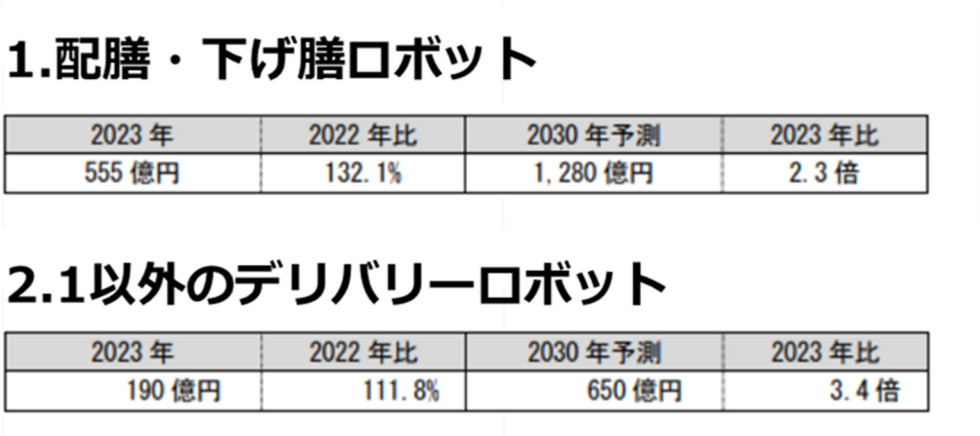
所属研究室
知能機械研究室

第 1 章　序論（波山）

1.1　研究背景

昨今、世界市場において自律移動ロボットの需要が増加している（図 1）[1]。例えば、身近な自律移動ロボットの運用法として、飲食店における配膳や工場・倉庫等における物資の運搬が挙げられる。これらの事例に関連し、前川氏の“工場内搬送および物流倉庫向け自律移動ロボットの開発”という研究がある[2]。この研究は、環境地図作成・自己位置推定・障害物回避機能を持たせた自律移動ロボット（図 2）に、工場もしくは倉庫で物資を運搬させることを目的とした研究である。前川氏は論文中にて、有軌道型・ガイド式無軌道型の自律移動ロボットは、レールの接地や床面にテープなどを張る必要があるのに対し、ガイドレス式無軌道型と呼ばれるものは、環境への加工が不要であるため、より柔軟であると述べている（図 3）。したがって、ガイドレス式無軌道型のようにガイド無しで自己位置推定ができる自律移動ロボットは、今後さらなる需要の増加が予想される。

そこで、本研究では移動ロボットの自己位置推定に焦点を当てる。



1.配膳・下げ膳ロボット

2023 年	2022 年比	2030 年予測	2023 年比
555 億円	132.1%	1,280 億円	2.3 倍

2.1以外のデリバリーロボット

2023 年	2022 年比	2030 年予測	2023 年比
190 億円	111.8%	650 億円	3.4 倍

図 1　自律移動ロボットの需要[1]

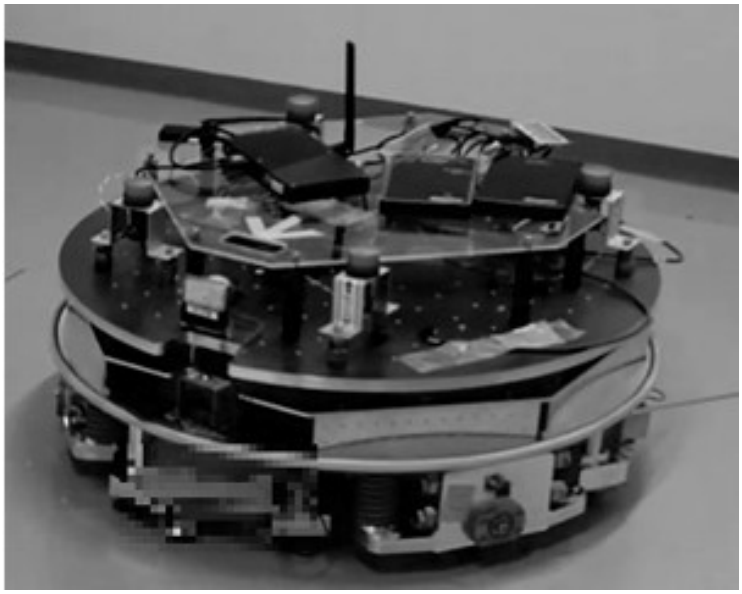


図 2　自律移動ロボットの例[2]

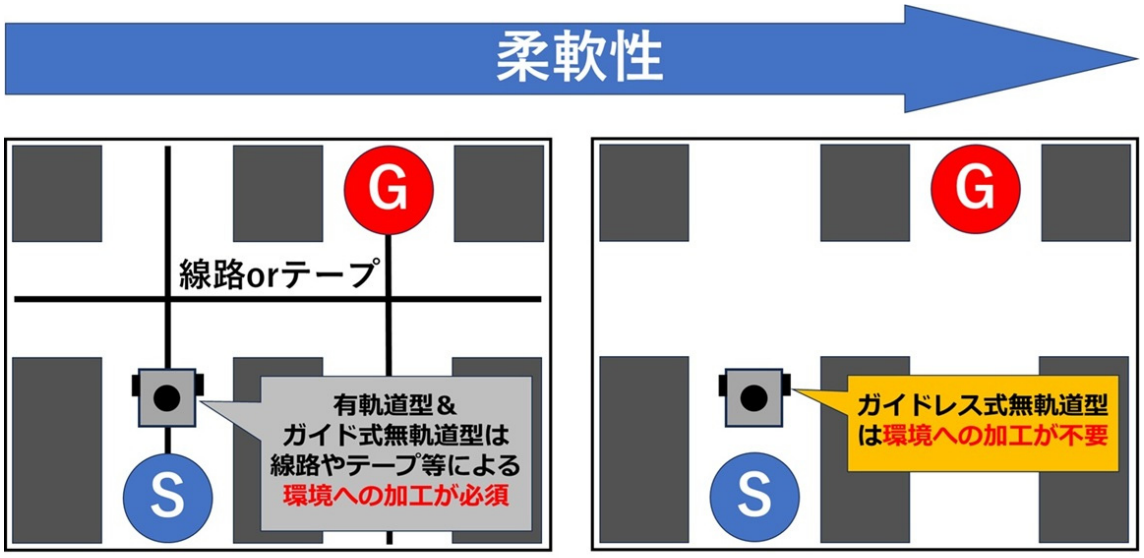


図 3　有軌道型・ガイド式無軌道型とガイドレス式無軌道型の比較

1.2　移動ロボットにおける自己位置推定の手法

移動ロボットにおける自己位置推定においてしばしば用いられる手法として、SLAM（Simultaneous Localization and Mapping）がある。SLAM とは、環境地図作成と自己位置推定を同時に行う技術である。SLAM で代表的に用いられるのが、LiDAR SLAM と呼ばれる手法である（図 4）。LiDAR SLAM とは、LiDAR（Light Detection And Ranging）と呼ばれるレーザーセンサから得られる距離情報と移動ロボットが環境内を移動する際に得られる移動量を元に環境地図を作成し、それと同時に、距離情報と作成した環境地図を照らし合わせ、自己位置推定を行うものである。LiDAR SLAM による自己位置推定の詳細については第 5 章 2 節 1 項で再度解説する。

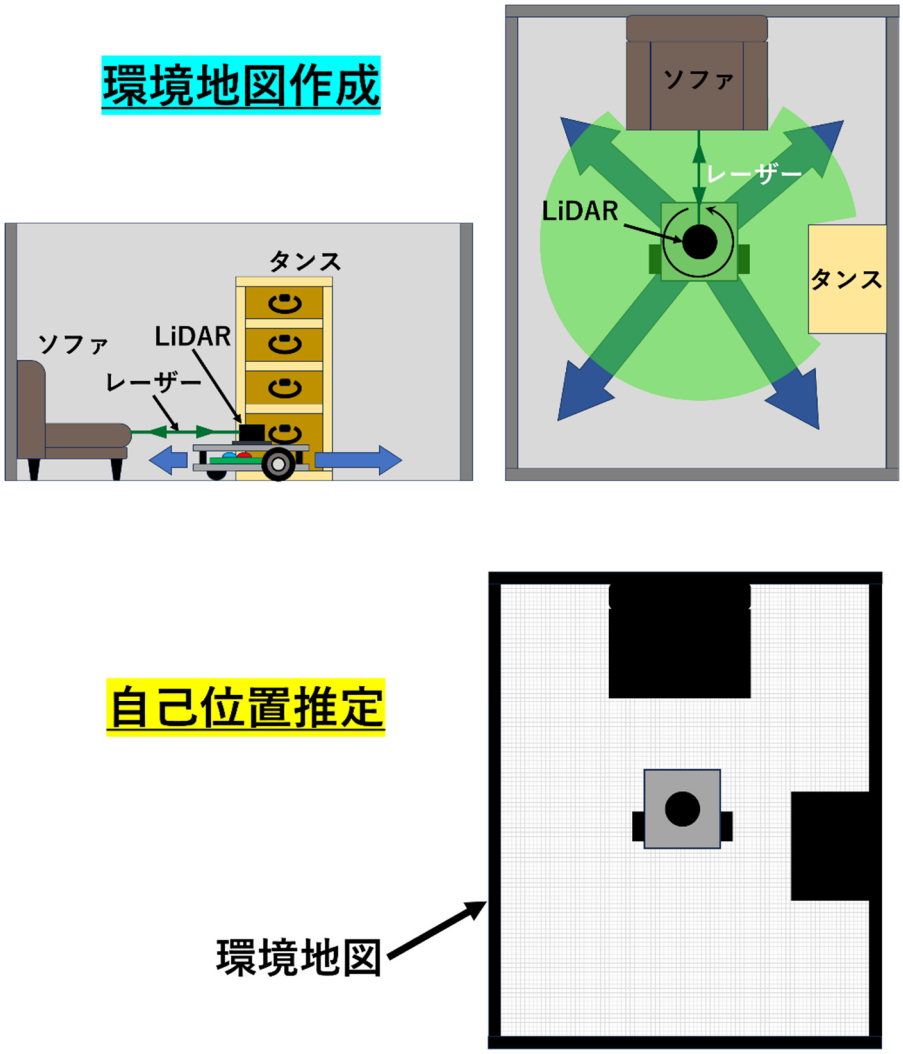


図 4　LiDAR SLAM の概要図

1.3　先行研究

LiDAR SLAM による自己位置推定の欠点に関する先行研究として、萬氏の“物体の属性を考慮した環境変化に対して頑強な自己位置推定”という研究が挙げられる[3]。この研究は、喫茶店での移動ロボットによるサービスの提供を想定して行われたものである。

図 5 は、萬氏の研究概要を図示したものである。まず、移動ロボットを実験環境で移動させ、環境地図を作成する（図 5①）。その後、実験環境に人や椅子といった元々実験環境に無かった障害物を追加する（図 5②）。すると、既に作成しておいた環境地図上に無い障害物が出現したことにより、移動ロボットの自己位置推定の精度が低下してしまう。

この問題に対し、萬氏は人・椅子・荷物などの障害物に移動属性を定義した。移動属性とは、障害物の種類に応じた移動のしやすさを定量化したものである。そして、萬氏は定義した移動属性を LiDAR SLAM における自己位置推定に取り込むことで、頑強な自己位置推定の手法を提案している。

図 5　物体の属性を考慮した環境変化に対して頑強な自己位置推定[3]

第 2 章　研究概要（波山）

2.1　本研究で解決を目指す問題

我々が研究で解決しようと考えたのは、第 1 章 3 節で紹介した、環境地図作成後の障害物追加により、自己位置推定の精度が低下してしまう問題である。

図 6 は従来の LiDAR SLAM において自己位置推定の精度が低下する状況の例を表した図である。この状況を以後本研究で取り扱うものとする。まず、あらかじめ廊下で移動ロボットを往復させ、環境地図を作成する（図 6①）。次に、元々廊下には存在しなかった人の壁（人を横一列に並べた障害物）を廊下に配置する（図 6②）。その後、移動ロボットが自己位置推定を行うと、人の壁を廊下の壁と誤認してしまう（図 6③）。それにより、ロボットが推定する自己位置推定は誤ったものとなってしまうのである。

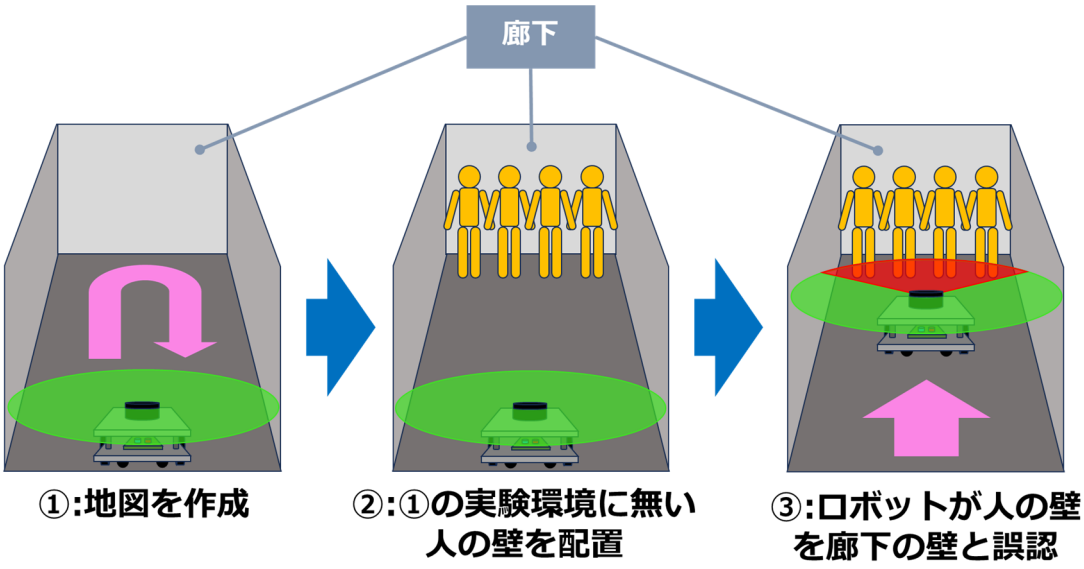


図 6　廊下における自己位置推定の問題点

2.2　提案手法

第 2 章 1 節で説明した従来の LiDAR SLAM における自己位置推定の問題点を改善する手法を以下で解説する。我々が提案するのは、環境地図を作成した時点で存在しなかった人の壁を、LiDAR による計測領域（以下、領域と呼称）から除外するシステムである。図 7 は従来の LiDAR SLAM と我々が提案する手法を比較したものである。まず、移動ロボットを廊下で往復させ、あらかじめ環境地図を作成する（図 7①）。その後、廊下に人の壁を配置する（図 7②）。

すると、図 7③のように、従来の LiDAR SLAM の自己位置推定において、赤色で示した人が存在する領域の信頼性が低下する。それに伴い、移動ロボットの自己位置推定の精度も低下してしまう。

そこで、図 7③’に示す提案手法では、移動ロボットに搭載する単眼カメラの映像を用いて人物認識を行うことで人の壁を認識させ、人の壁の人数に合わせた一定角度の信頼性の低い領域を除外する。それより、移動ロボットの自己位置推定を頑強にすることを目標とする。システムの流れの詳細については第 6 章 1 節、人物認識の手法については第 6 章 2 節で解説する。

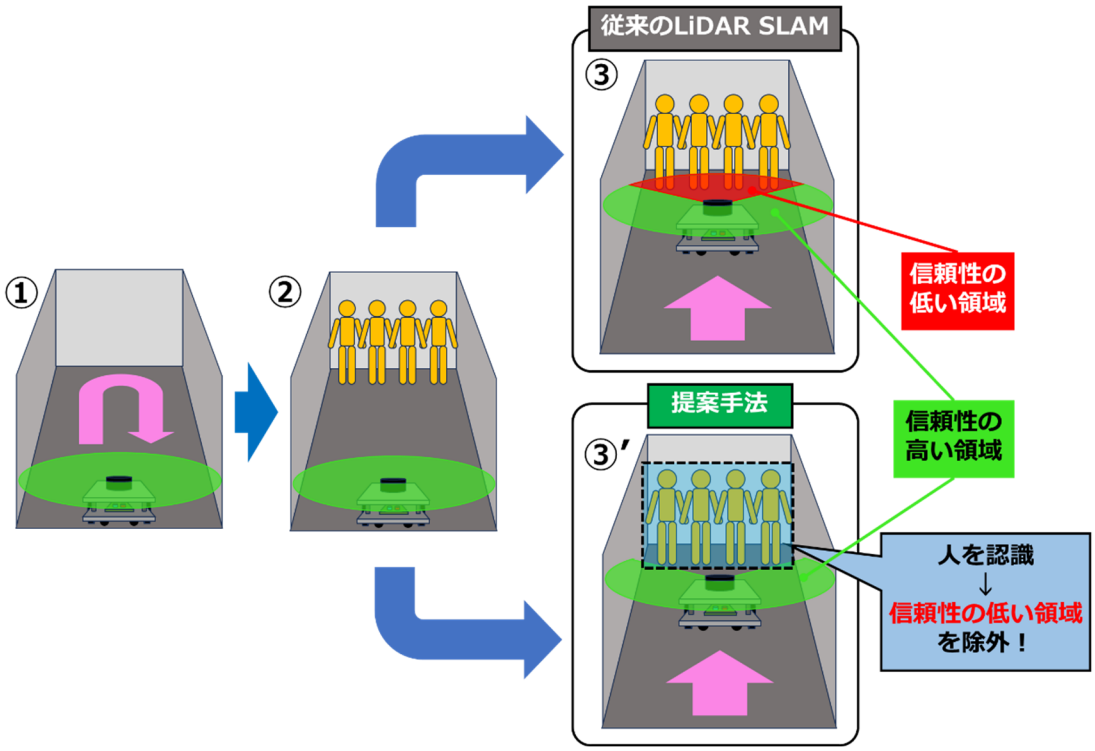


図 7　従来の LiDAR SLAM の自己位置推定と提案手法の比較

第 3 章　使用機材の概要（山下）

3.1　TurtleBot3 waffle pi

本研究では LiDAR SLAM を実行する為に TurtleBot3 waffle pi（図 8）（以下、TurtleBot3 と呼称）を採用した。TurtleBot3 には LiDAR が付属しており、このセンサを用いることで周囲の環境を計測できる。そして、計測した距離データを基に LiDAR SLAM を実行することができる。

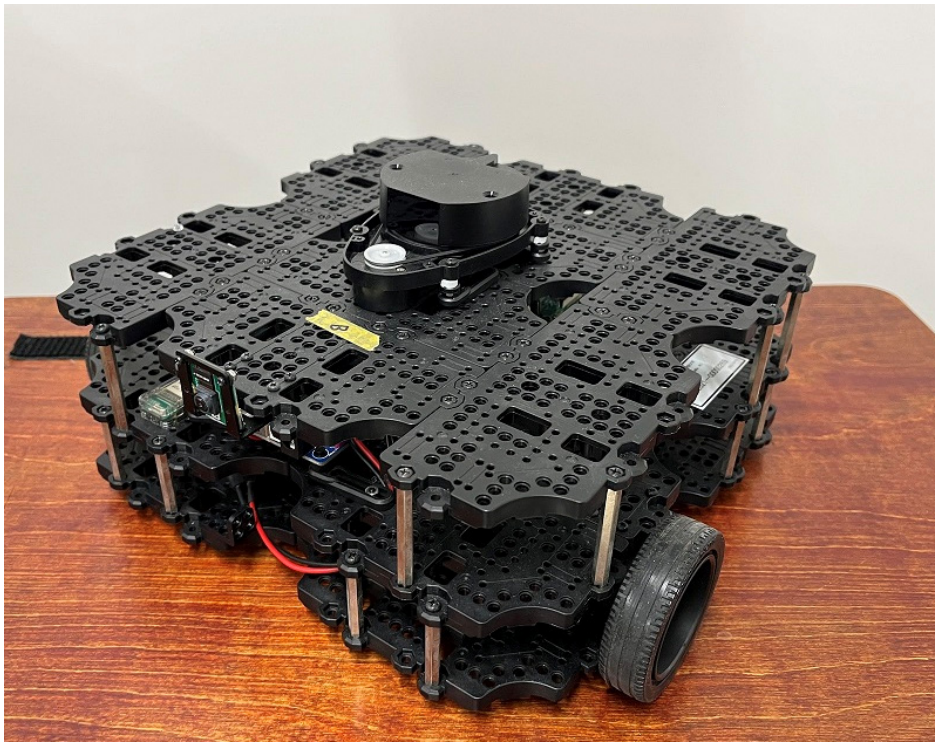


図 8　TurtleBot3 waffle pi

3.2　LDS-01

LDS-01（以下、LiDAR と呼称）は TurtleBot3 上部に設置されている LiDAR である（図 9)。LiDAR はレーザー光による距離測定を行える。そのデータを LiDAR SLAM に利用する。スペックは表 1 の通りである。

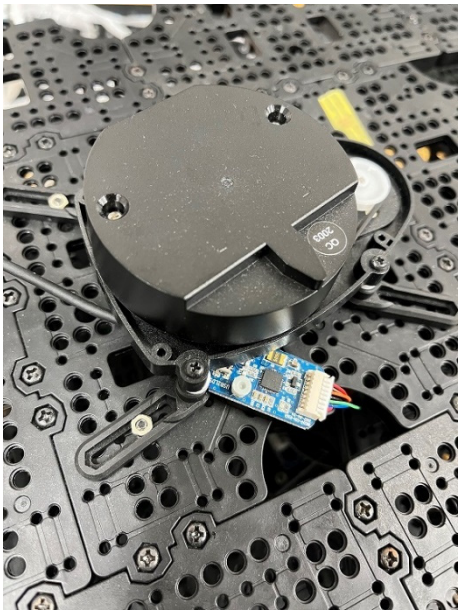


図 9　LDS-01

表 1　LDS-01 スペック

光源	半導体レーザーダイオード（λ=785nm）
検出距離	120mm ～ 3,500mm
角度範囲	360°
角度分解能	1°
サンプリングレート	1.8kHz
スキャンレート	300±10rpm

3.3　人物認識用 PC

本研究では、オープンソースで提供される ROS（Robot Operating System）というロボット向けのソフトウェアプラットフォームを用いる。さらに、同じ PC 上で人物認識用のプログラムを動作させる。ROS と人物認識用プログラムをインストールした人物認識用 PC を図 10 に示す。この PC には NVIDIA 製のグラフィックボードが搭載されている。また、Linux 系のオープンソース OS である Ubuntu20.04 をインストールして使用する。人物認識用 PC のスペックは表 2 の通りである。

図 10　人物認識用 PC

表 2　人物認識用 PC スペック

モデル名	NH55DC
OS	Ubuntu 20.04.6 LTS
CPU	Intel Core i7-10750H
GPU	NVIDIA GeForce GTX1660 Ti mobile
メモリ	16GB

3.4　操作用 PC

本研究では移動ロボットを遠隔で操作する為に操作用 PC（図 11）を使用する。操作用 PC にも Linux 系のオープンソース OS である Ubuntu20.04 をインストールして使用する。TurtleBot3 をキーボードで操作するにあたり、ROS が動作している人物認識用 PC に ssh コマンドを用いて接続した。操作用 PC のスペックは表 3 の通りである。

図 11　操作用 PC

表 3　操作用 PC スペック

モデル名	Let's note CF-SZ5ADYMS
OS	Ubuntu 20.04.6 LTS
CPU	Intel Core i5 6300U
メモリ	8GB

3.5　単眼カメラ

本研究では、人物認識を行う為の単眼カメラ（図 12）を人物認識用 PC に USB 接続する。用いる単眼カメラのスペックは表 4 の通りである。

図 12　単眼カメラ

表 4　単眼カメラスペック

モデル名	Logicool C920
最大解像度	1080p/30fps
カメラ画素数	3 メガピクセル

第 4 章　移動ロボットの作製（山下）

4.1　移動ロボット作製の目的

本研究の目的は「人の存在を考慮した頑強な自己位置推定の実現」である。この目的を達成するためには、移動ロボットが人物を認識し、その情報を自己位置推定に活用する必要がある。そして、その情報を取得する為にはカメラを使った人物認識が必要である。しかし、TurtleBot3 は人物認識のような高負荷な計算を必要とするプログラムに対して、計算リソースやメモリ容量が不足している為不向きであると我々は判断した。そのため、人物認識用 PC を TurtleBot3 に搭載する必要があると考えた。

4.2　各ハードウェアとソフトウェアの関係（波山）

図 13 は提案手法における各ハードウェアとソフトウェアの関係図である。まず、操作用 PC は、モーター等の制御と LiDAR SLAM を行う役割を担っている。次に、移動ロボットは、LiDAR を搭載した TurtleBot3 を改造し、人物認識用 PC、単眼カメラを追加で搭載した機体である。移動ロボットについては、第 4 章 4 節で詳細に解説する。

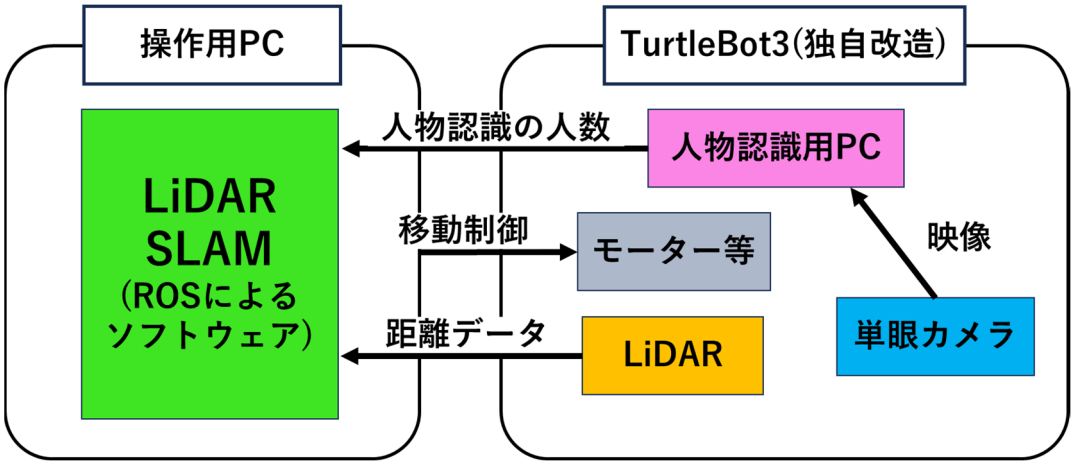


図 13　各ハードウェアとソフトウェアの関係

4.3　使用部品

ここでは、TurtleBot3 を改造するために必要なパーツについて解説する。

4.3.1　TurtleBot3 Waffle Plate-IPL-01

TurtleBot3 Waffle Plate-IPL-01（図 14）（以下プレートと呼称）は TurtleBot3 の本体を構成するパーツである。本研究では追加でプレートを使用することで TurtleBot3 上に新たにスペースを作製している。

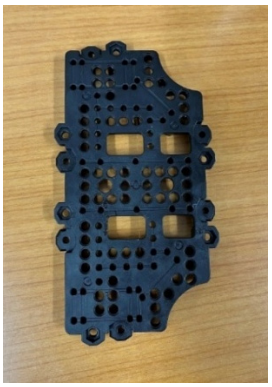


図 14　TurtleBot3 Waffle Plate-IPL-01

4.3.2　六角スペーサー（M3×45mm）

六角スペーサー（図 15）は TurtleBot3 のプレートを支える柱として採用した。

図 15　六角スペーサー（M3×45mm）

4.4　人物認識用 PC と単眼カメラの設置

人物認識用 PC と単眼カメラを TurtleBot3 に搭載するにあたり、新たに設置スペースを作製する必要がある。しかし、図 16 のように TurtleBot は三層構造であり、設置スペースが足りない。そこで三層目の上部にプレートと六角スペーサーを用いて新たに四層目を作製した（図 17)。三層目と四層目の間に人物認識用 PC と単眼カメラを設置し、三層目に設置されていた LiDAR は四層目の上部に移動させた。なぜなら、三層目に LiDAR を設置してしまうと、四層目を支えている六角スペーサーにレーザーが干渉してしまうためである。なお、この際、四層目の高さを高くしすぎると、TurtleBot3 の重心が高く横転してしまう可能性があった。そこで、重心を低く安定させる為に、四層目は人物認識用 PC を閉じた状態で入る高さに調整した。加えて、人物認識用 PC が閉じた状態で動作するように PC の設定を変更した。改造前後の比較を図 18 に示す。以上の改造のほか、三層目と四層目の間には、単眼カメラを取り付けている。

完成した移動ロボットは図 19 である。

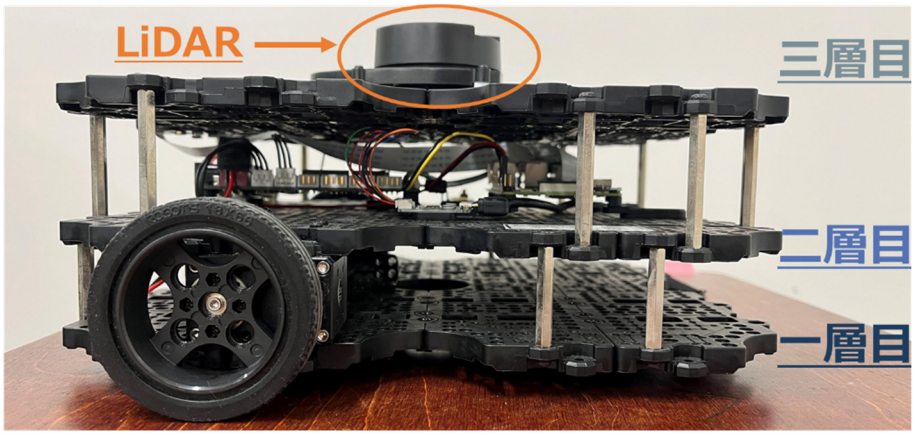


図 16　改造前

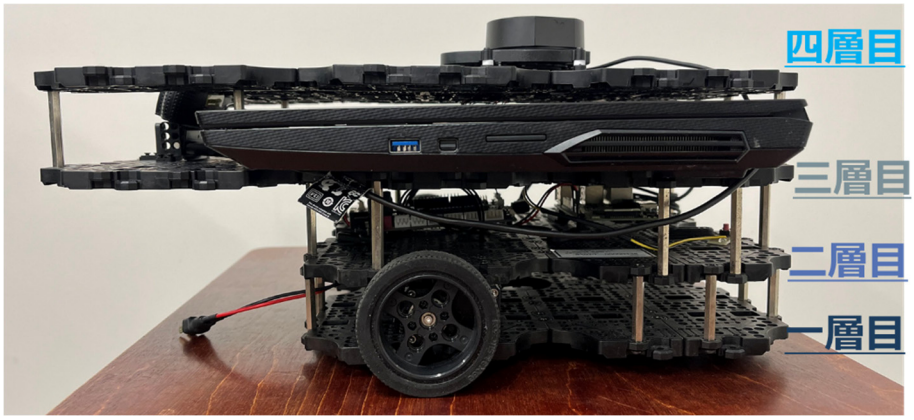


図 17　改造後

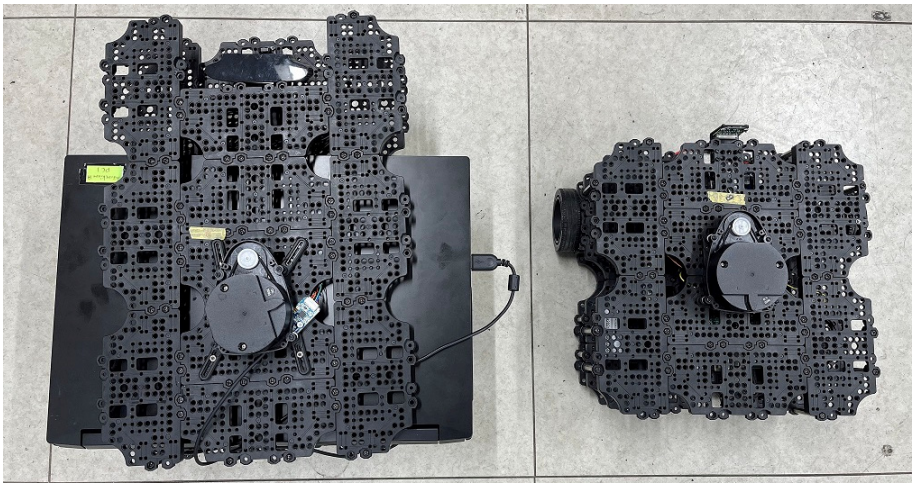


図 18　改造前後の比較（左:改造後　右:改造前）

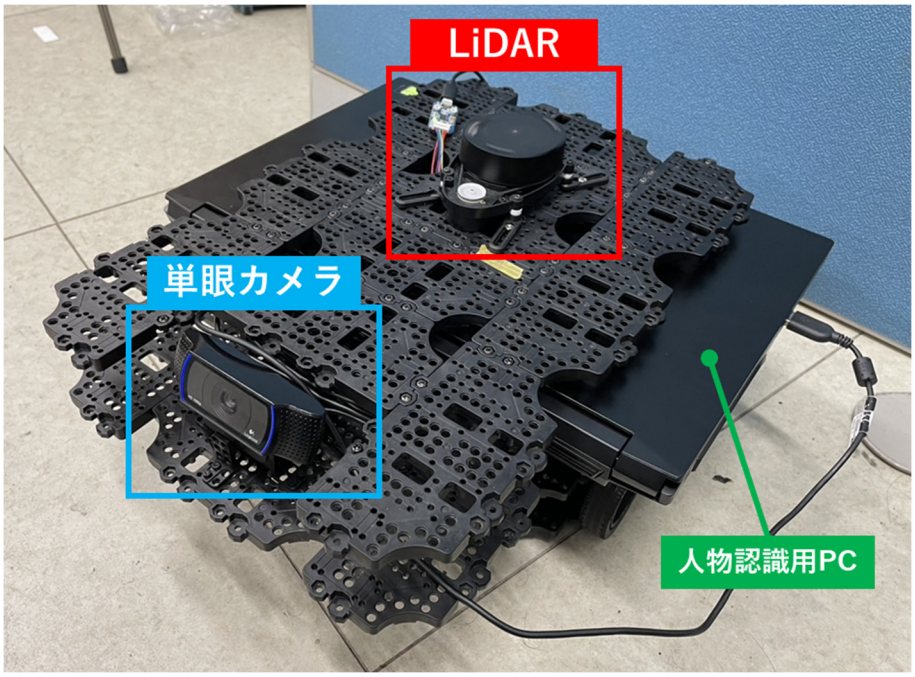


図 19　完成した移動ロボット

第 5 章　LiDAR SLAM とは（大島）

本章では、本研究で用いる LiDAR SLAM の仕組みと、自己位置推定で人を廊下の壁と誤認する理由について解説する。

5.1　環境地図作成

LiDAR SLAM の処理は環境地図作成と自己位置推定の 2 つに分けられる。

環境地図作成では、LiDAR で得られた距離データとタイヤの回転角度などを使用している。距離データは、レーザーが発射されて壁や障害物に届き、帰ってくるまでの時間を計測することで求められる。また、タイヤの回転角度から、タイヤの半径などあらかじめ分かっているハードウェアの情報を用いて、機体の移動距離と旋回角度を推定する手法をオドメトリという。

環境地図作成の流れとして、まず、距離データで得られた壁や障害物の情報を地図に書き加える。次に、オドメトリを用いて地図上の移動ロボットの位置を更新する。そして、更新された位置で得られた距離データを再度地図に書き加える。これらの処理を繰り返すことで環境地図が作成される。

5.2　自己位置推定

自己位置推定においては、まず、オドメトリを用いて地図上の移動ロボットのおおよその位置を更新する。そして、地図上における現在の位置周辺にある壁や障害物と距離データとの間でスキャンマッチングを行うことで、より正しいと推測される位置に更新していく。

5.2.1　スキャンマッチング

スキャンマッチングとは、環境地図上の壁や障害物と距離データを照らし合わせてより正しいと推測される位置に近づける手法である。図 20 を用いて説明する。図 20 中の緑の点は距離データをもとに描かれた障害物の位置、黒線は環境地図である。図 20 の左を見ると障害物と地図が大きくずれていることが分かる。対して図 20 の右を見ると障害物と地図が一致していることが分かる。この際、プログラム上では障害物と地図の一番近い壁との距離を計算し合計するという処理が行われており、合計値が一番小さくなる位置と方向に自己位置を修正している。この処理をスキャンマッチングという。しかし、地図上のすべての位置でスキャンマッチングを行ってしまうと膨大な時間がかかってしまうため、オドメトリを用いておおよそ自己位置が合っているという前提で、移動ロボットの周囲のみでスキャンマッチングを行っている。

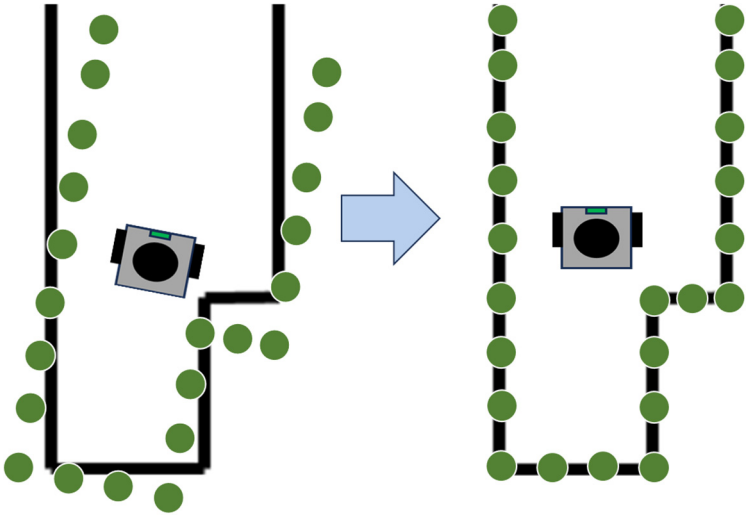


図 20　スキャンマッチング

5.2.2　人を廊下の壁と誤認してしまう場面

以上の解説を踏まえて、図 21 のような状況を考える。図 21（A）はコの字のような壁と、その前に人が数人立っている状況である。通常、環境地図作成は人がいない状態で行われるものであり、あらかじめ作られた環境地図には人がいないため、図 21（B）から図 21（C）のようにスキャンマッチングで修正されることが考えられる。しかし、正しい自己位置は修正前である、図 21（B）の状態である。その為、図 21（C）ではなく図 21（B）の状態が選択されるようアルゴリズムを変更する必要がある。そこで、我々は人に対応する障害物の距離データを除外しながら、LiDAR SLAM を行うことで解決を試みる。

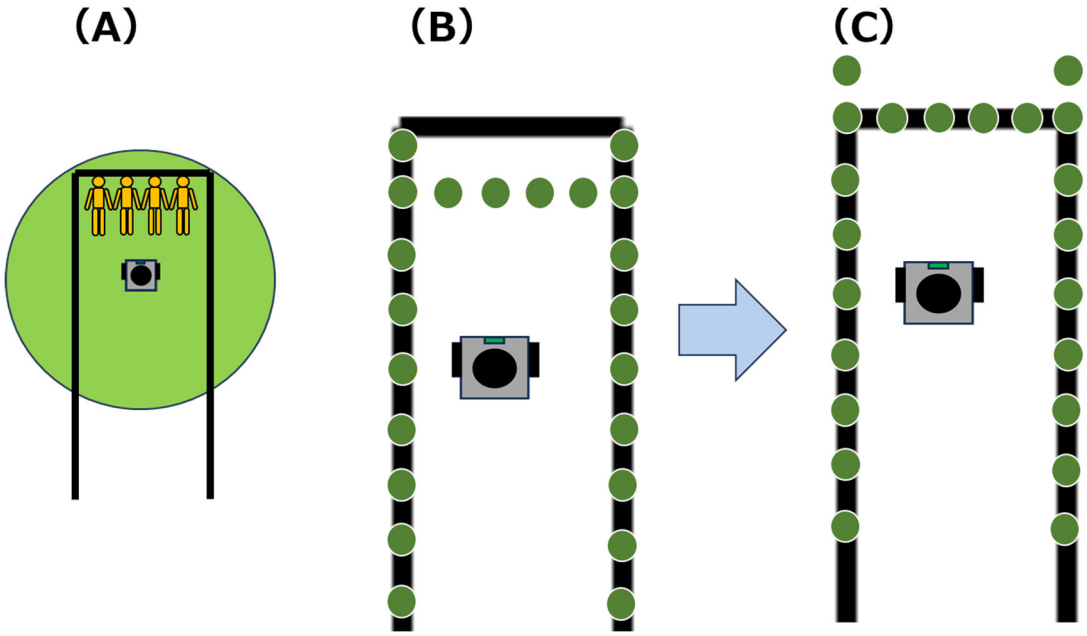


図 21　誤認するメカニズム

第 6 章　システムの流れ（大島）

6.1　提案手法のフローチャート

提案手法の処理の流れを図 22 のフローチャートで解説する。まず、通常の手段で環境地図を作成する（図 22（A））。この際、人は存在しないとし、環境地図が完成した後、人を配置する。次に、取り付けられたカメラの映像に写っている人数をカウントする（図 22（B））。そして、その人数分の距離データを除外し（図 22（C））、自己位置推定を行う（図 22（D））。これら人数のカウント、距離データの除外、自己位置推定はリアルタイムで動作している。本研究では、環境地図作成と自己位置推定に ROS が提供する GMapping という LiDAR SLAM 用のアルゴリズムを用いた。

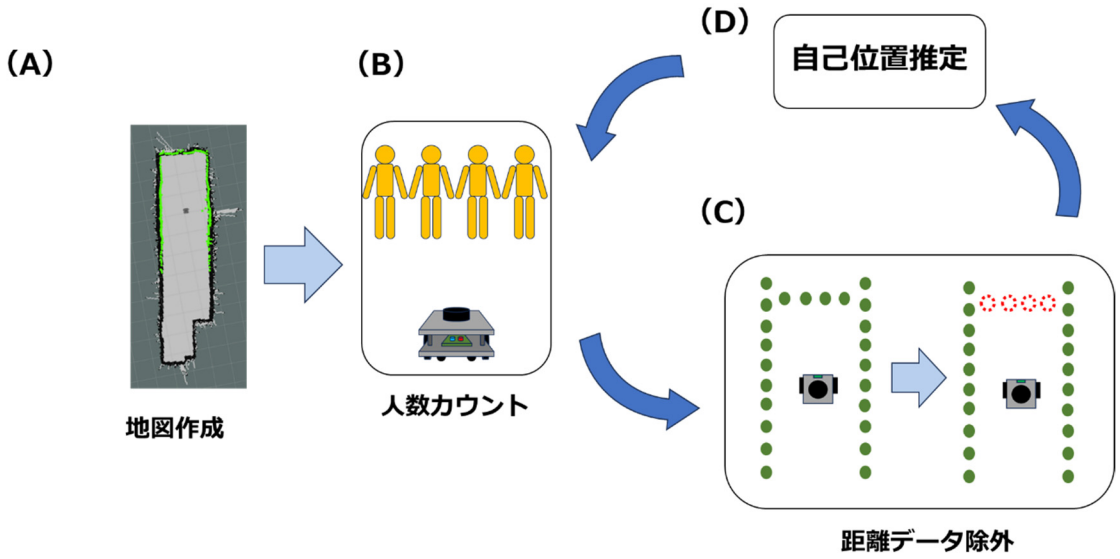


図 22　システムの流れ

6.2　人物認識用ライブラリの選定

先行研究である文献[3]では物体の属性を考慮した頑強な自己位置推定を行うため、物体認識ライブラリに YOLO を使用していた。一方、我々は人物認識のみが必要であることを前提に、使用するライブラリを選定する。人物認識には、代表的なライブラリとして MediaPipe[4]や OpenPose[5]がよく知られている。表 5 は 2 つのライブラリを比較したものである。MediaPipe は軽量なライブラリとなっており、処理速度は速いが多人数認識に対応していない。一方、OpenPose は非軽量であるが、多人数認識に対応している。 提案手法に必要なのはカメラに写っている人数をカウントするシステムであるため、OpenPose を使用することにした。OpenPose は MediaPipe と比較すると処理速度は遅いが、LiDAR は 1 秒間に 5 回 360 度分のデータが取れるのに対し、OpenPose のフレームレートが 10fps であるため、我々の目的には十分であると判断した。図 23 に OpenPose が動作している際の画面を示す。

表 5　人物認識ライブラリの比較表

	MediaPipe	OpenPose
処理速度	速	遅
多人数認識	×	○

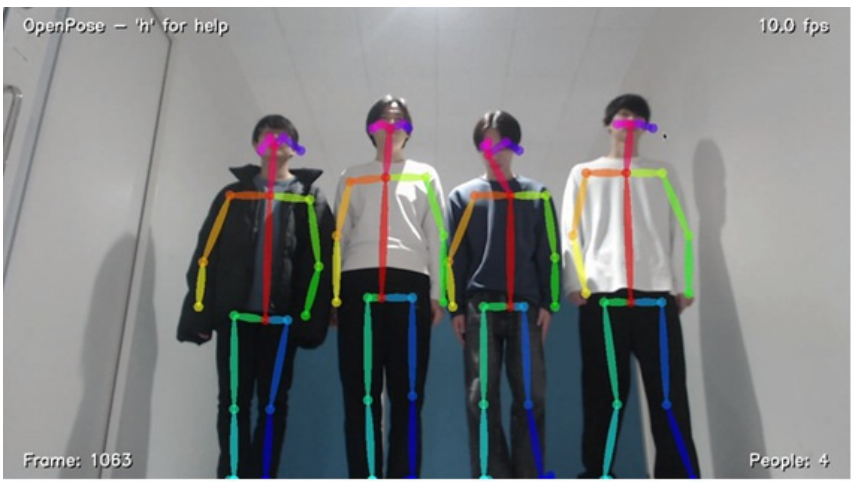


図 23　OpenPose の動作画面

6.3　人の数に合わせた LiDAR の領域の除外（波山）

6.3.1　人が存在する LiDAR の領域を除外するための角度の計算手法

本研究では、人の壁を認識した際に、人が存在する LiDAR の領域を除外する必要がある。そのため、除外する LiDAR の角度を計算する手法について説明する。

図 24 は、ロボットが人の壁を認識した場面を図示したものである。また、この実験環境の詳細については第 7 章 2 節にて解説する。廊下の幅・パーティションの壁の幅は 2.5m、人の壁とパーティションの壁の距離は 1m、LiDAR の最大検出距離（図 24 破線部分）は 3.5m であり、1 人分の人間の幅は 0.4m とする。また、提案手法を実現するためには、人の壁が存在する領域を除外しつつ、パーティションの壁の距離データは取得する必要がある。したがって、LiDAR の最大検出距離をパーティションの壁の両端に合わせることとした。その際の人の壁とロボットの距離は、以下の式より、約 2.27m である。

$$\sqrt{3.5^2 - 1.25^2} - 1 = 2.269\ldots \fallingdotseq 2.27[m]$$

この人の壁とロボットの距離を用いて、除外する LiDAR の領域の角度$\theta_n[rad]$（nは人の数）は式（1）で求めることが出来る。式（2）は、L_nは人の壁の幅を表している。

$$\theta_n = 2\tan^{-1}\left(\frac{L_n \times \frac{1}{2}}{2.269\ldots}\right) = 2\tan^{-1}\left(\frac{L_n}{2 \times (2.269\ldots)}\right) \tag{1}$$

$$L_n = 0.4n[m] \tag{2}$$

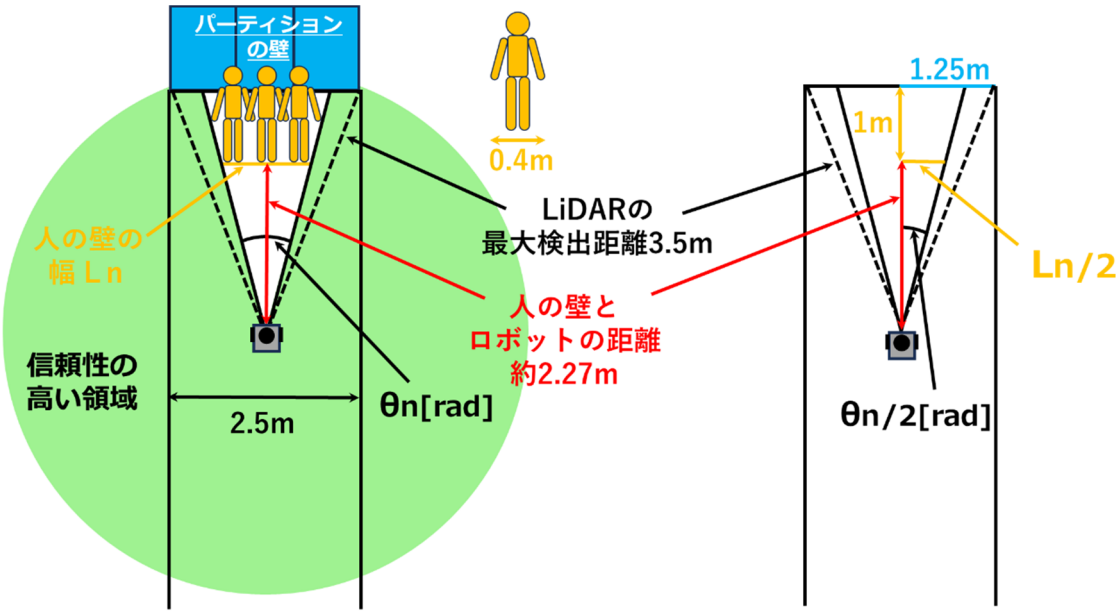


図 24　ロボットが人の壁（3 人）を認識した場面の例

人の壁が 4 人の場合（$n = 4$）を求める（図 25）。
人が 4 人（$n = 4$）の時のL_4は、式（2）より

$$L_4 = 0.4 \times 4$$
$$L_4 = 1.6[m]$$

である。
これより、θ_4は、式（1）より

$$\theta_4 = 2\tan^{-1}\left(\frac{L_4}{2 \times 2.269\ldots}\right)$$
$$\theta_4 = 2\tan^{-1}\left(\frac{1.6}{2 \times 2.269\ldots}\right)$$
$$\theta_4 = 0.677891515\ldots[rad]$$

である。また、プログラムへ反映させるときにはθ_4を二等分した値を用いる。用いた値は以下に示す。

$$\frac{\theta_4}{2} = 0.338945758\ldots[rad]$$

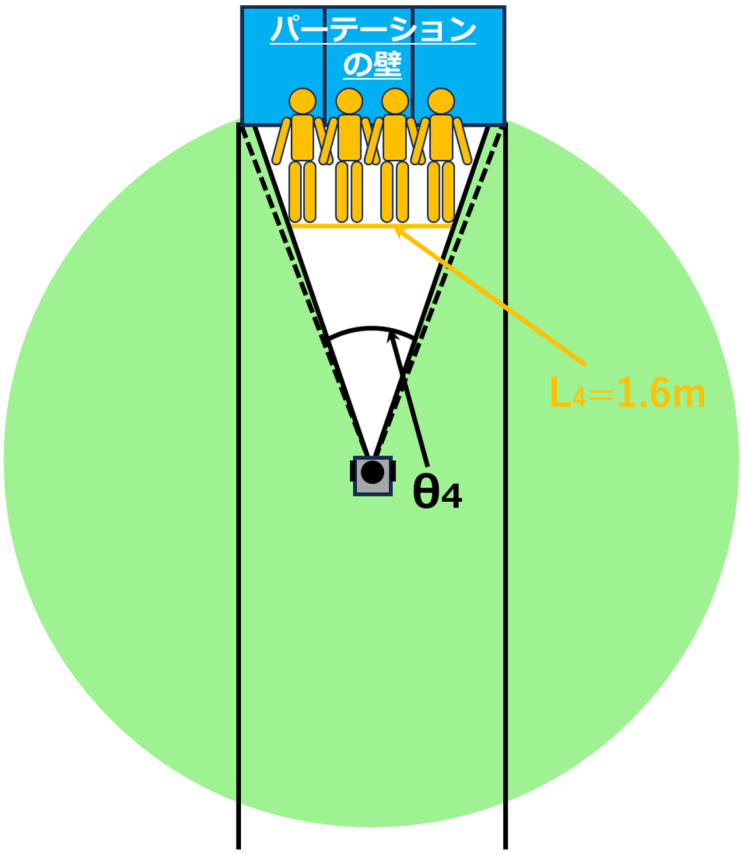


図 25　人の壁が 4 人の場合の角度 θ_4

人の壁が 3 人の場合（$n = 3$）を求める。（図 26）

人が 3 人（$n = 3$）の時のL_3は、式（2）より

$$L_3 = 0.4 \times 3$$
$$L_3 = 1.2[m]$$

である。

これより、θ_3は、式（1）より

$$\theta_3 = 2\tan^{-1}\left(\frac{L_3}{2 \times 2.269\ldots}\right)$$
$$\theta_3 = 2\tan^{-1}\left(\frac{1.2}{2 \times 2.269\ldots}\right)$$
$$\theta_3 = 0.516995028\ldots[rad]$$

である。また、プログラムへ反映させるときにはθ_3を二等分した値を用いる。用いた値は以下に示す。

$$\frac{\theta_3}{2} = 0.258497514 \ldots [rad]$$

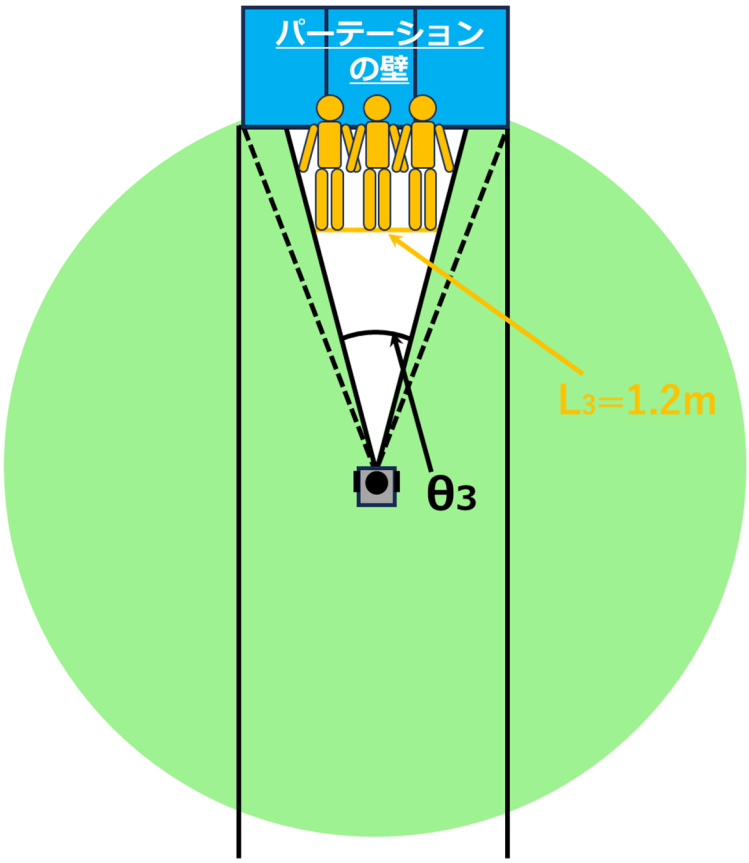


図 26　人の壁が 3 人の場合の角度 θ_3

人の壁が 2 人の場合（$n = 2$）を求める。（図 27）

人が 2 人（$n = 2$）の時のL_2は、式（2）より

$$L_2 = 0.4 \times 2$$
$$L_2 = 0.8[m]$$

である。

これより、θ_2は、式（1）より

$$\theta_2 = 2\tan^{-1}\left(\frac{L_2}{2 \times 2.269\ldots}\right)$$

$$\theta_2 = 2\tan^{-1}\left(\frac{0.8}{2 \times 2.269\ldots}\right)$$

$$\theta_2 = 0.348966148\ldots[rad]$$

である。また、プログラムへ反映させるときにはθ_2を二等分した値を用いる。用いた値は以下に示す。

$$\frac{\theta_2}{2} = 0.174483074\ldots[rad]$$

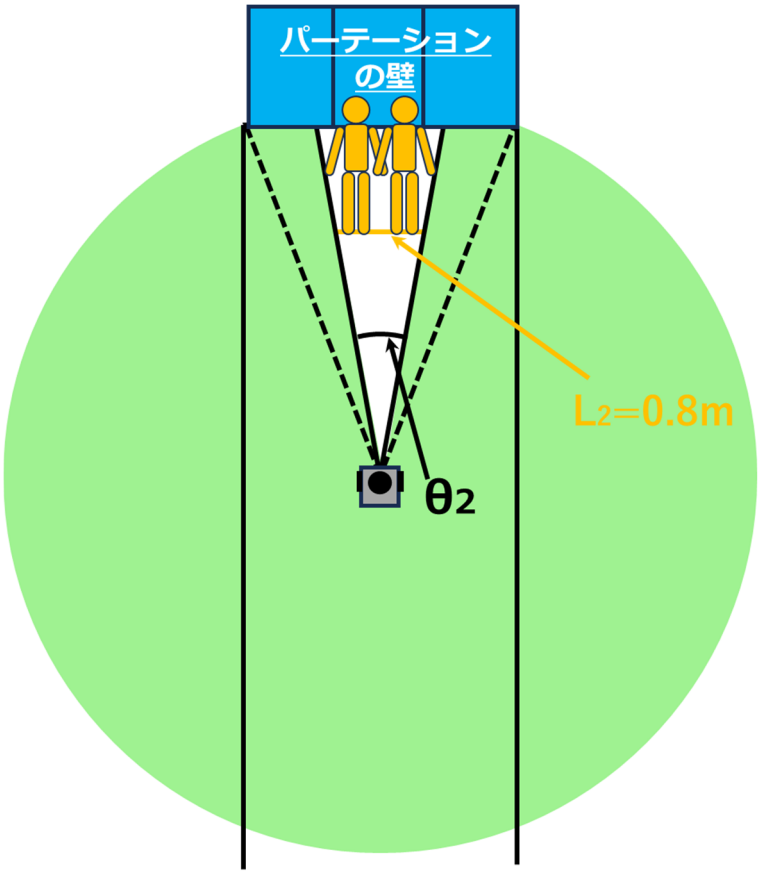


図 27　人の壁が 2 人の場合の角度 θ_2

人の壁が 1 人の場合（$n = 1$）を求める。（図 28）
人が 1 人（$n = 1$）の時のL_1は、式（2）より

$$L_1 = 0.4 \times 1$$
$$L_1 = 0.4[m]$$

である。

これより、θ_1は、式（1）より

$$\theta_1 = 2\tan^{-1}\left(\frac{L_1}{2 \times 2.269\ldots}\right)$$
$$\theta_1 = 2\tan^{-1}\left(\frac{0.4}{2 \times 2.269\ldots}\right)$$
$$\theta_1 = 0.175821244\ldots[rad]$$

である。また、プログラムへ反映させるときにはθ_1を二等分した値を用いる。用いた値は以下に示す。

$$\frac{\theta_1}{2} = 0.087910621\ldots[rad]$$

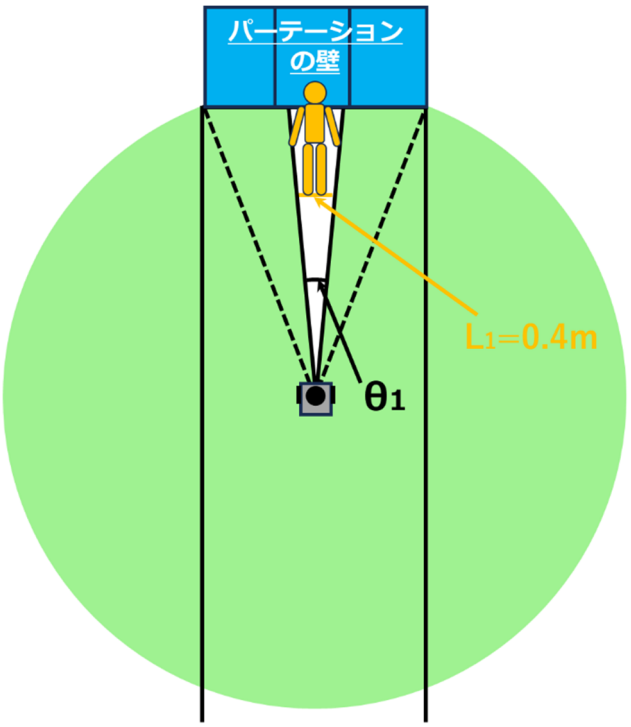


図 28　人の壁が 1 人の場合の角度 θ_1

6.3.2　プログラムへの反映

6 章 3 節 1 項で求めた$\frac{\theta_n}{2}$を、scan_change というプログラムに反映させる。scan_change については 6 章 6 節にて解説する。ここで、我々は 6 章 3 節 1 項で計算を行う際に、正確な計算ができておらず、scan_change では表 6 に示す$\frac{\theta_n'}{2}$を用いてプログラム及び実験を行った。今後の章で示すデータはすべて$\frac{\theta_n'}{2}$を用いたものである。

表 6　θ n/2 と θ 'n/2 と比較

n	θ n/2(rad)	θ 'n/2[rad]	θ n/2(°)	θ 'n/2[°]
4	0.338945758	0.34034	19.4201614	19.5000456
3	0.258497514	0.24871	14.8108165	14.2500333
2	0.174483074	0.16581	9.9971437	9.5002132
1	0.087910621	0.10036	5.0369076	5.7502044

このミスにより、実験結果に影響を及ぼす可能性があった。しかし、検証の結果、実験結果は変わらないと考えられる。以下に理由を説明する。LiDAR の仕様上、ロボットの真正面を 0° とし、度数法における 1° 刻みで角度が定義されているので（図 29）、scan_change のプログラム上でも$0.01745\ldots[rad]$刻みの数値しか現れない。つまり、実験結果に影響が出なかったのは偶然にも、θ_n[°]とθ_n'[°]の整数部分が一致していたからである（表 6 黄色部分）。以上の理由から、上記の 2 通りの数値を用いたプログラムは同じ結果をなるはずである。

そのため、7 章の実験結果は本節で示した方針通りに動作していると考えている。

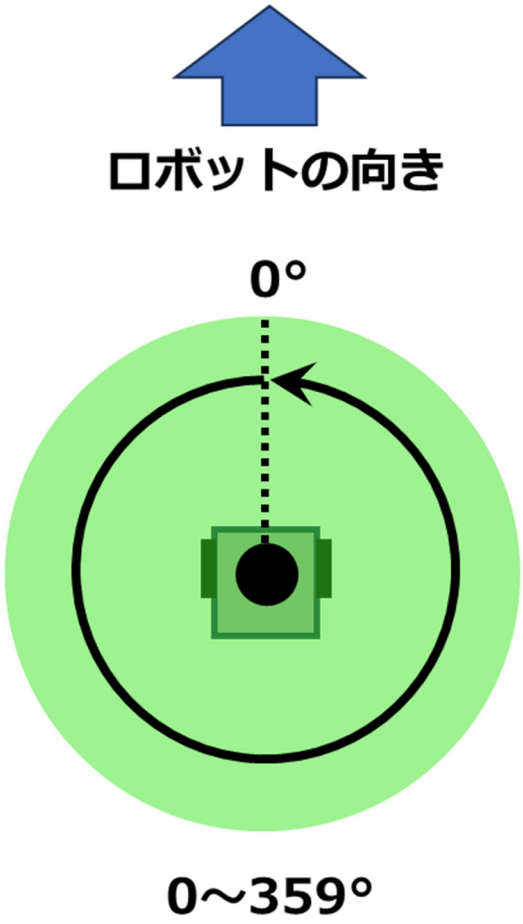


図 29　LiDAR の角度の仕様

上記で説明した我々のミスを踏まえ、$\frac{\theta_n'}{2}$を ignore_n[rad]（図 30 赤・青部分）とし、表 7 に示す。なお、ignore_n とは、scan_change 上で扱う変数である。さらに、その人の壁を構成する人数ごとの例を図 31、図 32、図 33、図 34 に掲載する。図 31～図 34 では、scan_change 上で起こっていることをイメージしやすいように弧度法ではなく度数法で表している。

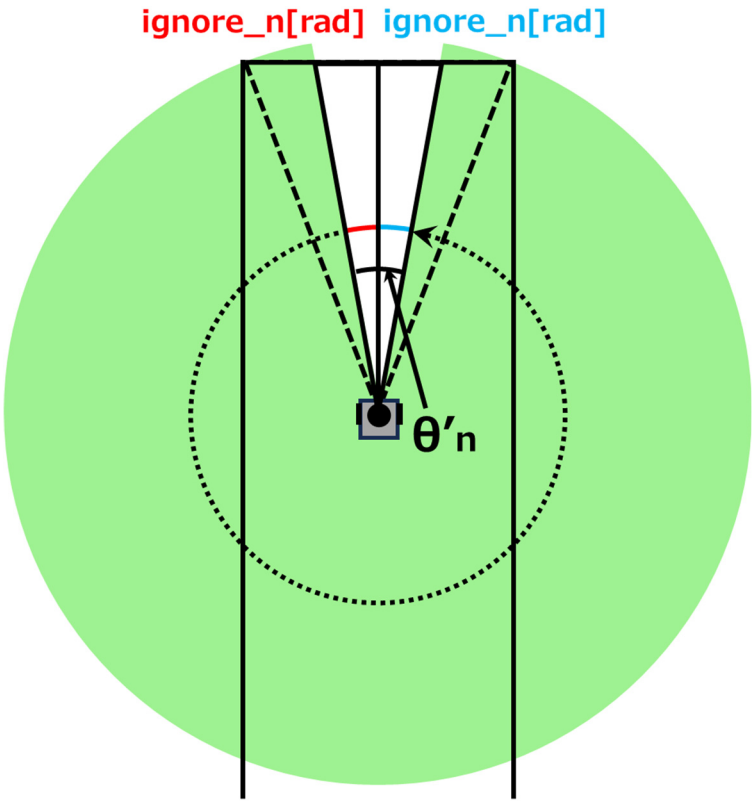


図 30　LiDAR の仕様による scan_change 上でのイメージ

表 7　ignore_n の値

ignore_4	0.34034
ignore_3	0.24871
ignore_2	0.16581
Ignore_1	0.10036

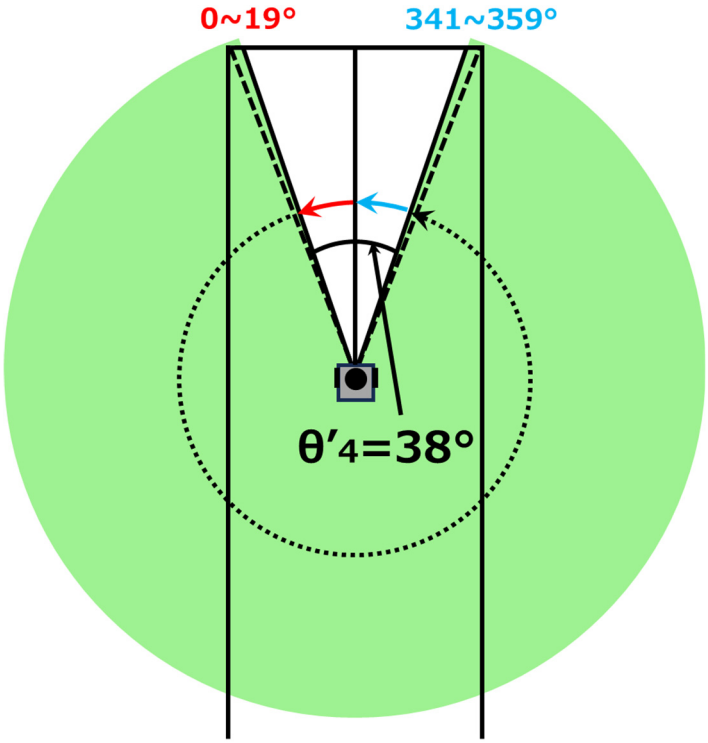


図 31　人の壁が 4 人（ignore_4）の場合

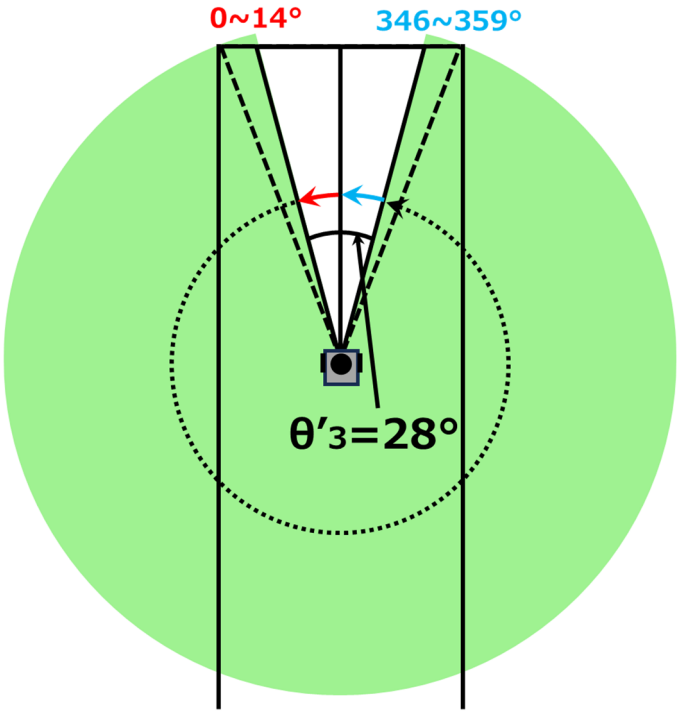


図 32　人の壁が 3 人（ignore_3）の場合

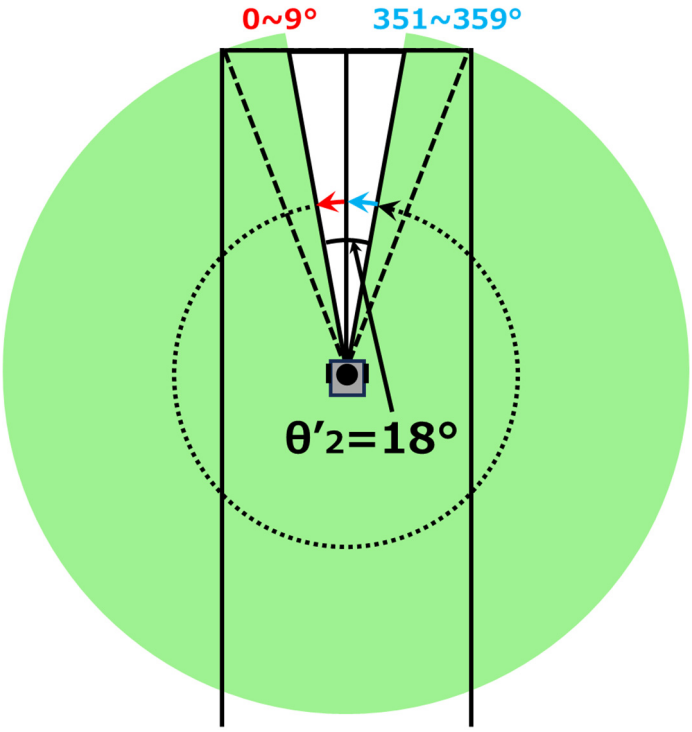


図 33　人の壁が 2 人（ignore_2）の場合

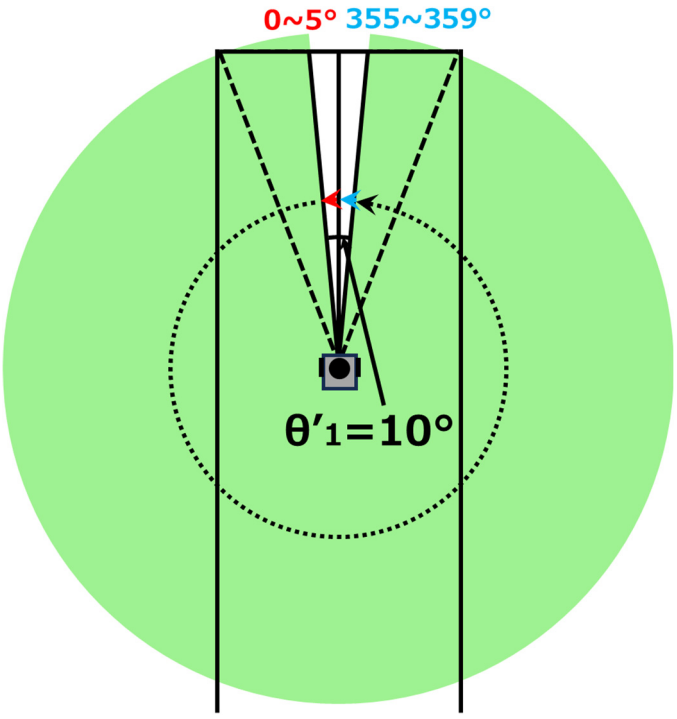


図 34　人の壁が 1 人（ignore_1）の場合

6.4　OpenPose から ROS へ認識した人数を送る方法（大島）

6.4.1　ROS とは

ROS 内ではプログラムがノードとして独立した実行ファイルのように扱われており、それぞれのノードはデータの送受信で連携がとられている。ROS を使用するには、まず roscore という ROS の基本的な処理を立ち上げるコマンドを実行する必要がある。このコマンドでは、ROS Master（ノードや 6 章 4 節 2 項で解説するトピックを管理するもの)、ROS Parameter Server（ノードが使用するパラメータを管理するサーバ)、rosout（ログ用のノード）を起動させている。roscore を実行した後、ROS で実行したいプログラムを使用することとなる。本研究では、Gmapping（ROS の LiDAR SLAM)、teleop-twist-keyboard（移動ロボットをキーボード入力で操作するプログラム)、scan_change（一定角度の距離データを除外するプログラム）の 3 つを使用している。

6.4.2　プログラム間の連携方法

6 章 4 節 1 項でも解説した通りノードは独立して動作しているため、連携をとるためにノード間でデータを送受信する必要がある。ROS では様々なプログラミング言語が用いられるため、独自のメッセージ型であるトピックとしてこれを行っている。図 35 に従来の手法で Gmapping を動作させたときのノードとトピックの関係図を示す。turtlebot3_lds は移動ロボットの LiDAR センサから得られた距離データを scan トピックとして送信するノードであり、送信されたトピックは turtlebot3_slam_gmapping が受信しているのが分かる。同様に他のノード間での連携もトピックで行われている。

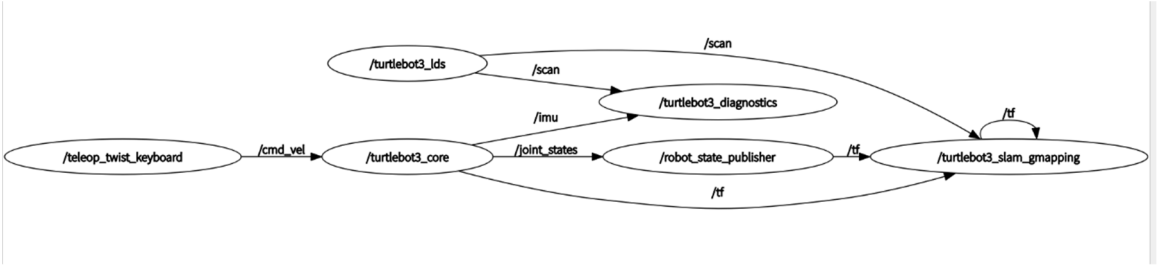


図 35　従来の GMapping のノードとトピックの関係図

6.4.3 rosparam

6 章 4 節 2 項で説明した通り、ノード間ではトピックとしてデータを送受信している。しかし、OpenPose は ROS のプログラムではないため、ノードとして登録されていない。そのため、別の方法で ROS にデータを送信する必要がある。本研究では rosparam[6]というコマンドを用いてパラメータサーバにパラメータを登録することでこれを行う。rosparam コマンドはターミナルでも実行可能であり、rosparam set （パラメータ名） （値）のように実行することでパラメータの変更が可能である。また、rosparam get （パラメータ名）のように実行すると任意のパラメータに登録されている値が取得できる。

6.5　OpenPose のプログラム変更

OpenPose はインストールした状態では ROS との連携が取れないため、人数のデータを取得して ROS に送信するようにプログラムを変更する必要がある。本研究では、OpenPose の tutorial_api_cpp の 16_synchronous_custom_postprocessing.cpp[7]に C++で追加を行った。追加した箇所はコード 1 の緑で示した部分である。まず、OpenPose では独自の Array 型のオブジェクトである poseIds で認識した人物の番号を管理している[8]。これは違うフレームであっても同じ番号であれば、同じ人物として認識されているということである。Array 型オブジェクトでは任意の次元のサイズを取得する関数 getSize（const int index）が定義されており、poseIds でこの関数を使用することで人物の番号の次元のサイズ、すなわち人数を取得できる（コード 1①)。また取得した人数を ROS に送信する際、すべてのフレームで送信してしまうと処理が重くなるため、1 フレーム前と人数が変わったときのみ ROS に送信する（コード 1②)。この際、ROS に人数を送信する手法である rosparam のコマンドを、人数の変数と std::string 型で結合した後 const char 型に変換し、system 関数を用いて実行した（コード 1③)。上記の処理は OpenPose が人物認識を行った後、出力が行われる前に実行する。これらの変更は、OpenPose Documentation[9]、ROS の tutorial[10]を参考に行った。また、付録に変更後の OpenPose のプログラムを記載する。

```
WUserPostProcessing()
  {
    // User's constructor here
    int prenum = 0;    ・・・②
  }

  int prenum;    ・・・②

  void initializationOnThread() {}
```

```
for (auto& datumPtr : *datumsPtr)
        {
                int people = datumPtr->poseIds.getSize(0);    ・・・①
                if (people != prenum)    ・・・②
                {
                        prenum = people;    ・・・②
                        std::string ros1 = "rosparam set detect " + std::to_string(people);    ・・・③
                        const char* ros = ros1.c_str();    ・・・③
                        int success = system(ros);    ・・・③
                }
        }
```

コード 1　OpenPose のプログラムの変更箇所

6.6　一定角度の距離データを除外するプログラム

OpenPose から送信された人数をもとに一定角度の距離データを除外する python プログラムを web ページ[11]を参考に作成した。コード 2 は完成したプログラムであり、緑で示した部分は web ページのプログラムに追加した部分である。まず新しく scan_change という名前でノードを作成し（コード 2①)、detect という名前で人数を格納するパラメータを作成する（コード 2②)。その後 scan_raw というトピックを取得し、convert_scan の処理を行う。convert_scan の処理では detect の数値を取得する。そして新たに作成した scan トピックに、detect の数値を用いて 6 章 3 節 2 項表 7 で決定した範囲の角度は非数“nan”を代入し、範囲外の角度はそのままの数値を代入している（コード 2③)。この処理は scan トピックの角度データと scan トピックの最小最大値、新たに作成した ignore_1 などの変数を用いて比較し、detect との論理演算子をとることで行っている。また、6 章 3 節 2 項表 7 の ignore_4 以上の角度を除外してしまうと後ろのパーティションの距離データが取得できなくなるため、detect が 5 以上の場合は 4 人の場合と同じ処理としている。そして、detect が 0 なら全範囲でそのままの数値の scan トピックを作成している。このように新しく作成された scan トピックを changed_scan トピックとして発行しているのである（コード 2④)。また、これらの処理は OpenPose の処理速度に合わせて 10Hz で動作するようにしている（コード 2⑤)。付録に全体のプログラムを記載する。作成したプログラムでは可読性を考慮して scan_raw という名前で変更前のトピックを定義しているが、turtlebot3_lds では scan という名前で発行されているため、プログラムを起動する launch ファイル中もしくは run コマンドのオプションにてトピックを remap してから実行する。remap とは ROS ノード起動時に書き換える処理のことである。

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

import rospy
import math    ・・・③
import numpy as np
from sensor_msgs.msg import LaserScan

scan_publisher = None
detect = 0    ・・・②
def convert_scan(data):
        global scan_publisher

        ignore_1 = 0.10036
        ignore_2 = 0.16581
        ignore_3 = 0.24871
        ignore_4 = 0.34034        ・・・③

        filtered_scan = LaserScan()
        filtered_scan.header = data.header
        filtered_scan.angle_min = data.angle_min
        filtered_scan.angle_max = data.angle_max
        filtered_scan.angle_increment = data.angle_increment
        filtered_scan.time_increment = data.time_increment
        filtered_scan.scan_time = data.scan_time
        filtered_scan.range_min = data.range_min
        filtered_scan.range_max = data.range_max
        detect = rospy.get_param('detect')    ・・・③
        for i, angle in enumerate(np.arange(data.angle_min, data.angle_max, data.angle_increment)):
                if detect == 1 and ((data.angle_min <= angle <= ignore_1 ) or (2*math.pi - ignore_1 <= angle <= data.angle_max)):
                        filtered_scan.ranges.append(float('nan'))
                elif detect == 2 and ((data.angle_min <= angle <= ignore_2 ) or (2*math.pi - ignore_2 <= angle <= data.angle_max)):
                        filtered_scan.ranges.append(float('nan'))
                elif detect == 3 and ((data.angle_min <= angle <= ignore_3 ) or (2*math.pi - ignore_3 <= angle <= data.angle_max)):
                        filtered_scan.ranges.append(float('nan'))
                elif detect >= 4 and ((data.angle_min <= angle <= ignore_4 ) or (2*math.pi - ignore_4 <= angle <= data.angle_max)):
                        filtered_scan.ranges.append(float('nan'))
                else:
                        filtered_scan.ranges.append(data.ranges[i])
        scan_publisher.publish(filtered_scan)    ・・・④

if __name__ == '__main__':
        rospy.init_node('scan_change', log_level = rospy.DEBUG)    ・・・①
        if not rospy.has_param('detect'):    ・・・②
                rospy.set_param('detect',0)    ・・・②
        rospy.Subscriber('/scan_raw', LaserScan, convert_scan)
        scan_publisher = rospy.Publisher('/changed_scan', LaserScan, queue_size = 10)    ・・・④
        r = rospy.Rate(10)    ・・・⑤
        while not rospy.is_shutdown():
                r.sleep()
```

コード 2　一定角度の距離データを除外するプログラム

6.7　GMapping の変更

従来の GMapping は scan という名前の scan トピックを取得し LiDAR SLAM を実行するプログラムである。しかし、6 章 6 節で解説した changed_scan という scan トピックを取得するようプログラムを変更する必要があるため、launch ファイル中で remap するよう記述した。

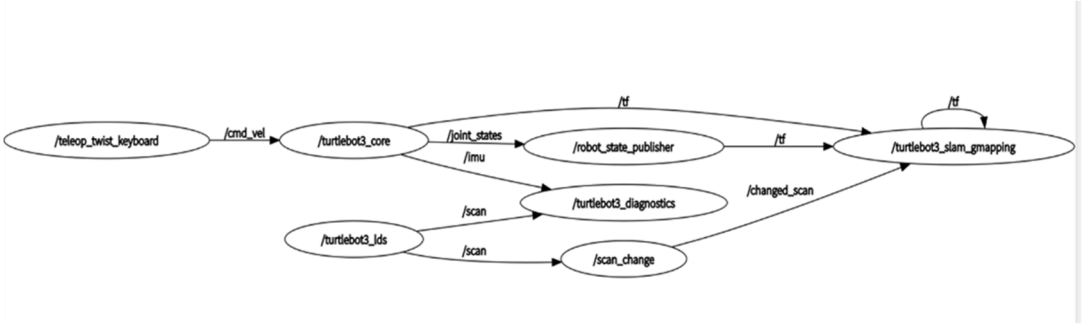
```
<launch>
  <!-- Arguments -->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="configuration_basename" default="turtlebot3_lds_2d.lua"/>
  <arg name="set_base_frame" default="base_footprint"/>
  <arg name="set_odom_frame" default="odom"/>
  <arg name="set_map_frame"  default="map"/>

  <!-- Gmapping -->
  <node pkg="gmapping" type="slam_gmapping" name="turtlebot3_slam_gmapping" output="screen">
    <param name="base_frame" value="$(arg set_base_frame)"/>
    <param name="odom_frame" value="$(arg set_odom_frame)"/>
    <param name="map_frame"  value="$(arg set_map_frame)"/>
    <rosparam command="load" file="$(find turtlebot3_slam)/config/gmapping_params.yaml" />
    <remap from="scan" to="changed_scan"/>
  </node>
</launch>
```

コード 3　GMapping のプログラムの変更箇所

6.8　提案手法のノードとトピックの関係図

本研究の提案手法でのノードとトピックの関係図を以下に示す。turtlebot_lds ノードから turtlebot3_slam_gmapping ノードへの scan トピックに scan_change ノードの処理が挿入され、changed_scan トピックに変換されていることがわかる。この処理で人の存在に対応する障害物を除外している。



第 7 章　実験（松山）

7.1　実験目的

改造した TurtleBot3 を移動ロボットとして用い、従来の LiDAR SLAM と提案手法の自己位置推定の比較実験を行い、人の存在に対する頑強性を確認することを目的として実験を行う。

7.2　実験詳細

実験は、工学院大学八王子キャンパス 4 号館 8 階の廊下の一部をパーティションによる壁で区切り、使用した。この環境を使用した理由を以下で説明する。

移動ロボットが自己位置推定をする際は、周囲 360 度の距離データに基づき自己位置推定を行う。そのため、前方に人の壁が現れたとしても、側面と後方側で距離データと地図がマッチングしていれば、自己位置推定にずれは生じにくい。しかし、長い廊下などの環境において、側面はランドマークとなるような場所が少ない単純な環境であり、移動ロボットが作成した環境地図と距離データのマッチングが困難である。また、後方は LiDAR の範囲内に障害物がないため、距離データを読み取ることができない。そのため、前方に人の壁が現れた場合、主に人の壁の距離データを元に自己位置推定を行うことになる。これにより、人の壁の後方にある本来の壁（パーティションの壁）の距離データを読み取ってしまったと誤認し、自己位置推定にずれが生じる。

図 36 が実験場所の見取り図、図 37 が実際の実験環境の様子である。図 36 に示されている通り、地図の下方から上方へ向かって、移動ロボットを直進させ、パーティションの壁から 3m 地点の目印で停止し、自己位置推定をする形で実験を行った。

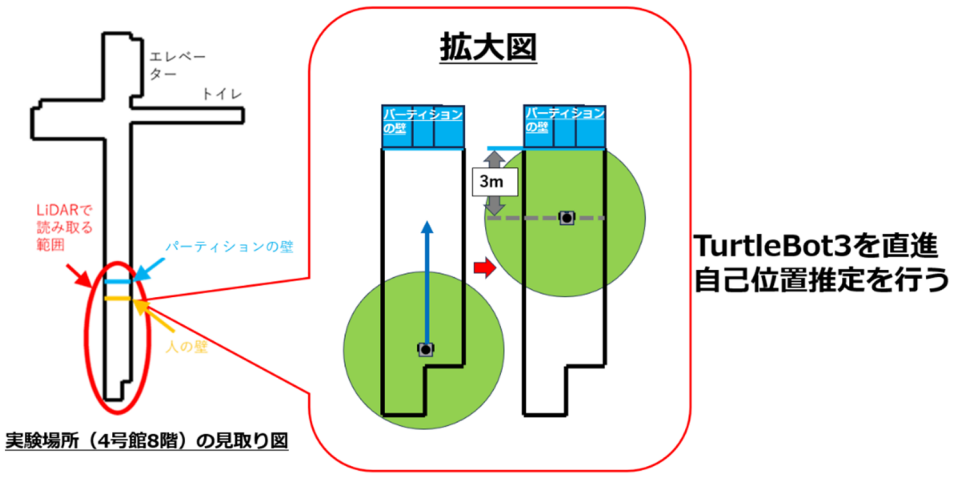


図 36　実験場所の見取り図

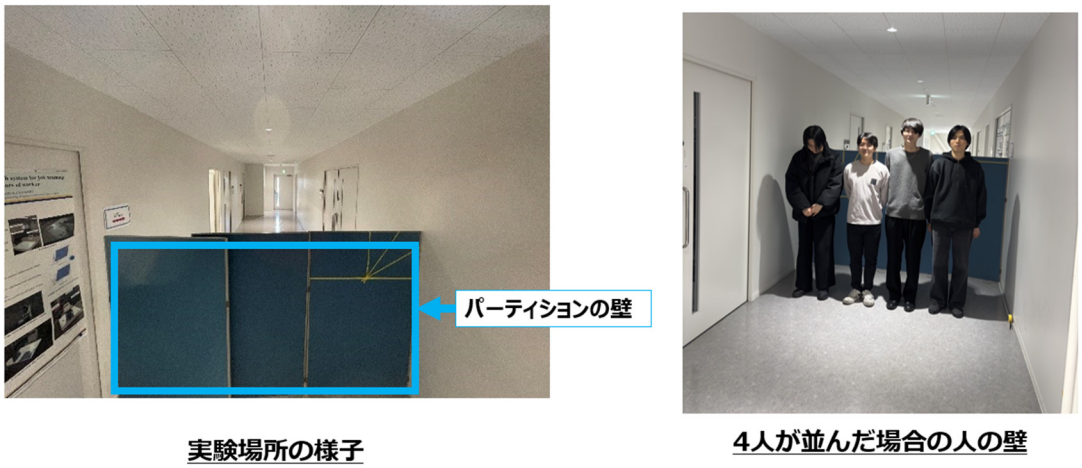


図 37　実際の実験環境の様子

また、実験では SLAM シミュレータ上で表示される環境地図を使用する。図 38 に作成される環境地図の見方を示した。

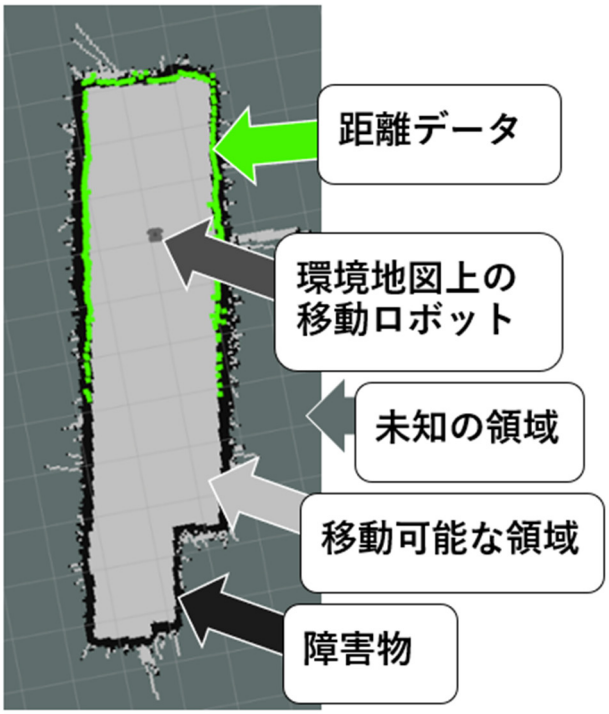


図 38　環境地図の見方

7.3　実験条件

実験は以下の図 39 に示すように、A、B、C、3 つの条件の比較で行った。

A: 事前に作成した地図で自己位置推定をした場合。

B: 地図作成後、廊下の途中に 1〜4 人で構成される人の壁を設置した状態で、従来の LiDAR SLAM を用いて自己位置推定を行った場合。

C: 地図作成後、廊下の途中に 1〜4 人で構成される人の壁を設置した状態で、人の存在を考慮して自己位置推定する場合。

なお、 B、C の人の壁はパーティションの壁から 1m 手前の地点に配置し 、A、B、C ともに現実の位置で、移動ロボットとパーティションの壁との距離が印をつけた 3 m の地点で自己位置推定を行った。また、人の壁を 4 人以下の構成とした理由は、人の壁が 5 人以上になった際は、人の壁が廊下をほとんど完全にふさぐような形となり、後方にあるパーティションの壁の距離データを読み取ることができなくなってしまうためである。

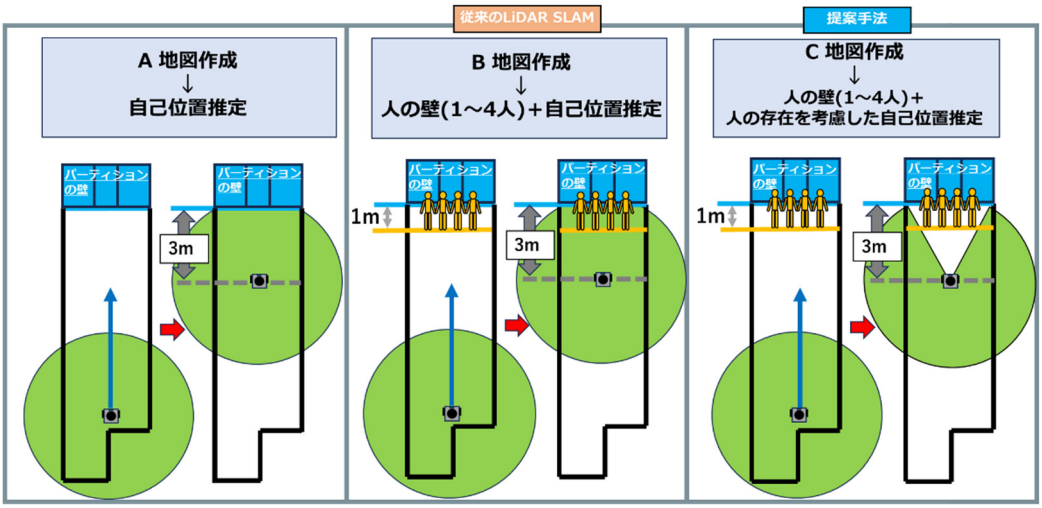


図 39　比較実験の条件

7.4　3 条件を比較する手法

図 39 の 3 つの条件を比較する手法について図 40 を用いて説明する。図 40 における A、B、C の 3 つの画像は、現実の位置で移動ロボットとパーティションの壁との距離が、A、B、C ともに 3m の地点の際に SLAM シミュレータ上で示された環境地図である。 図 40 の赤い矢印で示されているのが、推定されたロボットの位置を表している。水色の点線は地図作成されたパーティションの壁の位置、赤い点線は A の基準となる移動ロボットの位置を示したものである。

この図 40 の A の条件の場合にロボットの推定された位置と現実の位置が一致しているという前提とし、B と C の推定された移動ロボットの位置が、A の移動ロボットの推定されたロボットの位置からどれくらいずれているかを絶対値で比較した。

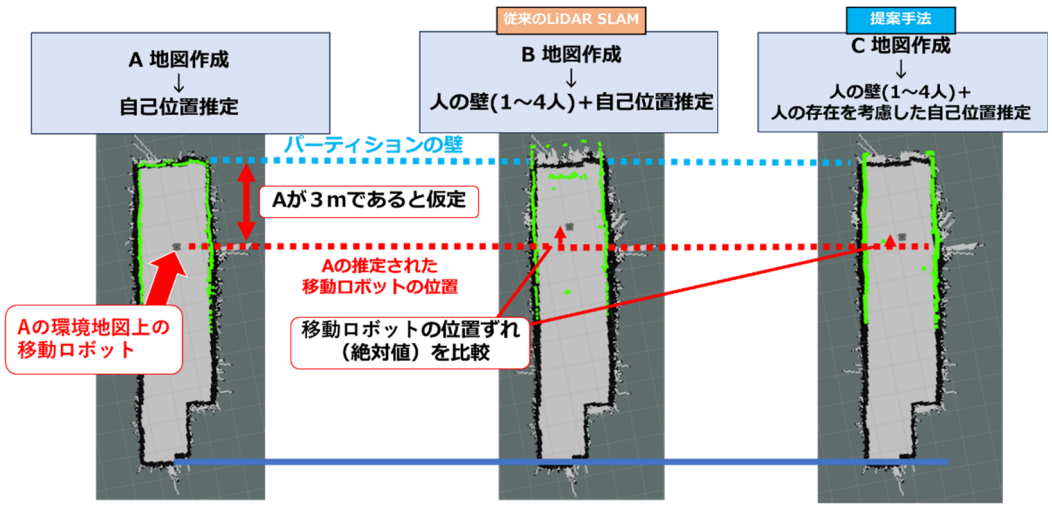


図 40　比較手法

7.5　実験結果

7.5.1　4 人の場合

図 41 に示すのが人の壁が 4 人の場合である。また、図 41 黄色・橙色の枠線で囲まれた部分を拡大したのが図 42 である。

図 42 の左は、B の上方の拡大図である。紫の矢印で示された緑色の線がパーティションの壁の距離データで、水色の矢印で示された黒の線がパーティションの壁の位置を示している。これらが重なった場合、マッチングがうまくいったことを示すが、重なっていないことがわかる。なお、黄色の矢印で示された緑の線が人の壁の距離データであり、今回の実験で視認性を上げるため、距離データのピクセルサイズを変更した。

右図は C の上方の拡大図である。こちらは、人の壁がある位置の距離データを除外しているため、地図上のおけるパーティションの壁とのマッチングの精度が向上していることがわかる。ただし、壁とのマッチングは完全には一致しなかった。

A における移動ロボットの位置を基準としたずれの結果は、B は 1.12m、C は 0.56ｍとなった。

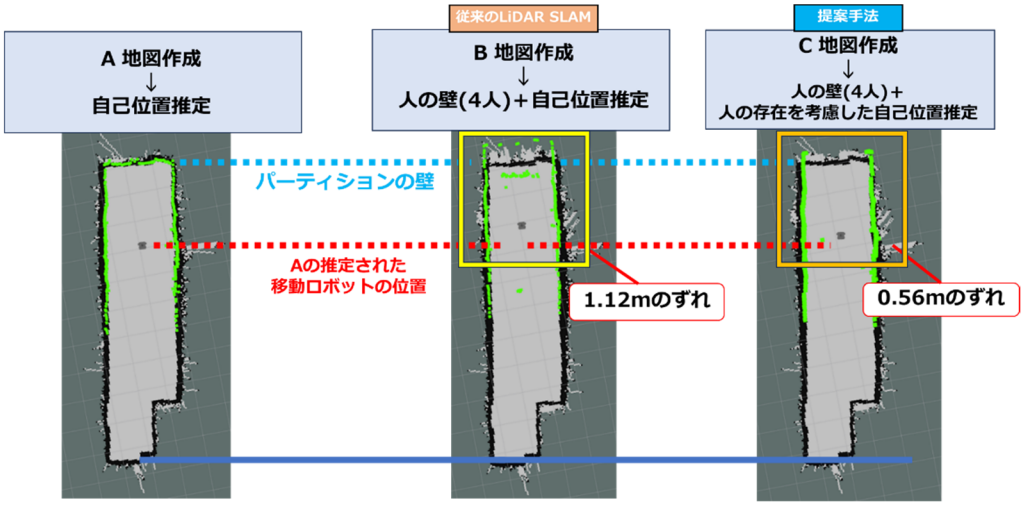


図 41　4 人の場合の結果

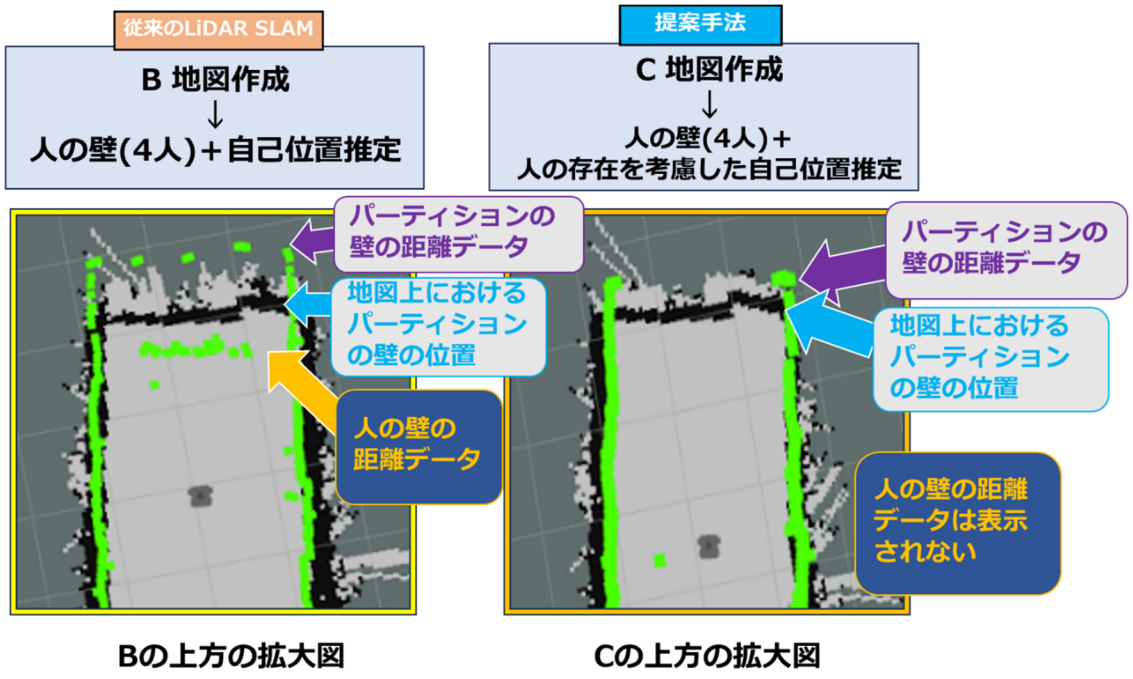


図 42　4 人の場合における B・C の上方の拡大図

7.5.2　3 人の場合

図 43 は人の壁が 3 人の場合の結果である。また、黄色・橙色の枠線で囲まれた部分を拡大したのが図 44 である。

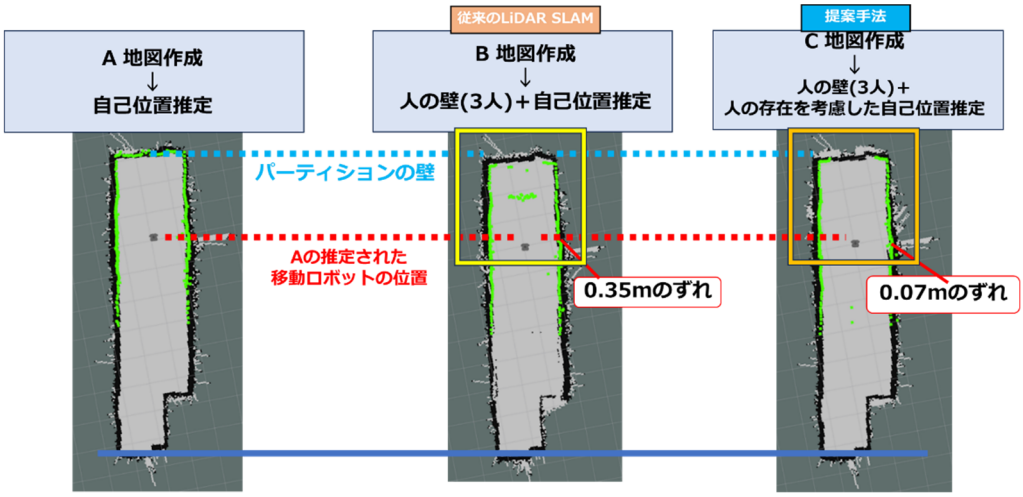


図 43　3 人の場合の結果

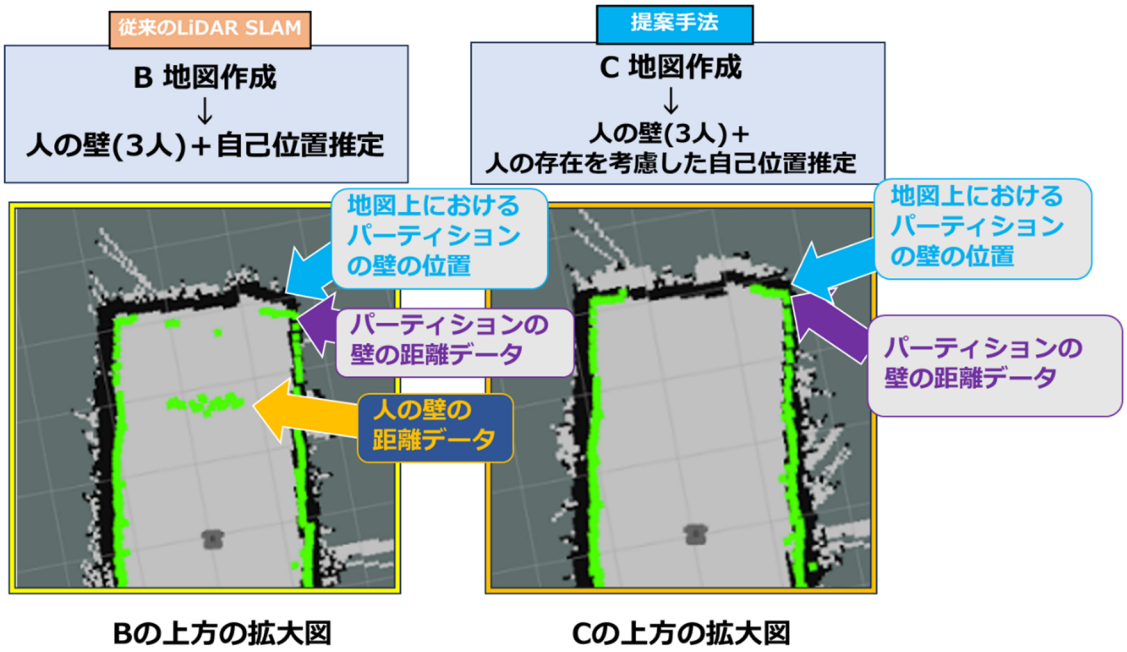


図 44　3 人の場合における B・C の上方の拡大図

B におけるパーティションの壁の距離データと、パーティションの壁の位置は、4 人の時よりは近づいた。C は、4 人の際と比べ、人の人数が減ったことにより、除外する距離データが減っている。

ずれの結果は、B は 0.35m、C は 0.07ｍとなった。

7.5.3　2 人の場合

図 45 は人の壁が 2 人の場合の結果である。また、図 46 は B と C の拡大図である。ずれの結果は、B は 0.28m、C は 0.07m となった。

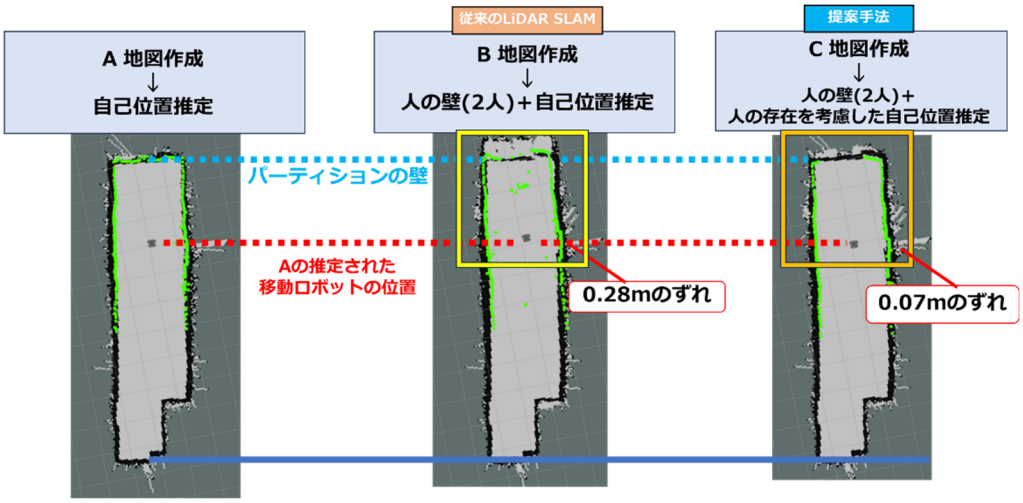


図 45　2 人の場合の結果

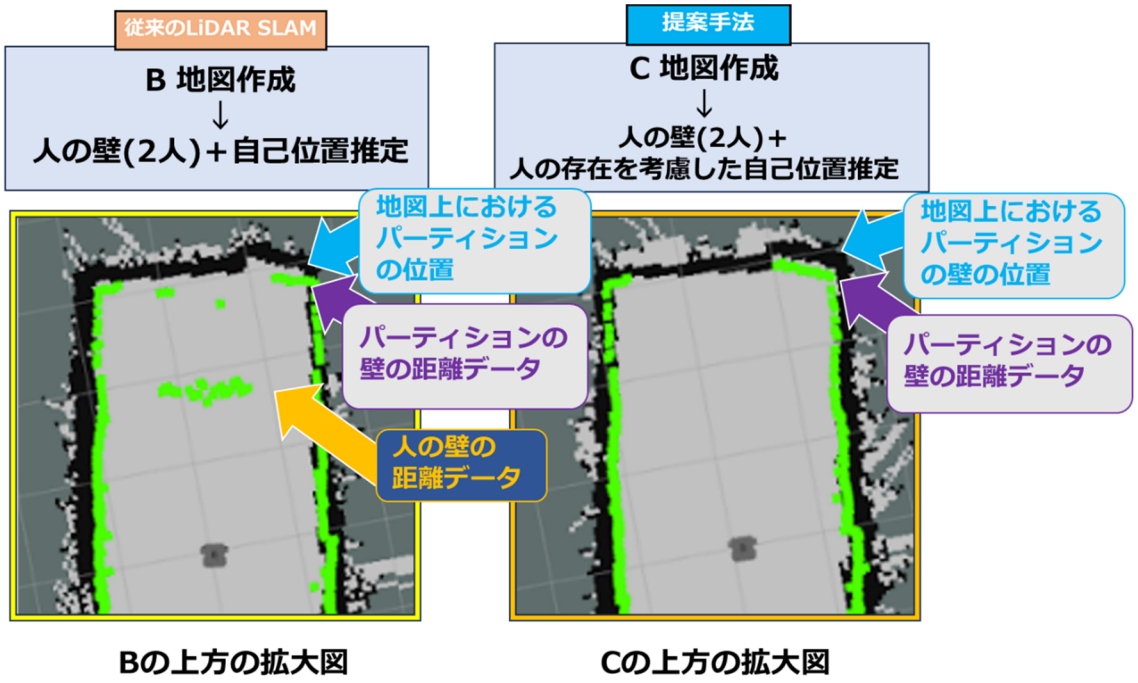


図 46　2 人の場合における B・C の上方の拡大図

7.5.4　1 人の場合

図 47 は人の壁が 1 人の場合の結果である。また、図 48 は B と C の拡大図である。ずれの結果は、B は 0.28m、C は 0.07m となった。

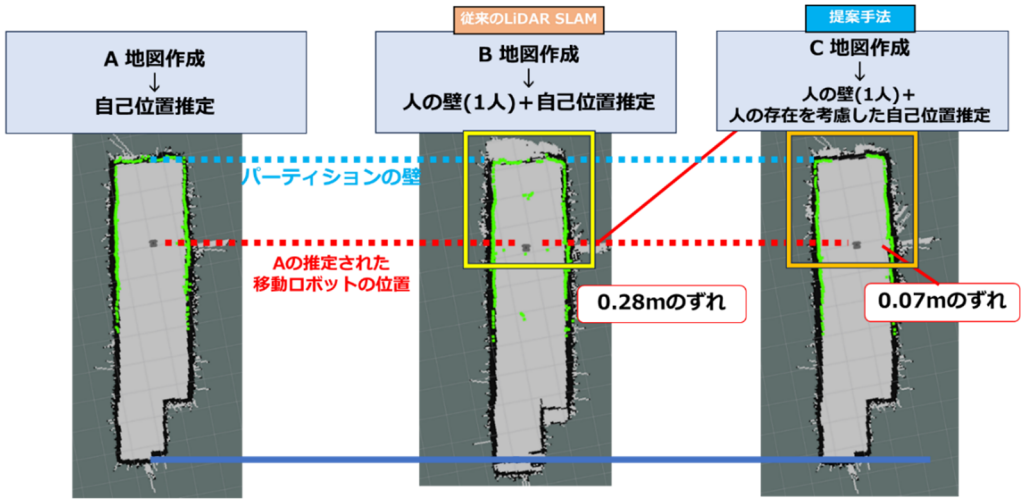


図 47　1 人の場合の結果

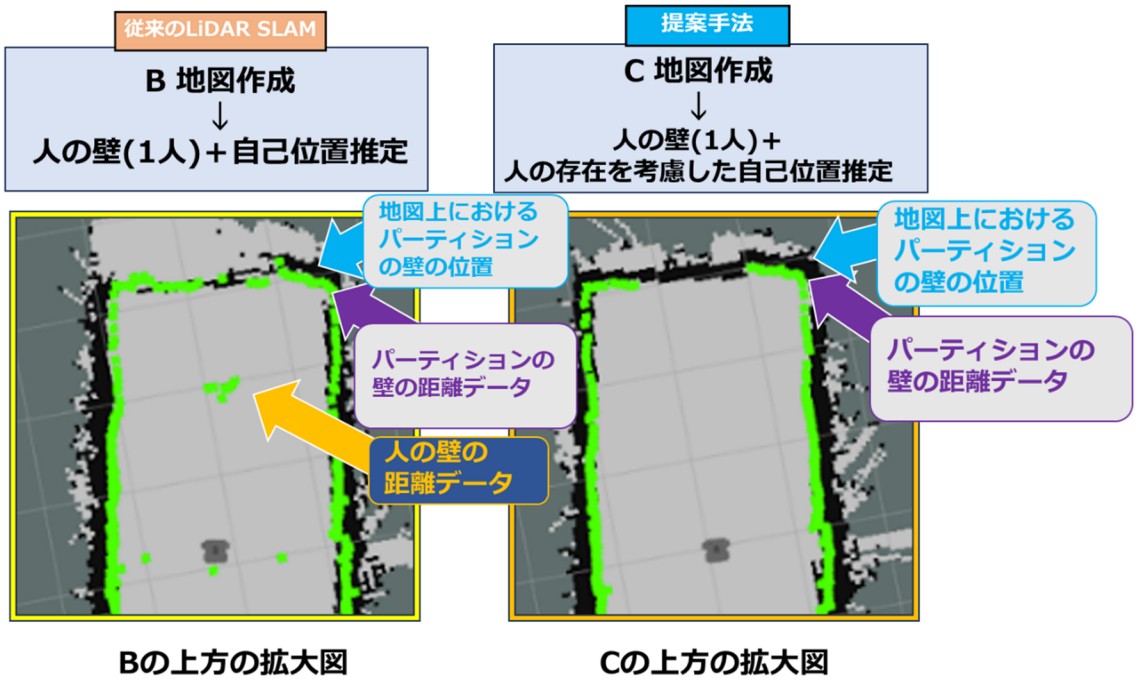


図 48　1 人の場合における B・C の上方の拡大図

7.6　考察

下記の図 49 が実験結果をまとめたグラフである。

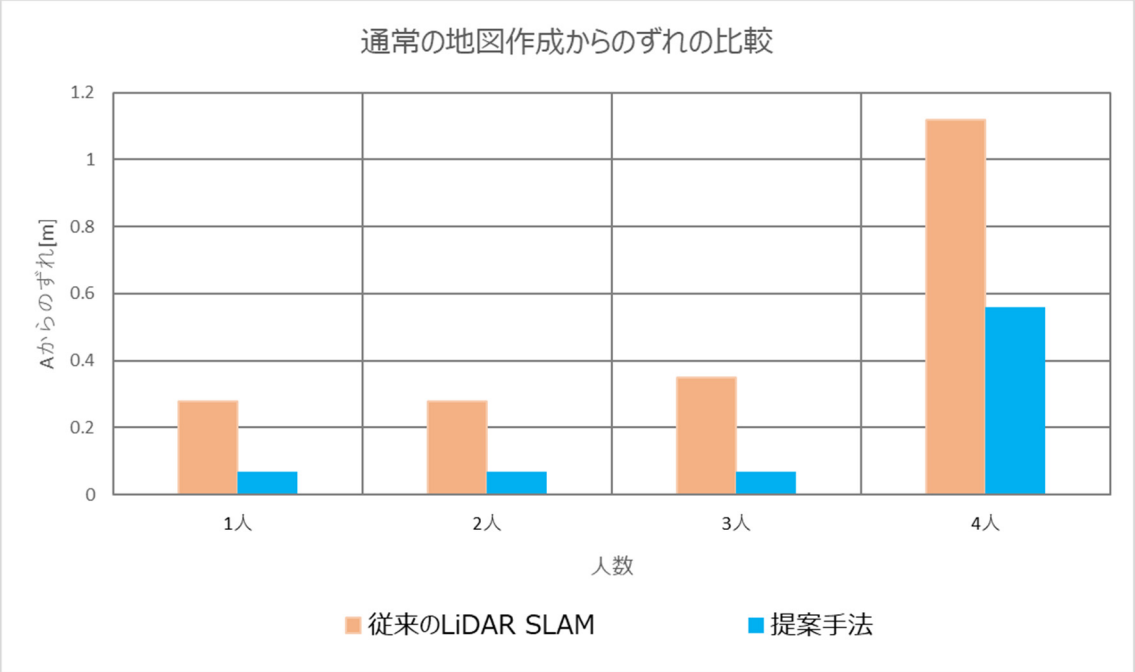


図 49　実験結果の比較

1～4 人すべての人数で自己位置推定の精度の改善が見られた。我々の考察では、従来の LiDAR SLAM においては、人の壁の距離データが、パーティションの壁の距離データであると誤認し地図とのマッチングがうまくいかなかったと推測する。提案手法においては、人の壁の距離データを除外しつつ、パーティションの壁の距離データの一部を読み取ったことにより、従来の LiDAR SLAM と比べ、うまくマッチングすることができたと考える。

一方で、提案手法にもずれが生じた理由は、人の壁の距離データを除外することによって、利用可能な距離データが減少し、マッチングの精度が低下したためと推測する。特に 4 人の時は、より多くの角度の距離データを除外したため、ずれが大きくなったと考える。

第 8 章　課題と結論（山下）

8.1　結論

本研究では、人の存在を考慮した頑強な自己位置推定の実現を目指した。そこで、移動ロボットに人物認識用 PC と単眼カメラを搭載する改造を行い、OpenPose を用いた人物認識システムと導入した。実験では従来の LiDAR SLAM の自己位置推定と、我々の提案手法の自己位置推定の比較実験を行った。その結果、1 から 4 人すべての人数で自己位置推定の改善が見られた。これにより、人が存在する環境下における移動ロボットの頑強な自己位置推定を実現することが出来た。

8.2　課題と解決策

本研究での提案手法には、まだ課題が残されている。それは、人が離れて立っていた場合、現段階の手法では図 50 左のように、中央の人がいない場所も距離データを取得できないことである。解決策として、図 50 右のように人が存在する領域のみを点群を除外する手法を導入することで、より頑強な自己位置推定を目指せると考えている。

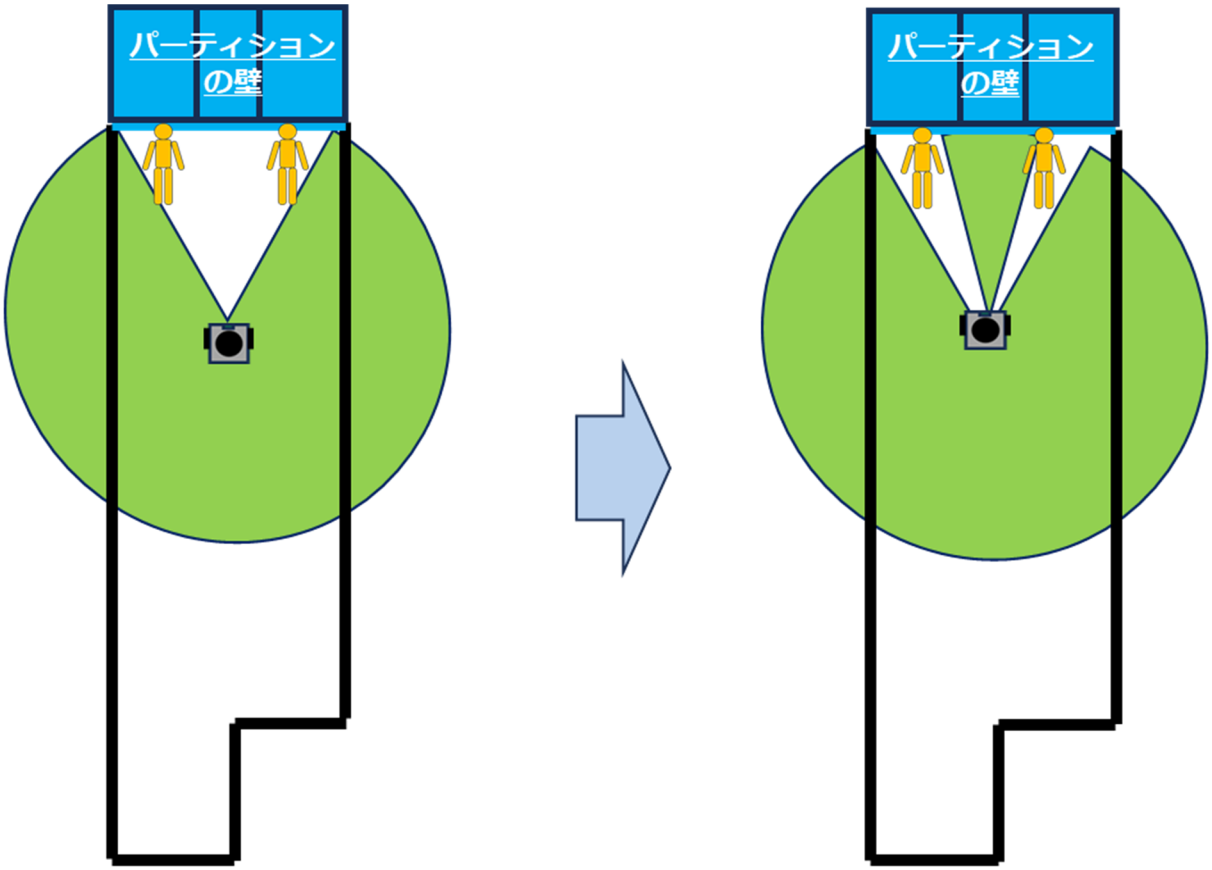


図 50　左:現在の手法　右:今後目指すべき手法

参考文献

[1]第 24010 号　世界のサービスロボット市場を調査｜Fuji Keizai Group
https://www.fuji-keizai.co.jp/press/detail.html?cid=24010（閲覧日：2025 年 2 月 6 日）

[2] 前川　清石，“工場内搬送および物流倉庫向け自律移動ロボットの開発”，システム制御情報学会誌，Vol. 64，No. 5，pp. 177-181，2020

[3] 萬　礼応，“物体の属性を考慮した環境変化に対して頑強な自己位置推定”，システム／制御／情報，Vol. 66，No. 4，pp. 121-126，2022

[4] google-ai-edge，“MediaPipe”，GitHub-Repository-
https://github.com/google-ai-edge/mediapipe（閲覧日：2025 年 1 月 31 日）

[5] CMU-Perceptual-Computing-Lab，“OpenPose”，GitHub Repository
https://github.com/CMU-Perceptual-Computing-Lab/openpose（閲覧日：2025 年 1 月 31 日）

[6] ROS.org，“rosparam”，Documentation
https://wiki.ros.org/rosparam（閲覧日：2025 年 2 月 8 日）

[7] CMU-Perceptual-Computing-Lab，
“16_synchronous_custom_postprocessing.cpp“，GitHub Repository
https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/examples/tutorial_api_cpp/16_synchronous_custom_postprocessing.cpp（閲覧日：2025 年 2 月 8 日）

[8] CMU-Perceptual-Computing-Lab，“poseIds”，Documentation
https://cmu-perceptual-computing-lab.github.io/openpose/web/html/doc/structop_1_1_datum.html#aba90dccffb5a830296231bd430c4766c（閲覧日：2025 年 2 月 8 日 9

[9] CMU-Perceptual-Computing-Lab，“OpenPose”，Documentation
https://cmu-perceptual-computing-lab.github.io/openpose/web/html/doc/index.html（閲覧日：2025 年 2 月 8 日）

[10] ROS.org, “Tutorials”, Documentation
https://wiki.ros.org/ja/ROS/Tutorials（閲覧日：2025 年 2 月 8 日）

[11] Robo Sapiens WEB SHOP, “ROS の LaserScan データの一定角度の範囲を無視する方法”, 2023 年 5 月 9 日
https://rb-sapiens-shop.com/blogs/article/ros-ignore-specific-degree-scan?srsltid=AfmBOopCz4HkZcrB-NRInplsk1SroN-WsVwIzGTM3y6KQ-brVPezeRTn
（閲覧日：2025 年 2 月 8 日）

謝辞

本研究を進めていくにあたり、2 年間にわたるご指導ご鞭撻を頂いた金丸隆志教授に心より感謝を申し上げます。4 年生前期では中々研究テーマが決定せず、悪戦苦闘する中、金丸教授からの助言により研究がようやく始まった際には、大変お世話になりました。

今後もこの 2 年間で得た経験を活かし、日々精進していきたいと思います。

付録

scan_change

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

import rospy
import math
import numpy as np
from sensor_msgs.msg import LaserScan

scan_publisher = None
detect = 0
def convert_scan(data):
        global scan_publisher

        ignore_1 = 0.10036
        ignore_2 = 0.16581
        ignore_3 = 0.24871
        ignore_4 = 0.34034

        filtered_scan = LaserScan()
        filtered_scan.header = data.header
        filtered_scan.angle_min = data.angle_min
        filtered_scan.angle_max = data.angle_max
        filtered_scan.angle_increment = data.angle_increment
        filtered_scan.time_increment = data.time_increment
        filtered_scan.scan_time = data.scan_time
        filtered_scan.range_min = data.range_min
        filtered_scan.range_max = data.range_max
        detect = rospy.get_param('detect')
        for i, angle in enumerate(np.arange(data.angle_min, data.angle_max,
data.angle_increment)):
                if detect == 1 and ((data.angle_min <= angle <= ignore_1 ) or
(2*math.pi - ignore_1 <= angle <= data.angle_max)):
```

```
                        filtered_scan.ranges.append(float('nan'))
                elif detect == 2 and ((data.angle_min <= angle <= ignore_2 ) or
(2*math.pi - ignore_2 <= angle <= data.angle_max)):
                        filtered_scan.ranges.append(float('nan'))
                elif detect == 3 and ((data.angle_min <= angle <= ignore_3 ) or
(2*math.pi - ignore_3 <= angle <= data.angle_max)):
                        filtered_scan.ranges.append(float('nan'))
                elif detect >= 4 and ((data.angle_min <= angle <= ignore_4 ) or
(2*math.pi - ignore_4 <= angle <= data.angle_max)):
                        filtered_scan.ranges.append(float('nan'))
                else:
                        filtered_scan.ranges.append(data.ranges[i])
        scan_publisher.publish(filtered_scan)

if __name__ == '__main__':
        rospy.init_node('scan_change', log_level = rospy.DEBUG)
        if not rospy.has_param('detect'):
                rospy.set_param('detect',0)
        rospy.Subscriber('/scan_raw', LaserScan, convert_scan)
        scan_publisher = rospy.Publisher('/changed_scan', LaserScan, queue_size = 10)
        r = rospy.Rate(10)
        while not rospy.is_shutdown():
                r.sleep()
```

OpenPose

```
// ------------------------ OpenPose C++ API Tutorial - Example 15 - Custom Post-processing ------------------------
// Synchronous mode: ideal for production integration. It provides the fastest results with respect to runtime
// performance.
// In this function, the user can implement its own post-processing, i.e., his function will be called after OpenPose
// has processed the frames but before saving or visualizing any result.

// Third-party dependencies
#include <opencv2/opencv.hpp>
// Command-line user interface
#include <openpose/flags.hpp>
// OpenPose dependencies
#include <openpose/headers.hpp>

// This worker will just invert the image
class WUserPostProcessing : public op::Worker<std::shared_ptr<std::vector<std::shared_ptr<op::Datum>>>>
{
public:
    WUserPostProcessing()
    {
        // User's constructor here
        int prenum = 0;
    }

    int prenum;

    void initializationOnThread() {}

    void work(std::shared_ptr<std::vector<std::shared_ptr<op::Datum>>>& datumsPtr)
    {
        try
```

```
        {
            // User's post-processing (after OpenPose processing & before OpenPose outputs) here
                // datumPtr->cvOutputData: rendered frame with pose or heatmaps
                // datumPtr->poseKeypoints: Array<float> with the estimated pose
            if (datumsPtr != nullptr && !datumsPtr->empty())
            {
                for (auto& datumPtr : *datumsPtr)
                {
                        int people = datumPtr->poseIds.getSize(0);
                        if (people != prenum)
                        {
                                prenum = people;
                                std::string ros1 = "rosparam set detect " + std::to_string(people);
                                const char* ros = ros1.c_str();
                                int success = system(ros);
                        }
                }
            }
        }
        catch (const std::exception& e)
        {
            this->stop();
            op::error(e.what(), __LINE__, __FUNCTION__, __FILE__);
        }
    }
};

void configureWrapper(op::Wrapper& opWrapper)
{
    try
    {
        // Configuring OpenPose

        // logging_level
```

```
        op::checkBool(
            0 <= FLAGS_logging_level && FLAGS_logging_level <= 255, "Wrong
logging_level value.",
            __LINE__, __FUNCTION__, __FILE__);
        op::ConfigureLog::setPriorityThreshold((op::Priority)FLAGS_logging_level);
        op::Profiler::setDefaultX(FLAGS_profile_speed);

        // Applying user defined configuration - GFlags to program variables
        // producerType
        op::ProducerType producerType;
        op::String producerString;
        std::tie(producerType, producerString) = op::flagsToProducer(
            op::String(FLAGS_image_dir), op::String(FLAGS_video),
op::String(FLAGS_ip_camera), FLAGS_camera,
            FLAGS_flir_camera, FLAGS_flir_camera_index);
        // cameraSize
        const auto cameraSize =
op::flagsToPoint(op::String(FLAGS_camera_resolution), "-1x-1");
        // outputSize
        const auto outputSize = op::flagsToPoint(op::String(FLAGS_output_resolution),
"-1x-1");
        // netInputSize
        const auto netInputSize = op::flagsToPoint(op::String(FLAGS_net_resolution),
"-1x368");
        // faceNetInputSize
        const auto faceNetInputSize =
op::flagsToPoint(op::String(FLAGS_face_net_resolution), "368x368 (multiples of 16)");
        // handNetInputSize
        const auto handNetInputSize =
op::flagsToPoint(op::String(FLAGS_hand_net_resolution), "368x368 (multiples of 16)");
        // poseMode
        const auto poseMode = op::flagsToPoseMode(FLAGS_body);
        // poseModel
        const auto poseModel =
op::flagsToPoseModel(op::String(FLAGS_model_pose));
        // JSON saving
```

```
        if (!FLAGS_write_keypoint.empty())
            op::opLog(
                "Flag `write_keypoint` is deprecated and will eventually be removed. Please, use `write_json`"
                " instead.", op::Priority::Max);
        // keypointScaleMode
        const auto keypointScaleMode = op::flagsToScaleMode(FLAGS_keypoint_scale);
        // heatmaps to add
        const auto heatMapTypes = op::flagsToHeatMaps(FLAGS_heatmaps_add_parts, FLAGS_heatmaps_add_bkg,

FLAGS_heatmaps_add_PAFs);
        const auto heatMapScaleMode = op::flagsToHeatMapScaleMode(FLAGS_heatmaps_scale);
        // >1 camera view?
        const auto multipleView = (FLAGS_3d || FLAGS_3d_views > 1 || FLAGS_flir_camera);
        // Face and hand detectors
        const auto faceDetector = op::flagsToDetector(FLAGS_face_detector);
        const auto handDetector = op::flagsToDetector(FLAGS_hand_detector);
        // Enabling Google Logging
        const bool enableGoogleLogging = true;

        // Initializing the user custom classes
        // Processing
        auto wUserPostProcessing = std::make_shared<WUserPostProcessing>();
        // Add custom processing
        const auto workerProcessingOnNewThread = true;
        opWrapper.setWorker(op::WorkerType::PostProcessing, wUserPostProcessing, workerProcessingOnNewThread);

        // Pose configuration (use WrapperStructPose{} for default and recommended configuration)
        const op::WrapperStructPose wrapperStructPose{
            poseMode, netInputSize, FLAGS_net_resolution_dynamic, outputSize,
```

```
keypointScaleMode, FLAGS_num_gpu,
            FLAGS_num_gpu_start, FLAGS_scale_number, (float)FLAGS_scale_gap,
            op::flagsToRenderMode(FLAGS_render_pose, multipleView),
poseModel, !FLAGS_disable_blending,
            (float)FLAGS_alpha_pose, (float)FLAGS_alpha_heatmap,
FLAGS_part_to_show, op::String(FLAGS_model_folder),
            heatMapTypes, heatMapScaleMode, FLAGS_part_candidates,
(float)FLAGS_render_threshold,
            FLAGS_number_people_max, FLAGS_maximize_positives,
FLAGS_fps_max, op::String(FLAGS_prototxt_path),
            op::String(FLAGS_caffemodel_path), (float)FLAGS_upsampling_ratio,
enableGoogleLogging};
        opWrapper.configure(wrapperStructPose);
        // Face configuration (use op::WrapperStructFace{} to disable it)
        const op::WrapperStructFace wrapperStructFace{
            FLAGS_face, faceDetector, faceNetInputSize,
            op::flagsToRenderMode(FLAGS_face_render, multipleView,
FLAGS_render_pose),
            (float)FLAGS_face_alpha_pose, (float)FLAGS_face_alpha_heatmap,
(float)FLAGS_face_render_threshold};
        opWrapper.configure(wrapperStructFace);
        // Hand configuration (use op::WrapperStructHand{} to disable it)
        const op::WrapperStructHand wrapperStructHand{
            FLAGS_hand, handDetector, handNetInputSize,
FLAGS_hand_scale_number, (float)FLAGS_hand_scale_range,
            op::flagsToRenderMode(FLAGS_hand_render, multipleView,
FLAGS_render_pose), (float)FLAGS_hand_alpha_pose,
            (float)FLAGS_hand_alpha_heatmap,
(float)FLAGS_hand_render_threshold};
        opWrapper.configure(wrapperStructHand);
        // Extra functionality configuration (use op::WrapperStructExtra{} to disable it)
        const op::WrapperStructExtra wrapperStructExtra{
            FLAGS_3d, FLAGS_3d_min_views, FLAGS_identification,
FLAGS_tracking, FLAGS_ik_threads};
        opWrapper.configure(wrapperStructExtra);
        // Producer (use default to disable any input)
```

```
        const op::WrapperStructInput wrapperStructInput{
            producerType, producerString, FLAGS_frame_first, FLAGS_frame_step,
FLAGS_frame_last,
            FLAGS_process_real_time, FLAGS_frame_flip, FLAGS_frame_rotate,
FLAGS_frames_repeat,
            cameraSize, op::String(FLAGS_camera_parameter_path),
FLAGS_frame_undistort, FLAGS_3d_views};
        opWrapper.configure(wrapperStructInput);
        // Output (comment or use default argument to disable any output)
        const op::WrapperStructOutput wrapperStructOutput{
            FLAGS_cli_verbose, op::String(FLAGS_write_keypoint),
op::stringToDataFormat(FLAGS_write_keypoint_format),
            op::String(FLAGS_write_json), op::String(FLAGS_write_coco_json),
FLAGS_write_coco_json_variants,
            FLAGS_write_coco_json_variant, op::String(FLAGS_write_images),
op::String(FLAGS_write_images_format),
            op::String(FLAGS_write_video), FLAGS_write_video_fps,
FLAGS_write_video_with_audio,
            op::String(FLAGS_write_heatmaps),
op::String(FLAGS_write_heatmaps_format), op::String(FLAGS_write_video_3d),
            op::String(FLAGS_write_video_adam), op::String(FLAGS_write_bvh),
op::String(FLAGS_udp_host),
            op::String(FLAGS_udp_port)};
        opWrapper.configure(wrapperStructOutput);
        // GUI (comment or use default argument to disable any visual output)
        const op::WrapperStructGui wrapperStructGui{
            op::flagsToDisplayMode(FLAGS_display,
FLAGS_3d), !FLAGS_no_gui_verbose, FLAGS_fullscreen};
        opWrapper.configure(wrapperStructGui);
        // Set to single-thread (for sequential processing and/or debugging and/or
reducing latency)
        if (FLAGS_disable_multi_thread)
            opWrapper.disableMultiThreading();
    }
    catch (const std::exception& e)
    {
```

```
            op::error(e.what(), __LINE__, __FUNCTION__, __FILE__);
        }
}

int tutorialApiCpp()
{
        try
        {
            op::opLog("Starting OpenPose demo...", op::Priority::High);
            const auto opTimer = op::getTimerInit();

            // OpenPose wrapper
            op::opLog("Configuring OpenPose...", op::Priority::High);
            op::Wrapper opWrapper;
            configureWrapper(opWrapper);

            // Start, run, and stop processing - exec() blocks this thread until OpenPose wrapper has finished
            op::opLog("Starting thread(s)...", op::Priority::High);
            opWrapper.exec();

            // Measuring total time
            op::printTime(opTimer, "OpenPose demo successfully finished. Total time: ", " seconds.", op::Priority::High);

            // Return
            return 0;
        }
        catch (const std::exception&)
        {
            return -1;
        }
}

int main(int argc, char *argv[])
{
```

```
    // Parsing command line flags
    gflags::ParseCommandLineFlags(&argc, &argv, true);

    // Running tutorialApiCpp
    return tutorialApiCpp();
}
```