

P D III

2D-LiDAR と YOLOv8 を用いた人追従システムの開発

指導教員 出村 公成 教授



金沢工業大学
工学部ロボティクス学科

金澤 祐典

令和 5 年度
2024 年 2 月 8 日

目次

第1章 序 論	1
1.1 はじめに	1
1.2 論文構成	2
第2章 従来研究	3
2.1 2D-LiDAR と YOLOv5 を用いた手法	3
2.2 2D-LiDAR の距離データをクラスタリングする手法	5
2.3 FCN と 2D-LiDAR を用いた手法	7
2.4 AOA タグと 2D-LiDAR を組み合わせた手法	9
2.5 従来研究における課題と本プロジェクトの位置づけ	12
第3章 提案手法	13
3.1 概要	13
3.2 要求仕様	13
3.3 システム構成	14
3.4 ソフトウェア構成	15
3.5 データセットの作成	16
3.6 YOLOv8 による学習	18
3.7 追従目標の特定	19
3.8 ロボット台車の制御	20
第4章 実験	21
4.1 実験目的	21
4.2 実験方法	21
4.2.1 実験機材	22
4.2.2 追従実験	23
4.2.3 最大追従速度実験	26

4.3 実験結果	27
4.3.1 追従実験	27
4.3.2 最大追従速度実験	28
4.4 考察	29
第5章 結論	30
5.1 まとめ	30
5.2 今後の課題	30
謝辞	31
参考文献	32
付録	34

図目次

2.1	Image of centroid (著者 [5] から転載)	4
2.2	Data Acquisition Environment (著者 [5] から転載)	4
2.3	Human recognition (著者 [7] から転載)	5
2.4	LIDAR data processing (著者 [7] から転載)	6
2.5	Kitchen scenario (著者 [7] から転載)	6
2.6	Architecture of the CNN used by PeTra (著者 [8] から転載)	7
2.7	Robotics mobile lab plan (left), Orbi-One robot (center), and KIO RTLS anchors (right). (著者 [8] から転載)	8
2.8	Comparison image of PeTra and LD by Rviz (著者 [8] から転載)	8
2.9	Person following in environments with obstacles (著者 [10] から転載)	10
2.10	The architecture of our person-following system (著者 [10] から転載)	10
2.11	Human legs detection and human tracking (著者 [10] から転載)	10
2.12	Kalman filter tracking in occluded environments (著者 [10] から転載)	11
2.13	Following trajectory in environments with obstacles (著者 [10] から転載)	11
3.1	System configuration chart	14
3.2	Software configuration diagram	15
3.3	Example of an overhead view image	17
3.4	Example data sets	17
3.5	Training results	18
3.6	Example of inference results with YOLOv8 using learned weights	18
3.7	Multiple leg sections detected on both legs	19
3.8	Target identification methods	19
4.1	Happy Edu	22
4.2	Image of tracking experiment environment (FMT Laboratory Room 206)	24
4.3	Image of tracking experiment environment (FMT Laboratory Room 326)	24

4.4	Straight road	25
4.5	Curved road	25
4.6	Right angle road	25
4.7	Maximum tracking speed experiment environment image (FMT Laboratory Room 326)	26
4.8	Image of maximum tracking speed experiment	26
4.9	Tracking experiment (Real view)	27
4.10	Tracking experiment (Internal view)	27
4.11	Tracking speed graph	28

表 目 次

2.1	Success rate of emergency stops on each road (著者 [5] から転載)	3
2.2	THE EFFECT OF TRAJECTORY AUGMENTATION (著者 [7] から転載)	6
2.3	Mean error and standard deviation[m] at each location (著者 [8] を改変)	8
2.4	Comparison of performance between FMM-DWA algorithm and MPEPC algorithm (著者 [10] から転載)	11
4.1	ASUS ROG Strix G16 specification	22
4.2	TurtleBot3 Big Wheel specification	23
4.3	UTM30-LX specification	23
4.4	Success rate of traking in each road	27
4.5	Maximum tracking speed experimental result	28

第1章

序 論

1.1 はじめに

近年の日本において、少子高齢社会による人手不足が課題となっている。2023年の65歳以上の人口は3623万人であり、総人口に占める65歳以上の割合(以下、高齢化率)は29.1[%]と過去最高である[1]。また、2070年での高齢化率は38.7[%]に達し、2.6人に1人が65歳以上であると推計されている[2]。加速する少子高齢化により、就業者不足の問題が深刻化しており、解決策の1つとしてロボットによる作業の自動化やサポートの導入が増えている。建設業では、2D-LiDARを用いた自動追従台車である「かもーん」が建設現場で導入されており、運用実績を上げ続けている[3]。また、製造業では2D-LiDARを用いた協働運搬ロボットである「サウザー」が実用化されており、「自動追従走行機能」によって運搬業務をサウサーで自動化しており、製造業界だけでなく空港や市役所などの公共環境における導入例があり、世界各地で約400台の販売実績がある[4]。以上のことから、2D-LiDARを用いた人追従ロボットへの需要と期待は増加し続けていることがわかる。

2D-LiDARを用いた人追従ロボットに関する手法には、深層学習を用いる手法[5][6][7][8]、背景減算によって人の両脚部分を検出する手法[9]、AOAタグを2D-LiDARと組み合わせる手法[10]などがある。これらの手法では、主に距離データをもとに人の両脚部分を検出するが、雑多な環境下では追従性能が低下する可能性がある。また、実験環境に椅子や机などのオブジェクトがなく、広域な経路での実験により追従性能を検証していることから、雑多な環境下での追従性能が評価されていない。2D-LiDARから提供される距離データでは、椅子や机などの脚部分が人の両脚部分と類似しているため、誤検出してしまった課題があり、これに伴い追従速度が低下する課題もある。

本プロジェクトでは、屋内環境による雑多な環境下で追従でき、ロボットの最大直進速度で追従できる人追従システムを開発する。

1.2 論文構成

本レポートの構成について述べる。第2章では、これまでの2D-LiDARを用いた人追従に関する従来研究について述べる。第3章では、本プロジェクトでの提案手法を述べる。第4章では、提案手法による人追従能力の検証結果について述べる。第5章では本プロジェクトをまとめ、結論及び今後の課題について述べる。

第2章

従来研究

2.1 2D-LiDARとYOLOv5を用いた手法

飯田一成らのプロジェクト [5] では、リアルタイム物体検出アルゴリズムである YOLOv5 を用いて、2D-LiDAR の距離データから人の脚部を検出している。

提案手法は、Fig. 2.1 のように 2D-LiDAR の距離データを画像化し、学習した YOLOv5 の物体検出により画像から人の脚部を検出する。人以外の物体を誤検出する場合があるため、検出する範囲を固定位置に設定している。また、衝突回避機能を実装することで、安全性を考慮した人追従機能を提案している。

実験方法は、追従実験と衝突回避実験がある。追従実験では Fig. 2.2 のような 3 つの経路を設定し、1 回のみの試行である。衝突回避実験では、人追従中に追従対象者とロボットとの間に障害物を設置し、停止したかを 3 つの状況に分けてそれぞれ 10 回ずつ試行している。実験結果は、追従実験では 3 つの経路において 20[m] の人追従ができていたが、雑多な環境下では追従中に停止することがあった。衝突回避実験では Table 2.1 に示すように、すべての試行において衝突回避ができていた。

Table 2.1: Success rate of emergency stops on each road (著者 [5] から転載)

	Distance between robot and obstacle		
	0.1[m]	0.2[m]	0.3[m]
Straight road	100[%]	100[%]	100[%]
Curved road	100[%]	100[%]	100[%]
Miscellaneous road	100[%]	100[%]	100[%]

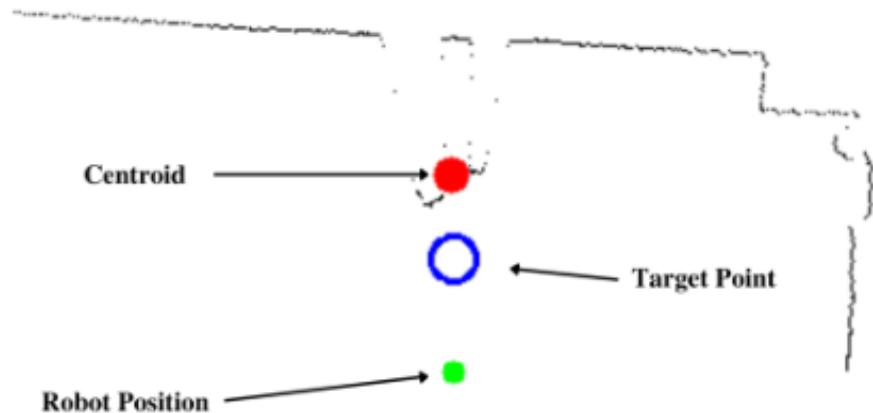


Fig. 2.1: Image of centroid (著者 [5] から転載)

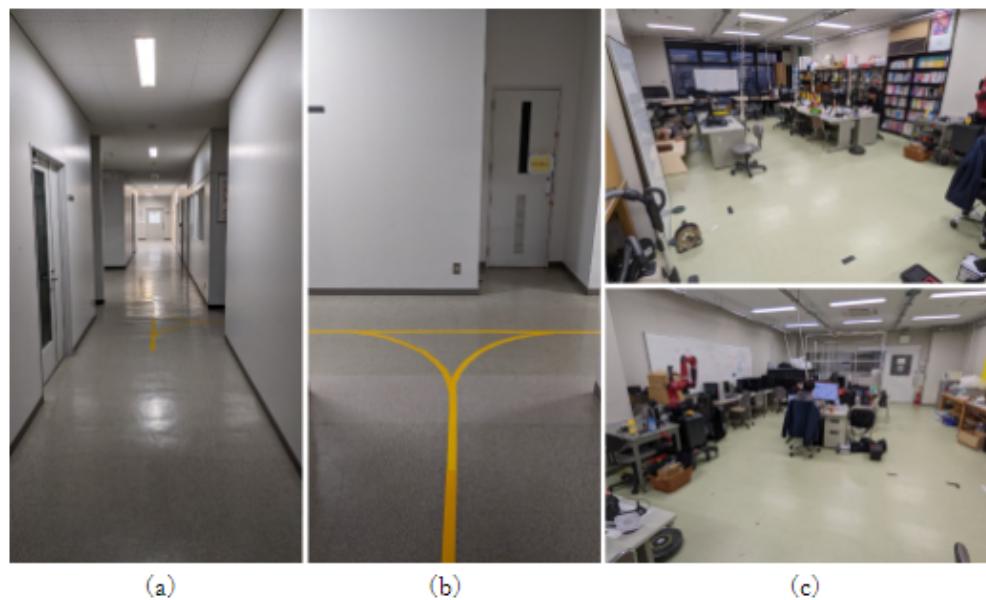


Fig. 2.2: Data Acquisition Environment (著者 [5] から転載)

2.2 2D-LiDARの距離データをクラスタリングする手法

Fei Luo らの研究 [7] では、キッチン内を想定し、人が歩行した軌跡のクラスタリングを行っている。

提案手法は、Fig. 2.4 が処理の全体像である。Fig. 2.4 (a) では、2D-LiDAR から提供される距離データをプロットしている。Fig. 2.4(b) では、密度ベースのクラスタリングアルゴリズムである DBSCAN (Density-Based Spatial Clustering of Applications with Noise) を用いて距離データをクラスタリングする。2.4(c) では、Fig. 2.3 のように定義した幾何学的特徴を用いてランダムフォレストにより、人間と非人間の 2 つに分類する。Fig. 2.4(c) では、カルマンフィルタを用いて人が移動した軌跡を生成し、ガウスノイズなどの軌跡拡張が行われ、LSTM (Long Short-Term Memory) と TCN (Temporal Convolutional Network) の両方に入力され、活動クラスに分類される。LSTM は、時間経過に伴って変化するデータを学習できる RNN(Recurrent Neural Network) の勾配消失問題を解消したものであり、TCN は時系列データに対して CNN (Convolutional Neural Network) を用いたものである。

実験方法は、学習した LSTM と TCN を用いて、Fig. 2.5 のような環境において 15 種類の歩行パターンを分類させ、LSTM と TCN の分類精度の評価を行う。実験結果は Table 2.2 のようになっており、LSTM より TCN のほうが分類精度が高いことが明らかになっている。

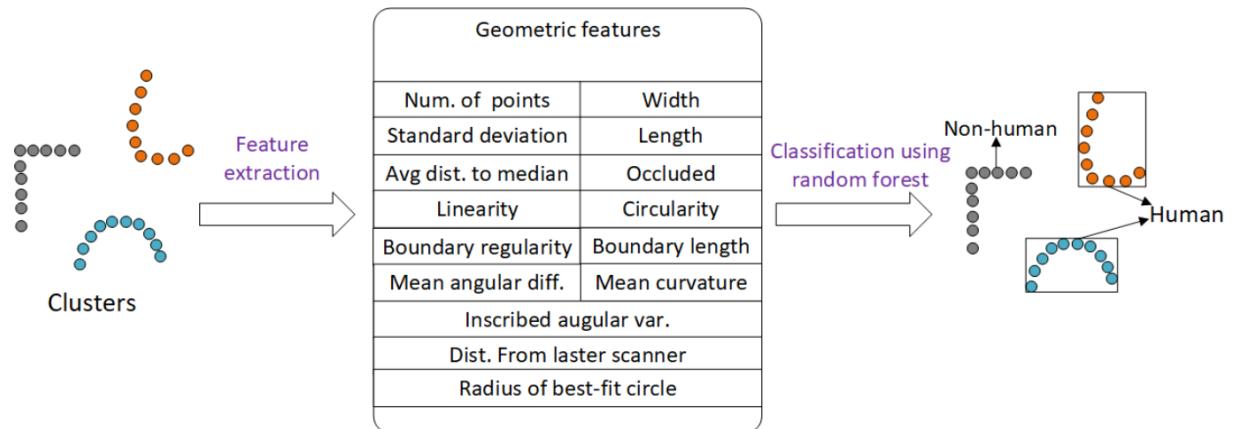
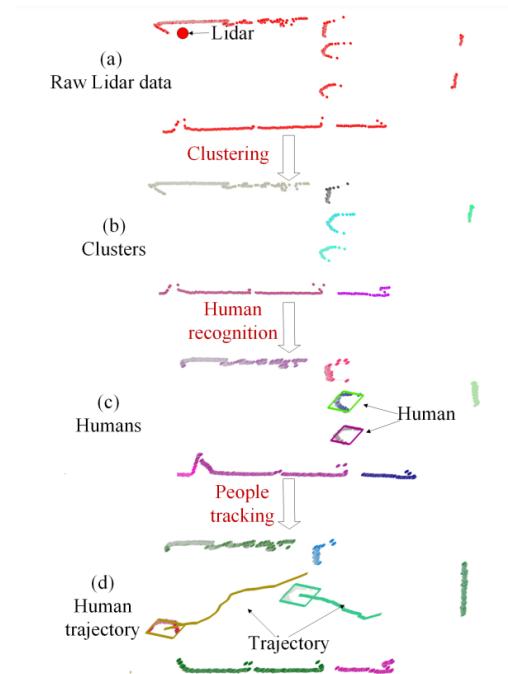


Fig. 2.3: Human recognition (著者 [7] から転載)

**Fig. 2.4:** LIDAR data processing (著者 [7] から転載)**Fig. 2.5:** Kitchen scenario (著者 [7] から転載)**Table 2.2:** THE EFFECT OF TRAJECTORY AUGMENTATION (著者 [7] から転載)

	OA	Recall	F1
TCN with trajectory augmentation	99.49%	99.53%	99.51%
TCN without trajectory augmentation	97.96%	97.93%	97.96%
LSTM with trajectory augmentation	99.39%	99.41%	99.39%
LSTM without trajectory augmentation	97.65%	97.79%	97.65%

2.3 FCNと2D-LiDARを用いた手法

Ángel Manuel Guerrero-Higueras らの研究 [8] では、FCN(Full Convolutional Neural Networks)を用いて、2D-LiDAR の距離データから人の両脚部分を検出することで人追従する「PeTra」を提案している。

提案手法は、2D-LiDAR からの距離データを画像化し、Fig. 2.6 に示すアーキテクチャを用いて、人の両脚部分をセグメンテーションすることにより、追従対象者を検出している。

実験方法は、Fig. 2.7 の左のような環境において、Location1、Location2、Location3 で行われる。また、Fig. 2.7 の右にあるタグを追従対象者が所持することにより、タグの位置を真値としている。PeTra の比較対象として、ROS(Robot Operationg System) が提供している LD(Leg Detector) を用いており、追従対象者が所持しているタグからの平均誤差と標準偏差 [m] を PeTra と LD で比較する。実験結果を、Fig. 2.8 と Table 2.3 に示す。Fig. 2.8 の黄色のマーカーは 2D-LiDAR の距離データであり、赤矢印はロボットの位置と向きであり、矢印の先がロボットの位置である。Fig. 2.8 の左が 2D-LiDAR の距離データのみの画像であり、中央は PeTra による推定結果が加えられた画像で、緑色のマーカーは人の中心、青色のマーカーは脚の位置を示している。右は LD による推定結果が加えられた画像である。赤いマーカーは脚の位置を示している。Fig. 2.8 から、LD より PeTra の方が脚のペアを正しく推定していることが分かる。また、Table 2.3 は Location1、Location2、Location3 における、PeTra と LD の追従対象者が所持しているタグからの平均誤差と標準偏差である。Table 2.3 から、LD より PeTra の方が誤差が 50% 少ないことが分かる。以上のことから、LD より精度が高い人追従ツールを実現していることが分かる。

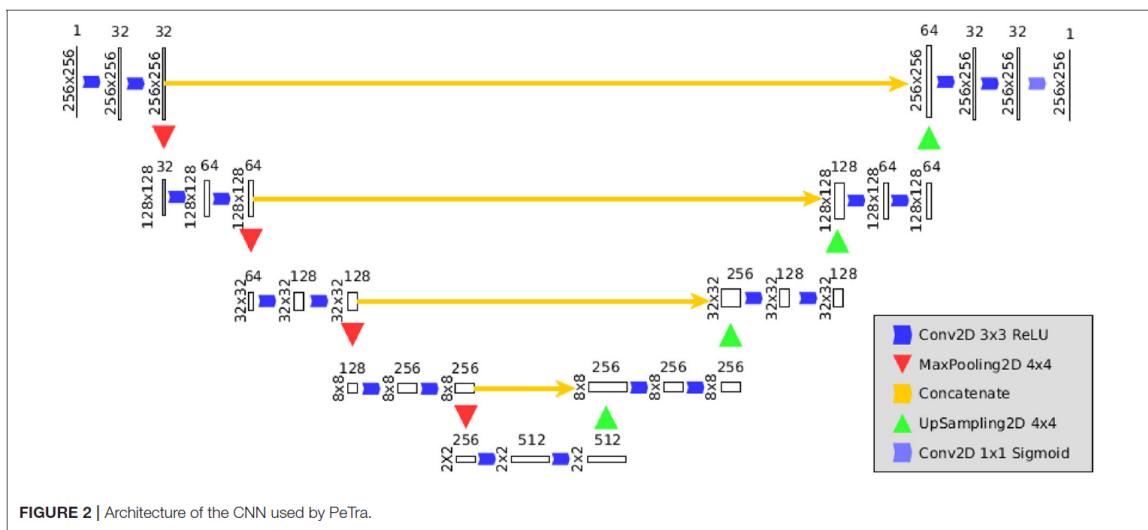


Fig. 2.6: Architecture of the CNN used by PeTra (著者 [8] から転載)

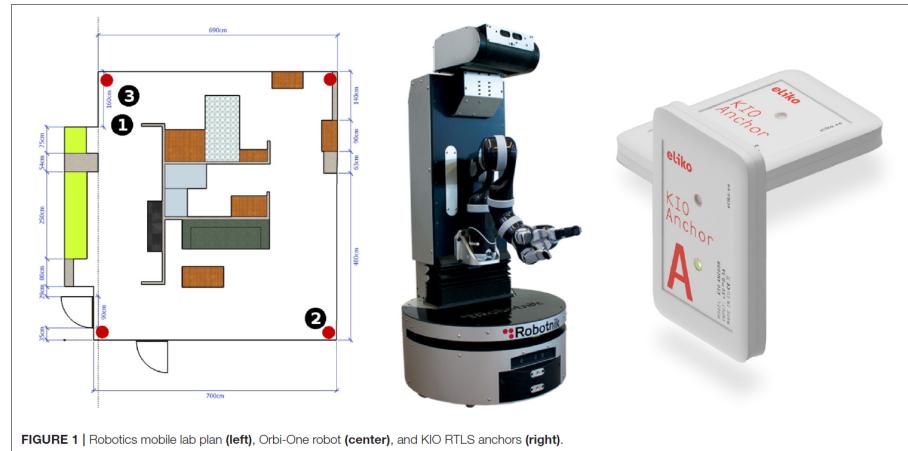


Fig. 2.7: Robotics mobile lab plan (left), Orbi-One robot (center), and KIO RTLS anchors (right). (著者 [8] から転載)



Fig. 2.8: Comparison image of PeTra and LD by Rviz (著者 [8] から転載)

Table 2.3: Mean error and standard deviation[m] at each location (著者 [8] を改変)

Method	Location 1	Location 2	Location3
PeTra	0.17(± 0.13)	0.43(± 0.22)	0.20(± 0.13)
LD	0.30(± 0.15)	0.75(± 0.28)	0.49(± 0.33)

2.4 AOA タグと 2D-LiDAR を組み合わせた手法

DAPING JIN らの研究 [10] では、Fig. 2.9 のような環境を想定し、AOA タグと 2D-LiDAR のデータから人を検出している。

提案手法は、システム全体が Fig. 2.10 のようになっており、2D-LiDAR と AOA タグのデータからカルマンフィルタを用いた追跡軌跡を統合することで人の位置を正確に追跡している。ロボットの制御では、Dynamic Window Approach を改良した FMM-DWA を用いて障害物に衝突しないロボット制御を実装している。AOA タグは、位置情報を送信するデバイスであり、追従対象者が AOA タグを所持し、ロボットが位置情報を受信している。2D-LiDAR による人の脚部検出では、2D-LiDAR のデータをクラスタリングし、各クラスタに対して、2D-LiDAR のデータ数や幅と長さなどの幾何学的特徴が生成され、幾何学的特徴に基づいて、Fig. 2.11 のように人の脚かその他に分類する。AOA タグの追跡軌跡、2D-LiDAR の追跡軌跡、カルマンフィルタによる追跡軌跡を組み合わせたものが Fig. 2.12 である。緑色の軌跡が AOA タグ、赤い軌跡は 2D-LiDAR、黒い軌跡はカルマンフィルタの軌跡である。AOA タグの追跡軌跡は、カルマンフィルタの追跡軌跡よりノイズが多いことがわかる。また、2D-LiDAR の追跡軌跡はカルマンフィルタより滑らかであると述べられている。2D-LiDAR での追従は、Fig. 2.12 の「Laser Tracking Failure」部分で追従ができなくなった。しかし、「Laser Tracking Recovery」の部分で再び追従対象を発見し追従を再開した。実験方法は、Fig. 2.9 のような障害物がいくつかある環境において、モデル予測制御である MPEPC と FMM-DWA によるロボット制御の比較である。実験結果を Table 2.4 に示す。ロボット台車の加速度では、FMM-DWA における加速度が $1[m/s^2]$ 未満である場合が 90% 以上であり、MPEPC における加速度が $1[m/s^2]$ 未満である場合が 68% であった。また、ロボット台車の旋回半径が MPEPC では、1m より小さい制御コマンドは全体の 8% であり、FMM-DWA の 2.6 倍である。さらに、人の軌道とロボットの軌道の追跡率は、FMM-DWA が 94% であるのに対し、MPEPC は 95% であり、明確な差はなかった。以上のことから、DAPING JIN らの研究では AOA タグと 2D-LiDAR のデータを組み合わせた追従対象者の検出と、FMM-DWA によるロボット台車の制御により追従性能の高い人追従システムを実現している。

今後の課題として、提案した人追従システムでは、動的な障害物の動きを予測することができないため、動的障害物予測を考慮した、より高速で滑らかな人追従システムを実現するとしている。

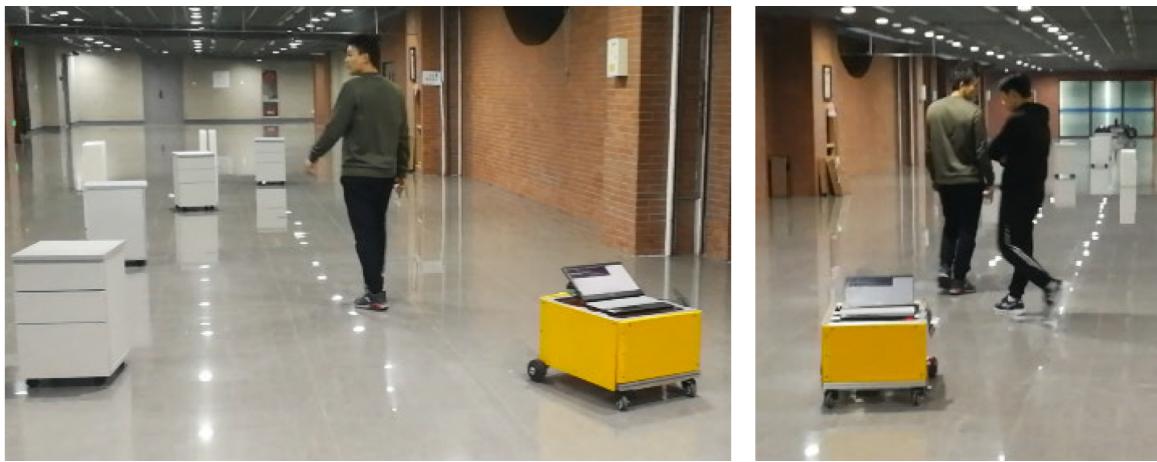


Fig. 2.9: Person following in environments with obstacles (著者 [10] から転載)

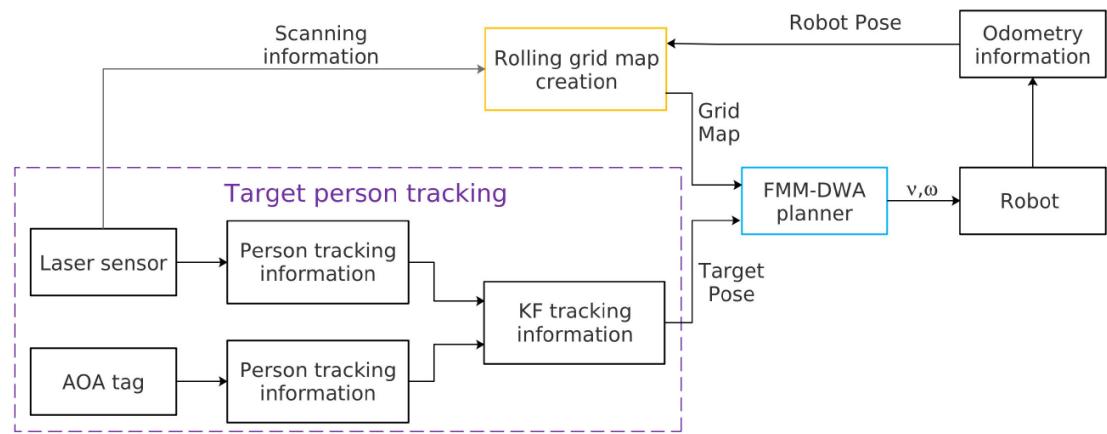


Fig. 2.10: The architecture of our person-following system (著者 [10] から転載)

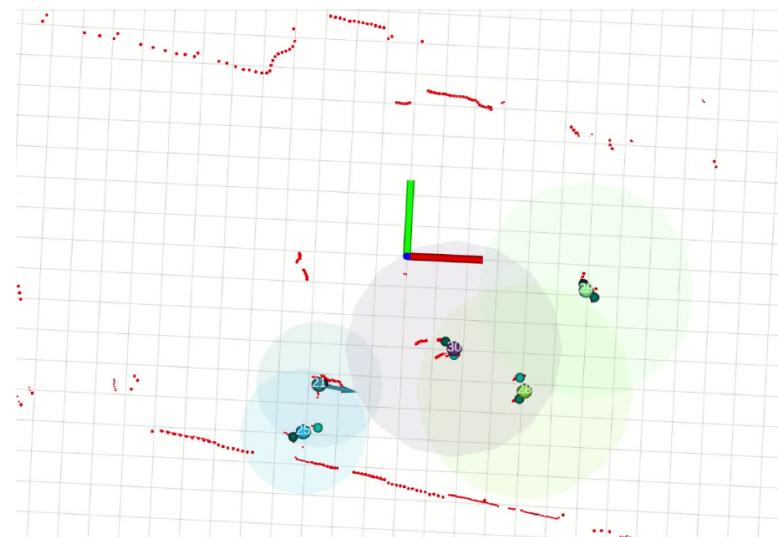


Fig. 2.11: Human legs detection and human tracking (著者 [10] から転載)

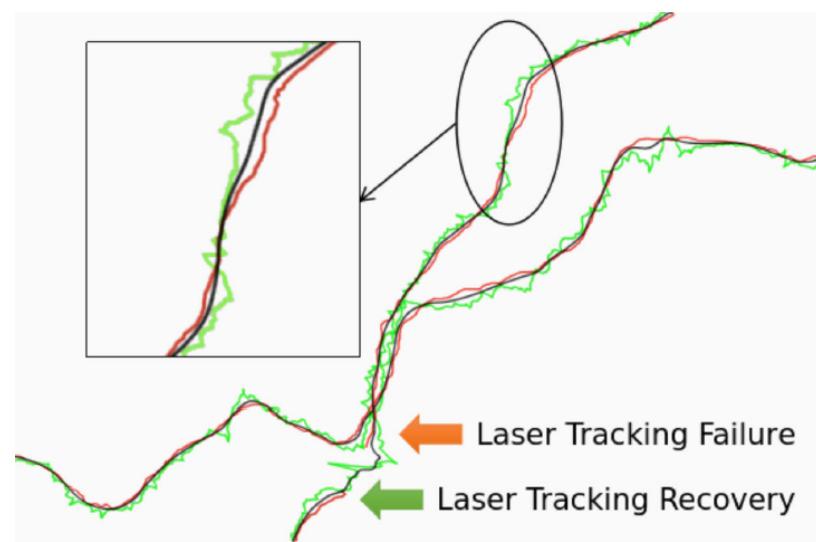


Fig. 2.12: Kalman filter tracking in occluded environments (著者 [10] から転載)

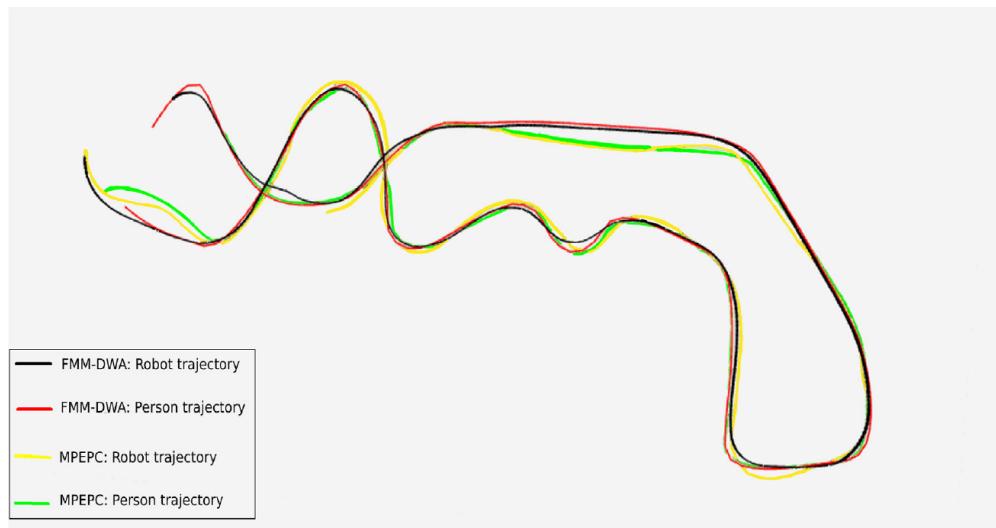


Fig. 2.13: Following trajectory in environments with obstacles (著者 [10] から転載)

Table 2.4: Comparison of performance between FMM-DWA algorithm and MPEPC algorithm (著者 [10] から転載)

Method	$\alpha > 1$	$r < 1$	S_r/S_h
FMM-DWA	8%	3%	94%
MPEPC	32%	8%	95%

2.5 従来研究における課題と本プロジェクトの位置づけ

従来研究の課題として、2D-LiDAR と YOLOv5 を用いた手法では、雑多な環境下において人追従中に周囲の環境に影響され、特定の追従目標を追従できない点が挙げられる。また、検出範囲が固定であることから、人間がロボットの検出範囲に合わせながら歩行しなければならず、人の歩行速度の低下に伴いロボットの追従速度も低下している点が課題として挙げられる。

2D-LiDAR の距離データをクラスタリングする手法では、人の脚部が幾何学的特徴からの検出であり、脚部に類似している物体が乱雑に配置されていた場合、人の脚部ではない物体が誤検出されてしまう課題が考えられる。

FCN と 2D-LiDAR を用いた手法と AOA タグと 2D-LiDAR を用いた手法では、物体が少ない環境下での実験であるため、雑多な環境下での追従性能が不明である。また、AOA タグと 2D-LiDAR を用いた手法では、人の脚部を幾何学的特徴を用いて検出しているため、雑多な環境下での追従性能の低下が考えられる。さらに、AOA タグの紛失などの課題が考えられる。

本プロジェクトでは、雑多な環境下において人追従し、ロボットの最大直進速度で追従可能な人追従システムを提案する。

第3章

提案手法

3.1 概要

本プロジェクトでは、雑多な環境下で人追従でき、ロボットの最大直進速度で追従するため、2D-LiDAR の距離データとリアルタイム物体検出アルゴリズムである YOLOv8 を用いた人追従システムを開発する。本プロジェクトで提案する人追従システムは、追従対象者のみがいる雑多な環境を想定している。雑多な環境では、人の脚部と形状が類似している椅子やポールなどの物体をランダムに多数設置している。2D-LiDAR から提供される距離データを俯瞰画像に変換し、学習した YOLOv8 と俯瞰画像を用いて追従対象者の検出をする。追従対象者の検出のみでは、周囲の物体を両脚部と誤検出する可能性があるため、追従目標の特定処理をシステムに組み込むことで、正確に追従対象者を追従することを実現する。

3.2 要求仕様

本プロジェクトでは、雑多な環境下で人追従でき、ロボットの最大直進速度で追従できる人追従システムの開発を目的としているため、以下の構成で要求仕様を設定する。使用的するロボットは、2023年に開催された RoboCup 2023 で開発したロボットである Happy Edu を使用する。Happy Edu のロボット台車は、ROBOTIS 社の TurtleBot3 Big Wheel であり、ロボット台車の前方に 2D-LiDAR を搭載している。2D-LiDAR は、北陽電機株式会社の UTM30-LX を使用しており、ロボットに搭載する PC は、NVIDIA GeForce RTX 4070 8GB を搭載しているノート PC を選定した。

以上の構成で以下の要求仕様を設定する。

1. 2D-LiDAR のデータで人追従ができる。
2. 雜多な状態の空間でも人追従ができる。
3. 0.5[m/s] 以下の歩行速度で追従する。

3.3 システム構成

本プロジェクトでは、Fig. 3.1 のような人追従システムを提案する。2D-LiDAR から提供される距離データを俯瞰画像に変換し、YOLOv8 のリアルタイム物体検出モデルにより人の両脚部分を検出する。YOLOv8 による人の両脚部分の検出は、別の物体を誤検出することがあるため、追従目標の特定処理を実装している。追従目標の特定後、ロボットが追従すべき目標座標を生成し、ロボットから目標座標までの距離と角度の偏差を収束させるため、PID 制御によりロボット台車を制御する。

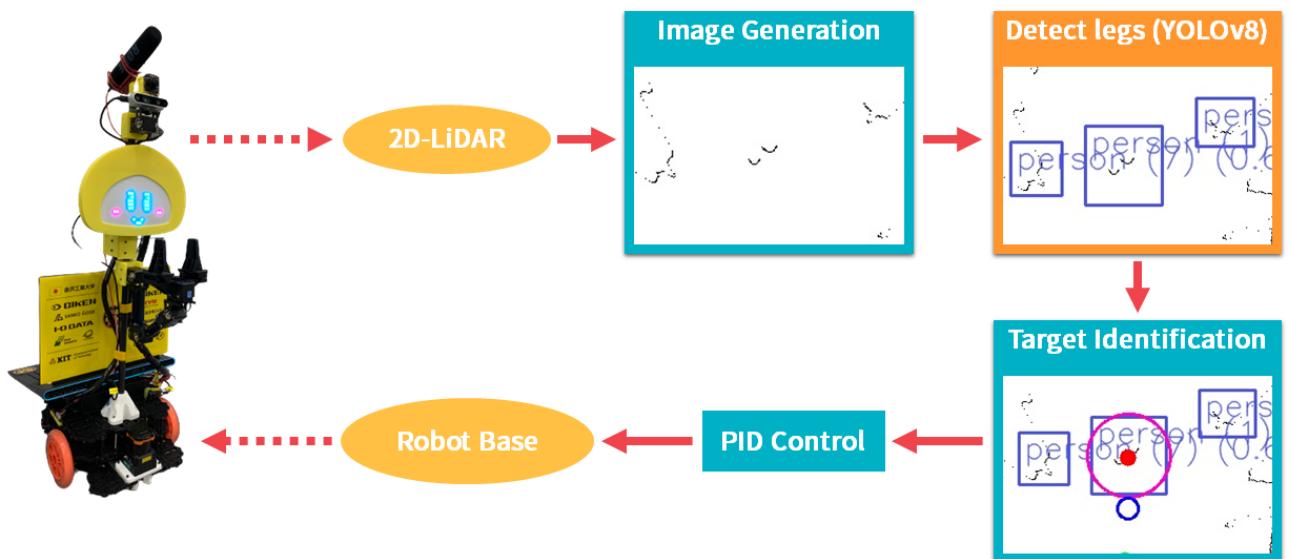


Fig. 3.1: System configuration chart

3.4 ソフトウェア構成

ロボット用ノートPC内のソフトウェア構成をFig. 3.2に示す。ロボット用ノートPCのオペレーティングシステムにはUbuntu 22.04 LTSを使用し、ミドルウェアにはRobot Operating System 2(以下、ROS2)を用いている。ロボットのソフトウェア開発には、主にPython言語を使用し、YOLOv8のROS2パッケージにはyolov8_rosパッケージを用いている。今回開発した人追従システムは、ROS2パッケージになっており、Fig. 3.2のfollow_me Package上の構成となっている。follow_me Package上には、laser_to_image Node、person_detector Node、base_controller Nodeがあり、ソースコードの可読性を上げるためにこのような構成にしている。また、ノード(Node)とは、ROS2でソフトウェア開発する上での最小単位である。

各ノードの処理内容は以下の通りである。

- YOLOv8 Nodes: 俯瞰画像から人の両脚部分を検出する。
- laser_to_image Node: 2D-LiDARの距離データを俯瞰画像に変換する。
- person_detector Node: 推論結果とともに、追従目標を特定し目標座標を生成する。
- base_controller Node: 目標座標までの距離と角度の偏差をPID制御により収束させる。

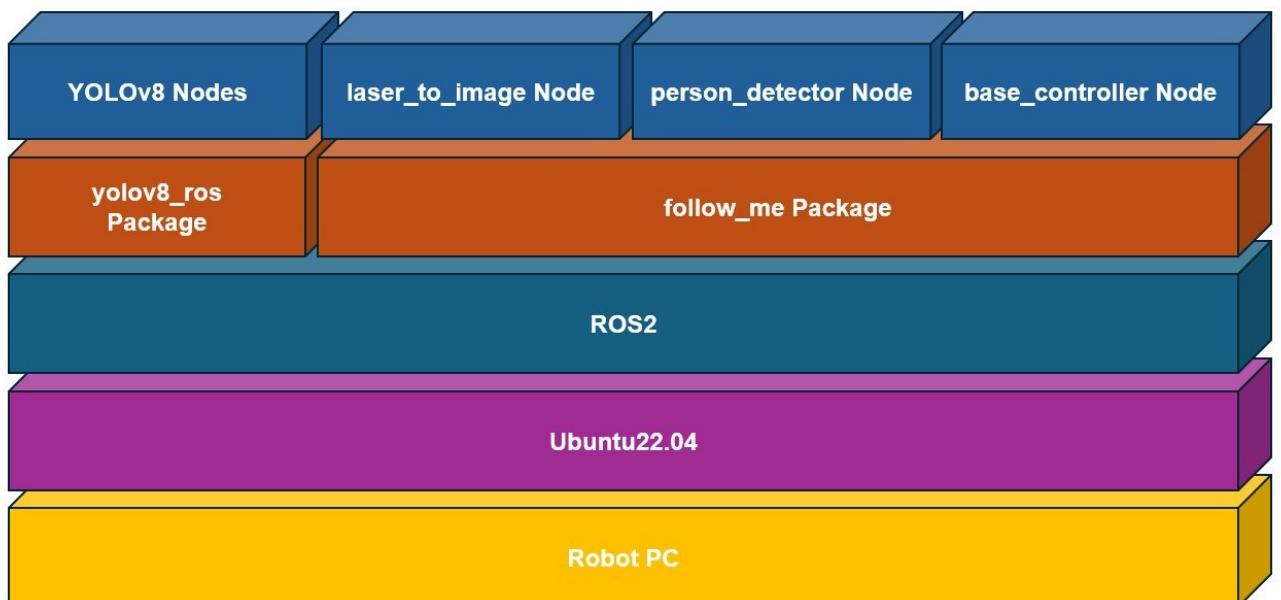


Fig. 3.2: Software configuration diagram

3.5 データセットの作成

データセットを作成するため、2D-LiDAR の距離データを収集する。データ収集時の環境は、ロボットが静止した状態においてロボットの前方で2人の人が歩行する。データ収集中の2人の歩行パターンを以下に示す。方向については、ロボットからみた方向である。

- 左から右方向に歩行する。
- 右から左方向に歩行する。
- 手前から奥方向に歩行する。
- 奥から手前方向に歩行する。
- 右手前から左奥方向に歩行する。
- 左奥から右手前方向に歩行する。
- 左手前から右奥方向に歩行する。
- 右奥から左手前方向に歩行する。
- 人同士が交差する。

また、歩行時の歩幅については以下に示す。

- 普段通りの歩幅で歩行する。
- 大股(歩幅、足4つ分程度)で歩行する。
- 小股(歩幅、足2つ分程度)で歩行する。

以上を5分間でランダムに行い、2D-LiDAR の距離データをトピック通信によってトピックとして配布し、ROS2 Bag を用いて保存する。トピック通信とは、ROS2における通信手法のことであり、トピック通信によって送受信されるデータがトピックである。また、ROS2 Bag は、ROS2におけるトピックの保存機能である。

ROS2 Bag のデータから、2D-LiDAR の距離データを Fig. 3.3 のような俯瞰画像へ変換する。変換方法を以下に示す。

1. 白画像を生成する。
2. 2D-LiDAR からの距離データと1ステップあたりの角度を取得する。

3. 距離データと1ステップあたりの角度から、距離データを画像の左上端を原点としたXY座標に変換する。

4. 変換したXY座標から、点を白画像に黒点でプロットする。

以上 の方法で、2D-LiDARの距離データから12032枚の画像データを生成した。

画像データからデータセットを作成する。人の両脚部分をpersonクラスとしてアノテーションを行った。アノテーションは、labelImgを用いて手動で行った。labelImgとは、GUIでアノテーションが可能なアノテーションツールであり、ファイルの出力形式としてYOLO形式がある。ファイル形式はYOLO形式で出力した。また、データセットには、画像の回転処理、モザイク処理、2枚の画像を合成し、新しく1枚の画像を生成するMix Up処理をすることでデータ拡張を行い、計21061枚のデータセットを作成した。Fig. 3.4にデータセットの例を示す。

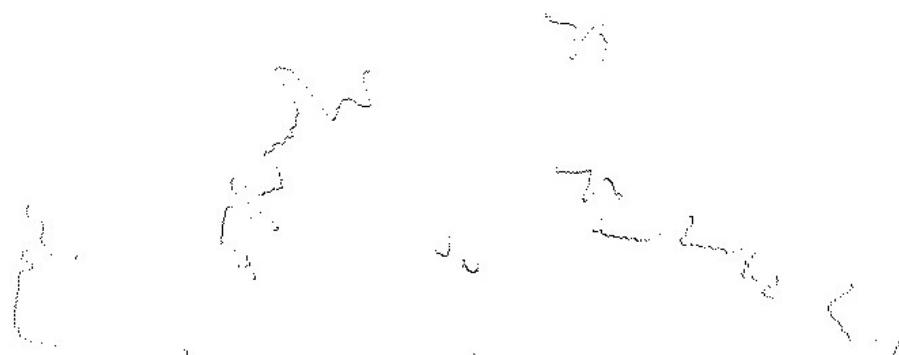


Fig. 3.3: Example of an overhead view image

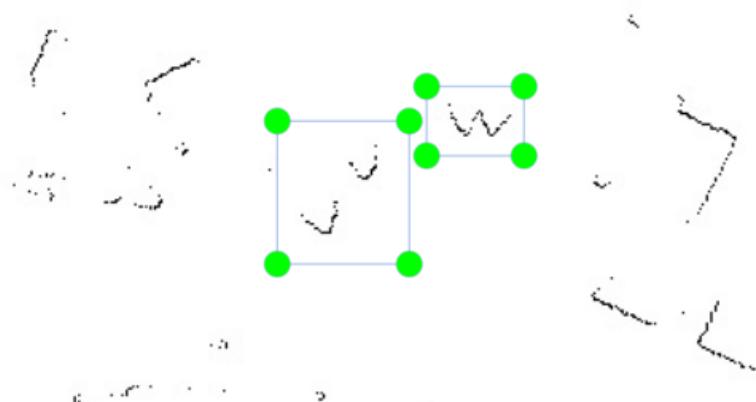


Fig. 3.4: Example data sets

3.6 YOLOv8による学習

作成したデータセットとYOLOv8を用いて人の両脚検出器を作成する。学習には、リアルタイム物体検出アルゴリズムであるYOLO (You Only Look Once)を使用し、学習モデルの初期重みは、ロボットに搭載するノートPCの性能が高いため、最もパラメータ数の多いYOLOv8xを選定した。学習するPCは、NVIDIA GeForce RTX 4090 16GBを搭載しているPCを使用し、バッチサイズは12、エポック数は500で学習を行った。過学習を防ぐため、過学習が発生する直前または発生したらすぐに学習を終了させる処理を100エポック以降で設定した。学習結果は、Fig. 3.5とFig. 3.6のようになっており、推論結果が90[%]を超えていることが分かる。

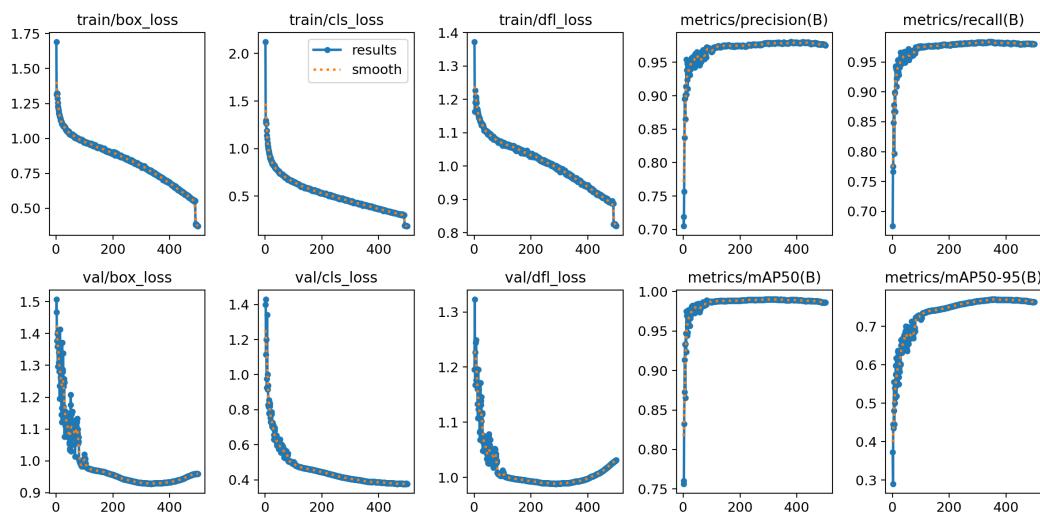


Fig. 3.5: Training results

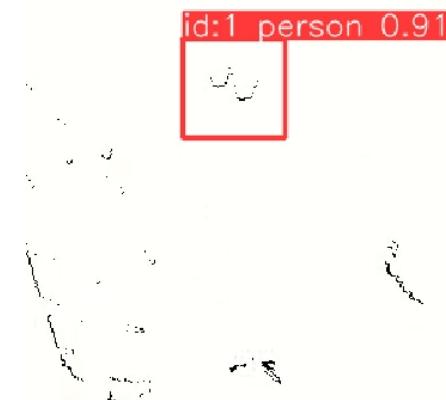


Fig. 3.6: Example of inference results with YOLOv8 using learned weights

3.7 追従目標の特定

YOLOv8による推論結果を用いて追従目標の特定をする。前節で作成した両脚検出器では、Fig. 3.7のように複数検出される場合があり、追従目標を特定する必要がある。YOLOv8には、Byte Trackなどの追跡アルゴリズムが標準機能で搭載されており、検出した物体にIDを付与することができる。しかし、1度検出を外れ、再度同じ物体が検出されてもIDが変わってしまうという問題がある。そこで、動的検出範囲を実装することで追従目標を特定する。

本プロジェクトで実装した追従目標の特定方法をFig. 3.8に示す。1フレーム前における追従目標のバウンディングボックスを中心とした、半径0.5[m]の円型範囲を現在のフレームに設定し、範囲内にバウンディングボックスの中心があればそれを追従対象とする。

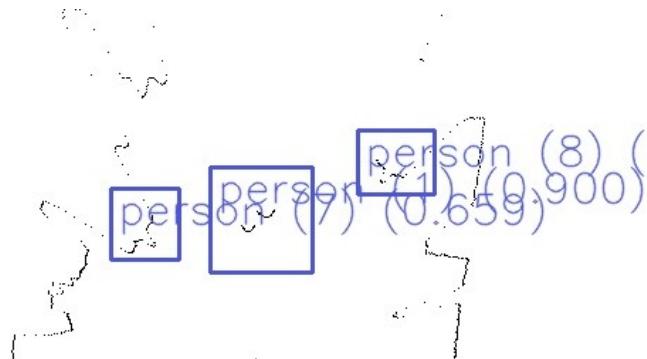


Fig. 3.7: Multiple leg sections detected on both legs

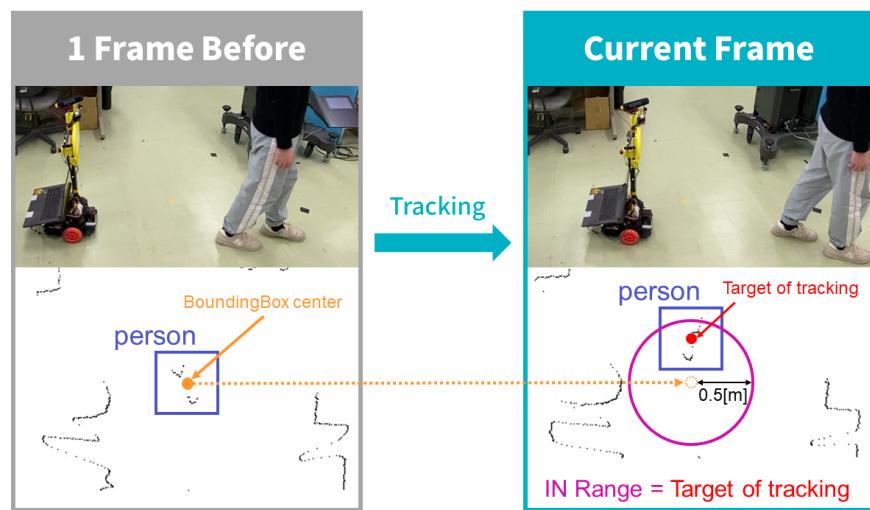


Fig. 3.8: Target identification methods

3.8 ロボット台車の制御

追従目標の特定により、追従目標のバウンディングボックスの中心から後方に定数で目標座標を生成し、ロボットから目標座標までの距離と角度の偏差を収束させるため PID 制御を実装した。

時刻 t での、ロボットの座標から目標座標までの角度の偏差を $\theta(t)$ とし、ロボット台車のエンコーダから取得できる旋回速度を $\omega(t)$ とする。また、追従目標への角度の偏差が 3.0[deg] 未満である場合に積分制御を開始する時刻を t_1 としたとき、ロボット台車の旋回速度の制御量 $u_{angle1}(t)$ は以下のようになる。

$$u_{angle1}(t) = K_{AP} \cdot \theta(t) + K_{AI} \cdot \int_{t_1}^{t-t_1} \theta(\tau) d\tau + K_{AD} \cdot \left\{ \frac{d}{dt} \theta(t) - \omega(t) \right\} \quad (3.1)$$

また、追従目標への角度の偏差が 3.0[deg] 以上である場合は PD 制御に切り替える。PD 制御時の制御量 $u_{angle2}(t)$ は以下のようになる。

$$u_{angle2}(t) = K_{AP} \cdot \theta(t) + K_{AD} \cdot \left\{ \frac{d}{dt} \theta(t) - \omega(t) \right\} \quad (3.2)$$

K_{AP} 、 K_{AI} 、 K_{AD} は PID ゲインであり、 K_{AP} は 0.005、 K_{AI} は 0.0002、 K_{AD} は 0.0009 で設定している。PID ゲインの設定値は、P ゲイン、D ゲイン、I ゲインの順に設定した。P ゲインでは、ロボット台車が左右に振動する 1 つ前の値を設定している。D ゲインでは、P 制御による小さい振動が発生しなくなる値を設定している。I ゲインでは、ロボット台車の直進制御をしない状態において、追従目標への角度の偏差の定常偏差が 0.1[deg] 未満になる値を設定している。(3.1) 式の第 1 項は、ロボットの座標から目標座標までの角度の偏差を比例制御している。第 2 項は、ロボットが追従対象者の方向を定常偏差なく向くため、目標座標までの角度の偏差を積分制御している。第 3 項では、実機での制御を考慮し、過多な制御量を微分制御により調整している。また、(3.2) 式のように PD 制御に切り替えるのは、角度の偏差が大きくなった場合に、積分値が時間経過に伴い過多になりすぎることで、ロボットが左右に振動してしまうからである。

また、追従目標への距離の偏差を $l(t)$ としたとき、ロボット台車の直進速度の制御量 $u_{linear}(t)$ は以下のようになる。

$$u_{linear}(t) = K_{LP} \cdot l(t) \quad (3.3)$$

K_{LP} は P ゲインである。(3.3) 式は、追従目標への距離の偏差を比例制御しており、 K_{LP} は 0.3 で設定している。P ゲインは、ロボット台車が前後方向に振動する 1 つ前の値を設定している。

第4章

実験

4.1 実験目的

本プロジェクトでは、雑多な環境下で人追従でき、ロボットの最大直進速度で追従できる人追従システムの開発を目的としている。これに伴った実験の目的は、雑多な環境下での人追従の精度とロボットの最大直進速度での人追従性能の2つの検証をすることである。以上のことから、追従実験と最大追従速度実験により、開発した人追従システムの性能を検証する。

4.2 実験方法

実験では、雑多な環境下での追従性能を検証する追従実験と、ロボットの最大直進速度での追従を検証する最大追従速度実験をする。実験中は、人は追従対象の1人のみとする。

要求仕様(2)を検証するため、追従実験では直線経路、曲線経路、直角経路をそれぞれ10回実験し、成功率を算出する。追従の成功率が各経路において90%以上であった場合に要求仕様(2)を満たしたものとする。また、要求仕様(3)を検証するため、10[m]以上の直線経路にて最大追従速度実験をする。0.1[m/s]から、0.1ずつ速度を上昇させ、追従できなくなった1つ前の速度を最大追従速度とする。Turtlebot3 Big Wheelの最大直進速度が0.5[m/s]であるため、開発した人追従システムの最大追従速度が0.5[m/s]であった場合に要求仕様(3)は満たされる。以上の実験は、2D-LiDARのデータを用いた実験であるため、要求仕様(2)、(3)が満たされたら、要求仕様(1)も満たされたものとする。

4.2.1 実験機材

本プロジェクトでは、2023年に開催されたRoboCup2023で開発したHappy Eduを使用する。Happy Eduの全体像をFig. 4.1に示す。Happy Eduに搭載するノートPCは、ASUSのROG Strix G16を用いている。ロボット台車には、ROBOTIS社の二輪差動駆動台車であるTurtleBot3 Big Wheelを用いてる。2D-LiDARには、北陽電機株式会社のUTM30-LXを用いている。ノートPC、ロボット台車、2D-LiDARの仕様は、それぞれTable 4.1、Table 4.2、Table 4.3に示す。

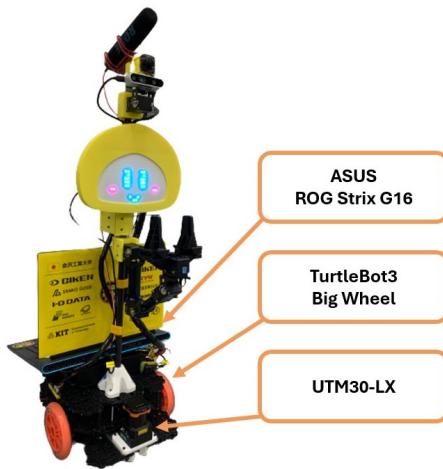


Fig. 4.1: Happy Edu

Table 4.1: ASUS ROG Strix G16 specification

Size	264.0 x 354.0 x 22.6[mm]
Weight	2500[g]
Model	G614JI-I7R4070
Form factor	Laptop
Resolution	1920 x 1200 [pixel]
CPU	Intel Corei7 13650HX
RAM	32[GB]
GPU	NVIDIA GeForce RTX 4070
VRAM	8[GB]

Table 4.2: TurtleBot3 Big Wheel specification

Maximum straight speed	0.5[m/s]
Maximum rotation speed	3.14[rad/s]
Maximum payload	30[kg]
Size	281 x 306 x 170.3 [mm]
Actuator	XM430-W210
Supply input terminal	3.3[V] / 800[mA], 5[V] / 4[A], 12[V] / 1[A]
Battery	Lithium polymer 11.1[V] 1800[mAh] / 19.98[Wh] 5[C]

Table 4.3: UTM30-LX specification

Power source	12[V]DC
Current consumption	700[mA]
Scanning range	0.1 to 30[m]
Scanning angle	270 [deg]
Angular Resolution	Step angle: approx.0.25[deg](360[deg]/1,440[steps])
Scanning time	25[msec]/scan
Weight	210[g]

4.2.2 追従実験

要求仕様(2)を検証するため、Fig. 4.2 と Fig. 4.3 のような雑多な環境を用意し、追従の成否を実験する。雑多な環境の用意では、人の脚部と類似している椅子やポールなどの円柱状の物体を多く設置している。

直線経路、曲線経路、直角経路のイメージをそれぞれ Fig. 4.4、Fig. 4.5、Fig. 4.6 に示す。直線経路、曲線経路、直角経路においてそれぞれ 10 回ずつ試行し、各経路での追従成功率を算出する。追従が成功した回数を $x_{success}$ とした時の追従成功率 $success\ rate$ は以下のようになる。

$$success\ rate = x_{success}/10 \quad (4.1)$$

(4.1) 式を用いて各経路での成功率を算出し、各経路での追従成功率が 90[%] 以上であれば要求仕様(2)を満たしたこととする。

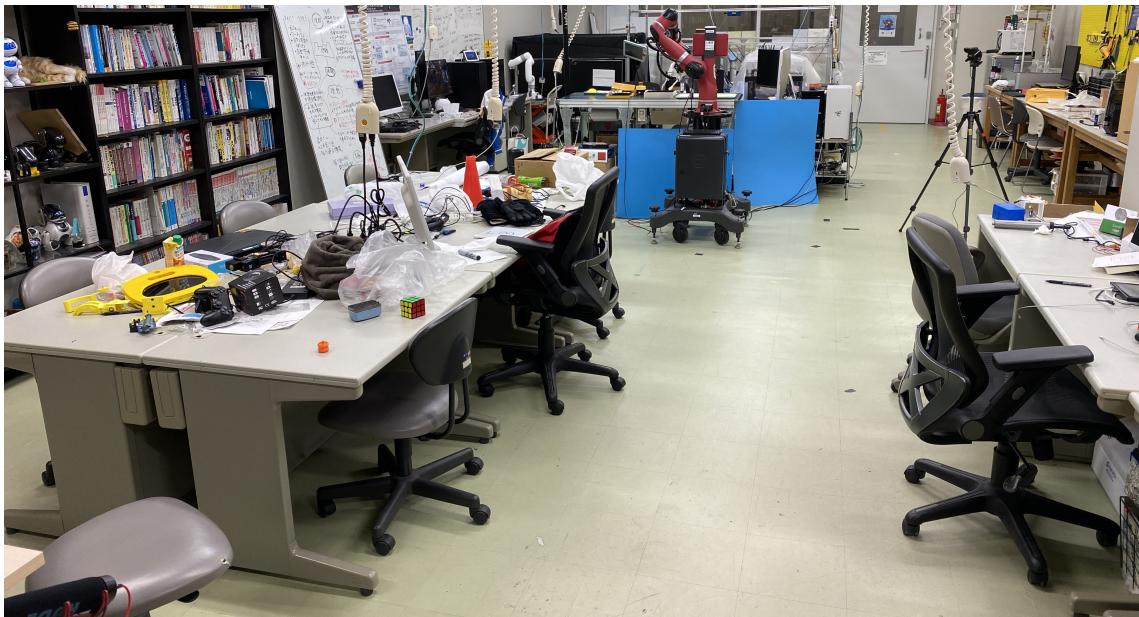


Fig. 4.2: Image of tracking experiment environment (FMT Laboratory Room 206)

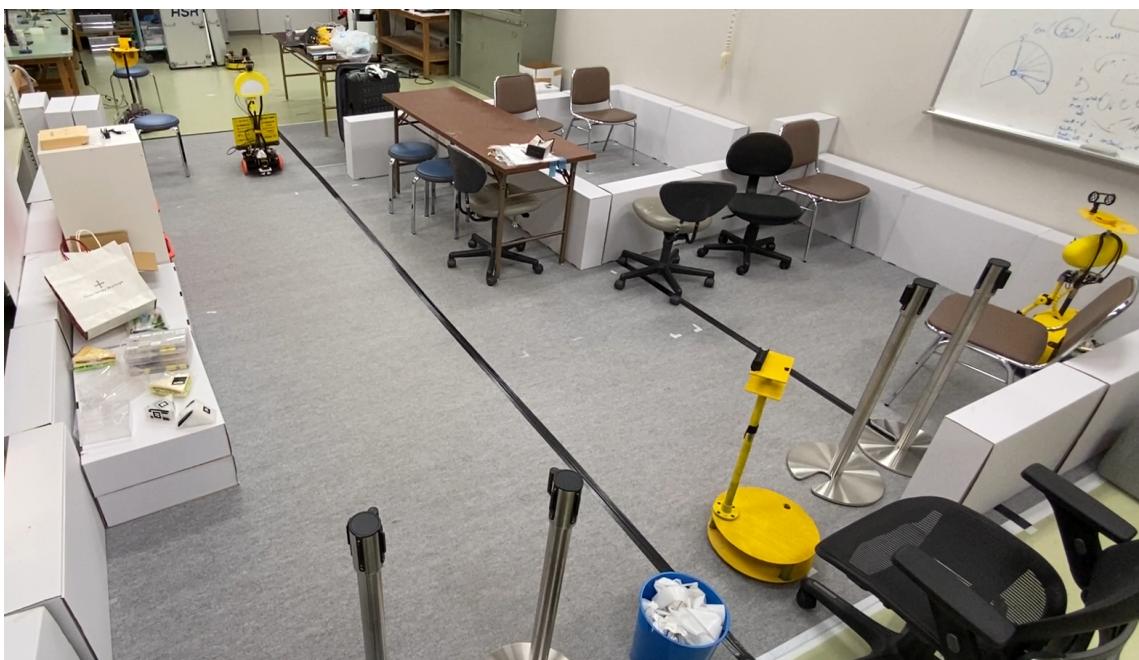


Fig. 4.3: Image of tracking experiment environment (FMT Laboratory Room 326)



Fig. 4.4: Straight road



Fig. 4.5: Curved road



Fig. 4.6: Right angle road

4.2.3 最大追従速度実験

要求仕様(3)を検証するため、Fig. 4.7 のような 10[m] の直線経路を用意し最大追従速度を計測する。実験する速度は 0.1[m/s] から開始し、0.1 ずつ速度を上昇させ、Happy Edu が追従できなくなった場合の 1 つ前の速度を最大追従速度とする。追従の成否は、Happy Edu が追従対象者を 10[m] 以上追従したことを追従成功とする。また、Happy Edu の追従速度は追従対象者の歩行速度に依存しており、追従対象者の歩行速度を指定した速度にする必要がある。そのため、Fig. 4.8 のような構成で実験する。0.1[m/s] の場合であれば、Fig. 4.8 の「Speed keeper」を 0.1[m/s] で直進させ、「Speed keeper」に追従対象者が追従する。さらに、追従対象者に Happy Edu が追従することで、指定した速度での実験をする。これを、0.1[m/s] から開始し、Happy Edu が追従できなくなるまで試行を繰り返す。



Fig. 4.7: Maximum tracking speed experiment environment image (FMT Laboratory Room 326)

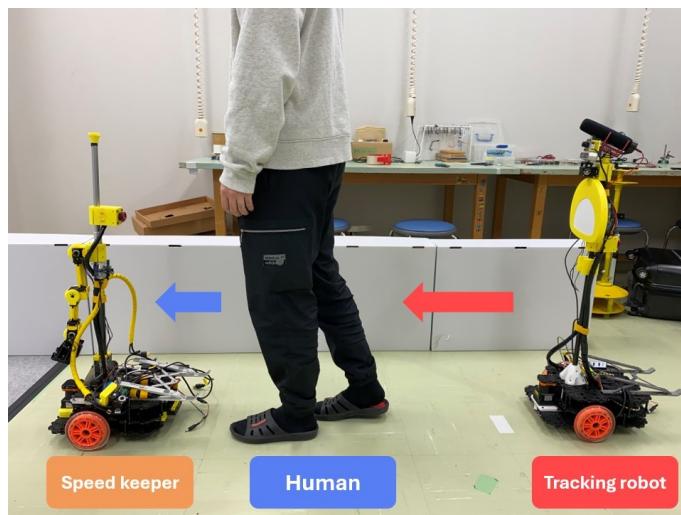


Fig. 4.8: Image of maximum tracking speed experiment

4.3 実験結果

4.3.1 追従実験

追従実験の結果を Table 4.4 に示す。追従実験では、直線経路、曲線経路、直角経路において、それぞれ 10 回ずつ試行し、計 30 回の追従実験を試行したが、すべての試行において追従が成功した。

追従実験中の実世界での様子と同時刻における Happy Edu の内部処理の様子をそれぞれ、Fig. 4.9、Fig. 4.10 に示す。Fig. 4.10 から、追従対象者以外の物体を誤検出していることがわかる。しかし、提案手法の「追従目標の特定」により正確に追従対象者を検出していることがわかる。また、Fig. 4.9 より、誤検出された物体は椅子の円柱部分であり、追従対象者の脚部と形状が類似している部分であった。

Table 4.4: Success rate of traking in each road

Road	Success rate [%]
Straight road	100
Curved road	100
Right angle road	100

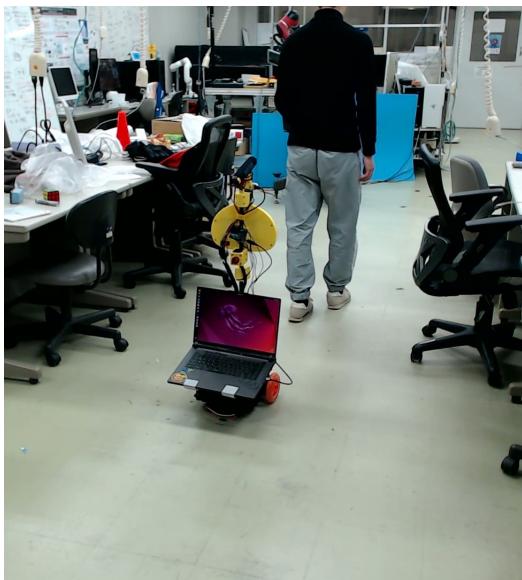


Fig. 4.9: Tracking experiment (Real view)

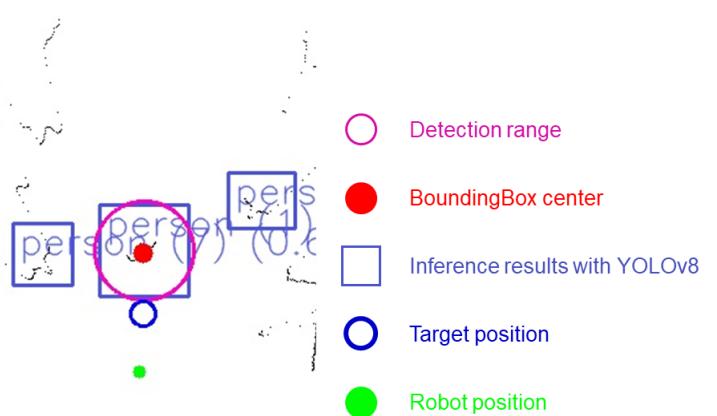


Fig. 4.10: Tracking experiment (Internal view)

4.3.2 最大追従速度実験

最大追従速度実験の結果を Table 4.5 と Fig. 4.11 に示す。Table 4.5 より、 $0.1[m/s] \sim 0.5[m/s]$ までは $11[m]$ 以上の追従が確認でき、 $0.6[m/s]$ では $10[m]$ 以上の追従が確認されなかった。また、Fig. 4.11 より $0.1[m/s] \sim 0.5[m/s]$ では、Happy Edu から追従対象者までの偏差が $0.5[m]$ 以下であるが、 $0.6[m/s]$ では追従対象者までの距離の偏差が増加していき、 $6[m]$ 付近で追従できなくなっていることが分かる。

以上より、本プロジェクトが提案する人追従システムの最大追従速度は $0.5[m/s]$ であることがわかる。

Table 4.5: Maximum tracking speed experimental result

Tracking speed [m/s]	Travel distance tracked [m]
0.1	11.73
0.2	11.31
0.3	11.58
0.4	11.53
0.5	11.48
0.6	6.231

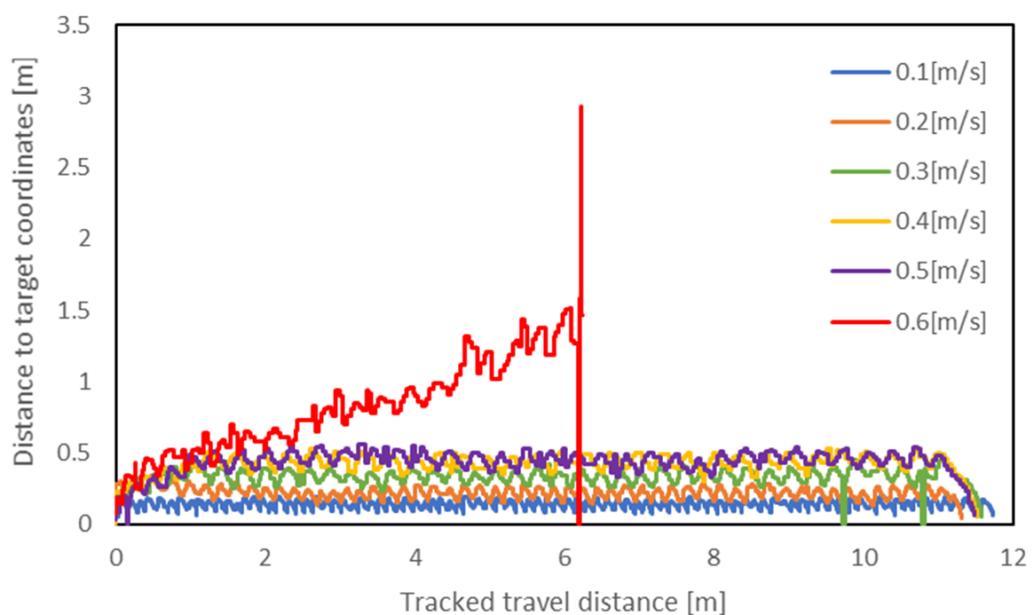


Fig. 4.11: Tracking speed graph

4.4 考察

実験結果から、雑多な環境下での直線経路、曲線経路、直角経路において 100[%] の人追従が確認されたため、要求仕様(2)を満たすことができた。YOLOv8 による両脚部の検出では、椅子などの物体を両脚部として誤検出してはいたが、正確に追従対象者を追従できた。これは、「追従目標の特定」により追従対象者を正しく追跡できていることが考えられる。

最大追従速度実験では、最大追従速度が 0.5[m/s] となり要求仕様(3)を満たすことができた。最大追従速度が 0.5[m/s] となった要因は、TurtleBot3 Big Wheel の最大直進速度が 0.5[m/s] であることに起因していると考えらる。このことから、本プロジェクトが提案する人追従システムは、人の平均歩行速度以上の最大直進速度で移動できるロボット台車に実装することにより、追従対象者がロボットに合わせて歩行速度を低下させる課題が解決できると考えられる。

第5章

結論

5.1 まとめ

本プロジェクトでは、2D-LiDARのデータとYOLOv8を用いた人追従システムの開発を行い、追従実験と最大追従速度実験により追従性能の検証をした。本プロジェクトが提案する手法は、2D-LiDARの距離データを俯瞰画像に変換し、21061枚の俯瞰画像から作成したデータセットとYOLOv8の物体検出アルゴリズムにより両脚部分の検出器を作成した。さらに、動的検出範囲を実装し追従目標の特定をすることで目標座標を生成し、Happy Eduから目標座標までの距離と角度の偏差を収束させるPID制御によりTurtleBot3 Big Wheelを制御することで、人追従システムを実現した。結果として、雑多な環境下での追従と0.5[m/s]の最大追従速度が確認でき、すべての要求仕様を満たすことができた。

5.2 今後の課題

今後の課題として、ロボット台車の最大直進速度が0.5[m/s]であることから、追従対象者が普段の歩行速度で歩行できない。また、本プロジェクトが提案するシステムは、人混み中での人追従を想定していないため、ロボットと追従対象者の間に障害物や人が出現した場合に追従できなくなる可能性がある。これらの課題を解決するため、人の平均歩行速度以上の移動ができるロボット台車を使用し、追従目標の特定処理を改良することが必要である。

謝 辞

本プロジェクトを行うにあたり全体を通してご指導、ご教授、議論などのご助力をいただきました本学ロボティクス学科の出村公成教授に深く感謝いたします。また、データセットの作成や実験にご助力いただいた、出村研究室の皆様にお礼申し上げます。最後に、これまで学生生活を支えていただいた両親に深く感謝いたします。

令和6年2月8日

参考文献

- [1] 総務省統計局, ”統計からみた我が国の高齢者”, (<https://www.stat.go.jp/data/topics/pdf/topics138.pdf>, 2024年2月1日閲覧).
- [2] 内閣府, ”高齢化の現状と将来像”, (https://www8.cao.go.jp/kourei/whitepaper/w-2023/zenbun/pdf/1s1s_01.pdf, 2024年2月1日閲覧).
- [3] 宮口幹太, “近年の建設工事用ロボット開発について” 計測と制御 第61巻 第9号, p. 641-644, 2022.
- [4] 大島章, 城吉宏泰, 柄川索, 松下裕介, 阪東茂, ”既存 AGV を超える特長を持った協働運搬ロボット「サウザー」”, 日本ロボット学会誌 第39巻 第1号, p. 65-66, 2021.
- [5] 飯田一成, 出村公成, ”深層学習を用いた人追従機能の開発”, 令和4年度金沢工業大学工学部ロボティクス学科プロジェクトデザイン III, 2023.
- [6] Claudia Álvarez-Aparicio, Ángel Manuel Guerrero-Higueras, Francisco Javier Rodríguez-Lera, Jonatan Ginés Clavero, Francisco Martín Rico and Vicente Matellán, ”People Detection and Tracking Using LIDAR Sensors”, Robotics 2019, 8, 75.
- [7] Fei Luo, Stefan Poslad, and Eliane Bodanese, ”Temporal convolutional networks for multi-person activity recognition using a 2D LIDAR”, IEEE Internet of Things Journal, Volume: 7, Issue: 8, 2020.
- [8] Ángel Manuel Guerrero-Higueras, Claudia Álvarez-Aparicio, María Carmen Calvo Olivera, Francisco J. Rodríguez-Lera, Camino Fernández-Llamas, Francisco Martín Rico and Vicente Matellán, ”Tracking People in a Mobile Robot From 2D LIDAR Scans Using Full Convolutional Neural Networks for Security in Cluttered Environments”, Frontiers in Neurorobotics, Volume 12, Article 85, 2019.

- [9] Mahmudul Hasan, Junichi Hanawa, Riku Goto, Hisato Fukuda, Yoshinori Kuno and Yoshi-nori Kobayashi, "Tracking People Using Ankle-Level 2D LiDAR for Gait Analysis", Advances in Artificial Intelligence, Software and Systems Engineering, pp 40-46, 2020
- [10] DAPING JIN , ZHENG FANG, (Member, IEEE), AND JIEXIN ZENG, "A Robust Au-tonomous Following Method for Mobile Robots in Dynamic Environments", IEEE Access, Volume: 8, pp. 150311-150325, 2020.

付録

パラメータの設定ファイルを、ソースコード 5.1 に示す。

ソースコード 5.1: follow_me_params.yaml

```
1 /follow_me/laser_to_img:  
2   ros__parameters:  
3     # 縮小サイズを取得. 1[px] = 0.01[m]  
4     discrete_size: 0.01  
5     # Max LiDAR Range  
6     max_lidar_range: 3.5  
7     # 画像を表示するフラッグ  
8     img_show_flg: False  
9  
10 /follow_me/person_detector:  
11   ros__parameters:  
12     # 追従対象との距離  
13     target_dist: 0.5  
14     # 追従対象を見失ったときに追従を再開する時の距離の誤差  
15     target_diff: 0.3  
16     # 追従ポイント(制御を止める領域)の半径  
17     target_radius: 0.1  
18     # 人を検出する範囲(円)の半径  
19     target_range: 0.4  
20     # 起動時に追従対象者を検出するまでの待機時間  
21     init_time: 3.0  
22     # 起動時に追従対象を検出するまでの flg  
23     none_person_flg: True  
24  
25 /follow_me/base_controller:  
26   ros__parameters:  
27     # ロボットからみて tolerance[°] 以内だったら積分制御しない視野角  
28     tolerance: 1.0  
29     # 積分制御をし始める視野角 [°]  
30     i_range: 3.0  
31     # 並進の P ゲイン=====  
32     lkp: 0.3  
33     # 旋回の PID ゲイン=====  
34     # P ゲイン  
35     akp: 0.005
```

```
36 # Iゲイン  
37 aki: 0.0002  
38 # Dゲイン  
39 akd: 0.0009
```

2D-LiDAR の距離データから俯瞰画像を生成するソースコードを、ソースコード 5.2 に示す。

ソースコード 5.2: laser_to_image.py

```
1 import numpy as np  
2 import os  
3 import sys  
4 import cv2  
5 import math  
6 import rclpy  
7 from rclpy.node import Node  
8 from rclpy.parameter import Parameter  
9 from sensor_msgs.msg import LaserScan, Image  
10 from rclpy.qos import qos_profile_sensor_data  
11 from rcl_interfaces.msg import SetParametersResult  
12 from cv_bridge import CvBridge, CvBridgeError  
13 # Custom  
14 from .modules.gradient import gradation_3d_img as gradation  
15  
16  
17 class LaserToImg(Node):  
18     def __init__(self):  
19         super().__init__('laser_to_img')  
20         # Publisher  
21         self.pub = self.create_publisher(Image, '/follow_me/laser_img',  
22                                         10)  
22         # Subscriber  
23         self.create_subscription(LaserScan, '/scan',  
24                                     self.cloud_to_img_callback, qos_profile_sensor_data)  
24         # OpenCV  
25         self.bridge = CvBridge()  
26         # Parameters  
27         self.declare_parameters(  
28             namespace='',  
29             parameters=[  
30                 ('discrete_size', Parameter.Type.DOUBLE),  
31                 ('max_lidar_range', Parameter.Type.DOUBLE),  
32                 ('img_show_flg', Parameter.Type.BOOL)])  
33         self.add_on_set_parameters_callback(self.param_callback)  
34         # Get parameters  
35         self.param_dict = {}  
36         self.param_dict['discrete_size'] =  
            self.get_parameter('discrete_size').value
```

```
37         self.param_dict['max_lidar_range'] =
38             self.get_parameter('max_lidar_range').value
39         self.param_dict['img_show_flg'] =
40             self.get_parameter('img_show_flg').value
41     # Values
42     self.color_list = gradation([0,0,255], [255,0,0], [1, 100],
43                               [True,True,True])[0]
44     # Output
45     self.output_screen()
46
47
48     def output_screen(self):
49         for key, value in self.param_dict.items():
50             self.get_logger().info(f"{key}: {value}")
51
52
53     def param_callback(self, params):
54         for param in params:
55             self.param_dict[param.name] = param.value
56             self.get_logger().info(f"Set param: {param.name} >>>
57             {param.value}")
58         return SetParametersResult(successful=True)
59
60
61     def cloud_to_img_callback(self, scan):
62
63         # discrete_factor
64         discrete_factor = 1/self.param_dict['discrete_size']
65         # max_lidar_range と discrete_factor を使って画像サイズを設定する
66         img_size =
67             int(self.param_dict['max_lidar_range']*2*discrete_factor)
68
69
70         # LiDAR データ
71         maxAngle = scan.angle_max
72         minAngle = scan.angle_min
73         angleInc = scan.angle_increment
74         maxLength = scan.range_max
75         ranges = scan.ranges
76         intensities = scan.intensities
77         #intensities = scan.intensities
78
79         # 距離データの個数を格納
80         num_pts = len(ranges)
81         # 721行 2列の空行列を作成
82         xy_scan = np.zeros((num_pts, 2))
83         # 3チャンネルの白色ブランク画像を作成
84         blank_img = np.zeros((img_size, img_size, 3), dtype=np.uint8) +
85             255
86         # ranges の距離・角度からすべての点を XY に変換する処理
87         for i in range(num_pts):
88             # 範囲内かを判定
89             if (ranges[i] > self.param_dict['max_lidar_range']) or
```

```
          (math.isnan(ranges[i])):
80      pass
81  else:
82      # 角度と XY 座標の算出処理
83      angle = minAngle + float(i)*angleInc
84      xy_scan[i][1] = float(ranges[i]*math.cos(angle)) # y 座標
85      xy_scan[i][0] = float(ranges[i]*math.sin(angle)) # x 座標
86
87      # ブランク画像にプロットする処理
88      for i in range(num_pts):
89          pt_x = xy_scan[i, 0]
90          pt_y = xy_scan[i, 1]
91          if (pt_x < self.param_dict['max_lidar_range']) or (pt_x >
-1*(self.param_dict['max_lidar_range']-self.param_dict['discrete_si
or (pt_y < self.param_dict['max_lidar_range']) or (pt_y >
-1 *
(self.param_dict['max_lidar_range']-self.param_dict['discrete_size'])
92          pix_x = int(math.floor((pt_x +
self.param_dict['max_lidar_range']) * discrete_factor))
93          pix_y =
int(math.floor((self.param_dict['max_lidar_range'] -
pt_y) * discrete_factor))
94          if (pix_x > img_size) or (pix_y > img_size):
95              print("Error")
96          else:
97              blank_img[pix_y, pix_x] = [0, 0, 0]
98
99      # CV2 画像から ROS メッセージに変換してトピックとして配布する
100     img = self.bridge.cv2_to_imgmsg(blank_img, encoding="bgr8")
101     self.pub.publish(img)
102
103     # 画像の表示処理。 imgshow_flg が True の場合のみ表示する
104     if self.param_dict['img_show_flg']:
105         cv2.imshow('laser_img', blank_img)
106         cv2.waitKey(3)
107         # 更新のため一旦消す
108         blank_img = np.zeros((img_size, img_size, 3))
109     else:
110         pass
111
112 def main():
113     rclpy.init()
114     node = LaserToImg()
115     try:
116         rclpy.spin(node)
117     except KeyboardInterrupt:
118         pass
119     node.destroy_node()
120     rclpy.shutdown()
```

追従目標を特定するソースコードを、ソースコード 5.3 に示す。

ソースコード 5.3: person_detector.py

```
1 import math
2 import time
3 import cv2
4 import rclpy
5 from rclpy.node import Node
6 from rclpy.parameter import Parameter
7 from rcl_interfaces.msg import SetParametersResult, ParameterEvent
8 from rcl_interfaces.srv import GetParameters
9 from sensor_msgs.msg import Image
10 from geometry_msgs.msg import Point
11 from cv_bridge import CvBridge, CvBridgeError
12 from yolov8_msgs.msg import DetectionArray
13
14
15 class PersonDetector(Node):
16     def __init__(self):
17         super().__init__('person_detector')
18         # OpenCV Bridge
19         self.bridge = CvBridge()
20         # Publisher
21         self.point_pub = self.create_publisher(Point,
22             '/follow_me/target_point', 10)
23         self.img_pub = self.create_publisher(Image, '/follow_me/image',
24             10)
25         # Subscriber
26         self.create_subscription(DetectionArray, '/yolo/detections',
27             self.yolo_callback, 10)
28         self.create_subscription(Image, '/yolo/dbg_image', self.img_show,
29             10)
30         self.create_subscription(ParameterEvent, '/parameter_events',
31             self.param_event_callback, 10)
32         # Service
33         self.srv_client = self.create_client(GetParameters,
34             '/follow_me/laser_to_img/get_parameters')
35         while not self.srv_client.wait_for_service(timeout_sec=0.5):
36             self.get_logger().info('/follow_me/laser_to_img server is not
37             available ...')
38         # Parameters
39         self.declare_parameters(
40             namespace='',
41             parameters=[
42                 ('target_dist', Parameter.Type.DOUBLE),
43                 ('init_time', Parameter.Type.DOUBLE),
44                 ('none_person_flg', Parameter.Type.BOOL),
45                 ('target_diff', Parameter.Type.DOUBLE),
46                 ('target_radius', Parameter.Type.DOUBLE),
```

```
40             ('target_range', Parameter.Type.DOUBLE)])
41         self.add_on_set_parameters_callback(self.param_callback)
42     # Get parameters
43     self.param_dict = {}
44     self.param_dict['target_dist'] =
45         self.get_parameter('target_dist').value
46     self.param_dict['init_time'] =
47         self.get_parameter('init_time').value
48     self.param_dict['none_person_flg'] =
49         self.get_parameter('none_person_flg').value
50     self.param_dict['target_diff'] =
51         self.get_parameter('target_diff').value
52     self.param_dict['target_radius'] =
53         self.get_parameter('target_radius').value
54     self.param_dict['target_range'] =
55         self.get_parameter('target_range').value
56     self.param_dict['discrete_size'] = self.get_param() #
57         laser_to_img からもってくる
58     # Value
59     self.person_list = []
60     self.target_data = [] # 追従対象のデータを保存するリスト
61     self.before_data = [0.0, 0.0, 0.0]
62     self.center_x = 0.0
63     self.center_y = 0.0
64     self.target_point = Point()
65     self.target_px = []
66     self.laser_img = 0.0
67     self.height = 0.0
68     self.width = 0.0
69     # Output
70     self.output_screen()

71 def output_screen(self):
72     for key, value in self.param_dict.items():
73         self.get_logger().info(f"{key}: {value}")

74 def param_event_callback(self, receive_msg):
75     for data in receive_msg.changed_parameters:
76         if data.name == 'discrete_size':
77             self.param_dict['discrete_size'] = data.value.double_value
78             self.get_logger().info(f"Param event: {data.name} >>>
79             {self.param_dict['discrete_size']}")

80 def param_callback(self, params):
81     for param in params:
82         self.param_dict[param.name] = param.value
83         self.get_logger().info(f"Set param: {param.name} >>>
84             {param.value}")
85     return SetParametersResult(successful=True)
```

```
80
81     def get_param(self):
82         req = GetParameters.Request()
83         req.names = ['discrete_size']
84         future = self.srv_client.call_async(req)
85         while rclpy.ok():
86             rclpy.spin_once(self, timeout_sec=0.1)
87             if future.done():
88                 break
89         return future.result().values[0].double_value
90
91     def yolo_callback(self, receive_msg):
92         if not receive_msg.detections:
93             self.center_x = self.center_y = None
94         else:
95             self.person_list.clear()
96             for person in receive_msg.detections:
97                 px = Point()
98                 px.x = person.bbox.center.position.x
99                 px.y = person.bbox.center.position.y
100                self.person_list.append(px)
101
102    def plot_robot_point(self):
103        # 画像の中心を算出
104        robot_x = round(self.width / 2)
105        robot_y = round(self.height / 2)
106        # 描画処理
107        cv2.circle(img = self.laser_img,
108                  center = (round(robot_x), round(robot_y)),
109                  radius = 5,
110                  color = (0, 255, 0),
111                  thickness = -1)
112
113    def plot_person_point(self):
114        cv2.circle(img = self.laser_img,
115                  center = (round(self.center_x), round(self.center_y)),
116                  radius = 8,
117                  color = (0, 0, 255),
118                  thickness = -1)
119
120    def plot_target_point(self):
121        cv2.circle(img = self.laser_img,
122                  center = (int(self.target_px[0]),
123                             int(self.target_px[1])),
124                  radius =
125                             int(self.param_dict['target_radius']/self.param_dict['discret']
```

```
127     def plot_target_range(self):
128         cv2.circle(img = self.laser_img,
129                     center = (int(self.before_data[2][0]),
130                                int(self.before_data[2][1])),
131                     radius =
132                                int(self.param_dict['target_range']/self.param_dict['discrete_size']),
133                     color = (196, 0, 255),
134                     thickness = 2)
135
136
137     def diff_distance(self, data):
138         return abs(data - self.before_data[0])
139
140     def euclidean_distance(self, data, before_data):
141         return math.sqrt((data.x-before_data.x)**2 +
142                           (data.y-before_data.y)**2)
143
144     def select_target(self, robot_px_x, robot_px_y):
145         # personまでの距離と座標のリストを作成
146         self.target_data.clear()
147         for person_px in self.person_list:
148             person_point = Point()
149             person_point.x = (robot_px_x -
150                               person_px.y)*self.param_dict['discrete_size']
151             person_point.y = (robot_px_y -
152                               person_px.x)*self.param_dict['discrete_size']
153             distance = math.sqrt(person_point.x**2 + person_point.y**2)
154             self.target_data.append([distance, person_point,
155                                     [person_px.x, person_px.y]])
156         # 0番目に1時刻前の追従対象との距離の誤差を格納する
157         self.target_data = [[self.diff_distance(data[0]), data[1],
158                             data[2]] for data in self.target_data]
159         # 検出範囲内のpersonを追従対象とする(起動時だけ一番近い人を追従対象
160         # にする)
161         if self.param_dict['none_person_flg']:
162             target = min(self.target_data)
163             param_bool = Parameter('none_person_flg',
164                                   Parameter.Type.BOOL, False)
165             self.set_parameters([param_bool])
166         else:
167             for data in self.target_data:
168                 diff = self.euclidean_distance(data[1],
169                                                self.before_data[1])
170                 if diff <= self.param_dict['target_range']:
171                     target = data
172                     break
173                 else:
174                     target = None
175
176         # targetがNoneだったらself.before_dataを初期化してnone_person_flgをTrueにする
```

```
165     if target is None:
166         #self.before_data = [0.0, 0.0, 0.0]
167         #param_bool = Parameter('none_person_flg',
168             Parameter.Type.BOOL, True)
169         #self.set_parameters([param_bool])
170         pass
171     else:
172         # 計算のために距離を保存
173         self.before_data = target
174     return target
175
176 def generate_target(self):
177     self.target_px.clear()
178     # 画像の中心を算出
179     robot_x = self.height / 2
180     robot_y = self.width / 2
181     # 追従目標を選定
182     target_person = self.select_target(robot_x, robot_y)
183     if not target_person is None:
184         result_point = target_person[1]
185         self.center_x = target_person[2][0]
186         self.center_y = target_person[2][1]
187     else:
188         result_point = False
189     return result_point
190
191 def img_show(self, receive_msg):
192     self.laser_img = self.bridge.imgmsg_to_cv2(receive_msg,
193         desired_encoding='bgr8')
194     self.height, self.width, _ = self.laser_img.shape[:3]
195     # ロボットの座標をプロット
196     self.plot_robot_point()
197     # 追従対象の検出範囲をプロット
198     if not self.param_dict['none_person_flg']:
199         self.plot_target_range()
200     # personがいるか判定
201     if self.person_list:
202         # 追従対象を生成
203         target_point = self.generate_target()
204         # 追従対象がいなければロボット台車を停止する
205         if not target_point:
206             self.target_point.x = 0.0
207             self.target_point.y = 0.0
208             self.point_pub.publish(self.target_point)
209         else:
210             robot_x = self.height / 2
211             robot_y = self.width / 2
212             # 目標座標を生成(px): 横x, 縦y
```

```

212         target_x = self.center_x
213         target_y = self.center_y +
214             (self.param_dict['target_dist']/self.param_dict['discrete_size'])
215         self.target_px.append(target_x)
216         self.target_px.append(target_y)
217         # 目標座標を生成(m): 縦x, 横y(ロボット座標系に合わせる)
218         self.target_point.x = (robot_x -
219             target_y)*self.param_dict['discrete_size']
220         self.target_point.y = (robot_y -
221             target_x)*self.param_dict['discrete_size']
222         # パブリッシュ
223         self.point_pub.publish(self.target_point)
224         # グラフに描画
225         self.plot_target_point()
226         self.plot_person_point()
227
228     # ros2 bag 用にトピックとして画像を配布
229     img = self.bridge.cv2_to_imgmsg(self.laser_img, encoding="bgr8")
230     self.img_pub.publish(img)
231
232     # 画像を表示
233     cv2.imshow('follow_me', self.laser_img)
234     cv2.waitKey(1)
235
236
237 def main():
238     rclpy.init()
239     node = PersonDetector()
240     try:
241         rclpy.spin(node)
242     except KeyboardInterrupt:
243         pass
244     node.destroy_node()
245     rclpy.shutdown()

```

ロボット台車を制御するソースコードを、ソースコード 5.4 に示す。

ソースコード 5.4: base_controller.py

```

1 import time
2 import math
3 import rclpy
4 from rclpy.node import Node
5 from rclpy.parameter import Parameter
6 from rcl_interfaces.msg import SetParametersResult, ParameterEvent
7 from rcl_interfaces.srv import GetParameters
8 from nav_msgs.msg import Odometry
9 from geometry_msgs.msg import Twist, Point
10
11

```

```
12 class BaseController(Node):
13     def __init__(self):
14         super().__init__('base_controller')
15         # Publisher
16         self.pub = self.create_publisher(Twist, '/cmd_vel', 10)
17         self.data_pub = self.create_publisher(Point,
18             '/follow_me/distance_angle_data', 10)
19         # Subscriber
20         self.create_subscription(Point, '/follow_me/target_point',
21             self.callback, 10)
22         self.create_subscription(Odometry, '/odom', self.odom_callback,
23             10)
24         self.create_subscription(ParameterEvent, '/parameter_events',
25             self.param_event_callback, 10)
26         # Service
27         self.srv_client = self.create_client(GetParameters,
28             '/follow_me/person_detector/get_parameters')
29         while not self.srv_client.wait_for_service(timeout_sec=0.5):
30             self.get_logger().info('/follow_me/laser_to_img server is not
31             available ...')
32         # Parameters
33         self.declare_parameters(
34             namespace='',
35             parameters=[
36                 ('tolerance', Parameter.Type.DOUBLE),
37                 ('i_range', Parameter.Type.DOUBLE),
38                 ('lkp', Parameter.Type.DOUBLE),
39                 ('akp', Parameter.Type.DOUBLE),
40                 ('aki', Parameter.Type.DOUBLE),
41                 ('akd', Parameter.Type.DOUBLE)])
42         self.add_on_set_parameters_callback(self.param_callback)
43         # Get parameters
44         self.param_dict = {}
45         self.param_dict['tolerance'] =
46             self.get_parameter('tolerance').value
47         self.param_dict['i_range'] = self.get_parameter('i_range').value
48         self.param_dict['lkp'] = self.get_parameter('lkp').value
49         self.param_dict['akp'] = self.get_parameter('akp').value
50         self.param_dict['aki'] = self.get_parameter('aki').value
51         self.param_dict['akd'] = self.get_parameter('akd').value
52         self.param_dict['target_radius'] = self.get_param() #
53             person_detector からもってくる
54         # Value
55         self.twist = Twist()
56         self.target_angle = 0.0
57         self.target_distance = 0.0
58         self.target_x = 0.0
59         self.target_y = 0.0
60         self.delta_t = 0.0
```

```
53         self.robot_angular_vel = 0.0
54         # Output
55         self.output_screen()
56
57     def output_screen(self):
58         for key, value in self.param_dict.items():
59             self.get_logger().info(f"{key}: {value}")
60
61     def param_event_callback(self, receive_msg):
62         for data in receive_msg.changed_parameters:
63             if data.name == 'target_radius':
64                 self.param_dict['target_radius'] = data.value.double_value
65                 self.get_logger().info(f"Param event: {data.name} >>>
66                 {self.param_dict['target_radius']}"))
67
68     def param_callback(self, params):
69         for param in params:
70             self.param_dict[param.name] = param.value
71             self.get_logger().info(f"Set param: {param.name} >>>
72             {param.value}")
73     return SetParametersResult(successful=True)
74
75     def get_param(self):
76         req = GetParameters.Request()
77         req.names = ['target_radius']
78         future = self.srv_client.call_async(req)
79         while rclpy.ok():
80             rclpy.spin_once(self, timeout_sec=0.1)
81             if future.done():
82                 break
83         return future.result().values[0].double_value
84
85     def point_to_angle(self, point):
86         return math.degrees(math.atan2(point.y, point.x))
87
88     def point_to_distance(self, point):
89         distance = math.sqrt(point.x**2 + point.y**2)
90         if point.x < 0.0:
91             distance = -distance
92         return distance
93
94     def callback(self, receive_msg):
95         self.target_x = receive_msg.x
96         self.target_y = receive_msg.y
97         self.target_distance = self.point_to_distance(receive_msg)
98         self.target_angle = self.point_to_angle(receive_msg)
99
100    def odom_callback(self, receive_msg):
101        self.robot_angular_vel = receive_msg.twist.twist.angular.z
```

```
100
101     # 比例制御量計算
102     def p_control(self):
103         return self.param_dict['akp']*self.target_angle
104
105     # 微分制御量計算
106     def d_control(self, p_term):
107         return self.param_dict['akd']*(p_term - self.robot_angular_vel)
108
109     # 積分制御量計算
110     def i_control(self, p_term, d_term):
111         value = 0.0
112         diff = (p_term + d_term) - self.robot_angular_vel
113
114         if not diff < self.param_dict['tolerance'] and diff <
115             self.param_dict['i_range']:
116             value = self.param_dict['aki']*self.target_angle*self.delta_t
117
118         return value
119
120     def pid_update(self):
121         # 制御量を計算
122         p_term = self.p_control()
123         d_term = self.d_control(p_term)
124         i_term = self.i_control(p_term, d_term)
125
126         linear_vel = self.param_dict['lkp']*self.target_distance
127         angular_vel = -1*(p_term + i_term + d_term)
128
129         if linear_vel < 0.0:
130             linear_vel = 0.0
131             angular_vel = 0.0
132
133         return linear_vel, angular_vel
134
135     def in_range(self):
136         result = False
137         if abs(self.target_distance) <= self.param_dict['target_radius']:
138             result = True
139
140     def execute(self, rate=100):
141         start_time = time.time()
142         before_time = 0.0
143
144         while rclpy.ok():
145             self.delta_t = time.time() - start_time
146             rclpy.spin_once(self)
```

```
148     # 許容範囲内外を判
149     if self.in_range():
150         linear_vel = 0.0
151         angular_vel = 0.0
152     else:
153         linear_vel, angular_vel = self.pid_update()
154
155     # 制御量をパブリッシュ
156     self.twist.linear.x = linear_vel
157     self.twist.angular.z = angular_vel
158     self.pub.publish(self.twist)
159
160     # 実験用に目標までの距離と角度をパブリッシュ
161     data = Point()
162     data.x = self.target_distance
163     data.z = self.target_angle
164     self.data_pub.publish(data)
165
166     time.sleep(1/rate)
167
168
169 def main():
170     rclpy.init()
171     node = BaseController()
172     try:
173         node.execute()
174     except KeyboardInterrupt:
175         pass
176
177     node.destroy_node()
178     rclpy.shutdown()
```

以上のソースコードをまとめて起動するソースコードを、ソースコード 5.5 に示す。

ソースコード 5.5: follow_me.launch.py

```
1 import os
2 from ament_index_python.packages import get_package_share_directory
3 import launch
4 from launch import LaunchDescription
5 from launch.actions import DeclareLaunchArgument
6 from launch_ros.actions import Node
7
8 def generate_launch_description():
9     config = os.path.join(
10         get_package_share_directory('recognition_by_lidar'),
11         'config',
12         'follow_me_params.yaml')
13
14     namespace = 'follow_me'
```

```
15
16     return LaunchDescription([
17         Node(
18             namespace=namespace,
19             package='recognition_by_lidar',
20             executable='laser_to_img',
21             name='laser_to_img',
22             parameters=[config],
23             output='screen',
24             respawn=True),
25         Node(
26             namespace=namespace,
27             package='recognition_by_lidar',
28             executable='person_detector',
29             name='person_detector',
30             parameters=[config],
31             output='screen',
32             respawn=True),
33         Node(
34             namespace=namespace,
35             package='recognition_by_lidar',
36             executable='base_controller',
37             name='base_controller',
38             parameters=[config],
39             output='screen',
40             respawn=True,
41             on_exit=launch.actions.Shutdown()),
42     ])
```
