

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 4 日

# 目次

<b>第1章 序 論</b>	<b>1</b>
1.1 はじめに . . . . .	1
1.2 論文構成 . . . . .	2
<b>第2章 従来研究</b>	<b>3</b>
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 . . . . .	13
<b>第3章 提案手法</b>	<b>14</b>
3.1 概要 . . . . .	14
3.2 要求仕様 . . . . .	14
3.3 システム構成 . . . . .	15
3.4 ソフトウェア構成 . . . . .	16
3.5 データセットの作成 . . . . .	17
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
<b>第4章 実験</b>	<b>21</b>
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
<b>第5章 結　言</b>	<b>30</b>
5.1 結言 . . . . .	30
<b>謝　辞</b>	<b>31</b>
<b>参考文献</b>	<b>32</b>
<b>付録</b>	<b>34</b>

# 図目次

2.1	Image of centroid [5] . . . . .	4
2.2	Data Acquisition Environment [5] . . . . .	4
2.3	Human recognition [7] . . . . .	5
2.4	LIDAR data processing [7] . . . . .	6
2.5	Kitchen scenario [7] . . . . .	6
2.6	Architecture of the CNN used by PeTra [8] . . . . .	7
2.7	Robotics mobile lab plan (left), Orbi-One robot (center), and KIO RTLS anchors (right). [8] . . . . .	8
2.8	Comparison image of PeTra and LD by Rviz [8] . . . . .	8
2.9	Person following in environments with obstacles [10] . . . . .	10
2.10	The architecture of our person-following system [10] . . . . .	10
2.11	Human legs detection and human tracking [10] . . . . .	10
2.12	AOA tracking trajectory and Kalman filter trajectory [10] . . . . .	11
2.13	Laser tracking trajectory and Kalman filter trajectory [10] . . . . .	11
2.14	Kalman filter tracking in occluded environments [10] . . . . .	12
2.15	Following trajectory in environments with obstacles [10] . . . . .	12
3.1	System configuration chart . . . . .	15
3.2	Software configuration diagram . . . . .	16
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	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 [5] . . . . .	3
2.2	THE EFFECT OF TRAJECTORY AUGMENTATION [7] . . . . .	6
2.3	Mean error and standard deviation[m] at each location [8] . . . . .	8
2.4	Comparison of performance between FMM-DWA algorithm and MPEPC algorithm [10] . . . . .	12
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ザーで行うことができ、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が不安定になる可能性がある。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人追従を評価していることから、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は人の両脚部分と類似しているため、誤検出してしまう課題があり、これに伴った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

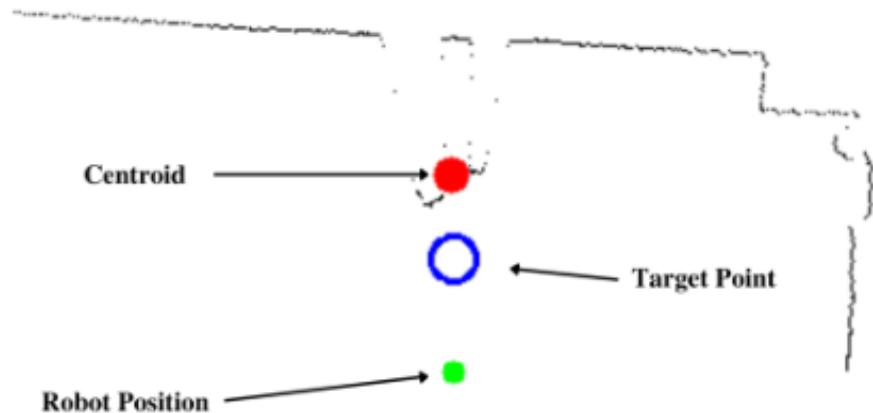
飯田一成らのプロジェクト [5] では、リアルタイム物体検出アルゴリズムである YOLOv5 を用いて、2D-LiDAR の距離データから人の脚部を検出している。

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。複数検出する場合があるため、検出する範囲を一定の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。

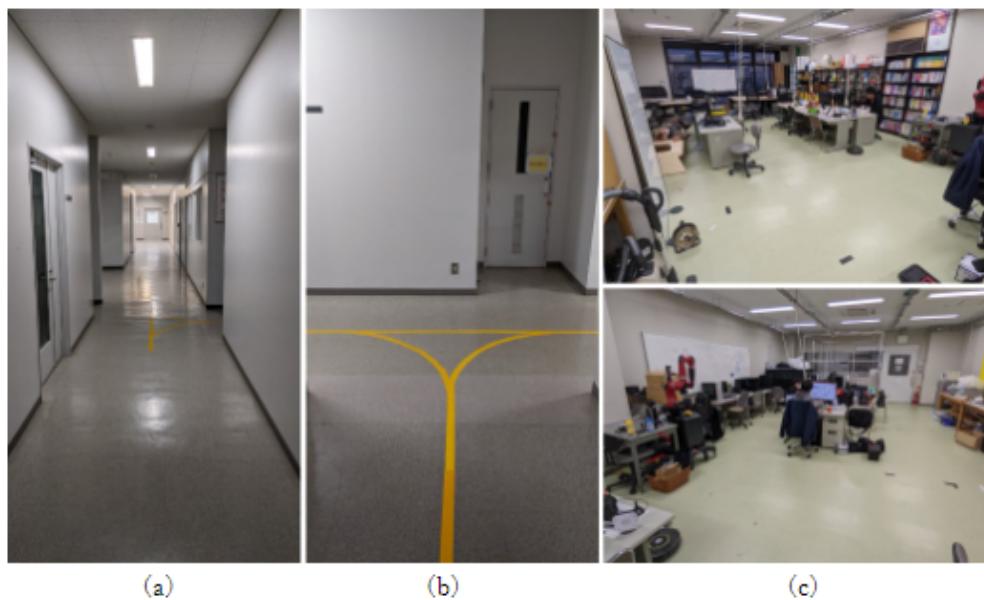
実験方法は、追従実験と衝突回避実験がある。追従実験では Fig. 2.2 3つの経路を設定し、1回のみの試行である。衝突回避実験では、人追従中に追従対象者とロボットとの間に障害物を設置し、停止したかを 3つの状況に分けてそれぞれ 10回試行している。実験結果は、追従実験では 3つの経路において 20[m] の人追従ができていたが、雑多な環境下では追従中に停止することがあった。衝突回避実験では Table 2.1 に示すように、すべての試行において衝突回避ができていた。

**Table 2.1:** Success rate of emergency stops on each road [5]

	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 [5]



**Fig. 2.2:** Data Acquisition Environment [5]

## 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とともに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) の両方に入力される。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) を用いているものである。

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。

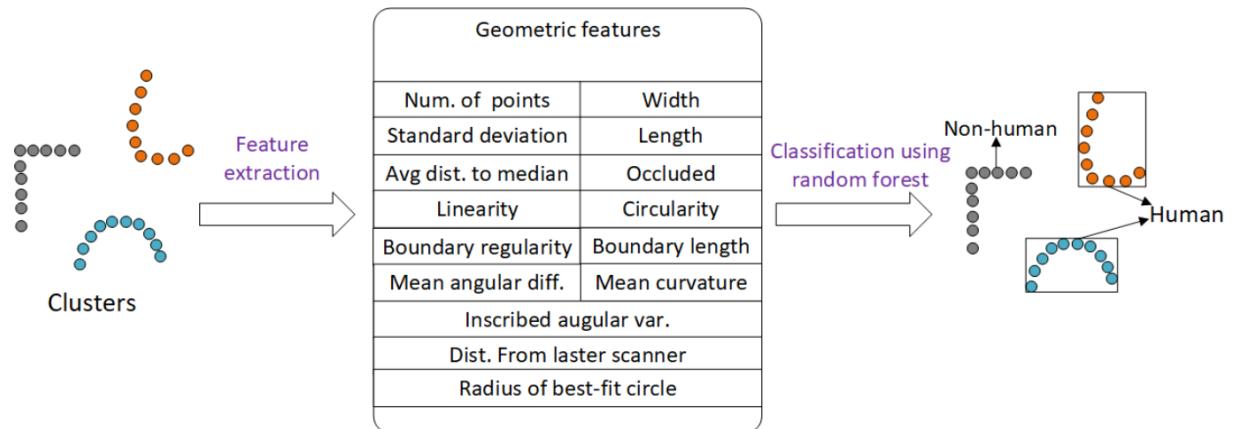
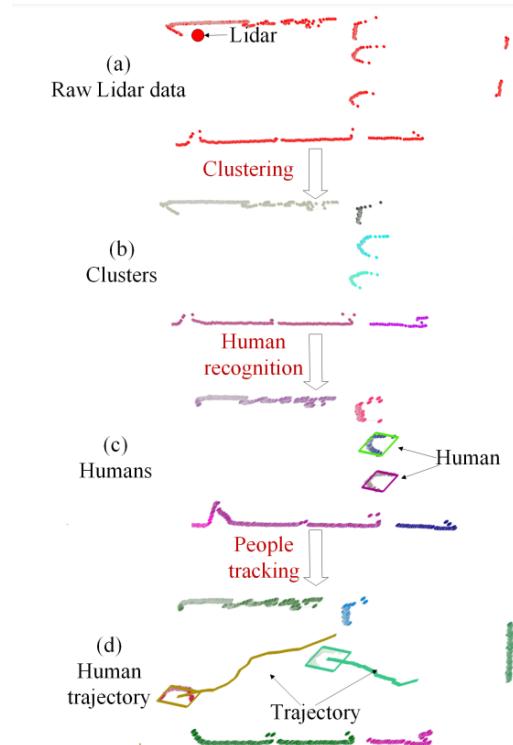


Fig. 2.3: Human recognition [7]



**Fig. 2.4:** LIDAR data processing [7]

**Table 2.2:** THE EFFECT OF TRAJECTORY AUGMENTATION [7]

	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%



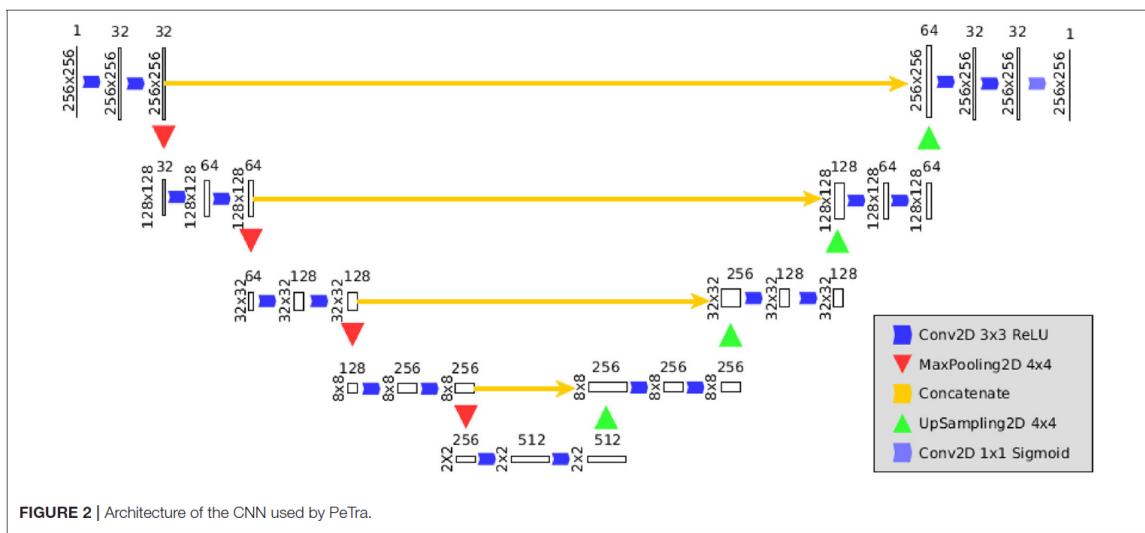
**Fig. 2.5:** Kitchen scenario [7]

## 2.3 FCN と 2D-LiDAR を用いた手法

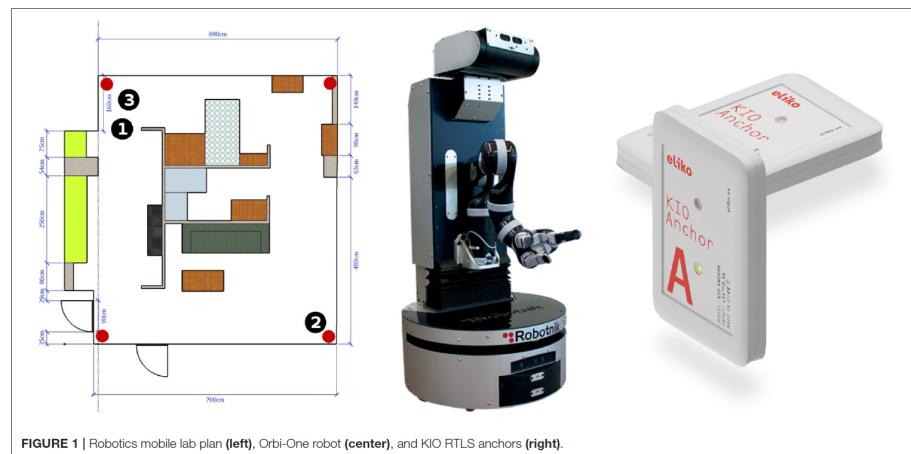
Ángel Manuel Guerrero-Higueras らの研究 [8] では、FCN(Full Convolutional Neural Networks) を用いて、2D-LiDAR の距離データから人の両脚部分を検出することで人追従する「PeTra」を提案している。

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により、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。

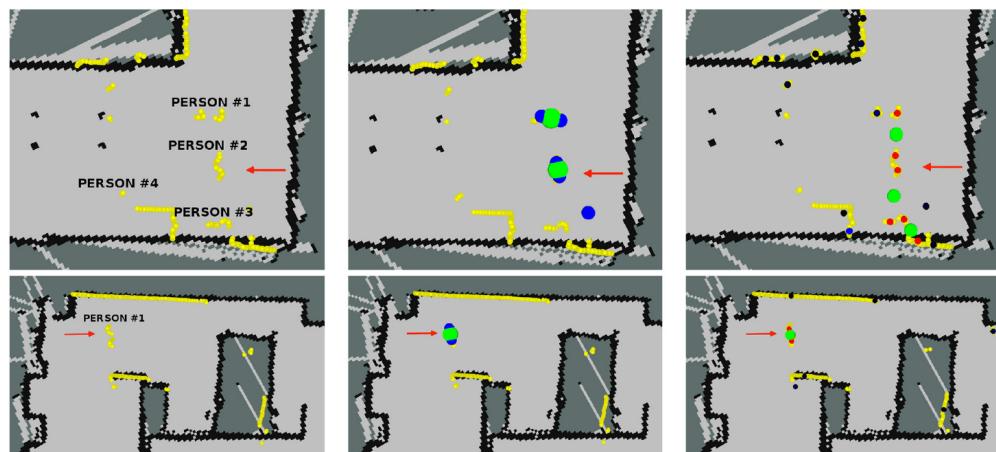
実験方法は、Fig. 2.7 の左のような環境において、Location1、Location2、Location3 で行われる。また、Fig. 2.7 の右にあるタグを追従対象者が所持することにより、タグの位置を真値としている。PeTra の比較対象として、ROS(Robot Operationg System) が提供している LD(Leg Detector) を用いており、追従対象者が所持しているタグからの平均誤差と標準偏差 [m] を PeTra と LD で比較する。実験結果を、Fig. 2.8 と Table 2.3 に示す。Fig. 2.8 の黄色のマーカーは 2D-LiDAR の距離データであり、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 [8]



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



**Fig. 2.8:** Comparison image of PeTra and LD by Rviz [8]

**Table 2.3:** Mean error and standard deviation[m] at each location [8]

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

## 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、システム全体は Fig. 2.10 のようになっており、2D-LiDAR と AOA タグのデータをカルマンフィルタを用いた追跡情報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、人に AOA タグを所持してもらい、ロボットが位置情報を受信している。Fig. 2.12 は、AOA タグとカルマンフィルタの追従軌跡を比較したものであり、赤いデータが AOA タグ、黒いデータがカルマンフィルタのデータである。AOA タグ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 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。Fig. 2.13 は、2D-LiDAR とカルマンフィルタの追跡軌跡を比較したものであり、赤いデータが 2D-LiDAR、黒いデータがカルマンフィルタのデータである。カルマンフィルタより 2D-LiDAR の追跡軌跡の方が滑らかであると述べられている。これらの AOA タグと 2D-LiDAR の追跡軌跡を組み合わせたものが Fig. 2.14 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。2D-LiDAR での追従は、Fig. 2.14 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  $1m/s^2$  未満である場合が 90% 以上であり、MPEPC における加速度が  $1m/s^2$  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 [10]

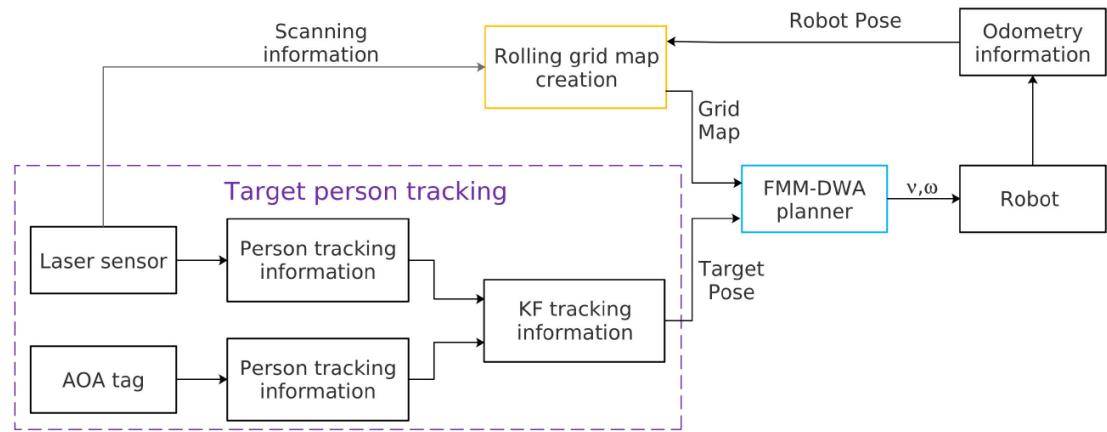


Fig. 2.10: The architecture of our person-following system [10]

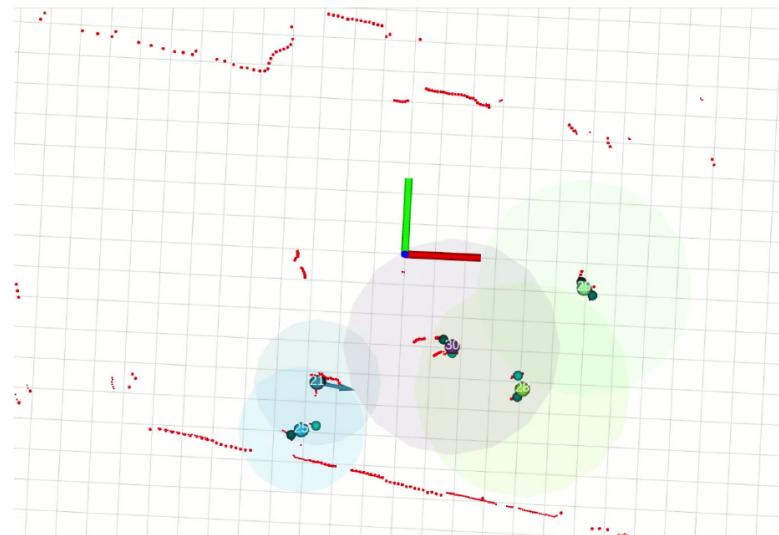


Fig. 2.11: Human legs detection and human tracking [10]

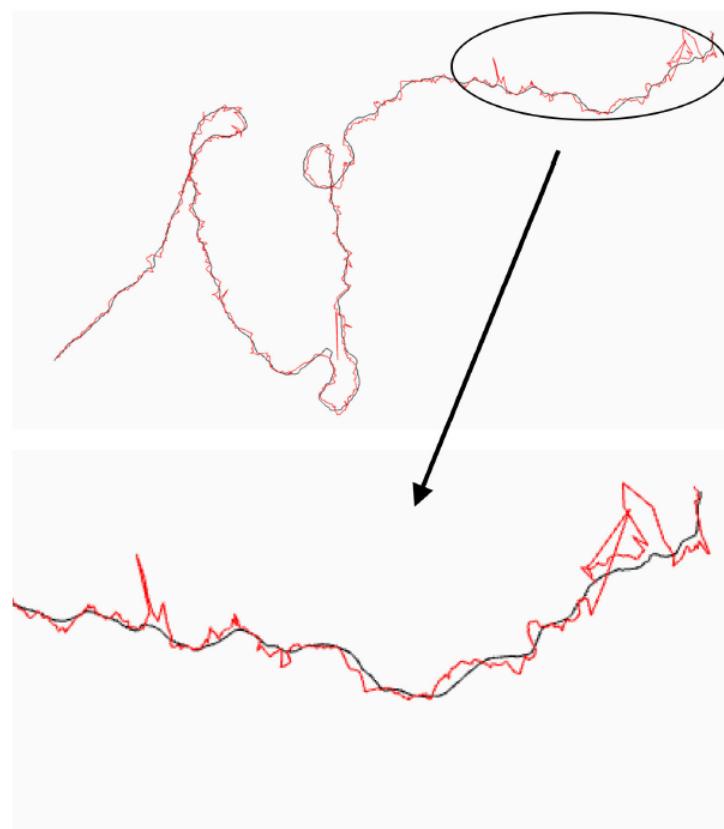


Fig. 2.12: AOA tracking trajectory and Kalman filter trajectory [10]

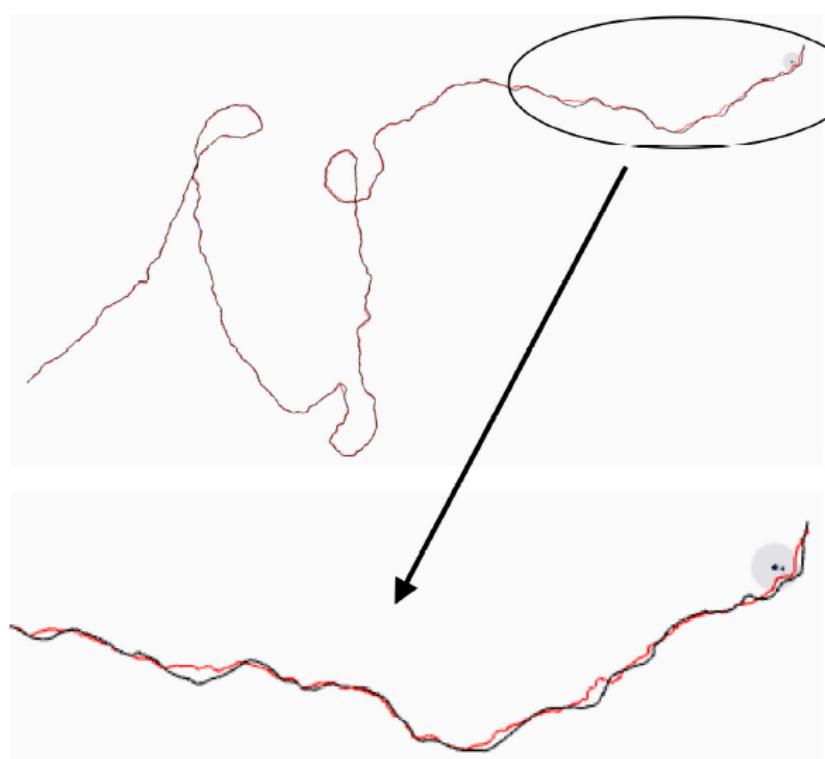
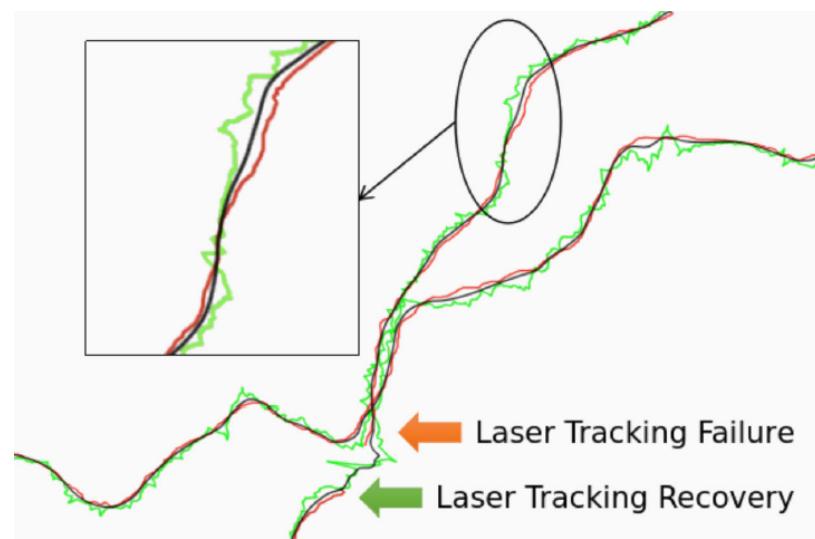
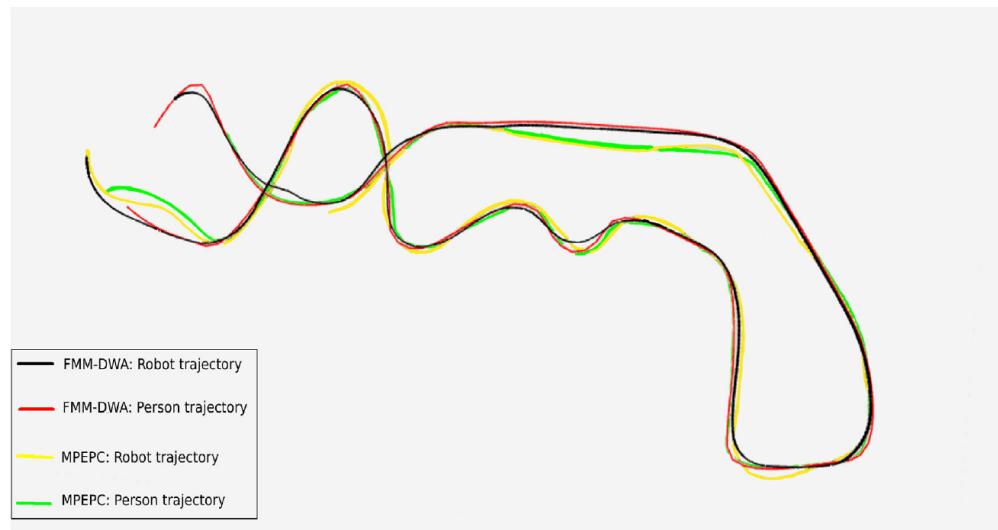


Fig. 2.13: Laser tracking trajectory and Kalman filter trajectory [10]



**Fig. 2.14:** Kalman filter tracking in occluded environments [10]



**Fig. 2.15:** Following trajectory in environments with obstacles [10]

**Table 2.4:** Comparison of performance between FMM-DWA algorithm and MPEPC algorithm [10]

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、物体が少ない綺麗な環境下での実験はされていないため、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や AOA タグの紛失などの課題が考えれる。本プロジェクトでは、雑多な環境下において特定の人を追従し、ロボットの最大直進速度で追従し続けらるよう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

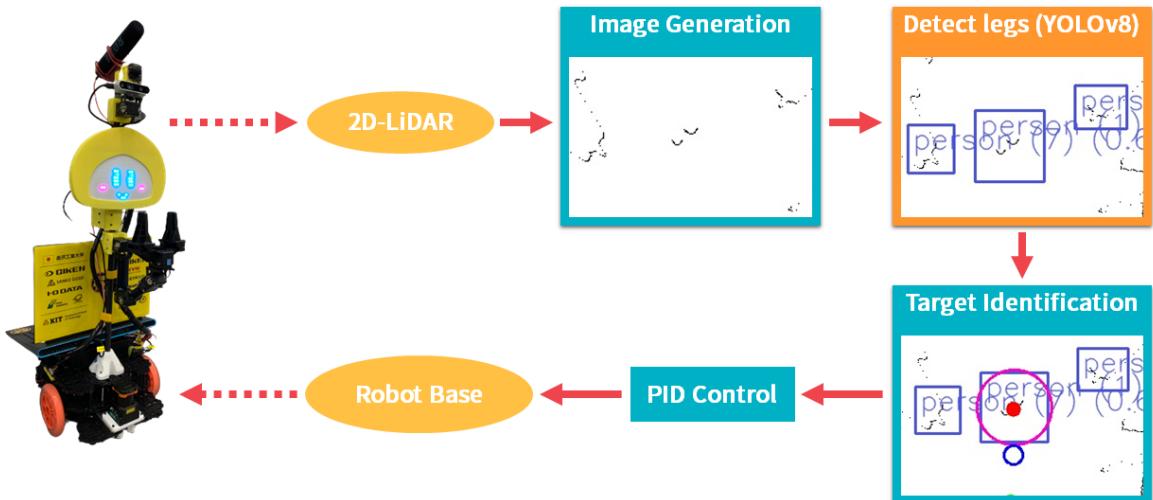
本プロジェクトでは、雑多な環境下で人追従でき、ロボットの最大直進速度で追従できる人追従システムの開発を目的としているため、以下の構成で要求仕様を設定する。使用的するロボットは、2023年に開催された RoboCup 2023で開発したロボットである Happy Edu を使用する。Happy Edu のロボット台車は、ROBOTIS の TurtleBot3 Big Wheel であり、ロボット台車の前方に2D-LiDAR を搭載している。2D-LiDAR は、北表電気株式会社の UTM30-LX を使用しており、制御PCは、NVIDIA GeForce RTX 4070 8GB を搭載しているノート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、追従対象の特定処理を実装している。追従対象の特定後、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Ubuntu22.04を使用し、ミドルウェアにはRobot Operationg 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。

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: 推論結果をもとに、追従対象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。

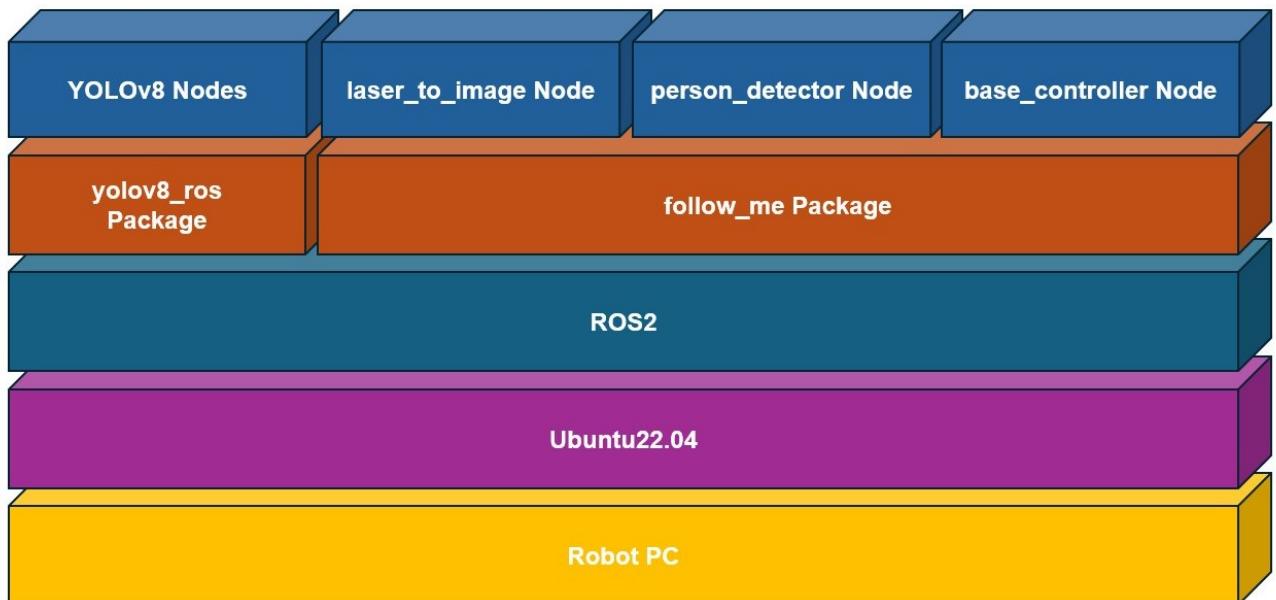
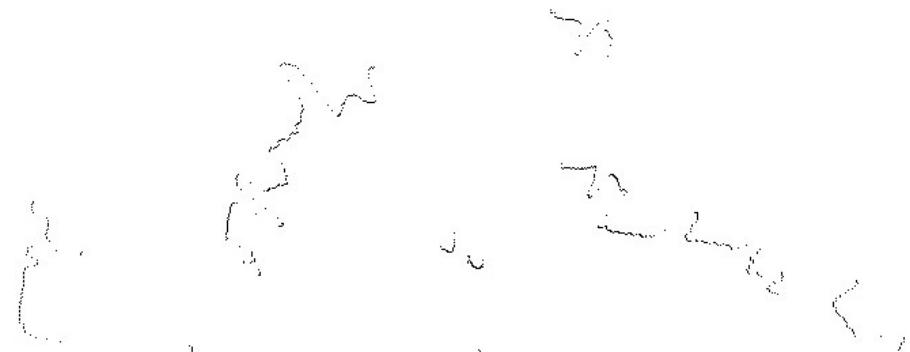


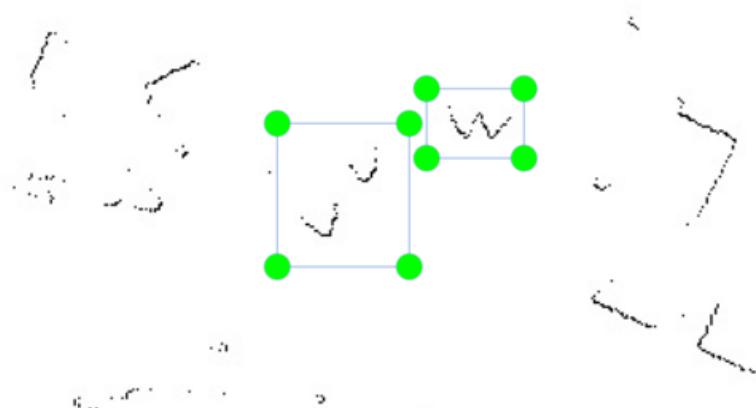
Fig. 3.2: Software configuration diagram

### 3.5 データセットの作成

2D-LiDARのデータをFig. 3.3のように俯瞰画像へ変換し、データセットを作成する。データセットを作成するときの環境は、ロボットが静止した状態において、タイとパンツとワイドパンツを履いた2人が大股や小股などで歩行し、歩行パターンはランダムである。これを5分間行い、俯瞰画像をトピック通信で発信し続け、ROS2 Bagを保存する。ROS2 Bagは、ROS2におけるトピックの保存機能である。ROS2 Bagのデータをから12032枚の画像データを生成し、人の脚部をpersonクラスとしてアノテーションを行った。Fig. 3.4にデータセットの例を示す。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った。



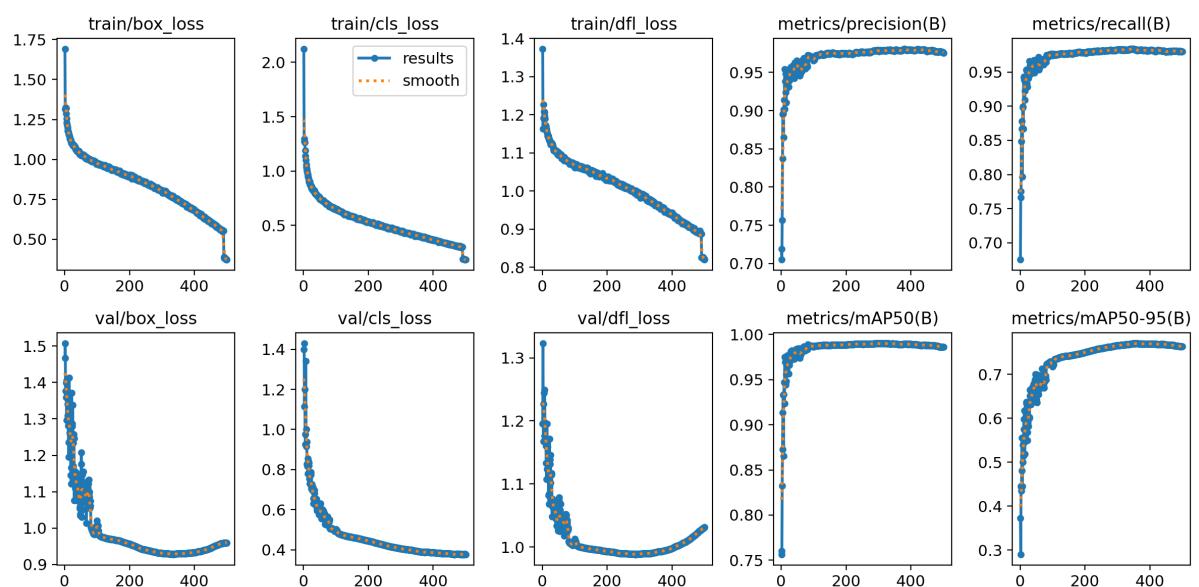
**Fig. 3.3:** Example of an overhead view image



**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エポック以降でを設定した。



**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。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特定の追従目標を追従し続ける。

本プロジェクトで実装した追従目標の特定方法をFig. 3.7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。円型範囲の半径は、ROS2のパラメータ通信で実装しているため、動的な値の変更が可能であり、歩行速度に合わせて円型範囲を拡大縮小することが可能である。

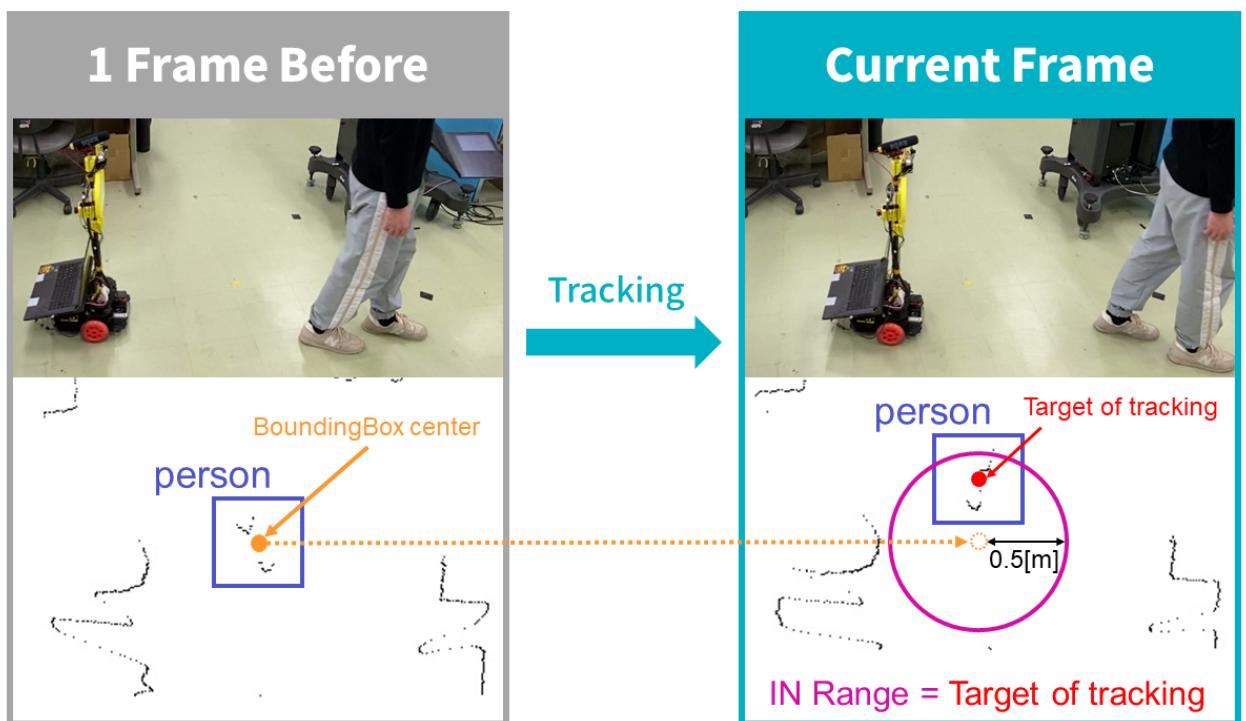


Fig. 3.7: Target identification methods

### 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  $3.0[\text{deg}]$  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  $u_{angle}(t)$  は以下のようになる。

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

$K_{AP}$ 、 $K_{AI}$ 、 $K_{AD}$  は調整パラメータであり、 $K_{AP}$  は 0.005、 $K_{AI}$  は 0.0002、 $K_{AD}$  は 0.0009 で設定している。(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、不足しているまたは過多な制御量を微分制御により調整している。積分制御を常にしていないのは、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。

$$u_{linear}(t) = K_{LP} \cdot L(t) \quad (3.2)$$

$K_{LP}$  は調整パラメータである。(3.2) 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。

## 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、追従できなくなる速度の直前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、北洋電機株式会社の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。

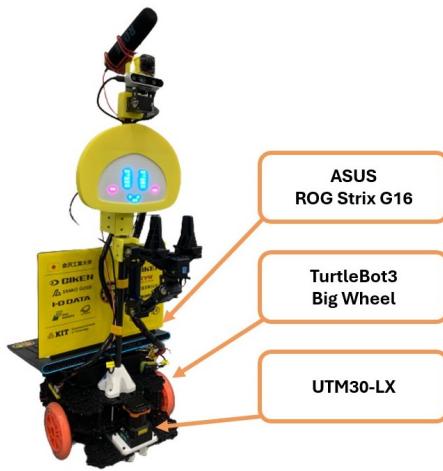


Fig. 4.1: Happy Edu

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]

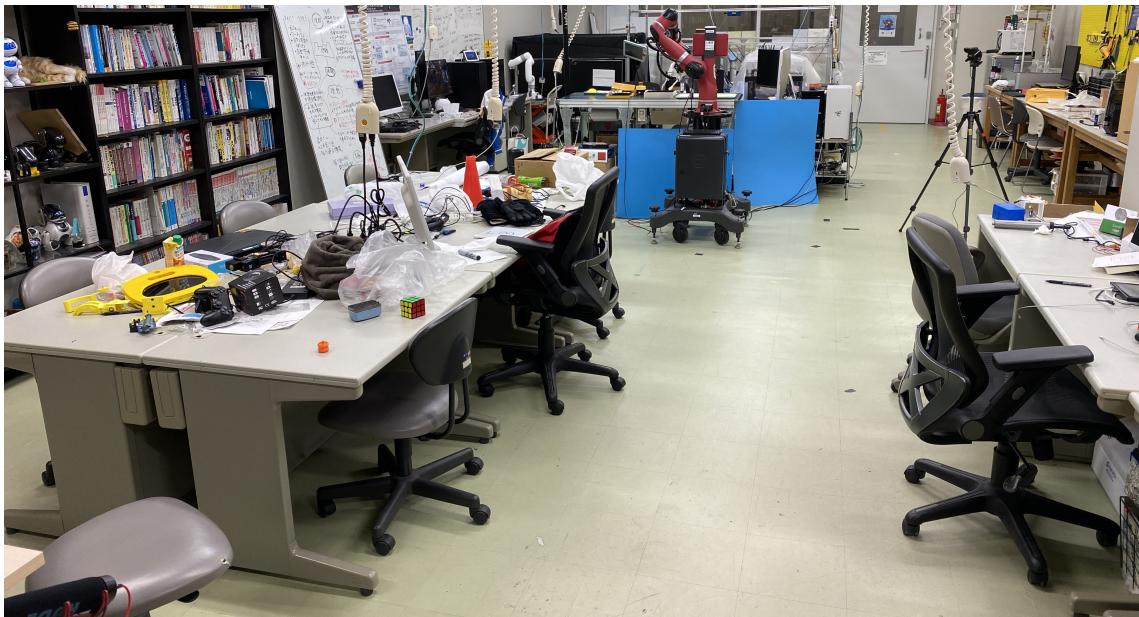
## 4.2.2 追従実験

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。

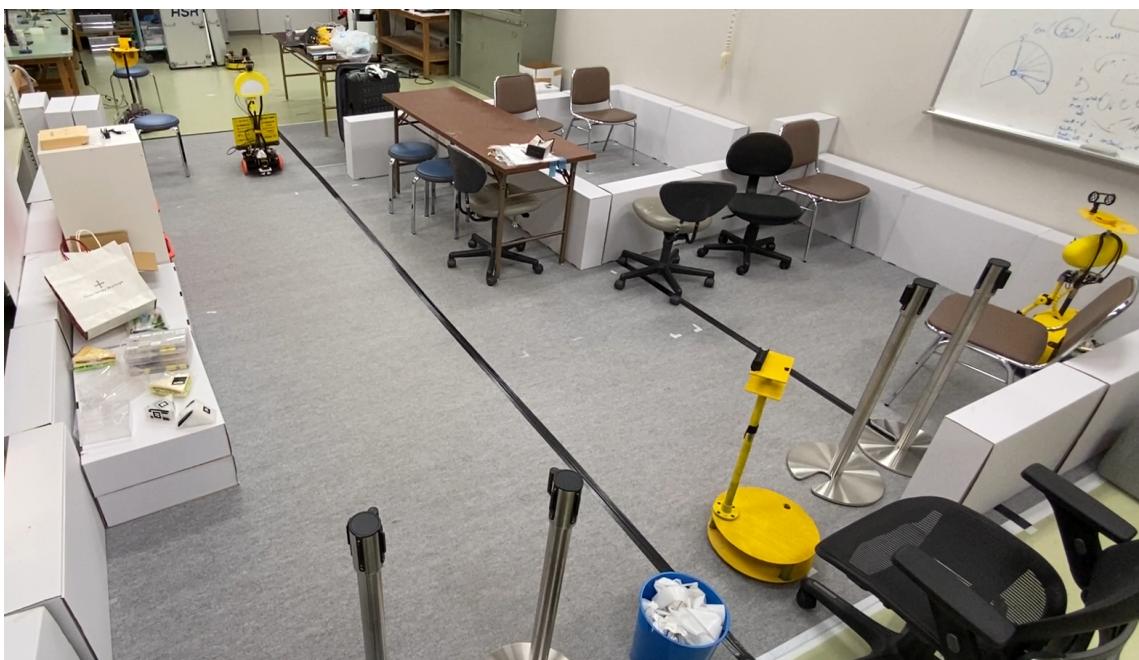
直線経路、曲線経路、直角経路のイメージをそれぞれ Fig. 4.4、Fig. 4.5、Fig. 4.6 に示す。直線経路、曲線経路、直角経路においてそれぞれ 10 回ずつ試行し、各経路での追従成功率を算出する。追従が成功した回数を  $x_{success}$  とした時の追従成功率  $success\ rate$  は以下のようになる。

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

(4.1) 式を用いて各経路での成功率を算出し、それぞれの成功率が 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



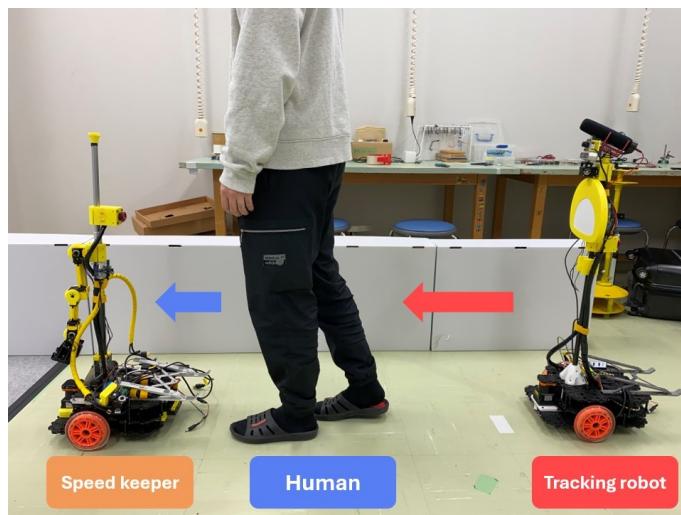
**Fig. 4.6:** Right angle road

### 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

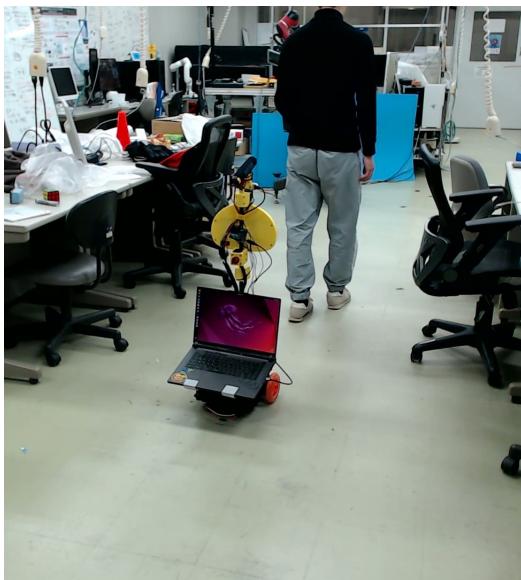
### 4.3.1 追従実験

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。

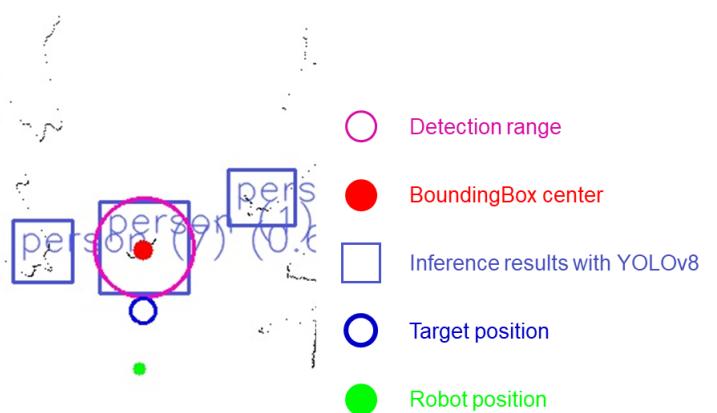
追従実験中の実世界での様子と同時刻における Happy Edu の内部処理の様子をそれぞれ、Fig. 4.9、Fig. 4.10 に示す。Fig. 4.10 から、追従対象者以外の物体を誤検出していることがわかる。しかし、誤検出はしているが、提案手法の「追従目標の特定」により正確に追従対象者を検出していることがわかる。また、Fig. 4.9 より、誤検出された物体は椅子の円柱部分であり、追従対象者の脚部と形状が類似している部分であった。

**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)

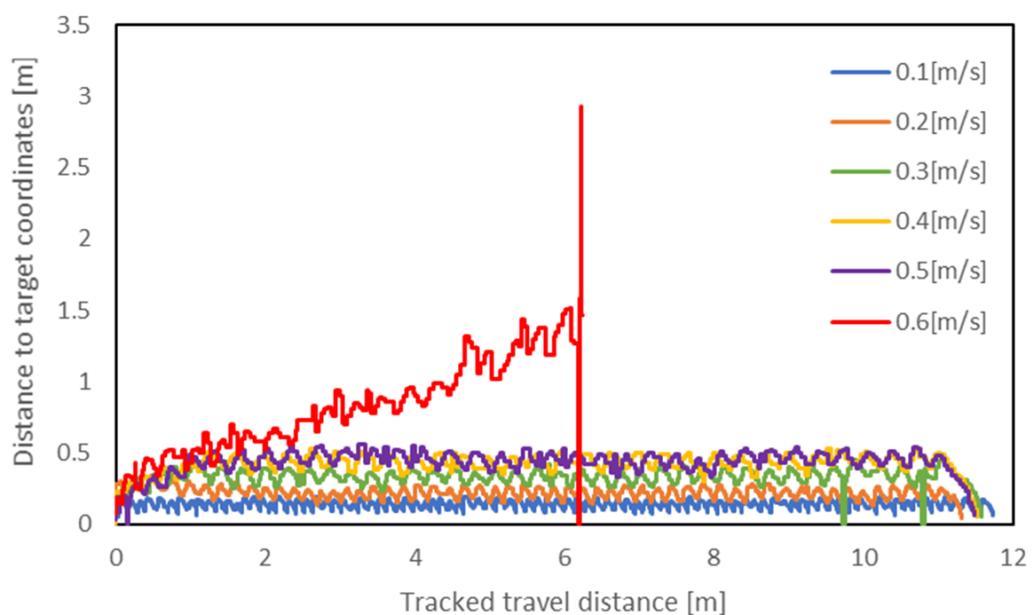
### 4.3.2 最大追従速度実験

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いたが、正確に追従対象者を追従できたのは、「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と考えられる。

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 言

### 5.1 結言

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從性能の検証をした。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、12032 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。

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さらに発展することが必要である。

## 謝 辞

本研究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4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](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: 0.0  
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,  
22                                         '/follow_me/laser_img', 10)  
23         # Subscriber  
24         self.create_subscription(LaserScan, '/scan',  
25                                   self.cloud_to_img_callback, qos_profile_sensor_data)  
26         # OpenCV  
27         self.bridge = CvBridge()  
28         # Parameters  
29         self.declare_parameters(  
30             namespace='',  
31             parameters=[  
32                 ('discrete_size', Parameter.Type.DOUBLE),  
33                 ('max_lidar_range', Parameter.Type.DOUBLE),  
34                 ('img_show_flg', Parameter.Type.BOOL)])  
35         self.add_on_set_parameters_callback(self.param_callback)  
36         # Get parameters  
37         self.param_dict = {}  
38         self.param_dict['discrete_size'] =  
39             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     def output_screen(self):
48         for key, value in self.param_dict.items():
49             self.get_logger().info(f"{key}: {value}")
50
51     def param_callback(self, params):
52         for param in params:
53             self.param_dict[param.name] = param.value
54             self.get_logger().info(f"Set param: {param.name} >>>
55             {param.value}")
56         return SetParametersResult(successful=True)
57
58     def cloud_to_img_callback(self, scan):
59
60         # discrete_factor
61         discrete_factor = 1/self.param_dict['discrete_size']
62         # max_lidar_range と discrete_factor を使って画像サイズを設定する
63         img_size =
64             int(self.param_dict['max_lidar_range']*2*discrete_factor)
65
66         # LiDAR データ
67         maxAngle = scan.angle_max
68         minAngle = scan.angle_min
69         angleInc = scan.angle_increment
70         maxLength = scan.range_max
71         ranges = scan.ranges
72         intensities = scan.intensities
73         #intensities = scan.intensities
74
75         # 距離データの個数を格納
76         num_pts = len(ranges)
77         # 721 行 2 列の空行列を作成
78         xy_scan = np.zeros((num_pts, 2))
79         # 3 チャンネルの白色ブランク画像を作成
80         blank_img = np.zeros((img_size, img_size, 3),
81                           dtype=np.uint8) + 255
82         # ranges の距離・角度からすべての点を XY に変換する処理
83         for i in range(num_pts):
84             # 範囲内かを判定
85             if (ranges[i] > self.param_dict['max_lidar_range']) or
```

```
80         (math.isnan(ranges[i])):  
81             pass  
82     else:  
83         # 角度とXY座標の算出処理  
84         angle = minAngle + float(i)*angleInc  
85         xy_scan[i][1] = float(ranges[i]*math.cos(angle)) #  
86             y座標  
87         xy_scan[i][0] = float(ranges[i]*math.sin(angle)) #  
88             x座標  
89  
90     # ブランク画像にプロットする処理  
91     for i in range(num_pts):  
92         pt_x = xy_scan[i, 0]  
93         pt_y = xy_scan[i, 1]  
94         if (pt_x < self.param_dict['max_lidar_range']) or (pt_x >  
95             -1*(self.param_dict['max_lidar_range']-self.param_dict['discrete_si  
96             or (pt_y < self.param_dict['max_lidar_range']) or  
97             (pt_y > -1 *  
98                 (self.param_dict['max_lidar_range']-self.param_dict['discrete_si  
99                 pix_x = int(math.floor((pt_x +  
100                     self.param_dict['max_lidar_range'] *  
101                     discrete_factor))  
102  
103                 pix_y =  
104                     int(math.floor((self.param_dict['max_lidar_range']  
105                         - pt_y) * discrete_factor))  
106                 if (pix_x > img_size) or (pix_y > img_size):  
107                     print("Error")  
108                 else:  
109                     blank_img[pix_y, pix_x] = [0, 0, 0]  
110  
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()
```

追従対象者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,
24             '/follow_me/image', 10)
25         # Subscriber
26         self.create_subscription(DetectionArray, '/yolo/detections',
27             self.yolo_callback, 10)
28         self.create_subscription(Image, '/yolo/dbg_image',
29             self.img_show, 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
37                 not available ...')
38         # Parameters
39         self.declare_parameters(
40             namespace='',
41             parameters=[
42                 ('target_dist', Parameter.Type.DOUBLE),
```

```

36             ('init_time', Parameter.Type.DOUBLE),
37             ('none_person_flg', Parameter.Type.BOOL),
38             ('target_diff', Parameter.Type.DOUBLE),
39             ('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'] =
78                 data.value.double_value
79             self.get_logger().info(f"Param event: {data.name} >>>
80             {self.param_dict['discrete_size']}")
```

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



```
159         if diff <= self.param_dict['target_range']:
160             target = data
161             break
162         else:
163             target = None
164 #
165         target が None だったら self.before_data を初期化して none_person_flg を Tr
166     if target is None:
167         #self.before_data = [0.0, 0.0, 0.0]
168         #param_bool = Parameter('none_person_flg',
169             Parameter.Type.BOOL, True)
170         #self.set_parameters([param_bool])
171         pass
172     else:
173         # 計算のために距離を保存
174         self.before_data = target
175     return target
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
```

```
def generate_target(self):
    self.target_px.clear()
    # 画像の中心を算出
    robot_x = self.height / 2
    robot_y = self.width / 2
    # 追従目標を選定
    target_person = self.select_target(robot_x, robot_y)
    if not target_person is None:
        result_point = target_person[1]
        self.center_x = target_person[2][0]
        self.center_y = target_person[2][1]
    else:
        result_point = False
    return result_point

def img_show(self, receive_msg):
    self.laser_img = self.bridge.imgmsg_to_cv2(receive_msg,
                                                desired_encoding='bgr8')
    self.height, self.width, _ = self.laser_img.shape[:3]
    # ロボットの座標をプロット
    self.plot_robot_point()
    # 追従対象の検出範囲をプロット
    if not self.param_dict['none_person_flg']:
        self.plot_target_range()
    # person がいるか判定
    if self.person_list:
        # 追従対象を生成
        target_point = self.generate_target()
        # 追従対象がいなければロボット台車を停止する
        if not target_point:
```

```

205             self.target_point.x = 0.0
206             self.target_point.y = 0.0
207             self.point_pub.publish(self.target_point)
208         else:
209             robot_x = self.height / 2
210             robot_y = self.width / 2
211             # 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_si
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
218                 る)
219             self.target_point.x = (robot_x -
220                 target_y)*self.param_dict['discrete_size']
221             self.target_point.y = (robot_y -
222                 target_x)*self.param_dict['discrete_size']
223             # パブリッシュ
224             self.point_pub.publish(self.target_point)
225             # グラフに描画
226             self.plot_target_point()
227             self.plot_person_point()
228
229             # 画像を表示
230             cv2.imshow('follow_me', self.laser_img)
231             cv2.waitKey(1)
232
233
234     def main():
235         rclpy.init()
236         node = PersonDetector()
237         try:
238             rclpy.spin(node)
239         except KeyboardInterrupt:
240             pass
241         node.destroy_node()
242         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',
23                                  self.odom_callback, 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
31              not 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'] =
48              self.get_parameter('i_range').value
49          self.param_dict['lkp'] = self.get_parameter('lkp').value
50          self.param_dict['akp'] = self.get_parameter('akp').value
51          self.param_dict['aki'] = self.get_parameter('aki').value
```

```
44     self.param_dict['akd'] = self.get_parameter('akd').value
45     self.param_dict['target_radius'] = self.get_param() #
46         person_detector からもってくる
47     # Value
48     self.twist = Twist()
49     self.target_angle = 0.0
50     self.target_distance = 0.0
51     self.target_x = 0.0
52     self.target_y = 0.0
53     self.delta_t = 0.0
54     self.robot_angular_vel = 0.0
55     # Output
56     self.output_screen()

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'] =
65                 data.value.double_value
66             self.get_logger().info(f'Param event: {data.name} >>>
67                 {self.param_dict['target_radius']}')

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 def get_param(self):
75     req = GetParameters.Request()
76     req.names = ['target_radius']
77     future = self.srv_client.call_async(req)
78     while rclpy.ok():
79         rclpy.spin_once(self, timeout_sec=0.1)
80         if future.done():
81             break
82     return future.result().values[0].double_value

83 def point_to_angle(self, point):
84     return math.degrees(math.atan2(point.y, point.x))

85
86 def point_to_distance(self, point):
87     distance = math.sqrt(point.x**2 + point.y**2)
88     if point.x < 0.0:
```

```
89             distance = -distance
90     return distance
91
92     def callback(self, receive_msg):
93         self.target_x = receive_msg.x
94         self.target_y = receive_msg.y
95         self.target_distance = self.point_to_distance(receive_msg)
96         self.target_angle = self.point_to_angle(receive_msg)
97
98     def odom_callback(self, receive_msg):
99         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 -
108                                    self.robot_angular_vel)
109
110    # 積分制御量計算
111    def i_control(self, p_term, d_term):
112        value = 0.0
113        diff = (p_term + d_term) - self.robot_angular_vel
114
115        if not diff < self.param_dict['tolerance'] and diff <
116            self.param_dict['i_range']:
117            value =
118                self.param_dict['aki']*self.target_angle*self.delta_t
119
120        return value
121
122    def pid_update(self):
123        # 制御量を計算
124        p_term = self.p_control()
125        d_term = self.d_control(p_term)
126        i_term = self.i_control(p_term, d_term)
127
128        linear_vel = self.param_dict['lkp']*self.target_distance
129        angular_vel = -1*(p_term + i_term + d_term)
130
131        if linear_vel < 0.0:
132            linear_vel = 0.0
133            angular_vel = 0.0
134
135        return linear_vel, angular_vel
136
137    def in_range(self):
```

```
135     result = False
136     if abs(self.target_distance) <=
137         self.param_dict['target_radius']:
138             result = True
139     return result
140
141     def execute(self, rate=100):
142         start_time = time.time()
143         before_time = 0.0
144
145         while rclpy.ok():
146             self.delta_t = time.time() - start_time
147             rclpy.spin_once(self)
148
149             # 許容範囲内外を判
150             if self.in_range():
151                 linear_vel = 0.0
152                 angular_vel = 0.0
153             else:
154                 linear_vel, angular_vel = self.pid_update()
155
156             # 制御量をパブリッシュ
157             self.twist.linear.x = linear_vel
158             self.twist.angular.z = angular_vel
159             self.pub.publish(self.twist)
160
161             # 実験用に目標までの距離と角度をパブリッシュ
162             data = Point()
163             data.x = self.target_distance
164             data.z = self.target_angle
165             self.data_pub.publish(data)
166
167             time.sleep(1/rate)
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     ])
```

---