



令和 5 年度  
卒業研究論文

題目	人追従搬送ロボットのシステム開発
----	------------------

学 科	電子制御工学科
学籍番号	s9123
氏 名	野口 史遠
提 出 日	令和 2 年 2 月 4 日

指導教員	高木 太郎 教授
------	----------



舞鶴工業高等専門学校  
電子制御工学科

# 論文要旨

本研究では、カメラのみを用いた低コストな人追従搬送ロボットの開発を目指し、Intel RealSense D435i を使用した画像処理技術と追従制御アルゴリズムの実装に取り組んだ。従来の搬送ロボットでは、LiDAR や 3D センサーといった高価なセンサーが多用されており、中小企業や農業分野での導入が困難であるという課題があった。本研究では、カメラのみで人の位置を認識し、ロボットを制御するシステムを構築することで、これらの課題を解決することを目的とした。

具体的には、デプスカメラから取得した深度データを用いて、画像上の座標を実空間の物理座標に変換し、人の位置を推定した。また、追従制御には、比例航法 (PN)、修正比例航法 (MPN)、およびゲインスケジューリング下修正比例航法 (GS-MPN) の 3 つのアルゴリズムを実装し、それぞれの性能を比較評価した。さらに、ROS2 を用いたシステム全体の統合を行い、上位層と下位層を連携させたりリアルタイム制御を実現した。

実験では、ロボットが直線および曲線の軌道を持つ目標物を追従する際の精度と滑らかさを評価した。結果として、GS-MPN が最もバランスの取れた性能を示し、急加速を抑えつつ安定した追従を実現した。これにより、従来手法で見られた不安定な動作や追従精度の低下が大幅に改善された。一方で、遮蔽物や複数対象物が存在する場合の課題が残されており、これらの状況に対応するための識別技術や外乱耐性の強化が今後の課題である。

本研究の成果は、コスト効率に優れた人追従ロボットの実現に向けた重要なステップとなり、物流や医療分野など多様な応用への可能性を示唆するものである。今後は、環境適応性の向上や予測制御技術の導入を通じて、さらに高性能で信頼性の高いロボットシステムの開発を目指す。

# 目次

第 1 章	はじめに	1
第 2 章	理論	2
2.1	ROS 2 とは	2
2.2	2 輪ロボットの運動学	2
2.2.1	順運動学	3
2.2.2	逆運動学	4
2.3	PID 制御とローパスフィルタ	4
2.3.1	PID 制御の構造	4
2.3.2	ローパスフィルタの導入	5
2.4	YOLOv5 による物体認識	5
2.5	デプスカメラと距離測定の方法	6
2.5.1	デプスカメラの仕組み	6
2.5.2	デプスデータからの人座標検出	6
2.6	人追従アルゴリズム	8
2.6.1	比例航法 (Proportional Navigation, PN)	8
2.6.2	修正比例航法 (Modified Proportional Navigation, MPN)	8
2.6.3	ゲインスケジューリング修正比例航法 (Gain-Scheduled Modified Proportional Navigation, GS-MPN)	9
第 3 章	システム構成	10
3.1	システム全体図	10
3.2	ハードウェア構成	10
3.2.1	マイコン (STM32 Nucleo Board STM32F446RE)	10
3.2.2	エンコーダー (AMT102-V)	11
3.2.3	深度カメラ (Intel RealSense D435i)	11
3.3	ソフトウェア構成	12
3.3.1	下位レイヤー	12
	エンコーダーライブラリ (encoder.c, encoder.h)	12
	モータードライバライブラリ (motor_driver.c, motor_driver.h)	13
	シリアル通信ライブラリ (serial_lib.c, serial_lib.h)	14
	メインコード (main.c)	15
3.3.2	上位レイヤー	16
	web_socket_node	16
	PID_node	17

	RealSense_node	.....	18
	serial_read_node	.....	20
	serial_send_node	.....	21
	Roboware_node	.....	21
	PN (Proportional Navigation)	.....	22
	MPN (Modified Proportional Navigation)	.....	22
	GS-MPN (Gain-Scheduled Modified Proportional Navigation)	.....	22
第 4 章	実験	.....	24
4.1	実験内容	.....	24
4.1.1	実験条件	.....	24
4.1.2	実験の手順	.....	25
4.2	実験結果及び考察	.....	25
4.2.1	追従性能 (MeanOffset と StdOffset )	.....	27
4.2.2	スムーズさ ( MeanAccel と StdAccel )	.....	27
4.2.3	視野内追従率 ( InFrameRate )	.....	28
4.2.4	総合評価	.....	28
第 5 章	おわりに	.....	29
参考文献		.....	30
謝辞		.....	31
付録		.....	32
A.1	ちちち	.....	32
A.2	つつつつつつつつつつつつつつつつつつつつつつ	.....	32
A.2.1	ててて	.....	32
A.2.2	ととととととととととととととととととととととととととととととと とととととととととと	..... .....	32 33

# 第 1 章 はじめに

近年，生産年齢人口の減少が社会的な課題となっており，ロボット技術の導入による作業の効率化が期待されている．例えば，草刈りロボットや運搬ロボットなど，人手不足を補うための自律ロボットが徐々に普及しつつある．

しかしながら，現在出回っている多くの搬送ロボットには，LiDAR( Light Detection and Ranging ) や 3D センサーなどのセンサーが搭載されているケースが多い．LiDAR は非常に精度の高い距離計測を可能にし，障害物回避や人追従において重要な役割を果たすが，その一方で，導入費用が高いため，特に中小企業や農業の現場では導入が難しい現状がある．このため，より低コストで同様の機能を実現する技術が求められている．

この問題に対処するため，本研究ではカメラのみを用いた人追従型ロボットの開発を目指す．カメラを使用することで，より安価でかつ軽量のシステムの構築が可能であり，初期コストの削減が期待される．また，カメラ技術の進展により，物体認識や追従の精度も向上しており，特定の対象を認識し追従するアルゴリズムも開発されている<sup>1)</sup>．しかしながら実機で検証が行われた例は少ない．

本研究では，カメラを用いた人追従アルゴリズムを構築後，実際の環境下で実験を行い，追従性能の精度を検証する．

## 第 2 章 理論

### 2.1 ROS 2 とは

ロボット技術は産業用から日常生活まで多岐にわたる応用が進んでおり、それに伴い、開発の効率化やシステムの拡張性が求められている。これらの課題に対応するため、オープンソースのロボットフレームワークである ROS ( Robot Operating System ) が開発され、幅広い支持を得ている。ROS は、通信、制御、センサーデータの統合など、ロボット開発における基盤となる機能を提供しており、研究開発において重要な役割を果たしている。

ROS 2 は DDS ( Data Distribution Service ) を通信基盤として採用している。これにより、ノード間通信のリアルタイム性が向上し、複数のロボットやセンサーから成る分散システムの構築が容易となった<sup>2)</sup>。特に、産業用ロボットや自動運転車のように、即時性が求められるシステムにおいて、その利点が顕著である。

さらに、クロスプラットフォーム対応も ROS 2 の特徴の一つである。ROS 1 は主に Linux 環境での動作を前提としていたが、ROS 2 では Windows や macOS など複数のプラットフォームで動作するよう設計されており、開発環境の柔軟性が大幅に向上した<sup>3)</sup>。これにより、多様なハードウェア環境への適応が可能となり、利用者の裾野が広がった。

これらの特徴により、ROS 2 は、研究開発だけでなく商業用途にも適したフレームワークとして注目を集めている。本研究においても、ROS 2 を採用することで、効率的かつ信頼性の高いロボットシステムの開発を目指す。

### 2.2 2 輪ロボットの運動学

2 輪ロボットは、シンプルな構造でありながら、高い機動性を持つため、広く研究や産業に利用されている。その運動学は、ロボットの速度制御や経路計画を行う上で不可欠な要素である。本節では、2 輪ロボットの運動学および逆運動学を導出する。

- $v$  : ロボットの並進速度 [m/s]
- $\omega$  : ロボットの角速度 [rad/s]
- $r$  : 車輪の半径 [m]
- $L$  : 左右車輪間の距離 [m]
- $\dot{\theta}_R, \dot{\theta}_L$  : 右車輪および左車輪の角速度 [rad/s]

とする。

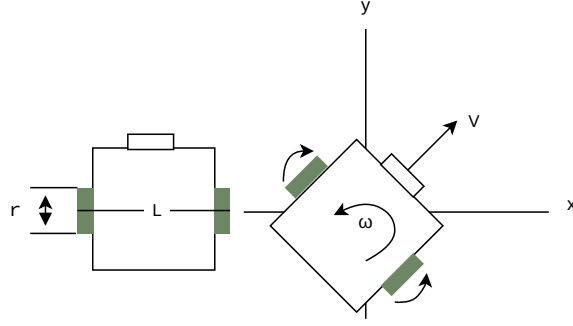


図 2.1 : 2 輪ロボットのモデル図

### 2.2.1 順運動学

順運動学では，各車輪の角速度  $\dot{\theta}_R$  および  $\dot{\theta}_L$  から，ロボットの並進速度  $v$ ，角速度  $\omega$ ，および自己位置を算出する．自己位置の計算は，ロボットの進行方向と回転を考慮して行う．

ロボットの並進速度  $v$  と角速度  $\omega$  は，次式で表される．

$$v = \frac{r}{2}(\dot{\theta}_R + \dot{\theta}_L) \quad (2.1)$$

$$\omega = \frac{r}{L}(\dot{\theta}_R - \dot{\theta}_L) \quad (2.2)$$

これに基づき，ロボットの自己位置  $(x, y)$  と向き  $\theta$  の時間変化を次のように表す．

$$\dot{x} = v \cos \theta \quad (2.3)$$

$$\dot{y} = v \sin \theta \quad (2.4)$$

$$\dot{\theta} = \omega \quad (2.5)$$

これらの微分方程式を離散化することで，時刻  $t_k$  における位置と向きを計算することができる．離散化した式は次のようになる．

$$x_{k+1} = x_k + v_k \cos \theta_k \Delta t \quad (2.6)$$

$$y_{k+1} = y_k + v_k \sin \theta_k \Delta t \quad (2.7)$$

$$\theta_{k+1} = \theta_k + \omega_k \Delta t \quad (2.8)$$

ここで， $\Delta t$  は計算ステップの時間間隔を示す．

順運動学を用いることで，各車輪の角速度を基にロボットの現在位置と進行方向をリアルタイムで更新することが可能である．



### 2.2.2 逆運動学

逆運動学では，ロボットの並進速度  $v$  と角速度  $\omega$  から，車輪の角速度  $\dot{\theta}_R$  および  $\dot{\theta}_L$  を求める．ロボットの基本的なモデルを図 2.1 に示す．

ロボットの並進速度  $v$  および角速度  $\omega$  は，車輪の角速度  $\dot{\theta}_R$  および  $\dot{\theta}_L$  により次式で表される．

$$v = \frac{r}{2}(\dot{\theta}_R + \dot{\theta}_L) \quad (2.9)$$

$$\omega = \frac{r}{L}(\dot{\theta}_R - \dot{\theta}_L) \quad (2.10)$$

式 (2.9) および式 (2.10) を連立すると，次のように導出できる．

$$\dot{\theta}_R = \frac{1}{r}(v + \frac{L}{2}\omega) \quad (2.11)$$

$$\dot{\theta}_L = \frac{1}{r}(v - \frac{L}{2}\omega) \quad (2.12)$$

これらの式により，目標速度  $v$  と  $\omega$  が与えられた場合に，車輪の角速度  $\dot{\theta}_R$  と  $\dot{\theta}_L$  を算出し，モータドライバを介して制御することでロボットの移動を実現できる．

## 2.3 PID 制御とローパスフィルタ

PID 制御 (Proportional-Integral-Derivative Control) は，制御工学における最も基本的かつ広く使用されているフィードバック制御の手法である．本節では，PID 制御の基本構造とローパスフィルタについて説明する．

### 2.3.1 PID 制御の構造

PID 制御は，目標値と現在値の偏差を基に制御量を算出し，システムを目標値へ収束させる手法である．制御量  $u(t)$  は次式で定義される：

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{de(t)}{dt} \quad (2.13)$$

ここで，

- $e(t)$ ：目標値と現在値の偏差
- $K_P$ ：比例ゲイン (P 制御)
- $K_I$ ：積分ゲイン (I 制御)
- $K_D$ ：微分ゲイン (D 制御)

各項の役割は以下の通りである：

- 比例制御 (P 制御)：偏差に比例した制御量を生成し，応答の速さを向上させる．
- 積分制御 (I 制御)：偏差を累積することで，定常偏差を排除する．
- 微分制御 (D 制御)：偏差の変化率を基に，過渡応答の振動を抑制する．

図 2.2 に PID 制御器の構成図を示す．本研究では，この PID 制御器を用いてロボットの各車輪の速度を制御している．

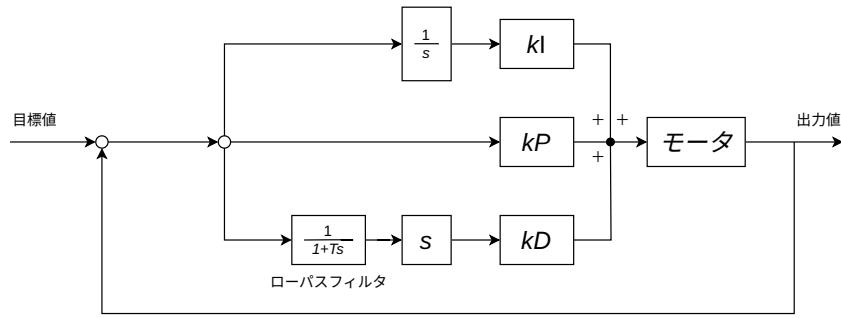


図 2.2 : PID 制御器の構成図

### 2.3.2 ローパスフィルタの導入

微分制御 (D 制御) は高周波ノイズに敏感であり, そのまま使用するとシステムの安定性を損なう可能性がある. この問題を軽減するために, 微分項にローパスフィルタを適用することが一般的である.

ローパスフィルタを適用した微分制御項は次式で表される:

$$D(s) = \frac{K_D s}{1 + T s} \quad (2.14)$$

ここで,

- $T$ : ローパスフィルタの時定数

フィルタを導入することで, 高周波ノイズが減衰され, システムの安定性が向上する.

## 2.4 YOLOv5 による物体認識

本研究では, 人の認識に YOLOv5 (You Only Look Once Version 5) を使用する. YOLO はリアルタイムで物体を検出するアルゴリズムであり, 1 枚の画像に対して物体のクラスとその位置を同時に出力する特徴を持つ. 単一のニューラルネットワークを用いて, 画像内の物体のバウンディングボックスとクラスラベルを直接予測する<sup>4)</sup>. そのため他の物体検出モデル (例えば SSD や Faster R-CNN) と比較して, 高速な推論速度と高い精度を両立している. このため, リアルタイム性が要求されるロボットシステムにおいて有用である.

本研究では, YOLOv5 を用いてデプスカメラ D435i から得られた RGB 画像内の人を認識する. YOLOv5 は以下のステップで物体を検出する:

1. RGB 画像を入力し, YOLOv5 のバックボーンにより特徴を抽出する.
2. ネットでスケールを調整し, 特徴を統合する.
3. ヘッドでバウンディングボックス, クラスラベル (人), および信頼度スコアを出力する.

出力されたバウンディングボックスは, ピクセル座標で記述されており, これをデプスカメラのデプスマップと組み合わせることで, 物理座標を算出する.

## 2.5 デプスカメラと距離測定の方法

デプスカメラは，対象物までの距離を計測するためのデバイスであり，ロボットの環境認識や障害物回避，物体追跡などの応用において重要な役割を果たす．本研究では，インテル® RealSense™ デプスカメラ D435i を使用し，カメラから取得したデプスデータを基に人追従アルゴリズムを実現する．本節では，デプスカメラの仕組みおよび D435i の特徴について説明する．

### 2.5.1 デプスカメラの仕組み

デプスカメラは，主に以下の 3 つの方式で距離を測定する：

- ステレオカメラ方式：2 つのカメラで撮影した画像から視差（パララックス）を計算し，物体までの距離を推定する．
- アクティブ赤外線方式：赤外線を投射し，反射光のパターンを解析することで深度情報を取得する．
- Time-of-Flight（ToF）方式：光の飛行時間を測定し，物体までの距離を直接計算する．

RealSense D435i は，ステレオカメラ方式を基本とし，アクティブ赤外線方式を補助的に使用する．これにより，低照度環境やテクスチャの少ない物体に対しても安定した距離測定が可能となっている．

### 2.5.2 デプスデータからの人座標検出

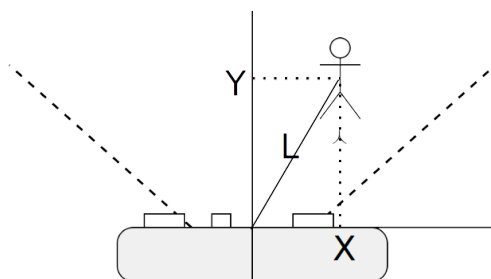


図 2.3：デプスカメラを用いた座標変換モデル

デプスカメラで取得したデータを用いて，画像上のピクセル座標を実空間の物理座標に変換することで，人の位置を検出する．図 2.3 に示すように，ピクセル座標系における対象物の位置を，カメラ座標系に変換する手順を以下に述べる．

デプスカメラのデプスマップは，各ピクセルに深度情報  $L$ （対象物までの斜距離）を格納している．この情報を用いて，画像座標系におけるピクセル座標  $(u, v)$  をカメラ座標系  $(X_c, Y_c)$  に変換する．なお，本実験では上下方向（ $Z$  軸）は考慮しない．

変換には以下の式を使用する：

$$X_c = \frac{(u - c_x) \cdot L}{f_x} \quad (2.15)$$

$$Y_c \approx L \quad (2.16)$$

ここで、 $u$  および  $v$  は画像座標系のピクセル位置、 $c_x$  はカメラの光学中心（水平方向の画面中心）のピクセル位置、 $f_x$  はカメラの焦点距離（ピクセル単位）、 $L$  はデプスマップから取得される深度値（対象物までの斜距離）である。

次に、カメラ座標系における2次元位置  $(X_c, Y_c)$  をロボット座標系  $(X_r, Y_r)$  に変換する。この変換は、カメラとロボット間の座標系の相対的な位置関係を記述する回転行列  $R$  と平行移動ベクトル  $T$  を用いることで行われる。

ロボット座標系への変換は、次の行列式で表される：

$$\begin{bmatrix} X_r \\ Y_r \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ 1 \end{bmatrix} \quad (2.17)$$

ここで、

- $R$  : カメラ座標系からロボット座標系への回転行列
- $T$  : カメラ座標系からロボット座標系への平行移動ベクトル

本実験では、カメラがロボットに平行に取り付けられているため、回転行列  $R$  は単位行列として扱う。また、カメラがロボットの旋回中心から 200 mm (0.2 m) 前方に取り付けられているため、平行移動ベクトル  $T$  は以下のように与えられる：

$$T = \begin{bmatrix} 0.0 \\ 0.2 \end{bmatrix} \quad (2.18)$$

実際に計算例を示す。以下のパラメータを仮定する：

- デプスカメラの焦点距離： $f_x = 378.09$  [px]
- デプスカメラの光学中心： $c_x = 319.39$  [px]
- デプスマップからの深度値： $L = 2.5$  [m]
- 対象物の水平ピクセル座標： $u = 400$  [px]
- カメラ位置のオフセット： $T_x = 0.2$  [m]

まず、画像座標系でのズレ  $\Delta u$  を計算する：

$$\Delta u = u - c_x = 400 - 319.39 = 80.61 \text{ [px]} \quad (2.19)$$

次に、カメラ座標系での水平座標  $X_c$  を求める：

$$X_c = \frac{\Delta u \cdot L}{f_x} = \frac{80.61 \cdot 2.5}{378.09} \approx 0.534 \text{ [m]} \quad (2.20)$$

奥行き  $Y_c$  を近似的に求める：

$$Y_c \approx L = 2.5 \text{ [m]} \quad (2.21)$$

最後に，ロボット座標系での位置  $(X_r, Y_r)$  を計算する：

$$X_r = X_c + T_x = 0.534 + 0 = 0.534 \text{ [m]} \quad (2.22)$$

$$Y_r = Y_c + T_y = 2.5 + 0.2 = 4.5 \text{ [m]} \quad (2.23)$$

この計算により，目標物のロボット座標系における位置が得られる．

## 2.6 人追従アルゴリズム

### 2.6.1 比例航法 (Proportional Navigation, PN)

比例航法は，目標物（人物）の追従において，距離と偏差（横方向のズレ）を用いて制御信号を生成する基本的なアルゴリズムである．以下に，本実験で使用する比例航法の式を示す．

$$V = k_v \cdot (\text{person\_distance} - 1.0) \quad (2.24)$$

$$\omega = k_\omega \cdot \frac{\text{person\_offset}}{\max(\text{person\_distance}, 1.0)} \quad (2.25)$$

- $k_v$ ：並進速度の比例ゲイン
- $k_\omega$ ：角速度の比例ゲイン
- $\text{person\_distance}$ ：ロボットと目標間の距離
- $\text{person\_offset}$ ：横方向の偏差

### 2.6.2 修正比例航法 (Modified Proportional Navigation, MPN)

比例方法では，大きな変化に対して大きな出力となり，ロボットの動きが激しくなってしまう．そこで修正比例航法では，偏差の変化率（偏差角速度）を考慮することで，ロボットの動作をより滑らかにする．偏差角速度は以下の式で計算される．

$$\dot{\text{offset}} = \frac{\text{person\_offset} - \text{previous\_offset}}{\Delta t} \quad (2.26)$$

角速度  $\omega$  は以下の式で修正される．

$$\omega = k_\omega \cdot \frac{\text{person\_offset}}{\max(\text{person\_distance}, 1.0)} + \lambda \cdot \dot{\text{offset}} \quad (2.27)$$

ここで，

- $\lambda$ ：偏差角速度に対するゲイン
- $\Delta t$ ：制御周期
- $\text{previous\_offset}$ ：前回の偏差値

これにより，急激な方向転換や不安定な動作が軽減され，滑らかな動作が可能となる．しかしながら，追従性能がやや劣る．

### 2.6.3 ゲインスケジューリング修正比例航法 (Gain-Scheduled Modified Proportional Navigation, GS-MPN)

ゲインスケジューリング修正比例航法では，偏差角速度に基づき動的なゲインスケジューリングを導入することで，柔軟な制御を実現する．

動的微分ゲイン  $k_d$  は以下の式で定義される．

$$k_d = \lambda_d \cdot \frac{1 - \exp(-a \cdot |\dot{\text{offset}}|)}{1 + \exp(-a \cdot |\dot{\text{offset}}|)} \quad (2.28)$$

- $\lambda_d$ ：微分ゲインの最大値
- $a$ ：微分ゲインの変化速度を調整するパラメータ

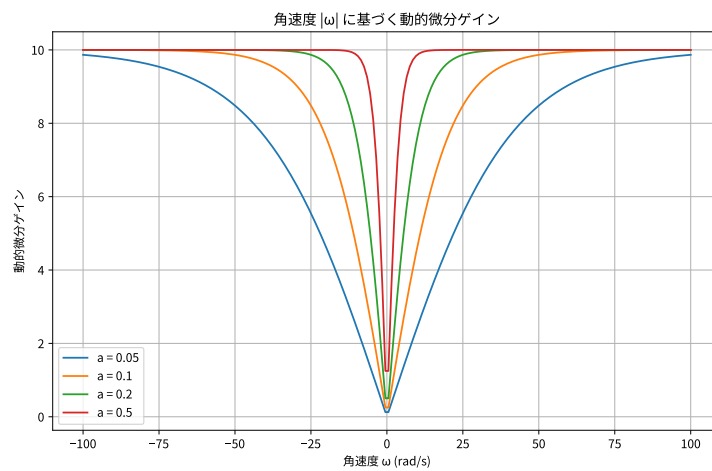


図 2.4：動的微分ゲイン

また，図 2.4 に角速度に基づく動的微分ゲインのグラフを示す．

角速度  $\omega$  は次式で計算される．

$$\omega = k_\omega \cdot \frac{\text{person\_offset}}{\max(\text{person\_distance}, 1.0)} + k_d \cdot \dot{\text{offset}} \quad (2.29)$$

## 第 3 章 システム構成

### 3.1 システム全体図

本研究で構築したシステムの全体図を図 3.1 に示す．本システムは，下位レイヤーと上位レイヤーの 2 つの主要な構成要素からなる．

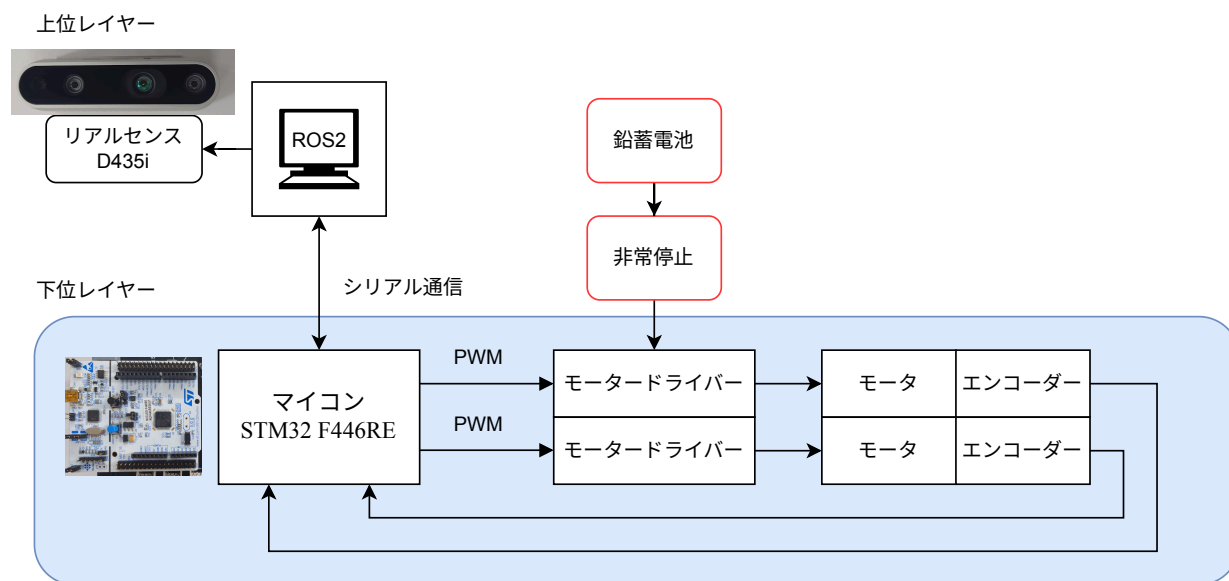


図 3.1 : システム全体図

### 3.2 ハードウェア構成

本システムのハードウェア構成について，以下に各要素を説明する．

#### 3.2.1 マイコン (STM32 Nucleo Board STM32F446RE)

本システムでは，STM32 Nucleo Board STM32F446RE マイコンボードを使用している．STM32 F446RE は高性能な ARM Cortex-M4 プロセッサを搭載しており，以下の特徴を持つ．

- 最大 180 MHz のクロック速度
- タイマーをエンコーダモードに設定可能
- 開発/評価ボードであり入手性が高い

本マイコンは，モーター駆動，エンコーダーのデータ取得を担当しており，ロボットの下位制御レイヤーを実現している．

### 3.2.2 エンコーダー（AMT102-V）

AMT102-V エンコーダーを採用し、モーターの回転角速度および回転位置を計測している。図 3.2 に使用するエンコーダーを示す。このエンコーダーは最大分解能は 2048 PPR( Pulse Per Revolution ) であり、非接触方式である。エンコーダーからの信号はマイコンで処理され、車輪の速度制御や自己位置推定に使用する。

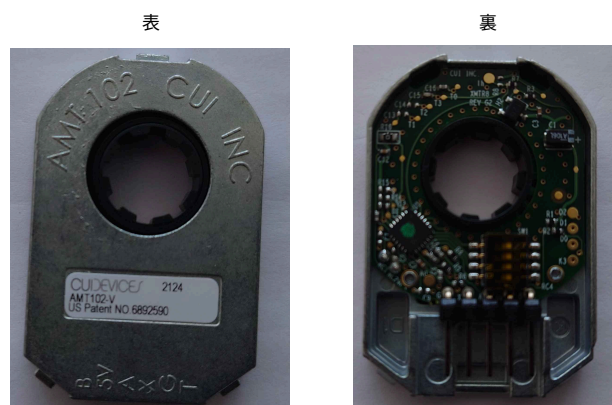


図 3.2 : AMT102-V

### 3.2.3 深度カメラ（Intel RealSense D435i）

深度カメラとして Intel RealSense D435i を採用している。D435i は、ステレオカメラ方式に基づく高精度な距離計測を特徴とし、以下のような仕様を持つ。

- ・ 最大測定距離：0.1 ～ 10 メートル
- ・ フレームレート：最大 90 FPS
- ・ 視野角：水平 87°，垂直 58°
- ・ IMU（慣性計測ユニット）搭載

本システムでは、D435i から取得した深度データを用いて目標（人）の位置を検出し、追従アルゴリズムに利用している。

## Intel RealSense D435i

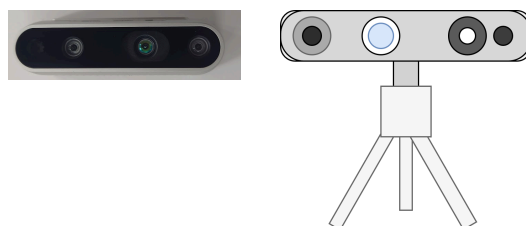


図 3.3 : RealSense



D435i の内部パラメータ（焦点距離と光学中心）は、以下の Python コードを使用して計測した。

```
1 import pyrealsense2 as rs
2
3 # パイプラインのセットアップ RealSense
4 pipeline = rs.pipeline()
5 config = rs.config()
6 config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
7 pipeline.start(config)
8
9 # 深度ストリームのプロファイルからカメラパラメータを取得
10 profile = pipeline.get_active_profile()
11 depth_stream = profile.get_stream(rs.stream.depth) # 深度ストリーム
12 intrinsics = depth_stream.as_video_stream_profile().get_intrinsics()
13
14 # 内部パラメータを取得
15 fx = intrinsics.fx # 焦点距離 (f_x)
16 fy = intrinsics.fy # 焦点距離 (f_y)
17 cx = intrinsics.ppx # 光学中心座標 x (c_x)
18 cy = intrinsics.ppy # 光学中心座標 y (c_y)
19
20 print(f"焦点距離: fx={fx}, fy={fy}")
21 print(f"光学中心: cx={cx}, cy={cy}")
22
23 # 使用後はパイプラインを停止
24 pipeline.stop()
```

Listing 1: RealSense 内部パラメータの取得

このコードを実行した結果、D435i の焦点距離と光学中心が以下の通り得られた。

- 焦点距離:  $f_x = 378.09$  [pixel],  $f_y = 378.09$  [pixel]
- 光学中心:  $c_x = 319.39$  [pixel],  $c_y = 237.10$  [pixel]

これらの内部パラメータは、ピクセル座標系からカメラ座標系への変換（式 (2.15), (2.16)）で使われる。

### 3.3 ソフトウェア構成

本システムのソフトウェア構成について、以下に各要素を説明する。

#### 3.3.1 下位レイヤー

本システムの下位レイヤーソフトウェアでは、モーター制御、エンコーダーのデータ処理、およびシリアル通信を実現するために自作ライブラリを使用している。開発環境は CubeIED(ver.1.16.1) である。以下に、各ライブラリの詳細について説明する。

エンコーダーライブラリ (encoder.c, encoder.h)

エンコーダーライブラリは、ロータリーエンコーダー (AMT102-V) の値を取得し、角速度や回転数を計算するための機能をもつ。このライブラリでは、エンコーダーから得られる値を高精度に利用

するために4通倍処理を行い、モーターの速度制御に使用している。以下に、一部ライブラリコードを示す。

```
1 // エンコーダーカウント値の取得
2 int Encoder_Read(Encoder* encoder)
3 {
4     int16_t count = (int16_t)(__HAL_TIM_GET_COUNTER(encoder->htim) - TIMER_MAX_COUNT /
5                             2);
6     return count;
7 }
8 void Encoder_Interrupt(Encoder* encoder, EncoderData* encoder_data)
9 {
10     int count = Encoder_Read(encoder);
11
12     encoder_data->count = count;
13     encoder_data->rot = count / (double)encoder->ppr;
14     encoder_data->deg = encoder_data->rot * 360.0;
15     encoder_data->distance = encoder_data->rot * (PI * encoder->diameter);
16
17     encoder_data->rps = (encoder_data->rot - encoder->before_rot) / (encoder->period *
18                             0.001);
19     encoder_data->velocity = encoder_data->rps * PI * encoder->diameter;
20     encoder->before_rot = encoder_data->rot;
21 }
```

Listing 2: エンコーダーカウント値の取得

また、このライブラリではカウント値のリセット機能も提供しており、特定の条件下でカウント値をゼロに戻すことが可能である。

### モータードライバライブラリ (motor\_driver.c, motor\_driver.h)

モータードライバライブラリは、DC モーターの速度と方向を制御するための機能をもつ。このライブラリでは、PWM 信号を用いてモーターを駆動し、回転方向の切り替えや速度制御を行っている。以下に、一部ライブラリコードを示す。

```
1 // 初期化関数
2 void MotorDriver_Init(MotorDriver* motor, TIM_HandleTypeDef* htimA, uint32_t channelA,
3                       TIM_HandleTypeDef* htimB, uint32_t channelB) {
4     motor->htimA = htimA;
5     motor->channelA = channelA;
6     motor->htimB = htimB;
7     motor->channelB = channelB;
8     // PWM 開始
9     HAL_TIM_PWM_Start(htimA, channelA);
10    HAL_TIM_PWM_Start(htimB, channelB);
11 }
12 // 速度設定関数
13 void MotorDriver_setSpeed(MotorDriver *motor, int speed) {
14     int pwm_value;
15     if (speed > 100) speed = 99; // ブーストラップ回路に対応
16     if (speed < -100) speed = -99; // ブーストラップ回路に対応
17
18     if (speed > 0) {
19         pwm_value = (speed * __HAL_TIM_GET_AUTORELOAD(motor->htimA)) / 100;
20     }
21 }
```

```

19     __HAL_TIM_SET_COMPARE(motor->htimA, motor->channelA, pwm_value);
20     __HAL_TIM_SET_COMPARE(motor->htimB, motor->channelB, 0);
21 } else {
22     pwm_value = (-speed * __HAL_TIM_GET_AUTORELOAD(motor->htimA)) / 100;
23     __HAL_TIM_SET_COMPARE(motor->htimA, motor->channelA, 0);
24     __HAL_TIM_SET_COMPARE(motor->htimB, motor->channelB, pwm_value);
25 }
26 }

```

Listing 3: モーターの速度設定 (motor\_driver.c)

## シリアル通信ライブラリ (serial\_lib.c, serial\_lib.h)

シリアル通信ライブラリは、PC や上位システムとのデータ通信を実現するために設計されている。このライブラリでは、固定長および可変長データの送受信をサポートしており、効率的かつ安全な通信を実現している。以下に、可変長データの送信および受信関数を示す。

```

1  // 可変長データの送信関数
2  void Serial_SendData(UART_HandleTypeDef *huart, int16_t *data, uint8_t data_count) {
3      uint8_t buffer_size = 2 + data_count * 2;
4      uint8_t *buffer = (uint8_t *)malloc(buffer_size);
5
6      buffer[0] = SERIAL_HEADER1;
7      buffer[1] = SERIAL_HEADER2;
8
9      for (uint8_t i = 0; i < data_count; i++) {
10         buffer[2 + i * 2] = (data[i] >> 8) & 0xFF;
11         buffer[3 + i * 2] = data[i] & 0xFF;
12     }
13
14     HAL_UART_Transmit(huart, buffer, buffer_size, HAL_MAX_DELAY);
15     free(buffer);
16 }
17 // 可変長データの受信関数
18 uint8_t Serial_ReceiveData(UART_HandleTypeDef *huart, int16_t *data, uint8_t data_count) {
19     uint8_t buffer_size = 2 + data_count * 2;
20     uint8_t *buffer = (uint8_t *)malloc(buffer_size);
21
22     if (HAL_UART_Receive(huart, buffer, buffer_size, HAL_MAX_DELAY) == HAL_OK) {
23         if (buffer[0] == SERIAL_HEADER1 && buffer[1] == SERIAL_HEADER2) {
24             for (uint8_t i = 0; i < data_count; i++) {
25                 data[i] = (buffer[2 + i * 2] << 8) | buffer[3 + i * 2];
26             }
27             free(buffer);
28             return 1; // 正常受信
29         }
30     }
31     free(buffer);
32     return 0; // エラー
33 }

```

Listing 4: 可変長データの送受信関数 (serial\_lib.c)

この関数では、データを指定された形式に従ってパケット化し、シリアル通信で送信する。先頭にヘッダー（‘SERIAL\_HEADER1’ と ‘SERIAL\_HEADER2’）を追加し、データ部分は 16 ビット

整数をビッグエンディアン形式で格納する．また，送信後に動的に確保したメモリを解放することで，メモリリークを防止している．

受信関数では，指定された形式に従って受信したデータをデコードし，データバッファに格納する．受信データのヘッダーを検証し，データが正しい形式であることを確認した後，各データを 16 ビット整数として復元する．受信データが無効の場合，エラーコードを返すことで通信エラーを適切にハンドリングする．

### メインコード (main.c)

メインコードでは，自作のエンコーダー，モータードライバ，シリアル通信ライブラリを統合し，システム全体の制御を実現している．このコードは，PC からの制御信号を受信してモーターの速度を設定するとともに，エンコーダーから取得した速度データを PC に送信する役割を果たす．

この制御ループでは，まずシリアル通信ライブラリの ‘Serial\_ReceiveData’ 関数を使用して PC からの制御信号を受信する．受信した制御信号は右車輪および左車輪の目標速度を表しており，それぞれ ‘controlSignalRight’ と ‘controlSignalLeft’ に格納される．これらの速度データは，モータードライバライブラリの ‘MotorDriver\_setSpeed’ 関数を用いて設定される．

次に，エンコーダーデータの送信を行う．10ms ごとにタイマーの値を確認し，タイミングが来た場合にエンコーダーライブラリの ‘Encoder\_Interrupt’ 関数を使用して速度データを更新する．更新された速度データは，シリアル通信ライブラリの ‘Serial\_SendData’ 関数を用いて PC に送信される．

このように，メインコードは上位システムとの通信，モーター制御，およびエンコーダーデータの送信を連携させ，ロボット全体のリアルタイム制御を実現している．

以下に，一部実装コードを示す．

```
1 // メイン制御ループ
2 while (1)
3 {
4     /* PC からの制御信号を受信 */
5     int16_t receivedData[2];
6     if (Serial_ReceiveData(&huart2, receivedData, 2))
7     {
8         controlSignalRight = receivedData[0];
9         controlSignalLeft = receivedData[1];
10
11         MotorDriver_setSpeed(&motorRight, -1 * controlSignalRight);
12         MotorDriver_setSpeed(&motorLeft, -1 * controlSignalLeft);
13     }
14
15     /* エンコーダーデータの送信 ( 10[ms] ごとに送信) */
16     if (HAL_GetTick() - lastSendTime >= 10)
17     {
18         lastSendTime = HAL_GetTick();
19
20         /* エンコーダーの速度データを更新 */
21         Encoder_Interrupt(&encoderRight, &encoderDataRight);
22         Encoder_Interrupt(&encoderLeft, &encoderDataLeft);
23
24         /* エンコーダー速度を送信 */
25         int16_t feedbackData[2] = {(int16_t)encoderDataRight.velocity, (int16_t)
            encoderDataLeft.velocity};
```

```

26     Serial_SendData(&huart2, feedbackData, 2);
27 }
28 }

```

Listing 5: メイン制御ループ (main.c)

### 3.3.2 上位レイヤー

本システムの上位レイヤーでは、以下の ROS2 ノードを利用してリアルタイムなデータ処理と通信を実現している。開発環境は ROS2 humble ,Python 3.10.12 である。

本研究で使った ROS2 ノードは以下の通りである。

- `web_socket_node`: Web ブラウザと通信し、ロボットに指令を与える。
- `PID_node`: PID 制御を用いてモーター制御信号を生成。
- `RealSense_node`: 深度カメラからデータを取得し、ターゲットの位置を推定。
- `serial_read_node`: 下位レイヤーとのシリアル通信を通じてエンコーダーデータを受信。
- `serial_send_node`: モーター制御信号を下位レイヤーに送信。
- `Roboware_node`: ロボット全体の管理、および追従アルゴリズムを搭載。

図 3.4 に `rqt_graph` によりノード間の接続状況を視覚化したものを示す。

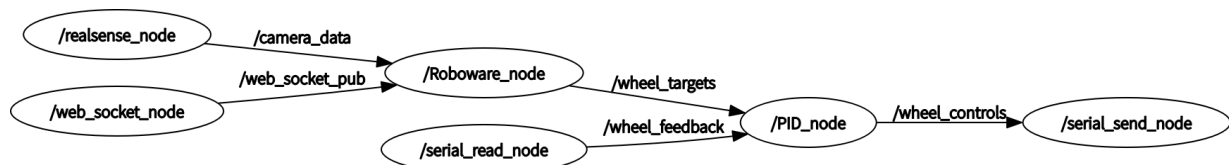


図 3.4 : `rqtgraph`

#### `web_socket_node`

`web_socket_node` は、FastAPI を使用して Web ブラウザと ROS2 ノード間の双方向通信を行う。Web ブラウザからはロボットの制御指令を送信でき、ロボットの現在位置や状態をリアルタイムで取得できる。

FastAPI は軽量かつ高性能な Web フレームワークであり、非同期通信をサポートするため、WebSocket を用いたリアルタイム通信にも対応している。WebSocket は、HTTP プロトコルを拡張した双方向通信のためのプロトコルであり、効率的なデータ送受信を可能にする。

- 購読トピック
  - `estimated_position` (Float32MultiArray): 推定位置データ
- 発行トピック
  - `web_socket_pub` (String): WebSocket 通信で受信したコマンド

以下に、WebSocket 通信の主要な実装部分を示す。

```

1 # FastAPI インスタンスの作成
2 app = FastAPI()
3
4 # WebSocket 通信のエンドポイント
5 @app.websocket('/ws')
6 async def websocket_endpoint(websocket: FastAPIWebSocket):
7     await websocket.accept()
8     try:
9         while True:
10             # クライアントからのデータを受信
11             receive_data = await websocket.receive_text()
12
13             # ROS トピックにデータをパブリッシュ
14             msg = String()
15             msg.data = receive_data
16             self.pub.publish(msg)
17
18             # サブスクライブしたデータをクライアントに送信
19             string_send_data = ",".join(map(str, self.send_data))
20             await websocket.send_text(string_send_data)
21     except Exception as e:
22         print(f'WebSocket error: {str(e)}')
23     finally:
24         print('WebSocket disconnected')

```

Listing 6: WebSocket 通信の主要部分 (web\_socket\_node.py)

この実装では、Web ブラウザから送信された制御信号を ROS トピック ‘web\_socket\_pub’ にパブリッシュし、ROS2 ノードで処理される。同時に、ROS2 ノードがサブスクライブしたデータを Web ブラウザに送信することでリアルタイム通信を実現している。

## PID\_node

PID\_node は、ロボットのモーター制御信号を生成するために PID 制御アルゴリズムを実装したノードである。PID 制御では、目標値と現在値の差（偏差）を基に比例（P）、積分（I）、微分（D）の3要素を組み合わせることで制御信号を生成する。さらに、本ノードでは微分項にローパスフィルタを適用し、ノイズの影響を低減している。

本ノードは以下の ROS トピックを使用する。

- 購読トピック
  - wheel\_targets (Float32MultiArray): 目標速度データ（右車輪，左車輪）
  - wheel\_feedback (Float32MultiArray): 現在速度データ（右車輪，左車輪）
- 発行トピック
  - wheel\_controls (Float32MultiArray): 制御信号（右車輪，左車輪）

以下に、PID 制御アルゴリズムの主要な実装部分を示す。

```

1 class PIDController:
2     def __init__(self, kp, ki, kd, tau=0.01):
3         self.kp = kp
4         self.ki = ki

```

```

5         self.kd = kd
6         self.tau = tau # ローパスフィルタの時定数
7         self.prev_error = 0.0
8         self.integral = 0.0
9         self.prev_derivative = 0.0 # 前回の微分値
10
11     def compute(self, target, current, dt):
12         error = target - current
13         self.integral += error * dt
14
15         # 微分項にローパスフィルタを適用
16         raw_derivative = (error - self.prev_error) / dt
17         derivative = (self.tau * self.prev_derivative + dt * raw_derivative) / (self.tau +
18                                     dt)
19         self.prev_derivative = derivative
20         self.prev_error = error
21
22     return self.kp * error + self.ki * self.integral + self.kd * derivative

```

Listing 7: PID 制御アルゴリズムの実装 (PID\_node.py)

このアルゴリズムでは、目標速度と現在速度の差を計算し、PID 制御の各要素を基に制御信号を生成する。ローパスフィルタを導入することで、ノイズによる微分項の影響を抑制し、安定した制御信号を生成する。

次に、制御ループの主要部分を以下に示す。

```

1 def control_loop(self):
2     current_time = self.get_clock().now()
3     dt = (current_time - self.last_time).nanoseconds / 1e9
4     self.last_time = current_time
5
6     # 右車輪と左車輪の制御信号を計算
7     control_signal_right = self.pid_right.compute(self.target_right, self.current_right,
8                                                    dt)
9     control_signal_left = self.pid_left.compute(self.target_left, self.current_left, dt)
10
11     # 制御信号をパブリッシュ
12     control_msg = Float32MultiArray()
13     control_msg.data = [float(control_signal_right), float(control_signal_left)]
14     self.pub.publish(control_msg)

```

Listing 8: 制御ループ (PID\_node.py)

この制御ループでは、以下の手順でモーター制御信号を生成する。

1. 目標速度と現在速度の偏差を計算し、PID 制御を実行。
2. 生成した制御信号を ROS トピック wheel\_controls にパブリッシュ。
3. デバッグ用に制御信号の詳細をログ出力。

これにより、ロボットの速度を目標値に追従させるためのリアルタイム制御が可能となる。

## RealSense\_node

RealSense\_node は、Intel RealSense D435i カメラを使用して深度データと RGB 画像を処理し、ター

ゲットの検出および距離・オフセット情報を生成する．このノードは，YOLOv5 を利用した物体検出アルゴリズムを実装しており，人物の位置と距離をリアルタイムで推定してパブリッシュする．本ノードは以下の ROS トピックを使用する．

- 発行トピック
  - camera\_data (Float32MultiArray): 人物の距離とオフセット情報

以下に，主要な処理と実装部分を示す．

```
1 # RealSense フレームの処理
2 def process_frames(self):
3     try:
4         frames = self.pipeline.wait_for_frames()
5         depth_frame = frames.get_depth_frame()
6         color_frame = frames.get_color_frame()
7         if not depth_frame or not color_frame:
8             return
9
10        # 深度データとRGB データを取得
11        depth_image = np.asanyarray(depth_frame.get_data())
12        color_image = np.asanyarray(color_frame.get_data())
13
14        # YOLOv5 による物体検出
15        results = self.model(color_image)
16        for result in results.xyxy[0]: # 検出結果をループ
17            box, conf, cls = result[:4], result[4], int(result[5])
18            if cls == 0: # クラス (人物) のみ処理
19                x1, y1, x2, y2 = map(int, box)
20                center_x, center_y = (x1 + x2) // 2, (y1 + y2) // 2
21
22                # 深度データから距離とオフセットを計算
23                raw_distance = depth_frame.get_distance(center_x, center_y)
24                offset_x = center_x - (color_image.shape[1] // 2)
25                filtered_distance = self.filter_distance(raw_distance)
26
27                # データをパブリッシュ
28                msg = Float32MultiArray()
29                msg.data = [filtered_distance, float(offset_x)]
30                self.publisher.publish(msg)
31
32                # デバッグ情報の表示
33                self.get_logger().info(
34                    f"Published camera data: Distance={filtered_distance:.2f}, Offset={
35                        offset_x}"
36                )
37                break
38        except Exception as e:
39            self.get_logger().error(f"Error processing frames: {str(e)}")
```

Listing 9: ターゲット検出と距離推定 (RealSense.node.py)

距離データは，近づく場合はそのままの値を使用し，遠ざかる場合は移動平均フィルタを適用している．このフィルタリングは，測定誤差やノイズの影響を低減するためである．

```
1 def filter_distance(self, current_distance):
2     if current_distance == 0.0: # 無効値は無視
3         return self.previous_distance
```



```

4
5     if current_distance < self.previous_distance:
6         # 近づいている場合: そのままの値を使用
7         self.distance_history = [current_distance] # 履歴をリセット
8         return current_distance
9     else:
10        # 遠ざかる場合: 移動平均フィルタを適用
11        self.distance_history.append(current_distance)
12        if len(self.distance_history) > self.history_size:
13            self.distance_history.pop(0) # 古い値を削除
14        return sum(self.distance_history) / len(self.distance_history)

```

Listing 10: 距離データのフィルタリング (RealSense\_node.py)

本ノードは、RealSense カメラから取得した深度データと RGB 画像をもとに、YOLOv5 を用いたターゲット検出と距離推定を行い、リアルタイムで ROS トピックに結果をパブリッシュする。

### serial\_read\_node

serial\_read\_node は、下位レイヤーから送信されるエンコーダーデータをシリアル通信を通じて受信し、ROS トピックにパブリッシュする役割を持つ。このノードは非同期通信をサポートしており、リアルタイムでのデータ受信とパブリッシュを実現している。

- 発行トピック

- wheel\_feedback (Float32MultiArray): エンコーダーデータ (右車輪, 左車輪)

以下に、主要なコード部分を示す。

```

1 class SerialReadNode(Node):
2     def __init__(self):
3         super().__init__('serial_read_node')
4         self.publisher = self.create_publisher(Float32MultiArray, 'wheel_feedback', 10)
5         # シリアル通信の設定
6         self.serial_port = serial.Serial('/dev/ttyUSB0', 115200, timeout=1)
7         self.create_timer(0.1, self.read_serial_data)
8     def read_serial_data(self):
9         try:
10            if self.serial_port.in_waiting > 0:
11                data = self.serial_port.read(8) # バイトを読み取る8
12                right_speed, left_speed = struct.unpack('>hh', data[2:]) # データのデコード
13                msg = Float32MultiArray()
14                msg.data = [float(right_speed), float(left_speed)]
15                self.publisher.publish(msg)
16                self.get_logger().info(f"Published wheel_feedback: {msg.data}")
17            except Exception as e:
18                self.get_logger().error(f"Error reading serial data: {str(e)}")

```

Listing 11: シリアルデータの受信とトピックへのパブリッシュ (serial\_read\_node.py)

このノードでは、シリアルポートから受信したエンコーダーデータを解析し、右車輪と左車輪の速度データとしてトピック wheel\_feedback にパブリッシュする。

## serial\_send\_node

serial\_send\_node は、上位レイヤーからのモーター制御信号を下位レイヤーに送信する役割を持つ。このノードは、ROS トピックから制御信号を受け取り、指定されたフォーマットでパケット化してシリアル通信を通じて下位レイヤーに送信する。

- 購読トピック
  - wheel\_controls (Float32MultiArray): モーター制御信号 (右車輪, 左車輪)

以下に、主要なコード部分を示す。

```
1 class SerialSendNode(Node):
2     def __init__(self):
3         super().__init__('serial_send_node')
4         self.subscription = self.create_subscription(
5             Float32MultiArray,
6             'wheel_controls',
7             self.send_serial_data,
8             10
9         )
10        # シリアル通信の設定
11        self.serial_port = serial.Serial('/dev/ttyUSB0', 115200, timeout=1)
12    def send_serial_data(self, msg):
13        try:
14            if len(msg.data) == 2:
15                right_control, left_control = int(msg.data[0]), int(msg.data[1])
16                data = struct.pack('>BBhh', 0xA5, 0xA5, right_control, left_control)
17                self.serial_port.write(data)
18                self.get_logger().info(f"Sent wheel_controls: {msg.data}")
19        except Exception as e:
20            self.get_logger().error(f"Error sending serial data: {str(e)}")
```

Listing 12: 制御信号の送信 (serial\_send\_node.py)

このノードでは、モーター制御信号を ROS トピック wheel\_controls から受信し、指定されたフォーマットでパケット化してシリアル通信を通じて送信する。

## Roboware\_node

Roboware\_node は、ロボットの速度制御を実現するためのノードであり、比例航法 (PN)、修正比例航法 (MPN)、およびゲインスケジューリング下修正比例航法 (GS-MPN) をサポートしている。これらのアルゴリズムを用いて、ターゲットを追従する速度と角速度を計算し、逆運動学を用いて車輪ごとの目標速度を決定する。

- 購読トピック
  - web\_socket\_pub (String): 操作モードおよびロボット制御信号
  - camera\_data (Float32MultiArray): 距離とオフセットデータ
- 発行トピック
  - wheel\_targets (Float32MultiArray): 車輪ごとの目標速度

以下に、各アルゴリズムでの速度計算部分を示す。

## PN (Proportional Navigation)

PN アルゴリズムでは、以下のように直進速度  $V$  と角速度  $\omega$  を計算する。

```
1 V = self.kp_v * (self.person_distance - 1.0)
2 omega = (self.navigation_constant) * self.kp_omega * self.person_offset / max((self.
    person_distance), 1.0)
```

Listing 13: PN の計算部分 (Roboware\_node\_np.py)

## MPN (Modified Proportional Navigation)

MPN では、偏差角速度の微分を用いて動作をなめらかにするため、以下のように計算する。

```
1 offset_rate = (self.person_offset - self.previous_offset) / self.dt # 偏差角速度
2 V = self.kp_v * (self.person_distance - 1.0)
3 omega = (
4     self.navigation_constant * self.kp_omega * self.person_offset / self.person_distance +
5     self.lambda_gain * offset_rate
6 )
7 self.previous_offset = self.person_offset # 偏差を更新
```

Listing 14: MPN の計算部分 (Roboware\_node\_mpn.py)

## GS-MPN (Gain-Scheduled Modified Proportional Navigation)

GS-MPN では、動的微分ゲインを導入し、偏差角速度に基づく調整を行う。

```
1 offset_rate = (self.person_offset - self.previous_offset) / self.dt # 偏差角速度
2 dynamic_kd = self.kd_lambda * (1 - math.exp(-self.a * abs(offset_rate))) / (1 + math.exp(-
    self.a * abs(offset_rate)))
3 V = self.kp_v * (self.person_distance - 1.0)
4 omega = (
5     self.navigation_constant * self.kp_omega * self.person_offset / self.person_distance +
6     dynamic_kd * offset_rate
7 )
8 self.previous_offset = self.person_offset # 偏差を更新
```

Listing 15: GS-MPN の計算部分 (Roboware\_node\_newmpn.py)

計算された直進速度  $V$  と角速度  $\omega$  から、逆運動学を用いて右車輪および左車輪の速度を以下のように計算する。

```
1 self.target_right = (2 * V + omega * self.L) / (2 * self.r)
2 self.target_left = (2 * V - omega * self.L) / (2 * self.r)
```

Listing 16: 逆運動学を用いた車輪速度の計算

また、カメラからの座標変換も行っている。

```
1 def position_callback(self, msg):
2     if len(msg.data) == 2:
3         pixel_offset = msg.data[1] # カメラ画像上の水平ズレ (ピクセル)
4         depth = msg.data[0] # カメラから取得した深度値 ( m )
5         # 座標変換
6         x_camera = (pixel_offset * depth) / self.fx # カメラ座標系のX
7         y_camera = depth # カメラ座標系のY
```

```
8
9      # ロボット座標系への変換
10     x_robot = x_camera + self.camera_offset_x
11     y_robot = y_camera
12
13     # 更新
14     self.person_offset = x_robot # ロボット座標系のX
15     self.person_distance = y_robot # ロボット座標系のY
```

Listing 17: 座標変換

本ノードでは、各種追従アルゴリズムを動的に選択できるよう設計されており、追従性能の向上と滑らかな動作を実現している。また、逆運動学を用いることで、車輪ごとの速度制御信号を正確に生成する。

## 第 4 章 実験

### 4.1 実験内容

本研究では、各アルゴリズム (PN, MPN, GS-MPN) の性能を比較し、追従精度や滑らかさ、安定性を評価する。また、実環境でのシステム全体の動作を検証する。

#### 4.1.1 実験条件

簡易的似作成した 2 輪移動ロボットを使用した。実験は屋内環境で行った。以下に示すように人を直線運動時と曲線運動時とを追従させ、追従性能、およびロボットのなめらかさを検証する。

- 条件 1: 人 (ターゲット) が直線に動く
- 条件 2: 人 (ターゲット) が曲線に動く

また、以下に各設定パラメータを示す。

- 比例ゲイン  $v$ : 5000
- 比例ゲイン  $\omega$ : 50.0
- 最大微分ゲイン: 0.1
- 動的微分ゲイン調整パラメータ: 0.1
- 比例航法定数 (N): 0.1
- サンプルング間隔: 0.1[s]

図 4.1 に実験の様子とロボットを示す。



図 4.1 : 実験の様子

#### 4.1.2 実験の手順

各アルゴリズムについて，以下の手順で実験を行った．

1. ロボットの前（1[m]）に追従物（人）を配置し，カメラの中心に来ていることを確認して実験を始める．
2. 開始後，人が条件 1，2 に則り 10[m] 移動．
3. エンコーダーデータやカメラデータを記録．
4. 記録した csv ファイルを解析．
5. これを 3 つのアルゴリズム，2 つの条件で実験を行う．

#### 4.2 実験結果及び考察

本実験では，各アルゴリズム（PN，MPN，GS-MPN）の性能を比較するために，以下の指標を計測した．

- MeanOffset (px): ターゲット中心からの平均オフセット
- StdOffset (px): オフセットの標準偏差
- MeanAccel ( $\text{mm/s}^2$ ): 平均加速度
- StdAccel ( $\text{mm/s}^2$ ): 加速度の標準偏差

表 4.1 に，各アルゴリズムの実験結果を示す．また，図 4.2 に MeanOffset と StdOffset，図 4.3 に MeanAccel と StdAccel のグラフを示す．

表 4.1：各アルゴリズムの実験結果

Experiment	MeanOffset (px)	StdOffset (px)	MeanAccel ( $\text{mm/s}^2$ )	StdAccel ( $\text{mm/s}^2$ )
PN_data1	41.86	54.03	29.08	29109.58
PN_data2	59.89	64.74	128.51	27609.34
MPN_data1	102.42	57.05	202.61	27873.37
MPN_data2	36.15	51.27	55.08	11523.68
GS-MPN_data1	66.29	69.30	101.00	11177.16
GS-MPN_data2	48.94	69.98	50.11	21241.42

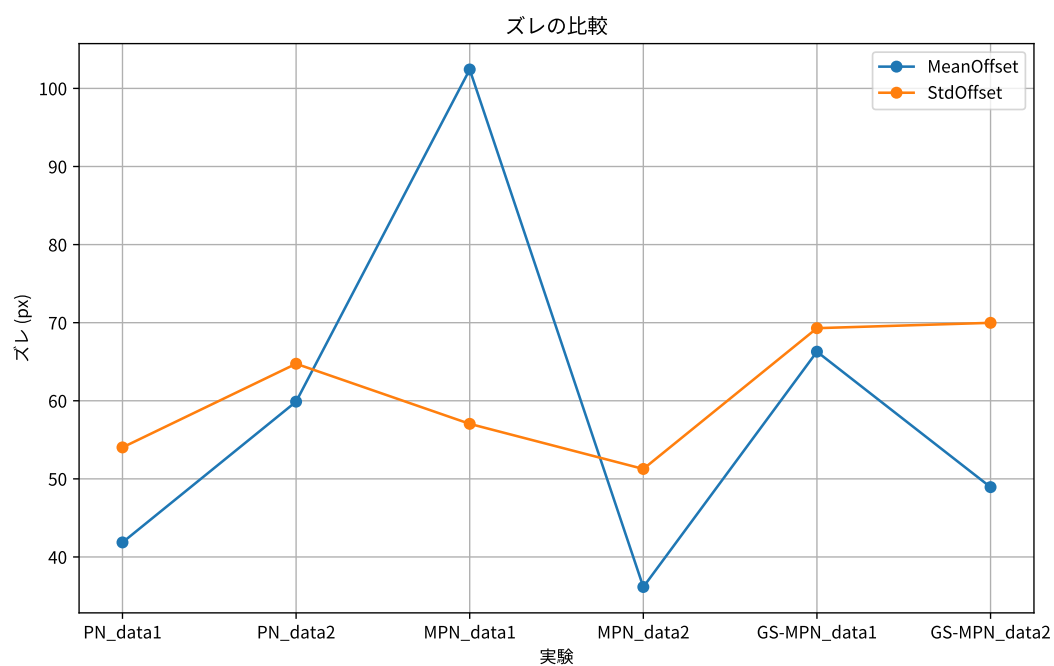


図 4.2 : MeanOffset および StdOffset の比較

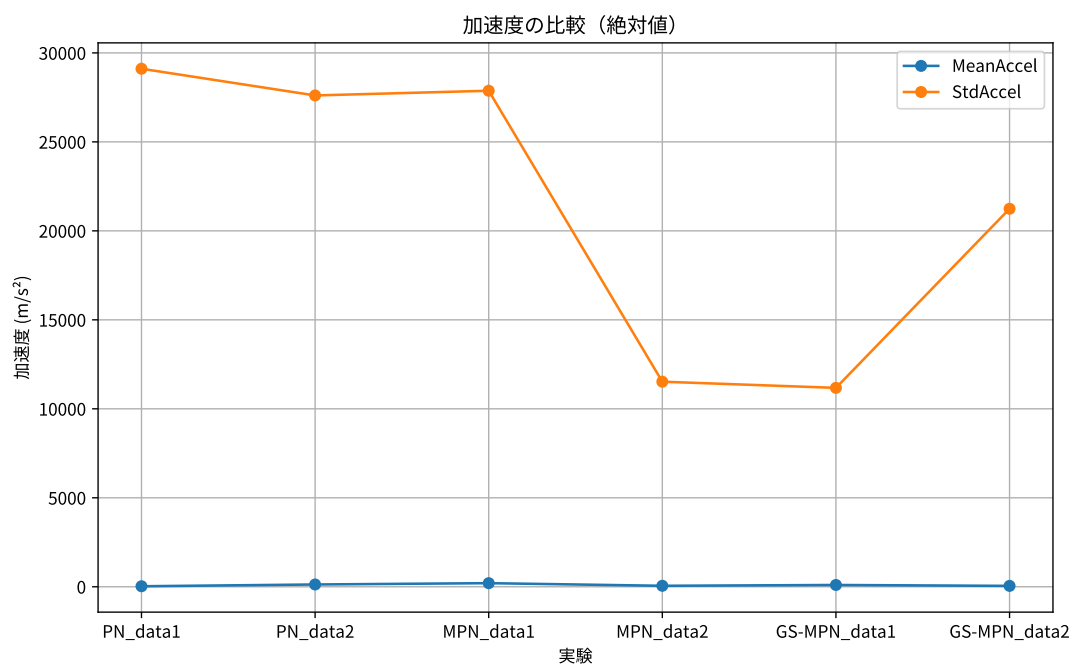


図 4.3 : MeanAccel および StdAccel の比較

#### 4.2.1 追従性能 (MeanOffset と StdOffset)

追従性能を評価するために、MeanOffset (画面中心からのズレの平均値) と StdOffset (ズレの標準偏差) を用いた。この指標は、目標物が画面内で中心付近にどれだけ正確に保持されているかを示す。表 4.2 に各アルゴリズムの結果を示す。

表 4.2 : 追従性能の評価結果

アルゴリズム	MeanOffset (px)	StdOffset (px)
PN_data1	41.86	54.03
PN_data2	59.89	64.74
MPN_data1	102.42	57.05
MPN_data2	36.15	51.27
GS-MPN_data1	66.29	69.30
GS-MPN_data2	48.94	69.98

PN は直線追従時に最も良好な MeanOffset を示したが、曲線追従ではズレが大きくなり安定性が低下した。一方で、MPN と GS-MPN は曲線追従時にズレが減少し、安定性が向上した。特に GS-MPN は、MeanOffset と StdOffset の両方でバランスの良い結果を示している。

#### 4.2.2 スムーズさ (MeanAccel と StdAccel)

スムーズさの指標として、MeanAccel (加速度の平均値) と StdAccel (加速度のばらつき) を用いた。これらはロボットの動作の安定性と滑らかさを評価する。表 4.3 に結果を示す。

表 4.3 : スムーズさの評価結果

アルゴリズム	MeanAccel (mm/s <sup>2</sup> )	StdAccel (mm/s <sup>2</sup> )
PN_data1	29.08	29109.58
PN_data2	128.51	27609.34
MPN_data1	202.61	27873.37
MPN_data2	55.08	11523.68
GS-MPN_data1	101.00	11177.16
GS-MPN_data2	50.11	21241.42

PN は加速度のばらつき (StdAccel) が非常に大きく、急加速が多発する結果となった。MPN では曲線追従時に加速度が抑えられ、StdAccel が減少した。GS-MPN ではさらにばらつきが抑えられ、最も滑らかな動作が実現された。



#### 4.2.3 視野内追従率 (InFrameRate)

視野内追従率 (InFrameRate) は、目標物がカメラの視野内に保持されている割合を示す。すべてのアルゴリズムで視野内追従率は 100% を記録し、目標物が常に視野内に維持されていることが確認された。

#### 4.2.4 総合評価

本実験では、各アルゴリズム (PN, MPN, GS-MPN) を対象に追従性能、スムーズさ、および視野内追従率を評価した。

まず、PN (Proportional Navigation) は直線追従性能において最も良好な結果を示した。MeanOffset が最小値を記録し、目標物を中心付近に保持する能力が高いことが確認された。一方で、StdOffset が増加する傾向があり、特に曲線追従時にはズレが顕著になることが課題として挙げられる。また、加速度のばらつき (StdAccel) が他のアルゴリズムに比べて極めて大きく、急激な動作が多発する結果となった。このことから、PN は直線的な動作には適しているものの、動的環境や急な方向転換を必要とする状況では不安定性が課題となる。

次に、MPN (Modified Proportional Navigation) は、PN と比較して曲線追従性能と動作の滑らかさが大幅に改善された。特に StdAccel の減少により、加速度のばらつきが抑制され、急激な加速や減速が少ない安定した動作が確認された。さらに、MPN\_data2 では、MeanOffset と StdOffset の値が他の条件よりも小さく、曲線追従における追従精度と安定性のバランスが良好であることが示されている。しかしながら、直線追従時には MeanAccel が最大値を記録しており、急加速が発生する傾向が見られた。これは、微分項の影響が過大である可能性を示唆しており、直進時の制御パラメータの最適化が求められる。

最後に、GS-MPN (Gain-Scheduled Modified Proportional Navigation) は、全体的に最もバランスの取れた性能を示した。動的微分ゲインを導入することで、急加速を抑制しながら安定した追従動作を実現していることが、MeanAccel および StdAccel の大幅な低下から確認された。さらに、曲線追従においてもズレが抑制され、MPN よりも安定性が向上している。一方で、MeanOffset が PN と比較してやや大きい値を示しており、直線追従精度のさらなる向上が課題として挙げられる。これには、動的微分ゲインの調整や、追加の制御項を導入することが効果的であると考えられる。

本実験の結果から、GS-MPN は滑らかな追従動作を必要とするシナリオにおいて最も有効であることが示唆された。直線および曲線追従の両方において安定性が向上し、加速度のばらつきが最小化されることで、ロボットの動作がより自然で滑らかになることが確認された。一方で、追従精度を向上させるための追加的な制御パラメータの調整や、動的環境における性能検証が今後の研究課題として残されている。

## 第 5 章 おわりに

本研究では、カメラのみを用いた人追従搬送ロボットの開発を目指し、Intel RealSense D435i から取得したデータを活用して人の位置を推定し、追従アルゴリズムに基づいてロボットを制御するシステムを構築した。特に、修正比例航法(MPN)やゲインスケジューリング下修正比例航法(GS-MPN)を用いることで、従来の比例航法(PN)では対応が難しかった滑らかな動作や安定性の向上を図った。また、デプスカメラの画像データを用いて物理空間での人の座標を高精度に推定する仕組みを実装し、ロボット制御に反映させた。これにより、低コストなセンサーを使用したロボットシステムの可能性を示すことができた。

本研究の成果として、次の点が挙げられる。まず、PN、MPN、GS-MPN の 3 つの追従アルゴリズムを比較評価し、GS-MPN が最もバランスの取れた追従性能と動作の滑らかさを実現することを確認した。また、ロボットのハードウェア構成として、D435i を中心としたカメラシステム、STM32 を使用した下位制御層、および ROS2 による上位制御層を統合したシステムを構築し、それぞれの役割を分担させることで、スムーズなデータ処理と制御を可能にした。さらに、追従性能の評価では、GS-MPN が急加速やズレの少ない安定した追従を実現し、従来手法を上回る性能を示した。

一方で、本研究にはいくつかの課題も残された。まず、カメラの視野内での追従には成功したものの、遮蔽物が発生した場合や複数の対象物が存在する場合における動作安定性については十分に検証が行えていない。特定のターゲットを継続的に追従するための識別アルゴリズムが必要である。さらに、外乱環境下での耐性についても課題が残る。例えば、屋外での風や路面の不整に対する追従性能の検証は未実施であり、システムの実用化に向けてはこれらの外乱要因への対応が求められる。

また、本研究ではカメラのデプスデータのみを用いて追従アルゴリズムを構築したが、さらなる認識精度の向上を目指してセンサーフュージョンを導入することが考えられる。具体的には、LiDAR や IMU など他のセンサー情報を統合することで、環境の多様な条件に対応可能なシステムが構築できると期待される。また、歩行経路の予測を導入することで、人の進行方向に先回りする制御を実現し、応答性を向上させることも今後の課題として挙げられる。

## 参考文献

- 1) 出村 公成, 萩原 良信, 升谷 保博, タン ジェフリー トウ チュアン: ROS2 と Python で作って学ぶ AI ロボット入門, 講談社 (2018)
- 2) ROS 2 Design Documents: <https://design.ros2.org/>, [Accessed: Dec. 25, 2024] .
- 3) ROS 2 Documentation: <https://docs.ros.org/en/humble/index.html>, [Accessed: Dec. 25, 2024] .
- 4) ultralytics: <https://docs.ultralytics.com/ja/yolov5/>, [Accessed: Dec. 25, 2024] .
- 5) 佐藤太郎: 高等専門学校における一般科目と専門科目, 京都出版 (2018) ..... **本の場合**
- 6) 鈴木次郎, 高橋三郎: 高等専門学校と大学の違い, 電子制御学会論文誌, Vol. 25, No. 13, 123/130 (2017) ..... **学会誌論文の場合**
- 7) 田中史朗, 伊藤五郎, 渡辺花子: 高専における就職活動, 電気電子工学講演会資料, 543/546 (2016) ..... **講演会等資料 ( ページが記載されているもの ) の場合**
- 8) 山本一二三, 中村五十六: 高専における就職活動, メカトロニクス講演会資料, 全 4 頁 (2016) ..... **講演会等資料 ( ページが記載されていないもの ) の場合**
- 9) 中村十三子: 機械加工と実習工場, 平成 29 年度舞鶴工業高等専門学校機械工学科卒業論文 (2018) ..... **卒業論文の場合**
- 10) T. Sato: General Subjects and Special subjects at National Institute of Technology, Kyoto Publishing (2018)
- 11) J. Suzuki and S. Takahashi: Difference between National Institute of Technology and Universities, Journal of the Electronic Control Society, Vol. 25, No. 13, 123/130 (2017)
- 12) S. Tanaka, G. Ito and H. Watanabe: Job Hunting in NIT, Proceedings of Conference on Electrical and Electronics Engineering, 543/546 (2016)
- 13) H. Yamamoto and I. Nakamura: Job Hunting in NIT, Proceedings of Mechatronics Conference, 4 pages (2016)
- 14) 舞鶴高専ホームページ: <http://www.maizuru-ct.ac.jp>

## 謝 辭

[illegible]

## 付 録

## A.1 ちちち

[illegible]

A.2 כככ  
כככ

### A.2.1 ててて

[illegible]

文を書く．ここに，本文を書く．

A.2.2 ととととととととととととととととととととととととととととととと  
ととととととと

[illegible]