舞鶴工業高等専門学校 電子制御工学科 卒業論文 (令和 2 年 2 月 4 日提出)

人追従搬送ロボットのシステム開発

野口 史遠 (s9123)

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

目 次

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な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 [1)]．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 [2)]．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 [3)]．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]
- ω：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．

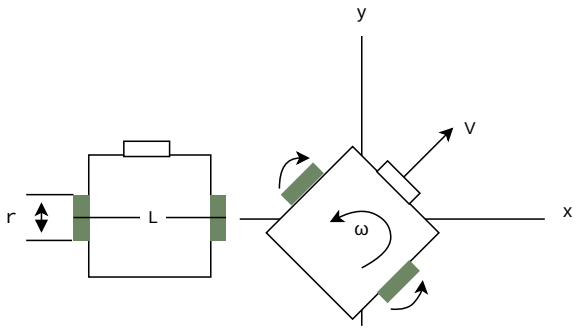

図 2.1 ：2 輪ロボットのモデル図

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 ω，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 ω は，次式で表される．

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

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

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

$$\dot{x} = v\cos\theta \tag{2.3}$$

$$\dot{y} = v\sin\theta \tag{2.4}$$

$$\dot{\theta} = \omega \tag{2.5}$$

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．

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

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

$$\theta_{k+1} = \theta_k + \omega_k\Delta t \tag{2.8}$$

ここで，Δt 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 ω 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 ω 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．

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

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

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．

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

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

これらの式により，目標速度 v と ω 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，モーター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：

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

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．

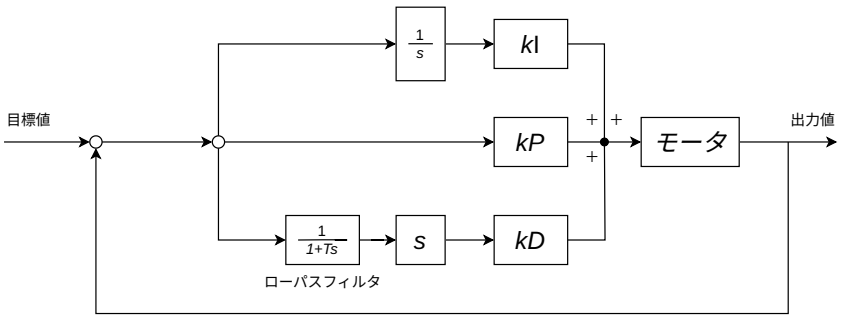

図 2.2 : PID 制御器の構成図

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：

$$D(s) = \frac{K_D s}{1 + Ts} \tag{2.14}$$

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 [4]．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．

2.4.1 人認識への応用

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. ネック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．

2.6　デプスデータからの人座標検出

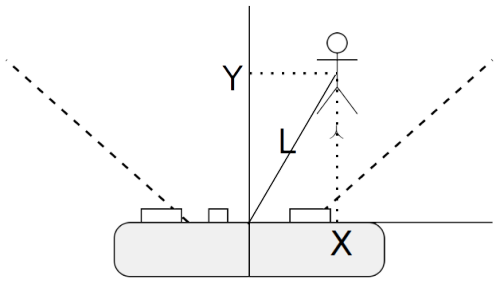

図 2.3 ：デプスカメラを用いた座標変換モデル

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．

2.6.1 ピクセル座標からカメラ座標への変換

デプスカメラのデプスマップは，画像として各ピクセルの深度情報（距離）を格納している．このデータを利用して，ピクセル座標 (u, v) をカメラ座標 (X_c, Y_c, Z_c) に変換する．変換には以下の式を使用する：

$$X_c = \frac{(u - c_x) \cdot Z_c}{f_x} \tag{2.15}$$

$$Y_c = \frac{(v - c_y) \cdot Z_c}{f_y} \tag{2.16}$$

$$Z_c = D(u, v) \tag{2.17}$$

ここで，

- u, v：ピクセル座標系の位置 [pixel]
- c_x, c_y：カメラの光学中心 [pixel]
- f_x, f_y：カメラの焦点距離 [pixel]
- $D(u, v)$：デプスマップから取得される深度値 [m]

この変換により，画像上の対象物がカメラ座標系における 3 次元位置として表現される．

2.6.2 カメラ座標からロボット座標への変換

次に，カメラ座標 (X_c, Y_c, Z_c) をロボット座標 (X_r, Y_r, Z_r) に変換する．カメラとロボットの座標系の関係は，以下のように平行移動と回転を適用して記述される：

$$\begin{bmatrix} X_r \\ Y_r \\ Z_r \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{2.18}$$

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

これにより，ロボット座標系において対象物の位置が求められる．

本研究では，D435i のデプスマップを利用して人の位置を検出し，ロボットの進行方向を計算する．図 2.3 に示すように，人の座標 (X_r, Y_r) を基に追従アルゴリズムを適用し，目標の移動方向を決定する．

2.7 人追従アルゴリズム

2.7.1 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．

$$V = k_v \cdot (\text{person\_distance} - 1.0) \tag{2.19}$$

$$\omega = k_\omega \cdot \frac{\text{person\_offset}}{\max(\text{person\_distance}, 1.0)} \tag{2.20}$$

- k_v：並進速度の比例ゲイン
- k_ω：角速度の比例ゲイン
- person_distance：ロボットと目標間の距離
- person_offset：横方向の偏差

2.7.2 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．

$$\dot{\text{offset}} = \frac{\text{person\_offset} - \text{previous\_offset}}{\Delta t} \tag{2.21}$$

角速度 ω は以下の式で修正される．

$$\omega = k_\omega \cdot \frac{\text{person\_offset}}{\max(\text{person\_distance}, 1.0)} + \lambda \cdot \dot{\text{offset}} \tag{2.22}$$

ここで，

- λ：偏差角速度に対するゲイン
- Δt：制御周期
- 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．

2.7.3 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．

$$k_d = \lambda_d \cdot \frac{1 - \exp(-a \cdot |\dot{\text{offset}}|)}{1 + \exp(-a \cdot |\dot{\text{offset}}|)} \tag{2.23}$$

- λ_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

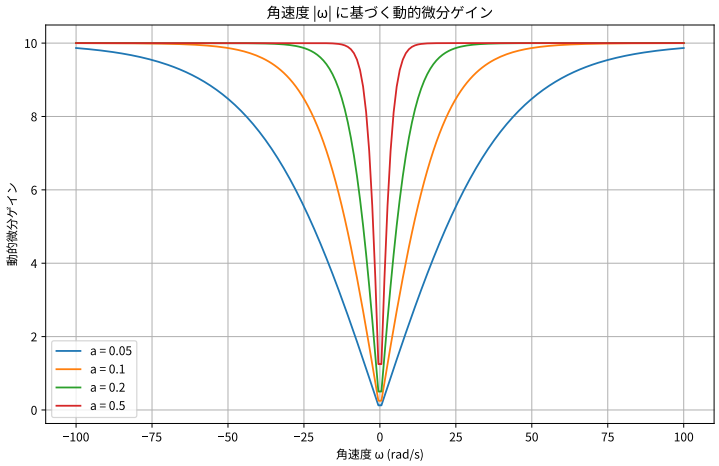

図 2.4 ：動的微分ゲイン

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 ω は次式で計算される．

$$\omega = k_\omega \cdot \frac{\text{person\_offset}}{\max(\text{person\_distance}, 1.0)} + k_d \cdot \dot{\text{offset}} \tag{2.24}$$

これにより，目標の動きが急変した場合でも柔軟に追従できる動作が可能となる．

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．

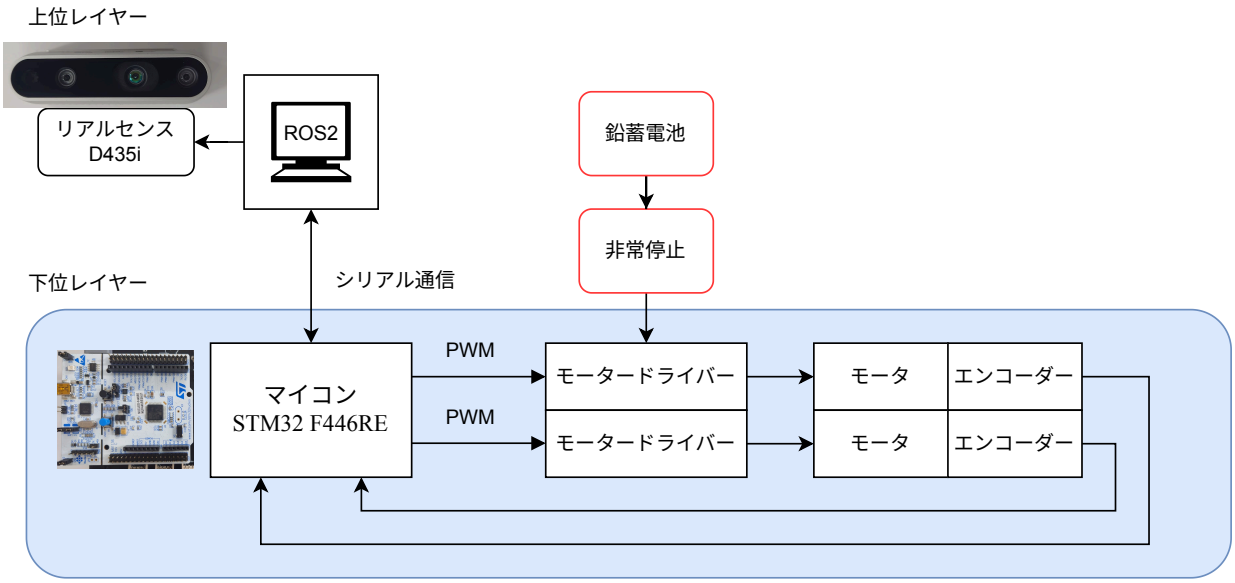

図 3.1 ： システム全体図

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．

表　　裏

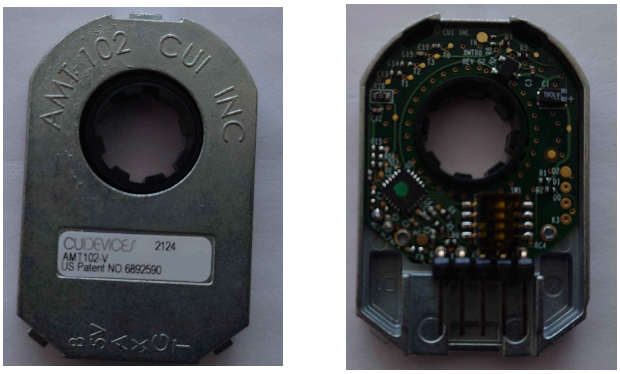

図 3.2 ： AMT102-V

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．

図 3.3 ： AMT102-V

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．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 逓倍処理を行い，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  int Encoder_Read(Encoder* encoder)
2  {
3      int16_t count = (int16_t)(__HAL_TIM_GET_COUNTER(encoder->htim) - TIMER_MAX_COUNT / 2);
4      return count;
5  }
6
7  void Encoder_Interrupt(Encoder* encoder, EncoderData* encoder_data)
8  {
9      int count = Encoder_Read(encoder);
10
11     encoder_data->count = count;
12     encoder_data->rot = count / (double)encoder->ppr;
13     encoder_data->deg = encoder_data->rot * 360.0;
14     encoder_data->distance = encoder_data->rot * (PI * encoder->diameter);
15
16     encoder_data->rps = (encoder_data->rot - encoder->before_rot) / (encoder->period *
           0.001);
17     encoder_data->velocity = encoder_data->rps * PI * encoder->diameter;
18
19     encoder->before_rot = encoder_data->rot;
20 }
```

図 3.4 ：エンコーダーカウント値の取得 (encoder.c)

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おる．以下に，一部ライブラリコードを示す．

```
// 初期化関数
void MotorDriver_Init(MotorDriver* motor, TIM_HandleTypeDef* htimA, uint32_t channelA,
    TIM_HandleTypeDef* htimB, uint32_t channelB) {
motor->htimA = htimA;
motor->channelA = channelA;
motor->htimB = htimB;
motor->channelB = channelB;
 // 開始PWM
HAL_TIM_PWM_Start(htimA, channelA);
HAL_TIM_PWM_Start(htimB, channelB);
}
// 速度設定関数
void MotorDriver_setSpeed(MotorDriver *motor, int speed) {
    int pwm_value;
    if (speed > 100) speed = 99;      //ブートストラップ回路に対応
    if (speed < -100) speed = -99;    //ブートストラップ回路に対応

    if (speed > 0) {
        pwm_value = (speed * __HAL_TIM_GET_AUTORELOAD(motor->htimA)) / 100;
        __HAL_TIM_SET_COMPARE(motor->htimA, motor->channelA, pwm_value);
        __HAL_TIM_SET_COMPARE(motor->htimB, motor->channelB, 0);
    } else {
        pwm_value = (-speed * __HAL_TIM_GET_AUTORELOAD(motor->htimA)) / 100;
        __HAL_TIM_SET_COMPARE(motor->htimA, motor->channelA, 0);
        __HAL_TIM_SET_COMPARE(motor->htimB, motor->channelB, pwm_value);
    }
}
```

図 3.5 ： 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．

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．

```
// 可変長データの送信関数
void Serial_SendData(UART_HandleTypeDef *huart, int16_t *data, uint8_t data_count) {
    uint8_t buffer_size = 2 + data_count * 2;
    uint8_t *buffer = (uint8_t *)malloc(buffer_size);

    buffer[0] = SERIAL_HEADER1;
    buffer[1] = SERIAL_HEADER2;

    for (uint8_t i = 0; i < data_count; i++) {
        buffer[2 + i * 2] = (data[i] >> 8) & 0xFF;
        buffer[3 + i * 2] = data[i] & 0xFF;
    }

    HAL_UART_Transmit(huart, buffer, buffer_size, HAL_MAX_DELAY);
    free(buffer);
}
// 可変長データの受信関数
uint8_t Serial_ReceiveData(UART_HandleTypeDef *huart, int16_t *data, uint8_t data_count) {
    uint8_t buffer_size = 2 + data_count * 2;
    uint8_t *buffer = (uint8_t *)malloc(buffer_size);

    if (HAL_UART_Receive(huart, buffer, buffer_size, HAL_MAX_DELAY) == HAL_OK) {
        if (buffer[0] == SERIAL_HEADER1 && buffer[1] == SERIAL_HEADER2) {
            for (uint8_t i = 0; i < data_count; i++) {
                data[i] = (buffer[2 + i * 2] << 8) | buffer[3 + i * 2];
            }
            free(buffer);
            return 1; // 正常受信
        }
    }
    free(buffer);
    return 0; // エラー
}
```

図 3.6 ： 可変長データの送受信関数 (serial_lib.c)

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．

```
// メイン制御ループ
while (1)
{
    /* からの制御信号を受信PC */
    int16_t receivedData[2];
    if (Serial_ReceiveData(&huart2, receivedData, 2))
    {
        controlSignalRight = receivedData[0];
        controlSignalLeft = receivedData[1];

        MotorDriver_setSpeed(&motorRight, -1 * controlSignalRight);
        MotorDriver_setSpeed(&motorLeft, -1 * controlSignalLeft);
    }

    /* エンコーダーデータの送信（10ごとに送信）ms */
    if (HAL_GetTick() - lastSendTime >= 10)
    {
        lastSendTime = HAL_GetTick();

        /* エンコーダーの速度データを更新 */
        Encoder_Interrupt(&encoderRight, &encoderDataRight);
        Encoder_Interrupt(&encoderLeft, &encoderDataLeft);

        /* エンコーダー速度を送信 */
        int16_t feedbackData[2] = {(int16_t)encoderDataRight.velocity, (int16_t)
            encoderDataLeft.velocity};
        Serial_SendData(&huart2, feedbackData, 2);
    }
}
```

図 3.7 ： メイン制御ループ (main.c)

参 考 文 献

1) 出村 公成, 萩原 良信, 升谷 保博, タン ジェフリー トゥ 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 Mechatoronics Conference, 4 pages (2016)
14) 舞鶴高専ホームページ：`http://www.maizuru-ct.ac.jp`

謝辞

ここに謝辞を記入する（必須ではない）．

付 録

A.1　ちちち

ここに，本文を書く．

A.2　つつつつつつつつつつつつつつつつつつつつつつつつつつつつつつつつつつつつつ

A.2.1　ててて

ここに，本文を書く．ここに，本

文を書く．ここに，本文を書く．

A.2.2 とと

ここに，本文を書く．