2022 年度（令和 4 年度）

創造工学セミナーⅡ　Final Report

# スーパーで人物追従する
# ロボットを導入する為の基礎研究


**研究メンバー**

S519019 呉　孝慈

S519026 荘司　和輝

S519059 吉岡　新

**指導教員**

金丸 隆志 教授

# 目次

# 第1章　緒言(荘司担当)

## 1.1 研究背景

### 1.1.1 人と買い物カートの接触

近年、新型コロナウイルスの世界的流行は人々の生活に大きな影響を与えた。特にコロナウイルス流行以前と比べると「人とモノが接触する際の意識」に変化があり、図 1.1 で示すように、多くの人が触るものを触らないという人が多い。その一方で未だものに触れなければいけない場面は日常生活で多い。その中でも我々はスーパーマーケットにおける人と買い物カートの接触に着目した。

### 1.1.2 小売業界の人手不足

さらに近年、様々な業界で人手不足が問題となっている。図 1.2 は各業種のパートタイム労働者の不足率を表し、横軸の D.I.(Diffusion Index) は労働者が「不足」と回答した事業所の割合から「過剰」と回答した事業所の割合を差し引いた値である。この図より、業界全体から見てもスーパーマーケットを含めた小売業界は人手不足が深刻であることが伺える。人手不足がさらに深刻化すれば働き手の負担が増えることから、今まで人が行っていたレジ打ちをはじめとした業務をロボットに行わせる動きが強まっていると考えられる。こういった背景から、人物を追従する買い物カートの実現を目標として研究開発を行う。

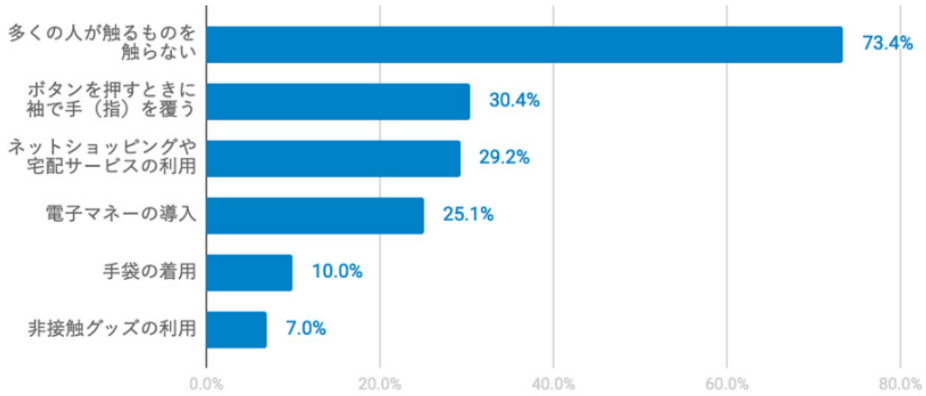


**図 1.1:人やモノとの接触を避けるために意識した、変えた行動 [1]**

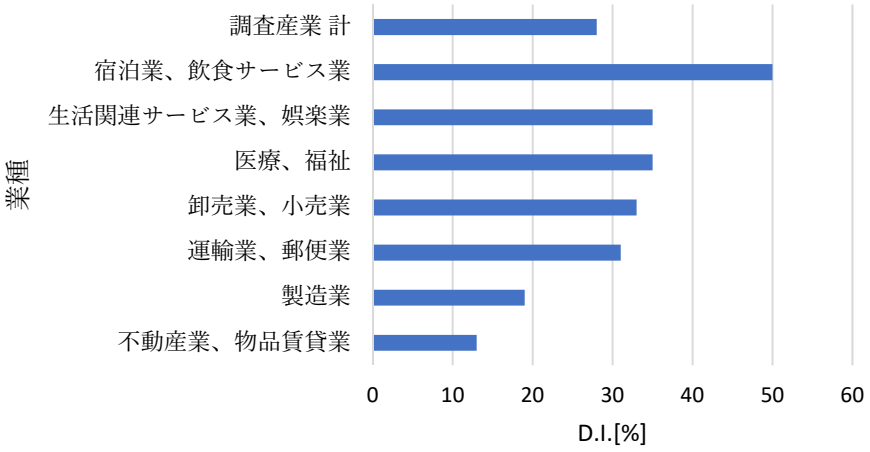


**図 1.2:産業別のパートタイム労働者の不足状況**

**(厚生労働省 労働経済動向調査(令和 4 年 5 月) [2]より作成)**

## 1.2 類似の研究

我々の研究する人物を追従する買い物カートに類似する製品の例を紹介する。図 1.3 はアメリカの企業ファイブ･エレメント･ロボティクス社が開発したロボット･カート「Dash」の写真である。Dash はアメリカで試用段階にあるロボット･カートである。このロボットの特徴は、売り場までの案内やカートに商品を入れる商品を自動決済ができることである。しかしながら、このロボット・カートにはスーパーマーケットという他の客、カート、子供が入り混じる環境にうまく対応できない問題点がある。

こうした様々なものが入り混じる環境に適応させるため、スーパーで人物追従するロボットを導入する為の基礎研究を本研究の目的とする。


**図 1.3: ファイブエレメンツ社が開発したロボット・カート Dash [3]**

## 1.3 研究概要

本研究で開発する人物追従をするロボット(以下 ロボットカートと呼ぶ)の条件について解説する。まず、以下の条件を課す。

・使用する環境は、屋内のスーパーマーケットとする。

・スーパーマーケットの地形は平坦であるものとする。

この条件下でのロボットカートの構想図を図 1.4 に示す。この構想図ではロボットカートが入り口付近で待機しており、顧客が入店した場合1人につき1台のロボットカートが追従を行う。これを実現するために、まずロボットカート前方にいる人物をセンサーによって認識する必要がある(以下 人物認識と呼ぶ)。しかし、スーパーマーケットでは追従する人物以外にも複数の人物がいる。そのため、認識した複数の人物の中からロボットカートが追従する対象者を特定する必要がある(以下 人物判別と呼ぶ)。そして、この追従対象者とロボットカートとの距離を一定に保つようにロボットカートを移動させることで追従対象者への追従を行う(以下 人物追従と呼ぶ)。これらの工程により、スーパーマーケットでの追従対象者への追従が可能であると思えるが、追従対象者が曲がり角で曲がった場合、ロボットカートに搭載されたセンサーから追従対象者を認識できなくなるだろう。そのような場合、センサーによって周囲の壁を認識しながらロボットカートが自律移動を行い、曲がり角の先にいる追従対象者を認識できる位置まで移動する必要がある(以下 人物捜索と呼ぶ)。

このように、スーパーマーケットでの人物追従を実現する為には

・人物認識機能

・人物判別機能

・人物追従機能

・人物捜索機能

をロボットカートに搭載する必要がある。

したがって、本研究ではロボットカートのハードウェア開発に加え、これらの機能に関するソフトウェアの開発を行うことで、スーパーマーケットで用いるロボットカートを実現することを目指す。

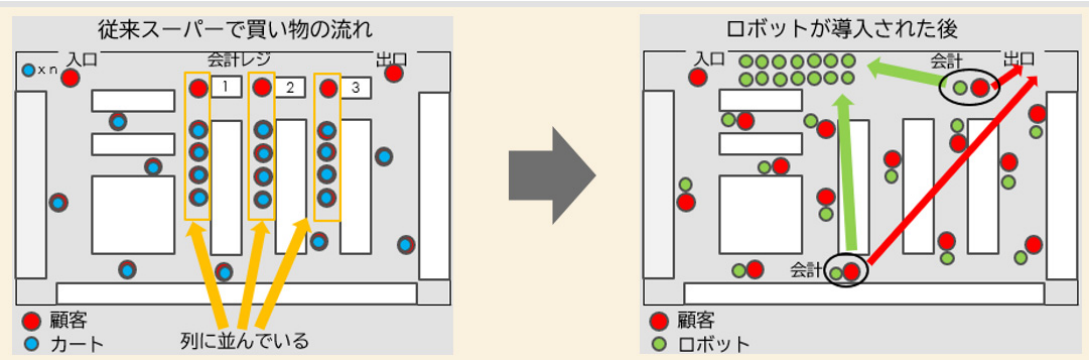


**図 1.4:ロボットカートの構想図**

# 第2章　使用機材(荘司担当)

## 2.1 ROS(Robot Operating System)インストール済みの PC

本研究では、ロボットの制御に ROS(Robot Operating System)を用いる。ROS とは、ロボット用のソフトウェアプラットフォームである。ROS をインストールした PC（Panasonic 製 Let's note CF.SZ5）を図 2.1 に示す。OS としては Linux ディストリビューションの一つである Ubuntu16.04 を用いる。先述の ROS はこの Ubuntu 上で動作するソフトウェアである。


**図 2.1:Let's note CF.SZ5**

## 2.2 TurtleBot3 waffle pi

本研究で開発する人物追従するロボットカートのベースとして TurtleBot3 waffle pi(図 2.2)(以下 TurtleBot3 と呼称)を利用する。ROS をインストールした PC で操作することが可能な ROS 公式ロボットである。レーザー距離センサー(2.4 参照)やカメラによって周囲の状況を把握することが出来る。本研究ではこのうちレーザー距離センサーのみを使い、それ以外の機能は別途追加する。

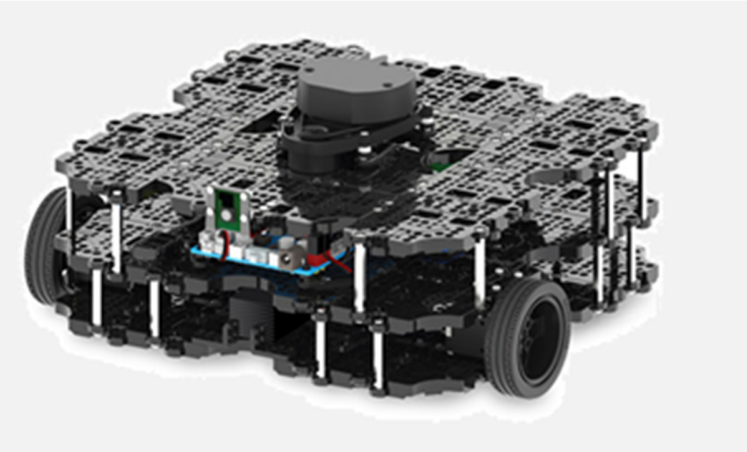

**図 2.2:TurtleBot3 waffle pi [4]**

## 2.3 RealSense D435

本研究では外界の映像を取得する為に、図 2.3 の RealSense D435(以下 RealSense と呼称)を使用する。RealSense とは Intel 社が発売しているデプスカメラであり、深度センサー、RGB カメラ、IR 投射器を搭載している。これらの機能により、高度な深度測定が可能となっており、カラー画像のほかカメラから距離を表現したデプス画像を出力することができる。RealSense の仕様を以下に示す。

**表 2.1:RealSense D435 の仕様 [5]**

| | |
|---|---|
| 使用環境 | 屋内/屋外 |
| 深度テクノロジー | アクティブ IR ステレオ（グローバルシャッター） |
| 主要コンポーネント | インテル RealSense モジュール D430+RGB カメラ、インテル RealSense ビジョンプロセッサーD4 |
| 深さ FOV(D×V×H) | 91.2×65.5×100.6 |
| 深度出力解像度とフレームレート | 最大 1280×720、最大 90fps |
| 最小深度距離 | 0.2m |
| 最大範囲 | 10m |
| RGB 解像度 | 1080p@30fps |
| コネクタ | USB3.0 |

**図 2.3:RealSense D435**

## 2.4 レーザー距離センサー(LDS.01)

本研究では、RealSense の視界から人物が外れた際に図 2.4 のレーザー距離センサ(LDS.01)で人物捜索する。LDS.01 とは TurtleBot3 に附属する 2 次元レーザースキャナーであり、スキャナーが回転することで周囲 360° にある物体の位置を点群データとして取得できる。このデータをもとに自己位置推定やマッピング、さらにナビゲーションを行うことが可能である。

**図 2.4:レーザースキャナー(LDS.01) [6]**

## 2.5 LiPo バッテリー(LB.012)

バッテリー(LB.012)(図 2.5)とは TurtleBot3 に附属のバッテリーであり、本体から取り外し、附属のアダプターで充電を行うことができる。主な仕様は以下の通りである。

**表 2.2:LiPo バッテリー(LB.012)の仕様 [7]**

| | |
|---|---|
| 電圧(V) | 11.1V |
| 容量(mAh) | 1800mAh |
| 電力量(Wh) | 19.98Wh |
| 電荷(C) | 5C |

**図 2.5:LiPo バッテリー(LB.012)**

## 2.6 OpenCR

OpenCR(図 2.6)とは TurtleBot3 に搭載されているマイクロコントローラ搭載ボードである。OpenCR はオープンハードウェアとソフトウェアから構成されており、ROS を使用した組み込みシステム向けに作られている。ロボットの基本的な動作はこのボードを介して行われる。

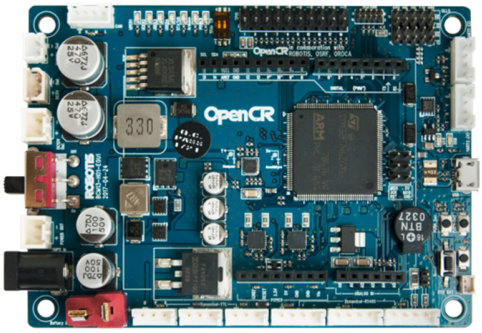


**図 2.6:OpenCR [8]**

## 2.7 Raspberry Pi

Raspberry Pi(図 2.7)はシングルボードコンピュータと呼ばれるハードウェアであり、TurtleBot3 に搭載される。Ubuntu と ROS をインストールすることで、ノート PC と ROS を介して通信し、ロボットの制御を行うことができる。

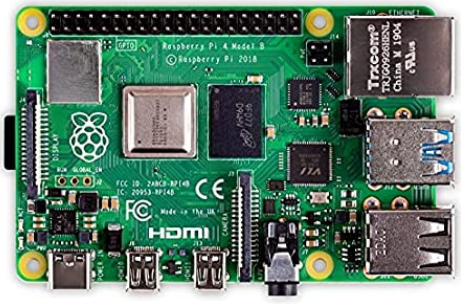


**図 2.7:Raspberry Pi [9]**

# 第3章　ハードウェアの開発(呉担当)

## 3.1 目的

我々はスーパーマーケットという環境での、買い物カートの自動化を目標としている。しかし、TurtleBot3 のデフォルトの状態（図 2.2）では買い物カートとしての役割を果たせないため、いくつかの改良を加えなければならない。本章では買い物カートとしての役割を果たすロボットカートを製作することを目標とする。図 3.1 がロボットカートの完成品であり、完成するまでどのような改良が加えられたかについてその過程を以下に詳しく記す。

ロボットカートに必要な変更点を示す。それぞれの詳細を 3.2 以降で述べる。

I.RealSense を設置すること

II.データ処理用の PC を設置すること

III.商品を載せる買い物かごを装着すること

IV.機器の設置による性能の低下が少ないこと


**図 3.1:ロボットカートの完成品**

## 3.2 概要

### 3.2.1 TurtleBot3 機内の改造

TurtleBot3 に必須である Raspberry Pi、OpenCR、バッテリーなどの電子部品に加え、RealSense と PC を新たに設置する必要がある。図 3.2 のように元々二層目にある全ての電子部品を車輪と同じ一層目に移し、二層目を RealSense と PC を置く場所とする。

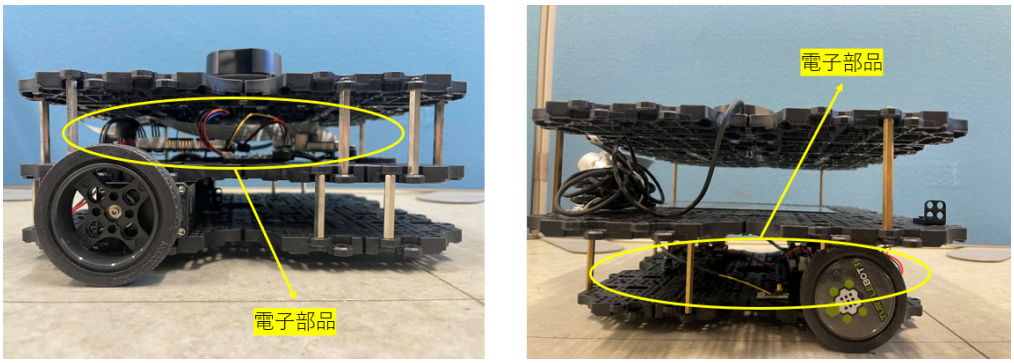


**図 3.2:電子部品が移動された図**

### 3.2.2 RealSense の設置

RealSense の取り付け方法について説明する。

使用する部品は以下の通りである。

●アルミニウム板 300×100mm……1 枚

●ネジ M2.5×8mm……4 個

●ナット M2.5……4 個

アルミニウム板加工の寸法を図 3.3 に示す。アルミニウム板の上に直径 2.5mm の穴を四つ開け、先端から 65mm の部分を境に上向き 45 度に曲げる。先端寄りの二つ穴は RealSense を固定するため利用し、残りの二つ穴は TurtleBot3 の二層目と固定する為に用いる。アルミニウムの加工が完成した後、ネジとナットを使って、TurtleBot3 とアルミニウム板と RealSense を固定する。RealSense を取り付けた後の様子を図 3.4 に参照する。

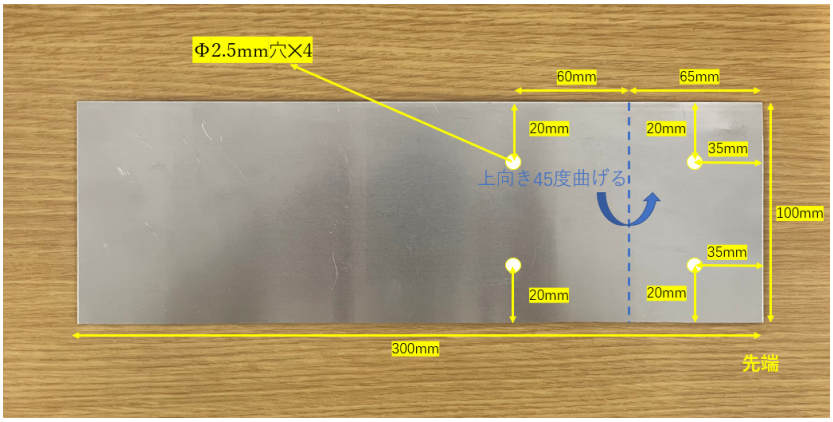


**図 3.3:アルミニウムの加工の寸法**

**図 3.4:RealSense を付けた後の様子**

### 3.2.3 PC の設置

PC を設置する方法について説明する。使用する部品は以下の通りである。

●TB3 Waffle Plate-IPL-01[10]……8 枚

本節では上記のうち二枚を組み合わせたもの（図 3.5）を Waffle Plate パーツと略称する。

●高さ 65mm の柱（図 3.6）……10 本

高さ 65mm の柱は元々用いられていた高さ 40mm の柱 22 本から変更するためのものである。

ROS をインストールした PC（2.1 参照）の幅は 280mm である。しかし、TurtleBot3 の幅は 255mm しかないため、幅を延長しなければならない。そのために上記の Waffle Plate パーツを 4 枚追加するのである。また、柱の長さを 40mm から 65mm に変更する理由は追加したパーツとタイヤが干渉することを避けるためである。柱の本数は 22 本から 10 本に減ったものの、安定性はあまり変わらなかった。改造前後の比較を図 3.7 に示す。

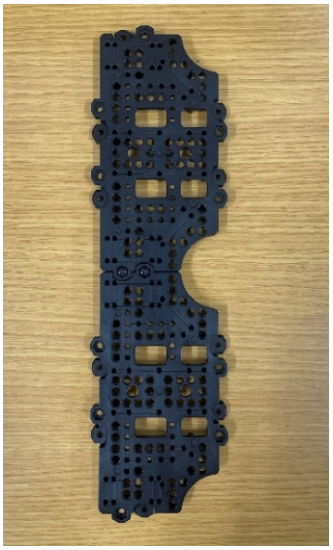

**図 3.5: Waffle.Plate パーツ**

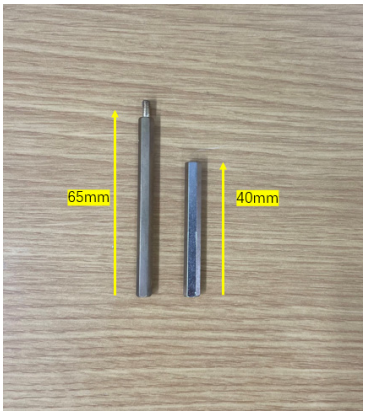


**図 3.6: 65mm 柱(左)、40mm 柱(右)**

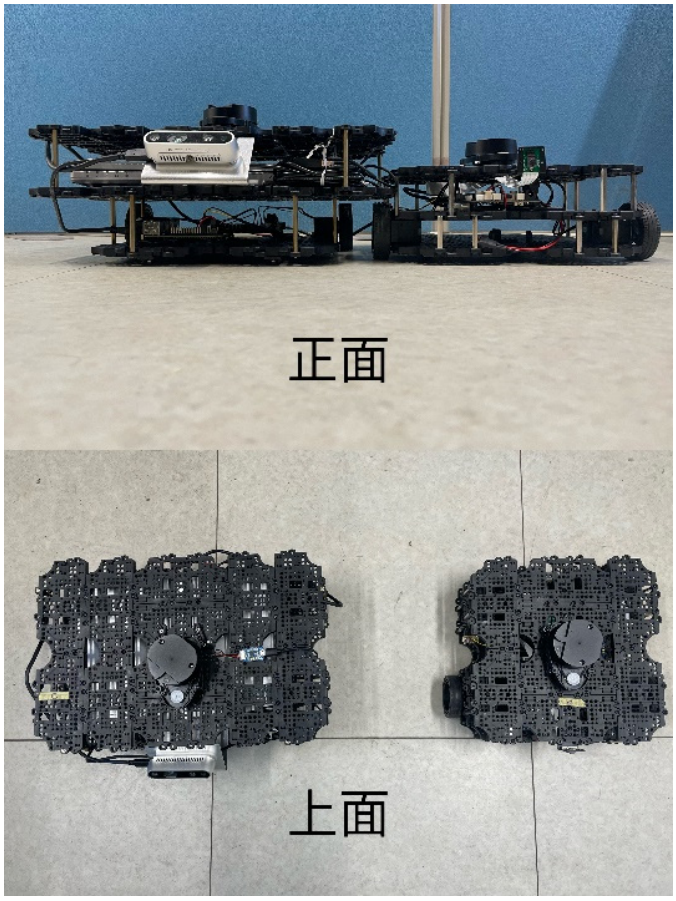


**図 3.7:改造前後の比較**

### 3.2.4 買い物かごの設置

買い物カートには買い物かごを置くスペースが必要である。そこでロボットカートの後方に買い物かごを置くための土台を製作する。使用する部品は以下の通りである。

●TB3 Waffle Plate-IPL-01[10]……8 枚

●TB3 Ball Caster-AO1[11]……2 個

本節では上記の部品を組み合わせたものをボールキャスター付き Waffle Plate パーツと略称する。

●ネジ M8×20mm……4 個

●ナット M8……4 個

●厚み 8mm 木板 600×450mm……1 枚

●厚み 5mm 木板 920×450mm……1 枚

先ず、二枚の木板を分割し、厚みが 5mm の板で支持部分を、厚みが 8mm の板で箱を図 3.8 のように作成する、そして両者を接着剤で貼り合わせる。支持部分の寸法は高さ 510mm、幅 205mm、長さ 225mm で、箱の寸法は高さ 75mm、幅 450mm、長さ 300mm である。支持部分の下には 4 つの穴が開けられており、ネジとナットでボールキャスター付き Waffle Plate パーツに固定されている。完成した土台とボールキャスター付き Waffle Plate パーツを連結した様子が図 3.8 に示されている。

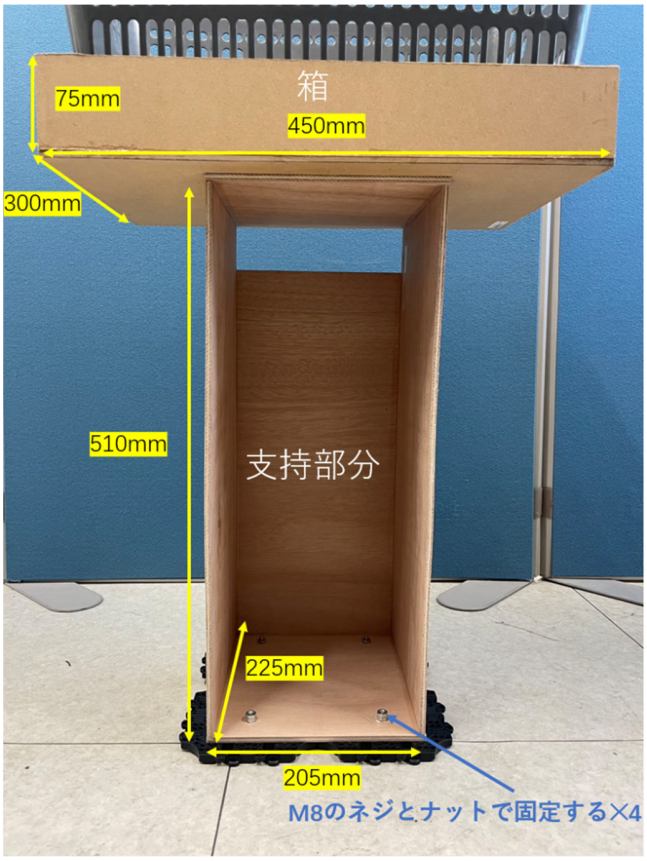


**図 3.8:買い物かごを設置するための土台**

### 3.2.5 タイヤ位置の選定

作り上げたロボットカートは直進･旋回の動力に低下が見られた他、直進時に軌道が逸れてしまう問題が見られた。運行中のロボットカートの様子を観察したところ、その原因は主にタイヤの空転であることが確認できた。その理由は荷重のかかるかごの位置とタイヤの位置が離れていることである。荷重のかかる位置とタイヤの位置を近づければタイヤの空転を防ぐことができると考えられる。そして、考えた改良による変更図を図 3.9 に、実際のタイヤの位置を変更した後のロボットカートの様子を図 3.10 に示す。

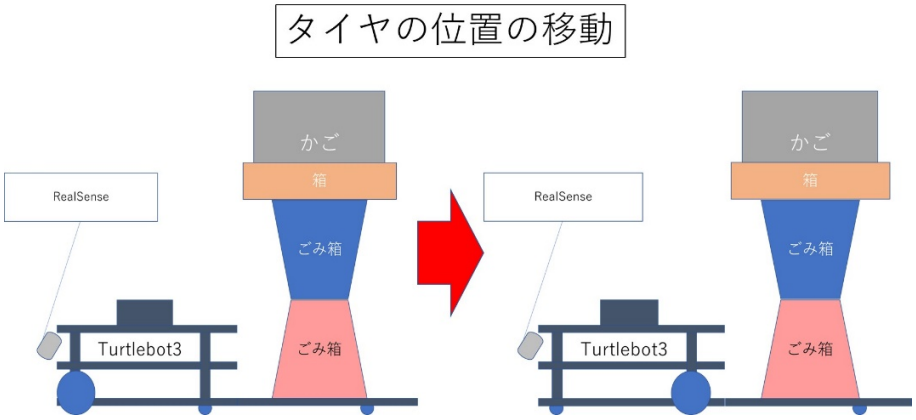


**図 3.9:タイヤの位置の変更図**

**図 3.10:タイヤ変更後の姿**

## 3.3 タイヤ位置変更の影響を調べる実験

### 3.3.1 実験方法

タイヤの位置を変更した前後でどれ程の効果があったのかを調べるため、計測を行った。図 3.11 は実験環境を示している。この実験では、「3mの直線移動」と「720° の旋回」の 2 つの運動にかかる時間をそれぞれ計測する。タイヤ位置の移動により、時間がどう変化するかを調べる。

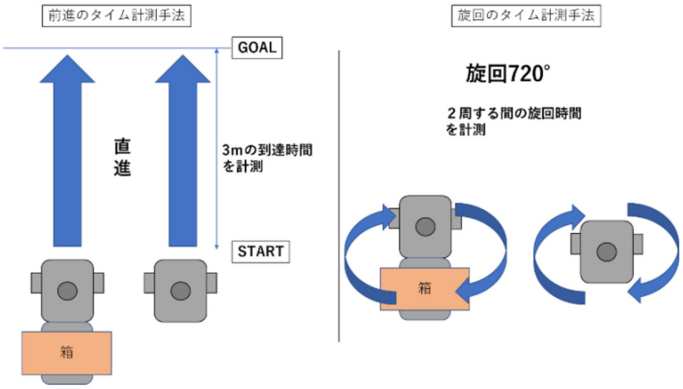


**図 3.11:実験方法のイメージ図**

### 3.3.2 実験結果

実験結果を表 3.1 に示す。かごを装着した TurtleBot3 のタイヤ移動前後でタイムを比べると直線で 5 秒向上、旋回で 47 秒向上と大幅な改善が見られた。これは、タイヤ位置の変更によりタイヤの空転が改善されたことによるものと考えられる。

**表 3.1:タイヤの位置変更前後効果の比較**

| | 直進(3m)にかかった時間(s) | 旋回(720° )にかかった時間(s) |
|---|---|---|
| 改造前の TurtleBot3 | 12s | 12s |
| かごを装着した TurtleBot3（タイヤ位置変更前） | 18s | 60s |
| かごを装着した TurtleBot3（タイヤ位置変更後） | 13s | 13s |

# 第4章　人物認識手法(呉担当)

## 4.1 目的

ロボットカートの製作にあたって最も重要といえる要素は人物認識である。この第 4 章では TurtleBot3 に搭載した RealSense で人物認識を行う手法を解説する。

## 4.2 概要

### 4.2.1 人物データ取得手法

我々が想定する環境では、周囲の壁や商品を載せたカートから人物を識別する必要がある。そのために、我々は 2.3 で解説した RealSense で得られた情報に対して画像処理を行う。体格や身にまとっている服などが異なる様々な容姿の人物を認識することを目指す。

### 4.2.2 カスケード分類器

我々が人物の判別に使用したものが Intel 社の OpenCV [12]に含まれるカスケード分類器である。カスケード分類器を用いた認識では図 4.1 のようにまずカラー画像からグレースケール画像に変換する。これは色の濃淡を際立たせる為である。そしてグレースケール画像をもとに学習済の xml ファイルに含まれる上半身の特徴量（色の濃淡）のデータから映像中の人物が認識される。

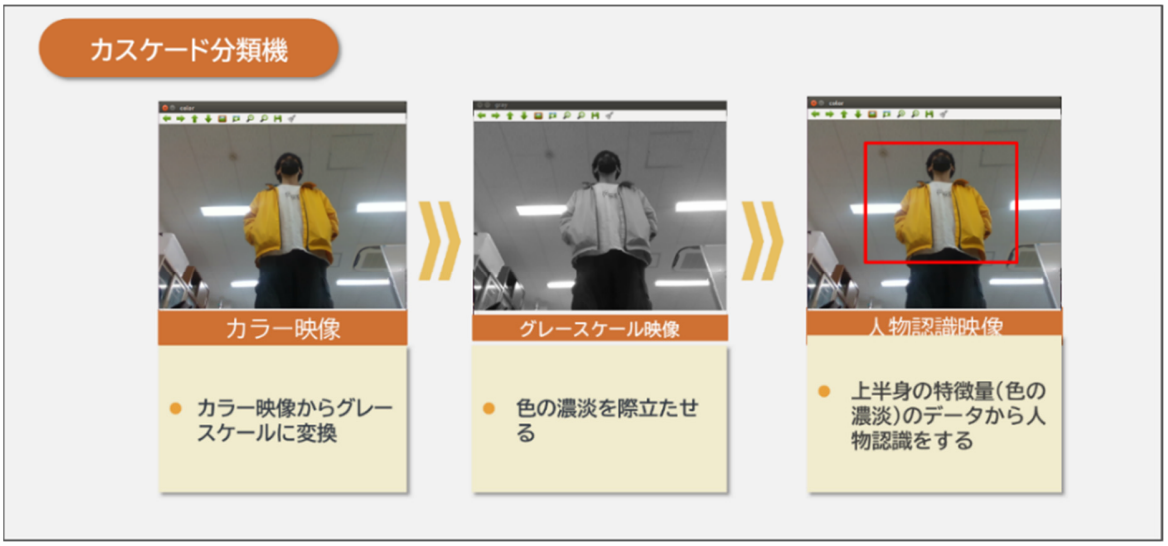


**図 4.1:カスケード分類器の仕組み**

今回用いたのは Haar-like 特徴量[13]を用いた識別器であり、複数の限られた領域の明暗差から求めることのできる特徴量である。その中でも、今回は上半身を認識するカスケード分類器(haarcascade_upperbody.xml)を使用する。

## 4.3 服の色による認識精度の変化を調べる実験

### 4.3.1 実験方法

カスケード分類器による人物認識の性能を調べるため TurtleBot3 の前の人物を、カスケード分類器を用いて認識し、認識精度を調べる実験を行う。実験使用する機器は RealSense の RGB カメラである。実験環境のイメージ図を図 4.2 に示す。実験を行った服の色は白・黄(明色)・緑(暗色)であり、人物はそれぞれの色の服を着て、TurtleBot3 に対して体の前面または後面を向けた状態で 100 フレーム分測定を行った。100 フレーム分の中に画像中の人物が赤い枠に囲まれたフレームが認識成功と判定し、そうではない場合は認識失敗と判定する。

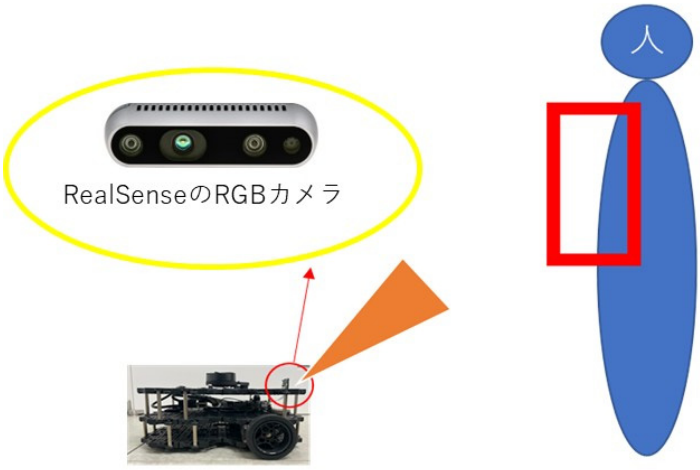


**図 4.2:実験環境のイメージ図**

### 4.3.2 実験結果

実験結果としては図 4.3 のようになり、体の前後の向きに関わる差以上に服の色によって認識率に大きな差が出ることが分かった。このグラフから分かったこととしては体の前面より背面の方が認識率に優れること及び暗い色の服が認識されやすく明るい色が認識されにくいことである。服の色によって認識率にばらつきがある点には対策が必要である。

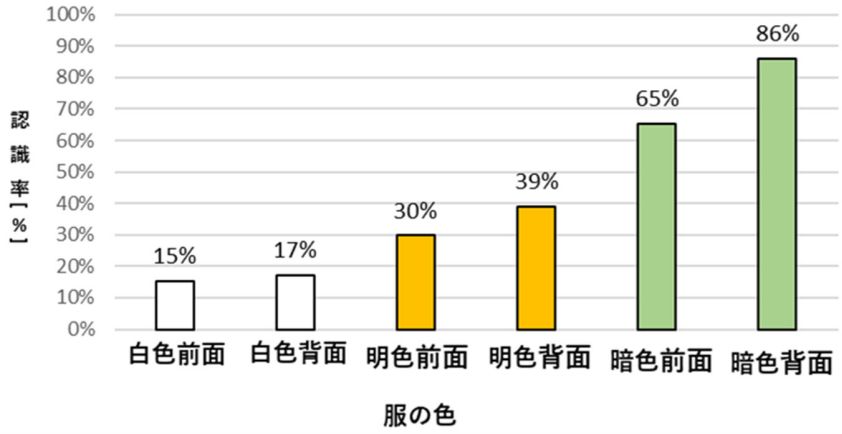


**図 4.3:服の色ごとの人物認識率比較**

## 4.4 対策手法

4.3 の実験から服の色によって認識率にばらつきがあり、対策が必要であることが分かった。白色、明色(黄色)、暗色(緑色)でここまで大きな差が出てしまった要因としては明るめの服が研究室の白い背景（壁紙）と同化してしまい、カスケード分類器が特徴量を検出できなくなってしまっていることが考えられる。これらの問題を解決する為、RealSense の深度センサによるデプス画像を利用することにした。RealSense の深度センサでは奥行きを得ることができる。それを利用し、RealSense から奥行きが 2m 以上離れている映像箇所を白く塗りつぶすことで疑似的な背景を作成する（図 4.4）。それにより、服の色に検出率が左右されず、人物を認識しやすい環境となると考えられる。


図 4.4:白い背景のイメージ図

## 4.5 仮想的な白い背景の作成手法

仮想的な白い背景の作成手法を図 4.5 で説明する。左が RealSense の RGB 画像、中央が RealSense のデプス画像である。デプス画像では、近い物体が明るい青色に、遠い物体が暗い赤色で表示される。デプス画像を用いて、2m 以上距離がある箇所を全て白く塗りつぶすことで、図 4.5 右の加工後のような仮想的な白い背景を作成することができる。この手法のプログラムへの実装は、インターネット上に公開されているものを参考にした[14]。この加工後の画像により、人物の認識率が向上するかを検証する。

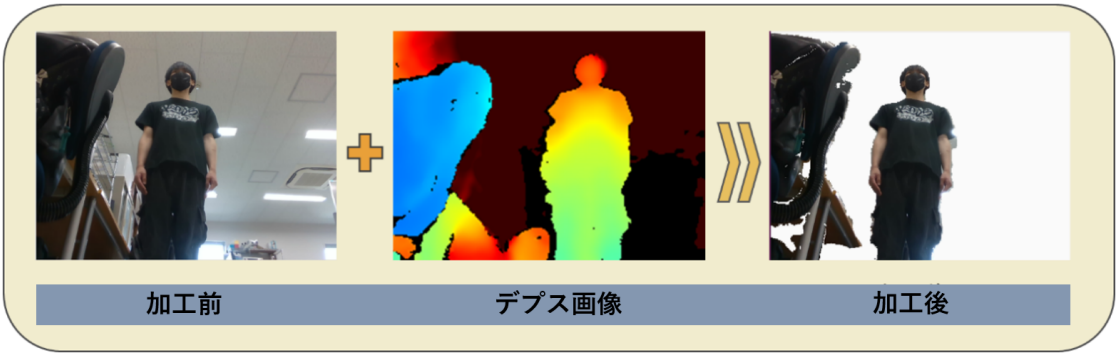


図 4.5:仮想的な白い背景の作成手法

## 4.6 白い仮想背景の効果を調べる実験

### 4.6.1 実験方法

前節で解説した白い仮想背景により人物認識率に改善が見られるかを検証する。体の前面または背面を向けて TurtleBot3 の前を往復し、それぞれ人物認識率の算出を行った。4.3 の実験と同じく行った服の色は白･黄･緑であり、それぞれの色で 100 フレーム分測定を行った。100 フレーム分の中に画像中の人物が赤い枠に囲まれたフレームが認識成功と判定し、そうではない場合は認識失敗と判定する。

### 4.6.2 実験結果

実験結果を図 4.6 に示す。4.3 で行ったオリジナル画像を用いた実験の結果と比べると白い仮想背景を追加したことで大きな改善が見られた。どの色の認識率をとっても最低89%以上と高水準であり、このことから白い仮想背景の追加が認識率の向上に有効であることがわかった。

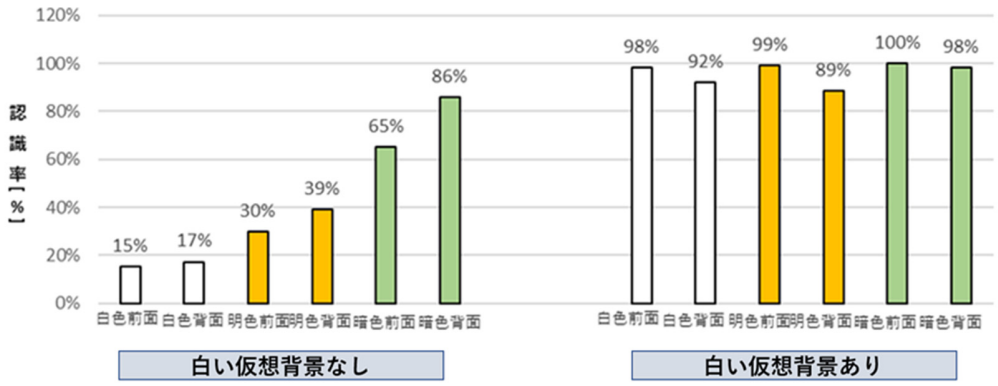


**図 4.6: 白い仮想背景の追加による人物認識率の比較**

# 第5章　人物判別手法(吉岡担当)

## 5.1 目的

第 4 章では RealSense を用いて映像を加工することで、人物認識の精度を向上させた。この人物認識では RealSense の映像範囲内の人物を全て認識する。しかし、本研究における想定環境では、映像範囲内に複数の人物が映りこむ場合があるため、追従対象となる人物を特定する必要がある。そのため、複数の人物から追従対象者を特定することを以下では人物判別と呼び、本章では服の色を用いた人物判別を行う。

## 5.2 概要

### 5.2.1 色空間の選定

ロボットカートの使用環境における明るさの変化を考慮して、服の色を用いた追従対象者の判別には、図 5.1 の右側に示される HSV 色空間を使用する。HSV 色空間では、色を色相(Hue),彩度(Saturation),明度(Value)の 3 つの成分で表現する。3 成分の中で、色相は赤や青などの色の種類、彩度は色の鮮やかさ、明度は色の明るさに関係している。このように HSV 色空間は、一般的な色の表現方法である RGB 色空間(図 5.1 左側)と異なり明度が独立しているという特徴がある。そのため、明度を除いた色相と彩度を用いて色認識をすることで、明るさの変化が色認識へ与える影響を抑えられる。したがって、本研究における人物判別には HSV 空間の色相と彩度を使用する。なお、一般的な色相と彩度の取り得る整数値の範囲は、それぞれ$0 \leq H \leq 360$と$0 \leq S \leq 100$であるが、画像処理で用いている OpenCV では、$0 \leq H \leq 179$と$0 \leq S \leq 255$で表現されており、本研究ではこれを利用する。

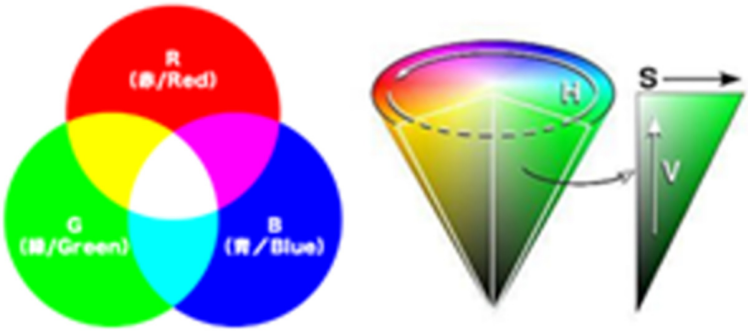


**図 5.1:RGB 色空間と HSV 色空間のイメージ図**

### 5.2.2 服色の取得

本研究では、文献[15]を参考に、人物が着用している服の色の違いにより人物判別を行う。RealSense の映像範囲に映る人物の服の色を取得するには、第 4 章の人物認識で得られた、画像内の人物位置を示す赤枠の位置を利用できる。しかし、赤枠内の範囲全体で色認識をすると、ピクセル数が多いため、リアルタイムでの色認識が困難になる。そこで、図 5.2 のように赤枠内から人物の胸部にあたる斜線で囲まれた範囲のみを対象にすることで、処理負荷を低減させる。この斜線範囲は、赤枠の横と縦の長さをそれぞれ$x$と$y$で表したときに、縦方向は赤枠の下から$\frac{y}{4}$、横方向は赤枠の中央$\frac{2}{3}x$の領域である。

斜線範囲から取得した服の色データのヒストグラムを表示したのが図 5.3 である。このグラフは、横軸が HSV の値であり、縦軸はそれぞれの HSV の値を持つピクセル数の割合である。縦軸を割合にしている理由は、カメラと人物の距離により人物認識の赤枠の大きさが変化し、取得する服の色データの数が一定ではないからである。

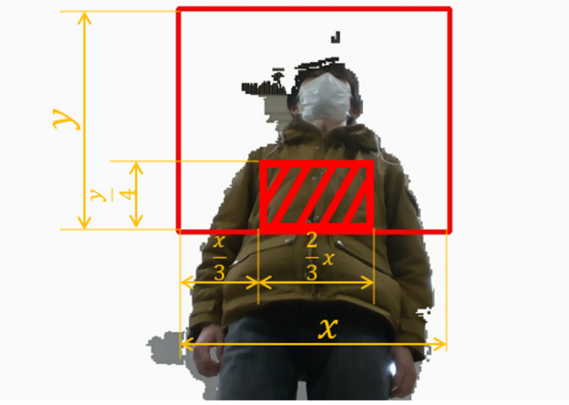


**図 5.2:服の色の取得範囲**

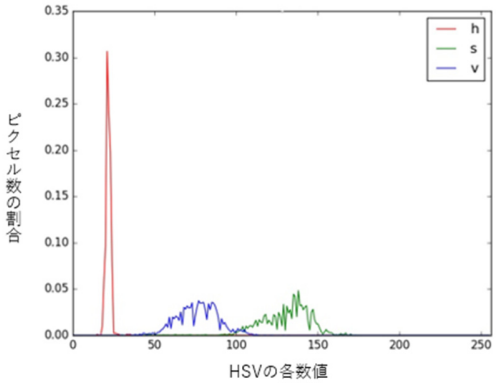


**図 5.3:HSV ヒストグラム**

### 5.2.3 HSV ヒストグラムの比較による人物判別

人物判別を行うにあたり、追従対象者の服色データを取得する必要がある。そこで、追従対象者の HSV ヒストグラムをテンプレートとして登録する。その後、カメラにとらえられた人物が追従対象者かどうかは、HSV ヒストグラム間の類似度が閾値以上かどうかで判別する。類似度の計算には以下の相関を求める式を用いる。

$$d(H_1, H_2) = \frac{\sum_I (H_1'(I) \cdot H_2'(I))}{\sqrt{\sum_I (H_1'(I)^2) \cdot \sum_I (H_2'(I)^2)}}$$

**式 5-1**

$$H_k'(I) = \frac{H_k(I) - 1}{N \cdot \sum_J H_k(J)}$$

**式 5-2**

ここで、$H_1$ と$H_2$はそれぞれ比較するヒストグラムを表す。また、式 5-2 中の N はヒストグラム横軸の区間の数（ビン数）を表す。本研究では、Python2 の cv2 モジュールに含まれる CompareHist 関数 [16]を用いてテンプレートと HSV ヒストグラムとの相関を求めた。

## 5.3 HSV ヒストグラムによる人物判別の実験

### 5.3.1 実験方法

5.2 で提案した人物判別手法の有効性を確かめる実験を行う。本実験では 3 種類の同系色の服を用意し、その内の 1 つをテンプレートとして登録する。その後、3 種類の服を着た人物がそれぞれ RealSense の映像範囲内を前後左右に動き、服ごとに人物認識した 500 フレーム分の HSV ヒストグラムとテンプレートとの類似度を求める。そして、類似度が閾値以上となる割合を調べる。追従対象だと判別する閾値は、色相かつ彩度の類似度が 0.8 以上とした。実験に用いる 3 種類の服を図 5.4 に示した。3 種類の服の内で①の服をテンプレートとしている。


**図 5.4:同系色の服 3 種類**

### 5.3.2 実験結果

図 5.5 は服ごとに追従対象者だと判別した割合を示している。テンプレートと同じ服①の判別率は 95.6%であり、テンプレートとは異なる服②と服③の判別率はそれぞれ 3.2%と 0.0%になった。図 5.6 は服ごとの色ヒストグラムの一例である。それぞれの色ヒストグラムを比較すると、色相はヒストグラムの形は似ているが割合に差があり、彩度の値は大きく異なることが見て取れる。これらの結果より、同系色の服においても正しく人物を判別できることが確認できた。

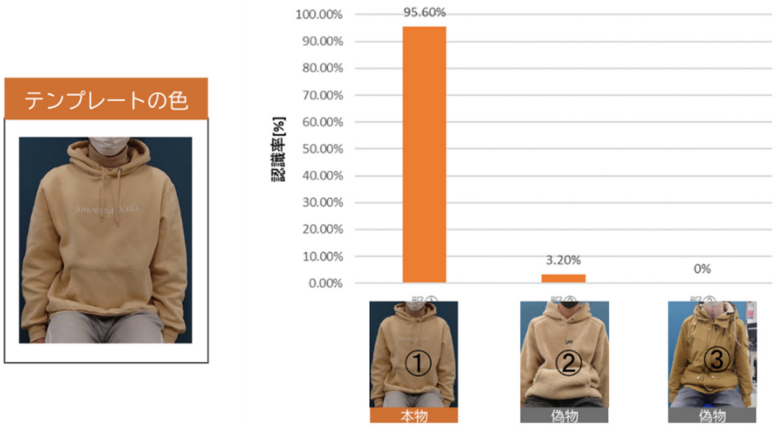


図 5.5:同系色の服の判別率

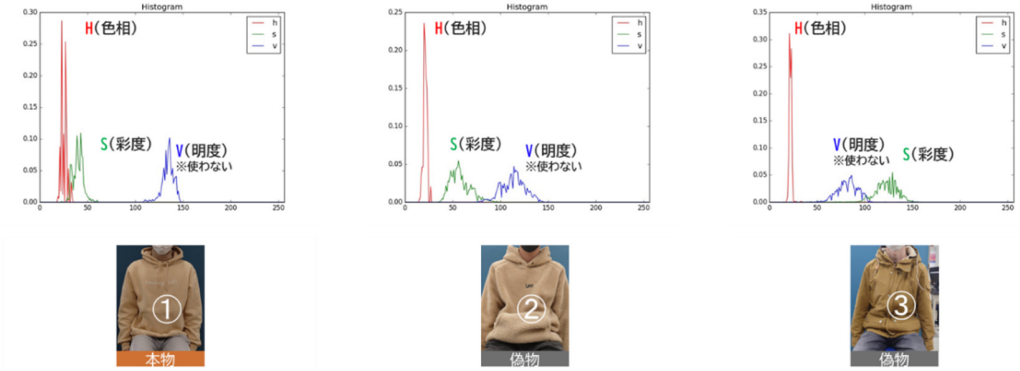


図 5.6:同系色の服のヒストグラム

# 第6章　人物追従手法(荘司担当)

## 6.1 目的

第 4 章において我々は RealSense で人物を認識することに成功し、更に白い仮想背景を設置することで高い水準で人物認識を行うことが可能となった。また、第 5 章では服の色を用いた追従対象者の判別を行った。一方、我々が想定する人物追従を行っていくためには前後左右に動く人物の追従を行うことが必須となる。その為、本章では第 4 章と第 5 章で得た人物認識と人物判別技術を活用し、前後左右に動く追従対象者へロボットカートが追従することを目標とする。

## 6.2 概要

### 6.2.1 人物追従

人物追従を実現するにあたって具体的に求められる機能は、人物が移動した際にロボットカートが人物と一定距離を保つことである。この機能を実現するには、第 4 章の人物認識で得られた、映像範囲内の人物の位置を示す赤枠の位置を利用する。すなわち、人物を認識した際、人物を認識した赤枠の映像内の位置によってロボットカートの方向を変えるのである（図 6.1）。例えば、人物が中央から左側へ移動すると人物を認識した赤枠を追うようにロボットカートも左方向に回転すればよい。

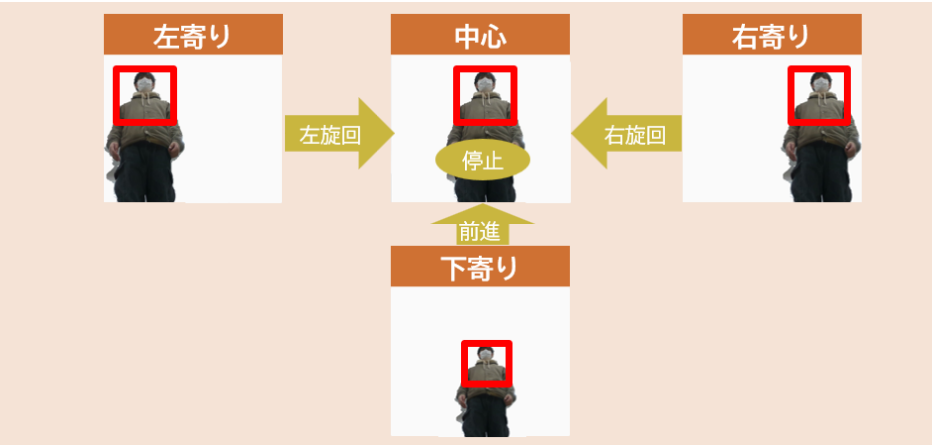


**図 6.1:赤枠の位置に基づきロボットの方向を変えるイメージ図**

## 6.3 RealSense のみでの人物追従実験

### 6.3.1 実験方法

本節では RealSense を搭載したロボットカート(図 3.1)で人物を見失うことなく、追従を継続して行うことができるか実証実験を行う。実験を行った手順は以下の通りである。

I.追従したい人物の服の色を登録する(5.2.3 参照)

II.登録を行った人物を歩行させ、左右に蛇行しながら直進する(Area1)

III.直角のコーナーを曲がる(Area2)

Area1 では直線の区間の追従を継続することができたら成功、Area2 では直角を曲がり、追従を継続することができたら成功とし、成功率を測定する。

環境のイメージ図は図 6.2 であり、図 6.3 はその写真である。

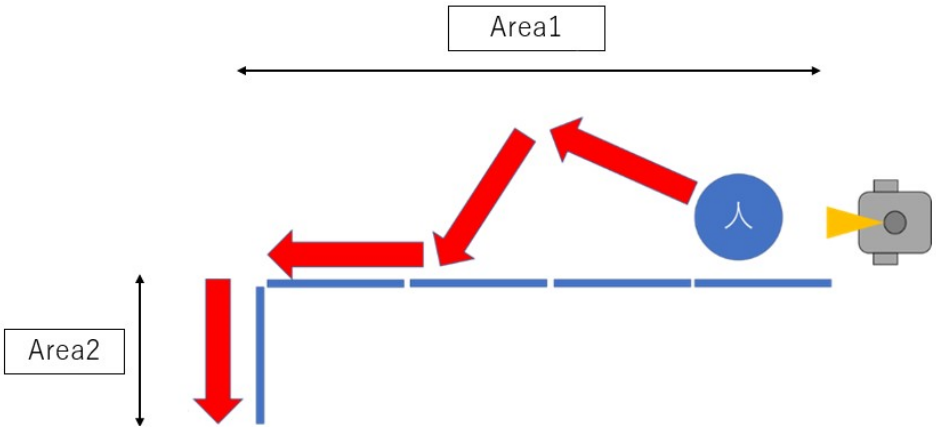


**図 6.2:実験環境のイメージ図**

**図 6.3:実験環境の写真**

### 6.3.2 実験結果

実験結果を示したのが図 6.4 である。Area1 の直線区間においては人物が左右に動くようなことがあっても高い精度で人物を追従することが可能であった。しかしながら、Area2 の直角コーナーを曲がる際、人物が物陰に隠れ（図 6.5）、追従対象を見失うことで追従の継続が困難となり、成功率が 0%となってしまった。このように、人物が物陰に隠れてしまった時に追従に失敗する問題に対して対策が必要である。

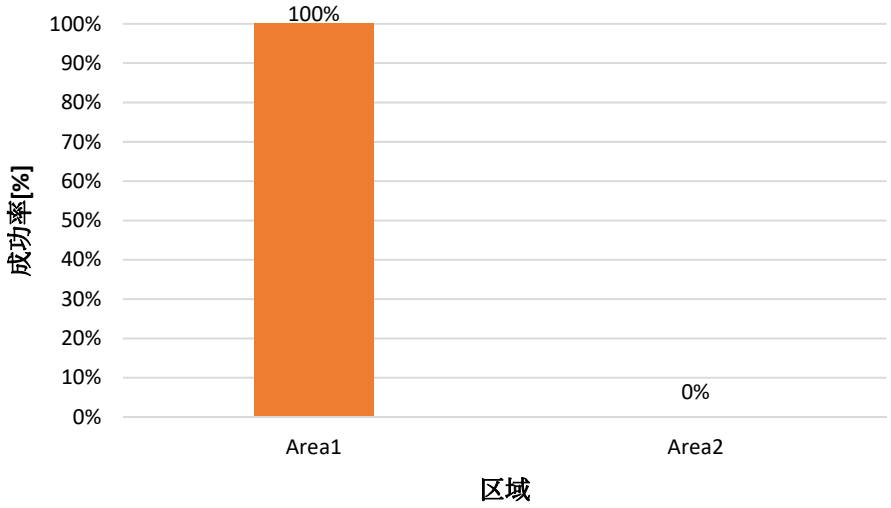


図 6.4:実験結果

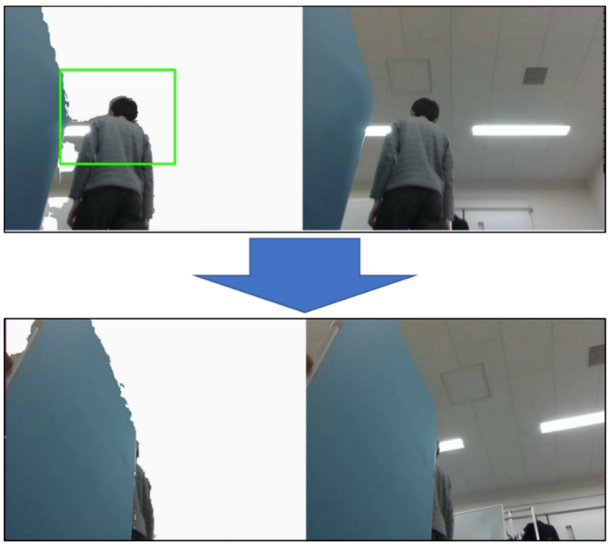

図 6.5:Area2 の直角コーナーを曲がった時の RealSense の映像

## 6.4 レーザー距離センサーを用いた人物捜索

6.3 節では高い障害物の物陰に人物が隠れてしまった時、人物追従が行えないことが問題となった。この問題に対して、レーザー距離センサー（2.4 参照）を用いて解決する。

TurtleBot3 に搭載されているレーザー距離センサーが測定する方向を図 6.6 に示す。また、付録のプログラムより該当する箇所を抜粋したものが以下である。

```
distanceL = min(data[0:29])
distanceR = min(data[331:360])
distanceML = min(data[80:90])
distanceBL = min(data[130:140])
distanceMR = min(data[270:280])
distanceBR = min(data[220:230])
distanceF  = min(min(data[0:5]), min(data[355:360]))
```

これはロボットカートの真正面を 0°とし取得したいデータの範囲を変数ごとに定義している箇所である。例えば distanceL の場合 0°から 28°の範囲にある物体までの最小距離を表しており、人物追従するロボットカートの左前方の範囲に物体があるとすれば、物体までの距離が数値として測定できるという訳である。distanceF に関しては範囲の指定の仕方が他と異なるがこれは 0°と 360°を跨ぐ真正面を測定しているためである。

レーザー距離センサーを用いて壁に沿わせながらロボットカートを移動し、障害物の物陰に隠れている人物が見える位置まで自律移動する機能を実現する。

自律移動の手順としては以下のとおりである。

I. 映像範囲内から人物が居なくなった場合にフレーム数をカウントし、100 フレーム間人物が映像範囲内にいないことが明らかになった場合、人物を最後に観測した方向の壁に向かってロボットカートが直進する。

II. 壁とロボットカートが一定の距離以内まで近づいた時、壁とロボットカートが平行になるように回転する。

III. 壁の角まで直進した時に 90°回転した後、物陰に隠れる追従対象者を認識して再追従する。

以降では自律移動を開始した後の手順について詳しく説明していく。

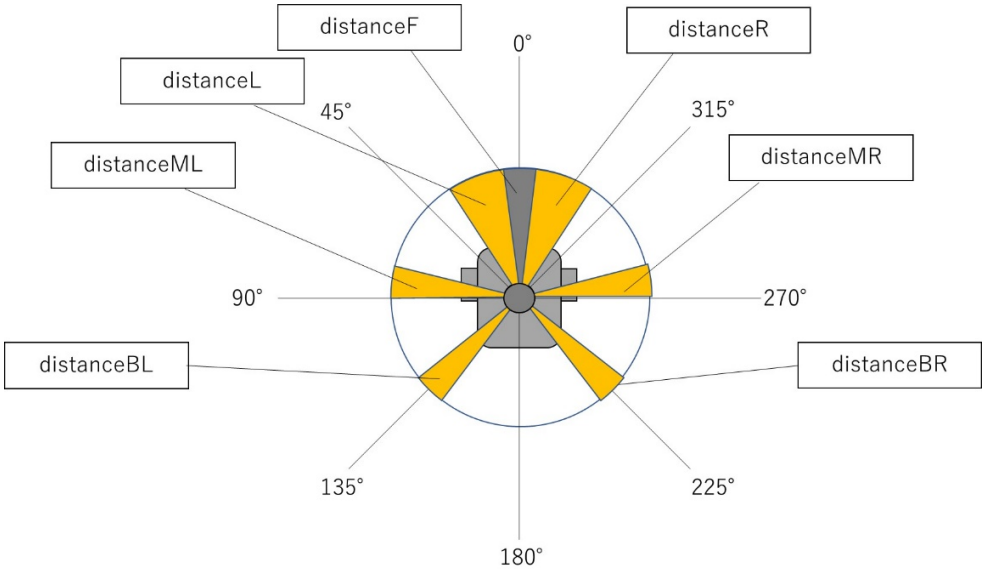


**図 6.6:TurtleBot3 上のレーザー距離センサーが測定する方向**

## 6.5 自律移動の仕組み(有限状態マシン)

### 6.5.1 概要

今回、自律移動のアルゴリズムとして有限状態マシン[17]を採用した。有限状態マシンとは 1 つの動作モード(state)から別のモードへと遷移する動的システムを表現したものであり、その特徴としてはいかなる状態においても有限個の「状態」のうち 1 つの状態をとることである。我々はこの有限状態マシンを導入する以前、条件分岐（IF 文）を用いた制御を行っていた。しかし、開発を続けていくと共に場合の数が増えプログラムが複雑になり、プログラムの変更を行う時に変更箇所が多くなってしまう問題があった。そうした問題を防ぐことができるアルゴリズムが有限状態マシンである。有限状態マシンであれば状態ごとにプログラムを構築していく為、改良を行いたい時にプログラムの一部を書き換えれば良い他、プログラムが見やすくなるという利点がある。

この有限状態マシンは「状態遷移図」を用いて表現され、本研究で用いた状態遷移図が図 6.7 である。ここでは、6.3 節の RealSense を用いた人物追従を「人物追従モード」と呼び、人物を見失った後のロボットカートの自律移動を「自律移動モード」としている。以降ではこの有限状態マシンの状態遷移の仕方について Step ごとに解説していく。

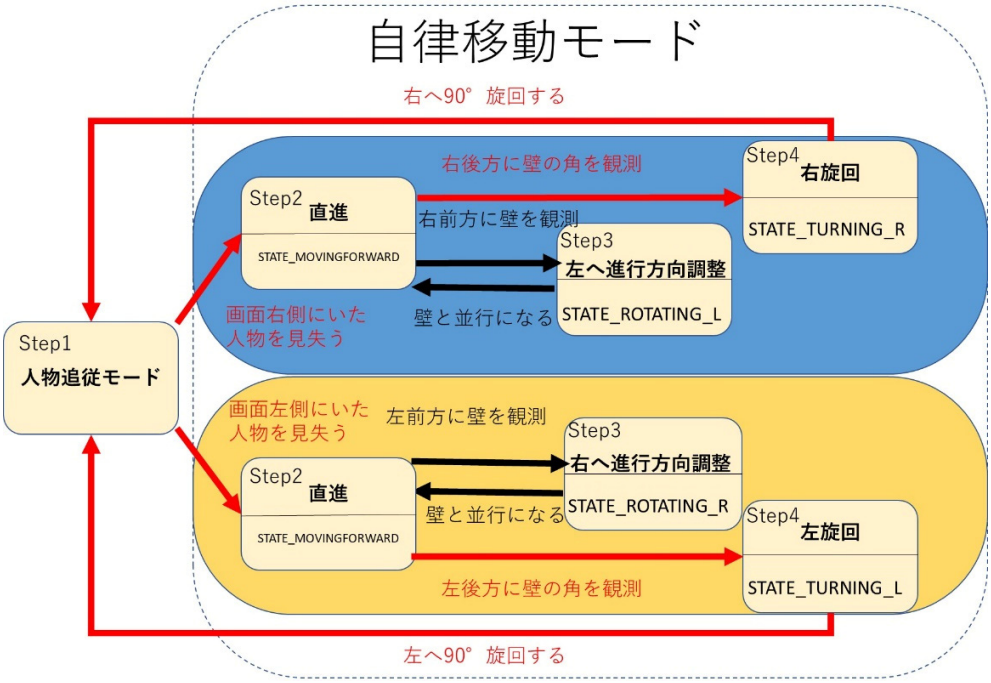


**図 6.7:ロボット制御における状態遷移図**

### 6.5.2 人物追尾モードから自律移動モードへの移行

まず、Step1 から Step2 にかけては人を見失った際に人物追従モードから自律移動モードへと移行する。遷移条件は最後に人物を観測してから人物が画角内に映らず 100 フレーム経過する条件であり、付録のプログラムより該当する箇所を抜粋したものが以下である。

```
        if not len(face_list):
            cmd_vel_pub.publish(pause_cmd)
            if distanceF>=0.7:
                not_found_cnt+=1
            else:
                not_found_cnt=0
            print("not_found:%d" % not_found_cnt)
            continue
```

一行目と二行目の文では、人物が映像範囲内で認識されなかった場合、ロボットカートへの一時停止命令である(pause_cmd)を発行している。更に、355°から 5°の範囲を現す distanceF が 0.7ｍ以上だった場合に人物が映らなかったフレーム数を示す変数 not_found_cnt に 1 ずつカウントが行われていくという仕組みである。なぜ、カウントを行うシチュエーションを distanceF が 0.7ｍ以上の場合に限定するかであるが、その理由は追従対象者がロボットカートに商品を入れるとき、RealSense の映像範囲外に出てしまい、この条件がなければ自律移動モードに移行してしまう為である。次に見失った人物が画面のどちらに消えていったかを判別している箇所を付録のプログラムより抜粋したものが以下である。

```
if (x+w/2) < (WIDTH/2-80):
    vel.angular.z=0.8
    flg=True
    direction=1
elif (x+w/2) > (WIDTH/2+80):
    vel.angular.z=-0.8
    flg=True
    direction=2
```

```
if not_found_cnt>100 and direction==1:
    not_found_cnt=0
    print("start find human phase1")
    find_human_phase1()
elif not_found_cnt>100 and direction==2:
    not_found_cnt=0
    print("start find human phase2")
    find_human_phase2()
```

上部のプログラムでは図 6.7 の状態遷移図で示すところの人物追従モードにおいて人物が最後に画角内の左右どちらに動いたかを変数を用いて管理している。左に動けば

direction=1 (phase1) になり、画角内の右に動けば direction=2 (phase2) へ変化する。下部のプログラムでは not_found_cnt が 100 カウントかつ direction==1 を満たしていた場合、find_human_phase1(以下 phase1 とする)、not_found_cnt が 100 カウントかつ direction==2 を満たしていた場合、find_human_phase2(以下 phase2 とする)が始まり自律移動が開始される。そして、Step2 から Step3･Step4 にかけては人物を最後に観測した場所に向かって移動を行う。以降の解説で重要となる単語が state というロボットカートの状態を示す変数である。この state を定義した箇所を付録のプログラムより抜粋したものが以下である。

```
STATE_MOVINGFORWARD = 0
STATE_TURNING_L = 1
STATE_TURNING_R = 2
STATE_ROTATING_L = 3
STATE_ROTATING_R = 4
```

state には 5 つの状態があり、それぞれに 0 から 4 までの数字が割り振られている。

### 6.5.3 直進と進行方向調整

次に、Step2 から Step3 への遷移について解説を行う。まず自律移動を始めた際 phase1、phase2 いずれにおいても STATE_MOVINGFORWARD という図 6.8 で示す様な直進の状態を取っている。そして直進をしている際に phase1 の場合左前方である distanceL の計測を行い、distancaL が 0.5ｍ未満になり、壁が接近した場合に図 6.9 で示す STATE_ROTATING_R へと状態が遷移する。phase2 の場合も右前方である distanceR が 0.5m 未満になり壁が接近した場合、図 6.9 で示す STATE_ROTATING_L へと状態が遷移する。STATE_ROTATING_R または STATE_ROTATING_L へ状態が遷移したら壁と平行になる様に進行方向の調整が行われる。壁と平行になり次第 STATE_ROTATING_R または STATE_ROTATING_L は Step2 の STATE_MOVINGFORWARD へと戻る。

```
state = STATE_MOVINGFORWARD
while not rospy.is_shutdown():
    if state == STATE_MOVINGFORWARD:
        print("pass1")
        print(distanceML)
        print(distanceBL)
        cmd_vel_pub.publish(move_cmd)

        if distanceL < distance_close:
            state = STATE_ROTATING_R
        elif distanceML>=1.7 and 0.5<=distanceBL<=1.0:
            print("tu-jyou")
            state = STATE_TURNING_L
    elif state == STATE_ROTATING_R:
        print("pass2")
        cmd_vel_pub.publish(rotate1_cmd)
        if distanceL >= distance_close: #distance_close は 0.5 で定義されている
            state = STATE_MOVINGFORWARD
```

```
state = STATE_MOVINGFORWARD
while not rospy.is_shutdown():
    if state == STATE_MOVINGFORWARD:
        print("pass1")
        print(distanceMR)
        print(distanceBR)
        cmd_vel_pub.publish(move_cmd)
```

```
        if distanceR < distance_close:
            state = STATE_ROTATING_L
        elif distanceMR>=1.7 and 0.5<=distanceBR<=1.0:
            print("tu-jyou")
            state = STATE_TURNING_R
    elif state == STATE_ROTATING_L:
        print("pass3")
        cmd_vel_pub.publish(rotate_cmd)
        if distanceR >= distance_close:#distance_close は 0.5 で定義されている
            state = STATE_MOVINGFORWARD
```

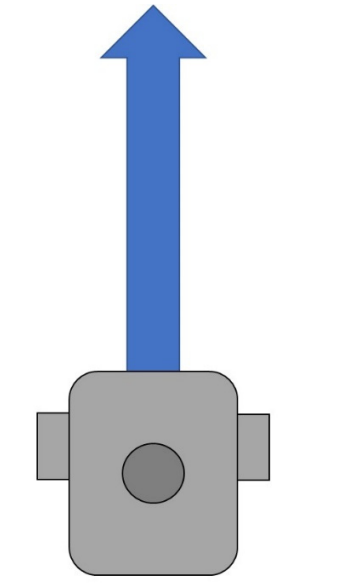


**図 6.8:STATE_MOVINGFORWARD**

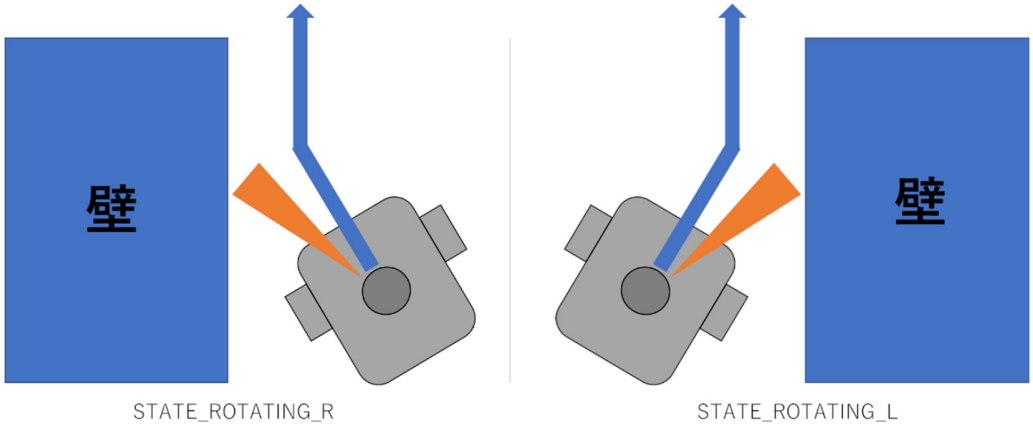


**図 6.9:STATE_ROTATING**

### 6.5.4 旋回

最後に Step2 から Step4 への遷移について解説を行う。直進をしている際に phase1 においては図 6.10 で示す様に左方向と左後方の距離である distanceML と distanceBL の計測を行う。distanceML が 1.7ｍ以上かつ distanceBL が 1ｍ以下となった場合に STATE_TURNING_L へと状態が遷移する。phase2 においては図 6.10 で示す様に右方向と右後方の距離である distanceMR と distanceBR の計測を行う。distanceMR が 1.7ｍ以上かつ distanceBR が 1ｍ以下となった場合に STATE_TURNING_R へと状態が遷移する。STATE_TURNING_L または STATE_TURNING_R へと state が変化すると旋回を行うファイルが実行される。該当する箇所を付録のプログラムより抜粋したものが以下である。

```
    elif  state == STATE_TURNING_L:
        print('turn')
        execfile("wall_left.py")
        return
```

```
    elif  state == STATE_TURNING_R:
        print('turn')
        execfile("wall_right.py")
        return
```

execfile とは実行ファイルなどの外部のプログラムを起動する命令である。ここでは旋回を実行するファイル wall_left.py、wall_right.py が起動されている。旋回を実行するファイルの詳細を以下に示す。

```
#! /usr/bin/env python
import rospy
from geometry_msgs.msg import Twist

cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size = 1)

rospy.init_node('auto')

rotate_cmd = Twist()
stop_cmd = Twist()

rotate_cmd.linear.x = 0
rotate_cmd.linear.y = 0
rotate_cmd.linear.z = 0
rotate_cmd.angular.x = 0
rotate_cmd.angular.y = 0
rotate_cmd.angular.z = 0.5

duration = 3.3 # sec

rate = rospy.Rate(10)

# required for stabilizing rospy.Time.now() in Gazebo simulation
rospy.sleep(0.0)
```

```
stop_time = rospy.Time.now() + rospy.Duration(duration)

while not rospy.is_shutdown():
   cmd_vel_pub.publish(rotate_cmd)

   if rospy.Time.now() > stop_time:
      cmd_vel_pub.publish(stop_cmd)
      break

   rate.sleep()
```

```
#! /usr/bin/env python
import rospy
from geometry_msgs.msg import Twist

cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size = 1)

rospy.init_node('auto')

rotate_cmd = Twist()
stop_cmd = Twist()

rotate_cmd.linear.x = 0
rotate_cmd.linear.y = 0
rotate_cmd.linear.z = 0
rotate_cmd.angular.x = 0
rotate_cmd.angular.y = 0
```

```
rotate_cmd.angular.z = -0.5

duration = 3.3 # sec

rate = rospy.Rate(10)

# required for stabilizing rospy.Time.now() in Gazebo simulation
rospy.sleep(0.0)

stop_time = rospy.Time.now() + rospy.Duration(duration)

while not rospy.is_shutdown():
   cmd_vel_pub.publish(rotate_cmd)

   if rospy.Time.now() > stop_time:
      cmd_vel_pub.publish(stop_cmd)
      break

rate.sleep()
```

上部が wall_left.py であり、下部が wall_right.py である。この 2 つのプログラムは rotate_cmd.angular.z という旋回を司るコマンドの数値が異なり、wall_left.py は 0.5 で左旋回、wall_right.py は-0.5 で右旋回を行う。どちらのプログラムにも記載されている duration = 3.3 # sec についてであるがこれは旋回時間を示しており、3.3 秒かけて 90°旋回が行われる。90°の旋回が終われば実行プログラムは終了する。

ここで先程示したプログラムへと処理が戻るが return 文へ到達する為、呼び出し元である人物追従モードのプログラムへと制御が戻る。以上の解説をもって再追従可能な位置ま

で自律移動をすることができる。このプログラムを用いて、人物を見失った際に再追従を行える位置まで自律移動を行えるのか実証実験を行う。

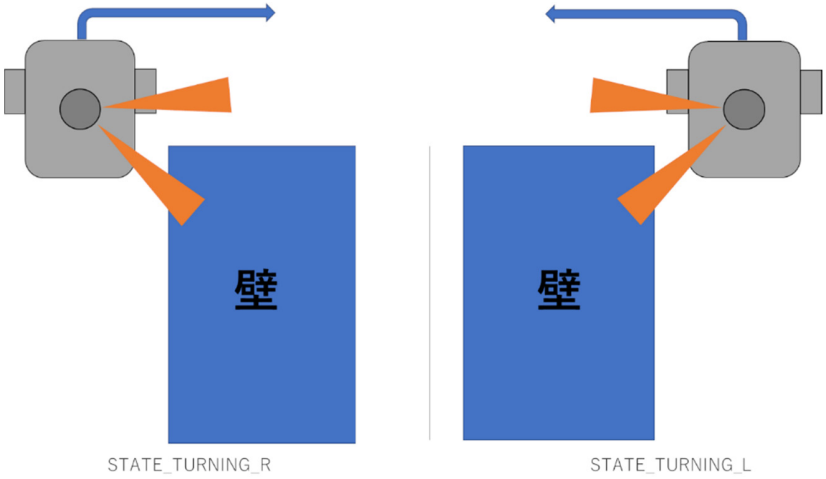


**図 6.10:STATE_TURNING**

## 6.6 有限状態マシンを用いた人物追従実験

### 6.6.1 実験方法

本節では人を見失いやすいスーパーマーケットの狭路を想定し、ロボットカートが人物を追従している最中に見失った際、有限状態マシンの考え方で再追従を行える位置まで自律移動を行えるか実証実験を行う。実験のイメージ図を図 6.11 で示す。人物が追従不能になった場合に直進、進行方向調整、旋回の過程を経て制御し、人物を再認識するまでの精度を確かめていく。実験には左右それぞれ 10 回、人物探索を行い、再追従の成功率の統計を取る。

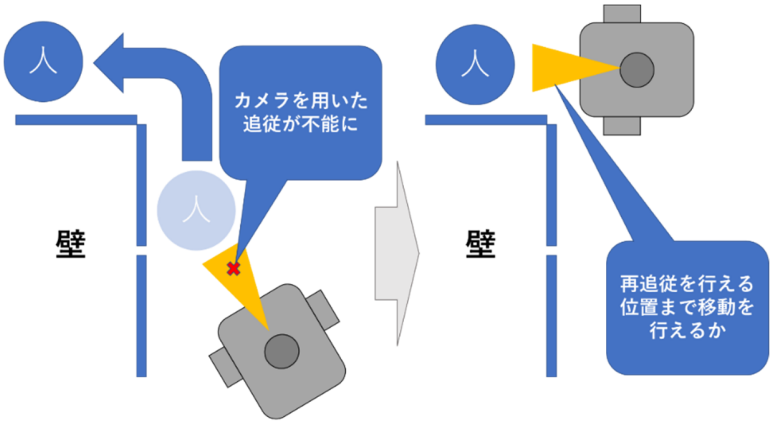


**図 6.11:実験のイメージ図**

### 6.6.2 実験結果

実験結果は図 6.12 の様になった。左旋回の精度としては 80%、右旋回の精度としては 100%であり、総合の成功率で見ると 90%という結果になった。左旋回においては精度が 80%という実用化には程遠い結果となった。失敗をしてしまった要因についてであるが 2 回ともロボットカートが壁から 1m 以上離れている状況において、壁の角をとらえず直進

してしまうことが要因であった。対策手法としては壁から 1m以上離れてしまった状態に対して新たな state を用意することが挙げられる。

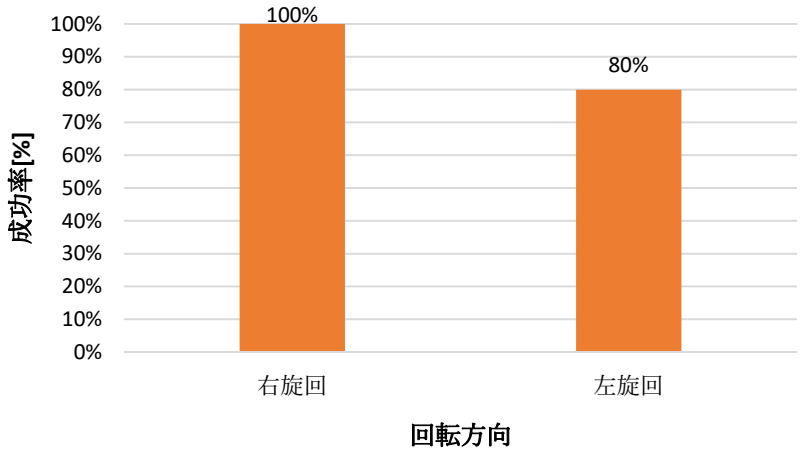


**図 6.12:実験結果**

# 第7章　結論（吉岡担当）

本研究では、「スーパーマーケットにおける人とカートの非接触を実現するロボットカートの開発」に注力し、ロボットカートの開発に取り組んだ。

3 章では、デフォルトの TurtleBot3 を改良して、買い物カートとしての使用に適したロボットカートを作成した。

4 章では、OpenCV のカスケード分類機を用いた映像範囲内の人物認識を行った。また、RealSense の深度センサーを用いて仮想的な白い背景を作ることで、人物認識の精度を向上させた。

5 章では、映像範囲内で認識された複数の人物から追従対象者を特定するために、服の色の比較によって人物判別を行い、同系色の服であっても正しく人物を判別することができた。

6 章では、追従対象者にロボットカートを追従させるために、人物認識で得られた赤枠の位置を元にロボットカートを制御し、追従対象者とロボットカートとの距離や方向を一定にすることで人物追従を可能にした。また、曲がり角でロボットカートが追従対象者を見失った場合、有限状態マシンを用いてロボットカートが自律移動をすることで、高い精度で追従対象者の再追従ができた。

これらの結果として

- RealSense に映る人物の認識に成功した
- 人物の服の色から追従対象者の人物判別に成功した
- 移動する追従対象者にロボットカートを追従させることに成功した
- 追従対象者を見失い、追従が困難になった際の再追従に成功した

という成果を得ることができた。しかし、動的障害物がある中での追従には対応していない他、ロボットの追従スピードが遅いといった実用面や安定性では問題がある。よって今回の性能評価において「スーパーで人物追従するロボットを導入する為の基礎研究」という題に対してはロボットが最低限必要な動作を行えるという点で評価できるが、実用化していくにあたっては改善していく必要があると評価した。

# 参考文献

[1]ASUKANET「「コロナショック前後のモノとの接触」に関する意識調査」
https://www.asukanet.co.jp/contents/news/2020/20200525.html
[2]厚生労働省「労働経済動向調査(令和 4 年 5 月)の概況」
https://www.mhlw.go.jp/toukei/itiran/roudou/koyou/keizai/2205/
[3]robonews.net「ロボット・ショッピングカートは実現するか？」
https://www.robonews.net/2016/07/03/5elements_dash/#more-5265
[4]ROBOTIS e-Manual「TurtleBot3」
https://emanual.robotis.com/docs/en/platform/turtlebot3/features/
[5]Switch Science「Intel RealSense Depth Camera D435」
https://www.switch-science.com/products/3633
[6]ROBOTIS e-shop「360 Laser Distance Sensor LDS-01」
https://e-shop.robotis.co.jp/product.php?id=11
[7]RT ROBOT SHOP「LIPO Battery 11.1V 1800mAh LB-012」
https://www.rt-shop.jp/index.php?main_page=product_info&products_id=2595
[8]ROBOTIS e-Manual「OpenCR 1.0」
https://emanual.robotis.com/docs/en/parts/controller/opencr10_jp/
[9]Raspberry Pi「Raspberry Pi 3 Model B」
https://www.raspberrypi.com/products/raspberry-pi-3-model-b/
[10]ROBOTIS e-shop「TB3 Waffle Plate-IPL-01 (8ea)」
https://e-shop.robotis.co.jp/product.php?id=10
[11]ROBOTIS e-shop「TB3 Ball Caster-A01 (1ea)」
https://e-shop.robotis.co.jp/product.php?id=7
[12]OpenCV「OpenCV」
https://opencv.org/
[13]群馬大学 太田研究室「Haar.like 特徴量を用いたカスケード分類器による前方車両の識別」
https://ohta.lab.inf.gunma.u.ac.jp/ailwiki/index.php?Haar.like%E7%89%B9%E5%BE%B4%E9%87%8F%E3%82%92%E7%94%A8%E3%81%84%E3%81%9F%E3%82%AB%E3%82%B9%E3%82%B1%E3%83%BC%E3%83%89%E5%88%86%E9%A1%9E%E5%99

%A8%E3%81%AB%E3%82%88%E3%82%8B%E5%89%8D%E6%96%B9%E8%BB%8A%E4%B8%A1%E3%81%AE%E8%AD%98%E5%88%A5
[14]Developers IO「インテル Realsence D435i で近くに来た物体だけを撮影してみました」
https://dev.classmethod.jp/articles/distance-detection-realsence/
[15] 松原 慎也, 池 勇勳, 梅田 和昇, "2 台のカメラを用いた骨格抽出による移動ロボットの 3 次元人物追跡," 精密工学会誌, vol.88, no.3 (2022) pp.276-281.
https://www.jstage.jst.go.jp/article/jjspe/88/3/88_276/_article/-char/ja/
[16]OPENCV 2.2 documentation 「ヒストグラム」
http://opencv.jp/opencv-2svn/py/imgproc_histograms.html
[17]MathWorks「有限ステートマシンのモデル化」
https://jp.mathworks.com/help/stateflow/gs/finite.state.machines.html

# 謝辞

最後に、この場をお借りして本研究を進めるにあたり、２年間多大なご指導を頂きました金丸隆志教授、協力して頂いた研究室の先輩方、学科の同級生に班一同、心より感謝いたします。

# 付録

人物追従を行い、人物見失い時に探索を行うプログラム

```
# -*- coding: utf-8 -*-
#!/usr/bin/env python
import pyrealsense2 as rs
import numpy as np
import cv2
import rospy
import time
from sensor_msgs.msg import Image,CompressedImage,LaserScan
from geometry_msgs.msg import Twist
from matplotlib import pyplot as plt

WIDTH = 640
HEIGHT = 480
color = (0, 0, 225)
pen_w = 3
flg=True
not_found_cnt=0
hist_cnt=0
direction=0

def callback_laser(msg):
    global distance
    global distanceL
    global distanceR
    global distanceML
    global distanceBL
    global distanceMR
    global distanceBR
    global distanceF
    data = np.array(msg.ranges)
    data = np.where(data<msg.range_min, msg.range_max, data)
    distance = min(min(data[0:30]), min(data[330:360]))
    distanceL = min(data[0:29])
    distanceR = min(data[331:360])
    distanceML = min(data[80:90])
    distanceBL = min(data[130:140])
    distanceMR = min(data[270:280])
    distanceBR = min(data[220:230])
    distanceF  = min(min(data[0:5]), min(data[355:360]))
```

```
def callback_shutdown():
    cmd_vel_pub.publish(pause_cmd)
    # ストリーミング停止
    pipeline.stop()
    cv2.destroyAllWindows()
    print("shutdown")

#映像の出力
def display_frame(frame1,frame2,name):
    images = np.hstack((frame1, frame2))
    cv2.imshow(name, images)

    cv2.waitKey(1)

#人物見失い時の障害物回避処理
def find_human_phase1():
    STATE_MOVINGFORWARD = 0
    STATE_TURNING_L = 1
    STATE_TURNING_R = 2
    STATE_ROTATING_L = 3
    STATE_ROTATING_R = 4
    state = STATE_MOVINGFORWARD

    while not rospy.is_shutdown():
        if state == STATE_MOVINGFORWARD:
            print("pass1")
            print(distanceML)
            print(distanceBL)
            cmd_vel_pub.publish(move_cmd)

            if distanceL < distance_close:
                state = STATE_ROTATING_R
            elif distanceML>=1.7 and 0.5<=distanceBL<=1.0:
                print("tu-jyou")
                state = STATE_TURNING_L

        elif state == STATE_ROTATING_R:
            print("pass2")
            cmd_vel_pub.publish(rotate1_cmd)
            if distanceL >= distance_close:
                state = STATE_MOVINGFORWARD

        elif  state == STATE_TURNING_L:
```

```
        print('turn')
        execfile("wall_left.py")
        return

    else:
        print('Doing nothing')

    rate.sleep()

#人物見失い時の障害物回避処理
def find_human_phase2():
  STATE_MOVINGFORWARD = 0
  STATE_TURNING_L = 1
  STATE_TURNING_R = 2
  STATE_ROTATING_L = 3
  STATE_ROTATING_R = 4
  state = STATE_MOVINGFORWARD

  while not rospy.is_shutdown():
    if state == STATE_MOVINGFORWARD:
      print("pass1")
      print(distanceMR)
      print(distanceBR)
      cmd_vel_pub.publish(move_cmd)

      if distanceR < distance_close:
        state = STATE_ROTATING_L
      elif distanceMR>=1.7 and 0.5<=distanceBR<=1.0:
        print("tu-jyou")
        state = STATE_TURNING_R

    elif state == STATE_ROTATING_L:
      print("pass3")
      cmd_vel_pub.publish(rotate_cmd)
      if distanceR >= distance_close:
        state = STATE_MOVINGFORWARD

    elif  state == STATE_TURNING_R:
      print('turn')
      execfile("wall_right.py")
      return

    else:
      print('Doing nothing')
```

```
    rate.sleep()

#映像と hsv ヒストグラムの取得
def get_hist():
  global not_found_cnt
  hist_h=np.empty((0,256),dtype=np.float32)
  hist_s=np.empty((0,256),dtype=np.float32)
  hist_v=np.empty((0,256),dtype=np.float32)

  while True:

    if not_found_cnt>100 and direction==1:
      not_found_cnt=0
      print("start find human phase1")
      find_human_phase1()
    elif not_found_cnt>100 and direction==2:
      not_found_cnt=0
      print("start find human phase2")
      find_human_phase2()

    frames = pipeline.wait_for_frames()
    aligned_frames = align.process(frames)
    color_frame = aligned_frames.get_color_frame()
    depth_frame = aligned_frames.get_depth_frame()
    if not depth_frame or not color_frame:
      continue

    color_image = np.asanyarray(color_frame.get_data())
    depth_image = np.asanyarray(depth_frame.get_data())

    # clipping_distance_in_metersm 以内を画像化
    white_color = 250 # 背景色
    depth_image_3d = np.dstack((depth_image, depth_image, depth_image))
    bg_removed = np.where((depth_image_3d > clipping_distance) | (depth_image_3d <= 0),
white_color, color_image)

    gray_frame=cv2.cvtColor(bg_removed,cv2.COLOR_BGR2GRAY)
    face_list =
cascade.detectMultiScale(gray_frame,scaleFactor=1.1,minNeighbors=9 ,minSize=(100,140))

    display_frame(bg_removed,color_image,"Frames")

    if not len(face_list):
```

```
        cmd_vel_pub.publish(pause_cmd)
        if distanceF>=0.7:
          not_found_cnt+=1
        else:
          not_found_cnt=0
        print("not_found:%d" % not_found_cnt)
        continue

      for(x,y,w,h) in face_list:
        hsv=cv2.cvtColor(bg_removed,cv2.COLOR_BGR2HSV)
        clrBox=bg_removed[y+h-h/4:y+h,x+w/3:x+w-w/3]
        hsvBox=hsv[y+h-h/4:y+h,x+w/3:x+w-w/3]

        display_frame(clrBox,hsvBox,"box")

        h, s, v = hsvBox[:,:,0], hsvBox[:,:,1], hsvBox[:,:,2]


        hh=cv2.calcHist([h],[0],None,[256],[0,256])
        ss=cv2.calcHist([s],[0],None,[256],[0,256])
        vv=cv2.calcHist([v],[0],None,[256],[0,256])

        pixel=len(h)*len(h[0])
        hh/=pixel
        ss/=pixel
        vv/=pixel

        hist_h=np.append(hist_h,hh.T,axis=0)
        hist_s=np.append(hist_s,ss.T,axis=0)
        hist_v=np.append(hist_v,vv.T,axis=0)

      return hist_h,hist_s,hist_v,face_list,color_image,bg_removed,depth_frame

#ヒストグラムを表示
def save_hist(hist_h,hist_s,hist_v,title):
  global hist_cnt

  plt.figure()

  plt.plot(hist_h,color='r',label='h')
  plt.plot(hist_s,color='g',label='s')
  plt.plot(hist_v,color='b',label='v')
  plt.legend()
  plt.xlim([0,256])
```

```
    plt.title(title+str(hist_cnt))

    plt.savefig("feedback/histogram/"+title+str(hist_cnt)+".jpg")

    return

def save_photo(img,x,y,w,h,title):
    hsv=cv2.cvtColor(bg_removed,cv2.COLOR_BGR2HSV)

    clrBox=bg_removed[y+h-h/4:y+h,x+w/3:x+w-w/3]
    hsvBox=hsv[y+h-h/4:y+h,x+w/3:x+w-w/3]

    images = np.hstack((clrBox,hsvBox))

    cv2.imwrite("feedback/recog_photo/"+title+str(hist_cnt)+".jpg",images)

    return

cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size = 10)
scan_sub = rospy.Subscriber('scan', LaserScan, callback_laser)

rospy.init_node('auto')

# ストリーミング初期化
config = rs.config()
config.enable_stream(rs.stream.color, WIDTH, HEIGHT, rs.format.bgr8, 30)
config.enable_stream(rs.stream.depth, WIDTH, HEIGHT, rs.format.z16, 30)

# ストリーミング開始
pipeline = rs.pipeline()
profile = pipeline.start(config)

# 距離[m] = depth * depth_scale
depth_sensor = profile.get_device().first_depth_sensor()
depth_scale = depth_sensor.get_depth_scale()
clipping_distance_in_meters = 2.3 #230cm 以内を検出
clipping_distance = clipping_distance_in_meters / depth_scale

# Align オブジェクト生成
align_to = rs.stream.color
align = rs.align(align_to)

threshold = (WIDTH * HEIGHT * 3) * 0.95
```

```
cascade_file = "/home/student/haarcascades/haarcascade_upperbody.xml"
cascade = cv2.CascadeClassifier(cascade_file)

vel=Twist()
move_cmd = Twist()
pause_cmd = Twist()
rotate_cmd = Twist()
rotate1_cmd = Twist()

move_cmd.linear.x = 1.5
move_cmd.linear.y = 0
move_cmd.linear.z = 0
move_cmd.angular.x = 1.5
move_cmd.angular.y = 0
move_cmd.angular.z = 0

pause_cmd.linear.x = 0
pause_cmd.linear.y = 0
pause_cmd.linear.z = 0
pause_cmd.angular.x = 0
pause_cmd.angular.y = 0
pause_cmd.angular.z = 0

rotate_cmd.linear.x = 0
rotate_cmd.linear.y = 0
rotate_cmd.linear.z = 0
rotate_cmd.angular.x = 0
rotate_cmd.angular.y = 0
rotate_cmd.angular.z = 0.5

rotate1_cmd.linear.x = 0
rotate1_cmd.linear.y = 0
rotate1_cmd.linear.z = 0
rotate1_cmd.angular.x = 0
rotate1_cmd.angular.y = 0
rotate1_cmd.angular.z = -0.5

distance = 100
distanceL = 100
distanceR = 100

distanceML = 40
distanceBL = 40
distanceMR = 40
```

```
distanceBR = 40
distance_close = 0.5

rate=rospy.Rate(40)

rospy.on_shutdown(callback_shutdown)

time.sleep(3)
temp_h,temp_s,temp_v,face_list,color_image,bg_removed,depth_frame=get_hist()

save_hist(temp_h[0],temp_s[0],temp_v[0],"Template")
for(x,y,w,h) in face_list:
  save_photo(bg_removed,x,y,w,h,"Template")

while not rospy.is_shutdown():
  hist_h,hist_s,hist_v,face_list,color_image,bg_removed,depth_frame=get_hist()

  i=0
  for (x,y,w,h) in face_list:

    h_dist=cv2.compareHist(temp_h[0],hist_h[i],cv2.HISTCMP_CORREL)
    s_dist=cv2.compareHist(temp_s[0],hist_s[i],cv2.HISTCMP_CORREL)
    v_dist=cv2.compareHist(temp_v[0],hist_v[i],cv2.HISTCMP_CORREL)

    #登録人物と一致
    if h_dist>=0.8 and s_dist>=0.8:
      not_found_cnt=0
      cv2.rectangle(bg_removed,(x,y),(x+w,y+h),(0,255,0),thickness=pen_w)

      if (x+w/2) < (WIDTH/2-80):
        vel.angular.z=0.8
        flg=True
        direction=1
      elif (x+w/2) > (WIDTH/2+80):
        vel.angular.z=-0.8
        flg=True
        direction=2
      else:
        vel.angular.z=0
        flg=False
      cmd_vel_pub.publish(vel)

      if flg:
        continue
```

```
        depth=depth_frame.get_distance(x+w/2,y+h)
        print("depth:%f" % depth)

        if depth>0.8:
            cmd_vel_pub.publish(move_cmd)
        else:
            cmd_vel_pub.publish(pause_cmd)

    #登録人物と不一致
    else:
        print("(hs:%d)[%d]  h:%f,s:%f,v:%f" % (hist_cnt,i,h_dist,s_dist,v_dist))
        save_hist(hist_h[i],hist_s[i],hist_v[i],"Histogram")
        save_photo(bg_removed,x,y,w,h,"photo")
        hist_cnt+=1
        not_found_cnt+=1
        print("not_found:%d" % not_found_cnt)
        cmd_vel_pub.publish(pause_cmd)
        cv2.rectangle(bg_removed,(x,y),(x+w,y+h),(255,0,0),thickness=pen_w)
    i+=1

display_frame(bg_removed,color_image,"Frames")

rate.sleep()
```