2019 年度（令和元年度）

創造工学セミナーⅡ Final Report

デプスカメラと画像処理を用いた撮影補助装置の開発

研究メンバー

s516023 工藤 翔太

s516049 中野 駿

s516059 藤井 周作

s516061 眞野 峻彰

指導教員

金丸 隆志 教授

目次

第 1 章 緒言（工藤担当）

1.1 研究背景

1.1.1 研究テーマ

近年、動画視聴サイトの利用が急増している。図 1-1 に示したように、スマートフォンからのインターネットの利用者数は増加傾向にあり、また図 1-2 に示すように無料動画サービスの利用者数も増え続けている。スマートフォンを持っている人の多くが動画サービスを利用しているといえると考えられる。

図 1-3 の 2018 年のスマホ動画サービスの利用率を見ると YouTube の利用者数が最も多い。それに関連し、YouTube で動画を投稿する「YouTuber」という職業も新たに生まれており、YouTube チャンネル数も増加傾向にある(図 1-4)。動画投稿が身近になる中で、ソロで活動する YouTuber が抱える問題の一つに、カメラを動かしながら自身を撮影する手段が少ないという問題がある。この問題を解決するために私たちは、追従機能を搭載した撮影補助装置の開発をする。テーマ遂行にあたり、私たちは図 1-5 のデプスカメラ「RealSense」を用いる。以下、デプスカメラを「RealSense」と称して説明していく。

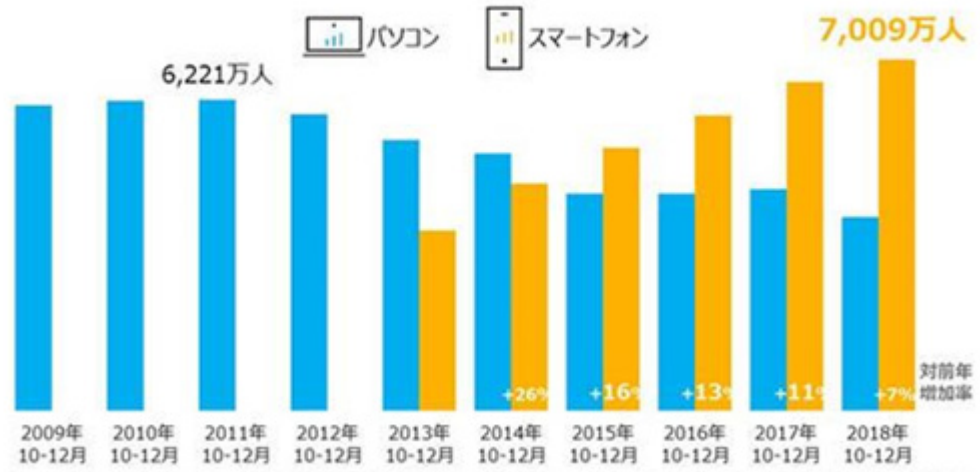

図 1-1 パソコンとスマートフォンからのインターネット利用者数の推移[1-1]

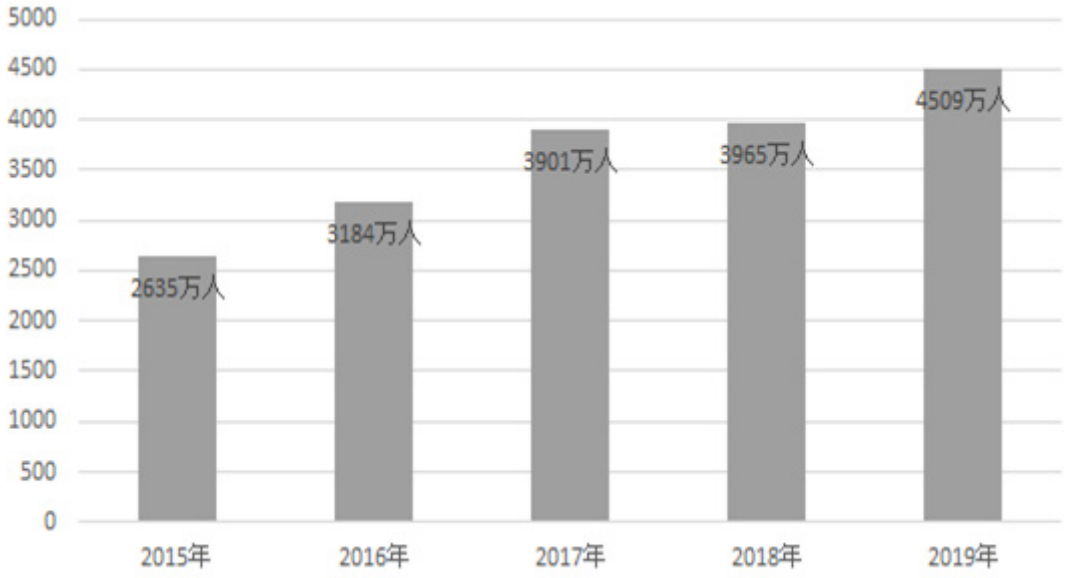

図 1-2 無料動画サービスの利用者数の推移[1-2]

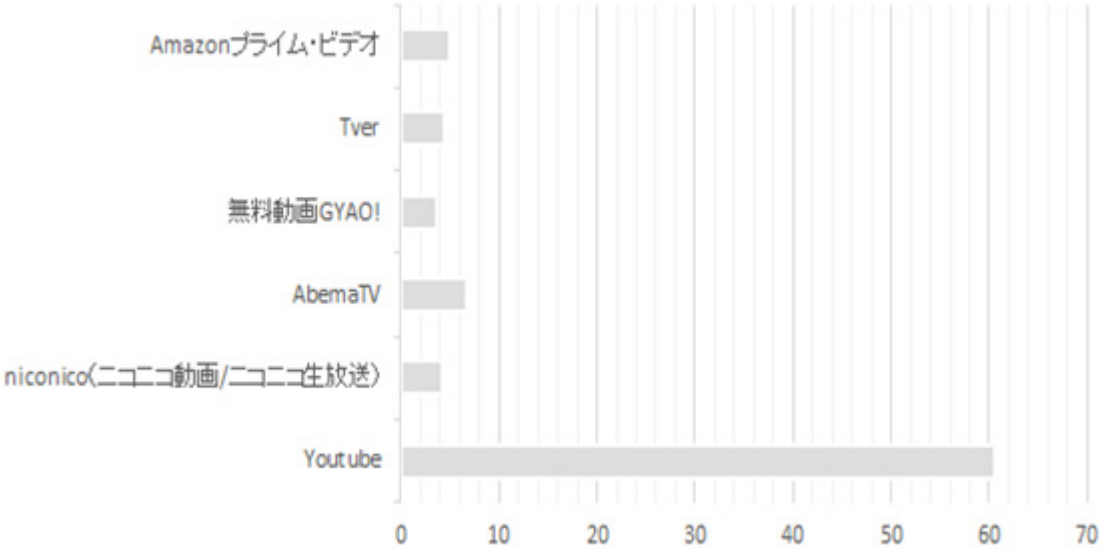

図 1-3 スマホ動画サービスの利用率（2018 年）[1-2]

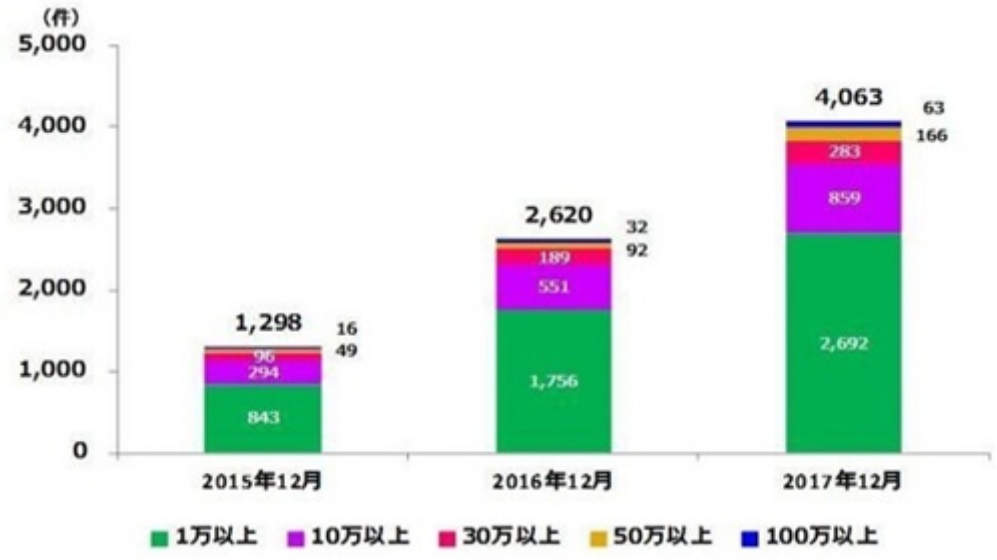

図 1-4 YouTube チャンネル数の推移[1-3]

図 1-5 RealSense D435[1-4]

1.1.2 RealSense D435

本項では本研究で使用する RealSense D435 について説明する。

RealSense は Intel 社が発売しているデプスカメラである。深度計測が可能なステレオビジョンの深度カメラで、深度センサ、RGB センサ、IR 投射器を搭載し、USB 給電で動作する。また、グローバルシャッターと広い視野角を備え、移動体を測定したい場合やデバイス自体が動く場合に高分解能の深度情報を提供する。また、死角を最小限に抑え、より広いエリアをカバーする。主な仕様を表にすると以下のようになる。

表 1-1 RealSense D435 の仕様[1-4]

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 90 fps
最小深度距離	0.2m
最大範囲	10m +パフォーマンスの精度、シーン、照明条件に応じて異なる
RGB 解像度	1080p @ 30fps
コネクタ	USB 3.0

1.2 先行研究

1.2.1 ステレオカメラによる距離計測を用いた移動ロボットの製作

本研究で製作する撮影補助装置に関連し、先行研究で開発したロボットを図 1-6 に示す。先行研究の目的はステレオカメラを用いた移動ロボットの考案と製作であった。ステレオカメラより取得した画像から Depth 画像を作成し、距離計測および対象物の検出を行なった。それ以前の研究で使用していた自作のステレオカメラから既製品のステレオカメラに変更したものの、Depth 画像から得られる距離の精度が低いという問題があった。今回の私たちの研究では、ステレオカメラに替わって RealSense を用いることにより、精度の問題を解消できると考えられる。

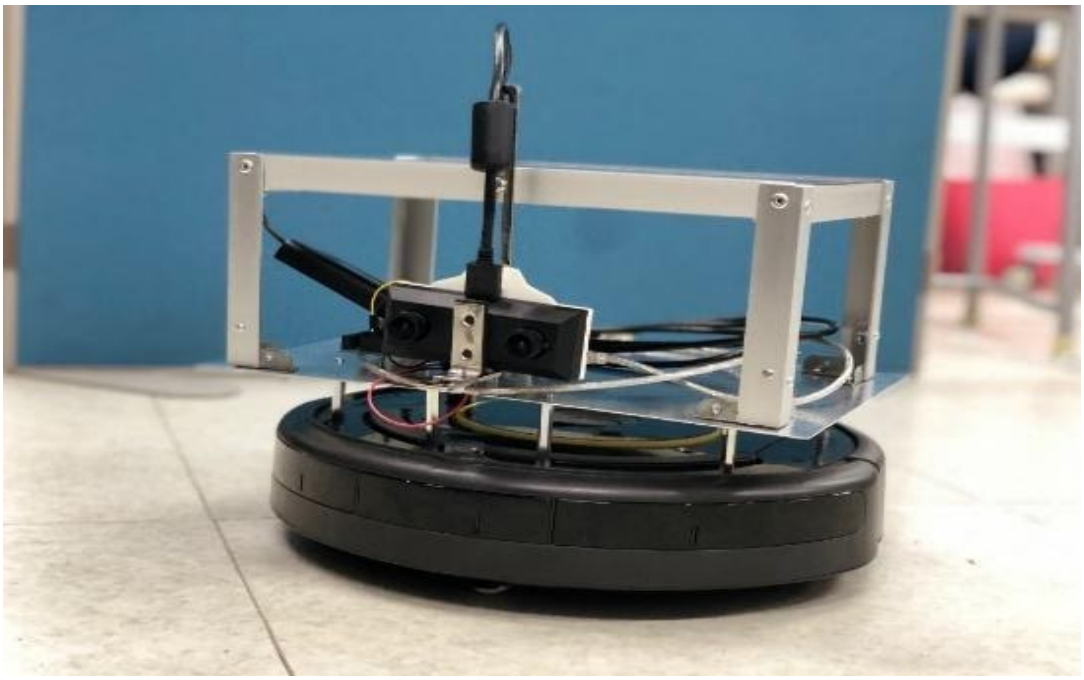

図 1-6 先行研究の移動ロボット[1-5]

1.3 研究概要

1.3.1 製作物の決定

撮影補助装置を製作するにあたって、カメラに映った人の検出を可能とするシステムを開発する。その際、人間の肌色と人間の形とをそれぞれ検出することで人物追従を実現する。それにより、これまで困難だった一人での動画撮影が可能になる。

なお、本研究の使用環境として、室内を想定する。

図 1-7 新たに製作した撮影補助装置

1.3.2 他社製品との比較

私たちが考える追従ロボットの例と類似した商品に Doog 社の「サウザー」がある。人や台車の追従機能を備え持つことにより、大型の倉庫などで人の運搬作業を支援する。センサとして使用されているのはレーザーセンサであり、追従対象を区別することができる。さらに無人ライン走行機能を備えており、また 2 つの機能の切り替えも簡単に出来る。高視野レーザーセンサで障害物を検出し、衝突を回避できること、昼夜を問わず追従運搬機能ができるなど、多様環境に対応していることがこの製品の強みである。

図 1-8 サウザー（Doog 社）[1-6]

また、SOLOSHOT 社の「SOLOSHOT3」は身体に装着したタグを自動で追尾撮影するロボットカメラである。GPS タグ（送信機）を自動追尾し、最大 4 時間の動画が撮影できる。さらに、動画撮影において自動でパン、チルト、ズームし最大 2000 フィート（約 610m）離れた場所から撮影できる。

図 1-9 SOLOSHOT３（SOLOSHOT 社）[1-7]

これらの類似した商品と私たちのシステムの違いは２点ある。まず、サウザーに搭載しているレーザーセンサを用いず、RealSense のみで人検出をして追従していく点である。また、SOLOSHOT3 とは異なり GPS タグを使用せずに人を検出できる点である。

1.3.3 研究の方向性

私たちは研究を進めていく上で、項目ごとに以下のように目標を設定した。

撮影補助装置の設計

動画撮影をサポートする装置を製作する。装置に使用する土台は以下の条件を満たすものとする。

1. 動画撮影ができるよう、撮影用スマートフォンを固定できる。
2. RealSense、Arduino、ノートパソコンなどの使用機器を搭載できる。
3. 移動ロボットに搭載可能な重さの土台である。

画像処理

1. RealSense により取得した RGB 画像と Depth 画像を使用し、人検出及び距離計測を実現する。
2. その結果を用い、撮影補助装置を実現する。

撮影補助装置の制御

1. Arduino を用いて移動ロボットの制御を行う。
2. 前進後退や回転の動作を行うためのプログラムを作成する。

以上の目標をそれぞれ達成することで、撮影補助装置を完成させる。

第 2 章 ハードウェアの設計（中野担当）

2.1 概要

2.1.1 目的

本章では本研究で製作する撮影補助装置のハードウェアの設計について解説する。

2.1.2 要求機能、制約条件

撮影補助装置の製作にあたって必要な要求機能、制約条件を以下に示す。

1. 必要な機器を搭載できる構造であること（使用機器については 2.2.2 項で述べる）。
2. 分解が容易な構造であること。

これらを踏まえて進めていく。

2.2 撮影補助装置の製作

2.2.1 撮影補助装置のアイデア

本格的な設計、製作に入る前に、2.1.2 項の条件を満たす撮影補助装置の構造について検討した結果、図 2-1 のようなものとなった。

撮影補助装置に必要な機器を搭載出来る空間が必要なため、この構造とした。詳細は、2.2.2 項で述べる。

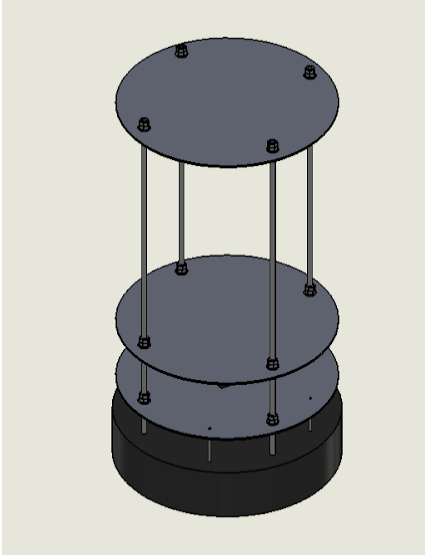

図 2-1 撮影補助装置のアイデア図

2.2.2 撮影補助装置の構造

本項では 2.2.1 項で紹介した撮影補助装置の構造について解説する。

上段は RealSense と動画撮影用のスマートフォンを搭載する。これは人（特に顔）を撮影するためには最も高い位置に設置する必要があるからである。中段はノートパソコンの Surface Pro 4 を搭載する。Surface Pro 4 を設置するために上段との間隔を広くとる必要がある。下段はルンバと接続する Arduino を搭載する。それぞれの機器については 2.2.3 項で解説する。

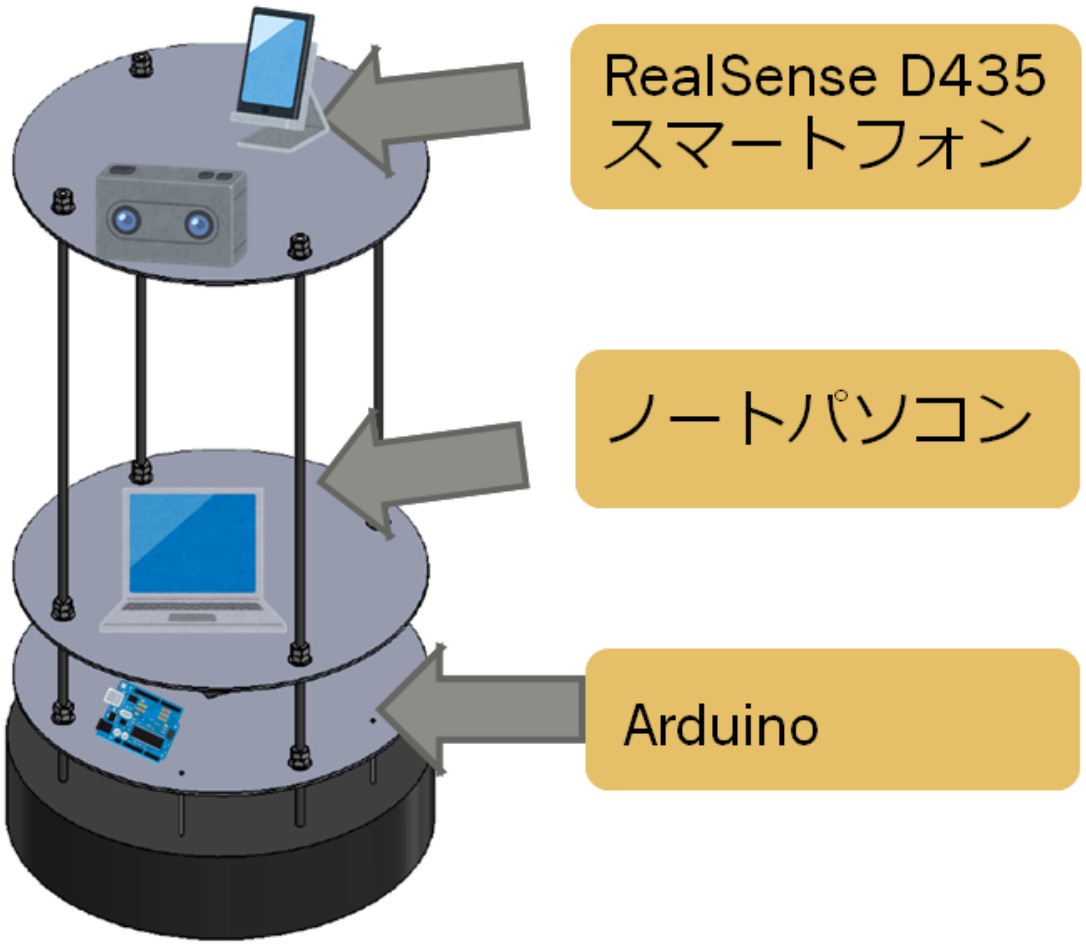

図 2-2 完成予想図

2.2.3 使用機器

本項では今回の撮影補助装置に搭載した機器について解説する。

Roomba 622

本研究の撮影補助装置の足には iRobot 社より発売されているロボット掃除機 Roomba 622（以下ルンバとする）を用いる。ルンバはルンバオープンインターフェイス（以下 ROI とする）と呼ばれるソフトウェアインターフェイスを使用する事でルンバを制御することが出来る。ROI の詳細は後の 3.2.2 項で解説する。

図 2-3 Roomba 622

表 2-1 Roomba の仕様

サイズ	最大幅 340mm×高さ 92mm
重さ	約 3.7kg(バッテリー含む)
電源方式	充電式ニッケル水素電池(充電時間:約 3 時間)
稼働時間	通常清掃時/最大 60 分

Arduino Mega 2560（以下 Arduino とする）

Arduino Mega 2560 と呼ばれるマイコンボードをルンバに対し指令を送るために用いる。図 2-4 および図 2-5 に示したようにルンバと比べて小さいため、撮影補助装置に容易に搭載することが出来る。USB ポートを備えるため、USB ケーブルでパソコンからの指令を受け取ることが出来る。Arduino Mega 2560 の仕様を以下の表 2-2 に示す。

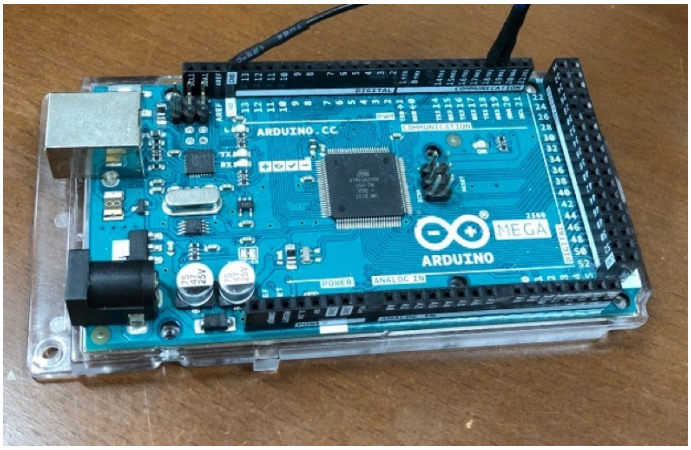

図 2-4 Arduino Mega 2560

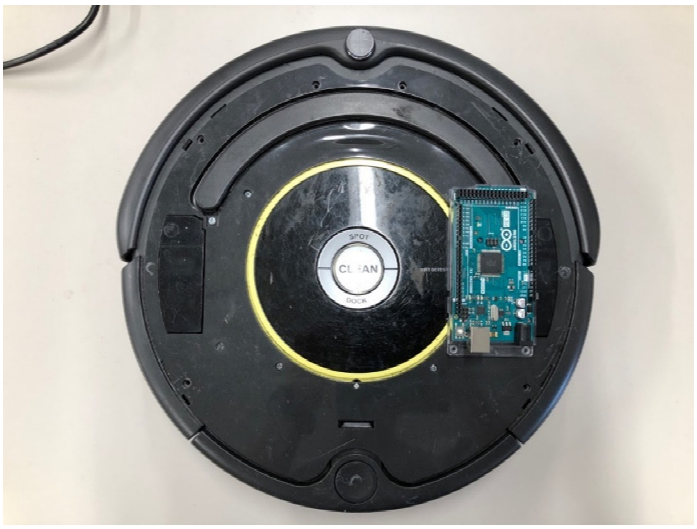

図 2-5 Roomba 622 と Arduino Mega 2560 の大きさの比較

表 2-2 Arduino Mega 2560 の仕様

MCU	Atmel ATmega 2560
動作電圧	5V
入力電圧(推奨)	7～12V
入力電圧(限界)	6～20V
デジタル入出力ピン	54 ピン
アナログ入力チャネル	16 ピン
I/O ピンでの直流電流	40mA
3.3V ピンでの直流電流	50mA
クロック速度	16MHz
サイズ(W×H×D)	114mm×57mm×15mm

Surface Pro 4（以下 Surface とする）

ソフトウェア開発に必要な Visual Studio や画像認識に必要な OpenCV を動かすためのパソコンとして Surface Pro 4 を用いる。動画の処理を行うために必要な性能を持ち、さらに、撮影補助装置に搭載可能なサイズを持つものを選んだ。

図 2-6 Surface Pro 4

表 2-3 Surface Pro 4 のスペック

OS	Windows10
CPU	Intel Core i7-6650U
GPU	Intel Iris Graphics 540
RAM	16GB
サイズ(W×H×D)	240mm×200mm×87mm

2.3 撮影補助装置の台座について

本節では、撮影補助装置の設計と製作について解説する。図面は 3 次元 CAD である SolidWorks を用いて作成した。

2.3.1 撮影補助装置上段の解説

本項では、撮影補助装置上段について解説する。

2.2.2 項の通り、撮影補助装置上段には RealSense と動画撮影用のスマートフォンを搭載する。

まず、直径 330mm、厚さ 3mm の円形アルミ板を用意し、ボール盤で図 2-7 のような穴を空ける。φ7 の穴は後述する自撮り棒をねじで固定するためのものである。φ3 の穴は RealSense をねじで固定するために用いる。φ10 の穴は撮影補助装置の柱となるねじ棒を通すために用いる。また、φ3 の穴とφ10 の穴は互いに干渉しないよう 25mm ほど離す。加工後に、動画撮影用のスマートフォンを取り付けるための自撮り棒(図 2-8)と RealSense を取り付ける。その詳細は 2.4.3 項にて述べる。

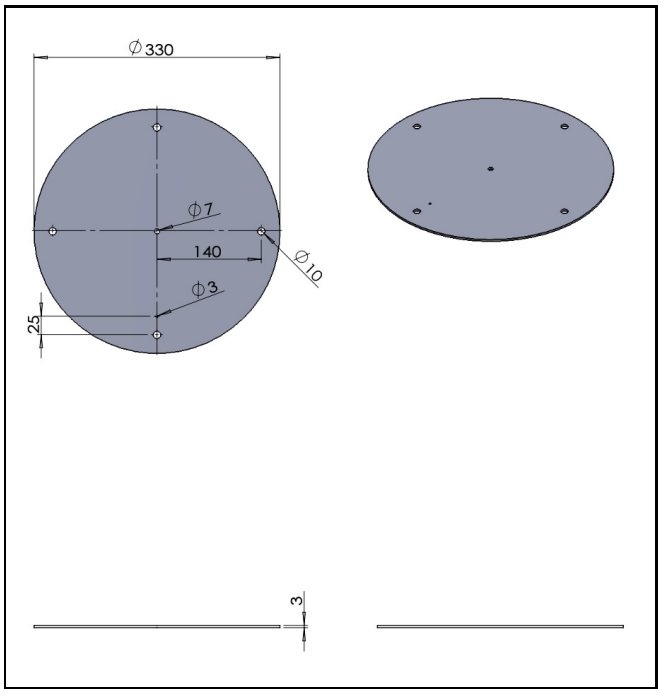

図 2-7 撮影補助装置上段の図面

図 2-8 自撮り棒

2.3.2 撮影補助装置中段の解説

本項では、撮影補助装置中段について解説する。

2.2.2 項の通り、撮影補助装置中段には Surface を搭載する。

製作手順は 2.3.1 項と同様である。また、撮影補助装置の動作中に Surface が落ちることがないよう、ゴム製の滑り止めマットを敷いた。

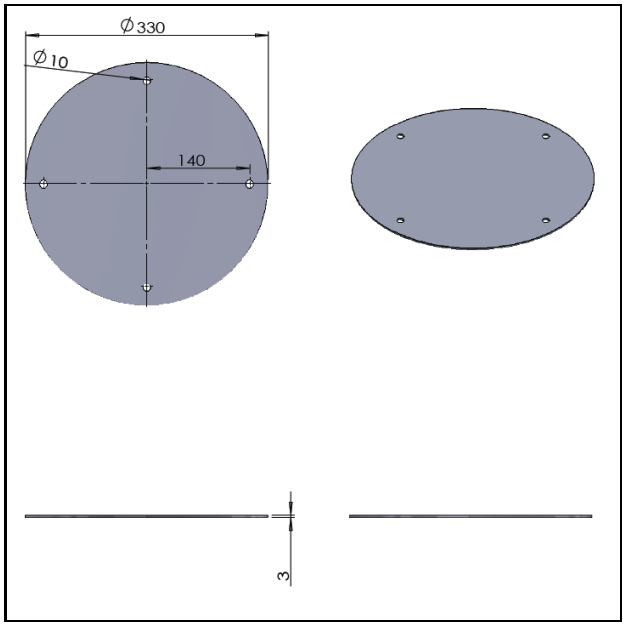

図 2-9 撮影補助装置中段の図面

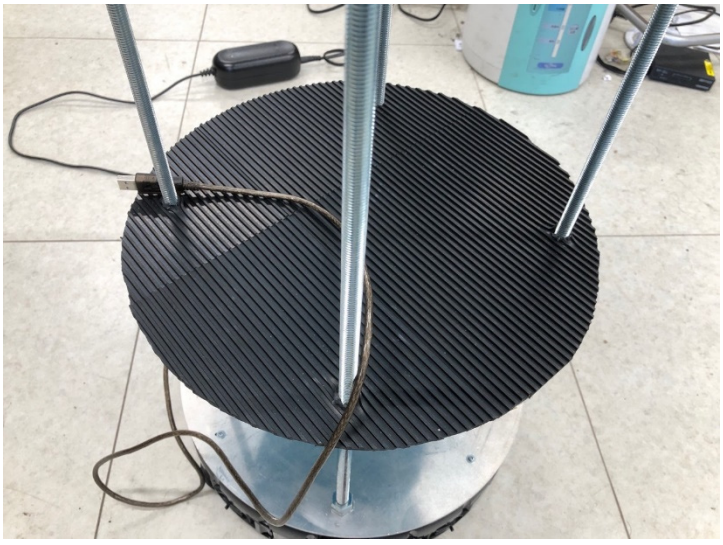

図 2-10 敷いた滑り止めマット

2.3.3 撮影補助装置下段の解説

本項では、撮影補助装置下段について解説する。

2.2.2 項の通り、撮影補助装置下段には Arduino を搭載する。

製作手順は 2.3.1 項と同様である。なお、下段にはねじ棒を通す穴の他に、ルンバと固定するためのφ3のねじ穴とルンバの電源ボタンとシリアルポートコネクタにアクセスするための窓を空けている。80㎜×50㎜の窓は大学の放電加工機を用いて空けた。

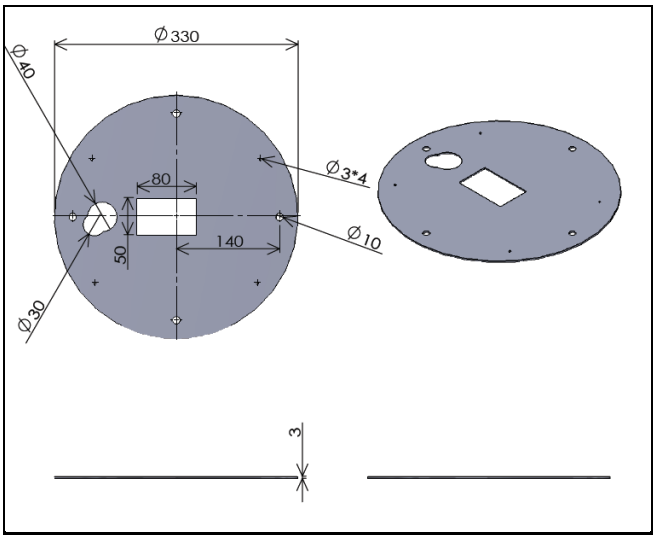

図 2-11 撮影補助装置下段の図面

図 2-12 放電加工機

2.3.4 撮影補助装置の柱の加工

撮影補助装置の柱として図 2-13 の棒ねじを用いる。M8×1000mm の棒ねじを 600mm に切断して実現した。

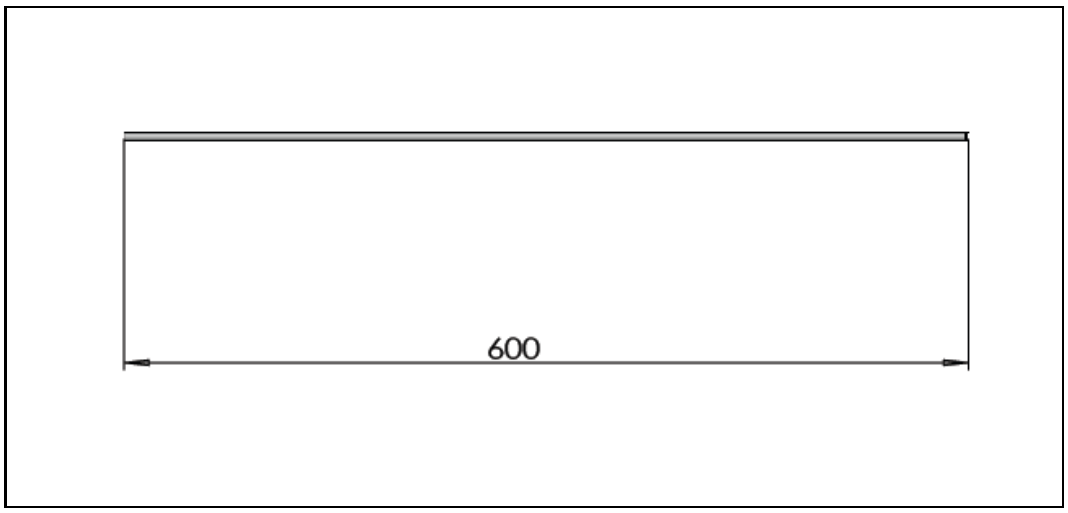

図 2-13 棒ねじの図面

2.4 撮影補助装置の組み立て

2.4.1 アルミ板と棒ねじの固定

2.3.1 項～2.3.3 項のアルミ板と 2.3.4 項の棒ねじを固定するために図 2-14 のようにナットを用いてアルミ板を挟みこんだ。これで、2.1.2 項で述べた分解が容易な構造であるという条件を満たした。

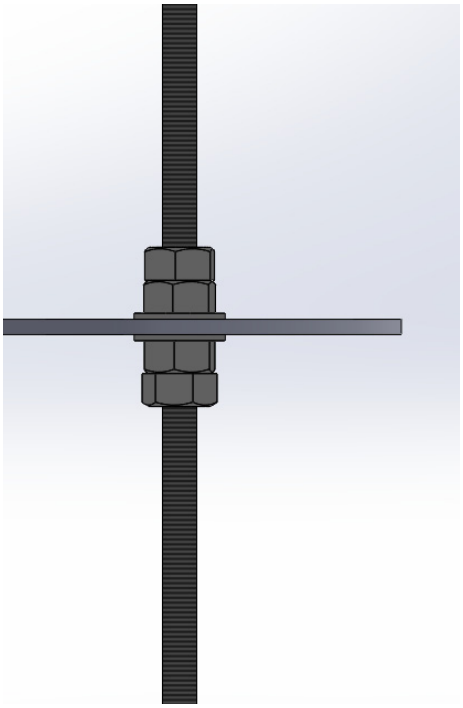

図 2-14 アルミ板と棒ねじの固定

2.4.2 アルミ板とルンバの固定

アルミ板とルンバの固定のために図 2-15 の固定部品（ねじ、ワッシャー、六角スペーサー）を組み立てた。固定部品を使って、図 2-16 のようにアルミ板とルンバを固定した。

図 2-15 固定部品

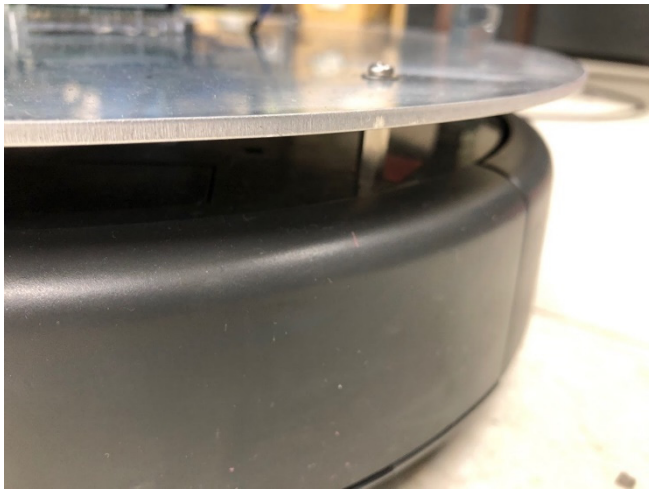

図 2-16 アルミ板とルンバの固定部位

2.4.3 自撮り棒、RealSense の取り付け

自撮り棒と RealSense にはそれぞれ図 2-17 および図 2-19 のようなねじ穴があるため、図 2-18 および図 2-20 のようにアルミ板を挟んでねじ止めすることで固定した。

図 2-17 自撮り棒のねじ穴

図 2-18 固定した自撮り棒

図 2-19 三脚のねじ穴

図 2-20 固定した三脚

図 2-21 は組み立て後の撮影補助装置の写真である。Arduino と Surface、RealSense の接続は第 3 章にて解説する。

図 2-21 組み立て後の写真

2.5 使用した部品

最後に、撮影補助装置の製作に使用した部品を表 2-4 に記す。

表 2-4 撮影補助装置の部品

部品名	寸法	個数
アルミ板	φ330mm、t=3mm	3
棒ねじ	M8×1000mm	4
ナット	M8×20mm	40
なべ小ねじ	M3×30mm	4
六角スペーサー	M3×20mm	4
ワッシャー	M3×8mm	4
滑り止めマット	φ330mm	1

第 3 章 ルンバと Arduino、Surface の接続と制御（中野担当）

3.1 概要

3.1.1 目的

本章では、ルンバと Arduino、Surface の接続方法と制御方法について解説する。さらに、ルンバオープンインターフェイス(ROI)についても解説する。

3.1.2 要求機能

本章で解説するルンバを Arduino、Surface により制御するには、以下の機能を使用する。

1. ルンバと Arduino のシリアル通信
2. Arduino と Surface のシリアル通信

3.2 ルンバと Arduino、Surface の接続

3.2.1 ルンバオープンインターフェイス（ROI）

ルンバオープンインターフェイス(ROI)とはルンバの行動を制御および操作したり、センサから情報を取得するためのソフトウェアインターフェイスである。ルンバには外部機器とシリアル通信を可能にする図 3-2 の ROI 端子があり、Arduino などと接続することで、ルンバを制御する事が出来る。

図 3-1 Roomba

図 3-2 ルンバの ROI 端子

3.2.2 ルンバと Arduino の接続方法

ルンバの ROI 端子はピンごとの機能を示したのが図 3-3 および図 3-4 である。今回のルンバと Arduino の接続には表 3-1 のものを用いる。

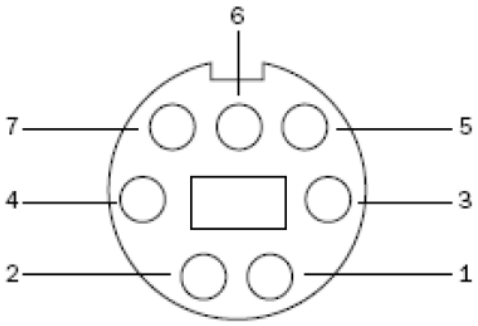

図 3-3 ルンバの ROI 端子のピン配列

Pin	Name	Description
1	Vpwr	Roomba battery + (unregulated)
2	Vpwr	Roomba battery + (unregulated)
3	RXD	0 – 5V Serial input to Roomba
4	TXD	0 – 5V Serial output from Roomba
5	BRC	Baud Rate Change
6	GND	Roomba battery ground
7	GND	Roomba battery ground

図 3-4 ルンバの ROI 端子の各ピンの名前と説明

表 3-1 ルンバと Arduino のピン番号の関係

ルンバのピン番号	用途	Arduino のピン番号
3	入力(ルンバ)―出力(Arduino)	18(TX1)
4	出力(ルンバ)―入力(Arduino)	19(RX1)
6 または 7	GND	GND

3.2.3 接続に使う使用機器

本項では、ルンバと Arduino の接続に使う機器について解説する。

1. ルンバ
2. Arduino Mega 2560

ルンバ、Arduino Mega 2560 については 2.2.3 項を参照すること。

3. シリアルケーブル

Arduino とルンバの ROI 端子を接続するに当たって、ROI 端子がミニ DIN7pin プラグに対応していることから、図 3-5 と表 3-2 のパーツを使用して図 3-6 のようなシリアルケーブルを作成した。作成したケーブルをルンバの ROI 端子に接続したのが、図 3-7 である。

図 3-5 ミニ DIN7pin プラグ

図 3-6 作成したシリアルケーブル

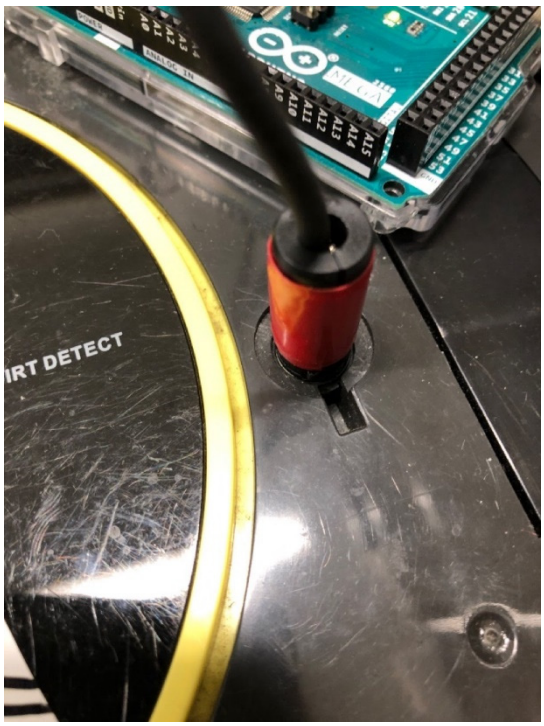

図 3-7 ルンバと接続したシリアルケーブル

表 3-2 シリアルケーブルの材料

名前	用途
ミニ DIN プラグ 7pin	ROI 端子に接続する
3 軸ケーブル	ミニ DIN プラグ 7pin とジャンパーワイヤを繋ぐ
ジャンパーワイヤ	Arduino に接続する
収縮チューブ	露出している導線部を覆う

3.2.4 Surface と Arduino の接続

Arduino は USB ポートを備えており、USB ケーブルと図 3-8 の USB ハブを使用して、図 3-9 のように Surface と接続できる。

図 3-8 USB ハブ

図 3-9 Surface と Arduino の接続

3.3 ルンバの制御

3.3.1 制御方法

ルンバの制御を行うためには、コマンドを Surface からルンバに送信する必要がある。ルンバには進んだ距離や角度のデータを得るためのエンコーダが搭載されている。これを用いると進む距離や角度を指定出来る。今回の研究では、昨年の研究で使用されていた、エンコーダーから得られる距離と角度を指定した数値と一致させるプログラムを流用した（プログラムは付録を参照すること）。

本研究で使用したコマンドを表 3-3 に示す。大文字または小文字のアルファベットの後に 4 桁の数字を入力し Surface から Arduino を経由して送信することで、ルンバを制御することが出来る。小文字の s は強制停止である。また、大文字の S と 4 桁の数字で速度の変更が可能である。コマンドの入力例とその動作は表 3-4 のようになる。

表 3-3 使用したコマンドの仕様

命令 ID	動作パターン	アルファベット	数字	備考
1	前進	W	0+3 桁の数字	
2	後進	X		指定した距離だけ進んで停止する
3	右旋回	D		
4	左旋回	A		指定した度数だけ旋回して停止する
5	前進	w		
6	後進	x		指定した速度で進み続ける
7	右旋回	d		
8	左旋回	a		指定した速度で旋回し続ける
9	強制停止	s	必要なし	
0	速度変更	S	0+3 桁の数字	1～4 の時の速度が変わる

※大文字の前進と後進の単位は[cm]、大文字の右旋回と左旋回時の単位は[°]であり、数字の範囲は 3 桁でないといけない。速度の単位は[mm/s]である。

表 3-4 コマンドの入力例と動作

コマンド	動作
W0030	前へ 30cm 進んで停止する
D0050	右へ 50° 旋回して停止する

3.3.2 動作確認

3.2.3 項、3.2.4 項で記した接続方法で Surface、ルンバ、Arduino を接続した。そして、Arduino 開発環境である Arduino IDE を Surface 上で実行して 3.3.1 項の動作確認を行った。

Arduino IDE のツールから図 3-10 のようなシリアルモニタを開く。そこで、「W0030」などのコマンドを入力および送信して動作確認を行ったところ、前進、後進、右旋回、左旋回、強制停止の全てが正常に行われることを確認した。

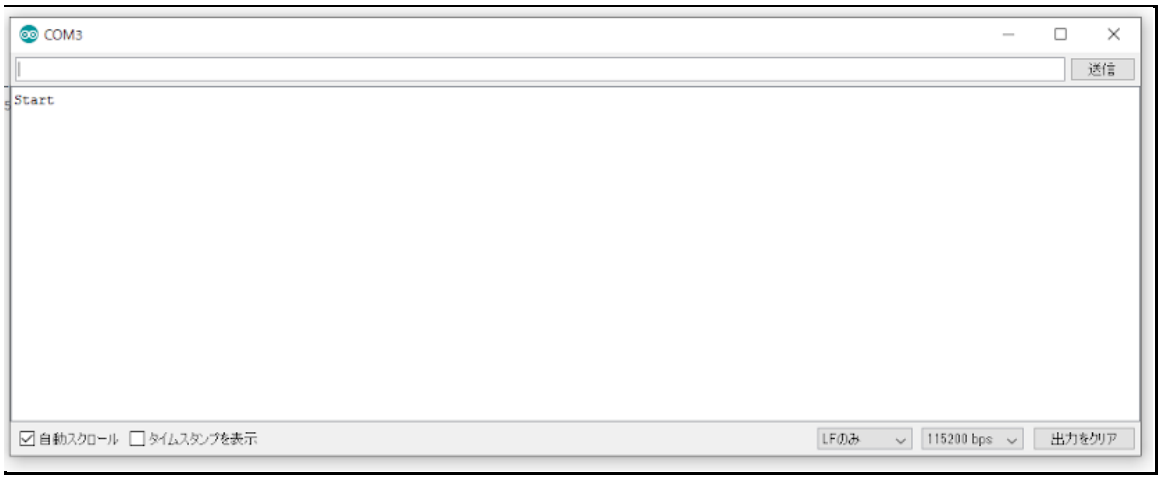

図 3-10 Arduino IDE のシリアルモニタ

第 4 章 人認識と距離測定（藤井担当）

4.1 概要

4.1.1 目的

撮影補助装置が人を追従するためには、人検出および人までの距離測定が必要になる。そこで、人検出と距離測定を同時に行えるプログラムを本章で解説する。

4.1.2 要求仕様

本研究では、距離測定を RealSense から得られる Depth 画像(図 4-1)を用いて行う。Depth 画像とは、RealSense からの距離に応じて色分けされた画像のことである。また、人検出には Depth 画像に加え、図 4-2 の RGB 画像も用いる。2 種類の画像を 1 つのカメラから取得できるため、装置に搭載する機器が少なくなるという効果もある。

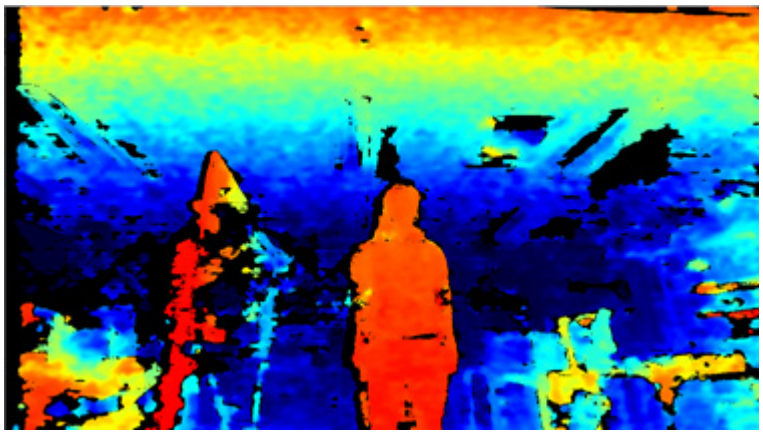

図 4-1 RealSense から取得できる Depth 画像

図 4-2 RealSense から取得できる RGB 画像

人の検出には、動画撮影の妨げになるマーカーを用いない検出方法として、以下の 3 つの手法を用いる。

1. 顔検出
2. 肌色検出
3. 形検出

顔検出と肌色検出は RGB 画像から、形検出は Depth 画像から画像処理を行い実現する。

上記の検出方法は、4.3 節、4.4 節、4.5 節で詳しく説明する。

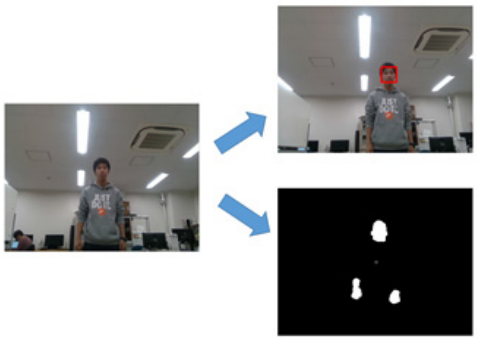

図 4-3 RGB 画像を用いた検出方法

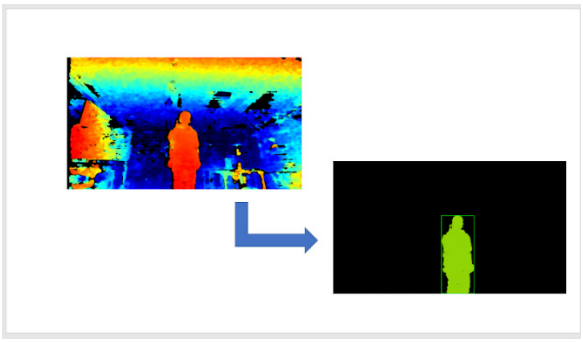

図 4-4 Depth 画像を用いた検出方法

4.1.3 使用したソフトウェア

人検出と距離測定を行なうための開発環境を解説する。

1. Visual Studio 2017

プログラムの開発ツールとして、 Visual Stadio 2017 を使用する。Visual Stadio 2017 はマイクロソフトのソフトウェア開発製品群および、それらを管理する統合開発環境である。

2. OpenCV 3.1.4

OpenCV とは、Intel 社が開発した画像処理･画像解析を行なうライブラリである。言語は、C/C++、 Java、 Python が使用可能である。また、Windows、Linux、 Mac OS、Android、iOS が公式にサポートされている。

4.2 距離測定の方法

4.2.1 概要

撮影補助装置が人と距離を一定に保つためには、装置と人との距離を測定する必要がある。本項では、距離の測定方法を解説する。

4.2.2 距離測定の方法

4.1.2 項で記したように、RealSense を使用して距離の計測を行なう。距離情報の取得には、RealSense SDK に提供される get_distance 関数を使用する。下記のコードの X、Y に Depth 画像上の座標を入力することで距離を取得できる。

```
dist_something = dframe.get_distance(X, Y);
```

4.3 顔検出

4.3.1 概要

本節では人検出方法の一つである顔検出について解説する。

顔検出では、RGB 画像から人の顔を検出する。本研究では、顔検出ができた場合はその領域を赤い矩形で囲む（図 4-5）。また、RGB 画像で取得した顔の座標を get_distance 関数に与えることで距離情報を得る。

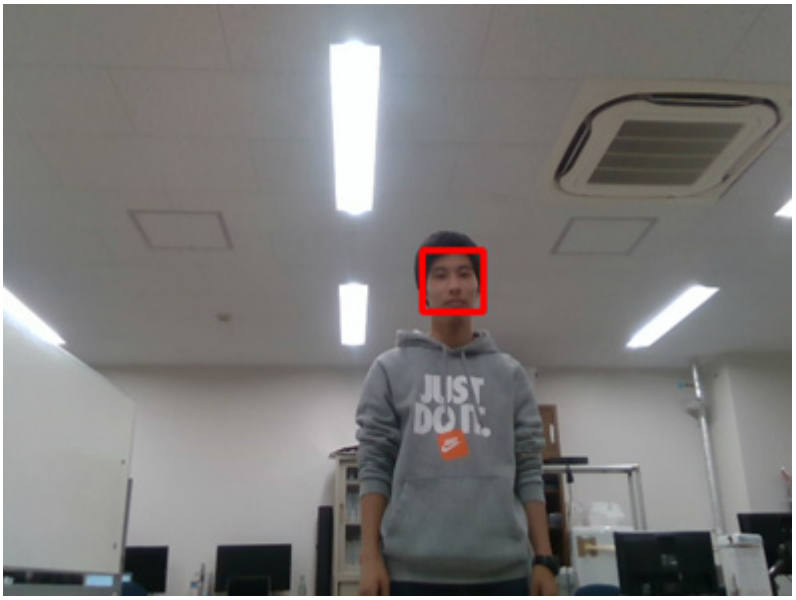

図 4-5 顔を検出して矩形で囲っている様子

4.3.2 検出方法

顔検出を実現するために、OpenCV 内のカスケード分類器を使用した。カスケード分類器とは、検出したいものの特徴をまとめたデータのことである。通常はカスケード分類器を作成するためには、大量の画像を用意してプログラムに学習させる必要がある。OpenCV にはあらかじめ人の正面顔を判断するカスケード分類器が用意されている為、本研究ではそれを使用する。

```
CascadeClassifier cascade; //カスケード分類器格納場所
cascade.load("C://git//opencv-3.4//data//haarcascades//haarcascade_frontalface_alt.xml"); //正面顔情報が入っているカスケード
vector<Rect> faces; //輪郭情報を格納場所
cascade.detectMultiScale(color_image, faces, 1.3, 6, 0, Size(30, 30));
for (int i = 0; i < faces.size(); i++) {
        rectangle(color_image, Point(faces[i].x, faces[i].y), Point(faces[i].x + faces[i].width, faces[i].y + faces[i].height), Scalar(0, 0, 255), 3, CV_AA); //検出した顔を赤色矩形で囲む
}
```

上記のコードを使用することで、検出をされた顔が矩形で囲まれる。

4.3.3 距離取得のプログラムの作成

4.3.2 項で作成したプログラムで顔検出が可能になった。次に、検出した顔までの距離測定プログラムの作成に移る。4.2.2 項で記したように get_distance 関数を使用して距離情報を取得する。実際に使用するプログラムを下記に示す。

```
cascade.detectMultiScale(color_image, faces, 1.3, 6, 0, Size(30, 30));
for (int i = 0; i < faces.size(); i++) //検出した顔の個数"faces.size()"分ループを行う
{
        rectangle(color_image, Point(faces[i].x, faces[i].y), Point(faces[i].x + faces[i].width, faces[i].y + faces[i].height), Scalar(0, 0, 255), 3, CV_AA);

        X = (w2 - w + 2 * faces[i].x + faces[i].width) / 2;
        Y = (h2 - h + 2 * faces[i].y + faces[i].height) / 2;
        dist_person = dframe.get_distance(X, Y);

        std::cout << "dist_face :" << dist << "¥n";
}
```

RGB 画像と Depth 画像では画像のサイズが異なる為、顔検出で取得した座標を一度 Depth 画像の座標に変換してから get_distance 関数に与える。また、4.3 節で説明する肌色検出でも同じように座標の変換を行う。

4.3.4 顔検出のパラメータの変更

顔検出は時間のかかる処理であるため、デフォルトのパラメータで実行すると、1 秒間に 3 回程度しか実行できない。そのため、そのまま利用すると滑らかに追従する撮影補助装置はできないだろう。そのため、プログラムで用いている OpenCV の detectMultiScale 関数のパラメータを変更して、処理速度の高速化を図る。実際の撮影補助装置で使用する Surface で顔検出プログラムを実行し、装置の使用にふさわしい値を探す。detectMultiScale 関数には、以下のパラメータが存在する。

```
cascade.detectMultiScale(Mat image, MatOfRect objects, double scaleFactor,
int minNeighbors, int flag, Size minSize, Size maxSize)
```

1. scaleFactor: 縮小量で大きさの変更比率の大きさ。1 より大きいほど検出は速いが、精度が落ちる。
2. minNeighbors: 検出されやすさ。小さいほど検出されやすいが、誤検出が増える。
3. minSize: 検出対象の最小サイズ。値を大きくすることで、小さい物体を無視できるため、処理が速くなる。

以上のパラメータを変更したときの検出可能距離と、映像の速度を計測する。検出可能距離は、0.5m ごとに計測した。

表 4-1 顔検出距離の結果

scaleFactor	minNeighbors	minSize	検出可能距離[m]	映像の FPS
1.2	3	20,20	5	3.4
1.2	3	30,30	3.5	4
1.3	3	30,30	3	4.8
1.3	6	40,40	2.5	4.9
1.3	3	40,40	3	5
1.2	4	30,30	3	3.9
1.3	6	30,30	3	5

上記の 7 通りの値で実験を行った。本研究では、検出可能距離と映像のなめらかさが両立する一番下の値を採用して顔検出を行う。

4.4 肌色検出

4.4.1 概要

本節では、肌色検出について解説する。肌色検出とは、RGB 画像内から人の肌を抜き出して 2 値化する検出方法である。肌を抜き出した後にそれぞれ領域の重心を求め、その点までの距離を取得する。処理を終えた画像を図 4-6 に示す。

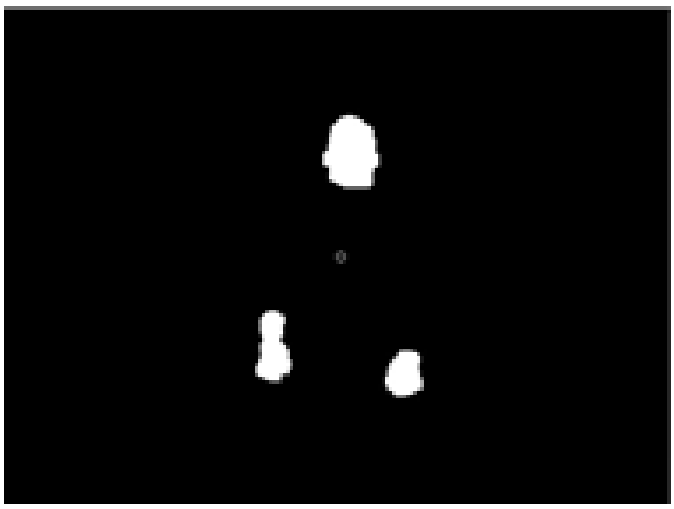

図 4-6 肌色と認識された領域を白く 2 値化した画像

ただし、図 4-7 のように人の肌以外を白く検出してしまう場合もある。

図 4-7 肌以外の領域も白く 2 値化されている画像

4.4.2 検出方法

肌色検出を実現するための手順は、以下の通りである。

1. RGB 画像を HSV に変換する。
2. 指定した条件を満たす色を白で 2 値化する。

4.4.3 HSV に変換するプログラムの作成

肌色検出は RGB でなく HSV 色空間を利用して検出を行なう。HSV 色空間とは、色を「色相(Hue)」「彩度(Saturation)」「明度(Value・Brightness)」の 3 要素で表現する方式である。RGB は原色の組み合わせで色を表現するが、この方法では各要素を変更したときに色がどう変化するかがわかりにくく、細かい調整には不向きである。そのため、今回の検出には HSV 色空間を用いる。

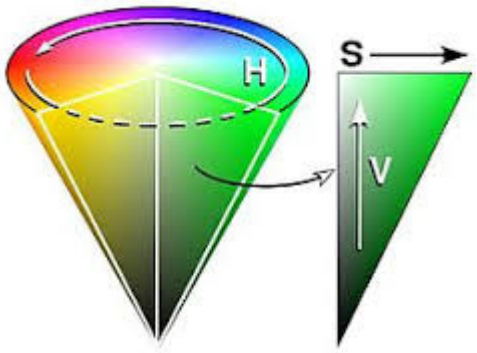

図 4-8 HSV 空間を視覚化した円錐[4-1]

また、HSV 色空間への変換は、下記のプログラムで実行できる。

```
cvtColor(color_image, hsv_image, CV_BGR2HSV, 3);// HSV 色空間への変換
```

4.4.4 肌を抜き出し 2 値化するプログラムの作成

HSV 色空間に変更後、閾値を定めて 2 値化を行なう。2 値化とは、濃淡のある画像を白と黒の 2 階調に変換する処理である。閾値を定めて、各画素の値を閾値と比較して白黒に置き換える。本研究では、肌色を白、それ以外を黒に変更するような閾値をもうける。

肌色を抜き出す閾値として今回は、下記の値を用いる。

```
#define H_MAX 40
#define H_MIN 0
#define S_MAX 150
#define S_MIN 30
#define V_MAX 255
#define V_MIN 0
```

また、肌を抜き出し 2 値化するプログラムは下記のようになる。

```
Scalar s_min = Scalar(H_MIN, S_MIN, V_MIN);
Scalar s_max = Scalar(H_MAX, S_MAX, V_MAX);
inRange(hsv_image, s_min, s_max, msk_image);
```

4.5 形検出

4.5.1 概要

本節では人検出方法の一つである形検出について解説する。

形検出では、Depth 画像から人型の物体を検出する。人型の物体を検出するにあたり、元の Depth 画像(図 4-1)にいくつかの画像処理を加える。最終的に形検出された領域を緑の矩形で囲む（図 4-9）。

図 4-9 形検出の完成画像

4.5.2 検出方法

形検出を実現するための手順は、以下の通りである。

1. Depth 画像を 2 値化する
2. 抜き出した箇所をラベリングする
3. 面積値と縦横の比を利用して人型かを判断する

4.5.3 2 値化するプログラムの制作

人型を検出するにあたり、まず画像の 2 値化を行なう。(4.4.4 項参照)

本研究では、Depth 画像から赤色を白色に、それ以外を黒色に置き換える。赤色を抜き取る理由としては、RealSense から近いことを表している赤色以外は想定している利用距離の 3m を越えるため利用しないからである。下記のコードのように読み込んだ画像から指定の GBR の要素を白色に変換して 2 値化する。

```
Mat red_image;
inRange(depth_image, Scalar(0, 0, 108), Scalar(44, 255, 255),  red_image);
```

プログラムの実行後の画像を図 4-10 に示す。画像内から赤に近い色を白に 2 値化できていることがわかる。しかし、人以外の物も白く塗られていることがわかる。ここから人のみを抽出する必要がある。

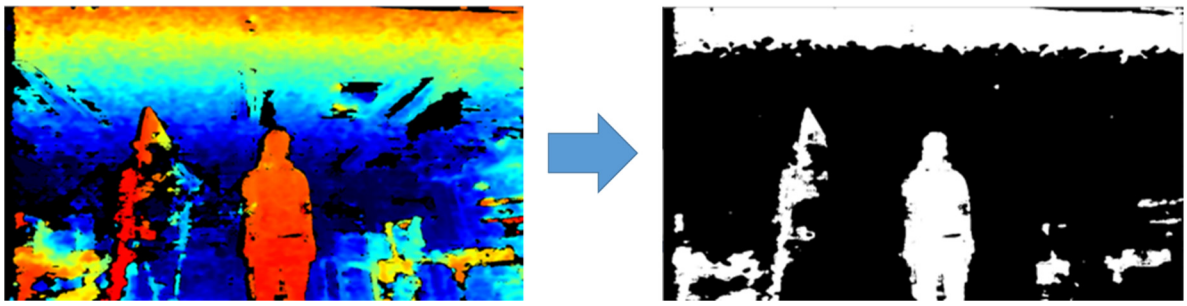

図 4-10 Depth 画像（左）から赤色を抜き取って 2 値化した画像（右）

4.5.4 ラベリングするプログラムの作成

次に 2 値化された図 4-10(右)画像にラベリング処理を行なう。ラベリング処理とは、2 値化処理された画像において、白の部分が連続した画素に同じ番号を割り振る処理である。通常、同じ番号ごとの面積や幅、高さなどの特徴量を求めて欠陥検査や分類処理に用いられる。このラベリング処理を行ないその後の処理で、面積や形を判断する。これによって、図 4-10(右)から人型の判断に利用する特徴を取得できる。また、領域ごとに処理が可能になり、この後の画像処理を簡単に行える。ラベリング処理は connectedComponentsWithStats 関数を用いて行なう。

```
int nLab = cv::connectedComponentsWithStats(red_image, LabelImg, stats, centroids);
```

上記のコードを実行し領域ごとに番号を振り分ける。その後、領域ごとに目視で確認できるよう色をランダムにつける。

```
std::vector<cv::Vec3b> colors(nLab);
colors[0] = cv::Vec3b(0, 0, 0);
for (int i = 1; i < nLab; ++i) {
        colors[i] = cv::Vec3b((rand() & 255), (rand() & 255), (rand() & 255));
}
```

connectedComponentsWithStats 関数で得た stas 変数を下記のように param 変数に代入することで、i 番目のラベルがついた領域のデータを取り出せる。

```
int *param = stats.ptr<int>(i);
```

また、物体ごとの縦横の比を計算しやすくするためにそれぞれを矩形で囲む。

```
int x = param[cv::ConnectedComponentsTypes::CC_STAT_LEFT];
int y = param[cv::ConnectedComponentsTypes::CC_STAT_TOP];
int height = param[cv::ConnectedComponentsTypes::CC_STAT_HEIGHT];
int width = param[cv::ConnectedComponentsTypes::CC_STAT_WIDTH];
int area = param[ConnectedComponentsTypes::CC_STAT_AREA];

rectangle(label_image, Rect(x, y, width, height), Scalar(0, 255, 0), 2);
```

このプログラムの実行後の画像を図 4-11 に示す。この処理によって、それぞれ領域ごとに処理を行えるようになる。

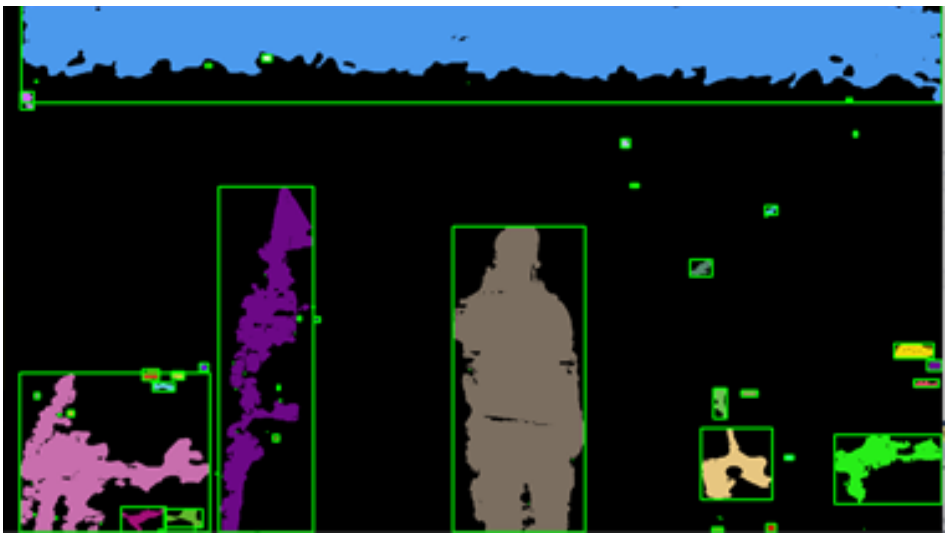

図 4-11 2 値化後にラベリング処理をした様子

4.5.5 人型以外を除去するプログラムの作成

4.5.4 項で得た面積、矩形の縦横の長さを用いて人型の選別を行なう。まず、面積を元に選別する。図 4-11 内のノイズのような小さな矩形を下記のコードで除外する。

```
int *param = stats.ptr<int>(i);
int area = param[ConnectedComponentsTypes::CC_STAT_AREA];

if(area > 10000){
        rectangle(label_image, Rect(x, y, width, height), Scalar(0, 255, 0), 2);
}
else if {
        rectangle(label_image, Rect(x, y, width, height), Scalar(0, 0, 0), -1);
}
```

このコードを使用したあとの画像を図 4-12 に示す。

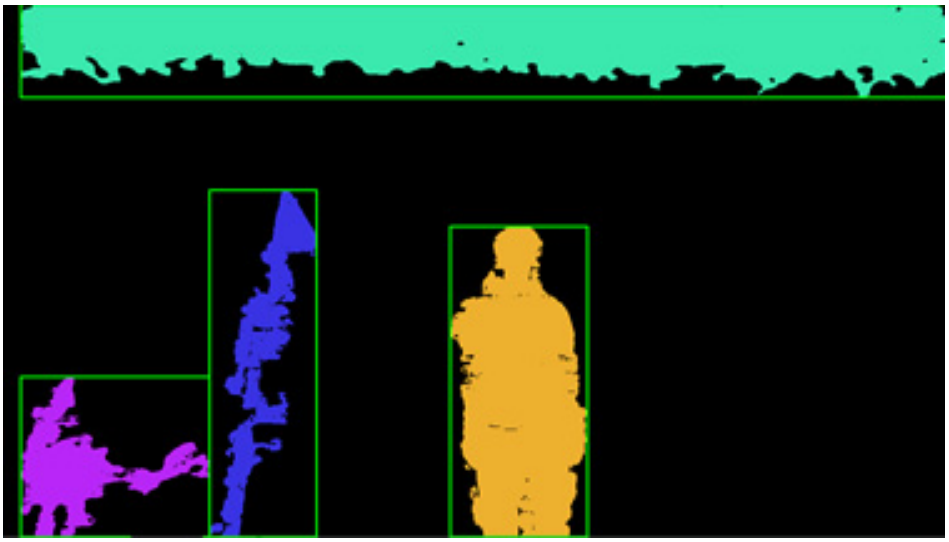

図 4-12 図 4-11 から面積の小さいものを取り除いた様子

次に矩形の形によって選別を行なう。人型の条件として、縦の長さが横の長さの 2 倍以上あるものとする。Depth 画像内に腰より上が写っていれば達成できる値として 2 倍を基準とした。上記のコードに条件を追加したものが下記のコードである。

```
int *param = stats.ptr<int>(i);
int area = param[ConnectedComponentsTypes::CC_STAT_AREA];//面積
int height = param[cv::ConnectedComponentsTypes::CC_STAT_HEIGHT];//矩形の高さ
int width = param[cv::ConnectedComponentsTypes::CC_STAT_WIDTH];//矩形の横幅

if((area > 10000) && (height > 2 * width) ){
        rectangle(label_image, Rect(x, y, width, height), Scalar(0, 255, 0), 2);
}
else if{
        rectangle(label_image, Rect(x, y, width, height), Scalar(0, 0, 0), -1);
}
```

プログラムを実行した結果を図 4-13 に示す。

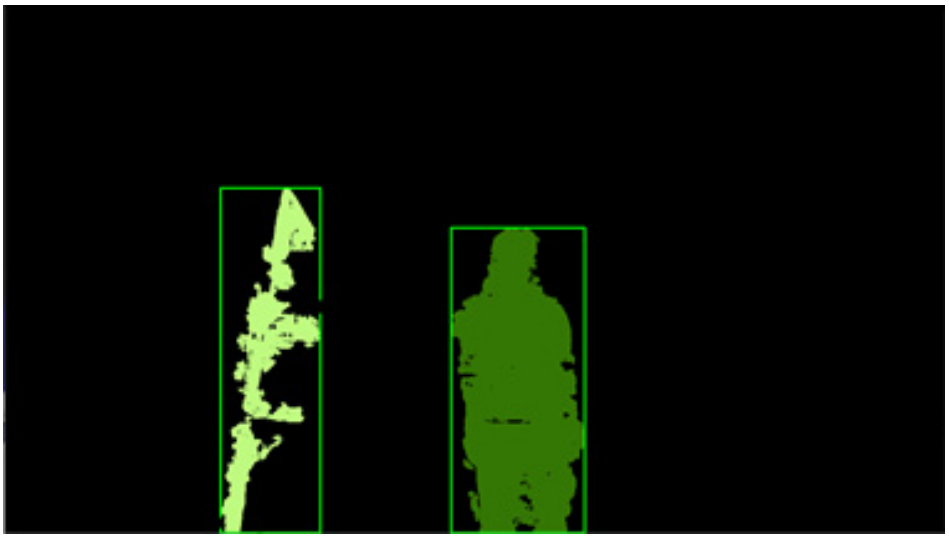

図 4-13 図 4-12 から縦横比が 2 倍ないものを取り除いた様子

4.5.6 条件の厳格化

4.5.5 項で人型の条件を定めて、プログラムの作成を行なった。しかし、本来検出しないはずのホワイトボードのようなものでも、図 4-14 のように検出されてしまうことがある。そこで、4.5.5 項の条件に項目を増やして、人以外を検出することを防ぐ。

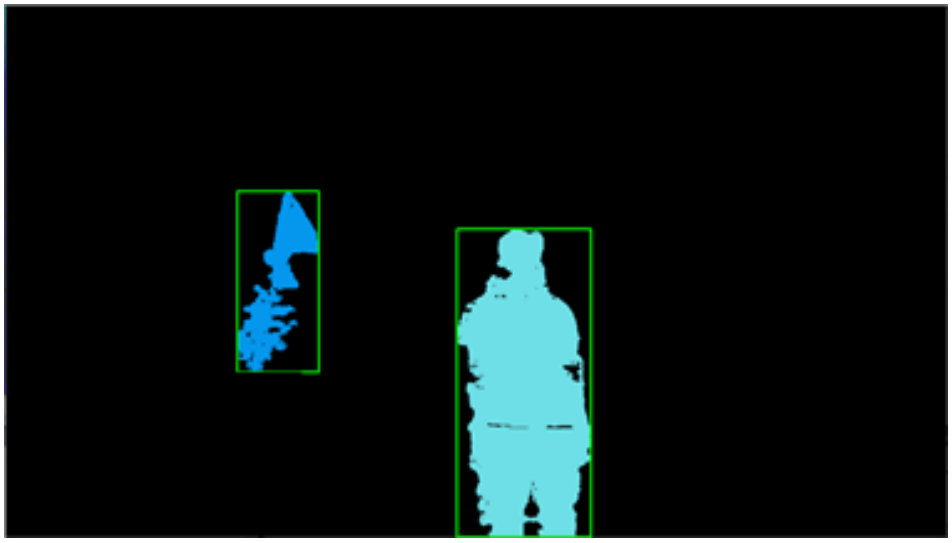

図 4-14 人型に加えてホワイトボードも認識されている画像

まず、画面端にあるものを検出しない条件を追加する。Depth 画像の横幅は 1280 である。その中の左右それぞれ 20 ずつの領域を検出しない領域に設定する。

次に、Depth 画像の下辺に接していないものも検出をしないように変更する。撮影補助装置での RealSense の設置位置(2 章参照)では、人が足下まで写ることはないので、人型は必ず画像の下辺と接することになる。図 4-14 のうち画像内の下辺に接していないものは、人以外であると判断する。Depth 画像の縦幅は 720 であり、下辺から 20 のうちに矩形の下辺がないものを検出の対象外とする。

上記の条件を追加したものが下記のコードになる。また、上記の数値を視覚的に確認できるようにした図を図 4-15 と図 4-16 に示す。

```
int *param = stats.ptr<int>(i);
int area = param[ConnectedComponentsTypes::CC_STAT_AREA];//面積
int height = param[cv::ConnectedComponentsTypes::CC_STAT_HEIGHT];//矩形の高さ
int width = param[cv::ConnectedComponentsTypes::CC_STAT_WIDTH];//矩形の横幅
int x = param[cv::ConnectedComponentsTypes::CC_STAT_LEFT];//矩形の左上の x 座標
int y = param[cv::ConnectedComponentsTypes::CC_STAT_TOP];//矩形の左上の y 座標

if((area > 10000) && (height > 2 * width) && (x >20) && (x + width < 1260) && (y + height > 700)){
        rectangle(label_image, Rect(x, y, width, height), Scalar(0, 255, 0), 2);
}
else if {
        rectangle(label_image, Rect(x, y, width, height), Scalar(0, 0, 0), -1);
}
```

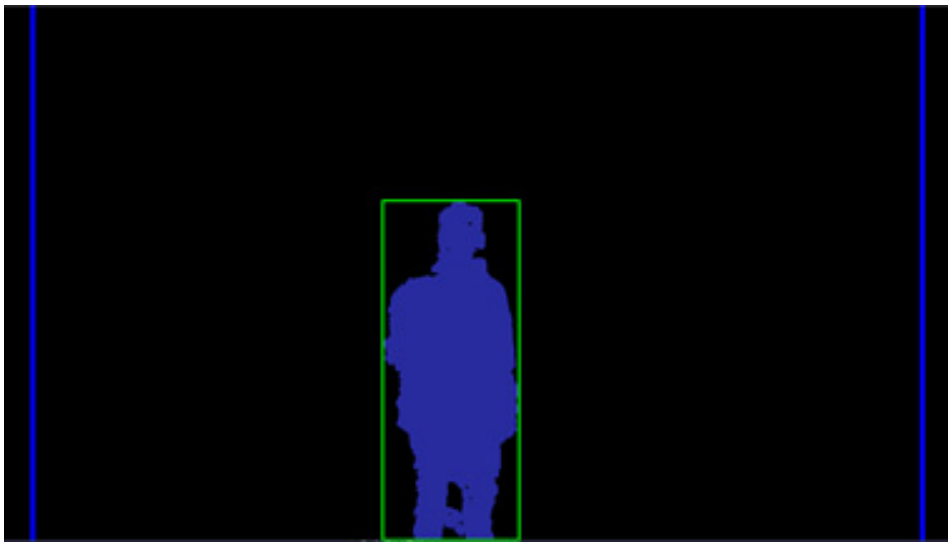

図 4-15 青い線より端にあるものは検出されない

図 4-16 赤い線よりも上に下辺があるものは検出されない

4.6 検出方法の組み合わせ

4.6.1 概要

4.5 節までで顔検出･肌色検出･形検出プログラムがそれぞれ完成した。顔検出では、パラメータの調節により顔以外を誤検出することはなくなった。しかし、肌色検出･形検出の二つを別々で運用した場合、人以外のものを検出してしまうことが多くあった。そこで、肌色検出･形検出を組み合わせることで信頼性の高い検出方法にする。

4.6.2 組み合わせるプログラムの作成

肌色検出と形検出が同一の物体を同時に検出したときに、人と判断することにする。同時に検出しているかの判断は、形検出で対象を囲っている矩形の中に、肌色検出で取得した重心座標があるかで判断する。判断の為のプログラムを下記に示す。

```
Rect Shikaku(Rect(x, y, width, height));//形認識の矩形
Point p(mc.x, mc.y);//肌色の座標

if (Shikaku.contains(p)) {
        std::cout << "肌色が含まれる" << "¥n";
}
else {
        std::cout << "不一致";
}
```

4.7 検出方法の評価

4.7.1 概要

顔検出･肌色検出･形検出それぞれの検出可能な距離と、精度の確認を行なう。精度によって撮影補助装置の制御する際の優先度を決定する。

4.7.2 実験方法

実験を行なうときは、各検出方法をそれぞれ一つだけを動作させる。また、測定可能距離の確認は 2m, 2.5m, 3m の 3 種類の距離で行なう。実験をする環境は、図 4-17 のように後ろに衝立を設置して背景に邪魔なものが映らないようにした場合と、図 4-18 のようにそのままの背景の場合との二通りで行なった。

図 4-17 衝立あり

図 4-18 衝立なし

実験の手順は以下の通りである。

1. RealSense の画面中央に人が位置するように設置する。
2. 10 秒間プログラムを動作させ、顔・肌色・形の検出を行なう。
3. 10 秒間の間に何秒間検出できていたかを確認する。
4. 上記の手順を各 5 回行い、検出できていた時間の平均を求め、結果を百分率で示す

4.7.3 実験結果

実験結果は表とグラフで示す。

表 4-2 顔検出の精度評価(衝立あり)

	2m	2.5m	3m
回数	[%]	[%]	[%]
1 回目	100	59.1	66.1
2 回目	100	84.8	2.9
3 回目	100	27.7	19.6
4 回目	100	82.9	51.4
5 回目	97.9	63	1.9
平均	99.58	63.5	28.38

表 4-3 顔検出の精度評価(衝立なし)

	2m	2.5m	3m
回数	[%]	[%]	[%]
1 回目	100	100	14.7
2 回目	100	87	37.8
3 回目	100	100	73.7
4 回目	100	92.3	42.4
5 回目	100	87.1	10.6
平均	100	93.28	35.84

表 4-4 肌色検出の精度評価(衝立あり)

	2m	2.5m	3m
回数	[%]	[%]	[%]
1 回目	99.6	100	100
2 回目	85.8	100	100
3 回目	97.7	100	88.9
4 回目	100	100	100
5 回目	97.6	97.6	100
平均	96.14	99.52	97.78

表 4-5 肌色検出の精度評価(衝立なし)

	2m	2.5m	3m
回数	[%]	[%]	[%]
1 回目	100	100	100
2 回目	100	100	100
3 回目	100	100	100
4 回目	100	100	100
5 回目	100	100	100
平均	100	100	100

表 4-6 形検出の精度評価(衝立あり)

	2m	2.5m	3m
回数	[%]	[%]	[%]
1 回目	100	100	検出不可
2 回目	100	100	検出不可
3 回目	100	100	検出不可
4 回目	100	100	検出不可
5 回目	100	100	検出不可
平均	100	100	検出不可

表 4-7 形検出の精度評価(衝立なし)

	2m	2.5m	3m
回数	[%]	[%]	[%]
1 回目	100	100	検出不可
2 回目	100	100	検出不可
3 回目	100	100	検出不可
4 回目	100	100	検出不可
5 回目	100	100	検出不可
平均	100	100	検出不可

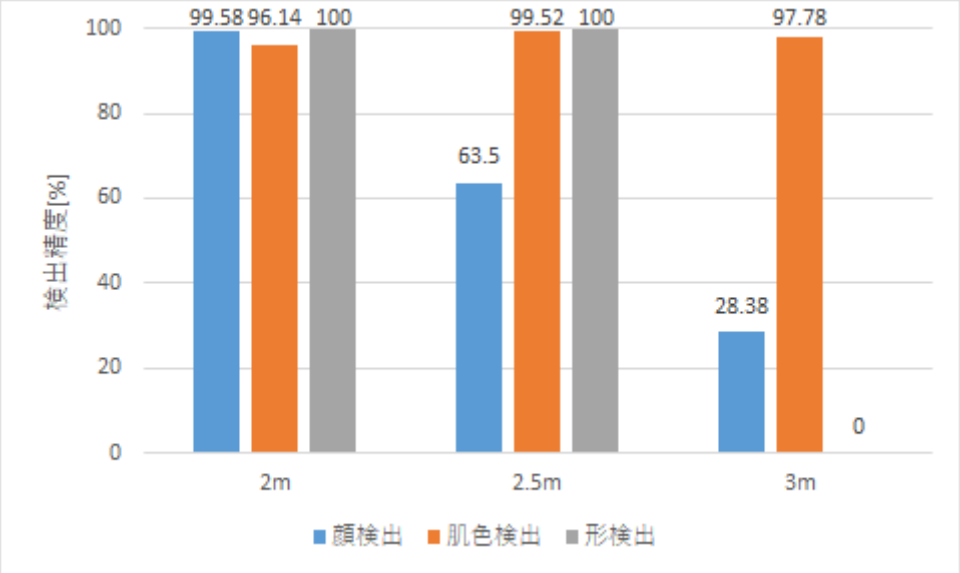

図 4-19 各検出方法の距離ごとの平均まとめ（衝立あり）

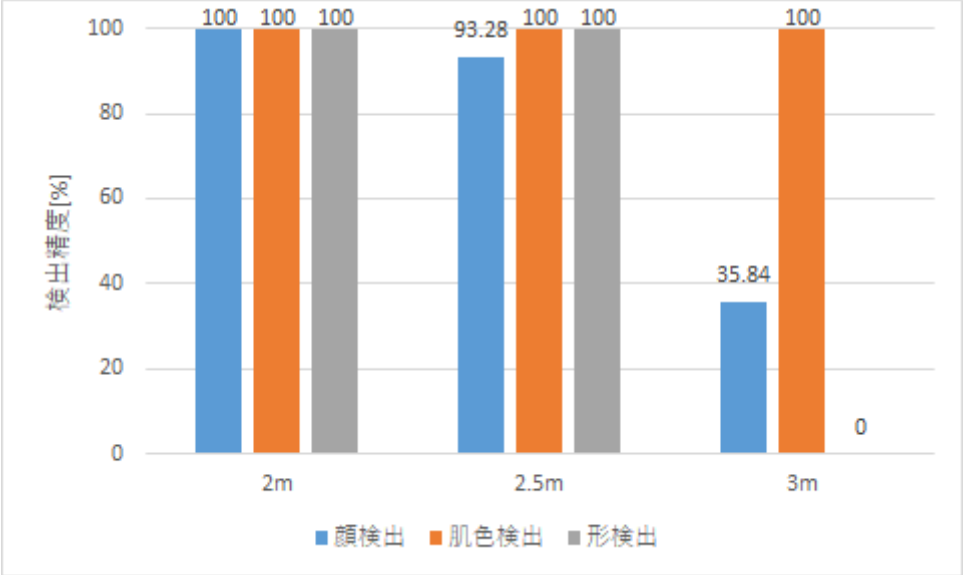

図 4-20 各検出方法の距離ごとの平均まとめ(衝立なし)

4.7.4 考察

実験結果より、人が 2m から 2.5m の位置に立った場合ではどの検出方法も高い精度で検出ができている。ただし、3m では形の検出ができなくなり、顔検出も精度が低下する。肌色検出は距離に関わらず精度が高いが、人の肌以外も抜きとることが多く単体での使用が難しい。

以上のことをまとめると、顔検出は単体での使用で十分に人検出として利用でき、肌色検出と形検出は、距離の制限や人以外への反応からそれぞれ単体での使用が難しいことがわかる。形検出は 2.5m までなら十分な精度で検出ができているので顔検出の補助として使用できる。

第 5 章 撮影補助装置の動作制御（藤井担当）

5.1 概要

5.1.1 目的

撮影補助装置が人を撮影用スマートフォンで映し続けるためには、人の動きに追従する必要がある。第 4 章にて RealSense を用いて人を検出することが可能になったので、検出された人に合わせて装置を動かすことで目標を達成できる。本章では人検出で取得できた距離情報と座標を用いて撮影者に追従するシステムの作成を行う。

5.1.2 要求性能

撮影補助装置に必要とされる動作は以下のとおりである。

1. 人を映像の中心に映し続けること
2. 映像内で人を一定の大きさで映し続けること
3. 任意のタイミングで映像の視点を固定すること

上記の性能を実現させるために必要な機能は以下の通りである。

1. 撮影者を撮影用スマートフォンの中央にとらえ続ける機能
2. 撮影者と装置の距離を一定に保つ機能
3. 任意のタイミングで装置を停止させる機能

5.1.3 プログラムのフロー

本章で作成するプログラムのフローを解説する。

起動したプログラムは、図 5-1 のように大きく分けて 4 つのフローを実行する。

まず、旋回コマンドを送信することでルンバはその場で回転を始める（①）。回転の最中に 4 章で解説した人検出を実行し（②）、人が検出されたときに回転を止める。人の位置に基づいて映像の中心に人が来るように角度を決めてルンバを回転させる(③)。その後、取得した距離に合わせて前後に移動し間隔を一定に保つ（④）。一度人を検出した後は、③と④を繰り返し撮影の補助を行う。また、人を見失ってしまった場合は、①に戻りフローを実行する。

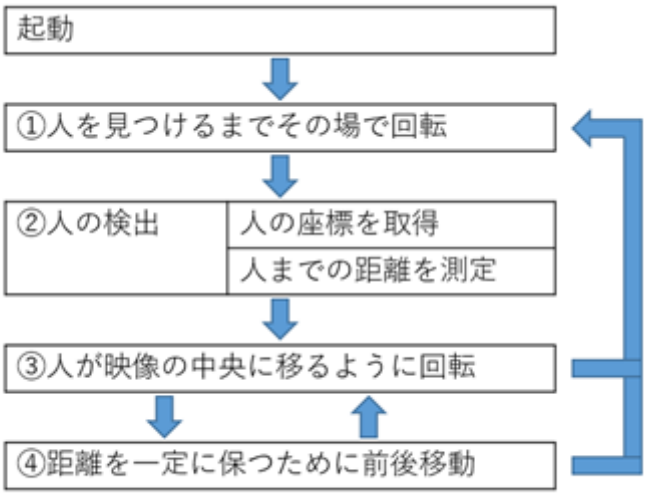

図 5-1 プログラムのフロー

5.2 撮影補助装置を制御するプログラムの製作

5.2.1 概要

プログラムのフローを実行するためのプログラムを作成する。ルンバの制御は 3.3.1 項で説明した、角度や距離を設定してルンバを動かす角度・距離制御と、モーターの回転数を設定して動かす速度制御の二種類を用いる。

5.2.2 ルンバを回転させるプログラムの作成

本項では図 5-1 のプログラムのフローのうち、人を見つけるまでその場で回転するプログラムの作成を行なう。このときのルンバの制御には速度制御のコマンドを用いる。その理由は、角度制御で細かく回転を繰り返すと映像がぶれてしまうこと、停止のコマンドですぐに回転をやめることができることなどである。

Surface から Arduino にデータを送るには、以下のコードのように arduino->writeSerialPort 関数に送りたいデータと文字数を与えて実行すればよい。人が検出されていないときに、撮影補助装置が回転させるのが以下のコードである。モーターの回転速度を 11 と小さくしているのは、RealSense の映像のブレで人検出ができなくなることを防ぐためである。

```
if{
//人を見つけた場合の処理
}
else {
        arduino->writeSerialPort("a0011", 5);
        std::cout << "Not Found" << "¥n";
}
```

5.2.3 人を映像の中心に映すプログラムの作成

本項では図 5-1 のプログラムのフローのうち、人検出後に人を映像の中心にくるように撮影補助装置を回転させるプログラムを作成する。

このときのルンバの制御には角度制御を利用する。速度制御ではルンバが停止した後の人の小さなずれを人の移動と判定してしまい、次のフローに進むことが困難だからである。

はじめに、人検出後に装置が何度回転すべきかを RealSense の水平方向の視野角と映像内の人の位置を比較することで算出する。

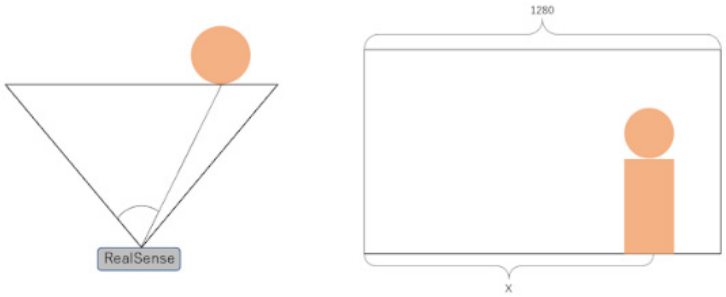

図 5-2 角度算出の例

図 5-2 のように水平角 91.2°、画面の横幅 1280、人の位置 X とすると、画面左端を基準とした角度 θ は次式で求められる。

$$\theta = 91.2 \times \frac{X}{1280}$$

画面の中央からの角度に変換すると、次式のようになる。

$$|\theta| = 91.2 \times \frac{X}{1280} - 45.6$$

本項で作成するプログラムを動作させる条件は、RealSense が人を検出し、映像の中心に人がいないことである。

以下に人を映像の中心に映すプログラムを示す。下記のプログラムは人が画面右側にいる場合のものである。左側に検出した場合は、ルンバには逆方向に回転するようにプログラムを作成する。

```
rad_float = abs(91.2 * X / 1280.0 - 45.6);
int rad = (int)rad_float;
```

```
std::string str_rad;
std::stringstream ss;
ss << rad << ' ';
ss >> str_rad;

if (rad >= 10) {
        Dec = "A00" + str_rad;
}
else {
        Dec = "A000" + str_rad;
}

Dec.erase(5, 1);
const char *y = Dec.c_str();
char *write_char = strdup(y);
arduino->writeSerialPort(write_char, 5);
std::cout << "trun left, rad :" << str_rad << "¥n";
```

5.2.4 人と装置の距離を保つプログラムの作成

本項では人と装置の距離を一定に保つプログラムの作成を行う。撮影時に人を一定の大きさに保ちながら撮るには、撮影用スマートフォンと人の距離を一定に保つ必要があるためである。

4.2 節で距離の測定が可能になり、測定した距離に応じてルンバに前後移動させる指示を送ることができるようになっている。撮影補助装置内の撮影用スマートフォンの高さ(2 章参照)から人をバストアップで撮影するには、2m の距離をあける必要がある。RealSense の距離測定の誤差を鑑みて、装置から人までの距離の設定は、2±0.1mを設定の範囲とする。

以下に人との距離を保つプログラムを示す。

```
int Centi = floor(100 * dist_person);

if (210 <= Centi) {
        arduino->writeSerialPort("w0011", 5);
        std::cout << "¥n";
        std::cout << "Move forward" << "¥n";
}
else if (Centi < 190) {
        arduino->writeSerialPort("x0011", 5);
        std::cout << "¥n";
        std::cout << "Move Behind" << "¥n";
}
else {
        arduino->writeSerialPort("s", 1);
        std::cout << "¥n";
        std::cout << "Best Spot ¥n";
}
```

5.2.5 一時停止するプログラムの作成

本項では撮影補助装置を一時停止させるプログラムの作成を行う。動画の撮影中に背景を見せたい場合や、人が中心から外れたときの使用を想定している。また、人とカメラの距離によっては、映像の中心に人がいるときでも人を見失ってしまうこともある。そのようなときにルンバを回転させずに動きを固定する場合も使用できる。

一時停止の条件は、直前 5 回の処理で人を検出していて、次の処理で人を検出できなかった時に実行される。故意に使用するには、撮影補助装置が動作していときに、RealSense の視野外に移動ことである。

一時停止するプログラムを下記に示す。

```
int a[6];
int i;

for (i = 6; i > 1; i--) {
        a[i] = a[i - 1];
}
a[i] = Z;//Z は処理ごとに人がいればプラスの値、いなければ-1

if ((abs(a[i + 1]) > 1) && (abs(a[i + 2]) > 1) && (abs(a[i + 3]) > 1) && (abs(a[i + 4]) > 1) && (abs(a[i + 5])) {
        arduino->writeSerialPort("s", 1);
        std::cout << "Exploring" << "¥n";

        a[i] = 666;
}
```

第 6 章 動作確認（眞野担当）

6.1 動作確認

6.1.1 概要

本章では撮影補助装置の動作確認を行うことで、要求機能（2.1.2 項を参照）が満たされていることをチェックする。撮影補助装置の評価は機体、人検出、距離測定、制御の要求機能をもとに行う。

6.1.2 動作確認のロケーション

動作確認は、工学院大学八王子キャンパス 4 号館 8 階の 4-853（知能機械研究室）内で行う。今回ソフトウェアで行った検出方法をすべて確かめるために誤認識しやすい棚などがある場所で行った(図 5-1)。また、本研究で完成した動画撮影補助装置を図 6-2 に示す。

図 6-1 動作確認を行った研究室

図 6-2 完成した撮影補助装置

6.2 確認手順

動作確認は以下の手順で行う。

1. 人検出が行われ、回転移動により人をカメラの中心にとらえられるか確認する。
2. 人との距離測定が行われているか確認する。
3. 測定した距離をもとに人との距離を保ち移動できているかを確認する。
4. 人が移動し、その後、それに対し、撮影補助装置が追従し測定距離が保たれているかを確認する。また、カメラの中心に人を収められているのかを確認する。
5. 人が衝立に一度隠れたとき、撮影補助ロボットが一時停止できるかを確認する。
6. 衝立から人が出てくる。そこで、再び対象人物を認識し移動できるかを確認する。
7. プログラムを終了し、撮影用カメラで撮影した動画がぶれたりして見にくい動画になっていないか、人がカメラ内に収まっている動画になっているかを確認する。

6.3 動作確認の結果

動作確認の結果を撮影補助装置に設置している撮影用スマートフォンから撮影したものと、装置と人の両方を映す俯瞰カメラから撮影したものをもとにまとめ、5.2.3 項の手順に沿って説明する。今回行った動作確認では、対象人物と撮影補助ロボットの距離を 2m と設定した。

1. RealSense から得られた RGB 画像と Depth 画像をもとに surface 上のプログラムが人を検出し、それに伴いルンバが回転移動を行い、図 6-3 と図 6-4 のように人を撮影補助装置の撮影用カメラ内に捉えたことを確認した。

図 6-3 撮影用カメラの映像①(人を検出した)

図 6-4 俯瞰カメラの映像(人を検出した)

2. 距離測定が行われ、正常に動作したことを確認した。

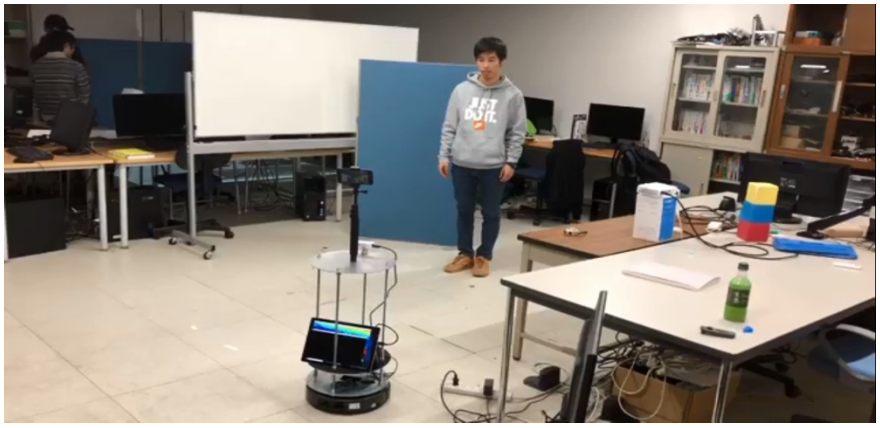

図 6-5 俯瞰カメラの映像②（距離計測が行われ、移動した）

3. 移動後も同様に距離計測が行われたが、何度か 2m より近すぎてしまうことや遠すぎてしまうことがあった。動作確認中に撮影補助装置から対象人物までの距離を測ったところ、表 6-1 のような結果になった。

表 6-1 撮影補助装置から対象人物までの距離

	1 回目	2 回目	3 回目	4 回目	5 回目
結果(cm)	200	200	210	194	225

4.　図 6-6、図 6-7 のように撮影補助装置が一時停止したことを確認した。

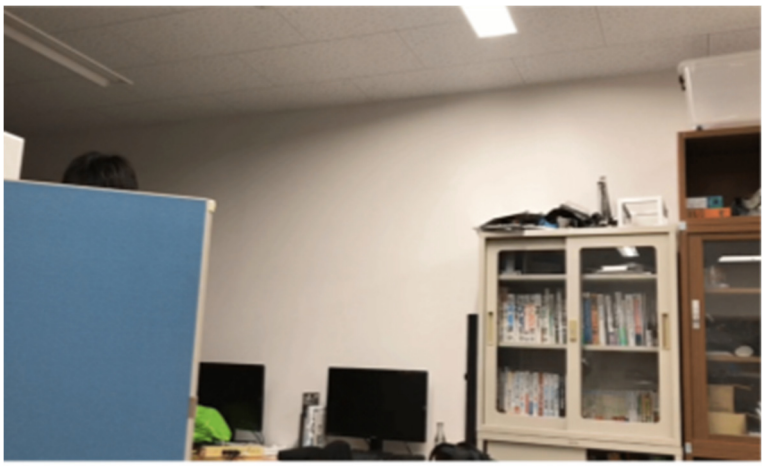

図6-6 撮影用カメラの映像②(衝立に隠れる)

図6-7 俯瞰カメラの映像②(衝立に隠れる)

5. 図 6-8、図 6-9 のように衝立から出てきた後も人物を検出し、約 2m の位置に移動した。

図6-8 撮影用カメラの映像③(衝立から出てくる)

図6-9 俯瞰カメラの映像③(衝立から出てくる)

6. 撮影用スマートフォンの動画にも、大きなブレがなく終了していたことを確認した。およそ 30 秒間の動画の中で、最初に対象となる人を見つけるために回転した時間と遮蔽版に隠れた時間を除いたおよそ 25 秒間は対象となる人が画面内に収まっていた。また、この 25 秒間のうちおよそ 18 秒間は画面の中心に対象となる人を映すことが出来た。

図 6-10 撮影用カメラの映像の最終確認（人が画面の中心に位置している）

6.4 考察

6.3 節の結果から、撮影補助装置が本研究における要求性能を満たしていることを確認した。

しかし、撮影補助装置と人の距離が設定している 2±0.1m から外れている場合が何度か確認された。撮影用スマートフォンは、撮影補助装置と人との距離が 2m になることを想定してセットしている。設定した距離から外れてしまうと、人の頭が画面から切れてしまう、背景を映せないなど想定していない映像が撮れてしまう。撮影補助装置と人の距離が設定から外れた理由として、RealSense の距離測定が移動しながら用いたときに精度が落ちることが原因ではないかと考えられる。

表 6-1 より設定距離から最も大きく外れた場合でも、差が+25cm と動画には大きな影響がない距離になっている(図 6-11、図 6-12)。この程度の誤差であれば問題なく動画撮影をできると考えられる。

図 6-11 撮影補助装置から 2m 人が離れたときの様子

図 6-12 撮影補助装置から 2.25m 人が離れたときの様子

第 7 章 結論（藤井担当）

7.1 目標

我々が製作した撮影補助装置は、デプスカメラと画像処理の技術を用いて人を追従し、一人でも動きのある動画を撮れるようにするものである。

7.2 目標への達成度

私たちは、目標を達成するべく研究を行った。表 7-1 において、〇が達成、△が一部達成、×が未達成を表している。

表 7-1 目標を達成するための条件

撮影補助装置の外観の考案と製作	〇
顔検出を用いた人検出	〇
形検出を用いた人検出	〇
肌色検出を用いた人検出	〇
RealSense を用いた距離測定	〇
ルンバの角度制御、速度制御の実現	〇
測定した人との距離に応じたルンバの前後移動	△
PC、Arduino、およびルンバの接続とシリアル通信	〇
Arduino を用いたルンバ内のモーターを操作、およびルンバの移動の実現	〇

ハードウェア開発では、必要機器を搭載、RealSense の人検出、撮影用スマートフォンでの撮影が可能な機体を製作することができた。

ソフトウェア開発では、人検出を 3 種類の検出方法を組み合わせ人の検出と距離測定を同時に行うことが可能になった。また、ルンバ制御では角度制御と速度制御を用いて人の動きに合わせて追従が可能となった。

7.3 問題

前節で述べたように各班が目標を達成したが、撮影補助装置には以下の問題点もあった。本節では、これらの問題点とその解決法について記述する。

7.3.1 問題点

撮影補助装置の問題点は以下の 3 点である。

問題 1. 人と距離を保つための前後移動が終わらないことがある

これは、RealSense の距離測定の精度が求めている精度に達していないからである。

問題 2. 人検出可能な距離が短い

4.7.3 項より仕様の 3m では十分な精度が得ることができなかった。これは、第 4 章で作成した検出方法が対応できる距離が短かったからである。

問題 3. 撮影用スマートフォンの映像がぶれる

これは、撮影用スマートフォンの固定方法が不十分だからである。また、装置全体の重心が高く機体が揺れてしまうことも原因として考えられる。

7.3.2 解決法

前項の問題点に対して考えられる解決法はそれぞれ以下の通りである。

問題 1 の解決法

本研究では人と装置の距離を 1.9m から 2m までと設定しているが、設定範囲を広げることで距離の差は広がるが前後移動はより早く終わると考えられる。

問題 2 の解決法

顔検出であればパラメータの変更、形検出であれば閾値の範囲の変更で検出距離を伸ばすことができる。

問題 3 の解決法

本研究では撮影用スマートフォンの固定法に一脚を使用しているが、三脚に変更することや、撮影補助装置の重心を下げることでブレを抑えられると考えられる。

7.4 総合評価

我々が製作した撮影補助装置も動作確認を行ったところ、撮影補助装置は人の動きに合わせて移動し人を撮影用スマートフォンに映し続けることができた。ただし、人との距離が離れすぎた場合は人を検出できないこともあった。また、人検出はできても距離測定が安定せず、撮影補助装置が前後移動を繰り返す場合もあった。これは、顔検出のパラメータの変更、形検出で使用する閾値の変更、人と装置の距離設定の見直しで上記の様な問題を解決できると考えられる。よって、今回の研究では、誤作動を起こす場合もあるが、人の動画撮影を十分に補助することが可能だったことより、目標を達成できたと考えられる。

第 8 章 参考文献・URL

[1-1]パソコン、スマートフォンからのインターネットの利用者数の推移

https://www.netratings.co.jp/news_release/2019/05/Newsrelease20190521.html

[1-2]スマートフォン動画アプリの利用状況

https://www.netratings.co.jp/news_release/2018/01/Newsrelease20180130.html

[1-3]2017 年国内 YouTuber 市場調査

https://www.cyberagent.co.jp/news/detail/id=21279

[1-4]Intel RealSense™ デプスカメラ D400 シリーズ

https://www.mouser.jp/new/Intel/intel-realsense-camera-400/

[1-5]ステレオカメラによる距離計測を用いた移動ロボットの製作

https://brain.cc.kogakuin.ac.jp/research/201903_RoombaThesis.pdf

[1-6]頼りになる自動追従運搬ロボット【サウザー】

https://jp.doog-inc.com/product-thouzer.html

[1-7]SOLOSHOT3 自動追跡型ロボットカメラマン

https://www.soloshot-japan.jp/

iRobot_Roomba_600_Open_Interface_Spec

https://cdn-shop.adafruit.com/datasheets/create_2_Open_Interface_Spec.pdf

Arduino MEGA 2560

https://ht-deko.com/arduino/mega2560.html

ルンバにスピッツを演奏させてみた

http://curious4dev.mydns.jp/post-1463/

[4-1]HSV 色空間

https://ja.wikipedia.org/wiki/HSV%E8%89%B2%E7%A9%BA%E9%96%93

2 値化 - IGUNOSS,lnc.

http://www.igunoss.co.jp/imageproc/imageproc1-5.html

ラベリング処理アルゴリズム　画像処理ソリューション

https://imagingsolution.blog.fc2.com/blog-entry-193.html

HSV 空間とは

https://www.peko-step.com/html/hsv.html

謝辞

最後に、本研究を進めるにあたり、2 年間多大なご指導をいただきました金丸隆志教授、撮影補助装置のハードウェアの設計と製作で親身に対応していただいたものづくり支援センターの方々、切磋琢磨し互いに高めあった研究室の同輩に深く感謝いたします。駄文ではありますが、これをもって謝辞とさせていただきます。

付録

Arduino のプログラム

```
#include <math.h>
#define RoombaSerial Serial1

static float g_odometry_x = 0;   //オドメトリの X 座標
static float g_odometry_y = 0;   //オドメトリの Y 座標
static float g_odometry_Angl = 0;//オドメトリの角度

//オドメトリの更新時間
static const int UPDATE_TIME_MSEC = 50;

//********************定数を定義********************
static const float ENCODER_MM_PAR_PULSE = 0.448;
//static const int   L = 250;//車輪間距離(中央-中央):235 (外-外):250
static const float   L = 238.0952;//車輪間距離(中央-中央):235 (外-外):250  770 は 238.0952 らしい

static const int RIGHT = 0;
static const int LEFT = 1;
//バッテリなどで使用
static const int CURRENT = 0;
static const int MAX = 1;

int fixedSpeed = 50;

//前回のエンコーダー値
static int g_beforEncoderR;
static int g_beforEncoderL;
int odometryInit()
{
  g_beforEncoderR = encoderOder(0);//右エンコーダー初期化
  g_beforEncoderL = encoderOder(1);//左エンコーダー初期化
}
```

```
void setup()
{
  RoombaSerial.begin(115200);
  Serial.begin(115200);

  // Start
  Serial.println("Start");
  RoombaSerial.write((byte)128);   // Start -&gt; Passive Mode
  delay(50);
  RoombaSerial.write((byte)130);   // Safe Mode
  delay(50);
  // Passive Mode
  //RoombaSerial.write((byte)128);   // Start -&gt; Passive Mode

  odometryInit();
}

//戻り値はエンコーダーの値
//引数：0は右車輪 1は左車輪
int encoderOder(int encoderNum)
{
  int bufSize;//バッファに貯まっているサイズ
  int temp;//シリアル通信できたもの
  unsigned int count;//エンコーダのカウンタ
  bufSize = RoombaSerial.available();

  RoombaSerial.write((byte)142);
  if ( encoderNum == RIGHT ) RoombaSerial.write((byte)43);  //右モーターの番号
  else                       RoombaSerial.write((byte)44);   //左モーターの番号

  while ( RoombaSerial.available() < 2 ) //2byte来るまで無限ループ
  { }

  if ( RoombaSerial.available() >= 2 ) //
  {
```

```
    //unsigned char temp;
    temp = RoombaSerial.read();
    count = temp * 256;
    //Serial.print( temp );
    //Serial.print( " , " );
    temp = RoombaSerial.read();
    count += temp;
    //Serial.print( temp );
    //Serial.print( " , " );
    //Serial.print( count );
    //Serial.print( "¥n" );
  }
  return count;

}



//戻り値は更新時間
unsigned long odometry()
{
  //変化量
  float dX , //⊿X
        dY ,//⊿Y
        dAngle ,//⊿Y
        dL ,//左車輪の移動量
        dR;//右車輪の移動量

  int tempR, tempL , counterR , counterL;
  tempR = encoderOder(0);//右エンコーダー
  tempL = encoderOder(1);//左エンコーダー
  unsigned long updateTime = millis();

  //前回の計測からカウンタが一瞬で半分以上(32768)変化したらオーバーフローしている
  if ( tempR - g_beforEncoderR < -32768 ) counterR = ((long)tempR + (long)65536 -
(long)g_beforEncoderR); //オーバーフローした場合
```

```
  else if ( tempR - g_beforEncoderR > 32768 ) counterR = ((long)tempR - (long)65536 -
(long)g_beforEncoderR); //マイナスになった場合
  else  counterR = tempR - g_beforEncoderR;

  if (          tempL - g_beforEncoderL < -32768 ) counterL = ((long)tempL + (long)65536 -
(long)g_beforEncoderL); //オーバーフローした場合
  else if ( tempL - g_beforEncoderL > 32768 ) counterL = ((long)tempL - (long)65536 -
(long)g_beforEncoderL); //マイナスになった場合
  else  counterL = tempL - g_beforEncoderL;

  //  Serial.print(tempL - g_beforEncoderL);
  //  Serial.print(" , ");
  //  Serial.print(tempR - g_beforEncoderR);
  //  Serial.print("¥n");
  g_beforEncoderR = tempR;//右エンコーダーの情報を更新
  g_beforEncoderL = tempL;//左エンコーダーの情報を更新

  dR = counterR * ENCODER_MM_PAR_PULSE;
  dL = counterL * ENCODER_MM_PAR_PULSE;

  dAngle = (double)(dL - dR) / L;
  g_odometry_Angl += dAngle / M_PI * 180.0;



  dX = ( dL + dR ) / 2.0 * cos( (g_odometry_Angl / 180) * M_PI ); //⊿X
  dY = ( dL + dR ) / 2.0 * sin( (g_odometry_Angl / 180) * M_PI ); //⊿Y

  g_odometry_x += dX;
  g_odometry_y += dY;
  return updateTime;
}



void loop()
{
  int command;
```

```
static boolean odometryMode = false;
if ( Serial.available() > 0 ) //データが来ていたら
{
  static int speedNum = 100;//速度
  int radiusNum = 0;
  command = Serial.read();
  //速度指定
  if ( '0' <= command && command <= '9' )
  {
    speedNum = command - '0';
    //スピード番号が 1～9 は 10%～90%に相当する
    //0 は 100%に相当する
    if ( speedNum == 0 )
    {
      speedNum = 10;
    }

    speedNum = 500 * speedNum / 10;
  }
  else
  {
    switch ( command )
    {
      case 's': stopRoomba();           break;

      // Set the fixed speed
      case 'S':
        speedNum = serialReadSpeed();
        fixedSpeed =  speedNum;     break;

      // Forward with speed
      case 'w':
        speedNum = serialReadSpeed();
        foward( speedNum );     break;

      // Forward with distance
```

```
case 'W':
  speedNum = serialReadSpeed();
  fowardDistance( speedNum );     break;

// Backward with speed
case 'x':
  speedNum = serialReadSpeed();
  backward(speedNum);     break;

// Backward with distance
case 'X':
  speedNum = serialReadSpeed();
  backwardDistance(speedNum);     break;

// Rotate right with speed
case 'd':
  speedNum = serialReadSpeed();
  turnRight( speedNum );  break;

// Rotate right with angle
case 'D':
  speedNum = serialReadSpeed();
  turnRightAngle( speedNum );  break;

// / Rotate left with speed
case 'a':
  speedNum = serialReadSpeed();
  turnLeft( speedNum );   break;

// Rotate left with angle
case 'A':
  speedNum = serialReadSpeed();
  turnLeftAngle( speedNum );   break;

//左コーナリング
case 'q':
```

```
  speedNum = serialReadSpeed();//速度を受信する
  radiusNum = serialReadRadius();//旋回半径を設定
  corneringLeft( speedNum , 100 ); break;
//右コーナリング
case 'e':
  speedNum = serialReadSpeed();//速度を受信する
  radiusNum = serialReadRadius();//旋回半径を設定
  corneringRight( speedNum , 100 ); break;


case 'b': motorsPWMControl( true );  break;
case 'B': motorsPWMControl( false ); break;


case 'c': Serial.print( encoderOder(RIGHT) );
  Serial.print( "¥n" );
  break;
case 'C': Serial.print( encoderOder(LEFT) );
  Serial.print( "¥n" );
  break;


//バッテリ容量
case 'p': Serial.print( getBattery( CURRENT ) );
  Serial.print( "¥n" );
  break;
case 'P': Serial.print( getBattery( MAX ) );
  Serial.print( "¥n" );
  break;

case 'o'://小文字ｏならオドメトリを送り始める
  odometryMode = true;   break;
case '0': //大文字０ならオドメトリを送るのをやめる
  odometryMode = false;  break;
case 'r'://座標リセットコマンド
  g_odometry_x = 0;   //オドメトリのＸ座標
```

```
          g_odometry_y = 0;   //オドメトリのY座標
          g_odometry_Angl = 0;//オドメトリの角度
          break;
      }
    }
  }

  static unsigned long odometryUpdateTime = millis();//時間初期化
  //前回の時間から30msec超えていたら
  if ( ( millis() >= (odometryUpdateTime + 30)) )
  {
    odometryUpdateTime = odometry();
    ///odometryUpdateTime //= millis();

    //もしオドメトリ表示モードだったら
    if ( odometryMode )
    {
      Serial.print( odometryUpdateTime );       Serial.print(",");
      Serial.print( g_odometry_x );   Serial.print(",");
      Serial.print( g_odometry_y );   Serial.print(",");
      Serial.print( g_odometry_Angl ); Serial.print("¥n");
    }

  }
}
//********************以下通信関係**********************
//速度設定
int serialReadSpeed()
{
  int speedNum = 0 //戻り値となる
                , readByte = 0;
  boolean signFlag;//符号がどっち  trueならプラス、falseならマイナス
  int temp;

  while ( true )
  {
```

```
  if (Serial.available() > 0)
  {
    temp = Serial.read();
    //数字かどうかチェック
    //数字だったら処理する
    if (  ('0' <= temp) && (temp <= '9') )
    {
      switch ( readByte )
      {
        case 0://Serial.print( temp-'0' ); Serial.print("¥n");
          if ( (temp - '0') == 0 ) signFlag = true;
          else signFlag = false;
          break;
        case 1://Serial.print( temp-'0' ); Serial.print("¥n");
          speedNum += (temp - '0') * 100;
          break;
        case 2://Serial.print( temp-'0' ); Serial.print("¥n");
          speedNum += (temp - '0') * 10;
          break;
        case 3://Serial.print( temp-'0' ); Serial.print("¥n");
          speedNum += (temp - '0') ;
          break;
      }
      readByte++;
    }
  }
  //4 文字受信したならばループ終了
  if ( readByte >= 4 ) break;
}
//負数だったならば
if ( !signFlag ) speedNum *= (-1 );
//速度上限と下限の設定
if ( speedNum > 500 ) speedNum = 500;
else if ( speedNum < (-500) ) speedNum = -500;

return speedNum;
```

```
}
//旋回半径設定
int serialReadRadius()
{
  int radiusNum = 0 //戻り値となる
                , readByte = 0;
  boolean signFlag;//符号がどっち  true ならプラス、false ならマイナス
  int temp;

  while ( true )
  {
    if (Serial.available() > 0)
    {
      temp = Serial.read();
      //数字かどうかチェック
      //数字だったら処理する
      if (  ('0' <= temp) && (temp <= '9') )
      {
        switch ( readByte )
        {
          case 0://Serial.print( temp-'0' ); Serial.print("¥n");
            if ( (temp - '0') == 0 ) signFlag = true;
            else signFlag = false;
            break;
          case 1://Serial.print( temp-'0' ); Serial.print("¥n");
            radiusNum += (temp - '0') * 1000;
            break;
          case 2://Serial.print( temp-'0' ); Serial.print("¥n");
            radiusNum += (temp - '0') * 100;
            break;
          case 3://Serial.print( temp-'0' ); Serial.print("¥n");
            radiusNum += (temp - '0') * 10;
            break;
          case 4://Serial.print( temp-'0' ); Serial.print("¥n");
            radiusNum += (temp - '0') ;
            break;
```

```
        }
        readByte++;
      }
    }
    //4 文字受信したならばループ終了
    if ( readByte >= 5 ) break;
  }
  //負数だったならば
  if ( !signFlag ) radiusNum *= (-1 );
  //速度上限と下限の設定
  if ( radiusNum > 2000 ) radiusNum = 500;
  else if ( radiusNum < (-2000) ) radiusNum = -500;

  return radiusNum;
}
//********************以下ルンバコントロール**********************

void motorsPWMControl( boolean mode)
{

  RoombaSerial.write((byte)144);   // Drive

  if ( mode )
  {
    RoombaSerial.write((byte)127);  // Velocity: 0x00c8 = 200
    RoombaSerial.write((byte)127);
    RoombaSerial.write((byte)127);  // Radius: 0x8000 = Straight
  }
  else
  {
    RoombaSerial.write((byte)0);  // Velocity: 0x00c8 = 200
    RoombaSerial.write((byte)0);
    RoombaSerial.write((byte)0);  // Radius: 0x8000 = Straight
  }
}
```

```
//前進命令
void foward(int speedNum)
{
  // Forward
  //Serial.println("Forward");

  //試しにbyte型配列を使ってみる
  byte sendSpeedData[2];
  sendSpeedData[0] = ( (byte *) ( (void*)(&speedNum) ) )[0];
  sendSpeedData[1] = ( (byte *) ( (void*)(&speedNum) ) )[1];



  RoombaSerial.write((byte)137);   // Drive
  //速度指定
  //配列の後ろから送信する
  RoombaSerial.write(sendSpeedData[1]);   // Speed
  RoombaSerial.write(sendSpeedData[0]);   // Speed
  RoombaSerial.write((byte)0x00);  // Velocity: 0x00c8 = 200
  RoombaSerial.write((byte)0x00);
}



//後退命令
void backward(int speedNum)
{
  speedNum *= (-1); //バックなので速度をマイナスにする
  byte sendSpeedData[2];
  sendSpeedData[0] = ( (byte *) ( (void*)(&speedNum) ) )[0];
  sendSpeedData[1] = ( (byte *) ( (void*)(&speedNum) ) )[1];

  // Backward
  //Serial.println("Backward");
  RoombaSerial.write((byte)137);   // Drive
  //速度指定
  //配列の後ろから送信する
  RoombaSerial.write(sendSpeedData[1]);   // Speed
```

```
  RoombaSerial.write(sendSpeedData[0]);   // Speed


  RoombaSerial.write((byte)0x80);  // Radius: 0x8000 = Straight
  RoombaSerial.write((byte)0x00);
}


//左その場回転
void turnLeft(int speedNum)
{
  byte sendSpeedData[2];
  sendSpeedData[0] = ( (byte *) ( (void*)(&speedNum) ) )[0];
  sendSpeedData[1] = ( (byte *) ( (void*)(&speedNum) ) )[1];
  //Serial.println("Turn");
  RoombaSerial.write((byte)137);   // Drive
  //速度指定
  //配列の後ろから送信する
  RoombaSerial.write(sendSpeedData[1]);   // Speed
  RoombaSerial.write(sendSpeedData[0]);   // Speed
  RoombaSerial.write((byte)0x00);  // Radius: 0x0001 = Turn in place counter-clockwise
  RoombaSerial.write((byte)0x01);
}


//右その場回転
void turnRight(int speedNum)
{
  byte sendSpeedData[2];
  sendSpeedData[0] = ( (byte *) ( (void*)(&speedNum) ) )[0];
  sendSpeedData[1] = ( (byte *) ( (void*)(&speedNum) ) )[1];
  // Turn in place clockwise
  //Serial.println("Turn");
  RoombaSerial.write((byte)137);   // Drive
  //速度指定
  //配列の後ろから送信する
  RoombaSerial.write(sendSpeedData[1]);   // Speed
  RoombaSerial.write(sendSpeedData[0]);   // Speed
  RoombaSerial.write((byte)0xff);  // Radius: 0x0001 = Turn in place clockwise
```

```
  RoombaSerial.write((byte)0xff);
}



// Forward with distance
void fowardDistance(int stopNum)  // stopNum [cm]
{
  int speedNum = fixedSpeed; // fixed sped


  g_odometry_x = 0;
  g_odometry_y = 0;
  g_odometry_Angl = 0;


  byte sendSpeedData[2];
  sendSpeedData[0] = ( (byte *) ( (void*)(&speedNum) ) )[0];
  sendSpeedData[1] = ( (byte *) ( (void*)(&speedNum) ) )[1];


  RoombaSerial.write((byte)137);   // Drive
  RoombaSerial.write(sendSpeedData[1]);   // Speed
  RoombaSerial.write(sendSpeedData[0]);   // Speed
  RoombaSerial.write((byte)0x00);  // Velocity: 0x00c8 = 200
  RoombaSerial.write((byte)0x00);


  while (1) {
    //変化量
    float dX , //⊿X
          dY ,//⊿Y
          dAngle ,//⊿Y
          dL ,//左車輪の移動量
          dR;//右車輪の移動量


    int tempR, tempL , counterR , counterL;
    tempR = encoderOder(0);//右エンコーダー
    tempL = encoderOder(1);//左エンコーダー
    unsigned long updateTime = millis();
```

```
    //前回の計測からカウンタが一瞬で半分以上(32768)変化したらオーバーフローしている
    if ( tempR - g_beforEncoderR < -32768 ) counterR = ((long)tempR + (long)65536 - (long)g_beforEncoderR); //オーバーフローした場合
    else if ( tempR - g_beforEncoderR > 32768 ) counterR = ((long)tempR - (long)65536 - (long)g_beforEncoderR); //マイナスになった場合
    else counterR = tempR - g_beforEncoderR;

    if ( tempL - g_beforEncoderL < -32768 ) counterL = ((long)tempL + (long)65536 - (long)g_beforEncoderL); //オーバーフローした場合
    else if ( tempL - g_beforEncoderL > 32768 ) counterL = ((long)tempL - (long)65536 - (long)g_beforEncoderL); //マイナスになった場合
    else counterL = tempL - g_beforEncoderL;

    g_beforEncoderR = tempR;//右エンコーダーの情報を更新
    g_beforEncoderL = tempL;//左エンコーダーの情報を更新

    dR = counterR * ENCODER_MM_PAR_PULSE;
    dL = counterL * ENCODER_MM_PAR_PULSE;

    dAngle = (double)(dL - dR) / L;
    g_odometry_Angl += dAngle / M_PI * 180.0;

    dX = ( dL + dR ) / 2.0 * cos( (g_odometry_Angl / 180) * M_PI ); //⊿X
    dY = ( dL + dR ) / 2.0 * sin( (g_odometry_Angl / 180) * M_PI ); //⊿Y

    g_odometry_x += dX;
    g_odometry_y += dY;

    //Serial.println( g_odometry_x);

    if ( Serial.available()) {
      if ( Serial.read() == 's' ) {
        RoombaSerial.write((byte)137);   // Stop
        RoombaSerial.write((byte)0x00);
        RoombaSerial.write((byte)0x00);
        RoombaSerial.write((byte)0x00);
```

```
        RoombaSerial.write((byte)0x00);
        break;
      }
    }

    if ( g_odometry_x >= 10 * stopNum ) {

      RoombaSerial.write((byte)137);   // Stop
      RoombaSerial.write((byte)0x00);
      RoombaSerial.write((byte)0x00);
      RoombaSerial.write((byte)0x00);
      RoombaSerial.write((byte)0x00);
      delay(10);
      break;
    }
  }
}

// Backward with distance
void backwardDistance(int stopNum)  // stopNum [cm]
{
  int speedNum = -fixedSpeed; // fixed sped
  g_odometry_x = 0;
  g_odometry_y = 0;
  g_odometry_Angl = 0;

  byte sendSpeedData[2];
  sendSpeedData[0] = ( (byte *) ( (void*)(&speedNum) ) )[0];
  sendSpeedData[1] = ( (byte *) ( (void*)(&speedNum) ) )[1];

  RoombaSerial.write((byte)137);   // Drive
  RoombaSerial.write(sendSpeedData[1]);   // Speed
  RoombaSerial.write(sendSpeedData[0]);   // Speed
  RoombaSerial.write((byte)0x80);  // Radius: 0x8000 = Straight
  RoombaSerial.write((byte)0x00);
```

```
 while (1) {
   //変化量
   float dX , //⊿X
         dY ,//⊿Y
         dAngle ,//⊿Y
         dL ,//左車輪の移動量
         dR;//右車輪の移動量

   int tempR, tempL , counterR , counterL;
   tempR = encoderOder(0);//右エンコーダー
   tempL = encoderOder(1);//左エンコーダー
   unsigned long updateTime = millis();

   //前回の計測からカウンタが一瞬で半分以上(32768)変化したらオーバーフローしている
   if ( tempR - g_beforEncoderR < -32768 ) counterR = ((long)tempR + (long)65536 -
(long)g_beforEncoderR); //オーバーフローした場合
   else if ( tempR - g_beforEncoderR > 32768 ) counterR = ((long)tempR - (long)65536 -
(long)g_beforEncoderR); //マイナスになった場合
   else counterR = tempR - g_beforEncoderR;

   if ( tempL - g_beforEncoderL < -32768 ) counterL = ((long)tempL + (long)65536 -
(long)g_beforEncoderL); //オーバーフローした場合
   else if ( tempL - g_beforEncoderL > 32768 ) counterL = ((long)tempL - (long)65536 -
(long)g_beforEncoderL); //マイナスになった場合
   else counterL = tempL - g_beforEncoderL;

   g_beforEncoderR = tempR;//右エンコーダーの情報を更新
   g_beforEncoderL = tempL;//左エンコーダーの情報を更新

   dR = counterR * ENCODER_MM_PAR_PULSE;
   dL = counterL * ENCODER_MM_PAR_PULSE;

   dAngle = (double)(dL - dR) / L;
   g_odometry_Angl += dAngle / M_PI * 180.0;
```

```
    dX = ( dL + dR ) / 2.0 * cos( (g_odometry_Angl / 180) * M_PI ); //⊿X
    dY = ( dL + dR ) / 2.0 * sin( (g_odometry_Angl / 180) * M_PI ); //⊿Y

    g_odometry_x += dX;
    g_odometry_y += dY;

    //Serial.println( g_odometry_x);

    if ( Serial.available()) {
      if ( Serial.read() ==  's' ) {
        RoombaSerial.write((byte)137);   // Stop
        RoombaSerial.write((byte)0x00);
        RoombaSerial.write((byte)0x00);
        RoombaSerial.write((byte)0x00);
        RoombaSerial.write((byte)0x00);
        break;
      }
    }

    if ( g_odometry_x <= -10 * stopNum ) {

      RoombaSerial.write((byte)137);   // Stop
      RoombaSerial.write((byte)0x00);
      RoombaSerial.write((byte)0x00);
      RoombaSerial.write((byte)0x00);
      RoombaSerial.write((byte)0x00);
      delay(10);
      break;
    }
  }
}

// Rotate left with angle
void turnLeftAngle(int stopNum) // stopNum [degree]
{
  int speedNum = fixedSpeed; //fixed speed
```

```
  g_odometry_x = 0;
  g_odometry_y = 0;
  g_odometry_Angl = 0;

  byte sendSpeedData[2];
  sendSpeedData[0] = ( (byte *) ( (void*)(&speedNum) ) )[0];
  sendSpeedData[1] = ( (byte *) ( (void*)(&speedNum) ) )[1];
  RoombaSerial.write((byte)137);   // Drive
  RoombaSerial.write(sendSpeedData[1]);   // Speed
  RoombaSerial.write(sendSpeedData[0]);   // Speed
  RoombaSerial.write((byte)0x00);  // Radius: 0x0001 = Turn in place counter-clockwise
  RoombaSerial.write((byte)0x01);
  while (1) {
    //変化量
    float dX , //⊿X
          dY ,//⊿Y
          dAngle ,//⊿Y
          dL ,//左車輪の移動量
          dR;//右車輪の移動量

    int tempR, tempL , counterR , counterL;
    tempR = encoderOder(0);//右エンコーダー
    tempL = encoderOder(1);//左エンコーダー
    unsigned long updateTime = millis();

    //前回の計測からカウンタが一瞬で半分以上(32768)変化したらオーバーフローしている
    if ( tempR - g_beforEncoderR < -32768 ) counterR = ((long)tempR + (long)65536 -
(long)g_beforEncoderR); //オーバーフローした場合
    else if ( tempR - g_beforEncoderR > 32768 ) counterR = ((long)tempR - (long)65536 -
(long)g_beforEncoderR); //マイナスになった場合
    else counterR = tempR - g_beforEncoderR;

    if ( tempL - g_beforEncoderL < -32768 ) counterL = ((long)tempL + (long)65536 -
(long)g_beforEncoderL); //オーバーフローした場合
```

```
    else  if  (  tempL  -  g_beforEncoderL  >    32768  )  counterL  =  ((long)tempL  -  (long)65536  -
(long)g_beforEncoderL); //マイナスになった場合
    else  counterL = tempL - g_beforEncoderL;

    g_beforEncoderR = tempR;//右エンコーダーの情報を更新
    g_beforEncoderL = tempL;//左エンコーダーの情報を更新

    dR = counterR * ENCODER_MM_PAR_PULSE;
    dL = counterL * ENCODER_MM_PAR_PULSE;

    dAngle = (double)(dL - dR) / L;
    g_odometry_Angl += dAngle / M_PI * 180.0;

    //Serial.println( g_odometry_Angl);

    if ( Serial.available()) {
      if ( Serial.read() ==  's' ) {
        RoombaSerial.write((byte)137);   // Stop
        RoombaSerial.write((byte)0x00);
        RoombaSerial.write((byte)0x00);
        RoombaSerial.write((byte)0x00);
        RoombaSerial.write((byte)0x00);
        break;
      }
    }

    if ( g_odometry_Angl >= stopNum ) {
      RoombaSerial.write((byte)137);   // Stop
      RoombaSerial.write((byte)0x00);
      RoombaSerial.write((byte)0x00);
      RoombaSerial.write((byte)0x00);
      RoombaSerial.write((byte)0x00);

      delay(10);
      break;
    }
```

```
  }
}



// Rotate right with angle
void turnRightAngle(int stopNum) // stopNum [degree]
{
  int speedNum = fixedSpeed; //fixed speed


  g_odometry_x = 0;
  g_odometry_y = 0;
  g_odometry_Angl = 0;


  byte sendSpeedData[2];
  sendSpeedData[0] = ( (byte *) ( (void*)(&speedNum) ) )[0];
  sendSpeedData[1] = ( (byte *) ( (void*)(&speedNum) ) )[1];
  RoombaSerial.write((byte)137);   // Drive
  RoombaSerial.write(sendSpeedData[1]);   // Speed
  RoombaSerial.write(sendSpeedData[0]);   // Speed
  RoombaSerial.write((byte)0xff);  // Radius: 0x0001 = Turn in place clockwise
  RoombaSerial.write((byte)0xff);
  while (1) {
    //変化量
    float dX , //⊿X
          dY ,//⊿Y
          dAngle ,//⊿Y
          dL ,//左車輪の移動量
          dR;//右車輪の移動量


    int tempR, tempL , counterR , counterL;
    tempR = encoderOder(0);//右エンコーダー
    tempL = encoderOder(1);//左エンコーダー
    unsigned long updateTime = millis();


    //前回の計測からカウンタが一瞬で半分以上(32768)変化したらオーバーフローしている
```

```
    if (          tempR - g_beforEncoderR < -32768 ) counterR = ((long)tempR + (long)65536 -
(long)g_beforEncoderR); //オーバーフローした場合
    else if ( tempR - g_beforEncoderR >   32768 ) counterR = ((long)tempR - (long)65536 -
(long)g_beforEncoderR); //マイナスになった場合
    else  counterR = tempR - g_beforEncoderR;

    if (          tempL - g_beforEncoderL < -32768 ) counterL = ((long)tempL + (long)65536 -
(long)g_beforEncoderL); //オーバーフローした場合
    else if ( tempL - g_beforEncoderL >   32768 ) counterL = ((long)tempL - (long)65536 -
(long)g_beforEncoderL); //マイナスになった場合
    else  counterL = tempL - g_beforEncoderL;

    g_beforEncoderR = tempR;//右エンコーダーの情報を更新
    g_beforEncoderL = tempL;//左エンコーダーの情報を更新

    dR = counterR * ENCODER_MM_PAR_PULSE;
    dL = counterL * ENCODER_MM_PAR_PULSE;

    dAngle = (double)(dL - dR) / L;
    g_odometry_Angl += dAngle / M_PI * 180.0;

    //Serial.println( g_odometry_Angl);

    if ( Serial.available()) {
      if ( Serial.read() ==  's' ) {
        RoombaSerial.write((byte)137);   // Stop
        RoombaSerial.write((byte)0x00);
        RoombaSerial.write((byte)0x00);
        RoombaSerial.write((byte)0x00);
        RoombaSerial.write((byte)0x00);
        break;
      }
    }

    if ( g_odometry_Angl <= -stopNum ) {
      RoombaSerial.write((byte)137);   // Stop
```

```
      RoombaSerial.write((byte)0x00);
      RoombaSerial.write((byte)0x00);
      RoombaSerial.write((byte)0x00);
      RoombaSerial.write((byte)0x00);

      delay(10);
      break;
    }
  }
}

//停止
void stopRoomba()
{
  // Stop
  //Serial.println("Stop");
  RoombaSerial.write((byte)137);   // Drive
  RoombaSerial.write((byte)0x00);
  RoombaSerial.write((byte)0x00);
  RoombaSerial.write((byte)0x00);
  RoombaSerial.write((byte)0x00);
}
//右コーナリング旋回
void corneringRight(int speedNum , int radius)
{
  byte sendSpeedData[2];
  sendSpeedData[0] = ( (byte *) ( (void*)(&speedNum) ) )[0];
  sendSpeedData[1] = ( (byte *) ( (void*)(&speedNum) ) )[1];

  radius *= (-1 );//右旋回は半径が負数
  byte sendRadiusData[2];
  sendRadiusData[0] = ( (byte *) ( (void*)(&radius) ) )[0];
  sendRadiusData[1] = ( (byte *) ( (void*)(&radius) ) )[1];

  // Turn in place clockwise
  //Serial.println("Turn");
```

```
  RoombaSerial.write((byte)137);   // Drive
  //速度指定
  //配列の後ろから送信する
  RoombaSerial.write(sendSpeedData[1]);   // Speed
  RoombaSerial.write(sendSpeedData[0]);   // Speed
  RoombaSerial.write(sendRadiusData[1]);   // Speed
  RoombaSerial.write(sendRadiusData[0]);   // Speed
  //RoombaSerial.write((byte)0xff);  // Radius: 0x0001 = Turn in place clockwise
  //RoombaSerial.write((byte)0xff);
}
//左コーナリング旋回
void corneringLeft(int speedNum , int radius)
{
  byte sendSpeedData[2];
  sendSpeedData[0] = ( (byte *) ( (void*)(&speedNum) ) )[0];
  sendSpeedData[1] = ( (byte *) ( (void*)(&speedNum) ) )[1];

  byte sendRadiusData[2];
  sendRadiusData[0] = ( (byte *) ( (void*)(&radius) ) )[0];
  sendRadiusData[1] = ( (byte *) ( (void*)(&radius) ) )[1];

  // Turn in place clockwise
  //Serial.println("Turn");
  RoombaSerial.write((byte)137);   // Drive
  //速度指定
  //配列の後ろから送信する
  RoombaSerial.write(sendSpeedData[1]);   // Speed
  RoombaSerial.write(sendSpeedData[0]);   // Speed
  RoombaSerial.write(sendRadiusData[1]);   // Speed
  RoombaSerial.write(sendRadiusData[0]);   // Speed
}

unsigned int getBattery(int mode)
{
  int bufSize;//バッファに貯まっているサイズ
  int temp;//シリアル通信できたもの
```

```
  unsigned int capacity;//バッテリ容量
  bufSize = RoombaSerial.available();

  RoombaSerial.write((byte)142);
  if ( mode == CURRENT  ) RoombaSerial.write((byte)25);  //右モーターの番号
  else                    RoombaSerial.write((byte)26);   //左モーターの番号

  while ( RoombaSerial.available() < 2 ) //2byte 来るまで無限ループ
  { }

  if ( RoombaSerial.available() >= 2 ) //
  {
    temp = RoombaSerial.read();  capacity = temp * 256;
    temp = RoombaSerial.read();  capacity += temp;
  }

  return capacity;
}
```

デプスカメラと画像処理を用いた撮影補助装置の開発

指導教員　金丸　隆志　教授
S5-16023　工藤 翔太　S5-16049　中野　駿
S5-16059　藤井　周作　S5-16061　眞野　峻彰

1. 緒言

近年，スマートフォンからの動画視聴サイトの利用者は増加傾向にある．主な動画視聴サイトの 1 つに「YouTube」があり，そこで動画を投稿するいわゆる「YouTuber」も増加傾向にある．しかし，単独で活動する YouTuber が抱える問題の 1 つに，撮影用カメラを動かしながら自身を撮影する手段が少ないというものがある．

そこで，この問題を解決するために，人への追従機能を備えた撮影補助装置の開発を目指す．

2. 開発にあたって

人への追従機能を実現するためには，人を検出し，その距離を計測する必要がある．この両方を実現するために，Intel 社の RealSense D435 を使用することにした．RealSense D435 は深度計測が可能なステレオビジョンの深度カメラであり，深度センサと RGB センサ，IR 投射器を搭載している．

また，撮影補助装置の移動手段には，ルンバオープンインターフェイスによって Arduino を用いた制御が可能なルンバを使用する．

3. ハードウェアの製作

製作した撮影補助装置のハードウェアを図 1 に示す．ルンバの上に三層構造の土台を乗せ，上段に動画撮影用のスマートフォンと RealSense D435 があり，中段に Visual Studio や OpenCV を動かすためのパソコンとして Surface Pro 4，下段にルンバに指令を送るマイコンボードの Arduino Mega 2560 を搭載する．

図 1　撮影補助装置のハードウェア

4. 機器の接続と動作の仕組み

Arduino とルンバを ROI 端子により接続し，Surface Pro 4 と RealSense D435，Arduino Mega 2560 を USB ケーブルで接続する．RealSense から取得した RGB 画像と Depth 画像より人検出と距離算出を実行し，その情報をルンバに送信する．

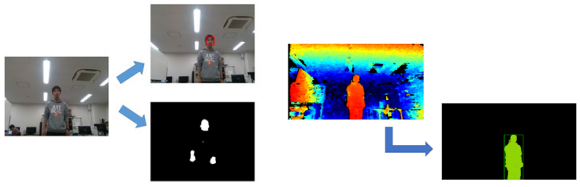

図 2　人検出方法

5. 実験

開発した撮影補助装置を用いて検証を行う．人検出と距離測定が出来ているか，人をスマートフォンのカメラの中心に捉えているか，人と一定距離を保ちながらの追従ができるか，人を見失ったときに一時停止できるかの確認を行う．また，スマートフォンで撮影した動画が見にくい動画になっていないかも確認する．

6. 結果

人検出と距離測定，人をスマートフォンのカメラの中心に捉えること，人を見失ったときに一時停止すること，いずれも正常に行われた．

一方で，人と一定距離を保ちながらの追従は距離測定の精度が完全でないことから前後移動を繰り返すことがあった．また，人検出可能な距離が短い，撮影した映像がぶれるなどの問題があった．これはそれぞれ，本研究で作成した検出方法では遠くの人を検出しにくいこと，装置の重心が高いことから撮影補助装置の揺れを完全にはなくせないことが原因である．

7. 結言

① デプスカメラと画像処理を用いた撮影補助装置を製作した．

② デプスカメラを使った人検出と距離測定のプログラムを開発した．

＜参考文献＞
池　宏武・他 6 名，“画像処理に基づく自動書類運搬装置の開発”　工学院大学卒業論文，2017 年 3 月