

2025年度 卒業論文

画像処理およびセンサを用いた  
遠隔在庫把握システムの構築

Development of a Remote Inventory Monitoring System  
Using Image Processing and Sensors

成蹊大学 理工学部 理工学科 機械システム専攻  
流体力学研究室

S226109 前澤 栄飛  
S226003 秋元 大生

指導教授：小川 隆申

提出日 2026年 1月 24日

# 目次

<b>第1章</b>	<b>序論</b>	<b>1</b>
1.1	研究背景	1
1.2	研究目的	2
<b>第2章</b>	<b>画像処理による在庫検出</b>	<b>3</b>
2.1	画像取得に用いた機器	3
2.2	OpenCV	5
2.3	輪郭抽出	10
2.4	類似商品の識別	14
<b>第3章</b>	<b>超音波センサによる在庫の把握</b>	<b>18</b>
3.1	超音波センサによる在庫把握の方法	18
3.2	研究に使用した装置	19
3.3	超音波センサの配置	19
3.4	超音波センサの距離測定に使用するプログラム	19
3.5	超音波センサの試行の結果と考察	20
<b>第4章</b>	<b>デプスカメラによるデータ収集</b>	<b>21</b>
4.1	デプスカメラによる在庫把握の方法	21
4.2	デプスカメラの配置と距離の検出方法	22
<b>第5章</b>	<b>LINE Bot を用いた遠隔の在庫把握</b>	<b>23</b>
<b>第6章</b>	<b>総合的考察</b>	<b>24</b>
6.1	本研究の限界と今後の課題	24
<b>参考文献</b>		<b>26</b>
<b>謝辞</b>		<b>27</b>
<b>付録A</b>	<b>プログラム</b>	<b>1</b>
A.1	ソースコード	1
<b>付録B</b>	<b>原稿非掲載データ</b>	<b>4</b>
B.1	実験結果	4

# 第1章 序論

## 1.1 研究背景

近年、様々な技術の発達に伴い、人々の生活はより快適になっている。その技術として、デジタル技術が挙げられる。これには、家電製品や自動車、センサ、カメラなどの様々な「モノ」がインターネットに接続し、相互にデータをやり取りする IoT (Internet of Things) <sup>[2]</sup>、感知器や測定器などを用いて対象の定量的な情報を取得するセンシング<sup>[2]</sup>、コンピューターに人間の知的活動を模倣させる AI (Artificial Intelligence) などが該当し<sup>[2]</sup>、農業、製造業、医療、物流と幅広い分野で活用が進んでいる<sup>[2]</sup>。具体的な活用事例として、スマートストアが挙げられる。これは、デジタル技術を取り入れ、様々な作業を自動化、最適化した小売店舗のことである。スマートストアでは、決済処理や商品管理、混雑状況の把握、入退店管理などにデジタル技術が活用されており、効率的で正確な店舗運営の実現に寄与している<sup>[3]</sup>。このように、デジタル技術の普及により、データの収集および活用が容易となり、業務の効率化やシステムの高度化が進んでいる。そして、大学においてもこのような技術の需要は高まっている。大学では、キャンパス内に設置された購買施設の1つとして、図1.1に示す無人店舗が存在する。この施設は、学生をはじめとする多くの人々に利用されており、学内コミュニティの活性化に寄与している。その一方、無人店舗の運営においていくつかの課題が存在し、特に重要な課題として、在庫状況を手軽に確認する手段が十分に整備されていない点が挙げられる。これにより、目当ての商品を購入するために無人店舗を訪れたにもかかわらず、売り切れや欠品によって商品入手できず、時間を浪費してしまう可能性がある。また、無人店舗から離れた場所では、販売されている商品を確認できないことから、無人店舗が利用選択肢として選ばれにくくなり、結果として無人店舗および利用者の双方に悪影響を及ぼすことが懸念される。そこで、無人店舗の在庫状況を遠隔から把握可能なシステムが導入されれば、無人店舗の利便性向上による利用者の増加や、さらなる学生コミュニティの活性化が期待できる。



図 1.1: 大学内の無人店舗

## 1.2 研究目的

本研究では、大学キャンパス内に設置された無人店舗に超音波センサやデプスカメラを設置し、在庫状況に関するデータを収集、管理する方法および収集したデータを利用者に共有する方法を検討する。さらに、これらの検討結果に基づき、利用者が遠隔から在庫状況を把握可能なシステムの構築を目的とする。

# 第2章 画像処理による在庫検出

無人店舗の在庫状況に関するデータの収集を実現するため、複数の技術的アプローチについて検討を行い、最適な在庫検出手法を明確にする。本章では、画像処理技術を用いた在庫検出について、OpenCV を用いた検出方法、輪郭抽出、座標による商品識別、検出した結果と考察を説明する。

## 2.1 画像取得に用いた機器

画像処理を行うにあたり、無人店舗の商品棚の画像を取得するため、スマートフォンのカメラ機能を利用した。本章では、画像処理技術を用いた在庫検出の実現性について検討することを主な目的としているため、コスト削減や画像取得の容易さの観点からこの選択をした。実際の画像取得に用いたスマートフォンを図 2.1 に、取得した商品棚の画像を図 2.2、図 2.3 に示す。



図 2.1: 画像取得用スマートフォン



図2.2: おにぎりを中心とする商品棚



図2.3: 様々な色の商品が並ぶ棚

画像を一定の品質で取得するため、カメラは等倍かつ品棚から約3mの位置で撮影を行う。図2.4に示す一部結果を可視化した画像において、商品の周りに検出を表現する矩形（バウンディングボックス）が表示されていることから、本章で用いたスマートフォンのカメラ機能は、画像処理による商品検出に十分な性能を満たしていると判断した。スマートフォンのカメラ性能を表2.1に示す。



図2.4: 一部商品棚の検出結果

表2.1: スマートフォンのカメラ性能

使用機種	Sony Xperia 10 VI 5G
使用カメラ	メインカメラ
画像解像度	1920×1080
総画素数	約207万画素

表 2.1 に示した使用するカメラの画像解像度が  $1920 \times 1080$  であることから、本章は、解像度  $1920 \times 1080$  と同様、もしくはそれ以上の性能を有するカメラを用いたときの画像処理技術による在庫検出について検討する。

## 2.2 OpenCV

OpenCV (Open Source Computer Vision Library) は、画像および動画に関する処理機能・検出機能をまとめたオープンソースのライブラリである [5]。Windows や Linux, iOS, Android などさまざまな OS に対応しており、Raspberry Pi などの端末上で利用することもできる。さらに、豊富な画像処理機能を搭載しており、高度な画像処理を比較的容易に実装できるという特徴を有しているため、本研究の目的達成に重要な、様々な種類の商品の高精度検出およびリアルタイム性のあるデータ収集に最適だと考え、今回 OpenCV を使用した。

### 2.2.1 グレースケール変換

OpenCV の機能にグレースケール変換がある。これは、色の情報を省き、明るさの度合いという情報のみで表現するための画像処理であり [6]、図 2.5 に示すグレースケール変換前後の画像比較から分かる通り、グレースケール変換によって、画像内の明暗を明確にすることができる。

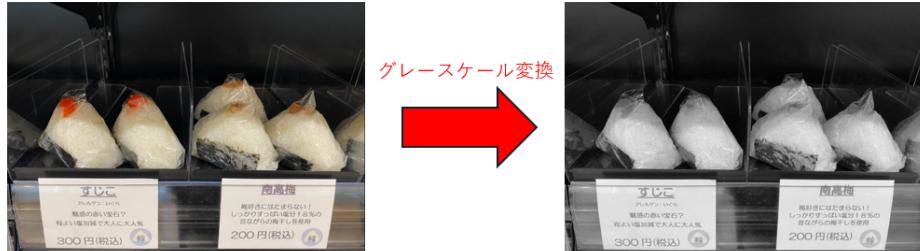


図 2.5: グレースケール変換を適用した画像

通常、処理を行う前のカラー画像（以下 RGB 画像とする）は、R (Red), G (Green), B (Blue) の 3 つの値（以下 RGB 値とする）で表現され、それぞれ  $0 \sim 255$  の数値をとる。一方、グレースケール変換を行った画像（以下グレースケール画像とする）は、輝度という 1 つの値で表現され [7]、 $0 \sim 255$  の数値をとる。図 2.6 に示す輝度の値による明るさの違いから分かる通り、値が小さいと暗く、大きいと明るく表現される。

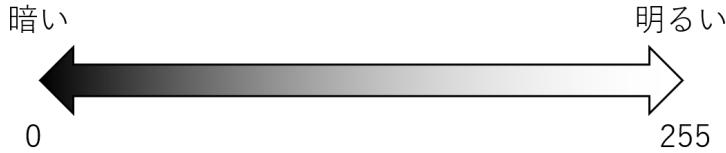


図2.6: 輝度の値による明るさ

グレースケール変換の仕組みについて説明する。変換方法は複数存在するが、OpenCVでは、加重平均法が用いられている[8]。グレースケール画像を表現する輝度という値は、RGB画像のRGB値から計算された値であり、加重平均法はR, G, Bの値それぞれに重み付けをして輝度を計算する方法である。OpenCVにおいて輝度は

$$\text{輝度} = 0.2989 \times R + 0.5870 \times G + 0.1140 \times B \quad (2.1)$$

である。このRGBそれぞれに対する重みは、緑の光が最も検知しやすく、青の光は比較的検知にくい[9]という人間の視覚特性をもとに設定されている。

グレースケール変換の活用が有効な場面を検討する。RGBはデジタル機器同様、人間の視覚が光の総量に強く依存する性質を利用して構造化されており[10]、光の三原色として表現される。そのため、RGB値をすべて255にして表現される色は白であり、RGB値をすべて0にして表現される色は黒である。(2.1)より、RGB値をすべて255にしたとき、輝度の値は最大である255をとり、RGB値をすべて0にしたとき、輝度の値は最小である0をとる。したがって、画像にグレースケール変換を適用した際、白に近い色を持つ物体は明るく表現され、黒に近い色を持つ物体は暗く表現される。本研究において、図2.7に示す実際の商品棚の一部から分かる通り、研究対象である無人店舗の一部の商品棚は、おにぎりのような白に近い色が特徴的な商品を中心に扱っている。また、図2.8に示す実際の商品棚の全体から分かる通り、無人店舗の商品棚全体が黒いデザインとなっている。したがって、おにぎりを中心に扱う商品棚に関しては、グレースケール変換を活用することで、商品と背景の差が明確になり、精度の高い検出が期待できる。加えて、グレースケール画像が輝度という1つの値で表現できる特徴から、情報量の削減による処理の高速化が期待できる。



図2.7: 白に近い色を持つ商品が多い棚



図2.8: 無人店舗の商品棚全体

## 2.2.2 HSV 変換

OpenCV の機能に HSV 変換がある。これは、RGB 画像を色相 (Hue), 彩度 (Saturation), 明度 (Value) の 3 つの要素で表現された画像（以下 HSV 画像とする）に変換する機能である [11]。色相 (Hue) について説明する。これは、具体的な色の種類を表す要素で、図 2.9 に示す実際のカラー ホイールから分かる通り、0~359 の値を用いて様々な色を表現する。しかし、OpenCV では効率的な処理を目的として 8 bit 画像を主な対象としているため、扱う数値を 0~255 の範囲で表現することが望ましい。そのため、図 2.10 に示す OpenCV のカラー ホイールから分かる通り、OpenCV は色相の値を  $1/2$  にスケーリングし、0~179 の値を用いて色を表現する [12]。

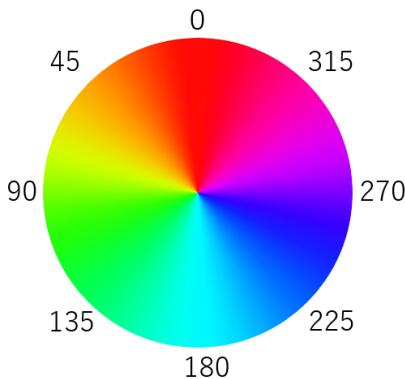


図2.9: 一般的な色相範囲

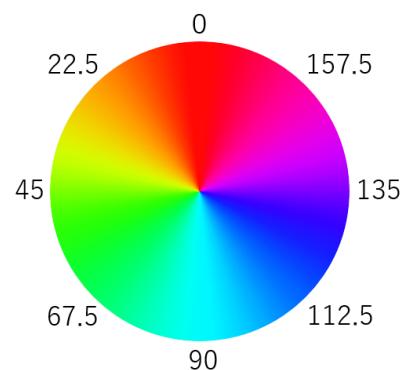


図2.10: OpenCV の色相範囲

彩度 (Saturation) について説明する。これは、色の鮮やかさや濃さを表す要素で [11]、色相と異なり、数値に明確な範囲が存在しないため、OpenCV は 0~255 の範囲に値をスケーリングして

表現する。図2.11に示す数値ごとの彩度表現から分かる通り、彩度の値が255をとると色は最も鮮やかになり、0の値をとると最も鈍くなる。

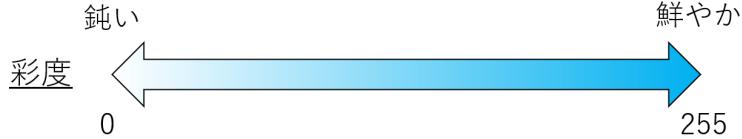


図2.11: 彩度の値による変化

明度 (Value) について説明する。これは、色の明るさを表す要素で [11]、彩度と同様に、数値に明確な範囲が存在しないため、OpenCV は0~255の範囲に値をスケーリングして表現する。図2.12に示す数値ごとの明度表現から分かる通り、明度の値が255をとると色は最も明るくなり、0の値をとると最も暗くなる。



図2.12: 明度の値による変化

HSV 変換の仕組みについて説明する。グレースケール変換同様、HSV 変換においても RGB 値を基に計算されている。RGB 値をそれぞれ  $R$ ,  $G$ ,  $B$  とし、これらを0~1の範囲で表現したものをそれぞれ  $R'$ ,  $G'$ ,  $B'$  ( $R' = \frac{R}{255}$ ,  $G' = \frac{G}{255}$ ,  $B' = \frac{B}{255}$ ) とする。また、RGB の最大値 ( $\max(R, G, B)$ ) を  $C_{max}$ 、最小値 ( $\min(R, G, B)$ ) を  $C_{min}$  とし、その差 ( $C_{max} - C_{min}$ ) を  $\Delta$  とする。色相 ( $H$ ) は、 $C_{min} = R$  のとき

$$H = \frac{1}{2}(60 \times (\frac{B' - G'}{\Delta}) + 180) \quad (2.2)$$

であり、 $C_{min} = G$  のとき

$$H = \frac{1}{2}(60 \times (\frac{R' - B'}{\Delta}) + 300) \quad (2.3)$$

であり、 $C_{min} = B$  のとき

$$H = \frac{1}{2}(60 \times (\frac{G' - R'}{\Delta}) + 60) \quad (2.4)$$

である。また、OpenCVでは $\Delta=0$ のとき、 $H=0$ として定義される。

彩度（S）は

$$S = \frac{\Delta}{C_{max}} \quad (2.5)$$

である。また、OpenCVでは $C_{max}=0$ のとき、 $S=0$ として定義される。

明度（V）は

$$V = C_{max} \quad (2.6)$$

である[13]。

HSVは、色を鮮やかさや明るさという直感的に分かりやすい要素で表現するため、原色の組み合わせで表現するRGBに比べ、各要素を変動させた場合の色の変化がイメージしやすく、細かな色の調整が簡単にできる。本研究において、図2.3に示した実際の商品棚の一部から分かる通り、無人店舗の一部の商品棚は、様々な色の商品を扱っている。そのため、輝度の値が商品によって異なり、図2.13に示す様々な色の商品が並ぶ棚のグレースケール画像を用いた検出結果から分かる通り、検出精度が低くなる。そこでHSV画像を活用することで、安定した精度の検出が期待できる。



図2.13: 様々な色の商品が並ぶ棚のグレースケール検出

## 2.3 輪郭抽出

商品の位置を特定するにあたり、グレースケール画像およびHSV画像から商品の輪郭抽出を行う。具体的には、商品と背景を判別するための閾値を設定し、これを満たす画素集合を輪郭として抽出する。閾値とは、判断の境目となる値のこと [14]、OpenCVでは、利用者が任意に設定できる。

### 2.3.1 グレースケール画像の輪郭抽出

グレースケール画像に適用する閾値を考える。2.2.1節で述べたように、グレースケール画像では、輝度という0～255の範囲をとる1つの値で表現されるため、閾値はこの値を用いて適用する。図2.14に示す照明が商品棚へ及ぼす影響から分かる通り、無人店舗の商品棚は4段ある棚の段数のうち、最上段のみ照明が強く当たる構造になっている。そのため、輝度の値で表現するグレースケール画像では、全体に同一の閾値を設定すると、最上段とその他の段で検出精度に差が生じる。この問題に対処するため、画像内座標を利用し、最上段とそれ以外の範囲を切り分け、最上段の閾値を180～255、それ以外の範囲を100～255にそれぞれ設定する。この閾値は、図2.7に示したグレースケール変換の適用する商品棚において、様々な閾値を試した結果、最も検出精度が高くなると判断した値である。照明の当たり方を考慮した閾値設定のイメージを図2.15に示す。



図2.14: 照明の商品棚への影響



図 2.15: グレースケール画像の閾値設定のイメージ

### 2.3.2 HSV 画像の輪郭抽出

HSV 画像に適用する閾値を考える。2.2.2 節で述べたように、HSV 画像では、0～179 の範囲をとる色相、0～255 の範囲をとる彩度、明度という 3 つの値で表現されるため、閾値はこれらの値を用いて適用する。図 2.3 に示した HSV 変換を適用する商品棚から分かる通り、様々な色の商品が存在するため、画像全体に同一の閾値を設定すると商品ごとの検出精度に差が生じる。この問題に対処するため、画像内座標を利用し、商品ごとに閾値を設定する。図 2.13 に示したグレースケール変換では検出精度が低くなる商品棚において、各商品に設定した閾値を表 2.2 に示す。この閾値は、図 2.13 に示したグレースケール変換を適用する商品棚において、存在する商品に対し様々な閾値を試した結果、最も検出精度が高くなると判断した値である。商品ごとの閾値設定のイメージを図 2.16 に示す。また、表 2.2 に示す閾値のうち、商品名が none となっているのは、図 2.13において商品が存在しない棚で誤検出が起こらないか確かめるためのものである。

表2.2: HSV画像における閾値設定

商品名	色相 (H)	彩度 (S)	明度 (V)
白玉粒あんベーグル	20~35	100~255	100~255
クランベリー&クリームチーズベーグル	20~35	100~255	100~255
イチジク&クリームチーズベーグル	20~35	100~255	100~255
明太ポテトベーグル	10~25	100~255	20~200
ビーフカレーベーグル	10~25	100~255	20~200
ベーコンペッパーべーグル	20~35	100~255	100~255
アップルシナモンベーグル	20~35	100~255	100~255
チーズベーグル	10~25	100~255	20~200
none	10~70	100~255	100~255



図2.16: HSV画像の閾値設定イメージ

### 2.3.3 輪郭抽出の手法

輪郭抽出を行うにあたり、RETR EXTERNAL と CHAIN APPROX SIMPLE という 2 つの定数を用いる。閾値を設定した画像は、各画素において輝度もしくは HSV の値が閾値を満たすか満

たさないかのどちらかに区別される。RETR EXTERNALは、閾値を満たす画素の集合のうち、その最外周の輪郭のみを抽出する[15]。閾値を満たす画素を白、満たさない画素を黒で表現した商品棚を図2.17に示す。また、RETR EXTERNALによる輪郭抽出のイメージを図2.18に示す。

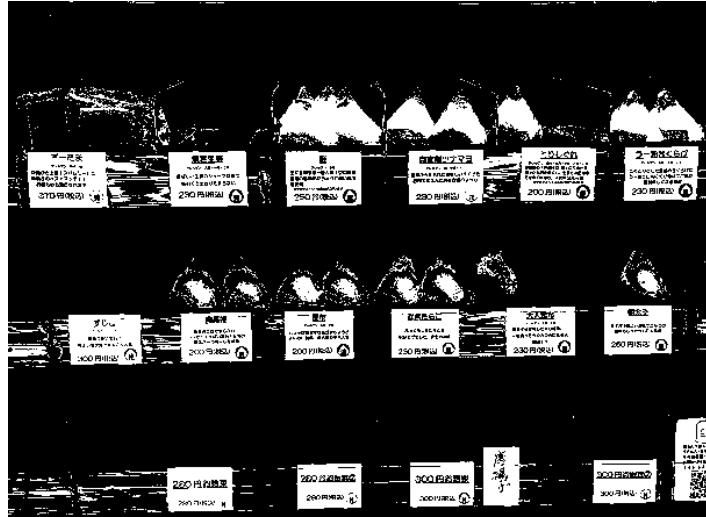


図2.17: 商品と背景を白と黒で表現した画像

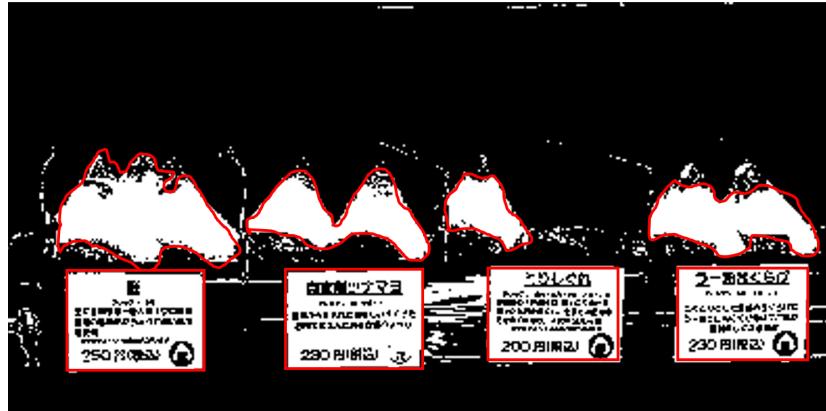


図2.18: RETR EXTERNALの輪郭抽出イメージ

輪郭は無数の点によって表現される。CHAIN APPROX SIMPLEは、輪郭が完全に直線になる部分の点を省いて表現する[15]。これによって、リアルタイム性が求められる本研究において、特に重要な処理の高速化を実現できる。CHAIN APPROX SIMPLEによる輪郭表現のイメージを図2.19に示す。

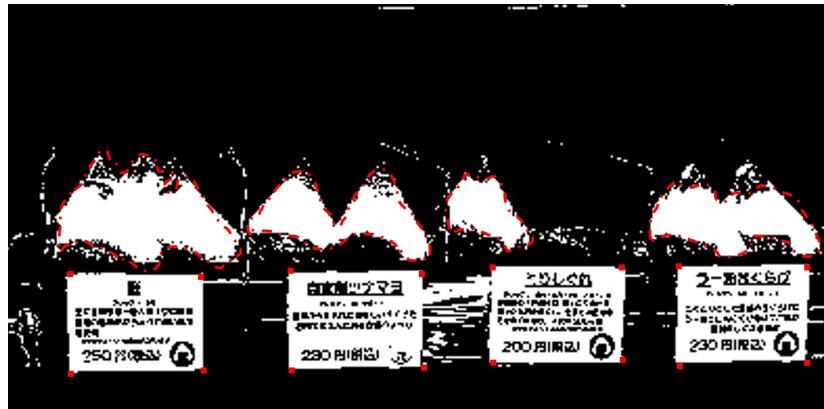


図 2.19: CHAIN APPROX SIMPLE の輪郭抽出イメージ

## 2.4 類似商品の識別

図 2.20 に示す見た目近い異なる商品から分かる通り、無人店舗には見た目から種類の識別が困難な商品が存在する。そこで、抽出した輪郭を基に、画像内座標を用いて商品の種類を識別する。



図 2.20: 見た目の近い商品

### 2.4.1 輪郭の座標表現

抽出した商品の輪郭を、1点の座標で表現する。手法としては、輪郭をバウンディングボックスと呼ばれる矩形で表し、その中心点の座標を算出する。図 2.21 に示すバウンディングボックスおよび中心点のイメージから分かる通り、バウンディングボックスは、抽出した輪郭を完全に含み、かつ最小になる矩形であり、バウンディングボックス左上の画像内座標  $(x, y)$ 、バウンディング

ボックスの幅  $w$ , バウンディングボックスの高さ  $h$  という 4 つの情報と, これらの情報から算出したバウンディングボックスの中心点の画像内座標で表現する. 中心点の画像内  $x$  座標  $C_x$  は

$$C_x = x + \frac{w}{2} \quad (2.7)$$

であり, 画像内  $y$  座標  $C_y$  は

$$C_y = y + \frac{h}{2} \quad (2.8)$$

である. バウンディングボックスで商品の輪郭を表現し, 画像内の商品の位置を 1 つの座標  $(C_x, C_y)$  で定義することによって, 商品の画像内の位置情報を利用する処理の高速化が期待できる.

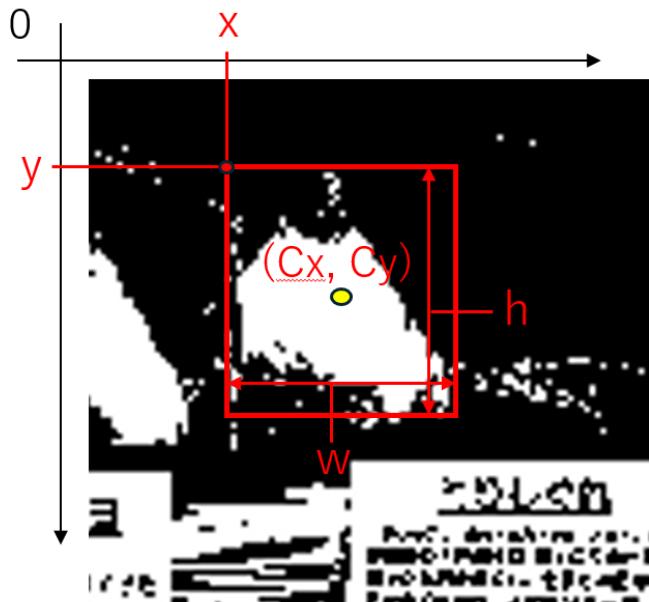


図 2.21: バウンディングボックスおよび中心点イメージ

図 2.17 に示した抽出した輪郭を白で表現した画像から分かる通り, 商品以外に対しても輪郭抽出は行われる. そのため, 商品が存在する範囲のみにバウンディングボックスの表現を適用し, 商品以外の輪郭の座標データを取得しないよう設定する. これによって, 処理の高速化が期待できる. 商品が存在する範囲のみでバウンディングボックスを表現するイメージを図 2.22 に示す.



図 2.22: 商品のみ輪郭抽出のイメージ

図 2.23 に示す輪郭のサイズ指定無しのバウンディングボックス表現から分かる通り，バウンディングボックスは極端に小さい輪郭や大きい輪郭も表現してしまう。そのため，座標データを取得するバウンディングボックスをサイズでフィルタリングする。具体的には，14500~75000 画素数（画素面積）のバウンディングボックスのみ座標データを取得するように設定しており，これは様々な条件でフィルタリングを行った結果，極端なサイズの輪郭を排除し，全商品の輪郭を十分に表現できると判断した閾値である。輪郭のサイズによるフィルタリングを行うことで，ノイズを除去し，余計なデータの取得，計算を避け，処理を高速化できる。輪郭のサイズによるフィルタリングを行ったバウンディングボックス表現を図 2.24 に示す。



図2.23: サイズ指定なしの検出

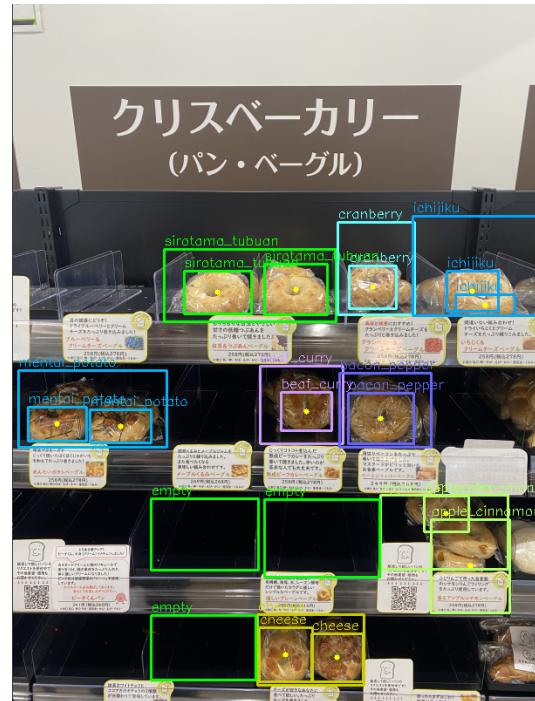


図2.24: サイズ指定ありの検出

## 2.4.2 座標による商品の識別

# 第3章 超音波センサによる在庫の把握

本章では、本研究で行った超音波センサによる在庫把握の手段について述べる。

## 3.1 超音波センサによる在庫把握の方法

本研究では、無人店舗内の商品棚における在庫の把握を超音波センサを用いることによって行った。商品棚の奥側に超音波センサを設置し、センサから発信された超音波が商品に反射して戻ってくるまでの時間を距離に計算し直すことで行う。距離は以下の式で求められる。

$$\text{距離} = \frac{\text{音速} \times \text{時間}}{2} \quad (3.1)$$

ここで、音速は約 343m/s である。計測した距離の長短で在庫の状況を判断する。使用した超音波センサは図 3.1 に示す Rainbow E-Technology 社の HC-SR04 である。このセンサは単体では動作しないため、マイコンボードと組み合わせて使用する必要がある。超音波センサで取得したデータを USB 経由でパソコンに送信し、そのデータをパソコンで評価する。



図 3.1: Rainbow E-Technology 社 HC-SR04

## 3.2 研究に使用した装置

### 3.2.1 超音波センサの性能

HC-SR04 の性能を表 3.1 に示す。測定可能距離は 0.02m～4.5m であり、商品棚での在庫把握は可能であると考えられる。

表 3.1: HC-SR04 の精度

動作電圧	3～5.5V
測定可能距離	0.02～4.5m
測定方式	超音波
動作温度	-10～70°C

### 3.2.2 マイコンボードを含めた装置全体

前述の通り、この超音波センサは単体では動作しないためマイコンボードと組み合わせる必要がある。マイコンボードには Raspberry Pi Pico 2 WH を使用した。装置の全体図を図 3.2 に示す。

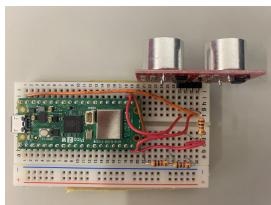


図 3.2: 装置全体図

## 3.3 超音波センサの配置

超音波センサを商品棚の奥側のスペースに配置し、商品との距離を計測する。

## 3.4 超音波センサの距離測定に使用するプログラム

超音波センサで距離を測定するにあたり、ソースコード A.1 に示す Python のプログラムをマイコンボードに書き込んだ。

ソースコード 3.1: findContours

```
1 from machine import Pin
2 import machine
3 import utime
4
5 hasshin = Pin(14, Pin.OUT) #14番を出力（発信）と定義
6 jyushin = Pin(15, Pin.IN) #15番を入力（受信）と定義
7
8 def read_distance():
9     hasshin.low() #9行から13行までのコードで超音波を一瞬だけ発信
10    utime.sleep_us(2)
11    hasshin.high()
12    utime.sleep_us(10)
13    hasshin.low()
14    while jyushin.value() == 0: #超音波が受信されていないとき
15        start = utime.ticks_us() #現時点での稼働時間をマイクロ秒単位で返す
16    while jyushin.value() == 1: #超音波が受信されているとき
17        goal = utime.ticks_us() #現時点での稼働時間をマイクロ秒単位で返す
18    #超音波は一瞬なので、jyushin.value()==0となり、無限ループにはならない
19    passed = goal - start
20    distance = float((passed * 340 * 0.0001) / 2) #cmで出力
21    print(distance)
22
23 while True:
24     read_distance()
25     utime.sleep(1)
```

センサで取得したデータをパソコンで評価するためのプログラムをソースコード 3.2 に示す。パソコンの OS は Windows を想定している。

ソースコード 3.2: findContours

```
1 from datetime import datetime
2 import serial
3
4 ser = serial.Serial('COM3', 9600)
5
6 while(1):
7     dt_now = datetime.now()
8     distance = ser.readline().decode('utf-8').strip()
9     if float(distance) > float(21.0):
10         print(f'{dt_now}在庫なし{distance}')
11     elif float(distance) > float(10.0):
12         print(f'{dt_now}在庫あり{distance}')
13     else:
14         print(f'{dt_now}在庫大量{distance}')
```

### 3.5 超音波センサの試行の結果と考察

# 第4章 デプスカメラによるデータ収集

本章では、本研究で行ったデプスカメラによる在庫把握の手段について述べる。

## 4.1 デプスカメラによる在庫把握の方法

本研究では、超音波センサだけではなくデプスカメラも用いて無人店舗内の在庫数の把握を行った。使用したデプスカメラは図 4.1 に示す DFRobot 社の SEN0581 と図 4.2 Arducam 社の B0410 である。



図 4.1: DFRobot 社 SEN0581

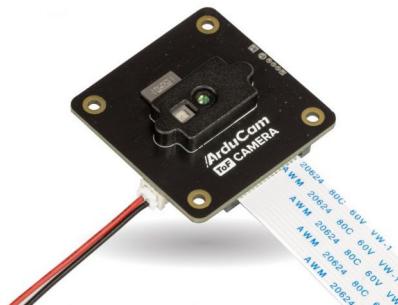


図 4.2: Arducam 社 B0410

#### 4.1.1 デプスカメラの精度

これらのデプスカメラの性能を表4.1に示す。

表4.1: D435 の精度

型番	B0410	SEN0581
解像度	240x180	100x100
コマ数	30fps	1~20fps
計測範囲	2m または 4m	0.15m~1.5m
画角	対角 70°	対角約 92°
接続方法	Raspberry Pi 本体に接続	UART または USB でパソコンに接続

#### 4.2 デプスカメラの配置と距離の検出方法

## **第5章 LINE Botを用いた遠隔の在庫把握**

# 第6章 総合的考察

本章では、(第2章)、第3章で実施した超音波センサによる在庫把握と、第4章で実施したデプスカメラによる在庫把握、(第5章)について考察を行う。

## 6.1 本研究の限界と今後の課題

本節では、本研究における限界を明示するとともに、今後の課題について述べる。

### 6.1.1 本研究の課題

#### 6.1.1.1 超音波センサに関する課題

無人店舗で取り扱っている商品は60種類とかなり多い。そのため、超音波センサを1商品に対し1台設置すると多額の費用が発生し、電源供給の課題も発生する。その課題を解決するには、1台のマイコンボードに対し2つ以上の超音波センサを設置するなどの工夫が必要となる。また、無人店舗において、店舗利用者が商品を手に取ったあと、商品棚の奥側にある在庫が前出しされないという問題がある。商品の前出しがされないと実際は在庫数が少ないと在庫数に余裕ありと判断されるため、在庫把握に超音波センサを用いる場合はこの問題を対処する必要がある。この問題を解決するために一部の小売店では図6.1のようなフェイシングスタンドを用いているが、無人店舗は商品の種類が多いため導入のハードルが高く、おにぎりのように潰れてしまいやすい商品があるため、導入は厳しい。



図6.1: フェイシングスタンド

#### 6.1.1.2 デプスカメラに関する課題

## 参考文献

[1]

[2]

## 謝辞

本研究にご協力ならびにご助言をいただいた全ての皆様に深く感謝いたします。特に、小川隆申教授ならびに謝文昂助教には、終始熱心なご指導をいただきました。併せて、流体力学研究室の皆様にはご助言とご支援をいただきましたことに深く感謝いたします。

# 付録A プログラム

## A.1 ソースコード

以下はソースコードである。超音波センサでの在庫数把握において、マイコンボードで実行したプログラムがA.1である。

ソースコード A.1: findContours

```
1 from machine import Pin
2 import machine
3 import utime
4
5 hasshin = Pin(14, Pin.OUT) #14番を出力（発信）と定義
6 jyushin = Pin(15, Pin.IN) #15番を入力（受信）と定義
7
8 def read_distance():
9     hasshin.low() #9行から13行までのコードで超音波を一瞬だけ発信
10    utime.sleep_us(2)
11    hasshin.high()
12    utime.sleep_us(10)
13    hasshin.low()
14    while jyushin.value() == 0: #超音波が受信されていないとき
15        start = utime.ticks_us() #現時点での稼働時間をマイクロ秒単位で返す
16    while jyushin.value() == 1: #超音波が受信されているとき
17        goal = utime.ticks_us() #現時点での稼働時間をマイクロ秒単位で返す
18    #超音波は一瞬なので、jyushin.value()==0となり、無限ループにはならない
19    passed = goal - start
20    distance = float((passed * 340 * 0.0001) / 2) #cmで出力
21    print(distance)
22
23 while True:
24     read_distance()
25     utime.sleep(1)
```

超音波センサ A.2に機械学習に用いた一連のプログラムを示す。

ソースコード A.2: MachineLearning

```
1 import tensorflow as tf
2 from tensorflow import keras
3 from tensorflow.keras.models import Sequential
4 from tensorflow.keras.layers import Activation, Dense, Dropout, Flatten
5 from tensorflow.keras.utils import to_categorical
6 from tensorflow.keras.preprocessing.image import array_to_img, img_to_array, load_img
7 from tensorflow.keras.optimizers import Adagrad
```

```

8  from tensorflow.keras.optimizers import Adam
9  from sklearn.model_selection import train_test_split
10 import pandas as pd
11 import numpy as np
12 from PIL import Image
13 import os
14 import re
15 import matplotlib.pyplot as plt
16
17 #画像を読み込む
18 def list_pictures(directory, ext='jpg|jpeg|bmp|png|ppm'):
19     return [os.path.join(root, f)
20             for root, _, files in os.walk(directory) for f in files
21             if re.match(r'([\w]+.(?:' + ext + '))', f.lower())]
22
23 X = []
24 Y = []
25 #0人の画像
26 for picture in list_pictures('./train_picture/zero/'):
27     img = img_to_array(load_img(picture, target_size=(24,32)))
28     X.append(img)
29     Y.append(0)
30 #1人の画像
31 for picture in list_pictures('./train_picture/one/'):
32     img = img_to_array(load_img(picture, target_size=(24,32)))
33     X.append(img)
34     Y.append(1)
35 #2人の画像
36 for picture in list_pictures('./train_picture/two/'):
37     img = img_to_array(load_img(picture, target_size=(24,32)))
38     X.append(img)
39     Y.append(2)
40 #3人の画像
41 for picture in list_pictures('./train_picture/three/'):
42     img = img_to_array(load_img(picture, target_size=(24,32)))
43     X.append(img)
44     Y.append(3)
45 # arrayに変換
46 X = np.asarray(X)
47 Y = np.asarray(Y)
48 # 画素値を0から1の範囲に変換
49 X = X.astype('float32')
50 X = X / 255.0
51 # クラスの形式を変換
52 Y = to_categorical(Y)
53 # 学習用データとテストデータに分ける
54 X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30, random_state=111)
55
56 # モデルの構築
57 model = Sequential()
58 model.add(keras.layers.Conv2D(16, (3, 3), padding='same', input_shape=X_train.shape[1:])) # 置込み層
59 model.add(Activation('relu')) # 活性化関数
60 model.add(keras.layers.MaxPooling2D(pool_size=(2, 2))) # プーリング層

```

```

61
62 model.add(keras.layers.Conv2D(32, (3, 3), padding='same'))
63 model.add(Activation('relu'))
64 model.add(keras.layers.MaxPooling2D(pool_size=(2, 2)))
65
66 model.add(keras.layers.Conv2D(64, (3, 3), padding='same'))
67 model.add(Activation('relu'))
68 model.add(keras.layers.MaxPooling2D(pool_size=(2, 2)))
69
70 model.add(Flatten()) # 二次元データを一次元データに変換
71 model.add(Dense(512)) # 全結合層
72 model.add(Activation('relu'))
73 model.add(Dropout(0.5))
74 model.add(Dense(4))      # クラスは4個
75 model.add(Activation('softmax')) # ソフトマックス関数
76
77 # モデルをコンパイル
78 model.compile(loss="categorical_crossentropy", # 損失関数 交差エントロピー誤差
79                 optimizer='adam', # 最適化アルゴリズム ADAM
80                 metrics=["accuracy", "Precision", "Recall", "F1Score"]) # 評価指標 正答率、適合率、再現率、F1Score
81 es_cb=keras.callbacks.EarlyStopping(monitor='val_loss', mode='auto')# EarlyStopping
82
83 # 学習を実行
84 history=model.fit(X_train, y_train, epochs=200, validation_data = (X_test, y_test), callbacks=
85     es_cb)
86
87 # テストデータを予測
88 predict_prob=model.predict(X_test)
89 predict_classes=np.argmax(predict_prob, axis=1)
90
91 # 混合行列の作成
92 mg_df = pd.DataFrame({'predict': predict_classes,
93                       'class': np.argmax(y_test, axis=1)})
94 pd.crosstab(mg_df['class'], mg_df['predict'])
95
96 # 評価指標を表示
97 score = model.evaluate(X_test,y_test,verbose=1)
98 print('Test_Loss:',score[0])
99 print('Test_Accuracy:', score[1])
100 print('Test_Precision:', score[2])
101 print('Test_Recall:', score[3])
102 print('Test_F1Score:', score[4])
103 pd.crosstab(mg_df['class'], mg_df['predict'])

```

## 付録B 原稿非掲載データ

### B.1 実験結果

例えばここに本文中に載せられなかった実験結果などを載せる。