2023 年度（令和 5 年度）

創造工学セミナーII　Final Report

人物認識による
人数カウントシステムに関する研究

研究メンバー

S520005　安藤　蓉美

S520012　ヴィラカンパ　レイ

指導教員

金丸　隆志　教授

所属研究室

知能機械研究室

目次

第 1 章　緒論　（安藤）

1.1　研究背景

本節では、人物認識による人数カウントシステムの構築を目指した研究背景について説明する。

1.1.1　混雑回避の傾向と混雑可視化の有効性

近年、新型コロナウイルス感染症によるパンデミックの影響から、外出時に混雑回避を意識して行動する人が多い傾向がある。新型コロナウイルス感染症に関する世論調査の一つとして NHK 放送文化研究所では、全国の 18 歳以上の 3,600 人を対象に「実践している感染対策」について調査を行った[1]。調査結果より、2022 年においては「密閉・密集・密接の回避」と回答した人が 69%、「人との距離をとる」と回答した人は 61%であった（図 1）。つまり、6 割以上の人が混雑状況を回避するよう意識して行動していることになる。

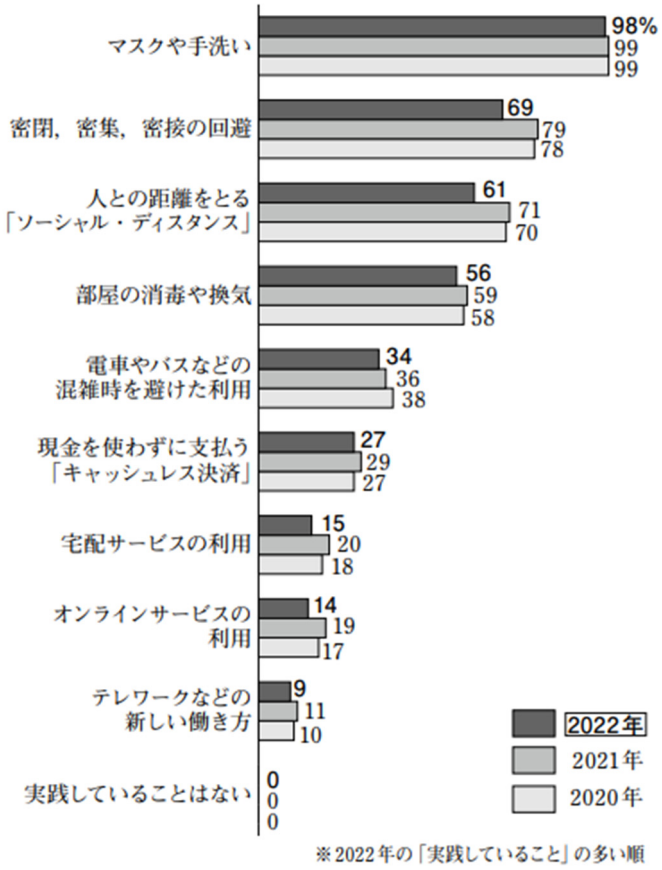


図 1　実践している感染対策[1]

この傾向に関連し、混雑が生じやすい交通機関や複合商業施設などでは、混雑状況を可視化する情報提供サービスが行われている。例えば、Yahoo! JAPAN により提供されている「Yahoo!乗換案内アプリ」では、電車のルート検索以外にも電車内の混雑予報の情報提供を行っている[2]。また、JR 東京駅構内にある商業施設グランスタ東京では、各店舗にリアルタイム空席情報配信サービス「VACAN」と、web 整理券サービス「VACAN Noline」を導入し、デジタルサイネージなどで混雑状況を発信している[3]。

また、東京都は混雑状況を可視化する情報提供サービスの有効性の検証実験を行った[4]。この実験では、リアルタイムに混雑度を可視化し、混雑回避ルート情報を Web アプリで提供することで、混雑回避を促すことが可能かを調べている。その結果、混雑回避ルート情報を利用した人の 34%は「回避できた」と回答し、58%は「どちらかというと回避できた」と回答した（図 2)。このことから、リアルタイムに混雑状況を可視化し、混雑回避ルートを提案することで、一定数の利用者に混雑回避を促すことが可能であると結論づけている。

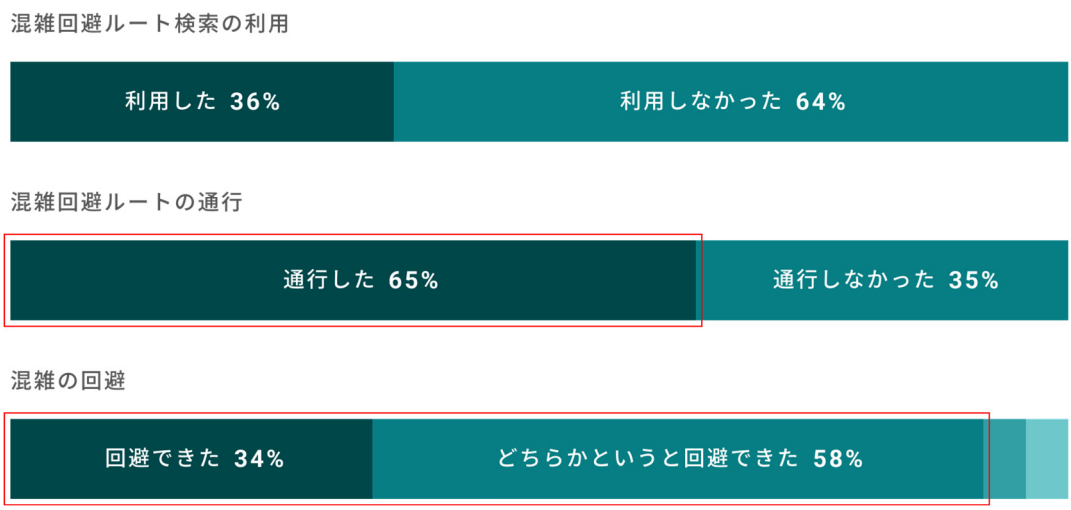


図 2　Web アプリ利用者のアンケート調査結果[4]

以上より、混雑状況をリアルタイムに可視化することは、混雑回避を促し、利用者の利便性や快適性を向上する可能性があると我々は考えた。

1.1.2　複合商業施設における人流データの活用

混雑状況を把握する際、利用者数などの人流データが用いられる。国土交通省によると、人流データとは「人の集積・通過や移動の履歴を計測した値および計測した値をもとに推計・加工した人の動きに関するデータ」のことを指している[5]。人流データの調査対象の1つに、カウント調査がある。カウント調査とは、店舗や施設、駅の出入口など特定の地点を、何人通過したか把握する通行量データを調査することを指している。

複合商業施設では、運営の維持・向上に向けたマーケティングリサーチの一つとしてカウント調査が行われている[6]。カウント調査を行うことで、利用目的や利用時間による利用者数が分かり、商品やサービスの改善・向上に繋げられる。また、複合商業施設は休日に混雑することが多いため、混雑状況を可視化することで混雑回避・緩和を促し、施設の快適性を向上させることができる。

以上から、我々は、複合商業施設における各店舗の利用者数を計測し、混雑状況を示すことに着目した。

1.1.3　カメラを用いた人数カウント

カウント調査の手法として、人による目視やセンサ、カメラ画像、GPS、Wi-Fi などを用いた方法がある[5]。屋内且つ、狭い範囲で正確な値を取得したい場合、カメラ、センサが有効とされている。

そこで、外的環境に比較的左右されないカメラを用いて、店舗の利用者数をリアルタイムにカウントする人数カウントシステムの構築を目指すことにした。

1.2　先行研究

本節では、カメラ画像を用いた人数カウントシステムの先行研究を紹介する。

1.2.1　Real-Time People Counting System for Customer Movement Analysis

Cho らは、カメラにより店舗の利用者数をリアルタイムに計測する、リアルタイム人数カウントシステムを提案した（図 3）[7]。提案されたシステムは、人が映っていない背景画像を予めモデル化し、モデル化した背景画像と各時刻の入力画像との輝度差により人を検出する。その後、人の動作を推定し、フロー解析することで、計算量を軽くし、リアルタイムに出入りする人のカウントを実現している。

実験では、実店舗に類似した 4 つの箇所で人数カウントシステムを実行し、システムの精度評価と計算量の比較評価を行っていた。比較対象として、flow mosaicking（FM）と directional people counter（DPC）、counting people crossing a line（CPC）の人数カウントシステムを用意している。

結果として、他の 3 つのシステムに比べ、短い処理時間で高い精度を出すシステムであると示されていた。また、計算量を軽くしたことで、入退店者数をリアルタイムに推定することを実現していた。

この論文では、Raspberry Pi のような処理能力の低いコンピュータでもリアルタイムなカウントを可能にするため、認識にディープラーニングを用いた計算量の多い手法を用いていない。背景上で動くものを認識しているので、人間以外の動体も認識してしまうと思われる。そこで、人間であることを正しく認識するシステムが必要ではないかと考えた。

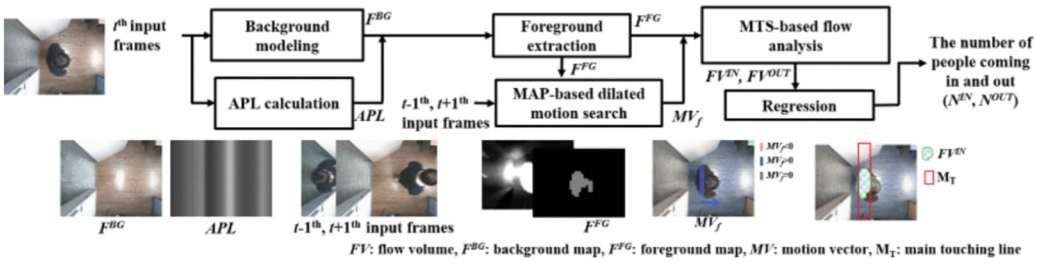


図 3　Cho らにより提案された人数カウントシステムの構造[7]

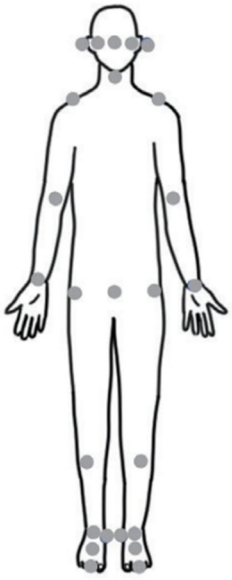
1.2.2　OpenPose を用いた歩行者行動特性の計測システムの開発

人間をディープラーニングを用いた手法により認識するカウントシステムとして、田中氏らによる OpenPose を用いた歩行者行動特性の計測システムがある[8]。OpenPose を用いると人の関節情報を取得できるので、その情報から人の中心座標を求め追跡し、移動方向別の人数計測を実現している（図 4)。また、足元の関節情報から中心座標を求め描画することで、人の軌跡計測を実現している。

開発したシステムの評価実験では、比較的人通りの少ない通路を歩く歩行者をカメラで撮影し、撮影した動画をシステムに入力するという方法で行っていた。

結果として、計測制度は 98%以上であり、高い精度を示していた。しかし、カメラによる計測の弱点である人同士の重なりによる計測失敗が課題として挙げられている。これは、人同士が重なってしまうことで、関節情報が取得できないことが原因だと述べている。

この研究には、映像 1 フレームあたりにかかる処理時間は記されていない。しかし、第 3 章で紹介する我々の実験では、OpenPose による人物の認識は、GPU 搭載 PC を用いたとしても毎秒 2 回程度しか実行できず、リアルタイム処理は難しいのでないかと考えられる。



関節	信頼度	関節	信頼度
Nose	0.20	L Knee	0.08
Neck	0.05	L Ankle	0.74
R Shoulder	0.10	R Eye	0.90
R Elbow	0.18	L Eye	0.90
R Wrist	0.32	R Ear	0.45
L Shoulder	1.00	L Ear	0.73
L Elbow	0.21	L Big Toe	0.79
L Wrist	0.20	L Small Toe	0.57
Mid Hip	0.11	L Heel	0.48
R Hip	0.48	R Big Toe	0.33
R Knee	0.44	R Small Toe	0.80
R Ankle	0.97	R Heel	0.36
L Hip	0.39		

図 4　OpenPose の関節情報と信頼度[8]

1.3　研究目的

1.1 節と 1.2 節より、我々は、以下の条件における人数カウントシステムの構築を目指すことにした。

1. 人物をディープラーニングを用いた手法で認識すること。
2. カメラを用いて、リアルタイムに人物認識ができること。
3. 店舗に出入りする人を、移動方向別にカウントできること（図 5）。
4. 店舗内にいる人数から、混雑状況を示せること（図 6）。

以上の条件から、店舗を出入りする人をリアルタイムにカウントし、店舗内の混雑状況を示す人数カウントシステムの構築を行う。

条件 1 と 2 に対しては、複数人を認識可能な複数人認識システムを用意し、性能比較実験を行う。各システムの認識精度と FPS を比較し、最適な複数人認識システムの選択を目指す。詳細は、第 2 章と第 3 章で示す。

条件 3 に対しては、一般向けに公開されている、既存の人数カウントシステムを用意し、その仕組みを参考にして、追跡機能とカウント機能の構築を目指す。詳細は、第 4 章と第 5 章で示す。

条件 4 に対しては、出入りした人を計測した結果から、3 段階の混雑度を示す評価システムの構築を目指す。こちらの詳細も、第 4 章と第 5 章で示す。

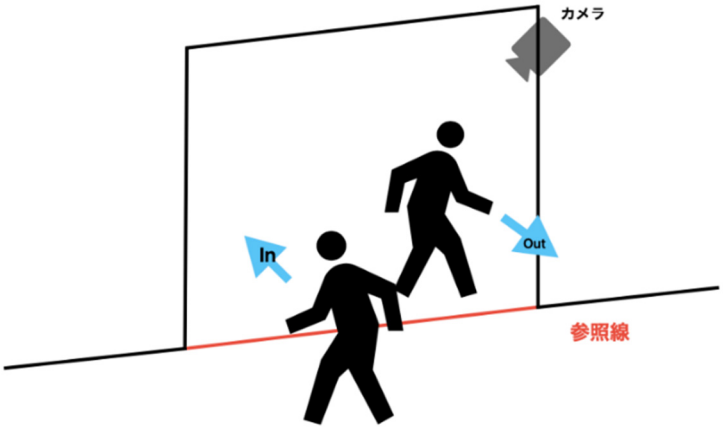


図 5　移動方向別のカウントイメージ図

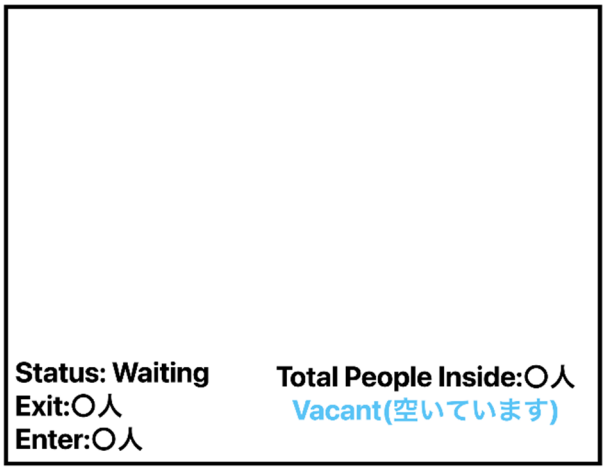


図 6　混雑状況の可視化イメージ図

第2章　機械学習ライブラリ PyTorch と複数人認識システム （ヴィラカンパ）

人数カウントシステムを構築するにあたり、カメラで複数人を認識する機能が必要である。複数人を認識することが可能な複数人認識には、ディープラーニングを用いたものがいくつか提案されており、その実行には機械学習ライブラリが必要となる。今回採用した PyTorch は、ディープラーニングを利用した機械学習モデルを開発・実行するための機能を備えたライブラリであり、画像認識や言語処理などの応用でよく使われる。また、他の機械学習ライブラリとは比べ、近年主流となっている。

本章では、本研究で用いた PyTorch と複数人認識システムについて説明する。

2.1　機械学習ライブラリ PyTorch

PyTorch は、Facebook(現在 META)社の AI リサーチ(FAIR)が 2019 年に開発したオープンソースのディープラーニングライブラリである[9]。その柔軟性、動的な計算グラフ、使いやすさから、機械学習分野で広く採用されている。

PyTorch には次のような特徴があるため、本研究で使用する。

＜PyTorch の主な特徴[10]＞

1. PyTorch は基本的な構成要素である Tensor を備えている。Tensor は数値計算に使用でき、GPU 加速と互換性がある。GPU 加速用の CUDA より、CPU と GPU 間のデータ転送を処理が容易になる[11]。

2. PyTorch は動的計算グラフ(Dynamic Computation Graph)を使用し、デバッグや直感的なモデル構築を容易になる[11]。

3. 即実行(Eager Execution)を採用しており、操作が呼び出されると即座に実行される。Python のコードと実際の計算が密接に関連しているため、柔軟かつ直感的、そしてユーザーにとって操作しやすいものとなっている[12]。

また、本研究で使用した PyTorch ライブラリの仕様を表 1 に示す。

表 1　PyTorch の仕様

ライブラリ名	バージョン
pytorch	2.0.1
pytorch-cuda	11.7
pytorch-mutex	1.0
torchaudio	2.0.2
torchvision	0.15.2

2.2　複数人認識システム

骨格推定を用いて人物を検出するシステムは数多く存在し、その 1 つが Google が開発したオープンソースの骨格推定システム MediaPipe である。しかし、MediaPipe は一度に一人しか検出を行えないため、OpenPose などのように一度に複数人の人を検出できる複数人認識システムを用意した。

本節では、4 つの複数人認識システム OpenPose、Lightweight OpenPose、YOLO ver.7、PoseNet、について説明する。

2.2.1　OpenPose（pytorch-openpose）

OpenPose は、2017 年に Carnegie Mellon 大学の Perceptual Computing Lab によって開発された（図 7）[13]。2 次元複数人物ポーズ推定（Human Pose Estimation）が可能な最初のオープンソースライブラリである。ディープラーニングを用いて、静止画や動画に映る人の関節点を検出し、人のポーズを推定することができる。

OpenPose を PyTorch で実装して公開されたものが、「pytorch-openpose[14]」であり、Hzzone によるものである。

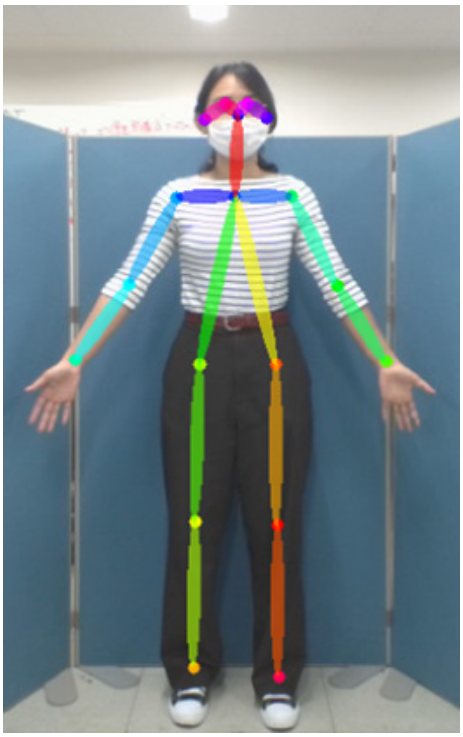


図 7　OpenPose[13][14]による骨格推定

2.2.2　Lightweight OpenPose（lightweight-human-pose-estimation.pytorch）

Lightweight OpenPose は 2016 年に Daniil Osokin により開発された、OpenPose の軽量版である（図 8）[15]。CPU 上でリアルタイムポーズ推定を実現するために、最小限の精度損失で最適化された OpenPose 実装モデルである。

OpenPose と大きな違いは、使用しているニューラルネットワークの種類である。Lightweight OpenPose は MobileNet を用いているのに対して、OpenPose は VGG-19 を使用している。

Lightweight OpenPose を PyTorch で実装して公開されたものが、「lightweight-human-pose-estimation.pytorch[16]」であり、Daniil-Osokin によるものである。

図 8　Lightweight OpenPose[15][16]による骨格推定

2.2.3　YOLOversion 7（yolov7-pose-estimation）

YOLO（You-Only-Look-Once）は、バウンディングボックスを使用し、リアルタイムで画像内にある様々な物体を検出・認識する高速な機械学習アルゴリズムである[17]。YOLOには多くのバージョンがあり、今回使用したのは2022年7月にリリースされたYOLO ver.7である（図 9）。前バージョンよりも高速で、より正確に物体検出が可能となる。

YOLO ver.7は、top-down手法を使用することでポーズ推定を行う。この手法では、物体検出器であるYOLOを用いてから、人物の骨格点の位置を推定し、各人物のポーズを計測する。

YOLO ver.7をPyTorchで実装して公開されたものが、「yolov7-pose-estimation [18]」であり、RizwanMunawarによるものである。

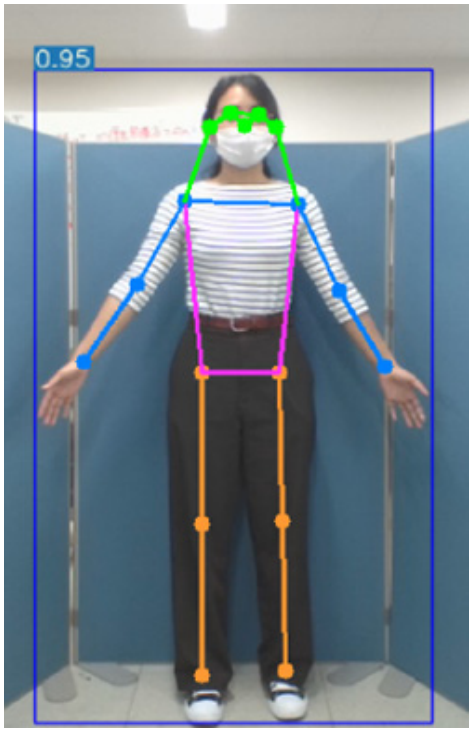


図 9　YOLO ver.7[17][18]による骨格推定

2.3.4　PoseNet（posenet-pytorch）

PoseNet は 2017 年に、Google 社により開発した機械学習モデルである（図 10）[19]。TensorFlow 機械学習プラットフォーム上に構築され、OpenPose と異なり、CPU で動作することが可能であるように設計されている。そのため、ブラウザ上でリアルタイム人物ポーズ推定の実装が可能となる。また、深層畳み込みニューラルネットワーク（CNN）を使用していることと、注釈付き画像や動画などの人間のポーズ情報を大規模に取り扱うため、さまざまな場面における骨格位置を正確に推定することが可能とする。

PoseNet を PyTorch で実装して公開されたものが、「posenet-pytorch [20]」であり、rwightman によるものである。

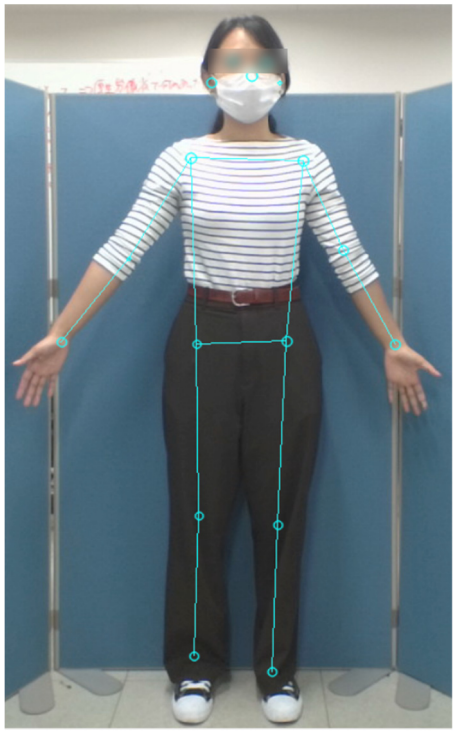


図 10 PoseNet[19][20]による骨格推定

第 3 章　複数人認識システムの性能比較実験

本章では、第 2 章で紹介した 4 つの複数人認識システムの性能比較実験について説明する。

3.1　実験目的　（安藤）

本実験では、複数人認識システム OpenPose、Lightweight OpenPose、YOLOv7、PoseNet の性能比較実験を行う。人数カウントシステムの構築を目指すにあたり、より速く正確に複数人を認識する機能が重要になると考えた。そこで、本実験では、4 つの複数人認識システムの FPS と認識精度を比較し、人数カウントシステムに最適な複数人認識システムを選択することを目指した。

3.2　使用機器　（ヴィラカンパ）

本節では、研究で使用した機器の説明をする。

3.2.1　GPU 搭載 PC

システム動作用パソコンとして、GPU 搭載の iiyama 社製 PC を使用した（図 11）。詳しい仕様を表 2 に示す。

図 11　使用した GPU 搭載 PC

表 2　GPU 搭載 PC の仕様

型番	iiyama LEVEL-15FR171-i7-UASX
OS	Windows 11 Pro
CPU	Intel Core i7-12700H
GPU	NVIDIA GeForce RTX 3070 Ti
メモリ	16GB

3.2.2　使用したカメラ

図 12 及び表 3 は、本研究で使用した web カメラの詳細である。

図 12　使用した web カメラ

表 3　web カメラの仕様

型番	Logicool C920 n
最大解像度	1080 p / 30 fps・720 p / 30 fps
カメラ画素数	3MP
フォーカスタイプ	オートフォーカス
対角視野（dFoV）	78°

3.2.3　カメラスタンド

図 13 は、本研究で使用したカメラスタンドである。これは、ハクバ製三脚 HK-834B と SLIK 製自撮り棒セルフィーポッド 940 を組み合わせたものである。本研究では、認識領域を見やすいように、カメラを一定の高さまで上げるためこのカメラスタンドを使用した。

図 13　使用したカメラスタンド

3.3　実験環境　（ヴィラカンパ）

実験は、八王子キャンパスの 4 号館 8 階廊下で行う（図 14)。図の緑線とピンク線の位置に養生テープを貼って実験を行う。中央の緑線が人の通過を判定するための参照線であり、そこから 50cm ごとにさらにテープを貼る。

図 14　八王子キャンパス 8 階の廊下

図 15 は、システムの設置図である。認識領域全体が映るよう、カメラを高さ 1.8m、傾き 10° で設置した。

図 15　システム設置イメージ図

図 16 は、実験環境のイメージ図である。参照線より前後 1.5ｍを認識領域として設定した。図 14 で線が引かれた範囲が認識領域ということになる。この認識領域内では、第 2 章で紹介した 4 つの複数人認識システムの認識率を検証した。

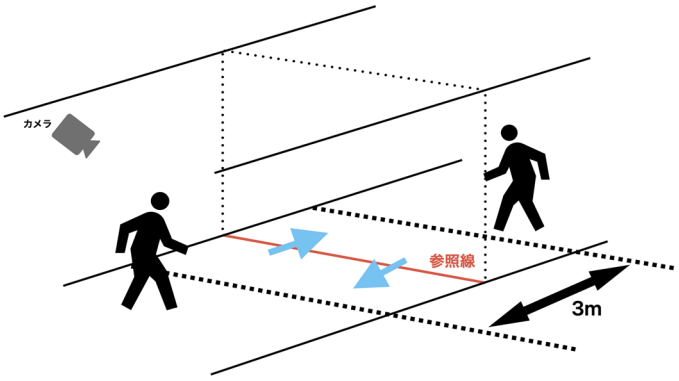


図 16　実験環境イメージ図

3.4　評価方法　（ヴィラカンパ）

本節では、本実験の評価方法について説明する。

3.4.1　FPS

FPS（Frame per Second）は、1 秒間の動画が何枚の画像で構成されているかを表す単位である。FPS が高いほど動画がよりなめらかになり、認識対象人物を追跡しやすくなる。

本実験では、構築する人数カウントシステムをリアルタイムで実行するために第 2 章で紹介した 4 つの複数人認識システムの FPS を検証する。

下記のプログラムにより、FPS を計測した（コード 1）。

```
total_fps = 0
frame_count = 0

while True:
   start_time = time.time()
   #FPS を計測したい処理
   end_time = time.time()
   fps = 1 / (end_time - start_time)
   total_fps += fps
   frame_count += 1

      print(total_fps/frame_count)
```

コード 1　FPS を表示するコード

3.4.2　認識精度

認識精度とは、歩いている人物を認識するシステムの正確さを表す。

人数カウントシステムを実行する際に、正しく人物認識を行なっている複数人認識システムを利用することで、カウントの失敗率を低くすることに繋がると考え、4 つの複数人認識システムの認識精度を検証する。

4 つの複数人認識システムの認識精度を評価するために、人の骨格を頭、肩、胴体、手足の 11 ヶ所に分けた（図 17)。そして、図 14 で示した各線上で、全ての骨格が表示された場合に認識率 100%として評価した。認識率は、次式で計算する。

$$\text{認識率} = \frac{\text{正しく認識された骨格数}}{11} \times 100\ (\%)$$

また、n 人で実験を行った場合の認識率は次式で計算する。

$$\text{認識率} = \frac{\text{正しく認識された骨格数}}{11n} \times 100\ (\%)$$

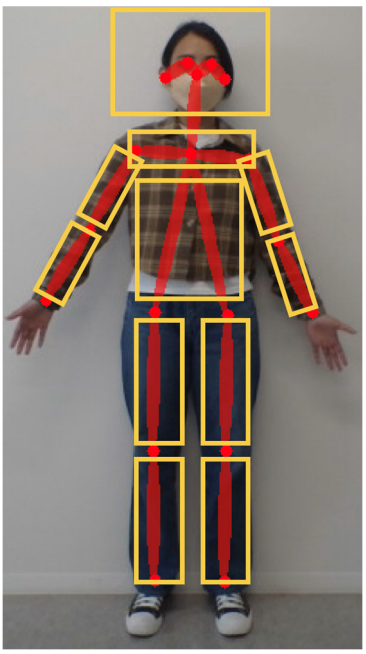


図 17　認識精度の評価イメージ図

3.5　実験方法　（安藤）

本節では、本実験の実験方法を説明する。

4 つの複数人認識システムの FPS と認識精度の比較実験は、以下の 3 つの手順で行った。

＜実験手順＞

1．カメラの前を一方向に通過する人数を設定する。（1 人、2 人、5 人）
2．各システムに、認識領域内で自由に人が歩いている姿を認識させる。
3．1 人の場合は 16 データ、2 人の場合は 30 データ、5 人の場合は 10 データの動画を保存する。
4．保存した動画を用いて FPS と認識精度を求める。

また、手順 1 では、通過人数別に以下の条件を設定した。

2 人の場合では、人の肩同士の間隔を 1m、0.6m、0.2m と設定し、10 データずつの動画を保存する（図 18）。

5 人の場合では、認識領域内に 5 人全員が入る間隔と設定する。

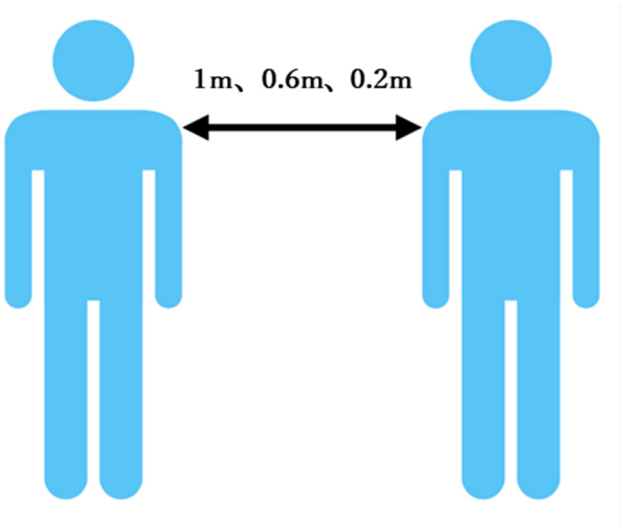


図 18　肩同士の間隔のイメージ図

3.6　実験結果　（安藤）

本節では、4 つの複数人認識システムの FPS と認識精度を比較実験した結果について説明する。

3.6.1　FPS

表 4 は、複数人認識システムの FPS を通過人数別に表している。表から、通過人数が変化しても、複数人認識システムの FPS の値はあまり変化しないことがわかった。

また、4 つの複数人認識システムの FPS は、Lightweight OpenPose が最も高い結果となった。

表 4　通過人数別における複数人認識システムの FPS

	1 人の場合	2 人の場合	5 人の場合
OpenPose	2.3	2.1	2.1
Lightweight OpenPose	46	46.3	49.4
YOLOv7	2	2	1.9
PoseNet	31.5	36	31.6

3.6.2　1 人の場合の認識精度

本項では、認識領域内を 1 人が通過した場合における各システムの認識精度を比較していく。

図 19 は、1 人が認識領域内における各システムの認識率を示している。図 20 は、カメラから見た認識領域の一番手前を 0m として、距離別の認識率を示している。図 20 を距離に関して平均したのが図 19 ということになる。

図 19 より、1 人の場合では、OpenPose と Lightweight OpenPose、YOLOv7 の認識率は 100%に近く、認識精度が高いことがわかった。一方、PoseNet は他の 3 つのシステムに比べ認識率が低いことがわかった。

図 20 より、OpenPose と Lightweight OpenPose、YOLOv7 の認識率は、カメラからの距離が離れたとしても、認識が安定していることがわかった。一方、PoseNet は距離に関係なく認識が不安定になっていることがわかった。これは、人を認識する時、肩や腕などの骨格表示が混合し、乱雑になったことが原因である。

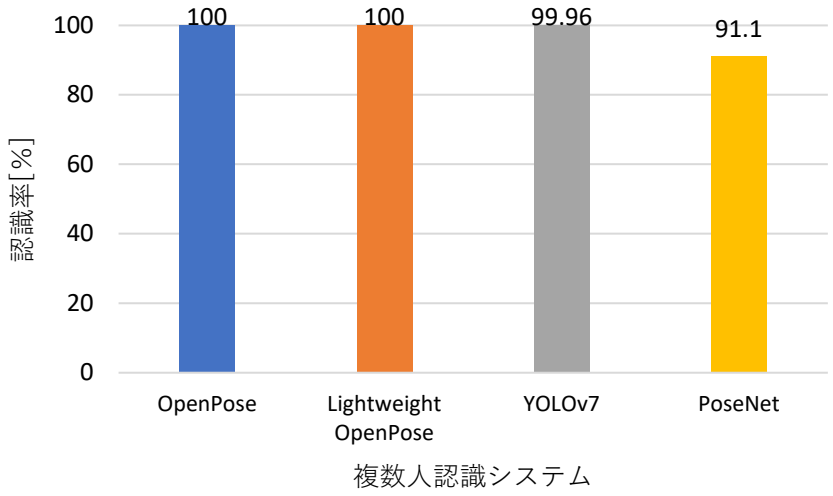


図 19　1 人の場合における各システムの認識率

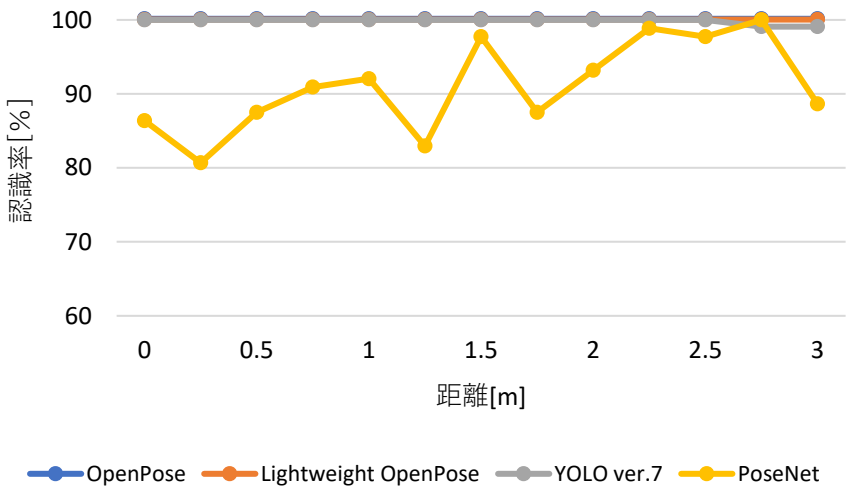


図 20　1 人の場合における距離別の認識率

3.6.3　2 人の場合の認識精度

本項では、複数人が通る場合に起こりうる重なりなどの状況における認識精度を比較するため、まず、2 人が認識領域を通過した場合における認識精度を比較していく。また、人の肩同士の間隔を 1m、0.6m、0.2m と設定することで、間隔が小さくなるにつれて生じる重なりにより変化する各システムの認識精度を比較していく。

図 21、図 22、図 25 は、2 人が認識領域内を通過した場合における各システムの認識率を、人の肩同士の間隔別に示している。図 23、図 24、図 26 は、カメラから見た認識領域の一番手前を 0m として、人の肩同士の間隔が 1m、0.6m、0.2m の場合における距離別の認識率を示している。

図 21 及び図 22 より、人の肩同士の間隔が 1m 及び 0.6m の場合では、2 人の間隔が広いため重なりが生じにくく、3.6.2 項で述べた 1 人の場合の認識精度とあまり変わらない結果となることがわかった。ただし、図 23 及び図 24 より、1 人の場合の認識率に比べ、OpenPose と Lightweight OpenPose は認識率が低くなっていることがわかった。これは、後で述べるように、カメラに背を向けて歩いた場合に、人の前後が反転して骨格表示されることが原因である。

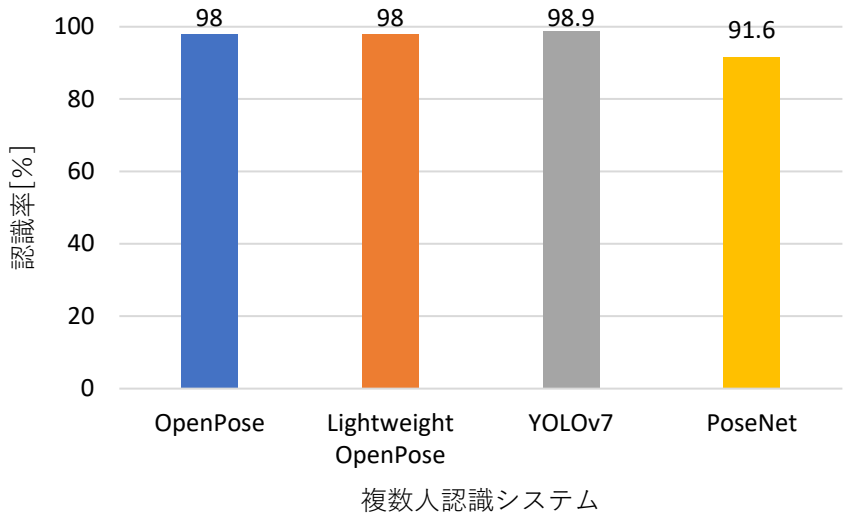


図 21　2 人の場合における各システムの認識率（1m）

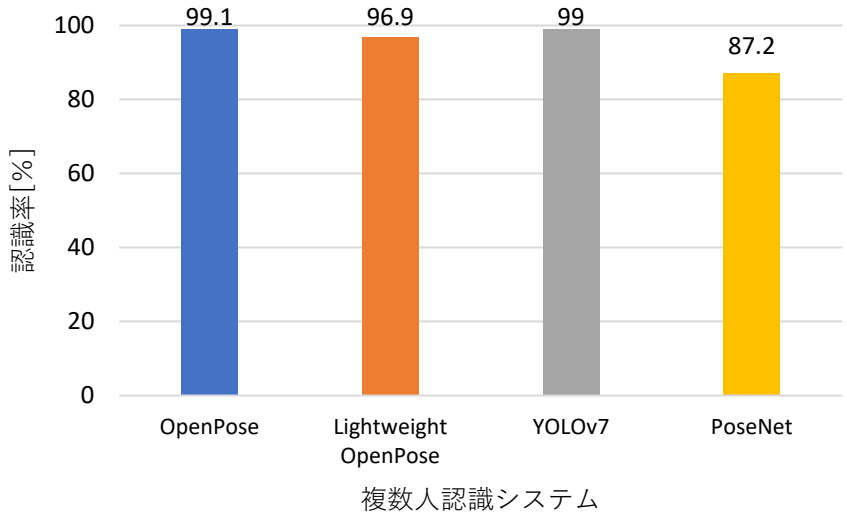


図 22　2 人の場合における各システムの認識率（0.6m）

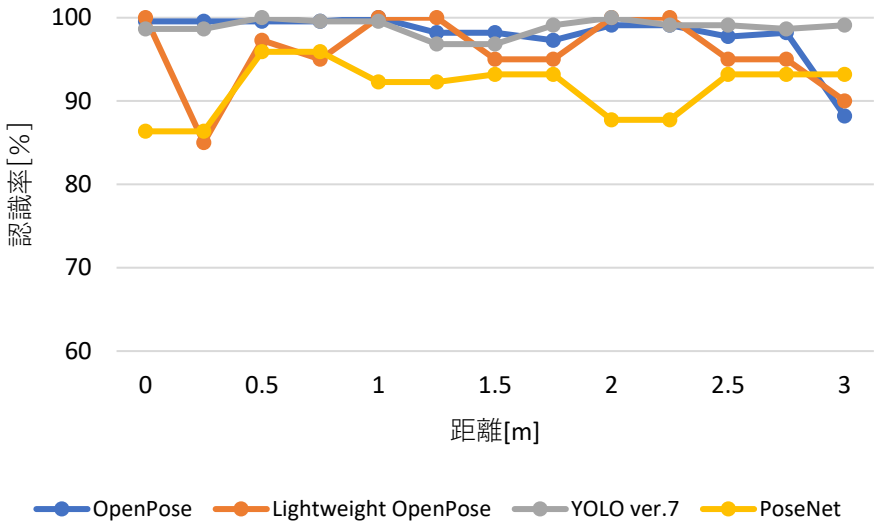


図 23　2 人の場合における距離別の認識率（1m）

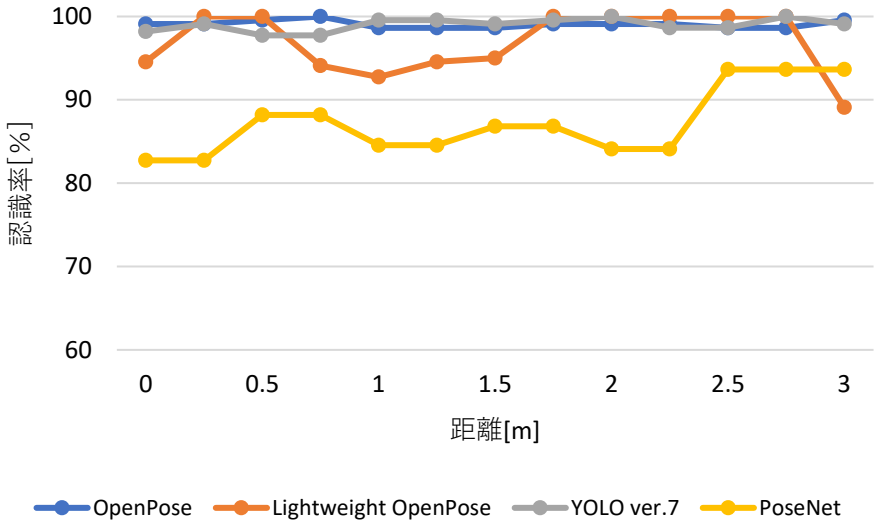


図 24　2 人の場合における距離別の認識率（0.6m）

人の肩同士の間隔が 0.2m の場合（図 25、図 26）では、2 人の間隔が狭くなったため人同士の重なりが生じやすくなり、全体的に認識率が低くなることがわかった。この原因を、システムごとに説明していく。

OpenPose では、認識率が急激に低くなっていることがわかった（図 25）。人同士の重なりに対して、2 人を 1 人として誤認識する場合が多いことが原因である（図 27）。腕同士が近づいた場合でも、頭や腕の認識が混合するなどの誤認識が多く生じていた。つまり、OpenPose は、人同士の重なりが少しでも生じると、認識率が低くなることがわかった。さらに、カメラから遠ざかるにつれて認識率が急激に低くなっているため、遠距離での認識に向かないことがわかった（図 26）。

Lightweight OpenPose も、OpenPose ほどではないが認識率が低くなっていることがわかった（図 26）。これは、カメラに背を向けて歩いた場合に、人の前後が反転して骨格表示されることが原因である（図 28）。

YOLOv7 の認識率は、1 人の場合とあまり変化しない結果となった（図 26）。稀に、2 人を 3 人として誤認識することがあった（図 29）。

PoseNet は、1 人の場合と同様に、骨格表示が不安定であるため認識率が低くなっている（図 30）。

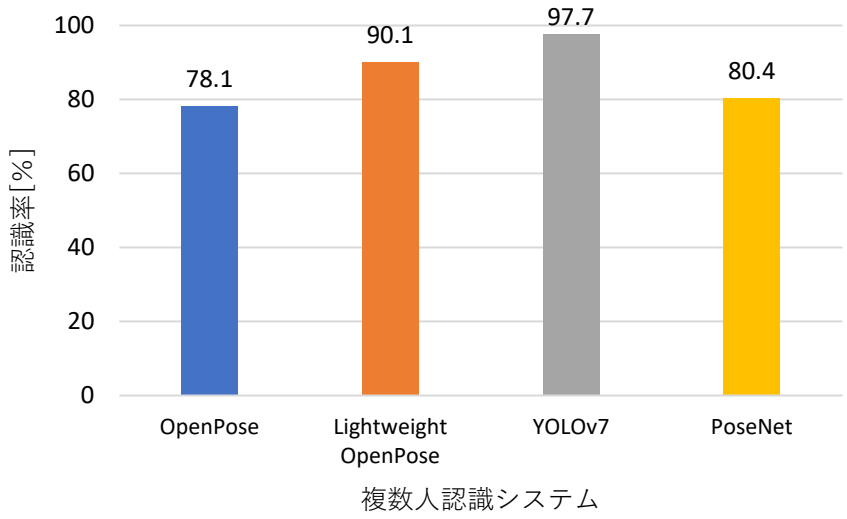


図 25　2 人の場合における各システムの認識率（0.2m）

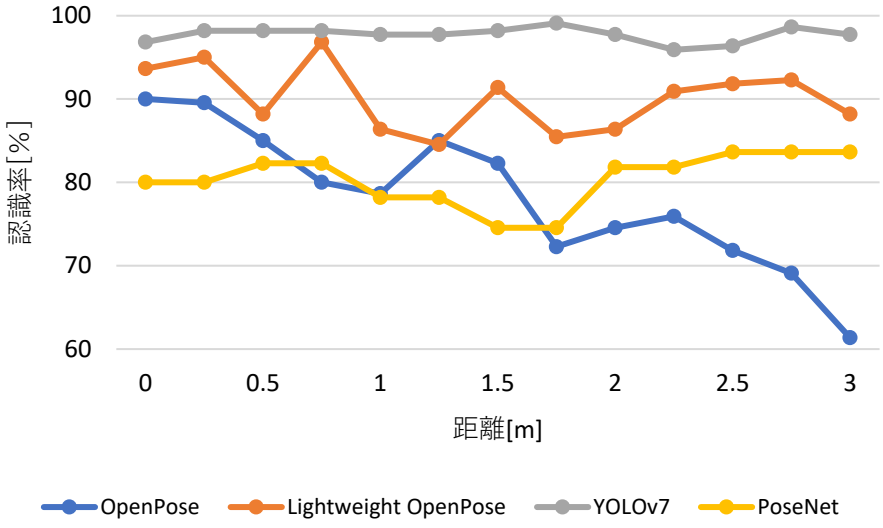


図 26　2 人の場合における距離別の認識率（0.2m）

図 27　OpenPose の認識失敗例（0.2m）

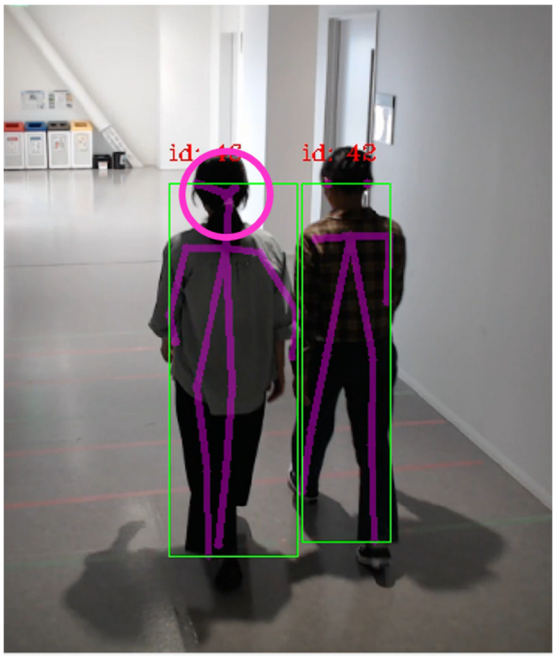


図 28　Lightweight OpenPose の認識失敗例（0.2m）

図 29　YOLOv7 の認識失敗例（0.2m）

図 30　PoseNet の認識失敗例（0.2m）

3.6.4　5 人の場合の認識精度

最後に、認識領域内を 5 人が通過した場合における各システムの認識精度を比較する。

図 31 は、5 人が認識領域内を通過した場合における各システムの認識率を示している。5 人の場合では、2 人の場合に比べ、混雑度が上がったため人同士の重なりが多く生じた。誤認識の傾向は 2 人の場合と同様であったが、頻度が多かったため、全体的に認識率が低くなる結果となった。

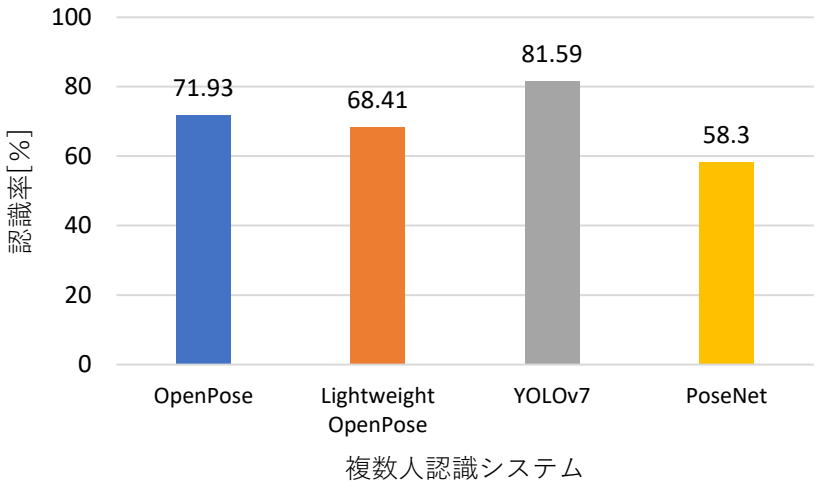


図 31　5 人の場合における各システムの認識率

3.7　複数人認識システムの決定　（安藤）

3.6.1 項より、Lightweight OpenPose は、FPS が最も高いことがわかった。このことから、4 つの複数人認識システムの中で、最もリアルタイムに近い状態で人物認識を行うことが可能であると考えた。また、3.6.2 項～3.6.4 項より、認識精度は YOLOv7 が最も高いことがわかった。認識精度が高いことは、正確に人数をカウントする上で有利になると考えられる。YOLO ver.7 の FPS を高くすることは難しいため、FPS が高く比較的認識精度の高い Lightweight OpenPose が、必要な性能が備わった複数人認識システムであると考えた。

以上より、Lightweight OpenPose を用いて、人数カウントシステム「People Counting with Lightweight OpenPose（以降 PCLWOP とする）」の構築を目指すことにする。

第 4 章　人数カウントシステム PCLWOP の構築　（ヴィラカンパ）

本章では、構築した人数カウントシステム PCLWOP ついて説明する。

4.1　参考人数カウントシステム PCSSD

人数カウントシステム PCLWOP を構築するため、saimj7 が開発した人数カウントシステム「People-Counting-in-Real-Time」（以降 PCSSD とする）[21]の追跡機能とカウント機能を参考にした。そこで、本節では、PCSSD の人物検出方法について説明する。

4.1.1　物体検出器：SSD

参考にしたシステム PCSSD では、人物を検出する機能のために、様々な物体を検出できる SSD（Single Shot MultiBox Detector）という物体検出器を使用している。

SSD とは、2016 年に、Wei Liu 氏らによって開発された物体検出器である[22]。SSD が認識できる物体の種類は学習データにより異なり、VOC2007 を用いた場合はコード 2 に示した 20 種類、COCO を用いた場合は 80 種類である。高い精度と速い処理速度のため、リアルタイムでの物体検出に適している。また、SSD は、複数のステップを必要とする物体検出器とは異なり、1 回の順伝搬(シングル・ショット)で物体検出できる。図 32 は、SSD による物体検出の結果の例である。

```
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
                    "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
                    "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
                    "sofa", "train", "tvmonitor"]
```

コード 2　SSD により検出可能な物体例

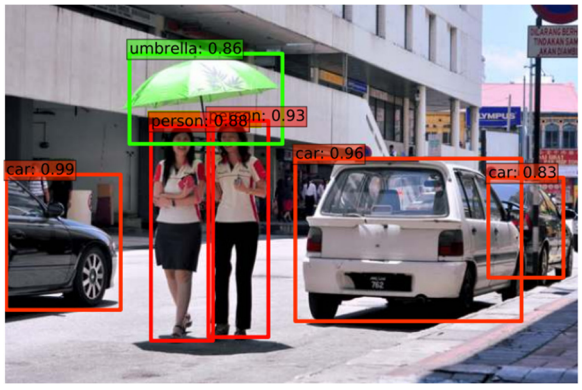


図 32　SSD 物体検出の例[19]

4.1.2　人物検出方法

PCSSD では、SSD により認識可能な物体から、人のみの検出結果を利用することで、人数カウントを行なっている（コード 3)。図 33 は、PCSSD によって、人が検出され、人数カウントが行われている様子である。

```
if CLASSES[idx]　!= "person":
        continue
```

コード 3　人以外の検出結果を除外するコード

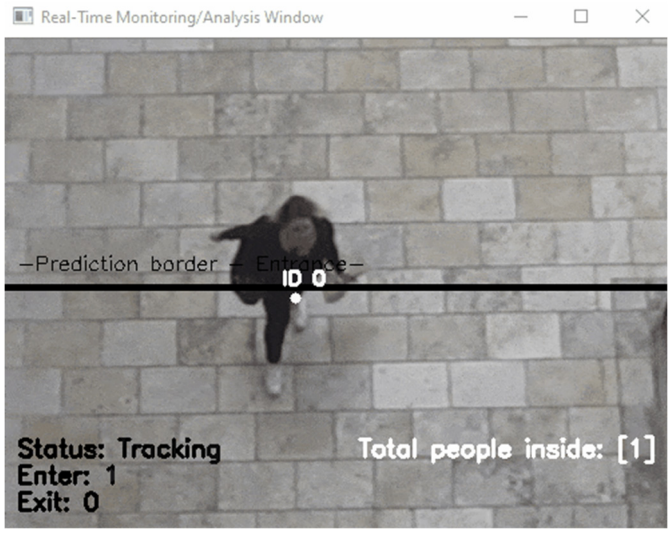


図 33　参考にした人数カウントシステム PCSSD[21]

4.2　LWOP による人数カウントシステム PCLWOP

4.1 節で紹介した SSD とは異なり、本研究で使用する LWOP は人物を検出するにあたり、物体検出器を使用しないことが特徴である。SSD などの物体検出器はバウンディングボックスを用いて人物の大まかな位置を特定し、検出する。一方、LWOP などの人物ポーズ推定モデルは、人物に関連するキーポイントの正確な位置を推定することで、実世界の環境において人物を認識し、追跡することが可能となる[23]。

本研究で人数カウントシステムを構築するにあたり、人物認識器として SSD などの物体検出ではなく、人物ポーズ推定モデルである LWOP を用いる。それにより、人物の認識性能が上がることが期待される。また、第 3 章の実験結果より、リアルタイムでの人数カウントが可能になると考えられる。

4.3　人数カウントシステム PCLWOP の構築流れ

本節では、人数カウントシステム PCLWOP の構築の流れについて説明する。LWOP を使用した人数カウントシステム PCLWOP を構築するにあたり、追跡機能とカウント機能を PCSSD より流用した。

4.3.1　追跡機能の導入

PCLWOP の追跡機能を PCSSD から流用することにした。追跡機能により、一人一人に ID が生成され、移動方向別に人を追跡し、同じ人を再度カウントされないことが実現される。

PCSSD には、「centroidtracker.py」というモジュールがあり、各人物にバウンディングボックス（矩形）を描画し、その中心点(centroid)に ID が順番に生成される。この機能を PCLWOP に導入した（図 34)。

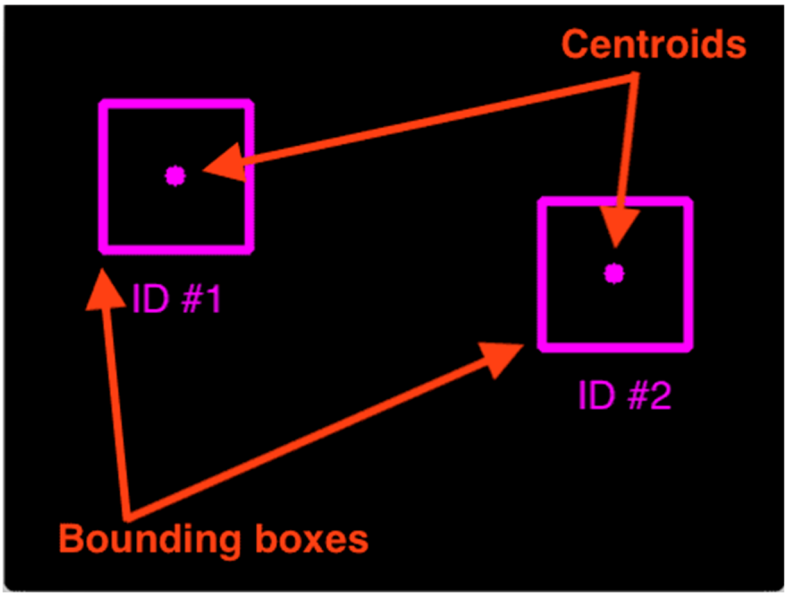


図 34　centroid イメージ図

4.3.2　カウント機能の導入

PCLWOP では、人がフレーム内を上下に移動することでカウントするようにしている（コード 4)。前フレームでの ID の位置と現在のフレームでの ID の位置の差異で人物が y 座標方向を移動した距離「direction」を計測する。

「direction」が「負」すなわち人物がフレーム内の上方向に移動する場合は、「totalUp」が 1 増やされ、人物が室内に入ったとカウントされる。「direction」が「正」すなわち人物がフレーム内の下方向に移動する場合は、「totalDown」が 1 増やされ、人物が室内に出たとカウントされる。また、「to.counted = True」のプログラムで人物が再度カウントされないように処理を行なっている。

以上によって算出された「totalUp」、「totalDown」は、出入りした人数のカウントデータとして利用できる。

```
y = [c[1] for c in to.centroids]
direction = centroid[1] - np.mean(y)
to.centroids.append(centroid)

if not to.counted:
    if direction < 0 and centroid[1] < H // 2:
        totalUp += 1
        move_out.append(totalUp)
        to.counted = True

    elif direction > 0 and centroid[1] > H // 2:
        totalDown += 1
        move_in.append(totalDown)
        to.counted = True
```

コード 4　カウント機能コード

また、「trackableobject.py」というモジュールでは、人物がカウントされたかどうかをチェックし、同じ人物が再度カウントされるのを防いでいる（コード 5)。

```
class TrackableObject:
…（省略）…
                    self.counted = False
…（省略）…
```

コード 5　同じ人を再度カウントしないためのコード

4.3.3　使用したキーポイント

4.3.1 で紹介したバウンディングボックスの描画には、肩と腰の点を使用することにした。Lightweight OpenPose により得られる骨格を図 35 に示す。

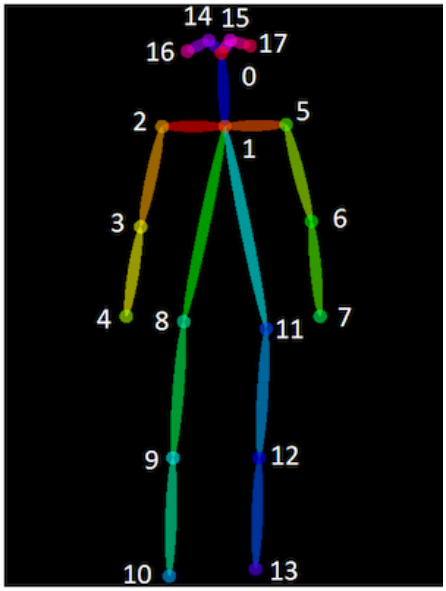


図 35　骨格推定のイメージ図

PCLWOP 上でのバウンディングボックスを描画するためには、点 2 と点 5 の x 座標と点 1 と点 8 の y 座標を使用した（コード 6)。このバウンディングボックスの中心点が、ID の振り付け位置となる。

```
startX = int(pose.keypoints[2][0])
startY = int(pose.keypoints[1][1])
endX = int(pose.keypoints[5][0])
endY = int(pose.keypoints[8][1])
```

コード 6　バウンディングボックスの決定

4.3.4　検出頻度の改善

PCSSD では、毎フレームに人物検出をすると処理が重いため、検出を 30 フレームに 1 回行っている。しかし、LWOP を使用した人物検出の処理は SSD の処理より軽いため、人物検出頻度を毎フレームに実行するようにした（コード 7)。

```
      if totalFrames % args_global["skip_frames"] == 0:

#…省略…

parser.add_argument("-s", "--skip-frames", type=int, default=1, help="# of skip frames between detections")

#…省略…
```

コード 7　検出頻度の改善コード

4.3.5　検出性能の改善

フレーム内に複数の人物が部分的に映る際に、複数人を画面サイズ以上に大きな一つのバウンディングボックスに囲まれることがあり、1 人として認識されることがあった。改良方法として、バウンディングボックスの 4 つの点がフレームの「W(幅)」と「H(高さ)」を超えないようにし、1 人に対して一つのバウンディングボックスが生成されるようにした（コード 8)。

```
if startX>=0 and startX<W and startY>=0 and startY<H and ¥
    endX>=0 and endX<W and endY>=0 and endY<H:
```

コード 8　検出性能の改善コード

また、バウンディングボックスの start（開始）地点と end（終了）地点が反転するのを防ぐため、コード 9 を追加した。

```
if endX < startX:
    tmp = endX
    endX = startX
    startX = tmp

if endY < startY:
    tmp = endY
    endY = startY
    startY = tmp
```

コード 9　バウンディングボックスの反転を防ぐためのコード

4.3.6　評価システムの構築

混雑状況を可視化する 3 段階の評価システムを構築した。図 36〜図 38 は評価システムの 3 段階イメージ図を示す。

1 段階目として、室内にいる人数が少なく、空いている時に表示される（図 36）。2 段階目では、室内の人数が設定された限界人数に近い時に表示される（図 37）。最後に、3 段階目では、室内の人数が制限された限界人数を超えている時に表示される（図 38）。

また、評価システムの警告メッセージの色を段階ごとに青、オレンジ、赤と徐々に誘目性の高い色と設定した。

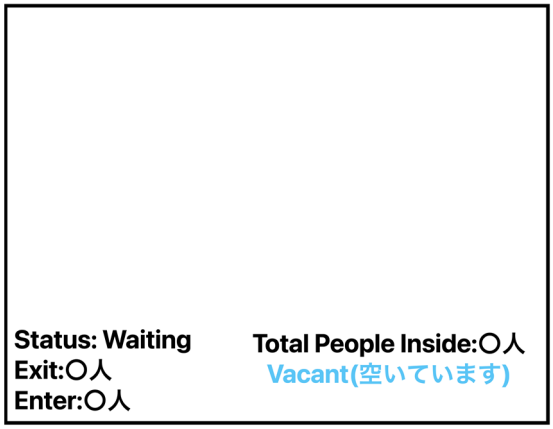


図 36　1 段階目の評価システム

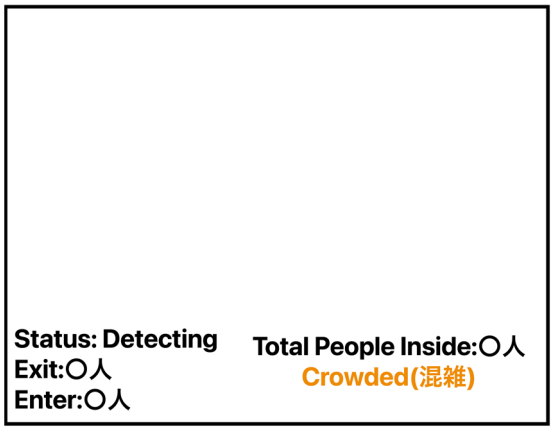


図 37　2 段階目の評価システム

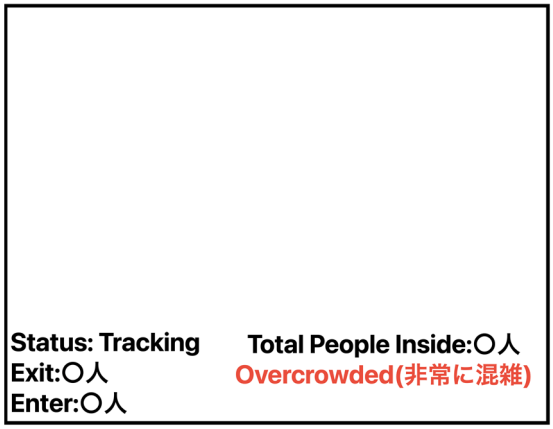


図 38　3 段階目の評価システム

最後に、室内にいる人数を準備し、表示可能にした（コード 10)。

```
total_inside = len(move_out) - len(move_in)
info_total = [
                    ("Total people inside", total_inside)
                    ]

stt1 = "Vacant Inside. Welcome."
stt2 = "Crowded Inside."
stt3 = "Overcrowded Inside."

if total_inside < 3:
    cv2.putText(img, stt1, (265, H - 30), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 255, 0), 2)
elif 3 <= total_inside < 6:
    cv2.putText(img, stt2, (265, H - 30), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 255), 2)
else:
    cv2.putText(img, stt3, (265, H - 30), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)

#…省略…

for (i, (k, v)) in enumerate(info_total):
    text = "{}: {}".format(k, v)
    cv2.putText(img, text, (265, H - ((i * 20) + 60)), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)
```

コード 10　室内にいる人数を準備・表示するコード

4.3.7　構築した人数カウントシステム PCLWOP

図 39 及び図 40 は、構築した人数カウントシステム PCLWOP での認識された人が店舗を入る（Enter）様子、そして出る（Exit）様子を示している。認識された人がフレーム内の上下方向に移動し、参照線を通過する際に、人数がカウントされる。生成された ID が変化することなく、人を正しく追跡することができた。

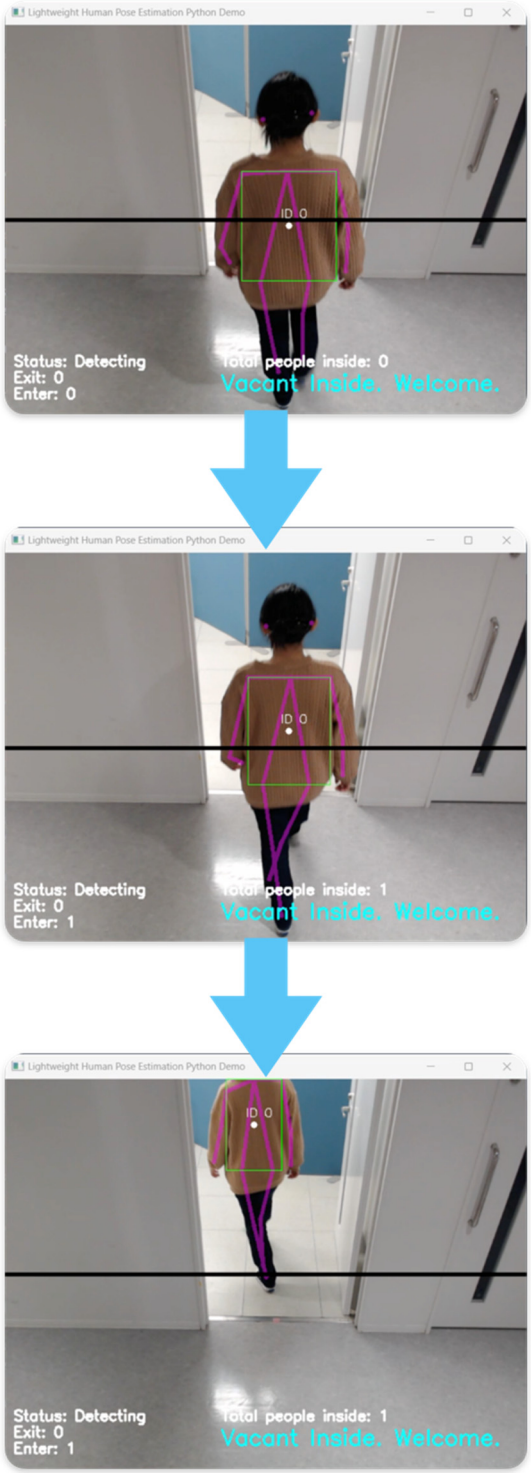


図 39　Enter の様子


Lightweight Human Pose Estimation Python Demo
ID: 1
Status: Detecting
Exit: 0
Enter: 1
Total people inside: 1
Vacant Inside. Welcome.
Lightweight Human Pose Estimation Python Demo
ID: 1
Status: Detecting
Exit: 1
Enter: 1
Total people inside: 0
Vacant Inside. Welcome.
Lightweight Human Pose Estimation Python Demo
ID: 1
Status: Detecting
Exit: 1
Enter: 1
Total people inside: 0
Vacant Inside. Welcome.

図 40　Exit の様子

第 5 章　PCLWOP と PCSSD の比較評価実験　（安藤）

本章では、我々が構築した PCLWOP と参考にしたシステムである PCSSD の比較評価実験について説明する。

5.1　実験目的

本実験では、PCLWOP と PCSSD のカウント成功率を比較し、人数カウントシステムの評価を行った。カウント成功率については、5.4 節で説明する。

5.2　使用機器

本実験では、3.2 節で示した使用機器と同様のものを用いた。

5.3　実験環境

本実験は、工学院大学八王子キャンパス 4 号館 8 階の知能機械研究室出入口付近で実験を行った（図 41)。認識領域は、縦 4m×横 1mとした。カメラの設置角度は 60°とし、高さは 2.3m とした。

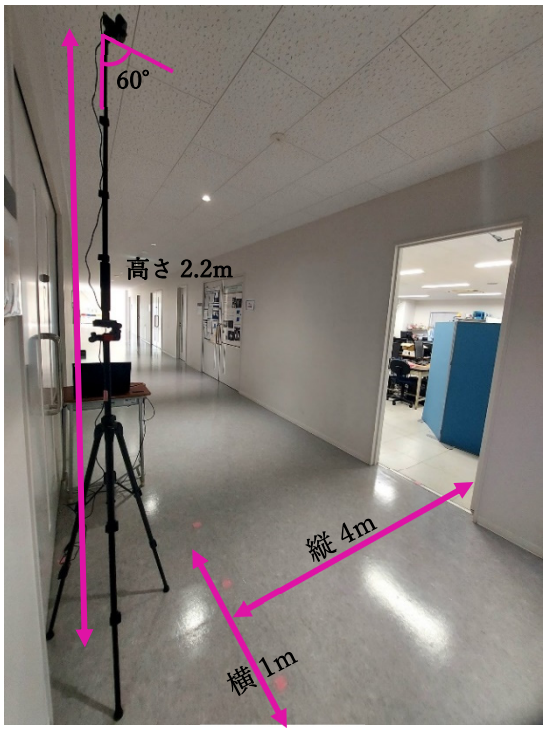


図 41　実験環境イメージ図

5.4　評価方法

本実験では、PCLWOP と PCSSD のカウント成功率を求め、評価を行った。カウント成功率とは、人が参照線を通過した時にカウントが成功した場合の割合を指す。カウント成功率を次式で計算する。

$$\text{カウント成功率} = \frac{\text{Enter 方向と Exit 方向のカウント成功数の合計}}{\text{データ数}} \times 100\ (\%)$$

5.5　実験方法

本節では、本実験の実験方法を説明する。

本実験は、6 人（A～F）で実験を行った。1 人、2 人、3 人、4 人がそれぞれ連続して通過する場合における実験を以下の 4 つの手順で行った。

1．出入口を連続して通過する人数を 1～4 人のどれかに設定する。
2．出入口を通過する人の人数を方向別にカウントさせる。
3．Enter 方向の 60 データと Exit 方向の 60 データ、つまり、全 120 データを作成する。
4．データで得たカウント成功回数から、カウント成功率を求める。

また、手順 1 では、様々な状況に応じて方向別にどの程度正しくカウントされるか調べるため、通過人数別に以下の条件を設定した（表 5)。

表 5　通過人数別の条件

	対象	おおよその歩行速度[m/s]	人同士の前後間隔[s]
1 人の場合	A	0.8	－
2 人の場合	A、B	1	3
3 人の場合	C、D、E	1.3	2.5
4 人の場合	C、D、E、F	1.3	2

加えて、手順 3 では、通過人数別に以下の条件を設定した。

2 人の場合では、同方向に進行する場合を 80 データ、それぞれ逆方向に進行する場合を 40 データ作成する。

3 人の場合では、3 人が同方向に進行する場合を 60 データ、1 人は逆方向に 2 人は同方向に進行する場合を 60 データ作成する。

4 人の場合では、4 人が同方向に進行する場合を 80 データ、1 人は逆方向に 3 人は同方向に進行する場合を 40 データ作成する。

5.6　実験結果

表 6〜表 9 は、移動方向別にカウント成功回数をまとめた表である。Enter 方向と Exit 方向のカウント成功回数から、PCLWOP と PCSSD のカウント成功率を求めた。図 42 は、PCLWOP と PCSSD のカウント成功率を、通過した人数別にまとめたグラフである。

PCLWOP のカウント成功率は、PCSSD より高い結果となった（図 42)。

PCLWOP は、1〜2 人が通過した場合、カウント成功率 90%以上という結果となった。人同士の重なりが少なく、1 人に 1 つ ID を生成し、追跡とカウントを行っていたため、カウント成功率 90%以上となった。しかし、一定数のカウントミスが生じていた。カウントミスの主な原因は、人に ID が振り付けられた瞬間、反対方向にカウントされたためであった。また、通過人数が 3〜4 人の場合では、歩行速度が上がり、人数が増えることで人同士の重なりが生じやすくなったため、カウント成功率は少し低くなり 89.2%という結果になった。カウントミスの傾向は、1〜2 人が通過した場合と同様に、人に ID が振り付けられた瞬間に反対方向へカウントされていた。加えて、参照線付近で重なりが生じ、カウントされないことがあった。

PCSSD のカウント成功率は、PCLWOP に比べ 30〜40%近く下回る結果となった（図 42)。通過人数がどの場合においても、人が通過することで生じた影を人として誤認識することが多発していたため、1 人に対して複数の ID が生成され、複数回カウントされることが多く生じた。また、PCLWOP のカウントミスの傾向と同様に、人に ID が振り付けられた瞬間、反対方向にカウントされることが生じていた。さらに、歩行速度が速く、通過間隔が短くなるにつれて、急激にカウント成功率が低くなった。

表 6　1 人が通過した場合におけるカウント成功回数

	Enter[回]	Exit[回]
PCLWOP	58	55
PCSSD	30	49

表 7　2 人が通過した場合におけるカウント成功回数

	Enter[回]	Exit[回]
PCLWOP	55	57
PCSSD	48	28

表 8　3 人が通過した場合におけるカウント成功回数

	Enter[回]	Exit[回]
PCLWOP	54	53
PCSSD	36	35

表 9　4 人が通過した場合におけるカウント成功回数

	Enter[回]	Exit[回]
PCLWOP	54	53
PCSSD	48	13

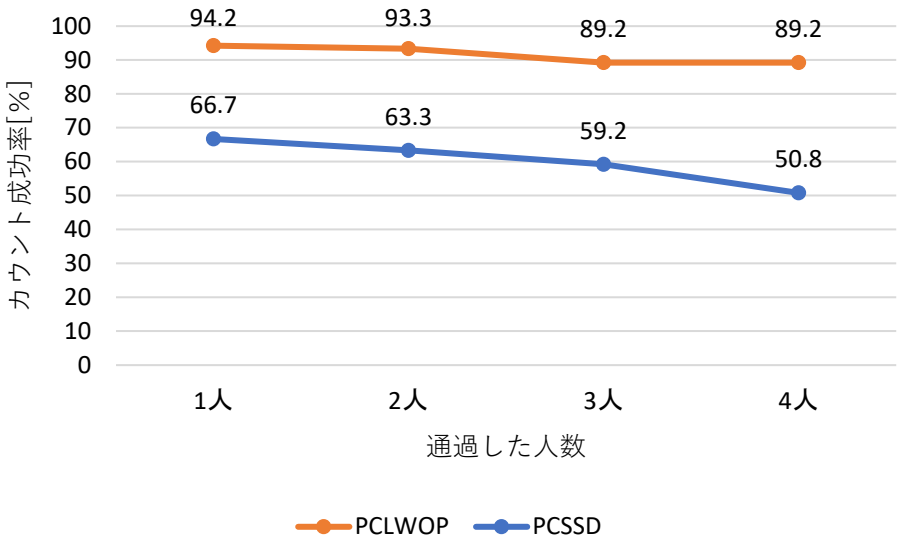


図 42　通過人数別における各システムのカウント成功率

5.7　考察

PCLWOP は、通過人数に関係なく PCSSD よりカウント成功率が高いことがわかった。

PCLWOP の性質として、Lightweight OpenPose により、リアルタイムでカウントすることが出来るという利点がある。また、1 人に一つ ID を生成し、追跡とカウントを行うことができた。しかし、通過人数がどの場合においても、人に ID が振り付けられた瞬間、反対方向にカウントされるという問題があることがわかった。これは、人の ID が生成された後、人の移動方向を判別する時に、反対方向として処理されたことが原因である。

一方、PCSSD の性質として、人が通過することで生じた影を人として認識するという問題があった（図 43）。そのため、1 人が通過した場合においても、複数人として認識されることが多発し、カウント成功率が大幅に低くなった。これは、SSD による人物検出が、人以外も人として誤検出してしまうことが原因であると考えられる。また、PCLWOP と同様に、人に ID が振り付けられた瞬間、反対方向にカウントされることがあった。

両方のシステムに共通する反対方向へのカウントの問題は、第 4 章で紹介した centroidtracker.py や trackableobject.py、およびその周辺プログラムの修正により改善可能であると考えられる。

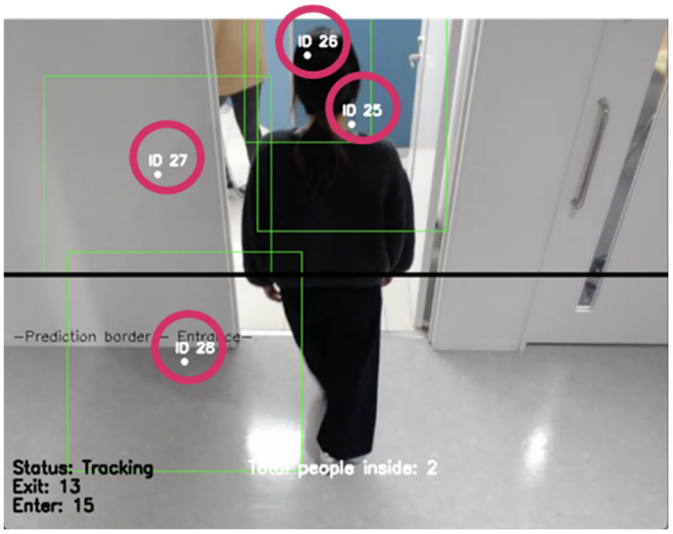


図 43　PCSSD のカウント失敗例

第 6 章　結論　（安藤）

本研究では、下記の条件を目標に、人数カウントシステムの構築を目指してきた。

1. 人物をディープラーニングを用いた手法で認識すること。
2. カメラを用いて、リアルタイムに人物認識ができること。
3. 店舗に出入りする人を、移動方向別にカウントできること。
4. 店舗内にいる人数から、混雑状況を示せること。

以上の条件を満たすよう、我々は、人数カウントシステム PCLWOP の構築を行った。

条件 1 と 2 に対して、ディープラーニングを利用して複数人を同時に認識可能な複数人認識システムを 4 つ用意し、それぞれの認識精度と FPS を比較実験した。FPS と認識精度が比較的高い Lightweight OpenPose を用いることで、リアルタイムに人物認識を行うことができた。

条件 3 に対して、参考にした人数カウントシステム PCSSD から、追跡機能とカウント機能を流用した。カメラに映った人を追跡することに加え、人を移動方向別にカウントすることができた。

条件 4 に対して、空間内にいる人数から、3 段階に混雑度を示す評価システムを構築した。そのため、空間内の混雑状況を示すことができた。

以上より、3 つの条件を満たす人数カウントシステム PCLWOP の構築を実現した。しかし、PCLWOP のカウント成功率は 90%前後という結果であった。5.7 節で述べたように、人の移動方向を判別する際に、人の移動方向と反対方向に判別されることが原因である。このことから、リアルタイムに出入りする人をカウントする人数カウントシステム PCLWOP の構築を実現したが、実用化を考慮すると、移動方向を判別する箇所のプログラムを改善する必要があると評価した。

参考文献

[1] コロナ国内初感染確認から 3 年人々の暮らしや意識はどう変わったのか｜NHK 放送文化研究所
https://www.nhk.or.jp/bunken/research/yoron/20230501_6.html (閲覧日：2024 年 1 月 30 日)

[2]電車の混雑傾向がスグわかる【混雑予報】機能を公開｜YAHOO! JAPAN 路線情報
https://blog-transit.yahoo.co.jp/info/20200601.html (閲覧日：2024 年 1 月 30 日)

[3] 東京駅に開業するエキナカ商業施設「グランスタ東京」店舗の空港情報配信と web を使った街順管理を株式会社のバカンのサービスで実現｜株式会社バカン｜PRTIMES
https://prtimes.jp/main/html/rd/p/000000046.000018933.html (閲覧日：2024 年 1 月 30 日)

[4] 実証 01 地下空間も含めたリアルタイム人流可視化｜過去の検証｜東京都 デジタルツイン実現プロジェクト
https://info.tokyo-digitaltwin.metro.tokyo.lg.jp/zissyou01/ (閲覧日：2024 年 1 月 30 日)

[5] (本編)地域課題解決のための人流データ利活用の手引き｜報道発表資料｜国土交通省
https://www.mlit.go.jp/report/press/tochi_fudousan_kensetsugyo17_hh_000001_00017.html (閲覧日：2024 年 1 月 30 日)

[6] 通行量調査・交通量調査｜株式会社フェイス・コムリエイト｜ マーケティングリサーチ・Web ク
https://facecom.co.jp/traffic-volume-survey/ (閲覧日：2024 年 1 月 30 日)

[7] Sung In Cho, Suk-Ju Kang, “Real-Time People Counting System for Customer Movement Analysis”, *IEEE Access*, vol. 6, Sep. 2018
https://ieeexplore.ieee.org/document/8478241 (閲覧日：2024 年 1 月 30 日)

[8] 田中 大河, 林田 和人, 柴田 滝也, “OpenPose を用いた歩行者行動特性の計測システムの開発”, 日本建築学会技術報告書, 29 巻 72 号, 書誌, (2023)
https://www.jstage.jst.go.jp/article/aijt/29/72/29_1122/_article/-char/ja/ (閲覧日：2024 年 1 月 30 日)

[9]Adam Paszke, Sam Gross, Fracisco Massa, Adam Lerer et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library .” Dec. 2019
https://doi.org/10.48550/arXiv.1912.01703 (閲覧日：2024 年 1 月 30 日)

[10] “END-TO-END MACHINE LEARNING FRAMEWORK”｜Resources｜PyTorch
https://pytorch.org/features/ (閲覧日：2024 年 1 月 30 日)

[11] “PyTorch”｜ Glossary｜Dagshub
https://dagshub.com/glossary/pytorch/ (閲覧日：2024 年 1 月 30 日)

[12] “Understanding Pytorch Eager and Graph Mode”｜BACKTRACE
https://backtrace.blog/2023/05/21/understanding-pytorch-eager-and-graph-mode/ (閲覧日：2024 年 1 月 30 日)

[13] Zhe Cao, Tomas Simon, Shih-En Wei, Yaser Sheikh. “Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields.” Apr. 2017
https://doi.org/10.48550/arXiv.1611.08050 (閲覧日：2024 年 1 月 30 日)

[14] Hzzone. “pytorch-openpose.” GitHub Repository
https://github.com/Hzzone/pytorch-openpose (閲覧日：2024 年 1 月 30 日)

[15] Daniil Osokin. “Real-time 2D Multi-Person Pose Estimation on CPU: Lightweight OpenPose.” Nov. 2018
https://doi.org/10.48550/arXiv.1811.12004 (閲覧日：2024 年 1 月 30 日)

[16] Daniil Osokin. “lightweight-human-pose-estimation.pytorch.” GitHub Repository
https://github.com/Daniil-Osokin/lightweight-human-pose-estimation.pytorch (閲覧日：2024 年 1 月 30 日)

[17] Chien-Yao Wang, Alexey Bochkovskiy, Hong-Yuan Mark Liao. “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.” Jul. 2022
https://doi.org/10.48550/arXiv.2207.02696 (閲覧日：2024 年 1 月 30 日)

[18] RizwanMunawar. “yolov7-pose-estimation.” GitHub Repository
https://github.com/RizwanMunawar/yolov7-pose-estimation (閲覧日：2024 年 1 月 30 日)

[19] George Papandreou, Tyler Zhu, Nori Kanazawa, Alexander Toshev et al. “Towards Accurate Multi-person Pose Estimation in the Wild.” Apr. 2017
https://doi.org/10.48550/arXiv.1701.01779 (閲覧日：2024 年 1 月 30 日)

[20] rwightman. “posenet-pytorch.” GitHub Repository
https://github.com/rwightman/posenet-pytorch (閲覧日：2024 年 1 月 30 日)

[21] saimj7. “People-Counting-in-Real-Time.” Github Repository
https://github.com/saimj7/People-Counting-in-Real-Time (閲覧日：2024 年 1 月 30 日)

[22] Wei Liu, Dragomir Anguelov, Dumitru Erhan et al. “SSD: Single Shot MultiBox Detector.” Dec. 2016
https://doi.org/10.48550/arXiv.1512.02325 (閲覧日：2024 年 1 月 30 日)

[23] “Pose Estimation: Concepts, Techniques, and How to Get Started” | datagen
https://datagen.tech/guides/pose-estimation/pose-estimation/ (閲覧日：2024 年 1 月 30 日)

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頂きました金丸隆志教授に心より感謝申し上げます。そして、本研究にご協力頂きました研究室の皆様、機械理工学科の皆様に心より感謝いたします。ありがとうございました。

付録

```
import argparse
import cv2
import numpy as np
import torch
import dlib

from models.with_mobilenet import PoseEstimationWithMobileNet
from modules.keypoints import extract_keypoints, group_keypoints
from modules.load_state import load_state
from modules.pose import Pose
from val import normalize, pad_width
from tracker.centroidtracker import CentroidTracker
from tracker.trackableobject import TrackableObject



class ImageReader(object):

    def __init__(self, file_names):
        self.file_names = file_names
        self.max_idx = len(file_names)

    def __iter__(self):
        self.idx = 0
        return self

    def __next__(self):
        if self.idx == self.max_idx:
            raise StopIteration
        img = cv2.imread(self.file_names[self.idx], cv2.IMREAD_COLOR)
        if img.size == 0:
            raise IOError('Image {} cannot be read'.format(self.file_names[self.idx]))
        self.idx = self.idx + 1
        return img
```

```
class VideoReader(object):

    def __init__(self, file_name):
        self.file_name = file_name
        try:
            self.file_name = int(file_name)
        except ValueError:
            pass

    def __iter__(self):

        self.cap = cv2.VideoCapture(self.file_name)
        if not self.cap.isOpened():
            raise IOError('Video {} cannot be opened'.format(self.file_name))
        return self

    def __next__(self):
        was_read, img = self.cap.read()
        if not was_read:
            raise StopIteration
        return img


def infer_fast(net, img, net_input_height_size, stride, upsample_ratio, cpu,
                   pad_value=(0, 0, 0), img_mean=np.array([128, 128, 128], np.float32),
img_scale=np.float32(1/256)):
    height, width, _ = img.shape
    scale = net_input_height_size / height

    scaled_img = cv2.resize(img, (0, 0), fx=scale, fy=scale, interpolation=cv2.INTER_LINEAR)
    scaled_img = normalize(scaled_img, img_mean, img_scale)
    min_dims = [net_input_height_size, max(scaled_img.shape[1], net_input_height_size)]
    padded_img, pad = pad_width(scaled_img, stride, pad_value, min_dims)
```

```
    tensor_img = torch.from_numpy(padded_img).permute(2, 0, 1).unsqueeze(0).float()
    if not cpu:
        tensor_img = tensor_img.cuda()

    stages_output = net(tensor_img)

    stage2_heatmaps = stages_output[-2]
    heatmaps = np.transpose(stage2_heatmaps.squeeze().cpu().data.numpy(), (1, 2, 0))
    heatmaps = cv2.resize(heatmaps, (0, 0), fx=upsample_ratio, fy=upsample_ratio, interpolation=cv2.INTER_CUBIC)

    stage2_pafs = stages_output[-1]
    pafs = np.transpose(stage2_pafs.squeeze().cpu().data.numpy(), (1, 2, 0))
    pafs = cv2.resize(pafs, (0, 0), fx=upsample_ratio, fy=upsample_ratio, interpolation=cv2.INTER_CUBIC)

    return heatmaps, pafs, scale, pad


def run_demo(net, image_provider, height_size, cpu, track, smooth):
    net = net.eval()
    if not cpu:
        net = net.cuda()

    ct = CentroidTracker(maxDisappeared=40, maxDistance=50)
    trackers = []
    trackableObjects = {}

    totalFrames = 0
    totalDown = 0
    totalUp = 0

    move_out = []
    move_in =[]

    stride = 8
```

```
    upsample_ratio = 4
    num_keypoints = Pose.num_kpts
    delay = 1



    W = None
    H = None

    for img in image_provider:
        orig_img = img.copy()
        rgb = cv2.cvtColor(orig_img, cv2.COLOR_BGR2RGB)

        if W is None or H is None:
            (H, W) = rgb.shape[:2]

        status = "Waiting"
        rects = []

        if totalFrames % args_global["skip_frames"] == 0:
            status = "Detecting"
            trackers = []

            heatmaps, pafs, scale, pad = infer_fast(net, img, height_size, stride, upsample_ratio, cpu)

            total_keypoints_num = 0
            all_keypoints_by_type = []
            for kpt_idx in range(num_keypoints):
                total_keypoints_num += extract_keypoints(heatmaps[:, :, kpt_idx],
all_keypoints_by_type, total_keypoints_num)

            pose_entries, all_keypoints = group_keypoints(all_keypoints_by_type, pafs)
            for kpt_id in range(all_keypoints.shape[0]):
                all_keypoints[kpt_id, 0] = (all_keypoints[kpt_id, 0] * stride / upsample_ratio - pad[1]) /
scale
                all_keypoints[kpt_id, 1] = (all_keypoints[kpt_id, 1] * stride / upsample_ratio - pad[0]) /
scale
```

```
current_poses = []
for n in range(len(pose_entries)):
    if len(pose_entries[n]) == 0:
        continue
    pose_keypoints = np.ones((num_keypoints, 2), dtype=np.int32) * -1
    for kpt_id in range(num_keypoints):
        if pose_entries[n][kpt_id] != -1.0:  # keypoint was found
            pose_keypoints[kpt_id, 0] = int(all_keypoints[int(pose_entries[n][kpt_id]), 0])
            pose_keypoints[kpt_id, 1] = int(all_keypoints[int(pose_entries[n][kpt_id]), 1])
    pose = Pose(pose_keypoints, pose_entries[n][18])
    current_poses.append(pose)

for pose in current_poses:
    pose.draw(img)

img = cv2.addWeighted(orig_img, 0.6, img, 0.4, 0)

for pose in current_poses:

    startX = int(pose.keypoints[2][0])
    startY = int(pose.keypoints[1][1])
    endX = int(pose.keypoints[5][0])
    endY = int(pose.keypoints[8][1])

    if endX < startX:
        tmp = endX
        endX = startX
        startX = tmp

    if endY < startY:
        tmp = endY
        endY = startY
        startY = tmp



    if startX>=0 and startX<W and startY>=0 and startY<H and ¥
```

```
                    endX>=0 and endX<W and endY>=0 and endY<H:
                    cv2.rectangle(img, (startX, startY),(endX, endY), (0, 255, 0))
                    rects.append((startX, startY, endX, endY))

                    tracker = dlib.correlation_tracker()
                    rect = dlib.rectangle(startX, startY, endX, endY)
                    tracker.start_track(rgb, rect)

                    trackers.append(tracker)

else:
    for tracker in trackers:
        status = "Tracking"
        tracker.update(rgb)
        pos = tracker.get_position()

        startX = int(pos.left())
        startY = int(pos.top())
        endX = int(pos.right())
        endY = int(pos.bottom())

        if startX>=0 and startX<W and startY>=0 and startY<H and ¥
            endX>=0 and endX<W and endY>=0 and endY<H:
            cv2.rectangle(img, (startX, startY),(endX, endY), (0, 255, 0))
            rects.append((startX, startY, endX, endY))


cv2.line(img, (0, H // 2), (W, H // 2), (0, 0, 0), 3)

objects = ct.update(rects)

for (objectID, centroid) in objects.items():
    to = trackableObjects.get(objectID, None)

    if to is None:
        to = TrackableObject(objectID, centroid)
```

```
        else:
            y = [c[1] for c in to.centroids]
            direction = centroid[1] - np.mean(y)
            to.centroids.append(centroid)

            if not to.counted:

                if direction < 0 and centroid[1] < H // 2:
                    totalUp += 1
                    move_out.append(totalUp)
                    to.counted = True

                elif direction > 0 and centroid[1] > H // 2:
                    totalDown += 1
                    move_in.append(totalDown)
                    to.counted = True

        trackableObjects[objectID] = to


        text = "ID {}".format(objectID)
        cv2.putText(img, text, (centroid[0] - 10, centroid[1] - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)
        cv2.circle(img, (centroid[0], centroid[1]), 4, (255, 255, 255), -1)

    info_status = [
                ("Enter", totalUp),
                ("Exit", totalDown),
                ("Status", status),
                ]

    total_inside = len(move_out) - len(move_in)
    info_total = [
                ("Total people inside", total_inside)
                ]
```

```
            stt1 = "Vacant Inside. Welcome."
            stt2 = "Crowded Inside."
            stt3 = "Overcrowded Inside."

            if total_inside < 3:
                cv2.putText(img, stt1, (265, H - 30), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 255, 0), 2)
            elif 3 <= total_inside < 6:
                cv2.putText(img, stt2, (265, H - 30), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 255), 2)
            else:
                cv2.putText(img, stt3, (265, H - 30), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)



            for (i, (k, v)) in enumerate(info_status):
                text = "{}: {}".format(k, v)
                cv2.putText(img, text, (10, H - ((i * 20) + 20)), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255,
255, 255), 2)

            for (i, (k, v)) in enumerate(info_total):
                text = "{}: {}".format(k, v)
                cv2.putText(img, text, (265, H - ((i * 20) + 60)), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255,
255, 255), 2)

                cv2.imshow('Lightweight Human Pose Estimation Python Demo', img)
                key = cv2.waitKey(delay)



                if key == 27 or key == ord('q'):
                    return
                elif key == 112:
                    if delay == 1:
                        delay = 0

                    else:
                        delay = 1
```

```
            totalFrames += 1


args_global=""
if __name__ == '__main__':
    parser = argparse.ArgumentParser(
        description="""Lightweight human pose estimation python demo.
                       This is just for quick results preview.
                       Please, consider c++ demo for the best performance.""")
    parser.add_argument('--checkpoint-path', type=str, required=True, help='path to the checkpoint')
    parser.add_argument('--height-size', type=int, default=256, help='network input layer height size')
    parser.add_argument('--video', type=str, default='', help='path to video file or camera id')
    parser.add_argument('--images', nargs='+', default='', help='path to input image(s)')
    parser.add_argument('--cpu', action='store_true', help='run network inference on cpu')
    parser.add_argument('--track', type=int, default=1, help='track pose id in video')
    parser.add_argument('--smooth', type=int, default=1, help='smooth pose keypoints')
    parser.add_argument("-s", "--skip-frames", type=int, default=1, help="# of skip frames between detections")

    args = parser.parse_args()
    args_global = vars(args)
    if args.video == '' and args.images == '':
        raise ValueError('Either --video or --image has to be provided')

    net = PoseEstimationWithMobileNet()
    checkpoint = torch.load(args.checkpoint_path, map_location='cpu')
    load_state(net, checkpoint)

    frame_provider = ImageReader(args.images)
    if args.video != '':
        frame_provider = VideoReader(args.video)
    else:
        args.track = 0
    run_demo(net, frame_provider, args.height_size, args.cpu, args.track, args.smooth)
```