平成２７年度　特別研究報告書

スマートフォンによる在室管理を用いた家電制御最適化の研究

龍谷大学　理工学部　情報メディア学科

T128005　田中 大地

指導教員　三好 力　教授

内容概要

現在、家電のネットワーク化が進み、家電同士を連携させ生活の利便性を高める動きが活発になっている。また、スマートフォンの普及と共に外出先から家の中にある家電を監視、制御するシステムの利用がはじまっている。しかし、照明やエアコン制御においては未だ人の操作で行われている場合も多く、それらの制御には在室状況を判断する必要もある。本研究では家電制御と入室時のドアの開閉動作に着目し、情報を取得するセンサとしてスマートフォンを用いて、利用者の在室状況に応じた家電制御を行うために、入退室から在室状況の推定を行う技術の確立を目指す。

目次

第 1 章　はじめに

1.1 研究背景

現在、スマートフォンの普及に伴い、スマート家電[1]と呼ばれるスマートフォンと連携させた家電製品、もしくはスマートグリッド等との連携により、自動的にエネルギー消費を最適化できる家電製品が市場に登場している。例えば、冷蔵庫であれば庫内のカメラ映像をスマートフォンの画面に映し出すことで、どの食材がストックされているのかを確認できる。洗濯機は洗濯終了時や乾燥フィルターが詰まった時にメールで知らせることができ、洗濯物の放置を防げるので洗い直しの手間が発生しないといったメリットがあるなど、外出先から家電の利用状況を確認し、それに合わせた生活を送ることが可能になった。

図 1.スマート家電

しかし、照明やエアコンといった室内で利用する家電についてはスマートフォンでスイッチの ON・OFF または設定項目の調整ができるといった、リモコンの代わりとなる機能にとどまっており、室内で直接操作することと大差がないのが現状である。

1.2 中古スマートフォン市場の拡大

スマートフォンの普及と共に中古携帯端末の市場も拡大している。中古携帯端末市場を「国内の消費者による中古携帯電話・中古スマートフォン等の購入」と定義し規模を推計したところ、2014 年度に市場は 227 万台で前年度比 26.8%増に達したとみられ、今後も市場は拡大 2020 年度には 360 万台になると予測される。[2]

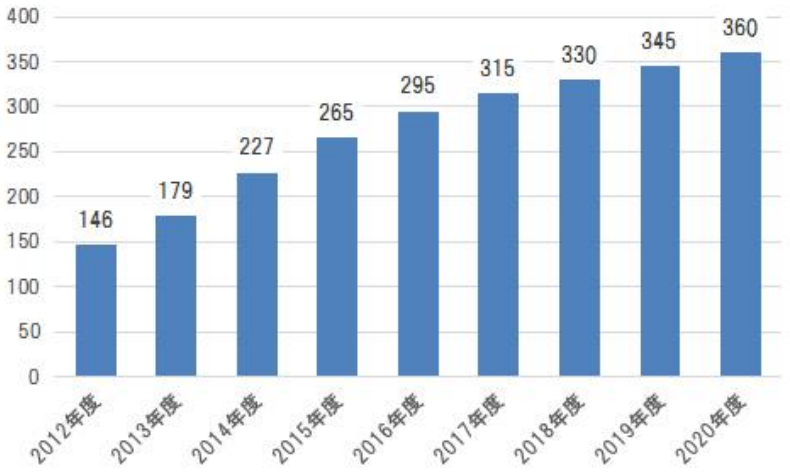

図 2.中古端末市場予測・消費者による購入（単位：万台）

1.3 スマートフォンに搭載されている各種センサ

スマートフォンにはインターネット機能やカメラ機能をはじめ、様々機能が付いているが、中でもあまり知られていないのが非常に多くのセンサを搭載していることだ。特に Android 搭載端末においては表 1 に示す通り、非常に多くのセンサを利用でき、これらはスマートフォンの機能やアプリケーションの動作に使われている。[3]

表 1　Android バージョンごとの主なサポートするセンサ

種類	センサータイプ（定数）	Android 4.4	4.0	2.3	2.2	1.5
加速度	TYPE_ACCELEROMETER	Yes	Yes	Yes	Yes	Yes
周辺温度	TYPE_AMBIENT_TEMPERATURE	Yes	Yes	N/A	N/A	N/A
重力	TYPE_GRAVITY	Yes	Yes	Yes	N/A	N/A
ジャイロスコープ	TYPE_GYROSCOPE	Yes	Yes	Yes	N/A※1	N/A※1
輝度（照度）	TYPE_LIGHT	Yes	Yes	Yes	Yes	Yes
重力加速度を除いた加速度	TYPE_LINEAR_ACCELERATION	Yes	Yes	Yes	N/A	N/A
磁界（磁気）	TYPE_MAGNETIC_FIELD	Yes	Yes	Yes	Yes	Yes
方位	TYPE_ORIENTATION	Yes※2	Yes※2	Yes※2	Yes※2	Yes
気圧	TYPE_PRESSURE	Yes	Yes	Yes	N/A※1	N/A※1
近接	TYPE_PROXIMITY	Yes	Yes	Yes	Yes	Yes
湿度	TYPE_RELATIVE_HUMIDITY	Yes	Yes	N/A	N/A	N/A
回転ベクトル	TYPE_ROTATION_VECTOR	Yes	Yes	Yes	N/A	N/A
ステップカウンター	TYPE_STEP_COUNTER	Yes	N/A	N/A	N/A	N/A
ステップ感知	TYPE_STEP_DETECTOR	Yes	N/A	N/A	N/A	N/A
温度	TYPE_TEMPERATURE	Yes※2	Yes※2	Yes	Yes	Yes

※1 このセンサタイプは Android 1.5 で追加されたが、Android 2.3 まで使用できない

※2 このセンサは利用可能だが非推奨になった

1.4 研究の目的

先に示したように、照明やエアコンのような比較的単純な制御で十分な家電においては、スマートフォンで直接操作するよりもセンサ情報を利用した自動制御が好ましいと考えられる。また、今後中古携帯端末の拡大から高性能にも関わらず古くて使われなくなったスマートフォンが、数多く流通することが考えられる。

本研究では、今後増加すると予想される古くて使われなくなった端末を再利用して、スマートフォンのカメラ機能とインターネット機能、各種センサ情報に着目し、室内での在室状況を判断し照明やエアコンを自動制御するシステムの開発を目的とする。

第 2 章　既存技術

2.1 センサによる照明制御

現在、省エネ性の高い LED 照明器具が普及しており、白熱電球に比べて消費電力が約 1/5 程度に抑えられると言われている。しかし、近年の環境意識の高まりや電力需給の逼迫から、より一層の省エネルギーを図ることが照明設備にも求められている。照明の制御には様々なセンサが使われているが、基本的にセンサ範囲内に人が通れば検知できるというもので、室内で利用する場合そのセンサ範囲が室内全域をカバーしていない限り、在室状況を判断することは不可能である。ここで、近年普及してきたセンサによる照明設備の制御技術を以下に示す。

2.1.1 熱線センサ

熱線センサ[4]は、人体から放射される赤外線を検知する方式のセンサで、一般的に PIR（Passive Infrared）センサと呼ばれている。その動作原理は、地球上の全ての物体がその温度と表面状態に応じた輻射熱を放射していることを利用したもので、人が検知エリアに入った時に、センサに入射する赤外線の量が人体表面と背景との温度差に相当した量だけ変化すること、つまり、熱源である人体と、床や壁などの背景との温度差（約 2～3℃以上）に反応して、人の動作を捉えることができる。

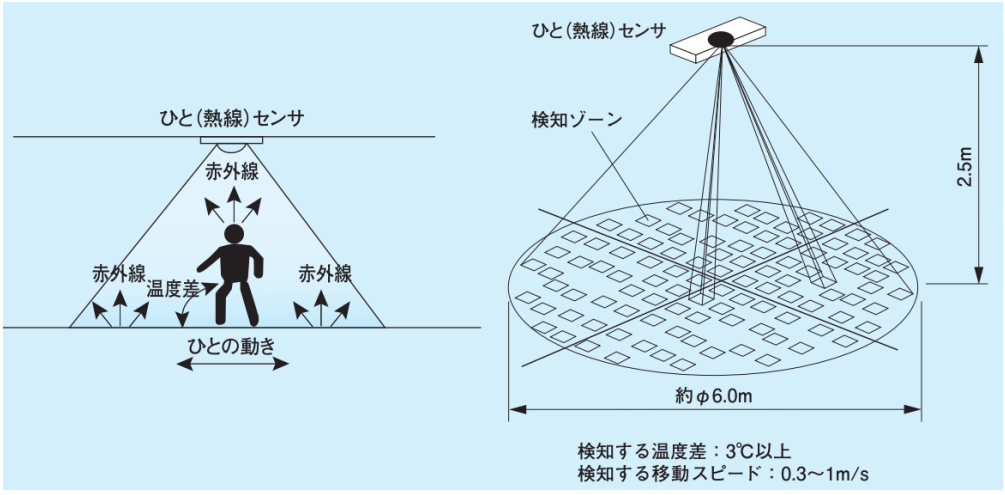

図 3.熱線センサの動作原理と検知範囲

2.1.2 電波式センサ

電波式センサ[4]はドップラーモジュールによる動体検知を行っているセンサで、マイクロ波（24GHz 帯）を発信する。センサから発せられた電波は、空間内の物体に面し反射や透過しながら空間内を進行するが、そのうち物体に当たり反射してきた電波をセンサ内蔵のアンテナにて受信する。この際、電波を反射させた物体が移動していた場合、発信した電波と受信した電波との間に物体の移動速度に応じた周波数の変化が生じるため、周波数差を利用して、人の移動、ドアの開閉を検知することができる。また、このセンサから発する電波は、無線 LAN と同等レベルの強度（10mW 以下）のため人体への影響はない。

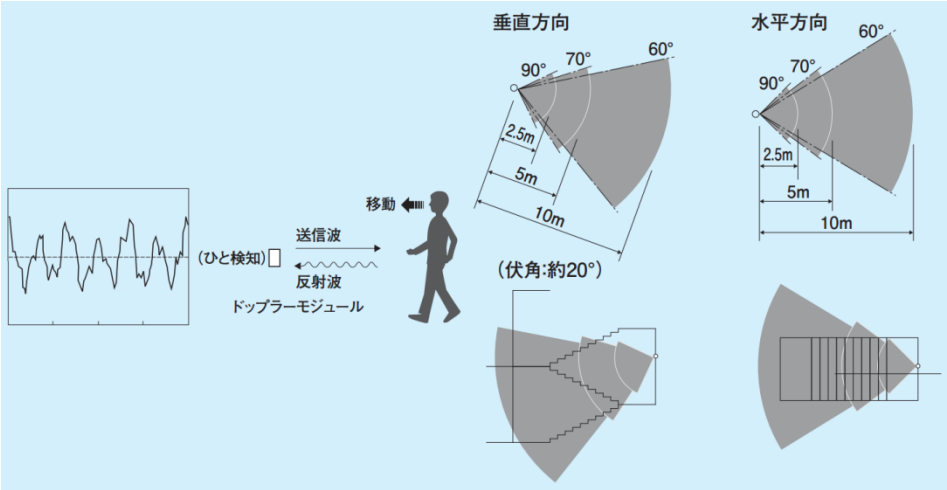

図 4.電磁波センサの動作原理と検知範囲

2.1.3 明るさセンサ

センサの検知範囲内の床面や机上面などからの反射光を入射光量として記憶し、その光量が常に一定になるように照明器具の光出力を自動調節することで、明るさを一定に保つ働きをするもの。[4]

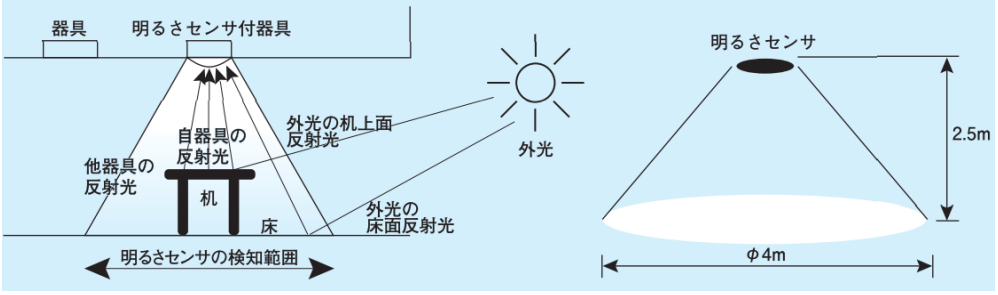

図 5.明るさセンサの動作原理と検知範囲

2.2 画像センサ

画像センサ[4]は、画像素子を利用して人の存在や行動、または空間の明るさを検知して照明制御を行うもので、その動作原理は設置空間を画像で取り込み、その画像の時間変化を解析して、背景画像と現在の画像の差分から人や明るさを検知するもの。

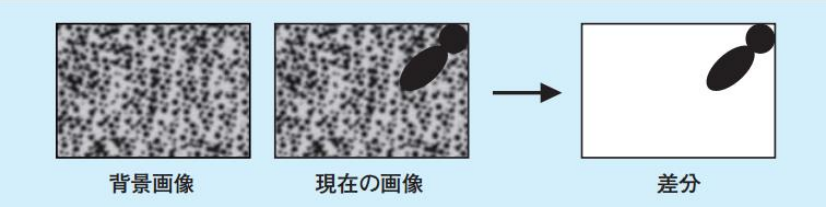

図 6.画像センサの検知アルゴリズム

2.3 背景差分法

カメラが固定されていて視界の移動がない場合に画像中から注目すべき物体を抽出する定番の手法として背景差分法（Background Subtraction Method）[5]がある。図 7 のように、移動物体が存在しない背景モデル（background model）をあらかじめ作成して、入力画像（current frame）と比較し、既知の背景部分を取り除くことで前景領域（foreground mask）を抽出し移動物体を検出するもの。

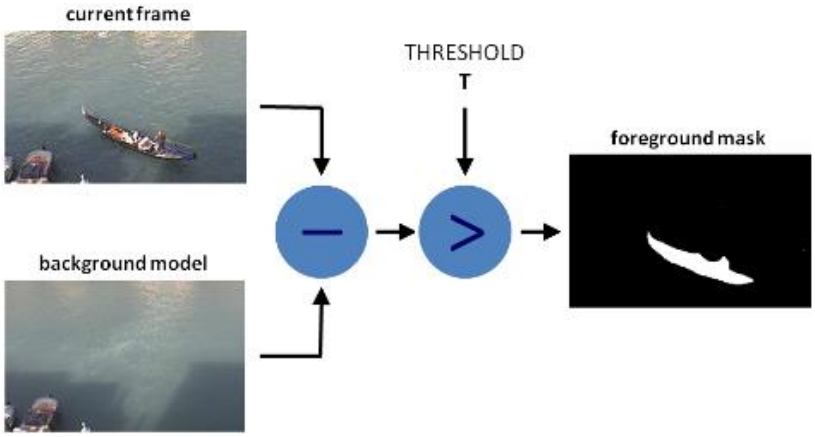

図 7.背景差分法の原理

2.4 赤外線リモコンデバイス「IRKit」

IRKit[6]は、Wi-Fi 機能の付いたオープンソースな赤外線リモコンデバイス。赤外線信号のデータを取得、発信することがでるため、家庭のエアコンやテレビ、ライトなど、赤外線で操作できる家電を Wi-Fi を通して、iPhone や iPad、Android スマートフォンなどから操作できる。また、iOS/Android の SDK（Software Development Kit ）が公開されており、任意のタイミングで赤外線信号を送ることのできる iOS/Android アプリを簡単につくることができる。

図 8.IRKit

2.5 「ステンレス・クリーン 白くまくん X シリーズ」

人、部屋の間取り、家具の位置まで捉える「くらしカメラ 3D」[7]を搭載した、日立アプライアンスの最上位モデルに当たるエアコン。日立のエアコンでは従来より、在室者の位置や室内の間取りを検知する「画像カメラ」と、在室者の周囲の温度を見る「温度カメラ」、ソファやテーブルなどの位置や形状を見る「ものカメラ」がある。「ものカメラ」は、近赤外線 LED を搭載し、画像カメラに近赤外線透過フィルターを起動させる構造。取得した近赤外線画像をもとに、家具の存在や位置、「気流が通るかどうか」の形状を検知する。これにより、気流の通り道を見つけ、家具などに遮られずに風を届けられるという。暖房時は人のいる位置や足元に温風を届け、冷房時は、人のいるエリアを中心としながら、冷風を効率よく循環させ、部屋全体を涼しくする。

第 3 章　提案手法

3.1 概要

照明やエアコンなど在室中に利用する家電を制御するためには利用者の入退室から在室状況を判断する必要がある。現在は人感センサを使って照明の自動制御を行うのが一般的となっているが、センサの範囲や性質によって反応しない場合があり、現状では十分でないと言える。そこで、スマートフォンをドアに設置し、カメラ機能を利用して撮影した動画から入退室者を追跡することで在室管理を実現し、家電制御を行うシステムを提案する。

本研究では、ドアの開閉と照明の点消灯が一連の動作になっていることを利用し、ドアの開閉時に入室か退室かを判断し、そこから在室人数を推定し、在室状況に応じて照明やエアコンを制御するシステムの開発を行うため、システム全体の内、最重要部分である入退室者の追跡を優先して開発し、実験でその性能を確認した。

3.2 システムの全体像

図 9 に示すように、ドアノブの付近にスマートフォンを設置し、カメラ機能でドアの開閉時に人の入退室を記録、動画像処理から入退室を判断し、在室人数を推定する。その後、在室人数とスマートフォンのセンサから取得した輝度と温度から照明とエアコンの必要性を判断し、設置したスマートフォンから赤外線リモコンデバイス「IRKit」に信号を送信することで、照明とエアコンの制御を行う。

図 9.本システムの流れ

3.3 入退室者の追跡

3.3.1 入退室について

入り口が一つしかない部屋に限定して考えると、ドアの前で入退室を記録することで在室人数を推定することができる。図 10 で入退室の流れを示すが、ドアの開閉と照明の点消灯が一連の動作になっていることがわかる。

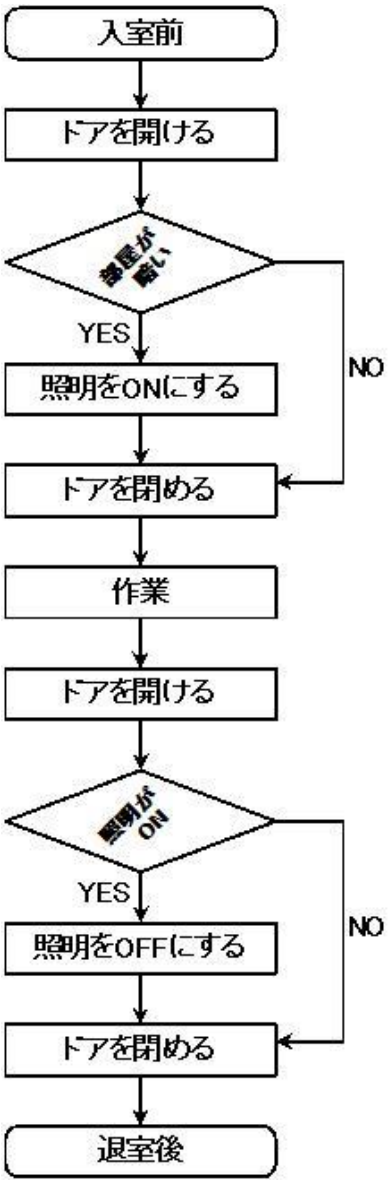

図 10.入退室の流れ

3.3.2　加速度変化による追跡タイミングの推定

一般的にドアの開閉は連続した動作であるが、開から閉に切り替わる時にドアが瞬間的に静止していると考えられる。つまり、その静止したタイミングに入退室者はドアを通過中であることから、ドアの加速度を分析することで入退室者がドアを通過中であることを推定することができる。

3.3.3 CamShift 法

入退室者の追跡には OpenCV ライブラリ関数の cvCamShift を用いる。追跡対象領域における色相値のヒストグラムに着目し、現フレームの画像中で追跡対象のヒストグラム特徴により近くなる位置にシフトしていく。そして、シフトの移動量が一定以下になるか、繰り返し回数が上限に達するまで処理を繰り返すことで対象領域の現在の位置を追跡する。これを利用して入退室者を追跡し進行方向の特定を行う。フローチャートを図 11 に示す。

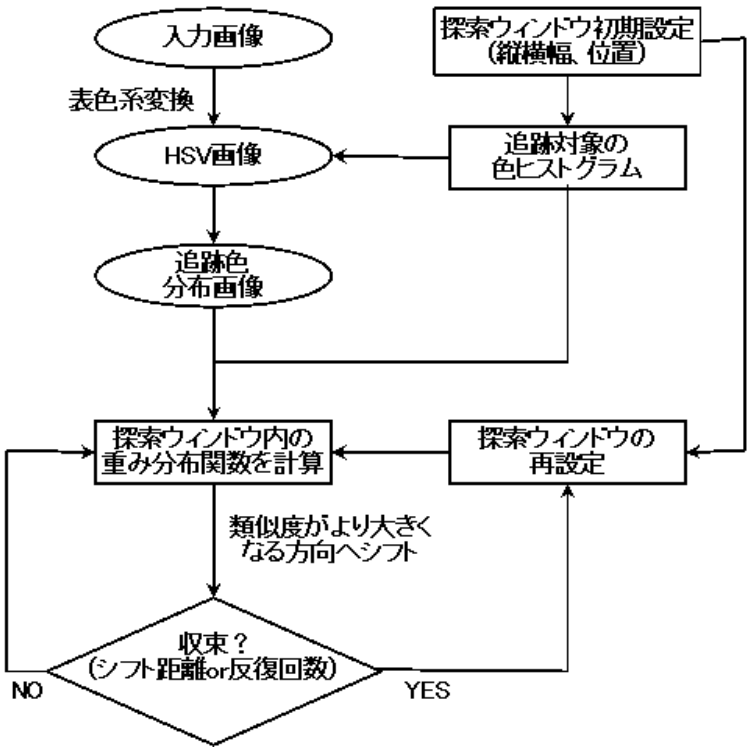

図 11.CamShift 法のフローチャート

3.4 開発環境

本研究のシステムの開発環境を表 2 に示す

表 2.開発環境

OS	Windows7
開発ツール	Microsoft Visual Studio 2015
開発言語	C++
ライブラリ	OpenCV 1.0

第 4 章　実験と考察

4.1 実験概要

本システムを実現するためには、追跡領域を自動で設定するために人がドアを通過するタイミングを理解する必要がある。そこで、スマートフォンのカメラから動画を撮影すると同時に、加速度センサを用いてドアの開閉時の加速度を記録した。加速度センサから得られたデータと、人の入退室動作を撮影した動画像から、ドアの開閉時の加速度変化と人の入退室動作との関係性がどのようなものかを調査した。また、入退室の判定には動画像から人と背景を判別し、入退室者を追跡することが必要不可欠である。移動物体の検出には背景差分法が用いられるが、本システムのように背景が大きく変化する場合には利用することができない。そこで本章では、ドアに固定したスマートフォンのカメラから撮影した動画を、3.2.2 で示した CamShift 法を用いて動画像処理を行い入退室者の追跡が可能となるか、処理の評価と考察を行った。

4.2 実験方法

4.2.1 加速度の変化

加速度の記録には、視覚的に加速度変化を確認できるだけではなく、テキストデータとして加速度を記録することができる Android アプリ『Accelerometer Monitor』を使用した。

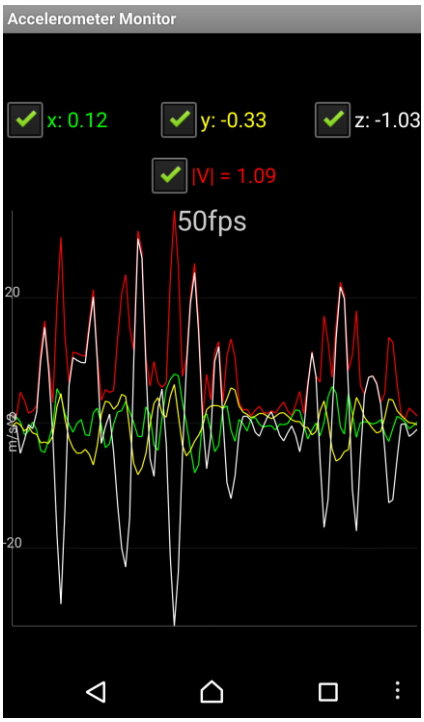

図 12.Accelerometer Monitor

4.2.2 動画撮影

内開きドアの外側のドアノブ付近(高さ約 90 ㎝)にスマートフォンを固定した。その状態からドアを歩行速度で 30 往復し、同時に各種アプリを使って実験に必要なドア開閉時の動画と加速度の記録を行った。使用したドアには図 13 のようなドアクローザが設置されており、通過後はゆっくりと自動的に閉まる。

図 13.ドアクローザ

4.2.3 CamShift 法による追跡

図 14 のように動画再生中に追跡対象が画面中央に来た時点で初期追跡領域を指定し、その後、図 15 のように追跡領域である赤い楕円が対象から外れることなく進行方向へ移動することで追跡完了とする。本システムを実現するには初期追跡領域を自動的に指定する必要があるが、ドアの加速度変化を利用することを想定しているため、本実験ではマウスを使って手動で指定し、背景が大きく変化する場合でも追跡可能かを確認する。

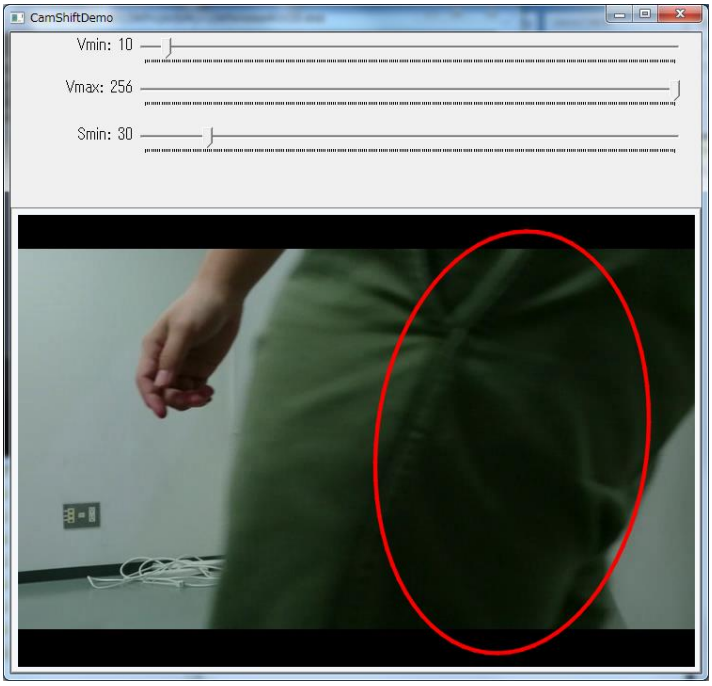

図 14.初期追跡領域の決定

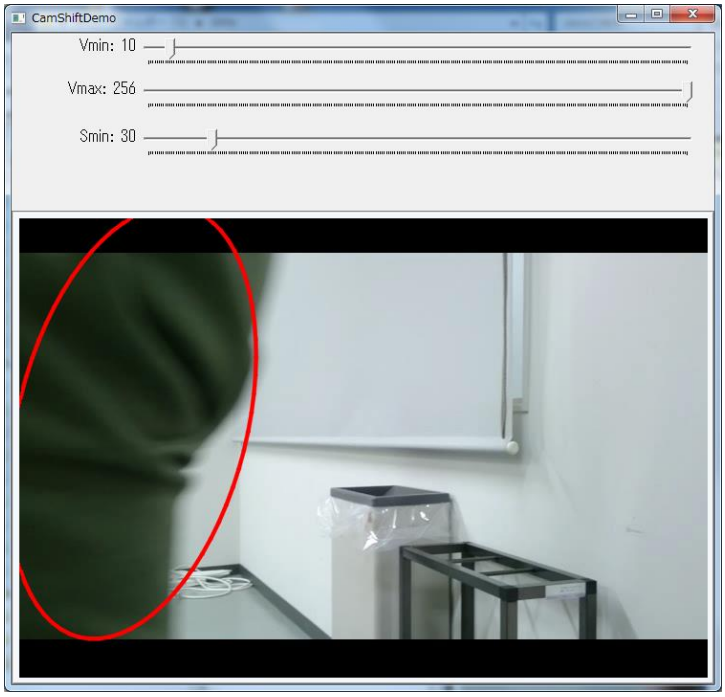

図 15.追跡完了

4.3 実験結果

4.3.1 加速度の変化

図 16 はドアを一往復した時の z 軸の加速度のグラフである。今回は変化量が最も大きく、ドアに対して垂直な軸にあたる z 軸に注目した。往・復、共に同様の加速度変化の特徴が見られた。同時に撮影した動画でドアの動きを見比べると、図 17 で示したように横軸 50ms 付近でドアが開き、100ms 付近でドアが開ききり、人が通過することがわかった。その後、ドアが閉じていき 230ms 付近で閉じる。最後に大きい値が見られるのはドアが閉まったときの衝撃である。

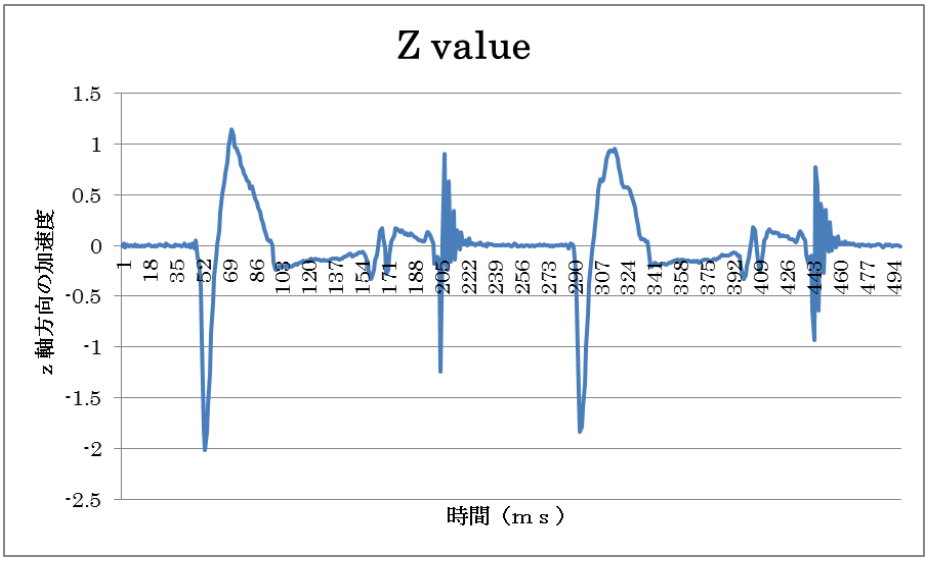

図 16. z 軸の加速度変化

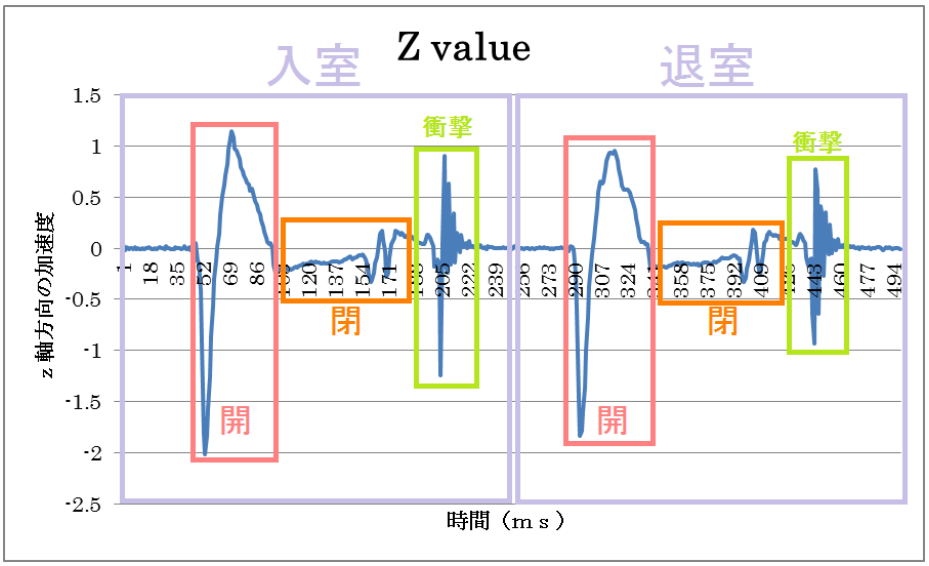

図 17. ドアの加速度と開閉動作の関係

4.3.2 追跡の結果

記録取得の際には入退室合計 30 回の試行のうち 26 回の追跡に成功した。初期追跡領域を正しく選択できた場合、図 14 で示した赤い楕円から外れることなく入退室者を追跡することができた。ドアの開閉範囲が適切でなかった場合、初期追跡領域の色相に近い物体が背景に写り込んでしまい、追跡領域がそちらに移ってしまうことがあった。表 3 に実験記録を、表 4 に性能評価を示す。

表 3.実験結果

入室	追跡結果	退室	追跡結果
1 回目	○	1 回目	○
2 回目	○	2 回目	○
3 回目	×	3 回目	×
4 回目	○	4 回目	○
5 回目	○	5 回目	○
6 回目	○	6 回目	○
7 回目	○	7 回目	○
8 回目	○	8 回目	○
9 回目	○	9 回目	○
10 回目	○	10 回目	○
11 回目	○	11 回目	○
12 回目	○	12 回目	○
13 回目	○	13 回目	○
14 回目	×	14 回目	×
15 回目	○	15 回目	○

表 4.性能評価

試行回数	追跡成功数	成功率
30	26	87%

4.4 考察

実験の結果からドアの開閉動作の加速度には特徴的な変化が見られることがわかった。ドアが静止していて、加速度変化が無い状態から一度大きく値が変化し、なだらかな変化に切り替わる瞬間が、ドアが開ききり、人が通過するタイミングであることといえる。また、CamShift 法を用いることで、背景差分法では扱えないような背景が大きく変化している場合でも人の追跡が可能なことが確認できた。加速度による入退室者の通過タイミングの推定と CamShift 法による追跡を利用することで、本システムの実現が可能となると考える。

第5章　まとめ

本論文では、ドアの開閉時の加速度の値が特徴的な変化をすることが確認できた。さらに、加速度の値とドアの動きの関わりを調べることで、人がドアを通過するタイミングが予測できることがわかった。次に、Camshift 法を用いた動画像解析を行い、追跡対象の色ヒストグラムから、カメラの視点が移動し背景が大きく変化する状態で移動する人物の追跡をすることができた。また、背景に追跡対象と類似する色の物体が存在する場合、それを追跡対象として誤認識してしまうことが確認できた。しかし、撮影された画面の大部分は、通過する人物が占めるため、初期追跡範囲を調整することで改善することができると考える。ドアの加速度変化の情報と、ドアに設置したカメラによる動画像解析を組み合わせることで、入退室者の追跡が可能となり、本システムに必要な在室人数の推定を行うことができる。

謝辞

本研究を進めるにあたり、数々のご指導を賜りました三好力教授に心から感謝の意を表します。また、研究過程で数々の助言を頂きました同研究室の皆様に感謝の意を表します。

参考文献

[1]. スマート家電とは？ | Panasonic Smart App | Panasonic -
<http://panasonic.jp/pss/ap/ >

[2]. 中古端末の市場規模、2014 年度は 227 万台、2020 年度に 360 万台へ
<http://www.mca-mbiz.jp/news/2015/07/snapshot-150728.html>

[3]. Android で動く携帯 Java アプリ作成入門（51）
<http://www.atmarkit.co.jp/ait/articles/1405/22/news040.html>

[4]. センサによる照明省エネ制御｜照明設計資料
<http://www2.panasonic.biz/es/lighting/plam/knowledge/design_knowledge.html>

[5]. OpenCV: How to Use Background Subtraction Methods
<http://docs.opencv.org/master/d1/dc5/tutorial_background_subtraction.html#gsc.tab=0>

[6]. IRKit - Open Source WiFi Connected Infrared Remote Controller
< http://getirkit.com/ >

[7]. 家具の位置も検知して、気流をコントロールするエアコン
<http://kaden.watch.impress.co.jp/docs/news/20140918_667200.html>

付録

```
#ifdef _CH_
#pragma package <opencv>
#endif

#ifndef _EiC
#include "cv.h"
#include "highgui.h"
#include <stdio.h>
#include <ctype.h>
#endif

IplImage *image = 0, *hsv = 0, *hue = 0, *mask = 0, *backproject = 0,
*histimg = 0;
CvHistogram *hist = 0;

int backproject_mode = 0;
int select_object = 0;
int track_object = 0;
int show_hist = 1;

//              CamShiftトラッキング用変数
CvPoint origin;
CvRect selection;
CvRect track_window;
CvBox2D track_box;
CvConnectedComp track_comp;

//              ヒストグラム用変数
int hdims = 16;
                                        //              ヒストグラ
ムの次元数
float hranges_arr[] = { 0,180 };        //ヒストグラムのレンジ
float* hranges = hranges_arr;
int vmin = 10, vmax = 256, smin = 30;

//
//              マウスドラッグによって初期追跡領域を指定する
//
//              引数:
//                      event       : マウス左ボタンの状態
//                      x                       : マウスが
現在ポイントしているx座標
//                      y                       : マウスが
現在ポイントしているy座標
//                      flags       : 本プログラムでは未使用
//                      param       : 本プログラムでは未使用
//

void on_mouse(int event, int x, int y, int flags, void* param)
{
        //              画像が取得されていなければ、処理を行わな
い
        if (!image)
                return;

        //              原点の位置に応じてyの値を反転（画像の反
転ではない）
        if (image->origin)
                y = image->height - y;

        //              マウスの左ボタンが押されていれば以下の
処理を行う
        if (select_object)
        {
                selection.x = MIN(x, origin.x);
                selection.y = MIN(y, origin.y);
                selection.width = selection.x + CV_IABS(x
- origin.x);
                selection.height = selection.y + CV_IABS(y
- origin.y);

                selection.x = MAX(selection.x, 0);
                selection.y = MAX(selection.y, 0);
                selection.width = MIN(selection.width,
image->width);
                selection.height = MIN(selection.height,
image->height);
                selection.width -= selection.x;
                selection.height -= selection.y;
        }

        //              マウスの左ボタンの状態によって処理を分
岐
        switch (event)
        {
        case CV_EVENT_LBUTTONDOWN:
                //              マウスの左ボタンが押され
たのであれば、
                //              原点および選択された領域
を設定
                origin = cvPoint(x, y);
                selection = cvRect(x, y, 0, 0);
                select_object = 1;
                break;
        case CV_EVENT_LBUTTONUP:
                //              マウスの左ボタンが離され
たとき、widthとheightがどちらも正であれば、
                //              trackObjectフラグを
TRACKING_STARTにする
                select_object = 0;
                if (selection.width > 0 && selection.height
> 0)
                        track_object = -1;
                break;
        }
}

//
//              入力された1つの色相値をRGBに変換する
//
//              引数:
//                      hue                     : HSV表色
系における色相値H
//              戻り値 :
//                      CvScalar: RGBの色情報がBGRの順で格納
されたコンテナ
//
CvScalar hsv2rgb(float hue)
{
        int rgb[3], p, sector;
        static const int sector_data[][3] =
        { { 0,2,1 },{ 1,2,0 },{ 1,0,2 },{ 2,0,1 },{ 2,1,0 },{ 0,1,2 } };
        hue *= 0.033333333333333333333333333333333f;
        sector = cvFloor(hue);
        p = cvRound(255 * (hue - sector));
        p ^= sector & 1 ? 255 : 0;

        rgb[sector_data[sector][0]] = 255;
        rgb[sector_data[sector][1]] = 0;
        rgb[sector_data[sector][2]] = p;

        return cvScalar(rgb[2], rgb[1], rgb[0], 0);
}

int main(int argc, char** argv)
{
        CvCapture* capture = 0;                 //
        キー入力結果を格納する変数



                                                //
        カメラを初期化する?

                capture =
cvCaptureFromAVI("data11.avi");

        if (!capture)
        {
                //              カメラが見つからなかった
場合
                fprintf(stderr, "Could not initialize
capturing...¥n");
                return -1;
        }

        printf("Hot keys: ¥n"
                "¥tESC - quit the program¥n"
                "¥tc - stop the tracking¥n"
                "¥tb - switch to/from backprojection
view¥n"
                "¥th - show/hide object histogram¥n"
                "To initialize tracking, select the object
with mouse¥n");

        cvNamedWindow("Histogram", 1);
        cvNamedWindow("CamShiftDemo", 1);
        cvSetMouseCallback("CamShiftDemo", on_mouse, 0);
        cvCreateTrackbar("Vmin", "CamShiftDemo", &vmin, 256,
0);
        cvCreateTrackbar("Vmax", "CamShiftDemo", &vmax, 256,
0);
        cvCreateTrackbar("Smin", "CamShiftDemo", &smin, 256,
0);
```

```
	for (;;)
	{
		IplImage* frame = 0;
		int i, bin_w, c;

		//		キャプチャ画像の取得に失敗したら処理を中断
		frame = cvQueryFrame(capture);
		Sleep(30);
	//再生速度調整
		if (!frame)
			break;

		//		キャプチャに成功したら処理を継続する
		if (!image)
		{
			/* allocate all the buffers */
//		各種画像の確保
			image = cvCreateImage(cvGetSize(frame), 8, 3);
			image->origin = frame->origin;
			hsv = cvCreateImage(cvGetSize(frame), 8, 3);
			hue = cvCreateImage(cvGetSize(frame), 8, 1);
			mask = cvCreateImage(cvGetSize(frame), 8, 1);
			backproject = cvCreateImage(cvGetSize(frame), 8, 1);
			//		ヒストグラム構造体の使用を宣言
			hist = cvCreateHist(1, &hdims, CV_HIST_ARRAY, &hranges, 1);
			//		ヒストグラム用の画像を確保し、ゼロクリア
			histimg = cvCreateImage(cvSize(320, 200), 8, 3);
			cvZero(histimg);
		}
		//		キャプチャされた画像をresultImageにコピーし、HSV表色系に変換してhsvImageに格納
		cvCopy(frame, image, 0);
		cvCvtColor(image, hsv, CV_BGR2HSV);

		//		trackObjectフラグが???なら、以下の処理を行う
		if (track_object)
		{
			int _vmin = vmin, _vmax = vmax;

			cvInRangeS(hsv, cvScalar(0, smin, MIN(_vmin, _vmax), 0),
		cvScalar(180, 256, MAX(_vmin, _vmax), 0), mask);
			cvSplit(hsv, hue, 0, 0, 0);
			//		trackObjectが0より小さいなら、以下の処理を行う
			if (track_object < 0)
			{
				//		追跡領域のヒストグラム計算とhistImageへの描画
				float max_val = 0.f;

		cvSetImageROI(hue, selection);

		cvSetImageROI(mask, selection);
				//		ヒストグラムを計算し、最大値を求める

		cvCalcHist(&hue, hist, 0, mask);

		cvGetMinMaxHistValue(hist, 0, &max_val, 0, 0);
				//		ヒストグラムの縦軸（頻度）を0-255のダイナミックレンジに正規化

		cvConvertScale(hist->bins, hist->bins, max_val ? 255. / max_val : 0., 0);
				//		hue,mask画像に設定されたROIをリセット

		cvResetImageROI(hue);

		cvResetImageROI(mask);
		track_window = selection;
		//		trackObjectを1にする
		track_object = 1;

		//		ヒストグラム画像をゼロクリア

		cvZero(histimg);
		//		各ビンの幅を決める
		bin_w = histimg->width / hdims;

		for (i = 0; i < hdims; i++)
		{

	int val = cvRound(cvGetReal1D(hist->bins, i)*histimg->height / 255);

	CvScalar color = hsv2rgb(i*180.f / hdims);

	cvRectangle(histimg, cvPoint(i*bin_w, histimg->height),
			cvPoint((i + 1)*bin_w, histimg->height - val),

			color, -1, 8, 0);
		}
		}
		//		バックプロジェクションを計算する
		cvCalcBackProject(&hue, backproject, hist);
		cvAnd(backproject, mask, backproject, 0);
		//		CamShift法による領域追跡を実行する
		cvCamShift(backproject, track_window,

	cvTermCriteria(CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 10, 1),

	&track_comp, &track_box);
		track_window = track_comp.rect;

		if (backproject_mode)

	cvCvtColor(backproject, image, CV_GRAY2BGR);
		if (image->origin)

	track_box.angle = -track_box.angle;
		cvEllipseBox(image, track_box, CV_RGB(255, 0, 0), 3, CV_AA, 0);
		}
		//		マウスで選択中の初期追跡領域の色を反転させる
		if (select_object && selection.width > 0 && selection.height > 0)
		{
		cvSetImageROI(image, selection);
		cvXorS(image, cvScalarAll(255), image, 0);
		cvResetImageROI(image);
		}

		//		画像を表示する
		cvShowImage("CamShiftDemo", image);
		cvShowImage("Histogram", histimg);

		//		キー入力を待ち、押されたキーによって処理を分岐させる
		c = cvWaitKey(10);
		if ((char)c == 27)
		//		while無限ループから脱出（プログラムを終了）
		break;
		switch ((char)c)
		{
		case 'b':
		//		表示画像をバックプロジェクション画像に切り替える
		backproject_mode ^= 1;
```

```
                                        break;
                                case 'c':
                                        //                      トラッキン
グを中止する
                                        track_object = 0;
                                        cvZero(histimg);
                                        break;
                                case 'h':
                                        //                      ヒストグラ
ムの表示/非表示を切り替える
                                        show_hist ^= 1;
                                        if (!show_hist)

                        cvDestroyWindow("Histogram");
                                        else

                        cvNamedWindow("Histogram", 1);
                                        break;
                                default:
                                        ;
                                }
                        }

                        cvReleaseCapture(&capture);
                        cvDestroyWindow("CamShiftDemo");

                        return 0;
}

#ifdef _EiC
main(1, "camshiftdemo.c");
#endif
```