2015 年度

卒　　業　　論　　文

顔認識機能を持つロボットシステム

指導教員　白井英俊　教授

中京大学　情報理工学部　機械情報工学科

学籍番号　H412095　日比　士

(2016 年　1 月)

卒業論文要旨

題目	顔認識機能を持つロボットシステム				
学籍番号	H412095	氏名	日比士	指導教員	白井英俊

本研究は顔認識機能を持つロボットシステムの作成を目的としたものである。顔認識機能とはWebカメラで顔を検出して事前に学習した複数のデータと比較して、その結果から最も似ているものを識別するものである。その結果に基づいてロボットに色々な動作をさせることを考えた。

研究のきっかけはゼミで学んだ顔認識の仕組みと小型ロボットと組み合わせてみると面白いと思ったからである。そのロボットの CPU として Raspberry Pi シリーズで最も処理性能がよかった Raspberry　Pi モデル 2 を利用した。また本研究で作成したロボット本体は DC モーターを２つ備えた車輪付きロボットとした。

本研究では顔検出システム、ロボットの制御、顔認識システムの３つに分けて開発を行った。OpenCV を利用して Haar-like 特徴とカスケード分類器を組み合わせ顔検出するシステムを作成した。顔検出プログラムによってカメラに映る顔を検出し、検出箇所を赤い矩形で囲う。その矩形の場所や大きさから座標を取得した。ロボットの制御では顔検出で取得した顔座標をもとに常に画面の中央に顔がくるようにロボットを前進,後退,左旋回,右旋回の 4 つの方法で移動させた。顔認識システムを作成する前に複数人の顔データを１人２０枚ずつ撮影した。撮影した顔データの機械学習を行うため照明や角度の違いに影響されにくいという特徴がある FisherFace を用いた。顔認識システムでは顔検出が行なわれた際に学習データから顔を比較し、その結果から人物名と認識信頼度を表示させた。そしてこの 3 つから構成されるシステムを実装し、顔を正確に認識しロボットが動作をするか確かめるため実験を行った。実験では研究室の仲間 7 名を集めそれぞれ 10 回ずつ行なった。実験結果から誤認識はあったもののロボットは hibi と認識され信頼度が高い場合、音を鳴らしロボットを前進させることができ、その他の人物と認識され信頼度が低い場合ロボットを静止させることができた。

以上のように顔検出、顔座標取得、顔座標に基づくロボットの制御、顔認識の一連のシステムについて成功を収めた。しかし顔を識別するのに時間がかかる課題も明らかになった。

この課題について検討を行った。顔を識別するのに時間がかかってしまう点ではカメラから撮影した顔データを無線(wifi)でパソコンに送り、識別を行いその結果を Raspberry Pi に送り返してロボットを動かすことにより時間を短縮で

きると考えた。卒業までにはこの修正を実現させたいと考えている。

この研究の展望として、音声認識、発話機能、表情認識機能など組み合わせていくことによって知能をもつ小型ロボットが作れるのではないかと考えている。

目 次

1. はじめに

本研究は顔認識機能を持つロボットシステムの作成を目的としたものである。顔認識機能とはWebカメラで顔を検出して事前に学習した複数のデータと比較して、その結果から最も似ているものを識別するものである。その結果に基づいてロボットに色々な動作をさせることを考えた。

研究のきっかけはゼミで学んだ顔認識の仕組みと小型ロボットと組み合わせてみると面白いと思ったからである。そのロボットの CPU として Raspberry Pi シリーズで最も処理性能がよかった Raspberry　Pi モデル 2 を利用した。また本研究で作成したロボット本体は DC モーターを２つ備えた車輪付きロボットとした。

本研究では顔検出システム、ロボットの制御、顔認識システムの３つに分けて開発を行った。OpenCV を利用して Haar-like 特徴とカスケード分類器を組み合わせ顔検出するシステムを作成した。顔検出プログラムによってカメラに映る顔を検出し、検出箇所を赤い矩形で囲う。その矩形の場所や大きさから座標を取得した。ロボットの制御では顔検出で取得した顔座標をもとに常に画面の中央に顔がくるようにロボットを前進,後退,左旋回,右旋回の 4 つの方法で移動させた。顔認識システムを作成する前に複数人の顔データを１人２０枚ずつ撮影した。撮影した顔データの機械学習を行うため照明や角度の違いに影響されにくいという特徴がある FisherFace を用いた。顔認識システムでは顔検出が行なわれた際に学習データから顔を比較し、その結果から人物名と認識信頼度を表示させた。そしてこの 3 つから構成されるシステムを実装し、顔を正確に認識しロボットが動作をするか確かめるため実験を行った。実験では研究室の仲間 7 名を集めそれぞれ 10 回ずつ行なった。実験結果から誤認識はあったもののロボットは hibi と認識され信頼度が高い場合、音を鳴らしロボットを前進させることができ、その他の人物と認識され信頼度が低い場合ロボットを静止させることができた。

以上のように顔検出、顔座標取得、顔座標に基づくロボットの制御、顔認識の一連のシステムについて成功を収めた。しかし顔を識別するのに時間がかかる課題も明らかになった。

この課題について検討を行った。顔を識別するのに時間がかかってしまう点ではカメラから撮影した顔データを無線(wifi)でパソコンに送り、識別を行いその結果を Raspberry Pi に送り返してロボットを動かすことにより時間を短縮できると考えた。卒業までにはこの修正を実現させたいと考えている。

この研究の展望として、音声認識、発話機能、表情認識機能など組み合わせ

ていくことによって知能をもつ小型ロボットが作れるのではないかと考えている。

本論文の構成は以下の通りである。第 2 章では本研究で用いた Raspberry Pi モデル 2 と Web カメラの紹介を行い、顔認識システムの構成について説明する。第 3 章ではカメラ側の制御に使用したライブラリ、顔の検出、顔座標の取得とその利用方法について説明する。第 4 章ではモーターを制御するために用いた Raspi Robot Board の紹介とロボット制御について説明する。第 5 章では顔データの収集方法、顔データの機械学習、カメラに写った顔を認識するシステムの作成について説明する。第 6 章では第 3 章、第 4 章、第 5 章で作成したカメラの制御とロボットの制御と顔認識システムを組み合わせ実装し、正確に認識できるか確かめる実験とその実験結果について述べる。第 7 章では実験結果から得られた考察を述べ、展望について議論する。

2. 研究の構成

本研究は、カメラで撮影した画像から誰が映っているのか認識し、その認識結果に応じた動作をするロボットを Raspberry Pi を用いて作成した。このように Raspberry　Pi と Web カメラが本研究の重要な構成要素となっており、それぞれ 2.1 節と 2.2 節で説明する。また、2.3 節では、これらを用いて作成した顔認識システムの構成について説明する。

2.1. Raspberry Pi

Raspberry　Pi とは、ARM プロセッサを搭載したシングルボードコンピュータである。Raspberry Pi にはモデルが 5 種類あるが、本研究では顔認識システムを Raspberry　Pi 上で動作させたかったのでメモリが一番大きく処理性能も一番良い Raspberry　Pi モデル 2 を使用した。

図 2.1.Raspberry　Pi モデル 2

Raspberry　Pi モデル 2 のハード面,ソフトウェア面の二つについて説明する。

2.1.1 ハード面

Raspberry　Pi は名刺サイズの大きさで LAN ポート,USB ポート,microUSB、HDML,3.5m ジャック,GPIO,microSD スロットが備わっている。詳細については表にまとめた。(表 2.1)

表 2.1.Raspberry Pi モデル 2 の仕様

モデル	Raspberry Pi モデル 2
Soc	Broadcom BCM2836
Cpu	ARM Cortex-A7　クアッドコア 900 MHz　　ARMv7　　ARM Cortex-A
GPU	250 MHz / Broadcom VideoCore IV[15] OpenGL ES 2.0 (24 GFLOPS) MPEG-2, VC-1[注釈 1], 1080p30 H.264/MPEG-4 AVC High Profile[13] ハードウェアデコーダ・エンコーダ
メモリ	1GB
USB2.0 ポート	4
映像入力/出力	15 ピン MPI カメラインターフェース/コンポジット RCA
音声入力/出力	IS/3.5mm ジャック,HDML,IS
ストレージ	microSD カード
ネットワーク	LAN9514
低レベル周辺機器	GPIO40 ピン
電源	900mA
電源/ソース	microUSB GPIO/5v
大きさ/質量	85.60mm×56.5mm/4556.5mm/45g
公式に提供される OS	Debian,Fedora,Arch,Linux,RISC OS Windows 10

2.1.2 ソフトウェア面

オペレーティングシステムはRaspbianである。RaspbianはDebian6.0Suqueezeにもとづいており、LXDEデスクトップ環境とMidoriブラウザに加えて、様々なプログラミングツールを利用できる。

本研究で Raspberry Pi2 を使用した理由は価格の安く超小型であるためロボットに組み込みやすいということに加えて、IP アドレスを設定すれば無線で通信を行えるからである。

2.2. 使用したカメラ

本研究で使用したカメラはELECOM UCAM-C0220FEWH(図2.2)である。

図2.2 使用したWebカメラ

Webカメラの主な仕様を表2.1に示す。

表2.2Webカメラの仕様

動作環境	WINDOWS Mac PlayStation3
画素数	200万画素
受像素子	1/5インチCMOSセンサー
最大解像度	1600×1200ピクセル
最大フレームレート	～640×480ピクセル時：30fps 1600×1200ピクセル時：4fps
色数/質量/ケーブル長	約1677万色(24bit)/62g(ケーブルを含まず)/1.5m
インターフェイス	USB2.0
外形寸法	幅56.0×奥行50.0×高さ41.0mm

Web カメラを使用した理由としては Raspberry Pi に用いることができる（UVC 仕様）ことと、小型で値段も安く誰でも手軽に手に入れられるからである。

2.3. 顔認識システムの構成

本研究で設計した顔認識システムは web カメラから顔の検出を行い、顔の座標を取得し、その座標からカメラに写っている顔が中央にくるようにロボットを制御する。そして顔の識別により、知っている人なら挨拶をし、知らない人なら逃げるというようなロボットに異なる行動をとらせるものである。このようなロボットの制御については 4 章で述べる。

3. OpenCV を利用した顔検出

本章では顔認識を行うためのカメラによる顔の検出、および顔座標の取得方法について述べる。

3.1. 使用したライブラリ

Web カメラから顔の検出を行うために使用したライブラリ OpenCV (ver 2.4)について紹介する。

OpenCV とは画像処理、構造解析,モーション解析と物体追跡,パターン認識,機械学習などの機能を持つインテル社が開発・公開したオープンソースのコンピュータビジョン向けのライブラリである。

3.2. 顔の検出

顔の検出は二つのアルゴリズムの組み合わせが主流となっている。一つは特徴の選び方でありもう一つは分類器の選び方である。本研究では特徴として Haar-like 特徴を選び分類器はカスケード分類器を選んだ。Haar-like 特徴とは、Haar 氏の考えた Haar ウエーブレットに似ているところから、この名前が付けられたものである。これは白黒の部分に分けられた矩形をテンプレートマッチングさせ、局所的にエッジ、直線、コーナーなどを検出し、これらを特徴とするものである。

カスケード分類器とは Haar-like 特徴などを使い、数多くの正しいサンプルと偽サンプルで訓練を行い、ブーストされた分類器をカスケード状に直列に並べて識別を行うためのアルゴリズムである。

Haar-like 特徴を用いカスケード分類器で顔が検出できるかどうか、画像処理研究でよく用いられる静止画像 Lena.jpg に対し、藤井ら(2014)のサンプルコード(プログラム 5_6)において、カメラ読み込みを静止画読み込みに変更して使用したものが図 3.1.である。図の赤枠で示したのが検出された顔の領域であり、この例では静止画であるが、顔が検出されていることがわかる。

図 3.1.lena 画像に対して顔検出した結果

静止画ではなく web カメラの画像から顔を検出するにはサンプルコード 5_6 をそのまま使用すれば可能である。なお、顔の検出においては画面のサイズを縦 480×横 640 としている。

3.3. 顔の座標を取得

顔の座標を取得するプログラムについて一部抜粋し説明する。
Object では入力画像にある指定の顔を検出し、その位置座標を返している。cascade.detectMultiScale 関数の第 1 引数には対象画像を指定し、以降の引数は Haar 特徴分類器のパラメータである。scaleFactor 引数は Haar 特徴をマッチングさせるときの拡大縮小率をどれだけの割合で変更するかのスケーリングファクタである。minNeighbors 引数は近傍にある検出した領域のまとめるときの、近傍をしているものである。引数は処理方法で、cv2.CASCADE__SCALE__IMAGE を指定すると、画像領域をスケーリングして検出を行う。

```
        #       Haar-like 特徴分類器による顔検出
        objects = cascade.detectMultiScale(frame, scaleFactor=1.1,
    minNeighbors=3,
    flags=cv2.CASCADE_SCALE_IMAGE, minSize=(0, 0))
        for x, y, w, h in objects:          #       検出数の繰り返し
                #       検出部に四角枠描画
          cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 4, cv2.CV_AA, 0)
        cv2.imshow (windowName_1, frame) # 検出結果表示
```

顔検出プログラムでは、検出された顔の場所と大きさの情報は次のように出力される。図 3.2 に Lena 画像（図 3.1）で検出された顔だけを示す。ここで、顔検出を示す赤い矩形は、左上部の座標（図 3.2 において(x,y)で示す）w とは検出箇所の幅であり h は検出箇所の高さである。

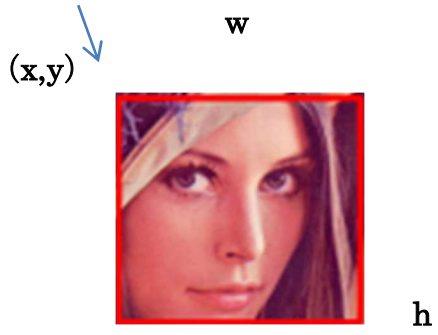

図 3.2. 顔検出時の座標

この座標データに基づき常に顔が画面の中央にくるようにロボットを操作すれば、常にロボットは人と向き合うようにできる。ロボットの制御については４章で説明する。

4. ロボットの制御

本章では web カメラから顔検出によって取得した座標からロボットを制御する方法について述べる。

4.1. 使用した機器

Raspberry Pi をロボット化するためのボードキット「RaspiRobot Board」(図 4.1.)を使用した。これより DC モーター2 個の制御が可能となった。

図 4.1.RaspiRobot Board

4.2. ロボット制御

ロボット制御についてのプログラムを次のように設定した。

ロボット制御プログラム

```
import raspirobotboard as rb
import RPi.GPIO as GPIO
rr = rb.RaspiRobot()

----------中途省略----------

if x < 100：
        rr.right(0.1)
        rr.stop()
    elif x > 450：
```

```
            rr.left(0.1)
            rr.stop()
        elif h > 250：
            rr.reverse(0.1)
            rr.stop()
        elif h < 80：
            rr.forward(0.1)
            rr.stop()
    else:
        pass
        rr.right(0.5)
        rr.stop()
        time.sleep(20)
```

顔検出によって取得した顔座標をもとにして、常に画面の中央に顔がくるようにロボットを前進,後退,左旋回,右旋回の 4 つの方法で移動させる。動く時間はいずれも 0.1 秒間に設定した。もしもカメラに顔が検出されない場合は 0.5 秒間右旋回をさせる。この時間を設定した理由としては顔が検出されない場合ロボットのカメラを毎回違う方向にカメラを向けるため 45° 回転になるようにしたからである。

モーターを回転させるためには二つのライブラリ RPi.GPIO.Pyserial をインストールする必要がある。このライブラリをインストールすることによってモーターを制御することができる。モーターを制御するために.RaspiRobotBoard を RaspberryPi に装着し RaspiRobotBoard に DC モーターと電源を繋がなければならない。その際に電源を供給するためにモバイルバッテリーを使用した。次に python を起動しコマンドを送れば操作可能である。

5. 顔認識システムの作成

本章では顔検出後に、顔の識別に用いる「顔認識」システムについて述べる。

5.1. 顔データの収集

顔の識別を行うには複数人の顔データが必要である。顔データを集めるために研究室のメンバーに協力してもらい 7 名の顔を 1 人 20 枚ずつ撮影した。この時に撮影した顔はすべて正面顔で統一してある。このプログラムはカメラからキャプチャを行い Haar-like 特徴量により顔、目、鼻、口の検出がすべてできたものだけ保存するものである（付録 2)。

5.2. 顔データの機械学習

5.1 節で述べたプログラムにより撮影・収集した画像を個人別にフォルダに分けた後に機械学習システムを使用する。OpenCV では顔認識の手法として、フィッシャーの線形判別分析(LDA)を使ったフィッシャー顔(FisherFace)、主成分分析(PCA)を用いた固有顔(EigenFace)があるが、本研究では FisherFace を用いた。その理由としては EigenFace と比較して FisherFace は照明や角度の違いに影響されにくいという特徴があり、FisherFace の方が予備実験で性能がよかったからである。

顔データの機械学習システムでは、初めに目の位置を水平になるように回転させ両目の間隔を一定に揃える。そして白黒変換して明るさの調整を行う。次に顎の形が四角ではないため背景が映り込んでしまうので背景をカットする。この作業をすることによってよりよい顔の学習データが得られる。この学習結果は reconizerSave.sav ファイルとして出力される。(付録 3)

5.3. カメラに写った顔を認識するシステム

カメラに写った顔を認識するシステムは、カメラで顔検出を行い(5.1 節参照)、機械学習により作成したデータベース(5.2 節参照)を用いて、検出した顔が誰かを識別する（顔を認識する）ものである。そもそも識別のための学習に用いたデータは人物名ではなく数(ID)が割り振られているので、その ID から人物名を求めて認識結果に表示されるようにした。また合わせて認識信頼度も表示するようにした。(
信頼度の数値が 0 に近ければ信頼度は低く 800 以上であれば認識に自信が持てる。

なお、当然のことながら、このシステムをでは(引数として)学習結果のファイルを指定しなければならない。

図 5.1 にシステムを使用した時に表示される画面の例を示す。この画面では、人物名と認識信頼度が表示されている。また誤認識も一部含まれている(imamura の表示がある)が、正解である hibi の表示では高い認識信頼度が示され、顔認識ができていることがわかる

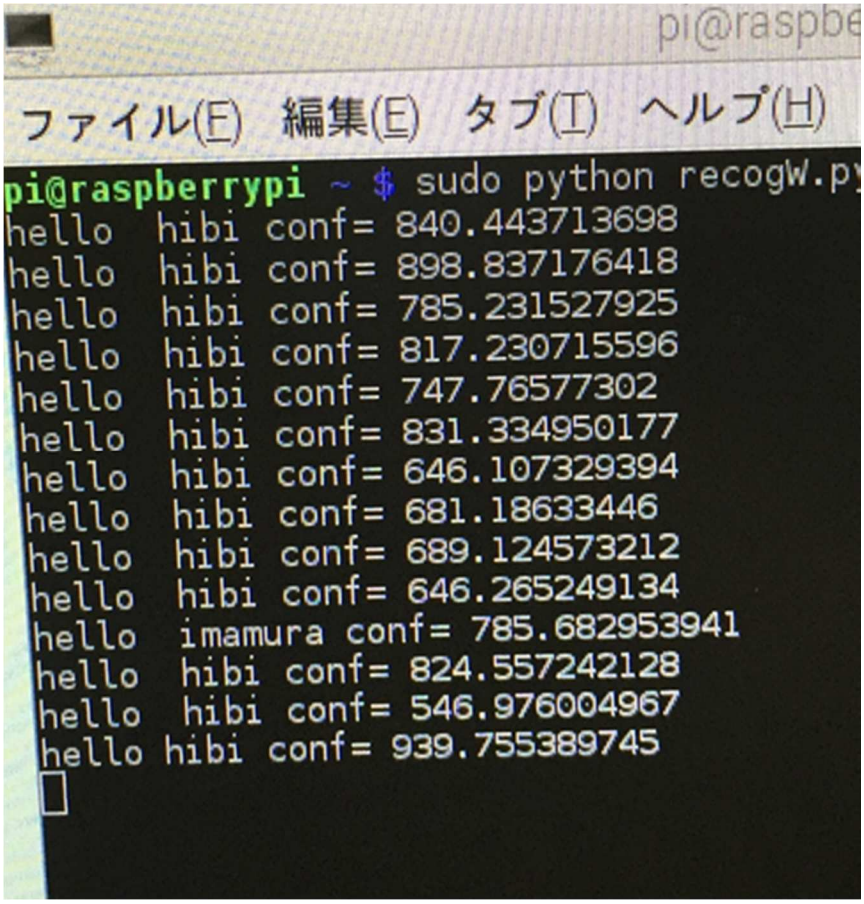

図 5.1.顔認識システムの画面表示例

6. システムの実証実験

本章では、5 章で説明した顔認識システムを Raspberry Pi 上で動かし、その認識率と識別結果からロボットに異なる動作をさせることについて述べる。

6.1. 実験概要

本システムの作成者である日比を認識し、かつ認識精度が十分高ければ、ロボットは音を出して近づく、それ以外の人物を認識した場合はその名前を表示だけで動かない、というシステムを作成する。そして、顔の学習 (5 章参照)に使用した協力者 7 名に対し、それぞれ 10 回ずつ作成システムにより顔認識を行う。そして、このシステムが目的通り機能するか確かめる、という実験を行う。

6.2.ロボットのプログラム

顔認識を行った識別結果からロボットに異なる動作をさせたプログラムを一部抜粋し説明する。

```
recognizer = cv2.createFisherFaceRecognizer()
recognizer.load(savedFile)

# targetFile = sys.argv[2]
# target = cv2.imread(targetFile)

src = cv2.VideoCapture(0)
if not src.isOpened():
        print u'映像が取得できません。'
        sys.exit()

windowName = 'Camera'
cv2.namedWindow(windowName,cv2.WINDOW_NORMAL)
Hito                                                                    =
[None,"hibi","pariasuka","kato","nakagawa","imamura","sibuya","nisiwaki"]

while True:
  retval, frame = src.read()
  if frame is None:
        break
```

```
    cv2.imshow(windowName,frame)
    key =    cv2.waitKey(50)

    objects = cascade.detectMultiScale(frame, scaleFactor=1.1,
                        minNeighbors=4,
flags=cv2.CASCADE_FIND_BIGGEST_OBJECT,
                        minSize=(30, 30))

    if (objects is not None) and (len(objects) >= 1):
        x,y,w,h = objects[0]
        face = normalizePict(frame)
        if face is not None:
            cand, conf=recognizer.predict(face)
            if (conf < 900):
                    print "hello ",Hito[cand], "conf=",conf
            elif (Hito[cand] == "hibi" ):
                    print "HALLO",Hito[cand],"conf=",conf
                    buzzer.start(50)
                    time.sleep(1)
                    buzzer.stop()
                    rr.forward(1)
                    rr.stop()
```

このプログラムでは顔を検出し学習データから人物を割り当て、名前を表示させた後、その識別結果の信頼度を出す。もしも信頼度が 900 未満ならば「hello 人物名 信頼度 」と表示されるのみであり、もしも信頼度が 900 以上であり人物名が日比(hibi)と表示されればブザーを鳴らし前進する。このように顔の認識結果からロボットに異なる動作をさせることができる。

6.3. 実験結果

顔の検出を行い座標から画面の中央に顔がくるようにモーターを制御させ識別する一連の動作を行うことに成功した。ただ Raspberry Pi 上で顔は検出するが識別するのに時間が 5 秒以上かかった。そして所有者である日比を認識し認識精度が高い場合に音を出して前進させることにも成功した。(図 6.1)

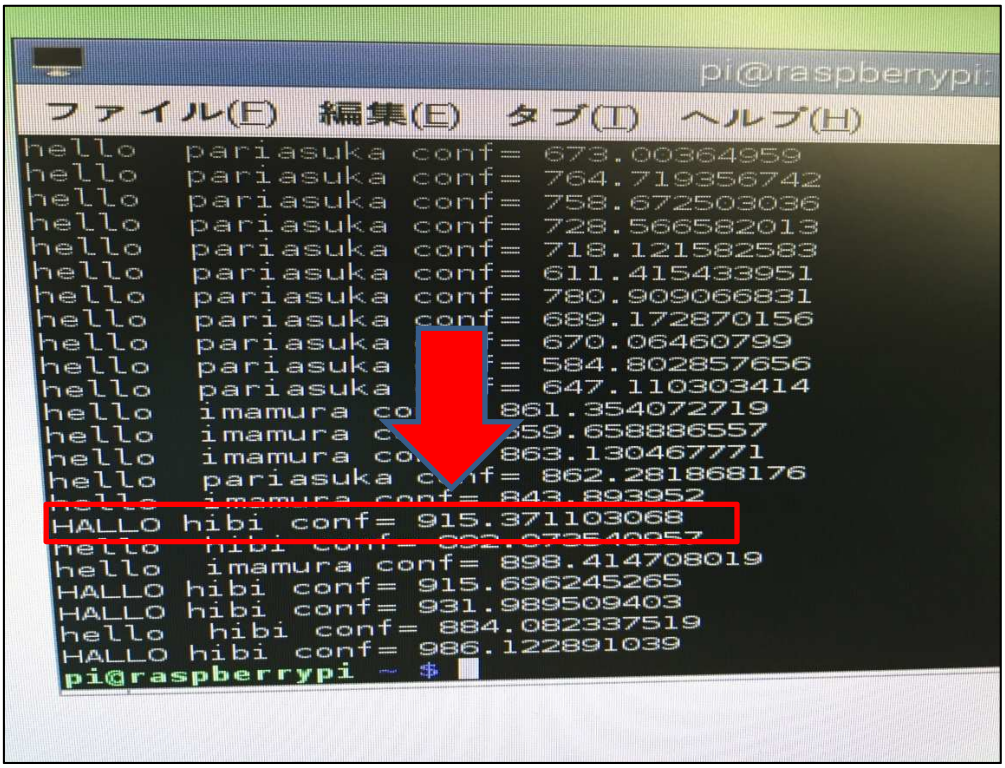

図 6.1.顔認識実験

顔の認識結果を表 6.2.にした。この表から認識率には個人差があることがわかった。全体としてそして認識率は 93%であった。また誤認識においては、すべてのケースで「今村」であった。

表 6.2.認識実験結果

入力/出力	日比	パリアス	加藤	今村	中川	西脇	渋谷
日比	7	0	0	3	0	0	0
パリアス	0	7	0	3	0	0	0
加藤	0	0	9	1	0	0	0
今村	0	0	0	10	0	0	0
中川	0	0	0	0	10	0	0
西脇	0	0	0	0	0	10	0
渋谷	0	0	0	0	0	0	10

7. 考察と展望

実験の結果から得られた考察を述べ、次に将来的な展望を述べる。

7.1. 考察

実験結果から誤認識が起きてしまうことがわかった。学習した画像データを増やすことによって認識率があがるのではないかと考えられる。

顔を識別するのに時間がかかってしまう点ではカメラから撮影した顔データを無線(wifi)でパソコンに送り、識別を行いその結果を Raspberry Pi に送り返してロボットを動かす改善案が考えられる。

7.2. まとめと展望

音声認識機能を利用することにより言葉でロボットを操作したり会話ができるようになりシステムを拡張することが考えられる。

本研究ではカメラで顔を検出し、認識システムを用いて誰が写っているか判別できるようになった。そしてロボットとも連動させることができるようになった。

将来的な展望として音声認識機能と表情認識機能を組み合わせることにより人工知能を持つペットロボットで役に立つであろう。

参考文献

桑井博之、豊沢聡、永田雅人（2014）『実践 OpenCV2.4 for python』. カットシステム.

石立 喬 （2015）「OpenCV と Visual C++による画像処理と認識」 http://homepage3.nifty.com/ishidate/opencv_13/opencv_13.html (2016 年 1 月 3 日アクセス)

Bikramjot Singh Hanzra (2015) 「Face Recognition using Python and OpenCV」 http://hanzratech.in/2015/02/03/face-recognition-using-opencv.html (2016 年 1 月 3 日アクセス)

謝辞

本研究を行うにあたり、ご指導を頂いた白井英俊教授に感謝致します。また、日頃の議論等でアドバイスをくれたゼミの同期の皆さまにも感謝致します。

付録１　顔検出プログラム

```
# 桑井ら(2014）に基づくプログラム
import sys
import cv2.cv as cv             #       旧 OpenCV ライブラリ
import cv2                      #       OpenCV ライブラリ
import numpy as np              #       Numpy ライブラリ

windowName_1 = u'顔検出'
cv2.namedWindow(windowName_1)           # 表示ウィンドウの作成

HAAR_FILE = 'haarcascade_frontalface_default.xml'
cascade = cv2.CascadeClassifier(HAAR_FILE)  # 特徴ファイルの読み込み
if cascade.empty():
        print u'特徴ファイルが読み込めません。'
        sys.exit()

src = cv2.VideoCapture(0)               #       映像取得（カメラ）
if not src.isOpened():          #       映像が読み込めない場合
        print u'映像が取得できません。'
        sys.exit()

while True:]
        retval, frame = src.read()      #       1 フレーム取得
        if frame is None:
                break
        #       Haar-like 特徴分類器による顔検出検出
        objects = cascade.detectMultiScale(frame, scaleFactor=1.1,
                minNeighbors=3,
                flags=cv2.CASCADE_SCALE_IMAGE, minSize=(0, 0))
        for x, y, w, h in objects:      #       検出数の繰り返し
                #       検出部に四角枠描画
          cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 4, cv2.CV_AA, 0)
        cv2.imshow (windowName_1, frame) # 検出結果表示
        key = cv2.waitKey(33)                   # キー入力待機（33 ms）
        if key == 27:           #       ESC キーを押したとき終了
                break
```

```
cv2.destroyAllWindows()          #       すべての表示ウィンドウの破棄
src.release()                    # ビデオファイル（キャプチャ機器）を閉じる
```

付録２　顔データ収集システム

```
# -*- coding: utf-8 -*
# © H. Sirai 2015
import sys, time
import cv2.cv as cv             #       旧 OpenCV ライブラリ
import cv2                      #       OpenCV ライブラリ
import numpy as np              #       Numpy ライブラリ

if (len(sys.argv) < 2):
    print "Usage: python collectFaces.py yourname"
    sys.exit()


FileName = sys.argv[1]
Seq = 1
FileExt = ".bmp"

faceColor = (255, 255, 255) # white
leftEyeColor = (0, 0, 255) # red
rightEyeColor = (0, 255, 255) # yellow
noseColor = (0, 255, 0) # green
mouthColor = (255, 0, 0) # blue


def outlineRect(image, rect, color):
    if rect is None:
        return
    x, y, w, h = rect
    cv2.rectangle(image, (x, y), (x+w, y+h), color)


def widthHeightDividedBy(image, divisor):
    """Return an image's dimensions, divided by a value."""
    h, w = image.shape[:2]
    return (w/divisor, h/divisor)
```

```
def detectOneObject(classifier, image, rect,
                                imageSizeToMinSizeRatio):
        flags = cv2.cv.CV_HAAR_SCALE_IMAGE
        scaleFactor = 1.2
        minNeighbors = 2
        x, y, w, h = rect
        minSize                =                widthHeightDividedBy(image,
imageSizeToMinSizeRatio)
        subImage = image[y:y+h, x:x+w]
        subRects = classifier.detectMultiScale(
            subImage, scaleFactor, minNeighbors, flags, minSize)
        if len(subRects) == 0:
            return None
        subX, subY, subW, subH = subRects[0]
        return (x+subX, y+subY, subW, subH)


windowName_1 = 'Camera'
cv2.namedWindow(windowName_1)          #       表示ウィンドウの作成

cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eyecascade = cv2.CascadeClassifier('haarcascade_eye.xml')
nosecascade = cv2.CascadeClassifier('haarcascade_mcs_nose.xml')
mouthcascade = cv2.CascadeClassifier('haarcascade_mcs_mouth.xml')

if (cascade.empty() or eyecascade.empty() or nosecascade.empty() or
mouthcascade.empty()):
        print u'特徴ファイルが読み込めません。'
        sys.exit()


src = cv2.VideoCapture(0)                      #       映像取得（カメラ）
if not src.isOpened():                         #       映像が読み込めない場
合
        print u'映像が取得できません。'
        sys.exit()
```

```
while True:
        retval, frame = src.read()                  #        1 フレーム取得
        if frame is None:
                break
        #        Haar-like 特徴分類器による顔検出検出
        objects      =      cascade.detectMultiScale(frame,      scaleFactor=1.1,
minNeighbors=4,
                  flags=cv2.CASCADE_FIND_BIGGEST_OBJECT,
minSize=(30, 30))
          if   (objects is not None) and (len(objects) == 1):
              face = objects[0]
              x, y, w, h = face
              # an eye in the upper-left part
              searchRect = (x+w/7, y, w*2/7, h/2)
              leftEye_o = detectOneObject(eyecascade, frame, searchRect, 64)
              # an eye in the upper-right part
              searchRect = (x+w*4/7, y, w*2/7, h/2)
              rightEye_o  =  detectOneObject(eyecascade,  frame,  searchRect,
64)
              # Seek a nose in the middle part of the face.
              searchRect = (x+w/4, y+h/4, w/2, h/2)
              nose_o = detectOneObject(nosecascade,   frame, searchRect, 32)
              # Seek a mouth in the lower-middle part of the face.
              searchRect = (x+w/6, y+h*2/3, w*2/3, h/3)
              mouth_o = detectOneObject(mouthcascade, frame, searchRect,
16)
              if ((face is not None) and (leftEye_o is not None) and
                    (rightEye_o is not None) and (nose_o is not None) and
                    (mouth_o is not None)):
                       fname = FileName+str(Seq)+FileExt
                       cv2.imwrite(fname, frame)
                       print fname," was generated¥n"
                       Seq += 1
              outlineRect(frame, face, faceColor)
```

```
            outlineRect(frame, leftEye_o, leftEyeColor)
            outlineRect(frame, rightEye_o,rightEyeColor)
            outlineRect(frame, nose_o, noseColor)
            outlineRect(frame, mouth_o, mouthColor)
        cv2.imshow (windowName_1, frame)        #       検出結果表示
        key = cv2.waitKey(33)                   #       キー入力待機（33 ms）
        if key == 27:                           #       ESC キーを押したとき
終了
            break
        time.sleep(1)

cv2.destroyAllWindows()             # すべての表示ウィンドウの破棄
src.release()                   # ビデオファイル（キャプチャ機器）を閉じる
```

付録 3　顔データの学習プログラム

```
# -*- coding: utf-8 -*
# © H. Sirai 2015
# training Data

import sys, os, time, math
import cv2.cv as cv             #       旧 OpenCV ライブラリ
import cv2                      #       OpenCV ライブラリ
import numpy as np              #       Numpy ライブラリ

if (len(sys.argv) < 2):
    print "Usage: python trainFaces.py pictpath"
    sys.exit()

path = sys.argv[1]
FileExt = ".bmp"

subdirs = os.listdir(path)
for x in subdirs:
    try:
        int(x)
    except:
        print "directory name under ",path," should be numbers"
        sys.exit()

faceColor = (255, 255, 255) # white
leftEyeColor = (0, 0, 255) # red
rightEyeColor = (0, 255, 255) # yellow
noseColor = (0, 255, 0) # green
mouthColor = (255, 0, 0) # blue


def outlineRect(image, rect, color):
    if rect is None:
        return
```

```
    x, y, w, h = rect
    cv2.rectangle(image, (x, y), (x+w, y+h), color)


def widthHeightDividedBy(image, divisor):
    """Return an image's dimensions, divided by a value."""
    h, w = image.shape[:2]
    return (w/divisor, h/divisor)

def detectOneObject(classifier, image, rect,
                                    imageSizeToMinSizeRatio):
        flags = cv2.cv.CV_HAAR_SCALE_IMAGE
        scaleFactor = 1.2
        minNeighbors = 2
        x, y, w, h = rect
        minSize                =                widthHeightDividedBy(image,
imageSizeToMinSizeRatio)
        subImage = image[y:y+h, x:x+w]
        subRects = classifier.detectMultiScale(
            subImage, scaleFactor, minNeighbors, flags, minSize)
        if len(subRects) == 0:
            return None
        subX, subY, subW, subH = subRects[0]
        return (x+subX, y+subY, subW, subH)

def preProcessFace(image, face, rightEye, leftEye):
    x, y, w, h = face
    FACE_ELLIPSE_CY = 0.4
    FACE_ELLIPSE_W = 0.50
    FACE_ELLIPSE_H = 0.80
    leftEye_X = leftEye[0]+leftEye[2]/2
    rightEye_X = rightEye[0]+rightEye[2]/2
    leftEye_Y = leftEye[1]+leftEye[3]/2
    rightEye_Y =    rightEye_o[1]+rightEye[3]/2
    # Get the center between two eyes
    eyesCenter_X = (leftEye_X+ rightEye_X) * 0.5
```

```
    eyesCenter_Y= (leftEye_Y + rightEye_Y) * 0.5
    # Get the angle between the 2 eyes.
    dy = (rightEye_Y - leftEye_Y)
    dx = (rightEye_X - leftEye_X)
    len = (dx*dx + dy*dy)**0.5
    # Convert Radians to Degrees.
    angle = math.atan2(dy, dx) * 180.0/math.pi
    #
    # Hand measurements shown that the left eye center should
    # ideally be roughly at (0.16, 0.14) of a scaled face image.
    DESIRED_LEFT_EYE_X = 0.16
    DESIRED_LEFT_EYE_Y = 0.14;
    DESIRED_RIGHT_EYE_X = (1.0 -    0.16)
    # Get the amount we need to scale the image to be the desired
    # fixed size we want.
    DESIRED_FACE_WIDTH = 70
    DESIRED_FACE_HEIGHT = 70
    desiredLen = (DESIRED_RIGHT_EYE_X - 0.16)
    scale = desiredLen * DESIRED_FACE_WIDTH / len
    #
    # Get the transformation matrix for the desired angle & size.
    rot_mat = cv2.getRotationMatrix2D((eyesCenter_X,eyesCenter_Y) , angle,
scale)
    # Shift the center of the eyes to be the desired center.
    ex = DESIRED_FACE_WIDTH * 0.5 - eyesCenter_X
    ey = DESIRED_FACE_HEIGHT * DESIRED_LEFT_EYE_Y -
eyesCenter_Y
    rot_mat[0][2] += ex;
    rot_mat[1][2] += ey;
    # Transform the face image to the desired angle & size & position!
    # Also clear the transformed image background to a default grey.
    gray=cv2.cvtColor(image,    cv2.COLOR_BGRA2GRAY)
    warped_size = ( DESIRED_FACE_WIDTH, DESIRED_FACE_HEIGHT)
    warped= cv2.warpAffine(gray, rot_mat, warped_size)
    # cv2.imshow("warped", warped)
    #    Do it seperately for the left and right sides of the face.
```

```
    warped=equalizeLeftAndRightHalves(warped)
    # cv2.imshow("warped",warped)
    #
    # Use the "Bilateral Filter" to reduce pixel noise by smoothing the image,
    # but keeping the sharp edges in the face.
    filtered = cv2.bilateralFilter(warped, 0, 20.0, 2.0);
    # cv2.imshow("filtered", filtered)
    # print filtered.shape
    #
    # Filter out the corners of the face, since we mainly just care about the middle
    # Draw a filled ellipse in the middle of the face-sized image.
    # faceCenter = ( DESIRED_FACE_WIDTH/2, round(DESIRED_FACE_HEIGHT *  FACE_ELLIPSE_CY)
    # size = ( round(DESIRED_FACE_WIDTH * FACE_ELLIPSE_W), round(DESIRED_FACE_HEIGHTt* FACE_ELLIPSE_H) )
    # mask=np.zeros(warped_size, bool)
    # mask=cv2.ellipse(mask, faceCenter, size, 0, 0, 360, 255, cv2.cv.CV_FILLED)
    # ERROR occured the above...
    # dstImg = np.ones(warped_size,np.uint8)
    # dstImg *= 128  # gray
    # dstImg = np.copyTo(filtered, where=mask)  # Copies non-masked pixels from filtered to dst
    # cv2.imshow("dstImg", dstImg)
    # return mask
    # masking  --- y = a(x-b)
    const = int(DESIRED_FACE_WIDTH*FACE_ELLIPSE_CY/2)
    constY = const
    ypos = DESIRED_FACE_HEIGHT-1
    xpos = DESIRED_FACE_WIDTH-1
    # print const, ypos, yposH, xpos
    for i in range(const):
        for j in range(constY):
            filtered[ypos-j][i] = 128 # gray
            filtered[ypos-j][xpos-i] = 128
```

```
        constY -= 1
    return filtered

def equalizeLeftAndRightHalves(faceImg):
    # It is common that there is stronger light from one half of the face than the other.
    # In that case, if you simply did histogram equalization on the whole face then it
    # would make one half dark and one half bright. So we will do histogram equalization
    # separately on each face half, so they will both look similar on average.
    # But this would cause a sharp edge in the middle of the face, because
    # the left half and right half would be suddenly different. So we also histogram
    # equalize the whole image, and in the middle part we blend the 3 images together
    # for a smooth brightness transition.
    w, h = faceImg.shape
    #
    # 1) First, equalize the whole face.
    wholeFace = cv2.equalizeHist(faceImg)
    #
    # 2) Equalize the left half and the right half of the face separately.
    midX = w/2;
    leftSide = faceImg[0:h,0:midX]
    rightSide = faceImg[0:h,midX:]
    leftSide=cv2.equalizeHist(leftSide)
    rightSide=cv2.equalizeHist(rightSide)
    #
    # 3) Combine the left half and right half and whole face together,
    # so that it has a smooth transition.
    for y in range(h):
        for x in range(w):
            if (x < w/4):          # Left 25%: just use the left face.
                v = leftSide[y][x]
            elif (x < w*2/4):   # Mid-left 25%: blend the left face & whole
```

```
face.
                    lv = leftSide[y][x]
                    wv = wholeFace[y][x]
                    # Blend more of the whole face as it moves further right
along the face.
                    f = (x - w*1/4) / (w*0.2)
                    v = round((1.0 - f) * lv + f * wv)
                elif (x < w*3/4):  # Mid-right 25%: blend the right face & whole
face.
                    rv = rightSide[y][x-midX]
                    wv = wholeFace[y][x]
                    # Blend more of the right-side face as it moves further right
along the face.
                    f = (x - w*2/4) / (w*0.25)
                    v = round((1.0 - f) * wv + f * rv)
                else：      # Right 25%: just use the right face.
                     v = rightSide[y][x-midX]
                faceImg[y][x] = v
    #
    return faceImg


# files = os.listdir(path+"/"+name)
# targets = [path+"/"+name+"/"+x for x in files if x.endswith(FileExt)]

windowName_1 = 'FileView'
cv2.namedWindow(windowName_1)            #        表示ウィンドウの作成

cascade    =    cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
        # 特徴ファイルの読み込み
if cascade.empty():
    print u'特徴ファイルが読み込めません。'
    sys.exit()

eyecascade = cv2.CascadeClassifier('haarcascade_eye.xml')
nosecascade = cv2.CascadeClassifier('haarcascade_mcs_nose.xml')
```

```
mouthcascade = cv2.CascadeClassifier('haarcascade_mcs_mouth.xml')

# src = cv2.VideoCapture(0)                    #        映像取得（カメラ）

preprocessedFaces = []  # 前処理済みの顔
labels = [] # ラベル

for subdir in subdirs:
    image_path= os.path.join(path, subdir)
    files = [file for file in os.listdir(image_path) if file.endswith(FileExt)]
    for f in files:
        frame = cv2.imread(os.path.join(image_path,f))
       #        Haar-like 特徴分類器による顔検出検出
        try:
          objects = cascade.detectMultiScale(frame, scaleFactor=1.1,
minNeighbors=4,          flags=cv2.CASCADE_FIND_BIGGEST_OBJECT,
minSize=(30, 30))
          if (objects is not None) and (len(objects) == 1):
            face = objects[0]
            x, y, w, h = face
            # an eye in the upper-left part
            searchRect = (x+w/7, y, w*2/7, h/2)
            leftEye_o = detectOneObject(eyecascade, frame, searchRect, 64)
            # an eye in the upper-right part
            searchRect = (x+w*4/7, y, w*2/7, h/2)
            rightEye_o = detectOneObject(eyecascade, frame, searchRect,
64)
            # Seek a nose in the middle part of the face.
            searchRect = (x+w/4, y+h/4, w/2, h/2)
            nose_o = detectOneObject(nosecascade,  frame, searchRect, 32)
            # Seek a mouth in the lower-middle part of the face.
            searchRect = (x+w/6, y+h*2/3, w*2/3, h/3)
            mouth_o = detectOneObject(mouthcascade, frame, searchRect,
16)
            if ((face is not None) and (leftEye_o is not None) and
                  (rightEye_o is not None) and (nose_o is not None) and
```

```
                    (mouth_o is not None)):
                        print f, rightEye_o[0]+rightEye_o[2]/2,
rightEye_o[1]+rightEye_o[3]/2,
                        print leftEye_o[0]+leftEye_o[2]/2,
leftEye_o[1]+leftEye_o[3]/2,
                        print nose_o[0]+nose_o[2]/2, nose_o[1]+nose_o[3]/2
                        warped=preProcessFace(frame, face, rightEye_o,
leftEye_o)
                        preprocessedFaces.append(warped) # add a new face
                        labels.append(int(subdir))
                        cv2.imshow("warped", warped);
                        #
                        outlineRect(frame, face, faceColor)
                        outlineRect(frame, leftEye_o, leftEyeColor)
                        outlineRect(frame, rightEye_o,rightEyeColor)
                        outlineRect(frame, nose_o, noseColor)
                        outlineRect(frame, mouth_o, mouthColor)
                cv2.imshow (windowName_1, frame)          #       検 出 結
果表示
                cv2.waitKey(300)
            except:
                print "bad file",f

cv2.destroyAllWindows()                 # すべての表示ウィンドウの破棄

recognizer = cv2.createFisherFaceRecognizer()   # EigenFace or FisherFace
recognizer.train(preprocessedFaces,np.array(labels))

recognizer.save('recognizerSaveF.sav')
print "recognizedSaveF.sav has been generated."

# import pickle
# F=open('dumpFile.txt','w')
# pickle.dump([preprocessedFaces,labels],F)
# F.close()
```

付録４　顔認識プログラム

```
# -*- coding: utf-8 -*

# recognize the face in video

import sys, os, time, math
import cv2.cv as cv             #       旧 OpenCV ライブラリ
import cv2                      #       OpenCV ライブラリ
import numpy as np              #       Numpy ライブラリ
import raspirobotboard as rb
import RPi.GPIO as GPIO rr = rb.RaspiRobot()

GPIO.setmode(GPIO.BCM)
BZ1 = 16
GPIO.setup(BZ1,GPIO.OUT)
buzzer = GPIO.PWM(BZ1,1000)

cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
if cascade.empty():
    print u'特徴ファイルが読み込めません。'
    sys.exit()

eyecascade = cv2.CascadeClassifier('haarcascade_eye.xml')
nosecascade = cv2.CascadeClassifier('haarcascade_mcs_nose.xml')
mouthcascade = cv2.CascadeClassifier('haarcascade_mcs_mouth.xml')

def widthHeightDividedBy(image, divisor):
    """Return an image's dimensions, divided by a value."""
    h, w = image.shape[:2]
    return (w/divisor, h/divisor)

def detectOneObject(classifier, image, rect,
                                imageSizeToMinSizeRatio):
        flags = cv2.cv.CV_HAAR_SCALE_IMAGE
        scaleFactor = 1.2
```

```
        minNeighbors = 2
        x, y, w, h = rect
        minSize                =                widthHeightDividedBy(image,
imageSizeToMinSizeRatio)
        subImage = image[y:y+h, x:x+w]
        subRects = classifier.detectMultiScale(
            subImage, scaleFactor, minNeighbors, flags, minSize)
        if len(subRects) == 0:
            return None
        subX, subY, subW, subH = subRects[0]
        return (x+subX, y+subY, subW, subH)

def preProcessFace(image, face, rightEye, leftEye):
    x, y, w, h = face
    FACE_ELLIPSE_CY = 0.4
    FACE_ELLIPSE_W = 0.50
    FACE_ELLIPSE_H = 0.80
    leftEye_X = leftEye[0]+leftEye[2]/2
    rightEye_X = rightEye[0]+rightEye[2]/2
    leftEye_Y = leftEye[1]+leftEye[3]/2
    rightEye_Y =    rightEye[1]+rightEye[3]/2
    # Get the center between two eyes
    eyesCenter_X = (leftEye_X+ rightEye_X) * 0.5
    eyesCenter_Y= (leftEye_Y + rightEye_Y) * 0.5
    # Get the angle between the 2 eyes.
    dy = (rightEye_Y - leftEye_Y)
    dx = (rightEye_X - leftEye_X)
    len = (dx*dx + dy*dy)**0.5
    # Convert Radians to Degrees.
    angle = math.atan2(dy, dx) * 180.0/math.pi
    #
    # Hand measurements shown that the left eye center should
    # ideally be roughly at (0.16, 0.14) of a scaled face image.
    DESIRED_LEFT_EYE_X = 0.16
    DESIRED_LEFT_EYE_Y = 0.14;
    DESIRED_RIGHT_EYE_X = (1.0 -    0.16)
```

```
    # Get the amount we need to scale the image to be the desired
    # fixed size we want.
    DESIRED_FACE_WIDTH = 70
    DESIRED_FACE_HEIGHT = 70
    desiredLen = (DESIRED_RIGHT_EYE_X - 0.16)
    scale = desiredLen * DESIRED_FACE_WIDTH / len
    #
    # Get the transformation matrix for the desired angle & size.
    rot_mat = cv2.getRotationMatrix2D((eyesCenter_X,eyesCenter_Y) , angle,
scale)
    # Shift the center of the eyes to be the desired center.
    ex = DESIRED_FACE_WIDTH * 0.5 - eyesCenter_X
    ey = DESIRED_FACE_HEIGHT * DESIRED_LEFT_EYE_Y -
eyesCenter_Y
    rot_mat[0][2] += ex;
    rot_mat[1][2] += ey;
    # Transform the face image to the desired angle & size & position!
    # Also clear the transformed image background to a default grey.
    gray=cv2.cvtColor(image,  cv2.COLOR_BGRA2GRAY)
    warped_size = ( DESIRED_FACE_WIDTH, DESIRED_FACE_HEIGHT)
    warped= cv2.warpAffine(gray, rot_mat, warped_size)
    # cv2.imshow("warped", warped)
    #  Do it seperately for the left and right sides of the face.
    warped=equalizeLeftAndRightHalves(warped)
    # cv2.imshow("warped",warped)
    #
    # Use the "Bilateral Filter" to reduce pixel noise by smoothing the image,
    # but keeping the sharp edges in the face.
    filtered = cv2.bilateralFilter(warped, 0, 20.0, 2.0);
    # cv2.imshow("filtered", filtered)
    # print filtered.shape
    #
    # Filter out the corners of the face, since we mainly just care about the
middle
    # Draw a filled ellipse in the middle of the face-sized image.
    #       faceCenter      =       (       DESIRED_FACE_WIDTH/2,
```

```
round(DESIRED_FACE_HEIGHT *  FACE_ELLIPSE_CY)
    # size = ( round(DESIRED_FACE_WIDTH * FACE_ELLIPSE_W),
round(DESIRED_FACE_HEIGHTt* FACE_ELLIPSE_H) )
    # mask=np.zeros(warped_size, bool)
    # mask=cv2.ellipse(mask, faceCenter, size, 0, 0, 360, 255,
cv2.cv.CV_FILLED)
    # ERROR occured the above...
    # dstImg = np.ones(warped_size,np.uint8)
    # dstImg *= 128   # gray
    # dstImg = np.copyTo(filtered, where=mask)  # Copies non-masked
pixels from filtered to dst
    # cv2.imshow("dstImg", dstImg)
    # return mask
    # masking   --- y = a(x-b)
    const = int(DESIRED_FACE_WIDTH*FACE_ELLIPSE_CY/2)
    constY = const
    ypos = DESIRED_FACE_HEIGHT-1
    xpos = DESIRED_FACE_WIDTH-1
    # print const, ypos, yposH, xpos
    for i in range(const):
        for j in range(constY):
            filtered[ypos-j][i] = 128 # gray
            filtered[ypos-j][xpos-i] = 128
        constY -= 1
    return filtered

def equalizeLeftAndRightHalves(faceImg):
    # It is common that there is stronger light from one half of the face than
the other.
    # In that case, if you simply did histogram equalization on the whole face
then it
    # would make one half dark and one half bright. So we will do histogram
equalization
    # separately on each face half, so they will both look similar on average.
    # But this would cause a sharp edge in the middle of the face, because
    # the left half and right half would be suddenly different. So we also
```

```
histogram
    # equalize the whole image, and in the middle part we blend the 3 images together
    # for a smooth brightness transition.
    w, h = faceImg.shape
    #
    # 1) First, equalize the whole face.
    wholeFace = cv2.equalizeHist(faceImg)
    #
    # 2) Equalize the left half and the right half of the face separately.
    midX = w/2;
    leftSide = faceImg[0:h,0:midX]
    rightSide = faceImg[0:h,midX:]
    leftSide=cv2.equalizeHist(leftSide)
    rightSide=cv2.equalizeHist(rightSide)
    #
    # 3) Combine the left half and right half and whole face together,
    # so that it has a smooth transition.
    for y in range(h):
        for x in range(w):
            if (x < w/4):           # Left 25%: just use the left face.
                v = leftSide[y][x]
            elif (x < w*2/4):   # Mid-left 25%: blend the left face & whole face.
                lv = leftSide[y][x]
                wv = wholeFace[y][x]
                # Blend more of the whole face as it moves further right along the face.
                f = (x - w*1/4) / (w*0.2)
                v = round((1.0 - f) * lv + f * wv)
            elif (x < w*3/4):   # Mid-right 25%: blend the right face & whole face.
                rv = rightSide[y][x-midX]
                wv = wholeFace[y][x]
                # Blend more of the right-side face as it moves further right along the face.
```

```
                f = (x - w*2/4) / (w*0.25)
                v = round((1.0 - f) * wv + f * rv)
            else :      # Right 25%: just use the right face.
                v = rightSide[y][x-midX]
            faceImg[y][x] = v
    #
    return faceImg

def normalizePict(frame):
        objects = cascade.detectMultiScale(frame, scaleFactor=1.1,
                minNeighbors=4,
flags=cv2.CASCADE_FIND_BIGGEST_OBJECT,
                minSize=(30, 30))
        if  (objects is not None) and (len(objects) == 1):
            face = objects[0]
            x, y, w, h = face
            # an eye in the upper-left part
            searchRect = (x+w/7, y, w*2/7, h/2)
            leftEye_o = detectOneObject(eyecascade, frame, searchRect, 64)
            # an eye in the upper-right part
            searchRect = (x+w*4/7, y, w*2/7, h/2)
            rightEye_o = detectOneObject(eyecascade, frame, searchRect,
64)
            # Seek a nose in the middle part of the face.
            searchRect = (x+w/4, y+h/4, w/2, h/2)
            nose_o = detectOneObject(nosecascade,  frame, searchRect, 32)
            # Seek a mouth in the lower-middle part of the face.
            searchRect = (x+w/6, y+h*2/3, w*2/3, h/3)
            mouth_o = detectOneObject(mouthcascade, frame, searchRect,
16)
            if ((face is not None) and (leftEye_o is not None) and
                (rightEye_o is not None) and (nose_o is not None) and
                (mouth_o is not None)):
                  #          print          rightEye_o[0]+rightEye_o[2]/2,
rightEye_o[1]+rightEy$
                  #        print              leftEye_o[0]+leftEye_o[2]/2,
```

```
leftEye_o[1]+leftEye_o$
                        # print nose_o[0]+nose_o[2]/2, nose_o[1]+nose_o[3]/2
                        return      preProcessFace(frame,      face,      rightEye_o,
leftEye_o)
                else:
                        return None

savedFile = sys.argv[1]

recognizer = cv2.createFisherFaceRecognizer()
recognizer.load(savedFile)

# targetFile = sys.argv[2]
# target = cv2.imread(targetFile)

src = cv2.VideoCapture(0)
if not src.isOpened():
        print u'映像が取得できません。'
        sys.exit()

windowName = 'Camera'
cv2.namedWindow(windowName,cv2.WINDOW_NORMAL)
Hito                                                                                    =
[None,"hibi","pariasuka","kato","nakagawa","imamura","sibuya","nisiwaki"]

while True:
   retval, frame = src.read()
   if frame is None:
        break
   cv2.imshow(windowName,frame)
   key =    cv2.waitKey(50)

   objects = cascade.detectMultiScale(frame, scaleFactor=1.1,
                        minNeighbors=4,
flags=cv2.CASCADE_FIND_BIGGEST_OBJECT,
                        minSize=(30, 30))
```

```
if (objects is not None) and (len(objects) >= 1):
    x,y,w,h = objects[0]
    face = normalizePict(frame)
    if face is not None:
        cand, conf=recognizer.predict(face)
        if (conf < 900):
                print "hello ",Hito[cand], "conf=",conf
        elif (Hito[cand] == "hibi" ):
                print "hello",Hito[cand],"conf=",conf
                buzzer.start(50)
                time.sleep(1)
                buzzer.stop()
                rr.forward(1)
                rr.stop()
    # cv2.imshow("Normalized",face)
    #
    if x < 100 :
        rr.right(0.1)
        rr.stop()
    elif x > 450 :
        rr.left(0.1)
        rr.stop()
    elif h > 250 :
        rr.reverse(0.1)
        rr.stop()
    elif h < 80 :
        rr.forward(0.1)
        rr.stop()
else:
    pass
    rr.right(0.5)
    rr.stop()
    time.sleep(20)
#
if key == 27:
```

```
        break

cv2.destroyAllWindows()
src.release()
GPIO.cleanup()
```