平成 27 年度　特別研究報告書

直感的に操作可能な
音声認識リモコンの検討

龍谷大学　理工学部　情報メディア学科

T120414　荒木　恭平

指導教員　三好　力　教授

内容梗概

現在、多くの家電が普及することで生活は便利になっている。それらの家電を操作するために、赤外線リモコンなどを用いて操作している。しかし、家電が多くなると、リモコンの数も同様に増えてしまう。その問題は、リモコンを一括で管理することで解決できる。一例として、音声認識リモコンというシステムがある。しかし、音声認識リモコンは家電を一括で操作するために、家電の種類を音声で指定する必要があった。そこで、人の視線と音声認識リモコンを組み合わせ、人の視線で家電を指定し、ボイスコマンド 1 単語で家電を操作するシステムの開発を提案した。

内容

第 1 章 はじめに

1.1 研究背景

現在，家庭では多くの生活家電に溢れている。生活家電を活用することによって、快適な生活をおくることができている。これらの家電の中には、冷蔵庫、電子レンジ、エアコン、自動炊飯器、パソコン、テレビや LED 照明機器などがある。これらを主要家電製品とし、主要家電製品の普及率は図 1 に示す。[1]

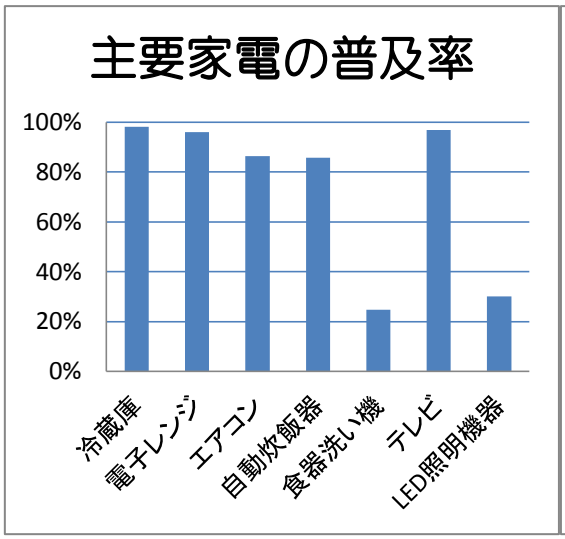

図 1.1：主要家電の普及率

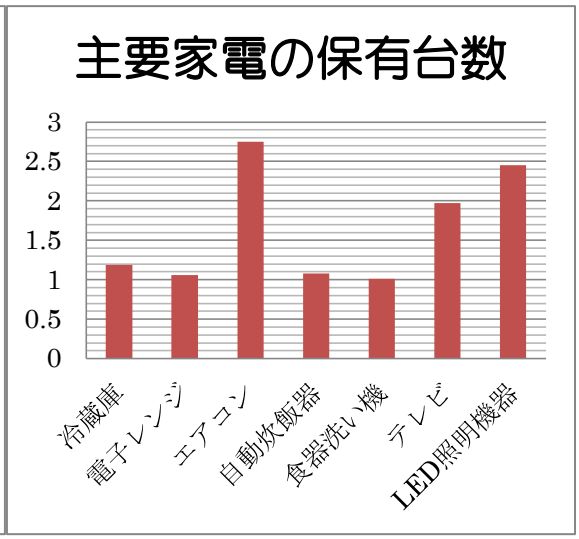

図 1.2：1 世帯の主要家電の保有台数

図 1.1 をみると、冷蔵庫、電子レンジ、エアコン、自動炊飯器やテレビは普及率がとても高く、ほぼ全世帯が所持しているのが読み取れる。これは快適な生活を過ごすのに、最低限必要な家電であるからと思われる。

図 1.2 をみると、普及率が高い冷蔵庫、電子レンジや自動炊飯器は 1 家に 1 台と少しである。また、食器洗い機もおよそ 1 家に 1 台である。次に、保有台数の多さを見てみると、エアコン、テレビや LED 照明機器が高い。これは 1 つの部屋に 1 台置かれる家電であることが予想できる。

以上のように、1 家に 1 台の家電もあれば 1 部屋に 1 台の家電もある。これらの家電があることによって快適な生活を過ごすことができている。では、実際に多くの家電はどのように使うことができるのかというと、家電に付属されているボタンや専用のリモコンにあるボタンを押すことで家電を操作している。電子レンジや自動炊飯器などは使用用途が家電を直接操作する必要があるので、リモコンを使う必要がなく、付属されているボタンで操作する。

しかし、家には多くのリモコンがあり、どのリモコン使用すれば、目的の家電を使用できるのかという問題がある。この問題を解決するために多数存在するリモコ

ンを学習リモコンに学習させ、これを音声で操作可能なリモコンとして作成されたものが音声認識リモコンである。

音声認識リモコンの場合、家電の指定や制御命令など音声で操作するため、指向性がなく直感的に操作できなくなる。ここでの直感的とは、推理などによらず、瞬間的・直接的に物事の本質を見てとることである。つまり、従来のリモコンであれば、家電にリモコンを向けることで直接的に命令する。それは直感的である。音声認識リモコンの場合は直感的に操作できないので、直感的に操作できるように、音声認識リモコンに指向性をもたせるシステムの開発を目的とする。

第 2 章 既存技術

2.1 リモコン[2]

リモコンは、リモートコントロール (remote control) またはリモートコントローラー (remote controller) の略。機器を離れた場所から操作すること（遠隔操作・遠隔制御）、または、そのときに操作のための信号を発信する側の機器（遠隔操作機器）。ただし、「遠隔操作機器」はリモコンからの信号を受信して操作される側の機器を指すこともある。
本項では一般的な呼称である「リモコン」で記述する。また以下では、（発信側の）遠隔操作機器について述べる。

リモコンは、機械や電子機器等を遠隔操作するためのシステムのうち、操作のための信号を送信する側の機器であり、通常、機械や機器の本体側に設けられ操作信号を受信する装置と対を成す（ただし、中には双方向通信で受信も行うものもある）。

本体との間の信号の送受信を有線で行うものと無線で行うもの（ワイヤレス・リモコン）とに大別されるが、現在では、赤外線や電波等の無線で信号の送受信を行うものが多く用いられる。一方で、携帯音楽プレーヤーのようにリモコンと本体間の距離がほぼ一定の状態で使用される簡易な装置等では有線のリモコンも用いられている。

リモート・コントローラーは、テレビや BD レコーダー等の AV 機器、エアコン等の家電製品、電子機器、玩具等の一般家庭用機器を中心に、幅広い機械・機器の操作に用いられる。

2.2 学習リモコン[3]

学習リモコン（がくしゅうりもこん）とは、複数のリモコン装置が発生させる操作信号を記憶（学習）して、一つのリモートコントロール装置で複数の機器を操作する事を可能にするための装置である。
現在、一般家庭においては、テレビ受像機・BD レコーダー・ミニコンポ・エアコン等といった機器を操作するための、様々な機器付属で消費者に提供されるリモコン装置が存在している。これらは主に赤外線によって操作を伝える信号を発信する事で、離れた場所から各々の機器を操作するよう設計されているが、当然ながら異なる機器のリモコンで他の機器を操作する事は出来ない。一部メーカーの BD レコーダーとテレビでは、BD レコーダーのリモコンでテレビのチャンネルや音量を操作できる物もあるが、予めメーカーが設計した組み合わせ同士以外の機器を操作する事は出来ない。

またリモコンは常に利用者の手元にあるため、飲み掛けのお茶を浴びせられたり、つい何処かに置き忘れてしまう事も多く、特に種類が増えてくると、それだけで利用者を混乱させ、余計な労力を強いる事にも繋がる。近年愛好者が増えてきている家庭で本格的な映写室を設けるホームシアターと呼ばれる家庭向け設備に到っては、リモコンの数だけで、気弱な消費者を萎縮させるのに充分な複雑さを持つ事も多い。

このリモコン氾濫時代にあって、それを解消するために利用されるのが学習リモコンである。また良く似た機器ではマルチリモコンがあるが、これもリモコンの氾濫を軽減させる目的で利用される。

2.3 irkit[4]

irkit は、WiFi 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 WiFi をとおして、iPhone や iPad、Android スマートフォンなどから操作できる。

2.4 音声認識層ソフト Julius[5]

Julius はフリーの高性能音声認識ソフトウェアである。オープンソースの高性能な汎用大語彙連続音声エンジンであり、数万語の語彙を対象とした文章発声の認識を行う能力を持っている。また、記述文法の基づく認識を行うこともできるため、文のパターンを人手で記述した認識用文法（有限状態文法）を用いることで、小語彙の音声対話システムや音声コマンド入力など比較的小規模な音声認識システムを容易に構築できる。

Julius は単体で動作できない。Julius で音声認識を行うには、以下のものが必

要になる。
・音響モデル（音素 HMM）音素ごとの音声波形パターンのモデル
・単語辞書…各単語の読みを定義する
・言語モデル（単語 3-gram）どのような単語列が出しやすいか，その 単語間の接続制約を決定する

2.5 音声認識リモコン[6]

音声認識ソフト Julius 及び高機能学習リモコン iRemocon の 2 つを組み合わせることで、音声認識リモコンができる。

システムの流れは、マイクから音声を取得したデータを文字列に変換するために、Julius を用いる。その取得した文字列をあらかじめ登録している文字列と比較し、合致した場合のみ処理を行う。処理内容は、iRemocon 側に登録してある家電の制御命令を家電に送信する制御コードを iRemocon に送信するシステムである。使用するためには、呼びかけプレフィックス、制御したい家電名と制御内容の 3 つが必要である。テレビをつけるためには、“コンピューター、テレビ、つけて”等のように言葉を発することで、パソコンが音声を認識し、家電の制御を行う。“コンピューター”に該当する呼びかけプレフィックスは誤作動を防ぐための安全装置で、この呼びかけプレフィックスを正しく検出できた場合のみ以下の文を読み取り家電の制御を行う。よって、呼びかけプレフィックスは誤認識を抑制できるため必要となる。“テレビ”というのは制御したい家電を指定し、“つけて”は家電の操作内容であるためにこれらも必要になる。

2.6 画像処理ライブラリ Opencv[7]

OpenCV（Open Source Computer Vision Library）は、Intel によって開発された、画像認識に関連する機能のライブラリのことである。。コンピューターで画像や動画を処理するのに必要となる、さまざま機能が実装されており、BSD ライセンスで配布されていることから学術用途だけでなく商用目的でも利用できる。加えて、マルチプラットフォーム対応されているため、幅広い場面で利用されていることが特徴である。

OpenCV を使うと、主に以下のような機能を利用できる。
・フィルター処理
・行列演算
・オブジェクト追跡（Object Tracking）
・領域分割（Segmentation）
・カメラキャリブレーション（Calibration）
・特徴点抽出
・物体認識（Object recognition）

・機械学習（Machine learning）
・パノラマ合成（Stitching）
・コンピュテーショナルフォトグラフィ（Computational Photography）
GUI（ウィンドウ表示、画像ファイル、動画ファイルの入出力、カメラキャプチャ）
これらの機能を利用することで、例えば物体の認識、パターン認識、動作の認識といった、コンピュータビジョンに関する高度な画像処理機能を容易に利用できるようになる。

2.7 深層学習[8]

深層学習とは機械学習の一種である。機械学習とは、人間が自然に行っている学習能力と同様の機能をコンピューター上でも行えるようにする手法である。それを実現するために脳の機能を模して作られたのがニューラルネットワークであり、深層学習はこのニューラルネットワークを多層構造にしたものである。ニューラルネットの構造上、入力層、隠れ層、出力層があり、この隠れ層の数を増やすことで重層構造にする。隠れ層が重層になることで、何段階かで認識を繰り返し、ピクセルから線、線からパーツ、パーツから全体の概念、というように階層的に特徴を抽出して、正確な識別ができるようになる。この特徴量を自動で抽出してくれるのが深層学習の優れた点である。ここでの特徴量とは、特定の概念を特徴づけるための変数である。この特徴量を正確に抽出することが、画像識別や音声認識では重要となる。

2.8 パターンマッチング[9]

パターンマッチングとはコンピューターで、複数の文字列や図形、ファイルなどを比較し、同一または類似したものであるかどうかを調査すること。本研究では画像に対してパターンマッチングを用いる。

第 3 章 提案手法

3.1 既存手法の問題点

既存のリモコンは、リモコンを家電に向け、ボタンを押すことで操作をしていた。これは指向性をもたせることで、誤作動防止と家電指定を一括で行っている。問題点として、常に利用者の手元にあるため、つい何処かに置き忘れてしまう事も多く、特に種類が増えてくると、それだけで利用者を混乱させ、余計な労力を強いる事にも繋がる。リモコンの数だけで、気弱な消費者を萎縮させるのに充分な複雑さを持つ事も多い。

既存の音声認識リモコンは、音声で一括に家電を操作できるので、常に利用者の手元にリモコンがある必要がない。問題点として、家電を操作するためには、3 語の単語が必要であり、音声が長くなるので、音声認識の誤認識の可能性や直感的な操作ができないという問題点が存在する。

3.2 システムの提案

従来のリモコンと音声認識リモコンを組み合わせ、音声認識リモコンに指向性をもたせることで、誤作動が少なく、直感的に操作可能なリモコンができると考える。

システムを構築するのにサーバークライアントシステムを用いる。このシステムを用いることで、処理を役割分担させる。サーバー側は家電の認識もしくは視線の検出を行い、irkit を操作するという処理を行わせる。クライアント側は web カメラやマイクから画像や音声を取得し、サーバーサイドに常時送信する処理を行わせる。サーバーは移動する必要がないので、据え置きのパソコンを流用する。クライアントは人が装着し、人がみている画像を取得する場合と家電に設置する場合を考えるので、設置が簡単にでき、軽量でなければならない。候補としては、カメラやマイクが搭載されている眼鏡型端末の HMD（Head Mount Display）や web カメラや usb マイク接続できる小型のマイコンボードである Raspberry pi が挙げられる。本実験では Raspberry pi を用いる。

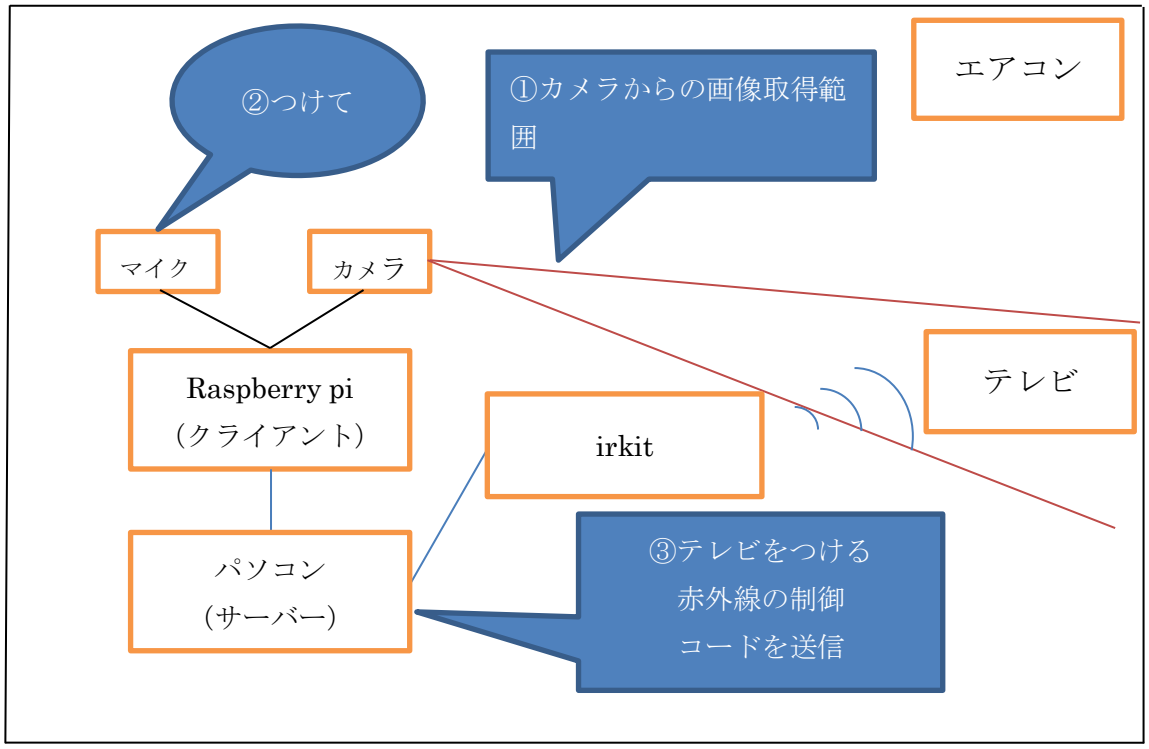

図 3.1：システムのイメージ図（家電認識）

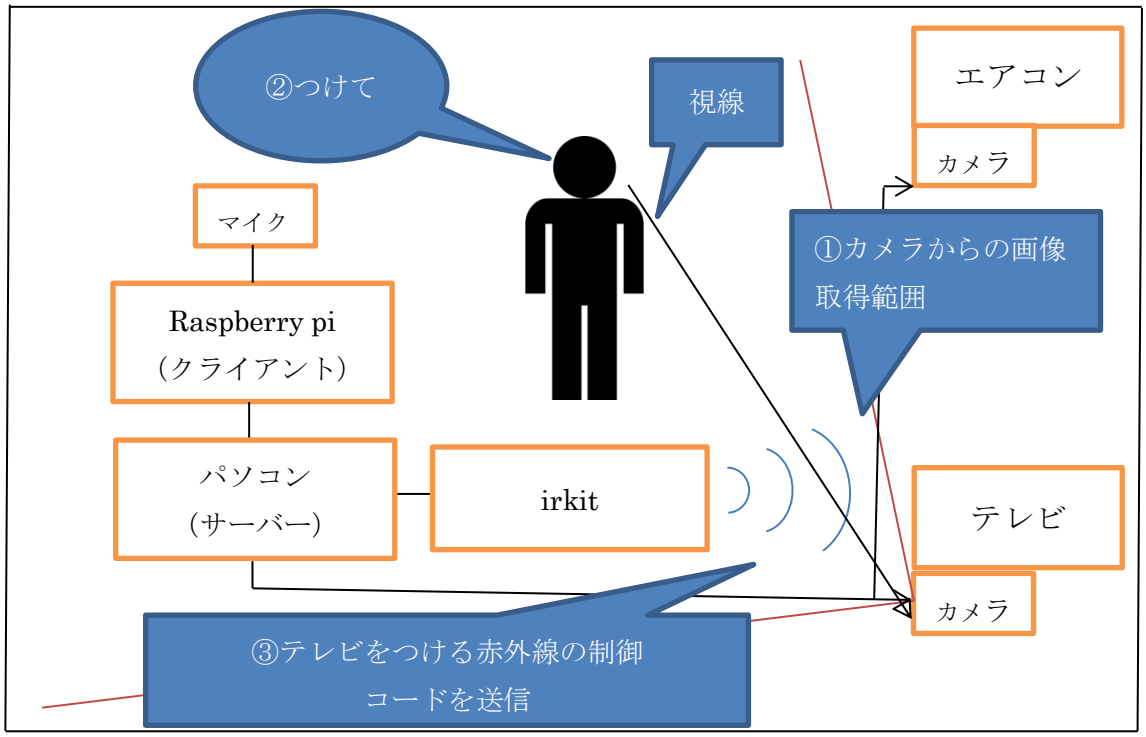

図 3.2：システムのイメージ図(視線検出)

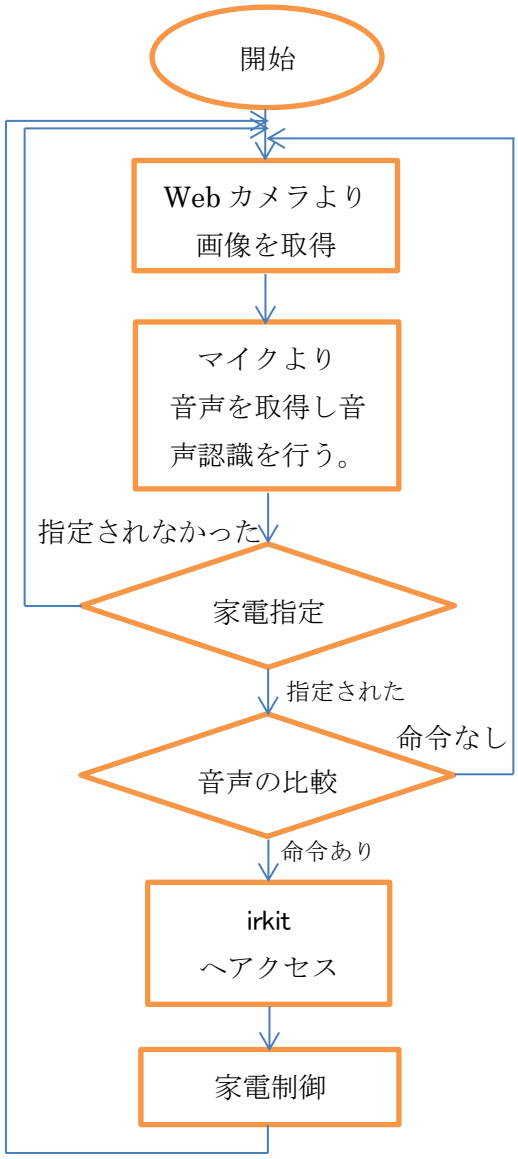

図 3.3：システムのフローチャート

図 3.1 および 3.2 は家電認識を行うシステムと視線認識を行うシステムのイメージ図である。図 3.3 はシステムのフローチャートである。

家電認識では Raspberry Pi に接続された、Web カメラと USB マイクから常時、画像と音声を取得する。常時取得している画像から家電指定をされているかを判別する。家電指定されている場合のみ、音声の比較に移る。音声があらかじめ登録されている命令文と一致した場合のみ、irkit へ制御コードを送信する。制御コードを受信した irkit は家電を制御する。

視線認識では、パソコンに接続されたカメラから人の視線を検出する。視線が家電に向けられていた場合のみ音声の比較に移る。以後は家電認識と同様である。

3.3 家電の指定方法

本研究では、指向性をもたせた音声認識リモコンを開発するのに、以下の 2 つの手法を提案し、どちらが適しているか比較検討する。

3.3.1 家電の指定方法で画像認識を用いる提案

使用者本人の視線と同様の画像を取得できる位置にカメラを配置することで、人が家電に視線を向けるとそのカメラの取得範囲に家電がうつり、その家電が制御対象として指定される案である。カメラから取得した画像に、家電が存在するのかを検出するためには家電認識を行う必要がある。家電認識を行うには、深層学習またはパターンマッチングを用いる。深層学習では、多数の画像を与え、自動で特徴量を抽出した学習モデルを作成する。この学習モデルを用いて、カメラからの画像で家電が写っているのかを判定する。パターンマッチングでは、画像判定であるので、画像の中から、特定のパターンを探し出す「テンプレートマッチング法」を用いる。テンプレートとは型紙のことであり、それを画像上で移動させながら比較していく方法である。このテンプレートマッチングを用いて家電の識別判定をする。

3.3.2 家電の指定方法で視線検出を用いる提案

家電にカメラを置き、使用者がそのカメラの方へ視線を向けると、その視線がこちらを向いていると認識し、その家電が指定されているという手法である。カメラから取得した画像で、カメラの方へ視線を向けているのかを調べる必要がある。そのため、人の視線を検出する必要がある。検出方法には OpenCV の顔の認識、目の認識を行う機能を用いて視線を向けているのかを判別する。視線検出には、Web カメラから人の顔画像を取得することで視線検出を行う。詳細な視線検出は Web カメラだけでは行うことが難しいが、家電側を向いているのか向いていないのかを検出することは可能だと考える。

人の視線は眼球中心と瞳孔の延長線上である。これら 2 つの座標を得ることで視線の検出ができる。眼球中心を web カメラで取得することはできないので、代わりに目頭と目尻の中点座標を取得することで代替する。瞳孔は円形であるので、円を検出するための Hough 変換を用いて、瞳孔を検出する。これだけでは、顔の向きを考慮していないので、視線検出を行った場合、正面を向いていないと視線検出できない。しかし、家電を見ているかを判別するには、顔が家電方向ではなく、視線だけが向いている可能性がある。つまり、顔の方向も必要となる。顔の方向検出には、顔が斜めに向いているほど、両目の距離が近くなることを利用する。よって、正面を向いていれば両目の距離は長くなり、斜め 45 度を向いていれば正面の両目の距離の半分になる。90 度斜めであれば片目しか見えなくなる。これらの視線方向と顔の方向を用いて、家電側を向いているか向いていないかを判別する。

第 4 章　実験

提案した手法で問題解決が可能になったかを評価するために「家電指定の正確さ」が重要である。この家電指定が正確に行われないと正しく家電を制御できない。家電の指定ができていない場合、例えば“電源をつける”制御命令が来たとしても、どの家電の電源をつけるのかを判別ができない。ゆえに、家電の指定は高精度でなければならない。本研究では、家電指定は画像認識を用いる家電判定と視線検出を用いる家電判定を行っている。以上の点より、どちらの判定方法が本システムに適しているのかを検討するため、家電判定が正確にできているのかを実験を行い、比較する。

4.1 家電判定

4.1.1　深層学習による手法

深層学習を行う際、既存の深層学習フレームワークである caffe[12]を用いる。このフレームワークを用いて家電の分類をするためには、家電分類機である学習モデルを作成する必要がある。この学習モデルを作成するにあたって、ネットワークを決める必要がある。本実験では参考文献[12]のチュートリアルに沿って実験したが、それに用いたネットワークは CiFar-10 用のネットワーク[12]を変更し用いた。変更した箇所を図 4.1 と表 4.1 に示す。また、全文を付録にのせておく。

```
layer {
  name: "cifar"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mean_file: "mean.binaryproto"
  }
  data_param {
    source: "train_leveldb"
    batch_size: 100
```

```
layer {
  name: "cifar"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    mean_file: "mean.binaryproto"
  }
  data_param {
    source: "test_leveldb"
    batch_size: 100
```

```
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 3
    weight_filler {
```

図 4.1：Cifar-10 の cifae10_quick_train_test.prototxt の変更箇所

表 4.1：Cifar-10 の cifar10_quick_solver.prototxt の変更箇所

Test_iter	10
Test_interval	50
Base_ir	0.0009
Display	10
max_iter	3000
Snapshot	500
Solver_mode	GPU

本研究では深層学習の学習モデルはテレビ、エアコン、その他の 3 値分類機である。

実験に用いた学習用のデータは google 画像検索から対象の家電名を検索し、1 つの家電で 1000 枚集めた。実験画像として求める条件として、対象の家電単体の画像や対象の家電以外の画像が含まれている画像である。本実験では 3 値分類であるので、合計 3000 枚の画像を収集した。これを左右反転させ 6000 枚に水増しをして実験データを揃えた。

4.1.2　パターンマッチングによる手法

パターンマッチングを行うのには、OpenCV のテンプレートマッチング機能を用いた。本実験ではテレビ、エアコン、その他の 3 値分類を行うため、テレビとエアコンのテンプレート画像が必要となる。

図 4.2：テレビのテンプレート画像

図 4.3：エアコンのテンプレート画像

図 4.2 および図 4.3 の画像はパターンマッチングで用いたテンプレート画像である。このテンプレート画像とカメラから取得した画像を比較するためにパターンマッチングを行った。実験を行うには、OpenCV の cv2.matchTemplate 関数を用いて、templete.py というプログラムを作成した。「python templete.py　“カメラからの

取得画像名” “テンプレート画像名”」
と実行することで、パターンマッチングの結果が返される。

4.2 視線検出

視線検出を行う際に、画像処理・画像解析および機械学習等の機能を持つライブラリである、OpenCV のパーツ検出器を用いる。家電に設置しているカメラから取得した画像に人が写っていた場合、その人の視線を検出するために、顔の認識、目の認識を行う。

図 4.4：視線検出の実験に用いた画像

図 4.4 の画像は視線検出の実験に用いた画像の一例である。図 4.4 の画像から視線検出のために目のパーツ検出器に与えると、背景の一部などを目と誤認識してしまう可能性があるため、顔領域をとりだしてから、目のパーツ検出器に与える必要がある。したがって、先に図 4.4 の画像から顔認識し、顔領域を切り出す。次に、目のパーツ検出器に与えて両目の座標を取り出すことで、背景の一部などを目のパーツと誤認識してしまう可能性を排除する。また、瞳孔検出で用いる Hough 変換の関数を用いるのに、円の半径などのパラメーターを決める必要がある。本実験では、その関数に与えたパラメーターは以下の表 4.2 である。

表 4.2：Hough 変換のパラメーター

Method	cv2.cv.CV_HOUGH_GRADIENT
Dp	2
minDist	50
Param1	10
Param2	10
minRadius	0
maxRadius	30

表 4.6 のパラメーターを用いて、eye_checker.py というプログラムを作成した。実験を行うには、「python eye_checker.py “顔画像名”」と実行することで、視線検出の結果が返される。また、実験に用いたデータは実験画像として求める条件は、顔写真だけの画像である。正面を向いている画像、斜め左を向いている画像や斜め右を向いている画像で、均等に画像を収集した。

4.3 実験環境

画像認識を用いる家電指定の実験を行うにあたって、深層学習のために予め用意した学習用画像に含まれておらず、テンプレート画像にも使用していない家電の画像を用いる必要がある。したがって、上記の条件に当てはまる一台のテレビとエアコンを実験対象とする。

図 4.5：実験対象のテレビ画像

図 4.6：実験対象のエアコン画像

今回用いた web カメラは有効画素数 315 万画素で 1920×1080 ピクセルの web カメラである。図 4.5 および 4.6 は web カメラから取得した実験対象のテレビとエアコンの画像である。

4.5 実験方法

画像認識を用いる家電指定の実験の場合、web カメラから画像を取得した時、人の位置によって人と家電との距離や角度が異なるので、正確に家電の指定を行えているかを評価するためには、下図の図 4.7 のように、web カメラとテレビの距離および角度ごとの画像を取得したうえで家電の認識率を求める必要がある。図 4.5 および 4.6 の実験対象を用いて、カメラと対象の角度や距離においての正答率をだすことを目的とする。本実験では、実験対象との距離の正答率をだすために、実験対象との角度 90 度において、開始距離は 1.5m から終了距離は 3.6m まで 0.3m の間隔ごとの画像を web カメラで取得する。実験対象との角度での正答率をだすためには、実験対象との距離 2m において、開始角度は 10 度から終了角度は 90 度まで 10 度の間隔ごとの画像を web カメラで取得する。深層学習による手法では web カメラから取得した画像を作成した学習モデルに与える。その結果は 3 値分類の正答率を返されるので、それぞれの結果を正答率と距離のグラフおよび正答率と角度にグラフにまとめる。パターンマッチングによる手法では web カメラから取得した画像と図 4.2 および図 4.3 のテンプレート画像をずらし比較を繰り返すことで、マッチングをする。マッチング結果は 1 以下の小数点で返されるが、テレビの場合、テレビの一部分とマッチングした結果の正答率が高いことが多く、それが多数の領域が存在するので、どの値がテレビ全体での結果なのかを判別できない。そこで閾値を決めることで、閾値以上の結果の場合は“できた”未満であった場合“できなかった”と結果が返すようにした。その実験結果をそれぞれの距離と角度においての結果をもとに表を作成する。

視線検出を用いる家電指定の実験の場合、人の顔が写った多数の画像を与えて、正確に視線検出を行えたのかを判別する。本実験では、google 画像検索で“顔写真”と検索して画像を 100 枚集め、それらの画像を用いて、正答率を求める。

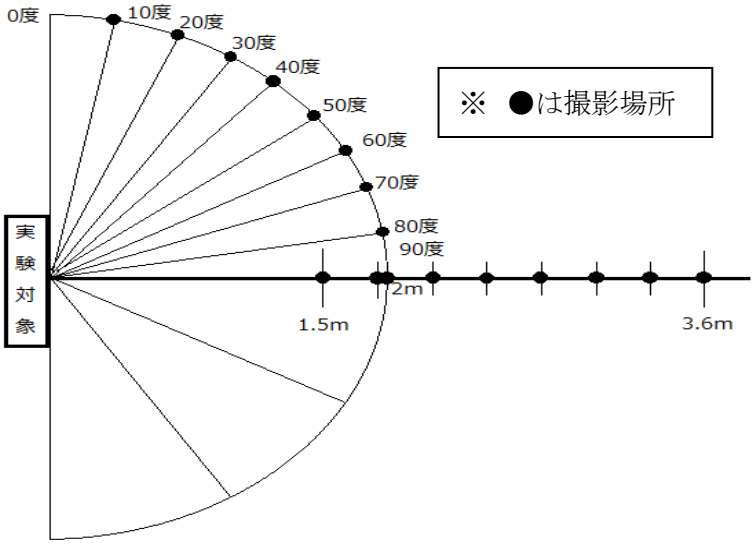

図 4.7：実験方法のイメージ図

4.5 実験結果

4.5.1　画像認識の結果

4.5.1.1　深層学習

図 4.5 で示した対象をテレビの場合はカメラからの距離 0.3m 間隔や角度 10 度毎に、図 4.6 で示した対象をエアコンの場合はカメラからの距離 0.1m 間隔や角度 10 度毎に学習モデルに与え、家電分類を行った実験結果を以下の図に示す。

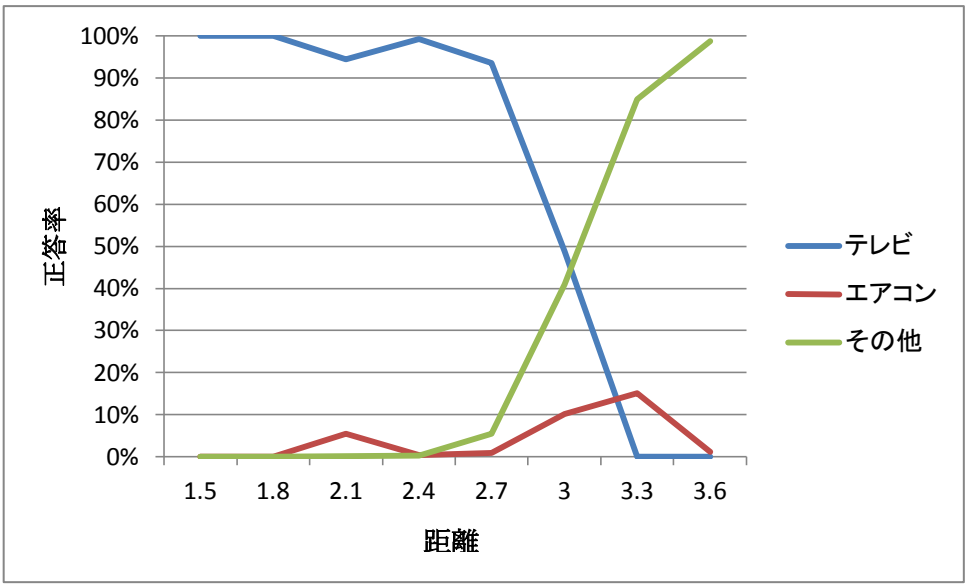

図 4.8：距離 1.5m～3.6m（テレビ）

図 4.8 はテレビから距離 1.5m～3.6m までを間隔 0.3m おきでテレビの画像を取得し、分類機に与え、分類した結果である。図 4.8 を見ると、2.7m までの距離である場合は、テレビである確率はほぼ 100%であると示している。次に計測した点、カメラとの距離 3m の場合は、テレビである確率は約 45%であると示していることが確認できる。以上より、3m あたりが閾値付近であることがわかる。

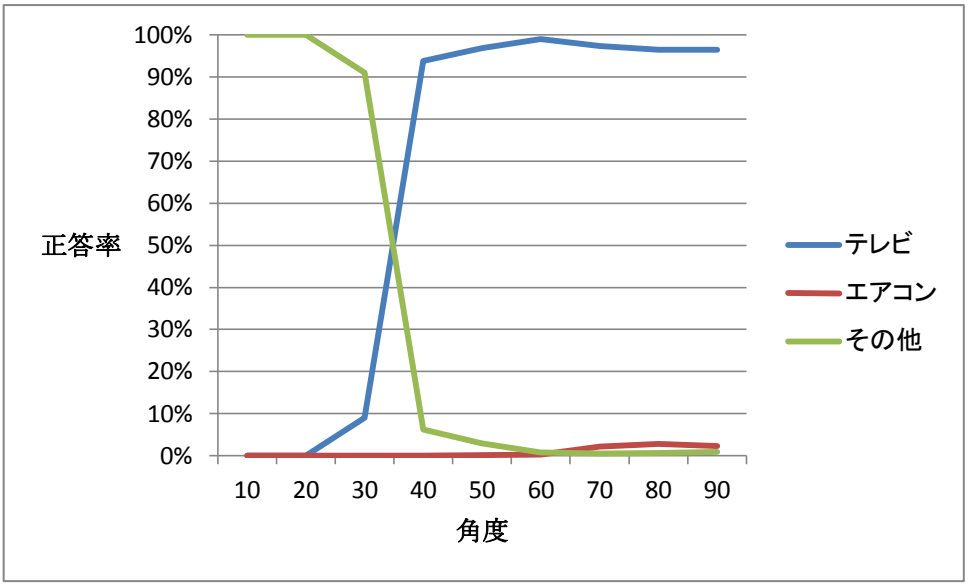

図 4.9：角度 10～90 度、距離 2.0m（テレビ）

図 4.9 はカメラとの距離 2.0m において、カメラとテレビのなす角度を 10～90 度まで間隔 10 度ずつのテレビの画像を取得し、学習モデル 1 の分類機に与え、分類した結果である。図 4.9 を見ると、30 度まではほぼテレビではないと識別し、40 度でほぼテレビであると識別している。35 度あたりが閾値付近であることがグラフの交点から読み取ることができる。

以上より、深層学習では距離 3m 以内角度 35 度以上までは、正確に識別できることがわかった。

また、実行時間は 2.54 秒であった。

図 4.10：角度 30 度

図 4.11：角度 40 度

図 4.10 および図 4.11 は閾値付近の実験に用いた画像である。

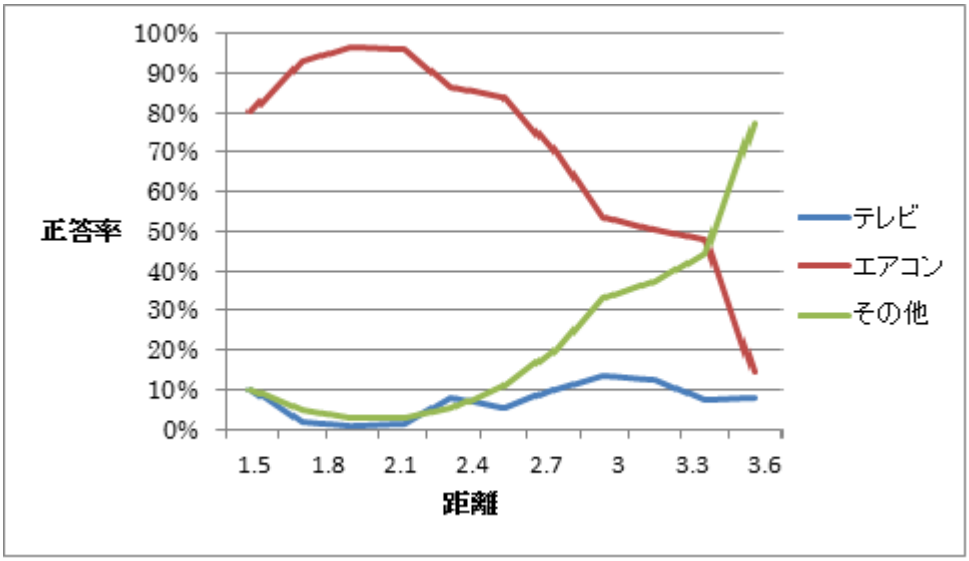

図 4.12：距離 1.5m～3.6m（エアコン）

図 4.12 はエアコンから距離 1.5m～3.6m までを間隔 0.3m おきでエアコンの画像を取得し、分類機に与え、分類した結果である。図 4.12 を見ると、近すぎる場合は少し精度が低くなっている。また、距離が遠くなるにつれ徐々に認識率が下がり、3.3m のところではエアコンではないという結果が出力されている。

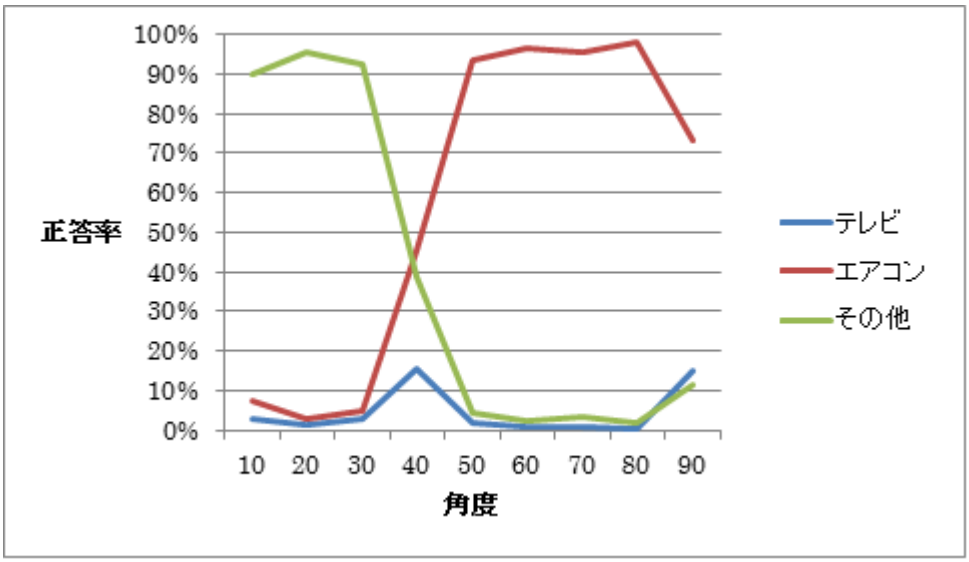

図 4.13：角度 10～90 度、距離 2.0m（エアコン）

図 4.13 は距離 2.0m において、カメラとエアコンのなす角度を 10～90 度まで間隔 10 度ずつのテレビの画像を取得し分類機に与え、分類した結果である。図 4.13 を見ると、30 度まではほぼエアコンではないと識別し、40 度ではエアコンとそれ以外との閾値付近となっていることがわかる。また、50 度に達するとほぼエアコンであることがわかる。

4.5.1.2 パターンマッチング

図 4.5 で示した対象をテレビの場合はカメラからの距離 0.3m 間隔や角度 10 度毎に、図 4.6 で示した対象をエアコンの場合はカメラからの距離 0.3m 間隔や角度 10 度毎に学習モデルに与え、家電分類を行った。実験結果を表 4.3～4.6 に示す。なお、どの実験も 10 回行ったが結果は同じであった。

表 4.3：距離 1.5～3.0m（テレビ）

距離	正答
1.5m	できた
1.8m	できた
2.1m	できた
2.4m	できなかった
2.7m	できなかった
3.0m	できなかった
3.3m	できなかった
3.6m	できなかった

表 4.3 はテレビからの距離 1.5～3.6m まで、テレビとの距離を 0.3m ずつずらした画像を、テンプレート画像とマッチングをした結果である。

表 4.4：角度 10～90 度、距離 2.0m（テレビ）

角度	正答
10 度	できなかった
20 度	できなかった
30 度	できなかった
40 度	できた
50 度	できた
60 度	できた
70 度	できた
80 度	できた
90 度	できた

表 4.4 はテレビからの距離 2.0m の場所において、テレビとのなす角を 10 度～90 度まで 10 度ずつをずらした画像を、テンプレート画像とマッチングをした結果である。40 度まではマッチングがしたが、30 度からはマッチングはしなかった。

表 4.5：距離 1.5～3.6m（エアコン）

距離	正答
1.5m	できた
1.8m	できた
2.1m	できた
2.4m	できた
2.7m	できなかった
3.0m	できなかった
3.3m	できなかった
3.6m	できなかった

表 4.5 はエアコンからの距離 1.5～3.6m まで、距離を 0.3m ずつずらした画像を、テンプレート画像とマッチングをした結果である。
1.5m から 2.4m の間はマッチングした。その後 2.7m からはマッチングすることはなかった。

表 4.6：角度 10～90 度、距離 2.0m（エアコン）

角度	正答
10 度	できなかった
20 度	できなかった
30 度	できなかった
40 度	できなかった
50 度	できなかった
60 度	できなかった
70 度	できた
80 度	できた
90 度	できた

表 4.6 はエアコンからの距離 2.0m の場所において、エアコンとのなす角を 10 度～90 度まで 10 度ずつをずらした画像を、テンプレート画像とマッチングをした結果である。60 度まではマッチングがしなかったが、70 度からはマッチングをした。また、実行時間は 0.24 秒であった。

4.5.2　視線検出の結果

以下の画像は実験を行った画像の一例である。

図 4.14：顔認識結果

図 4.15：左目の認識結果

図 4.16：右目の認識結果

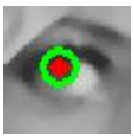

図 4.17：左目の認識結果

図 4.18：右目の認識結果

図 4.14 は図 4.5 から顔部分を切り出した図である。図 4.8 から目を切り出したのが図 4.15 および図 4.16 である。次に、Hough 変換を行い、瞳孔検出を行ったのが図 4.17、図 4.17 である。そこから、両目の瞳孔の座標をとりだした結果が以下である。

表 4.7：両目の瞳孔の座標

	W(pixcel)	H(pixcel)
左目	323	266
右目	265	262

表 4.7 は両目の瞳孔の座標である。この座標を用いて、視線検出をおこなった結果、図 4.14 の画像では正しく判定ができた。また、他の実験画像で行った結果、100 枚中 37 枚が正しく、残りは誤りであった。つまり 37%の正答率であった。その誤った結果の画像の多くは口と瞳孔の認識の失敗であった。その枚数は 42 枚であった。つまり OpenCV の誤認識や瞳孔の誤認識が 62%であった。また実行時間は 0.28 秒であった。

第５章　考察

深層学習では、今回作成した学習モデルを用いた場合、家電を識別するのに 2 秒ほどの時間を要した。テレビでは、30 度までは正確に識別ができ、距離は 3m まで正確に識別することが可能であった。エアコンでは、40 度までは正確に識別ができ、距離は約 3.3m までは正確に識別することができた。以上のように、対象との距離が離れる程、対象画像の周りにノイズが入るので、誤認率があがることが予想される。しかし、カメラの中央を切り出し再び識別させることで解決ができる。また、対象との角度が小さくなるほど、誤認率が高くなるが、30 度以下の位置で見ることがほぼないと考えられる。よって本システムには利用可能であると考えられる。

パターンマッチングを用いた場合、テレビでは 40 度までは正確に識別ができ、距離は約 2m までは正確に識別ができた。エアコンでは、ほとんど識別することがなかった。だが、家電を識別するのに要した時間は 0.2 秒ほどであった。パターンマッチングはテンプレート画像との比較であるので、処理速度はかなり早いが、別のクラスとの比較をするには別のテンプレート画像を与える必要がある。また、テンプレートに対して傾斜を持っている画像や、相似形であるが大きさが異なる画像は検出できなかったので、少しでも特徴が変わると正確な判定はできない。つまり、それらを解決するためには別のテンプレート画像を与える必要がある。よって、多クラス分類や高精度な結果出力をするにはより多くのマッチング処理が必要となるので、処理時間が増えてしまうことがわかった。

視線認識では、処理時間が早いが、顔認識、目の認識、のどちらか 1 つでも失敗すれば、正しい視線は検出されない。今回用いた OpenCV の分類機を用いて、これらの認識を行ったが、画像により認識の失敗などが多々あった。また、瞳孔の座標検出では、Hough 変換を用いたがあらかじめに与えておいたパラメーターでは全ての画像で、正しい検出することがなく、画像によって、パラメーターの変更が必要であった。

以上より、今回のシステムでは、より多くの家電に対応すべきであるので、家電の指定方法で画像認識を用いるべきであることが妥当であった。

第 6 章　まとめ

本論文では、音声認識リモコンに指向性を持たせることで、直感的に操作可能な音声認識リモコンの作成方法を提案し、指向性をもたせるために web カメラ、音声で家電を操作するためにマイク、家電を制御するために irkit を用いた。そして、より効率がよく高性能なシステムの検討として、指向性を持たせる手法に家電を識別するためには深層学習とパターンマッチング、簡易的な視線検出するには、OpenCV のパーツ分類機用いて家電の判別を行った。本実験では、家電識別はテレビ、エアコン、その他の 3 値分類であった。結果として、簡易的な視線検出よりパターンマッチングや深層学習による家電分類の方が本研究に適していた。

今後の課題は、様々な家電に対応するため、深層学習の分類機に家電を学習させることである。また、高精度な分類機の作成をすることで誤認率を下げることが可能だと考えられる。

謝辞

本論文を作成するにあたり，ご指導頂きました三好力教授に深謝いたします．また，多忙の中日頃の議論にご協力くださった同三好研究室の皆様や，学友の皆様に心より感謝いたします．

参考文献

[1]. 全国消費実態調査
<http://www.garbagenews.net/archives/2277306.html>
[2]. リモコン
https://ja.wikipedia.org/wiki/リモコン
[3]. 学習リモコン
https://ja.wikipedia.org/wiki/学習リモコン
[4]. irkit
http://getirkit.com
[5]. 音声認識ソフト Juilus
http://julius.osdn.jp
[6]. CiNii 論文 - 音声認識で何でも制御 もうリモコンの操作は要らない
<http://bizboard.nikkeibp.co.jp/houjin/cgi-bin/nsearch/md_pdf.pl/0000319931.pdf?NEWS_ID=0000319931&CONTENTS=1&bt=LIN&SYSTEM_ID=HO>
[7]. 画像処理 – OpenCV
http://www.buildinsider.net/small/opencv/001
[8]. 深層学習
http://stonewashersjournal.com/2015/03/05/deeplearning1/
[9]. テンプレートマッチング
http://codezine.jp/article/detail/86
[10]. Template Matching
http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html
[11]. Caffe で手軽に画像分類
http://techblog.yahoo.co.jp/programming/caffe-intro/
[12]. Caffe
http://caffe.berkeleyvision.org/

付録

cifar10_quick_train_test.prototxt

```
name: "CIFAR10_quick"
layer {
  name: "cifar"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mean_file: "mean.binaryproto"
  }
  data_param {
    source: "train_leveldb"
    batch_size: 100
    backend: LEVELDB
  }
}
layer {
  name: "cifar"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    mean_file: "mean.binaryproto"
  }
  data_param {
    source: "test_leveldb"
    batch_size: 100
    backend: LEVELDB
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.0001
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "pool1"
  top: "pool1"
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu2"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "pool2"
  top: "conv3"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu3"
  type: "ReLU"
  bottom: "conv3"
  top: "conv3"
}
layer {
  name: "pool3"
  type: "Pooling"
  bottom: "conv3"
  top: "pool3"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool3"
  top: "ip1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 64
    weight_filler {
      type: "gaussian"
      std: 0.1
    }
```

```
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 3
    weight_filler {
      type: "gaussian"
      std: 0.1
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```

cifar10_quick.prototxt

```
name: "CIFAR10_quick_test"
input: "data"
input_dim: 1
input_dim: 3
input_dim: 200
input_dim: 200
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
    stride: 1
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "pool1"
  top: "pool1"
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
    stride: 1
  }
}
layer {
  name: "relu2"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "pool2"
  top: "conv3"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    stride: 1
  }
}
layer {
  name: "relu3"
  type: "ReLU"
  bottom: "conv3"
  top: "conv3"
}
layer {
  name: "pool3"
  type: "Pooling"
  bottom: "conv3"
  top: "pool3"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool3"
  top: "ip1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 64
  }
}
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 3
  }
}
layer {
```

```
  name: "prob"
  type: "Softmax"
  bottom: "ip2"
  top: "prob"
}
```

cifar10_quick_solver.prototxt

```
# reduce the learning rate after 8 epochs (4000 iters) by a factor of 10
# The train/test net protocol buffer definition
net: "examples/cifar10/cifar10_quick_train_test.prototxt"
# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 10
# Carry out testing every 500 training iterations.
test_interval: 50
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.0009
momentum: 0.9
weight_decay: 0.004
# The learning rate policy
lr_policy: "fixed"
# Display every 100 iterations
display: 10
# The maximum number of iterations
max_iter: 3000
# snapshot intermediate results
snapshot: 500
snapshot_prefix: "examples/cifar10/cifar10_quick"
# solver mode: CPU or GPU
solver_mode: GPU
```

bunrui.py

```
from caffe.proto import caffe_pb2
import numpy as np
import caffe

mean_blob = caffe_pb2.BlobProto()
with open('mean.binaryproto') as f:
    mean_blob.ParseFromString(f.read())
mean_array = np.asarray(
mean_blob.data,
dtype=np.float32).reshape(
    (mean_blob.channels,
    mean_blob.height,
    mean_blob.width))
classifier = caffe.Classifier(
    'deploy.prototxt',
    'caffe_train_iter_3000.caffemodel',
#       mean=mean_array,
    raw_scale=255)
#分類する画像指定
image = caffe.io.load_image('test/40-50-230-270/k_58.JPG')
predictions = classifier.predict([image], oversample=False)
pred = np.argmax(predictions)
#結果の出力
print(predictions)
print(pred)
```

paturn.py

```
import cv2
import sys
import numpy as np
from matplotlib import pyplot as plt

img_rgb = cv2.imread(sys.argv[1])
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)
template = cv2.imread(sys.argv[2],0)
w, h = template.shape[::-1]

res = cv2.matchTemplate(img_gray,template,cv2.TM_CCORR_NORMED)
threshold = 0.5

(minval, maxval, minloc, maxloc) = cv2.minMaxLoc(res)
print "({0}, {1}) score = {2}¥n".format(maxloc[0], maxloc[1], maxval)
#閾値と比較
loc = np.where( res >= threshold)
for pt in zip(*loc[::-1]):
    cv2.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0,0,255), 2)
cv2.imwrite(sys.argv[3],img_rgb)
```

eye_checker.py

```
# -*- coding:utf-8 -*-
import numpy
import cv2
import sys
import numpy as np
import math
from math import sqrt
###
"""
顔認識後にそれぞれの認証を行う。
"""
###

def distance(x1, y1, x2, y2):
  return sqrt(pow(x1 - x2,2) + pow(y1 - y2,2))

cascade_face_path = 'haarcascades/haarcascade_frontalface_alt.xml'
#cascade_mouse_path = "haarcascades/haarcascade_mcs_mouth.xml"
cascade_eye_path = "haarcascades/haarcascade_eye.xml"

########################顔認識 start
image_path = sys.argv[1]
color = (255, 255, 255) #白
image = cv2.imread(image_path)
image_gray = cv2.cvtColor(image, cv2.cv.CV_BGR2GRAY)
cascade = cv2.CascadeClassifier(cascade_face_path)
#実行
facerect = cascade.detectMultiScale(
                image_gray,     scaleFactor=1.1,     minNeighbors=1,
minSize=(1, 1))
if len(facerect) <= 0:
    exit()
face_pix = 0
#検出した顔を囲む矩形の作成
for rect in facerect:
    cv2.rectangle(image,    tuple(rect[0:2]),tuple(rect[0:2]+rect[2:4]),    color,
thickness=2)
#認識結果の保存
cv2.imwrite("face.jpg", image)
#顔認識部分を切り出し、保存
for rect in facerect:
    x = rect[0]
    y = rect[1]
    w = rect[2]
    h = rect[3]
    tmp = h*3/4
    mouse_y = y
    mouse_x = x
    eye_y = y
    eye_x = x
    face_pix = w
    cv2.imwrite('face1.jpg', image[y:y+h, x:x+w])
    cv2.imwrite("face_low.jpg", image[y+tmp:y+h, x:x+w])

mouse_y = mouse_y + tmp

cv2.imshow("face1.jpg", image)


########################顔認識 end

########################目 start
eye_c = 0
image_path = "face1.jpg"
image = cv2.imread(image_path)
gray = cv2.cvtColor(image, cv2.cv.CV_BGR2GRAY)
cascade = cv2.CascadeClassifier(cascade_eye_path)
eyerect = cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=1,
minSize=(1, 1))

if len(eyerect) > 0:
  i = 0
  # 検出した目を囲む矩形の作成
  for rect in eyerect:
    i = i + 1
    x = rect[0]
    y = rect[1]
    w = rect[2]
    h = rect[3]
    eye_y = eye_y + y
    eye_x = eye_x + x

    cv2.imwrite("eye%d"%(i)+".jpg", image[y:y+h, x:x+w])
    cv2.rectangle(image,    tuple(rect[0:2]),tuple(rect[0:2]+rect[2:4]),    color,
thickness=2)

    img = cv2.imread("eye%d"%(i)+".jpg",0)
    img = cv2.medianBlur(img,5)
#瞳孔検出
    cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)
    circles = cv2.HoughCircles(img,cv2.cv.CV_HOUGH_GRADIENT,2,50,

param1=10,param2=10,minRadius=0,maxRadius=30)

    print ("目の座標%d"%(i)+".jpg")
    if i == 1:
      eye_y1 = eye_y + circles[0,0,1]
```

```
            eye_a1 = eye_x
            eye_x1 = eye_x + circles[0,0,0]
            eye_b1 = eye_x + w
#           print eye_x1,eye_y1
            eye_center = eye_x * (w/2)
            if eye_x1 > eye_center:
                if eye_x1 > (eye_center + 20):
                    eye_c = 3
                else:
                    eye_c = 2
            else:
                if eye_x1 < (eye_center - 50):
                    eye_c = 1
                else:
                    eye_c = 2
#初期化
            eye_y = eye_y - y
            eye_x = eye_x - x
        else:
            eye_y2 = eye_y + circles[0,0,1]
            eye_a2 = eye_x
            eye_x2 = eye_x + circles[0,0,0]
            eye_b2 = eye_x + w

#           print eye_x2,eye_y2
            eye_y =eye_y - y
            eye_x =eye_x - x

else:
    print("no eye")
#eye_x1 が右目のとき
if eye_x1 > eye_x2:
    #x1 を左目にする
    tmp = eye_x2
    eye_x2 = eye_x1
    eye_x1 = tmp
    tmp = eye_y2
    eye_y2 = eye_y1
    eye_y1 = tmp

if eye_b1 > eye_b2:
    tmp = eye_b2
    eye_b2 = eye_b1
    eye_b1 = tmp
    tmp = eye_a2
    eye_a2 = eye_a1
    eye_a1 = tmp

if (eye_a2-eye_b1) < (0.4*face_pix):
    face_c = 4
    if (eye_a2-eye_b1) < (0.6*face_pix):
        face_c = 5
    else:
        face_c = 6

print face_c,eye_c
if (face_c + eye_c) == 7:
    print("見てるよ")
else:
    print("見てない")

# 何かキーが押されたら終了
while(1):
    if cv2.waitKey(10) > 0:
        break
```