修士論文

駅構内の案内サインを対象とした
全天球画像による光学文字認識

2020 年度

比留川 翔哉

明星大学大学院
情報学研究科情報学専攻
19MJ-003

概要

日本の駅は訪日外国人や日本人共に迷いやすい．そのため駅構内を案内するシステムが望まれるが，駅の数が膨大であるためデータベースの作成・更新はコストがかかるので既存のシステムでは難しい．本研究では，「文字認識を用いた駅構内のナビゲーション」という最終目標とする．その最終目標を達成するために本論文では，全天球カメラで駅構内の複数の案内サインを撮影し，その案内サインに含まれる文字を認識することを目的とする．本目的を達成するために，生成型学習法で生成した学習画像と畳み込みニューラルネットワークで文字認識する手法を提案する．文字認識の入力画像は，全天球カメラで撮影された全天球画像内の文字画像を用いる．本提案手法は，全天球画像内の看板画像を抽出する．抽出した看板画像から文字や矢印，ピクトグラムを抽出し文字認識を行う．その文字認識の結果を辞書を用いた誤り訂正を行い正しい結果に訂正する．駅で撮影した全天球画像を用いた評価実験では，看板抽出と文字認識では高い精度で抽出・識別できたが，辞書による誤り訂正は正しい認識を誤った結果に訂正してしまう結果となった．今後の課題として，各提案手法の精度向上と最終目標を達成することが挙げられる．

目次

1　序論

1.1　背景

2019 年まで，日本を訪れる外国人観光客が年々増加していて，2800 万人以上が訪日している [1]．また観光の際，83.8% の割合でスマートフォンを利用して情報収集を行い，鉄道を利用して観光している [2]．しかし，電車を利用した外国人の 25% は，目的地へのルートや電車の乗り換えが分かりづらいと回答し，アンケートに答えた日本人の 50% も迷った経験があると回答している [3]．これらの問題に対して，様々な研究やサービスが開発・提供されている [4, 5, 6]．これらは事前に駅構内の構内図や路線情報などの事前情報をデータベース化し，使用している．2020 年 10 月現在，日本国内の駅数は 9206 駅 [7] あり，データベースを用いる既存システムでは作成とデータ更新に膨大なコストがかかる．そのため日本国内の駅全てに対応するのは困難である．一部の主要駅もしくは，乗り換え駅のみ対応させることは，コスト削減につながる．しかし，ナビゲーションを必要とする駅としない駅を完全に区別することは不可能であり，かつ，訪日外国人が利用する可能性があるすべての駅で必要であると考える．

駅構内には，駅利用者に駅構造や出口などを情報提供する案内サインが設置されている．案内サインには吊り下げ型案内サイン (図 1(a))・壁設置型案内サイン (図 1(b))・床設置型案内サイン (図 1(c)) が存在する．吊り下げ型案内サインは，駅利用者の動線[*1]上に設置されている．壁設置型はその動線付近の壁・柱に設置されている．床設置型は，改札外・ホーム上に設置されている場合が多い．これらの案内サインは，利用者を目的地への誘導するために設置されているが，矢印の方向が曖昧であったり，1 枚もしくは複数枚の案内サインの情報量が多いため見逃す等の問題がある．複数の案内サインによる情報過多の例を図 2 に示す．図 2 の情報は，目的の電車のホームへ移動する際に必要な案内が複数設置されているため，識別可能である．しかし，出口案内や精算機，忘れ物センターへの案内は，乗入れ路線への案内の個数が多すぎるため，見落とす可能性がある．頻繁に同じ駅を使用した利用者であれば，複数の案内サインの情報や，記憶した駅構造から目的地へたどり着くことは容易である．しかし，訪日外国人観光客やその駅の構造を熟知していない利用者では，たどり着くことが難

[*1] この場合，出口・改札・トイレ・各路線のホームの出入り口などの駅利用者が歩くと想定もしくは誘導した線を指す

しい場合がある．

本研究では，既存システムではすべての駅に対応させるのは困難であるため，別のアプローチで駅構内のナビゲーションの達成を目指す．そのため，案内サインと全天球カメラを用いて文字認識を行う．

1.2 目的

本研究では，最終目標である「案内サインを用いた駅構内のナビゲーション」を達成するために，全天球カメラで撮影した駅構内の案内サインを文字認識する手法を提案する．本研究は，以下の 5 つのフェーズを含む．

1. 全天球カメラからの看板画像の抽出
2. 全天球カメラによる歪みの緩和
3. 案内サインに含まれる文字のセグメンテーション
4. 機械学習を用いた文字認識
5. 辞書を利用した文字認識結果の訂正

(1) 案内サインの抽出は，全天球カメラで撮影された全天球画像から看板と思われる領域を抽出することを目的とする．このフェーズは，全天球カメラで撮影された看板と思われる領域の文字を認識するために行う．このフェーズでは，看板だけではなく同じ色相範囲のすべての領域を抽出する．看板もしくはそれ以外の領域の判定は，歪み緩和時の判定と文字認識の結果を用いて判別する．本論文では，看板の領域を取得できる精度を算出し，駅構内の案内サインに対する有効性や，案内サインのタイプによる精度差などの検証を行う．

(2) 全天球カメラによる歪みの緩和は，湾曲した看板画像を矩形に戻し，文字のセグメンテーションの精度を向上することを目的とする．この歪み緩和は，前フェーズで取得した領域の外形情報を利用し，全天球カメラによる歪みを緩和することができる．本論文では，歪みの緩和の有効性と歪み緩和による文字認識の精度の変化を検証する．

(3) 案内サインに含まれる文字のセグメンテーションは，歪み緩和時に取得できる背景色の色相範囲を用いて背景色を削除し，残った文字を取得することを目的とする．本フェーズは，第 4 フェーズで 1 文字ずつ文字認識を行うために文字のセグメンテーションを行う．本論文では，取得できた文字数から精度を算出し，正しく文字を抽出できているか検証する．

(a) 吊り下げ型案内サイン

(b) 壁設置型案内サイン

(c) 床設置型案内サイン

図 1　日本国内の駅にある案内サイン

図 2　複数の案内サインによる情報過多の問題の一例 [8]

(4) 機械学習を用いた文字認識は，前フェーズまでに抽出できた文字画像を用いて文字認識を行うことを目的とする．認識には，畳み込みニューラルネットワーク (以下，「CNN[*2]」とする．) を用いる．本論文では，認識できた文字数から精度を算出し，認識率を算出する．

(5) 文字認識結果を訂正する辞書の利用は，前フェーズの認識結果を正しい認識結果に訂正し最終的な結果の精度を向上させることを目的とする．本論文では，文字認識の結果を正しく訂正できた割合と，正しい結果を誤った結果に訂正した割合を算出する．

1.3　本論文の構成

本論文の以降の構成は次の通りである．第 2 章では，光学文字認識の現状や問題点について述べる．第 3 章では，各フェーズの先行研究と関連研究について述べる．第 4 章では，各フェーズを含む手法の詳細，第 5 章では提案手法の具体的な実装方法，第 7 章では提案手法の有効性を検証するための実験内容と実験結果について述べる．第 8 章と第 9 章では，第 7 章の実験結果に対する考察と導出される結論，提案手法の課題について述べる．

[*2] Convolutional Neural Network

2　光学文字認識の現状と問題点

2.1　光学文字認識

光学文字認識 (Optical Character Recognition，以下，「OCR」という．) は，文字の形状を含む画像から文字コードに変換する技術である．近年では，Google Cloud Vision API [9]，Adobe Acrobat [10], などの OCR をサポートするサービスやアプリケーションが存在する．また，Evernote [11]，Google Drive [12] などのクラウドにアップロードすることで OCR を使用できるサービスも多く存在する．光学文字認識の研究においては様々な手法が提案されており，テンプレートマッチング，CNN や再帰型ニューラルネットワーク (以下「RNN[*3]」という．)，CNN と RNN を混合させた再帰型畳み込みニューラルネットワーク (以下，「CRNN[*4]」という．) など，様々な手法が提案されている．

多くの光学文字認識では，入力された文字画像が事前に作成した辞書データと比較し，最も特徴量が近い文字を認識結果として出力する．辞書作成に想定されていない環境や対象物を撮影した画像を入力データとして使用すると，誤った認識結果が出力される場合がある．そのため一般的に光学文字認識を利用する前処理として，入力データに文字認識が利用している辞書データに近似させる補正を行う．

OCR を使用して，テキストデータに変換することにより，以下のようなことが行える．

- 文章の加筆・修正
- 文章の検索
- 文章の二次利用

文章の二次利用の例として，紙媒体管理からデジタルデータ管理への移行の補助や，案内看板の文字認識による自律思考型ロボットの位置特定補助，駅構内の案内サインの文字認識による国内・海外旅行者のナビゲーションシステムなどが考えられる．

本研究では，「案内サインを用いた駅構内のナビゲーション」を最終目標とした文字認識の手法を提案する．

[*3] Recurrent Neural Network

[*4] Convolutional Recurrent Neural Network

2.2 既存 OCR が使用する補正

現在の OCR は，精度向上のために画像の自動補正を行った上で文字認識を行う．主な自動補正の種類は以下の通りである．

- 平滑化 (図 3)
- 輝度補正 (図 4)
- 歪み補正 (図 5)

図 3 の平滑化は，適切に補正を行うことで，文字認識の精度低下につながるノイズの除去による精度向上が見込まれる．平滑化には，モルフォロジー変換やフィルタを用いた処理が利用されることが多い．適切に補正を行わなかった場合，文字形状の破壊やノイズの不完全除去による精度低下の原因になる．図 3(a) は，「あ」の文字画像にノイズを付加した文字画像である．付加したノイズは赤い丸で囲んであるところである．図 3(b) と (c) は，平滑化を行った画像である．(b) は手動で適切に補正を行いノイズが完全に除去できた画像，(c) は適切ではない値を使用して平滑化を行いノイズが残った文字画像である．

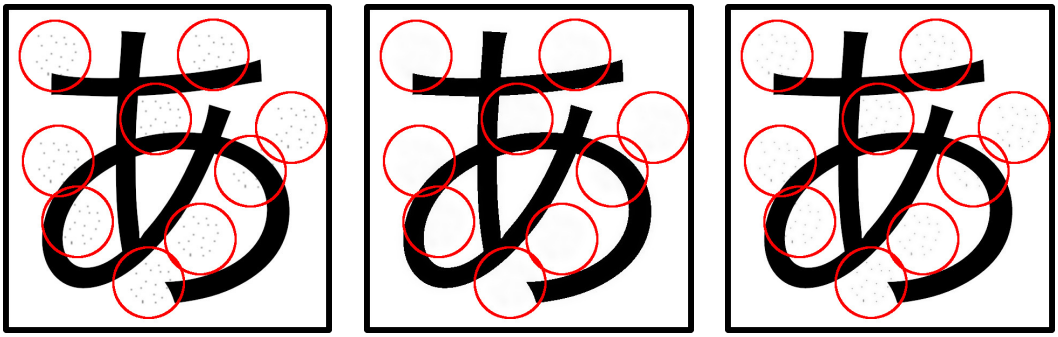
図 4 の輝度補正は，適切に補正を行うことで，背景色と文字色の輝度を一定にすることによる精度向上が見込める．輝度補正は，ゲイン調整やバイアス調整をが利用されることが多い適切に補正を行わなかった場合，文字の白飛びや，文字形状の特徴量欠落による精度低下の原因になる．図 4(a) は，「あ」の文字画像にグラデーションを加えた文字画像である．図 4(b) と (c) は，輝度補正を行った画像である．(b) は手動で適切に補正を行い背景色が一定であり，かつ，文字形状が完全に残った文字画像，(c) は適切ではない値を使用して輝度補正を行い背景色はほぼ一定だが，文字形状が破壊された文字画像である．

図 5 の歪み補正は，適切に補正を行うことで湾曲や変形を修正し，学習済みデータ内の特徴量に近似させる．近似させることで，精度向上が見込める．しかし，適切に補正を行わなかった場合，複雑な変形や強い歪みなどの完全に歪み補正が行えないため，精度低下の原因になる．図 5(a) は，「あ」の文字画像を歪み加工を行った文字画像である．図 5(b) と (c) は，歪み補正を行った画像である．(b) は手動で適切に補正を行い歪みを除去した文字画像，(c) は適切ではない値を使用して歪み補正を行い新たな歪みが発生した文字画像である．

これらの補正は，適切に行うと事前に取得した特徴量や学習済みデータに近似することができるため，精度が向上する．しかし，適切に行えていない補正は，事前に取得したデータとの乖離が補正前より大きくなる可能性があるため，精度が低下することが多い．そのため，完全な補正を行える手法，もしくは，補正を行わない手法が最も精度が高くなると考える．

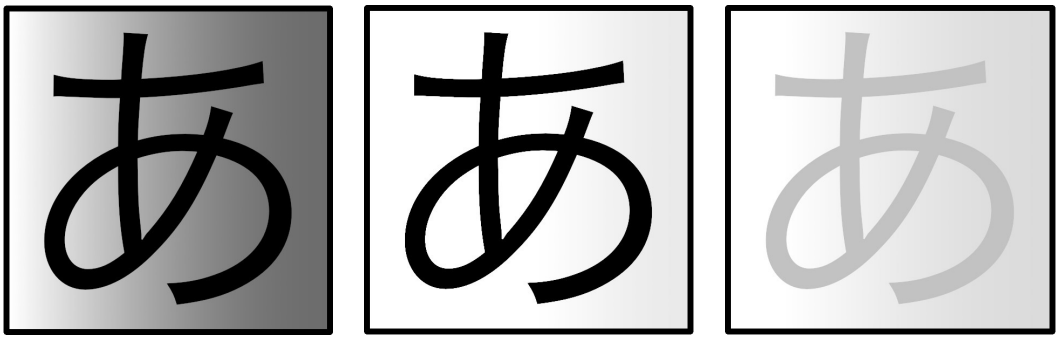
2.3　全天球カメラを用いた文字認識

全天球カメラは，360 度の情景を静止画もしくは動画で全天球画像として撮影可能なカメラである．カメラのタイプにより様々な撮影方法で全天球画像を生成する．本研究で全天球カメラを用いる理由は，以下の 2 つである．第一の理由として，駅構内



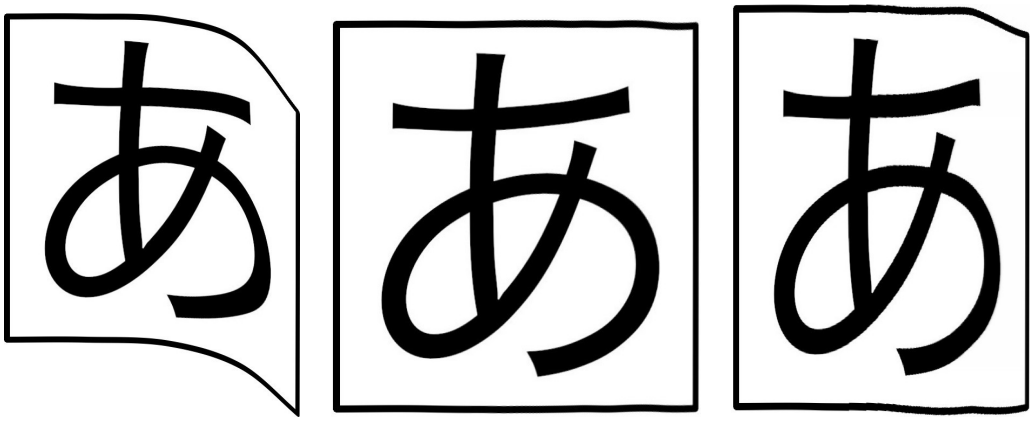
(a) 平滑化前の文字画像　(b) 適切に平滑化した文字画像　(c) 適切に平滑化していない文字画像

図 3　平滑化の補正の画像



(a) 輝度補正前の文字画像　(b) 適切に輝度補正した文字画像　(c) 適切に輝度補正していない文字画像

図 4　輝度補正の画像



(a) 歪み補正前の文字画像　(b) 適切に歪み補正した文字画像　(c) 適切に歪み補正していない文字画像

図 5　歪み補正の画像

のナビゲーションのために通常のカメラを使用する場合，ナビゲーション利用者の進行方向しか撮影できない．進行方向以外をナビゲーション利用中に撮影すると，進行方向以外の撮影方向を見ることになるので，非常に危険であり，かつ，目的地への情報を見逃す可能性がある．また，事前情報を使用しないナビゲーションでは，利用者の周辺状況や目的地と利用者の相対位置など通常のナビゲーションでは利用できる情報が利用できない．そのため，通常では撮影できない看板の裏面の情報や，進行方向以外の看板など，通常のカメラでは見落としてしまう情報を活用することで情報量を補うことができると考える．全天球カメラと通常のカメラで撮影した画角の違いを図 6 に示す．図 6 は，ほぼ同位置で撮影した画像で，図 6(a) は全天球カメラで撮影した画像，図 6(b) はスマートフォンで撮影された画像である．図 7 のように，文字に対する解像度はスマートフォンで撮影した画像のほうが高いため文字認識は有利である．しかし，生成型学習法を用いるため解像度の違いは無視できると考える．また，図 6(a) のように，全天球カメラでは全方向撮影可能であるため，スマートフォンより全天球カメラのほうが取得できる情報が圧倒的に多い．

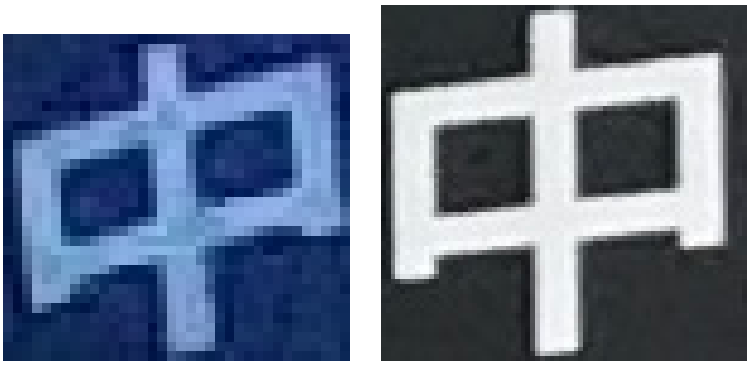
第二の理由として，複数の撮影対象がある場合，1 回の撮影で全方向撮影可能であるため，複数のカメラを用いるより手軽に撮影できる．駅構内のナビゲーションに使用する場合，複数のカメラを用いる手法はセットアップが非常に手間がかかるため，1 台の市販の全天球カメラを用いる手法が手軽に利用できると考える．

以上の 2 点から，本研究では全天球カメラを用いる．

(a) 全天球カメラで撮影した JR 八王子駅 (6720 × 3360 ピクセル)

(b) スマートフォンで撮影した JR 八王子駅 (4032 × 3024 ピクセル)

図 6　JR 八王子駅の駅構内におけるスマートフォンのカメラと全天球カメラの画角の違い



(a) 全天球カメラで撮影した「中」(36 × 34 ピクセル)　(b) スマートフォンで撮影した「中」(76 × 74 ピクセル)

図 7　カメラによる文字の解像度の違い

3　関連研究

3.1　補正に関する研究

荒木ら [13] は，図 8 のように，書籍を裁断せずにスキャンした画像の補正を行った．対象とした補正は，本の厚みによる歪みや輝度・文字ボケである．この手法では，スマートフォンのカメラで撮影した書籍の輪郭の取得が難しい画像では正しく補正を行う事ができない．また，OCR に使用する補正方法を提案しているため，本研究とはアプローチが異なる．

志久ら [14] は，傾斜文字を補正して文字認識を行う手法を提案している．この手法は，図 9 のように傾斜文字を囲う文字枠を作成し，その文字枠を正方形に補正する．その後に，CNN を用いて文字認識を行っている．しかし，この手法では，三次元の変形や本の厚みによる湾曲した歪みなどの文字の認識は行う事が出来ない．成田ら [15] は，スキャナで取得した歪みのない英数字の文字画像を図 10 のように x，y，z 軸で回転させる．その回転させた画像を学習し，看板の文字を認識する手法を提案している．しかし，この手法では，スキャナで歪んでいない文字を取得させてから学習データを作成するため，認識させる文字種を増やすことが困難である．また，三次元の回転以外の湾曲した歪みなどの文字認識は精度が低下する．本研究は，補正を行わずに文字認識する手法を提案しており，想定する歪みも異なる．

3.2　生成型学習法に関する研究

本研究のように歪んだ文字を認識する研究には，生成型学習法を用いたもの [16, 17, 18, 19, 20] がある．これは認識対象の歪みを想定して学習画像を生成・データ拡張することで認識精度を上げる手法である．これらのデータ拡張は，ボケ (図 11(b))・ブレ (図 11(c))・低解像度 (図 11(d)) に対応するもの [17] や，文字のグリフを変更し一般的なフォントデータから特殊形状のフォントを認識するもの [18]，書籍の厚みで歪んだ文字を認識するもの [19] がある．筆者の先行研究 [20] では，スマートフォンのカメラで撮影した案内サインを生成型学習法で認識可能かを調査した．この研究は，スマートフォンの画像を用いて撮影された案内サインを，日本語は 61.207%，英語は 23.636% で認識した．英語の認識率が悪い理由は，案内サインにおける英語は日本語の補助に使用されるため，英語のフォントサイズが小さい．そのため，英語の会場度

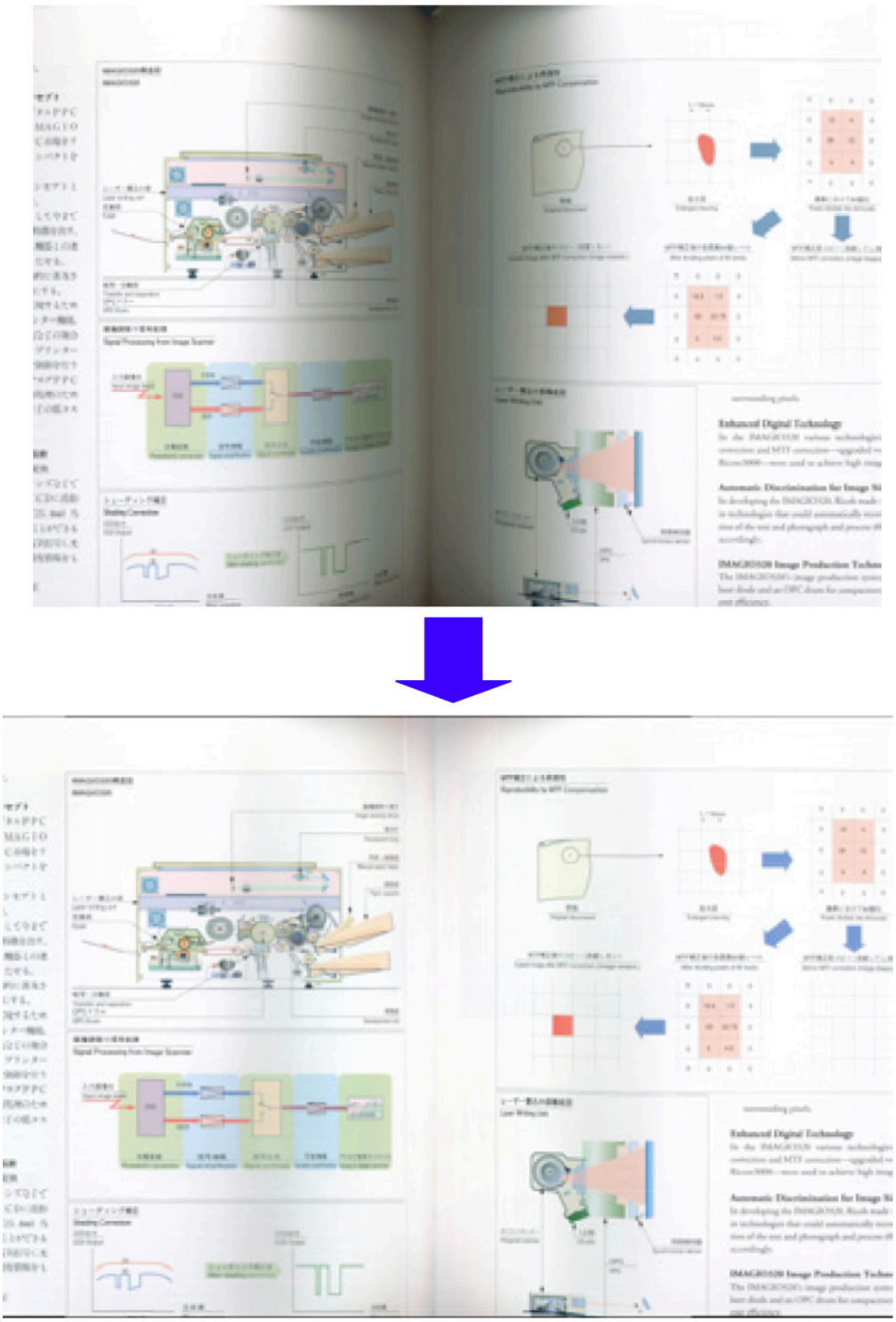


図 8　荒木ら [13] の提案手法による補正結果の画像

図 9　志久ら [14] の提案手法による補正結果の画像

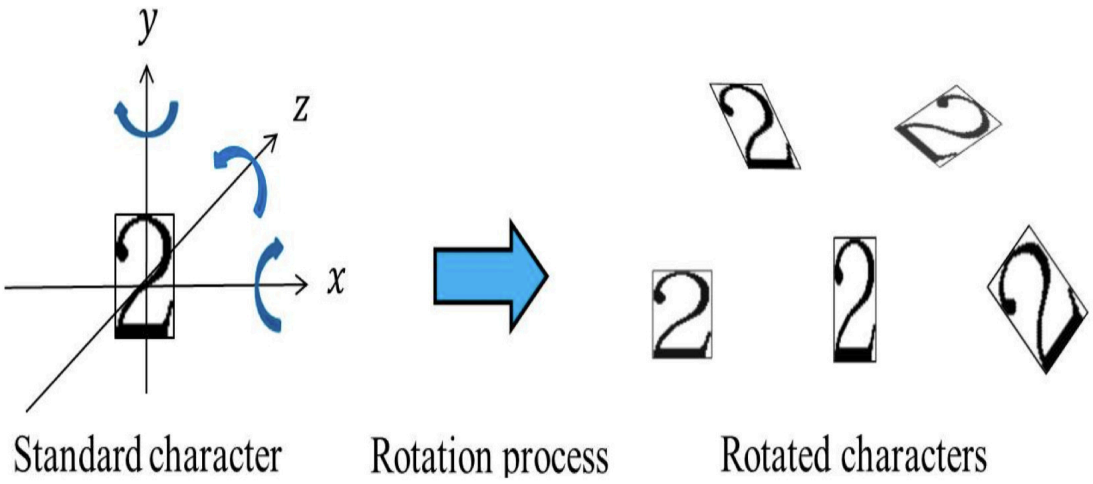


図 10　成田ら [15] の提案手法による生成文字の画像

は非常費低い．しかし，日本国内の案内サインであれば，図 1 のように日本語が識別できればナビゲーションに使用できる．また，日本語の認識結果は，後述する単語辞書を用いることにより精度向上が見込める．そのため本研究では案内サインの歪みに対して生成型学習法を使用する．

3.3　機械学習を用いた文字認識の研究

生成型学習法を使用しない光学文字認識も多くの研究がある．近年の標準的な文字認識の研究として，ニューラルネットワークを用いた日本語のひらがな・カタカナの文字認識の研究 [21] や中国語の文字認識 [22] がある．これらの研究は，認識するテストデータに変形等の補正を行い，学習データに近い文字の形にしなければならない．本研究では認識対象に合わせて様々な学習画像を生成し，無加工に近いテストデータ

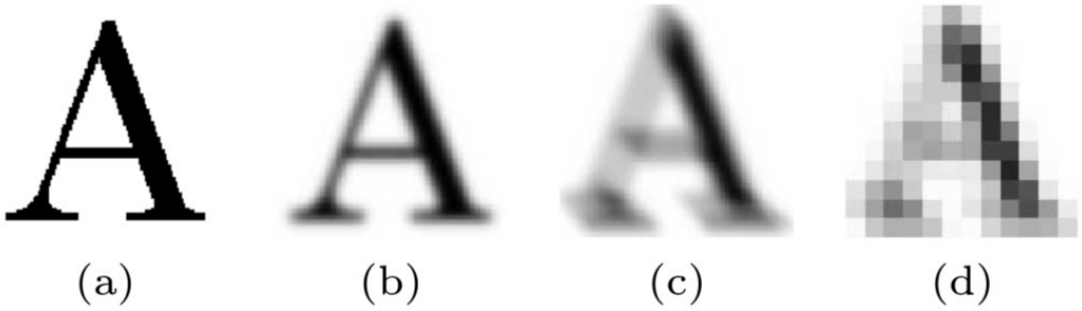


図 11　石田ら [17] の提案手法による生成文字の画像

を認識するためアプローチが異なる．また，CBAM[*5]を改良した ICBAM を用いた中国の店舗の看板の文字認識の手法も提案されている [23]．この認識方法は，ICBAM で抽出できる低レベル (文字の外形) の特徴量 (図 12(a)) と，Attention 特徴量 (図 12(b)) を用いて様々な文字を認識している．案内サイン内の文字は，照明による文字外形の破損している可能性があるため，低レベルの特徴量が正しく取得できない場合があると考える．

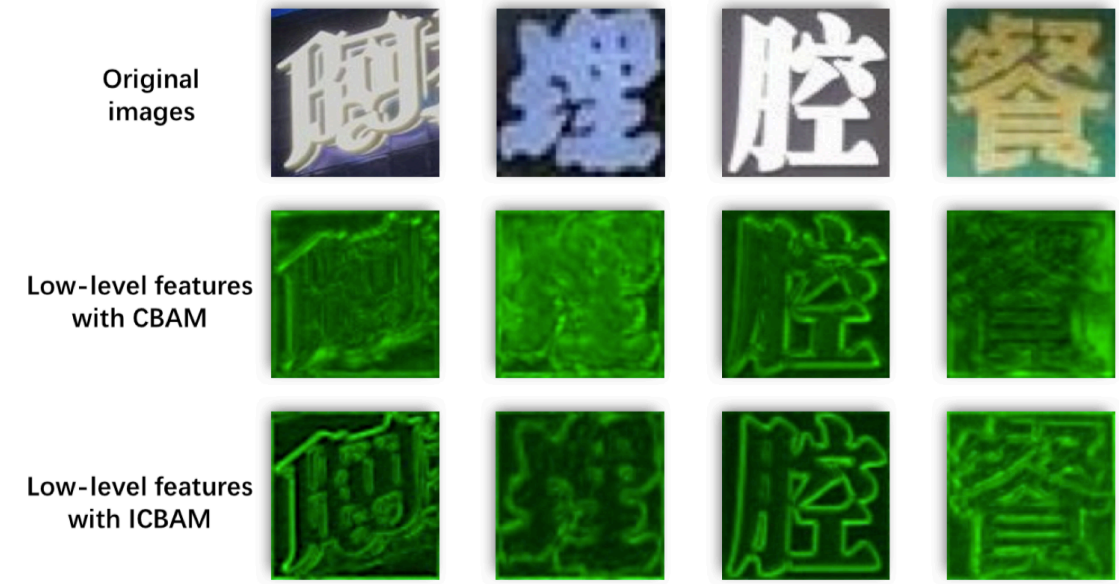
3.4　看板の抽出もしくは看板の文字を認識する研究

本提案手法の看板検出は，HSV 色空間と矩形判定を用いて屋外の看板を検出する手法 [24] がある．この手法は，カラー看板を 98.1%，白黒看板を 90.2% の精度で抽出した．本研究では，この手法に基づき案内サインを抽出する．

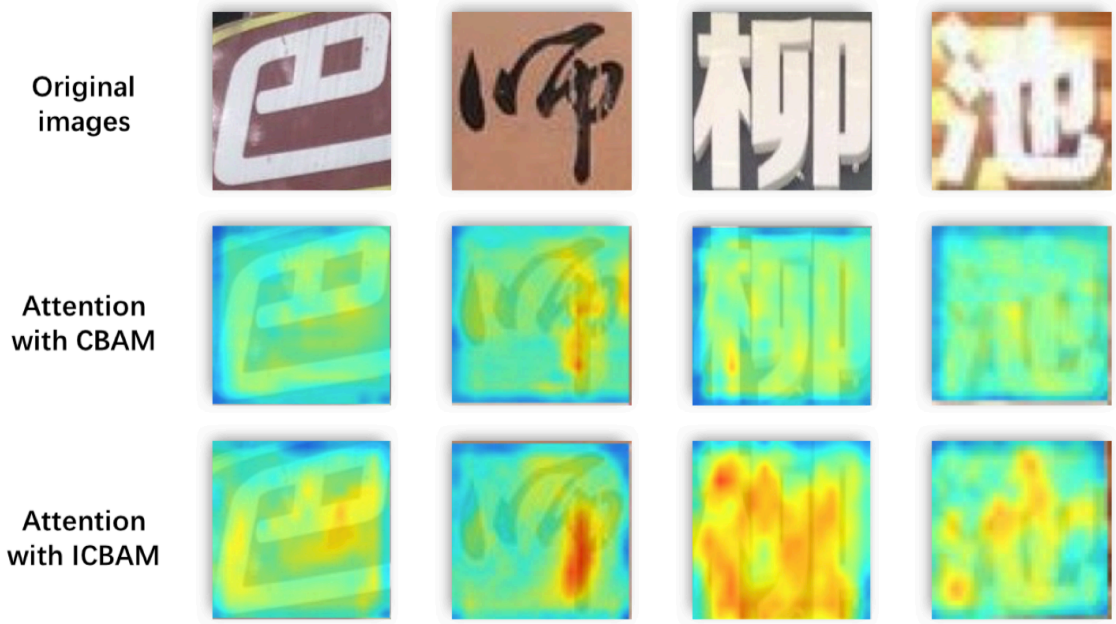
看板を文字認識する研究では，情景画像内の看板を検出・文字認識するテキストスポッティングの研究 [25, 26] や，韓国国内の駅構内の案内サインの出口番号と矢印を認識する研究 [27] がある．テキストスポッティングの文字認識では，膨大なラベル付与済みのデータセットを用いて単語単位で認識する．そのデータセットは駅案内サインに対応するものがないため，本研究では生成型学習法を用いて認識する．韓国国内の案内サインの認識の研究は，認識対象が日本語を含む文字であることと認識する手法に畳み込みニューラルネットワークを用いる点が異なる．

文字認識を用いたナビゲーションは，Android を搭載したミニカーとサーバを用いて案内サインを認識し，認識結果と RFID(Radio Frequency Identifier) を用いて目的地までの誘導を行う手法 [28] がある．この手法は，案内サインの文字を 2 値化し，

[*5] Convolutional Block Attention Module



(a) 低レベルの特徴量



(b) Attention 特徴量

図 12　ICBAM が抽出した特徴量 [23]

サーバによる文字認識を行う．本研究とは屋内のナビゲーションという目的は同じだが，RFID を使用しないため様々な建造物に対してナビゲーションが行える点と，様々な文字種がある日本の案内サインでは，認識するラベル数が膨大であるため認識できない点がある．

3.5 全天球カメラを用いた研究

全天球カメラを用いた補正や位置推定の方法は多数提案されている．石井ら [29] は，全天球カメラで撮影された画像に対し直線検出を行い，水平方向の補正を行う手法を提案した．本研究の認識対象である案内サインは図 1 のように矩形である．そのため，案内サインの形状情報に直線検出を用いることで，高精度に補正を行えると考える．また，Yashiro ら [30] は，全天球カメラで撮影した画像の特徴量と GPS を用いて建造物内の位置推定を行う手法を提案した．この手法は，全天球カメラで撮影した歪んだ画像の特徴量と通常のカメラで撮影した画像の特徴量を比較し，位置推定を行っている．本研究で用いる文字認識より単純な比較で位置推定を行っているため，全天球カメラと CNN を用いた文字認識は可能であると考える．

本研究と同じ目的である，スマートフォンに取り付けた全天球カメラと OCR と物体検出を用いて，点字ブロック上で撮影した案内サインの矢印と文字から目的地方向を通知するシステム [31] が提案されている．このシステムは，全天球カメラで撮影された画像から図 13 のように範囲内にある案内サイン内の矢印と文字の関連性と，物体検出による矢印の認識，Google Vision API による文字認識の 3 つから目的地方向を特定している．特定した方向とスマートフォンのジャイロセンサから「[n] 時方向 [目的地名]」($1 \leq n \leq 12$) と通知し，その方向と点字ブロックの分岐方向の正答率を算出している．このシステムでは，COVID-19 の対策として実験室内で再現し実験を行っているため，撮影環境が実際の駅や空港の環境と異なる．また，図 13 のように撮影範囲を限定しているため，撮影位置によっては案内サインが見切れる可能性がある．本研究では，全天球カメラを用いて実際の駅で撮影されたすべての案内サインを対象としている点や，点字ブロックを考慮しない点などの前提条件が異なる．また，矢印がない案内サインやピクトグラムを含む案内サインも対象とする点も異なる．

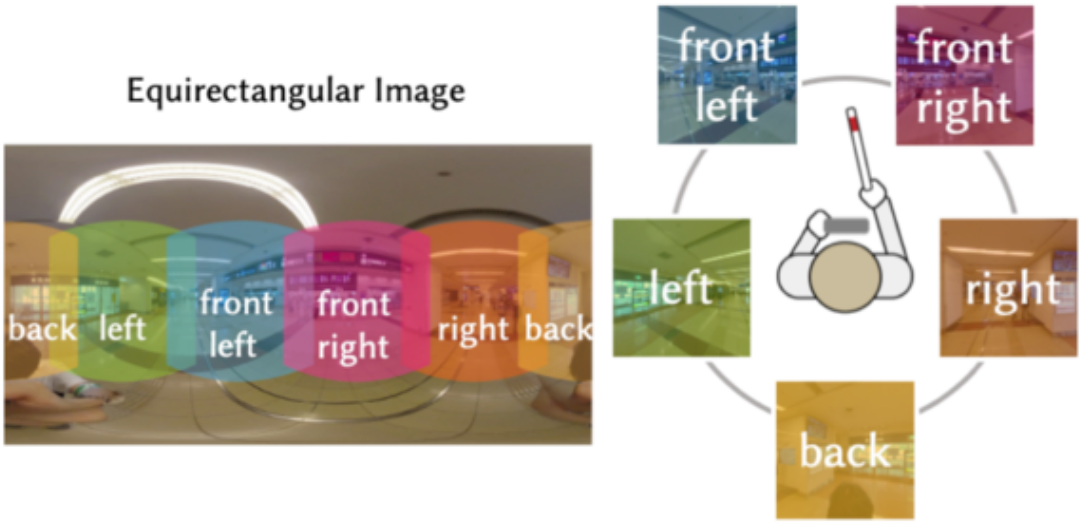


図 13　案内サインの矢印と文字から目的地方向を通知するシステム [31] の切り出し範囲

3.6　ナビゲーションに関するアプリケーションや研究

既存のナビゲーションアプリでも，駅構内や屋内のナビゲーションに対応したものがある [4, 5]．また，空港内に BLE(Bluetooth Low Energy) ビーコンを設置して視覚障害者をナビゲーションする研究 [32] や，LiDAR[*6]やデュアルカメラを搭載したスーツケース型のナビゲーションロボットの研究 [33] などがある．これらのアプリケーションは，GPS や加速度センサ，地磁気センサなどのスマートフォンに搭載されているセンサを用いたものや，建造物内に Wi-Fi や BLE などのビーコン，建造物の屋内構造の情報を使用したものである．これらの各センサや屋内構造の情報などの事前に調査してデータベース化して使用する．データベースを使用する手法はコストがかかるため，新規に対応させる建造物や工事による屋内構造の変化への対応が難しいという問題点がある．

プロジェクションマッピングを用いた動的案内サインによる人流の誘導の研究 [6] では，東京国際空港ターミナルの到着ロビーと発着口に混雑状況によって表示内容が変わる案内サインを表示させる手法を提案した．この手法は，出発口の待機列の人数が減少した．しかし，この手法は，プロジェクションマッピングが行える広い空間や，

[*6] Light Detection and Ranging

混雑状況の取得が必要であるため，駅には使用できない問題点がある．

本研究では，これらの問題点を解決するために，全天球カメラで撮影された案内サインを使用して文字認識を行う手法を使用する．

3.7 文字認識結果の修正に関する研究

文字認識の結果を修正する手法は古くから研究されている．1990 年から 2000 年代は自然言語処理を用いた手法が多く研究されていた [34, 35]．これらの手法では，本研究で対象とするような名詞のみの認識結果は，形態素解析や構文解析が行えないため，修正することは難しい．また，ピクトグラムや矢印など自然言語処理では対処できないものは，これらの手法では修正することができない．

近年の ICDAR(International Conference on Document Analysis and Recognition) でも，文字認識の結果に対してエラー検知とエラー修正の 2 つの方面から様々な手法が提案されている [36]．これらの手法で使用されている辞書の言語は英語とフランス語で，辞書はデジタル書籍や新聞，ウィキソースなどで構築されている．この辞書を元にエラー検知・修正を行っている．提案されている手法は，5 グラム言語モデルとレーベンシュタイン距離を使用した Kneser-Ney スムージングの手法や，RNN の一種である LSTM(Long-Short Term Memory) モデルを用いた手法，LSTM モデルと Attention 機構を用いた seq2seq モデルの手法などがある．これらは，英語とフランス語を対象としているが，辞書の言語を日本語にすることにより，日本語のエラー検知・修正は可能であると考える．しかし，前述の通り名詞のみの認識結果は，様々な解析が行えないため，語句の関係性が学習できない．また，ピクトグラムや矢印も，これらの手法では修正することは難しい．

4　提案手法

本研究の提案手法は，全天球カメラで撮影された画像から看板画像を抽出し，その看板画像の全天球カメラによる歪みの緩和を行う．歪みを緩和した看板画像を 1 文字の画像にセグメンテーションを行い，CNN を用いて文字認識をする．文字認識を行う CNN は，生成型学習法を用いて学習画像を生成し学習する．認識結果の精度は，カメラの設定や各セグメンテーションの精度によって大きく差がでると予想できる．そのため，辞書を用いて間違った認識結果を訂正し，最終的な認識結果とする．

本提案手法では以下の順序で文字認識を行う．本章では，以下の各手順について説明する．

1. 全天球カメラを用いた撮影
2. 全天球画像を用いた看板抽出
3. 全天球カメラによる歪みの緩和
4. 文字セグメンテーション
5. CNN を用いた文字認識
6. 辞書を利用した文字認識結果の訂正

4.1　全天球カメラを用いた撮影

本研究では，「駅構内の情報を利用せずに駅構内のナビゲーション」を想定した撮影方法で撮影された画像を用いる．撮影に使用するカメラは，2 つの魚眼レンズで前後を撮影する「RICOH THETA Z1」 [37] を使用する (図 14)．このカメラで撮影した図 6(a) のような全天球画像を使用して文字認識を行う．

撮影方法は，駅構内で歩行中の撮影者が全天球カメラを頭上に持ち上げて撮影を行う．この撮影方法は，撮影者の身体による案内サインの遮蔽が引き起こすことがないため，情報の欠落が起こらない．

また，撮影者の身長によるが，他の駅利用者による案内サインの遮蔽も引き起こすことが少ない．本研究では，身長 170 cm 以上の撮影者 2 名で撮影を行ったが，他の駅利用者と案内サインは被らなかった．身長 170 cm 未満の場合，他の駅利用者と案内サインが被るってしまう問題が起こる可能性がある．このカメラの底部には，三脚や自撮り棒がマウントできるネジ穴がある．そのため，低身長の場合，高さを延長しス

マートフォンで遠隔操作で撮影することにより，この問題を解決できると考える．

4.2　全天球画像を用いた看板抽出

本フェーズでは，全天球カメラで撮影された画像を既存研究 [24] の手法を用いて案内サインを抽出する．本フェーズによる処理のイメージを図 15 に示す．図 15 の一番上の画像は入力画像である全天球画像，下の 5 つの画像はそれぞれ赤 (Red)・青 (Blue)・緑 (Green)・白 (White)・黒 (Black) の色相範囲による抽出結果である．

この抽出手法は，天候や照明などによる影響を受けにくい HSV 色空間に変換し，赤・緑・青の色相範囲を抽出し，矩形判定を用いて看板を抽出する．抽出時には，その看板の相対位置と案内サインの背景色の情報も取得する．抽出に用いた色相範囲を図 16 に示す．抽出された画像の外形を取得し，その外形の範囲を用いて入力画像である全天球画像を切り抜く．本フェーズでは，案内サインと色相範囲内にある天井や壁などのノイズを含む画像を抽出する．これは，建物内の照明やカメラの設定により入力画像内の看板の色が変わってしまう可能性があるためである．そのため，このフェーズでは案内サインの取りこぼしを防ぐために案内サイン以外も抽出する．

色相範囲を使用する手法以外にも，画像内の直線を検出して抽出・補正を行う手法が考えられる．しかし，本研究の対象は駅構内にある案内サインであるため，壁や床や天井，駅構内にある設置物による誤検出が膨大になる．実際に駅構内の全天球画像

図 14　本研究で使用した全天球カメラ「RICOH THETA Z1」 [37]

に直線検出を行い，可視化した図を図 17 に示す．図 17 は，全天球画像にエッジ抽出を行い直線検出を行った結果を入力画像にプロットした画像である．図 17(a) はすべての直線を出力した結果，図 17(b) は垂直方向の直線であり，かつ，長さが閾値以上の直線を出力した結果である．直線の数が図 17(b) の本数程度にしぼり込める場合，補正は可能だと思われる．しかし，この結果をすべての駅で撮影された全天球画像に適応するためには，エッジ抽出に使用する 2 値化時の閾値と，検出された直線の長さを判定する閾値の 2 つを自動で適切に選定する必要がある．屋内は照明の影響が強く出るため自動で選定するのは困難であると考え，本研究では使用しない．また，本研究は全天球カメラで撮影された画像を使用するため，水平方向の湾曲した歪みが非常に強い．図 15 の右下の看板の抽出した画像のように，水平方向の歪みや文字の外形が強く影響するため，そのまま直線検出を使用するのは難しい．水平方向の直線検出した結果を図 18 に示す．図 18(a) と図 18(b) は図 18(c) をプロットする処理の途中経過の図である．図 18(a) は外形に対し接線のような直線と，文字の直線を検出した直線の 2 つのパターンが確認できる．接線と看板が接地している座標を複数取得することにより外形が取得できる．しかし，図 18(a) は処理の途中経過をプロットした画像であるため，歪みの緩和には使用できない．直線検出から歪みを緩和するには，図 18(c) から接線を抽出する方法や，歪み対する接線のみを検出する方法が必要である．

4.3　全天球カメラによる歪みの緩和

本フェーズでは，前フェーズで取得したノイズ画像を含む案内サインの外形情報を利用して全天球カメラの歪みを緩和する．本研究の条件では，看板の外形情報しか歪み緩和に使用できない．そのため，外形の上辺と下辺から歪み方を推定して歪みを緩和する．

本研究は，歪みの完全除去ではなく，歪み緩和のみ行う．歪みの完全除去には，水平方向と垂直方向の歪みを推定する必要がある．二方向の歪み推定には，字形や 1 行内の歪みなど対象内にある物を使用する方法や，カメラのレンズから歪みを推定する方法がある．字形や行から歪み方を取得する場合，看板内と看板外にノイズが多い本研究の条件では，ノイズが字形や行の誤認識に繋がり，使用するのが難しいと考える．また，レンズから歪みを推定する方法は，「RICOH THETA Z1」によって合成・展開された全天球画像を使用するためこの手法では困難である．Raw 画像を用いてキャリブレーションを行うことによりレンズから歪みを推定できる．しかし，全天球画像の

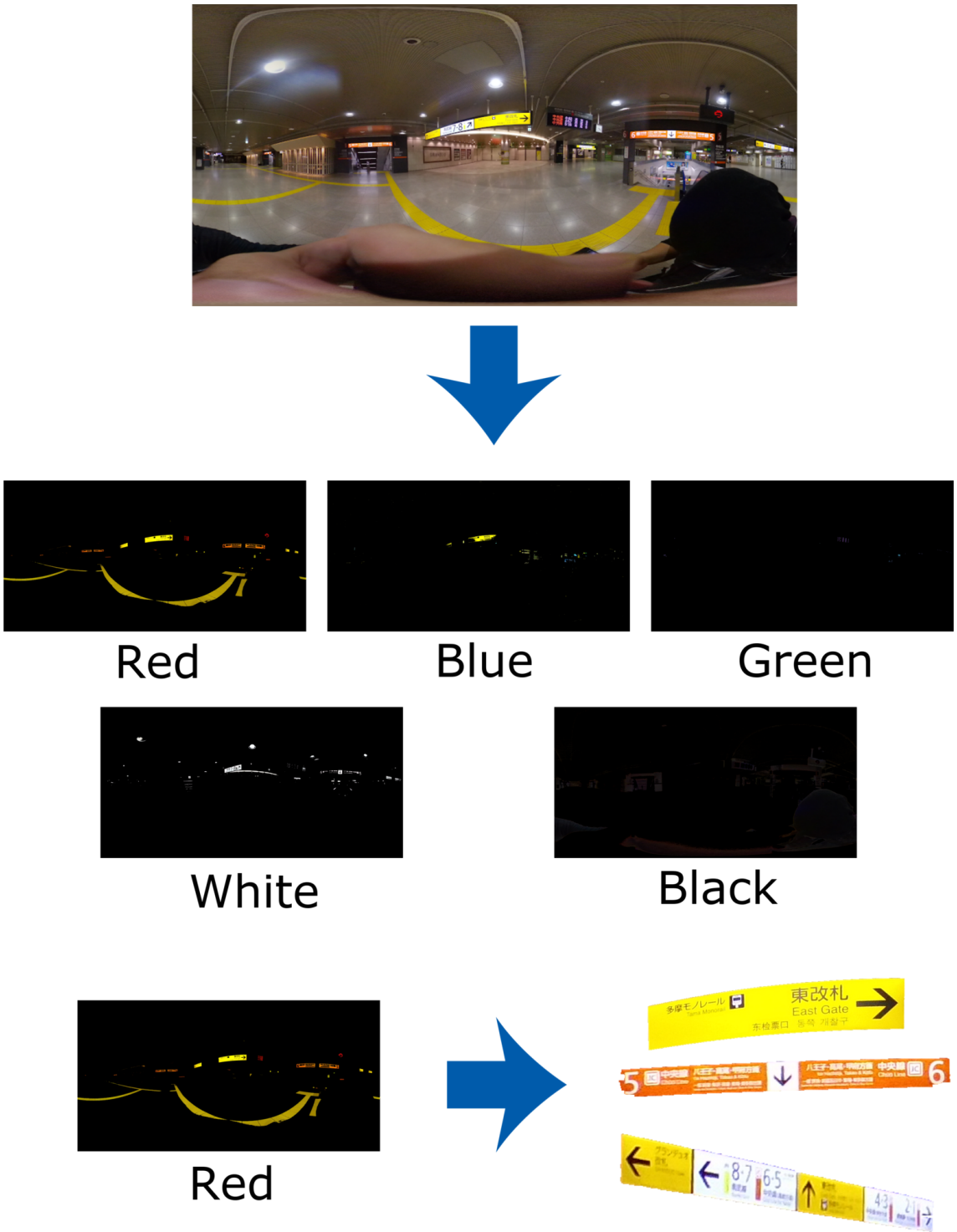


図 15　全天球画像を用いた看板抽出のイメージ図

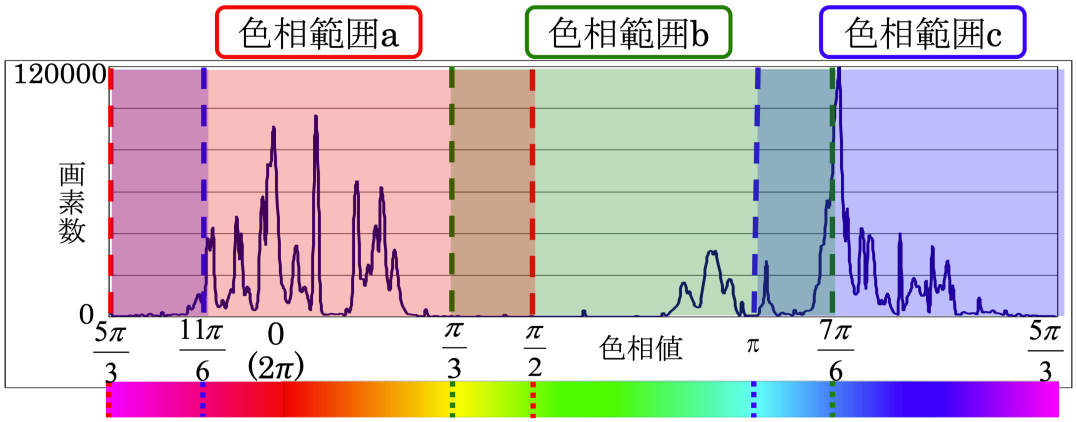
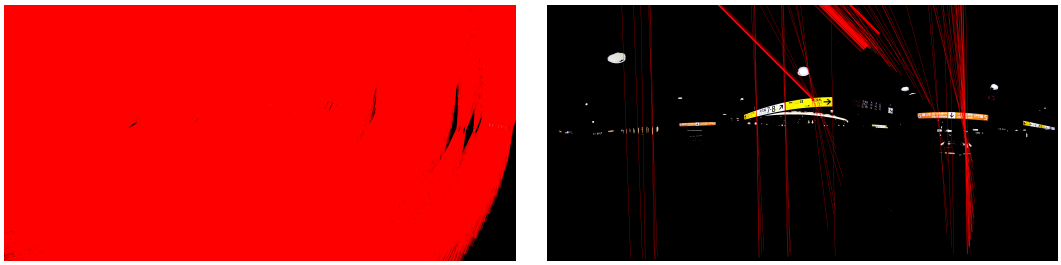


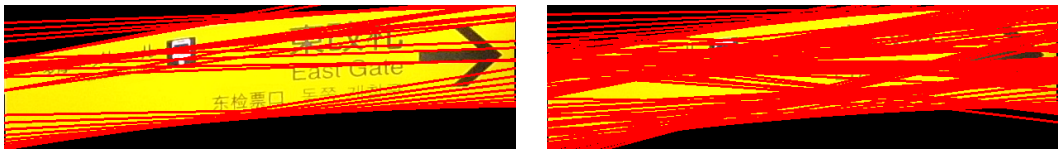
図 16　看板抽出に使用する色相範囲 [24]



(a) すべての直線を表示した画像

(b) 垂直方向であり，かつ，検出した長さが閾値以上の直線のみ出力した画像

図 17　駅構内の全天球画像に直線検出を行った失敗例



(a) 水平方向の直線の 30 本を表示させた画像

(b) 水平方向の直線の 100 本を表示させた画像

(c) 水平方向の直線のすべて (8911 本) を表示させた画像

図 18　案内サインに水平方向の直線検出を行った例

性質上，完全な平面にする場合は複数枚に分けて展開する必要があるため，今回は使用しない．

4.4　文字セグメンテーション

本フェーズは，案内サインの抽出時の色相範囲から推測できる背景色を削除し，案内サイン内にあるピクトグラムと矢印と文字の情報を抽出する．壁・天井・床などのノイズ画像は殆どの場合，単色で抽出される．そのため，案内サイン以外の情報は破棄される．

抽出された文字は，文字種によって複数のパーツに分かれる可能性がある．複数のパーツに分かれる文字は，「う」「は」「i」などが考えられる．そのため行の幅と高さから 1 文字か複数文字かを判定し結合する．

複数のパーツに分かれる文字の結合には，k 平均法 (k-means 法) のクラスタ数を自動で推定する g-means 法や x-means 法などを使用する方法がある．これらのクラスタ分析では，各文字・ピクトグラムのサイズの違いや，歪み方によって正しく分類できない．

4.5　CNN を用いた文字認識

本フェーズでは，生成型学習法と CNN を用いた研究 [19, 20] に基づいて文字認識を行う．

4.5.1　生成型学習法とは

画像認識や文字認識を行う場合，認識させる画像を想定して学習画像を用意し，学習させなければならない．文字認識や識別する種類が多い画像認識では，学習画像の撮影や取得が難しい．本研究では，様々な条件を考慮するため，手動で撮影することが不可能な枚数を用意する必要がある．また，本研究では全天球カメラを使用するため，インターネット上で公開されている画像では不足し，利用可能なデータセットは現在確認できていない．この問題点を解決する手法が生成型学習法 [16] である．生成型学習法は，原画像を加工して学習画像を生成する手法である．

生成型学習法のメリットとして，以下の 2 つが挙げられる．

1. 学習画像の撮影が不要

2. 様々な条件を付与した学習画像が生成可能

生成型学習法は，コンピュータ上で学習画像を生成するため，認識対象である案内サイン内の文字を撮影せずに学習することができる．日本語の文字認識の場合，ひらがなやカタカナ，漢字，アルファベットなどの様々な文字種の学習画像を用意する必要がある．さらに，各文字の学習画像も様々な条件で撮影された画像が必要になるため，全ての条件の画像を撮影して用意することは不可能である．そのため，膨大な数の学習画像を用意する必要がある文字認識では有効な手段であると考える．また，生成時に背景色や文字色，歪みなどの条件を加えることにより，補正を行わずに様々な環境で撮影された文字画像の認識ができる [19, 20]．補正を行わずに認識することができるため，第 2.2 節で述べた不適切な補正による精度低下の問題を文字認識では回避できる．本研究では，原画像をフォントデータ，原画像の加工には，過去に行った研究 [19, 20] の手法を使用して全天球カメラの歪みを再現する．

4.5.2 CNN とは

CNN は, 入力層 (Input) と畳み込み層 (Convolution Layer), プーリング層 (Pooling Layer)，出力層 (Output) で構成されるニューラルネットワークの一種である．畳み込み層は，入力画像を指定したフィルタを使用し畳み込み演算を行う層である．プーリング層は，指定した演算方法で特徴量を残しつつサイズを小さくする層である．

CNN の特徴として合成性と移動不変性に強いことが挙げられる．合成性は，画像の特徴を層ごとに抽出し，組み合わせて利用することである．これにより，上位の層では輪郭を抽出し，下位の層では詳細を抽出するなど層の構成を理解するのが容易である．移動不変性は，フィルタを使用して特徴を抽出するため，学習画像とテスト画像の検出物の位置変動に頑健であることを指す．移動不変性により，学習画像とテスト画像の文字位置を合わせる必要がない．

4.5.3 他の認識・識別方法との比較

CNN の他にもテンプレートマッチングや，部分空間法，サポートベクターマシン (Support Vector Machine，以下，「SVM」という．)，k 近傍法を使用した文字認識を行う手法がある．また，時系列データを扱う認識方法として，RNN や CRNN がある．

テンプレートマッチングは，入力画像の端からテンプレート画像で走査し，類似度が高いラベルと位置を検出する手法である．この手法は，単純なアルゴリズムであるた

め理解しやすいが処理が遅い．無補正の識別の場合，生成型学習法で生成する画像では精度が低下する可能性があるため使用出来ない．部分空間法は，各クラスごとの部分空間に近似しているかを比較し識別する手法である．この手法では，未知のパターンに強いが，識別するクラスが多い文字認識では次元数が増えるため不向きである．SVM は，線形しきい素子を用いて識別する手法である．この手法では，未知のパターンに強いが，基本的に 2 クラスの分類に使用されるため文字認識では不向きである．k 近傍法は，入力画像と近い学習データ k 個の多数決で識別する手法である．この手法は，単純なアルゴリズムであるため理解しやすいが，k の値によっては精度低下につながる．さらに，学習データが多ければ識別に時間がかかる．そのため，この手法は使用できない．RNN や CRNN では時系列データを扱う識別に使われる．文字認識では複数の文字で構成された単語の認識で近年使用されている．本研究のように識別する単語数が多い場合では，識別するクラス数が膨大になるため精度低下が懸念される．

以上より CNN 以外は，生成型学習法を用いた文字認識ではデメリットが大きい．そのため，本研究では CNN を使用する．

4.6　生成型学習法による学習画像の生成

本提案手法では，前述の通り学習画像をすべて生成する．生成する条件は，認識対象である案内サインを全天球カメラで撮影された文字に類似するように設定を行う．また，駅構内のナビゲーションを想定した場合，文字以外にもピクトグラムや矢印の情報の認識が必要になるため，JIS Z 8210 で規格化されているピクトグラムと矢印を元に生成する．

4.7　辞書を利用した文字認識結果の訂正

文字認識後の認識結果は，看板や文字のセグメンテーションが失敗した画像や，文字画像が目視でも識別できない画像の結果が含まれている場合がある．これらの結果は，正しく認識することができないため，誤った結果を出力している可能性がある．そのため，鉄道の路線名や駅名，駅周辺のランドマークの名称などの単語辞書を使用することにより正しい結果に修正する．

5　実装

本章では，前章で説明した各フェーズの実装部分について詳細に説明する．使用する言語は「Python 3.8.6」である．

5.1　全天球画像を用いた看板抽出

本手法では，既存研究 [24] の手法を参考に看板抽出を行う．入力画像は，「RICOH THETA Z1」で撮影された全天球画像を HSV 色空間に変換し使用する．HSV 色空間に変換することにより，天候や照明などによる影響を受けにくくなり抽出精度が向上する．使用する色相範囲と色名の略称は，赤 (R1, R2)・緑 (G)・青 (Bl)・白 (W)・黒 (Bk) の 5 色である．既存研究 [24] では，白と黒の看板は色空間を使用せずに濃淡から判別する手法を使用していたが，駅構内で撮影された全天球カメラの画像では正しく抽出できなかったため，今回は色相範囲の条件で抽出を行う．ただし，白と黒の HSV 色空間の場合，色相 (Hue) の値が一意に決まらないため，全範囲とする．白と黒を除く各色相の範囲は図 16 の値を使用する．

本提案手法で使用する値の彩度 (Saturation)・明度 (Value) は，JR 八王子駅，JR 立川駅で撮影した画像から選定した．具体的な各値を表 1 に示す．各値の名称は [色名の略称]_[種類]_[範囲] になっている．OpenCV の HSV 色空間の範囲は色相が 0 から 179，彩度と明度が 0 から 255 になっている．赤の色相範囲が 2 つに別れている理由は，Python の OpenCV の仕様上，色相を 179 より大きい値に指定した場合正しく動作しない．そのため，複数の色相範囲を用いる．これらの値を用いて色相範囲内にある色のみを抽出するマスクを作成して，看板画像のみの画像を生成する．この時に抽出できた 1 枚の画像から各看板のセグメンテーションを行う．

各看板のセグメンテーションは，抽出された看板の画像の外形情報を用いてセグメンテーションを行う．外形の検出には OpenCV の findContours メソッドを用いる．このメソッドは本来であれば精度向上のため 2 値画像を使用するべきだが，看板の色相範囲をすでに抽出済みのため看板以外の背景はすべて透過されている．この状態の画像の外形検出は，背景以外のすべての外形を取得できるため問題ない．また，2 値画像を作成する場合，閾値を設定して 2 値化を行うが撮影環境が各駅によって異なるため閾値は一意に定まらない．適応型閾値も撮影環境によっては正しく 2 値化できないため使用するのが難しい．外形が検出できた場合，入力画像である全天球画像からそ

の範囲を切り出し，看板画像とする．

全天球画像から切り出す理由は，看板内には背景色の色相範囲外の文字色が使われることが多いため，看板とその中の文字を抽出するためである．

これらの工程を行い，全天球カメラで撮影した全天球画像から各案内サインの画像を抽出する．この抽出された画像は，床・壁・天井などの看板以外の画像も含まれる．看板以外の領域は，最終的に文字認識のフェーズまで看板と誤検知したまま進む可能性があるが，各フェーズの判定や文字認識の結果の信頼度から判定できるため，誤検知をある程度許容する．

5.2 全天球カメラによる歪みの緩和

全天球カメラの歪み緩和では，外形情報から歪みを推測し幾何学変換を行う．入力画像は，前フェーズで取得した看板の画像である．その画像の外形情報を取得し，各辺の情報から歪みの逆位相の座標データを生成し幾何学変換を行う．想定される歪みの種類と特徴を表 2 に示す．これらは，上辺と下辺の座標を 3 点ずつ取得し，曲がり具合を判定することにより各タイプと各辺の境目の座標が取得できる．その取得した情報から各辺の座標に切り分け，歪みとする．上辺と下辺では，歪み方が異なる場合があるため，上辺では上辺の歪み，中央では上辺と下辺の歪み，下辺では下辺の歪みを補正できるように各辺の情報を使用し，歪みの逆位相の座標群を各辺のサイズに合わせた矩形になるように生成する．この処理を右左辺でも行う．

表 1　実験に使用した色相・彩度・明度と色相範囲の値

色名	タイプ	色相	彩度	明度
赤 (範囲 1)	min	$R1\_H_{min} = 0$	$R1\_S_{min} = 150$	$R1\_V_{min} = 60$
	max	$R1\_H_{max} = 45$	$R1\_S_{max} = 255$	$R1\_V_{max} = 255$
赤 (範囲 2)	min	$R2\_H_{min} = 150$	$R2\_S_{min} = 150$	$R2\_V_{min} = 60$
	max	$R2\_H_{max} = 179$	$R2\_S_{max} = 255$	$R2\_V_{max} = 255$
青	min	$G\_H_{min} = 30$	$G\_S_{min} = 45$	$G\_V_{min} = 100$
	max	$G\_H_{max} = 105$	$G\_S_{max} = 55$	$G\_V_{max} = 255$
緑	min	$Bl\_H_{min} = 90$	$Bl\_S_{min} = 90$	$Bl\_V_{min} = 40$
	max	$Bl\_H_{max} = 165$	$Bl\_S_{max} = 255$	$Bl\_V_{max} = 255$
白	min	$W\_H_{min} = 0$	$W\_S_{min} = 0$	$W\_V_{min} = 230$
	max	$W\_H_{min} = 179$	$W\_S_{min} = 255$	$W\_V_{min} = 255$
黒	min	$Bk\_H_{min} = 0$	$Bk\_S_{min} = 80$	$Bk\_V_{min} = 150$
	max	$Bk\_H_{max} = 179$	$Bk\_S_{max} = 255$	$Bk\_V_{max} = 255$

生成した座標群は，OpenCV の remap メソッドを用いて変形させる．remap メソッドは以下の式を使用する．

$$dst(x, y) = src(map_x(x, y), map_y(x, y))$$

dst は出力画像，src は変換元の画像，map_x と map_y は指定する座標である．map_x と map_y に歪みの逆位相の座標群を入力することにより，画像内の歪みを相殺することができる．この処理で処理できる歪みは，全天球カメラのレンズによる湾曲だけである．縦方向や横方向に伸びる歪みは，正しい縦横比が取得できないため修正できない．しかし，看板単位を目視で確認すると各方向に少々伸びているため，多少の違和感を覚えるが，文字単位を目視で確認した場合 1 文字の画像サイズが小さいため，違和感を覚えない．また，生成型学習法と CNN を使用することにより，伸びる歪みは精度に影響しないと考える．

歪みの緩和の処理のイメージを図 20 に示す．図 20(a) は抽出された看板画像の外形を赤い線で可視化した画像，図 20(b) は外形を可視化していない図 20(a) を remap メソッドで変形させた画像である．外形が正しく取得できた場合，図 20(b) のように正しく歪みを緩和することができる．図 20(c) は，歪み緩和に使用した map_x と map_y を看板画像にプロットしたものである．赤い点は元の座標，青の点は外形から取得した歪みから算出した座標である．remap メソッドを用いて青い点の座標を対応する赤い点に移動させることで，歪みを緩和している．

5.3 文字セグメンテーション

文字のセグメンテーションは，歪みを緩和した看板画像と輪郭情報から文字のパーツを抽出して，背景色を削除し結合を行う．入力画像は前フェーズで取得した，歪みを緩和した看板画像である．その看板画像内の背景色を除く輪郭を検索し，看板の文

表 2　想定される歪みの種類と特徴

カメラに写った位置	特徴	図
カメラ中央	上辺が下辺より長い． 歪みが強い場合下辺に左右辺の情報が含まれる．	図 19 中央
中央より左	上辺が下辺の長さがほぼ同値．画像が右上がり． 上辺には右辺，下辺には左辺の情報が含まれる．	図 19 左
中央より右	上辺が下辺の長さがほぼ同値．画像が右下がり． 上辺には左辺，下辺には右辺の情報が含まれる．	図 19 右

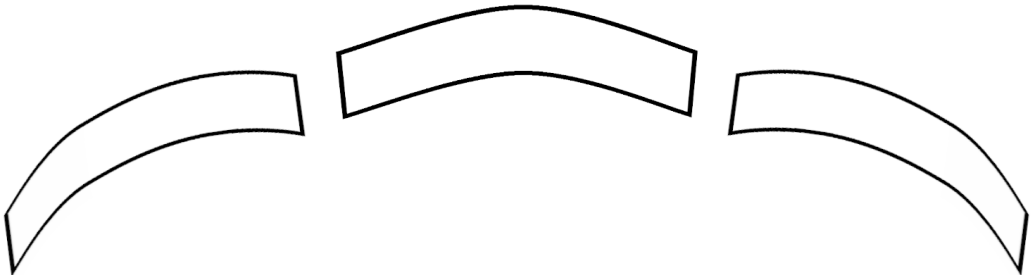


図 19　想定される歪みのイメージ図

字の外形を取得する．背景色は抽出時に取得した色相範囲と，看板の画像の色情報から特定する．その外形を元に，範囲内の文字画像を取得する．取得した文字画像は第 4.4 節で述べたとおり，複数のパーツに分かれる文字の場合，各パーツを結合し，1 文字にする必要がある．そのため，取得した外形の座標周辺の文字パーツを結合する．座標周辺を検索する際，看板画像のサイズによって検索範囲を変更している．しかし，想定より縦が長いもしくは横が長い画像では，複数の文字を結合してしまう問題がある．この問題を解決するために，3 つ以上のパーツを結合する場合，複数のパターンで結合し，文字認識の信頼度を利用して成否を判定する．結合した文字と特定した背景色の背景を重ね合わせて出力する．文字セグメンテーションのイメージを図 21 に示す．

5.4　CNN を用いた文字認識

5.4.1　生成型学習法を用いた学習画像の生成

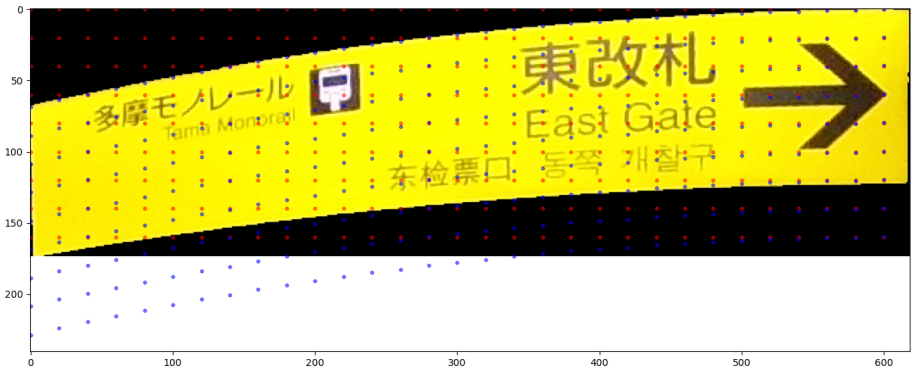
学習画像の生成には，卒業研究と過去の研究 [19, 20] の手法を用いて学習画像の生成を行う．学習画像の生成には基となる原画像が必要になる．この原画像は，案内サインに用いられるフォントと，背景色と文字色を使用して生成する．使用する文字種は，2 パターン使用する．それぞれの使用する文字群を以下に示す．

- JR 新宿駅で使用されている文字種 (表 3)
- 日本国内にある駅名や路線名に使用されている文字種 (表 4)

JR 新宿駅で使用されている文字種は，撮影した全天球画像を目視で読み取れた文字を使用する．日本国内にある駅名や路線名に使用されている文字種は，駅名や路線名の一覧のデータベース [39] から取得した文字と，ひらがな・カタカナ・アルファベッ

(a) 看板抽出で抽出された看板の外形を可視化した看板画像

(b) 歪み緩和後の看板画像



(c) 外形から取得した歪みを基に算出した map_x と map_y をプロットした看板画像

図 20　歪みの緩和の処理を行った例

(a) 入力画像

(b) セグメンテーションの結果 (上:文字の結合後の各文字, 下:結合前の各文字パーツ)

図 21　文字セグメンテーション処理を行った例

ト・数字を追加した文字種を使用する．原画像に使用した各パラメータを表 5 に示す．表 5 の各パラメータを 1 つもしくは複数使用し原画像を生成する．矢印の原画像の一部を図 22，ピクトグラムの原画像の一部を図 23 に示す．この文字種とパラメータを用いて原画像を各種類生成し，その原画像に歪みを加えて学習画像にする．

学習画像の生成は，第 5.2 節の逆の手法である，歪みを再現した座標群を元に幾何

学変換を用いる．この座標群は，以下の数式を用いて算出する．

$$map_x(x,y) = \frac{x^{1+\gamma_x}}{w^{\gamma_x}} \tag{1}$$

$$map_x(x,y) = |\frac{(w-x)^{\gamma_x+1}}{w^{\gamma_x}} - w + 1| \tag{2}$$

$$map_x(x,y) = x \tag{3}$$

$$map_y(x,y) = y - \frac{x^2}{h \cdot (n-\gamma_y)} \tag{4}$$

$$map_y(x,y) = y - \frac{(h-x)^2}{h \cdot (n-\gamma_y)} \tag{5}$$

$$map_y(x,y) = y \tag{6}$$

w は画像の横幅，h は画像の縦幅，γ_x と γ_y は歪み定数である．これらの数式は，remap を用いて生成した画像を目視で確認し，抽出された文字に近似しているものを使用した．γ_x と γ_y の値を変更することにより歪み方を調整できる．生成した原画像と学習画像の一例を図 24，想定される歪みと生成に使用した数式の対応表を表 6 に示す．

5.4.2　CNN を用いた学習

本研究では CNN を用いて文字認識を行う．使用したライブラリは Keras と TensorFlow である．使用したモデルは，VGG16 [38]，各ノードの重みの初期値

表 3　JR 新宿駅の使用文字種と文字数

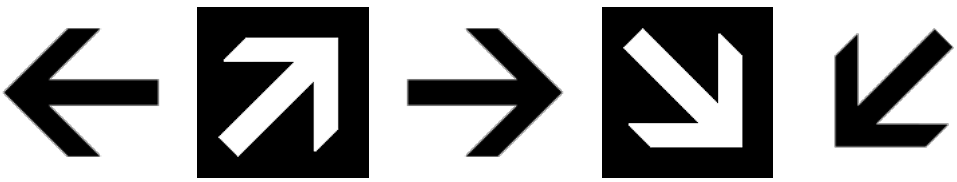
文字種	文字数
ひらがな	19 文字
カタカナ	19 文字
アルファベット (大文字)	17 文字
アルファベット (小文字)	24 文字
漢字	89 文字
数字	10 文字
記号	11 文字
矢印	4 文字
ピクトグラム	7 文字
合計	200 文字

表 4　日本の駅に使用されている文字種と文字数

文字種	文字数
ひらがな	75 文字
カタカナ	79 文字
アルファベット (大文字)	26 文字
アルファベット (小文字)	26 文字
漢字	1375 文字
数字	10 文字
記号	13 文字
矢印	8 文字
ピクトグラム	19 文字
合計	1631 文字

表 5　生成する文字画像のパラメータ

条件名	使用する条件
背景色	白 (#FFFFFF), 黒 (#000000), 緑 (#009821), 黄 (#FFD900)
文字色	黒 (#000000), 白 (#FFFFFF), 白 (#FFFFFF), 白 (#FFFFFF)
日本語フォント	新ゴ Pro M
英語フォント	ミリアド R
数字フォント	ミリアド R
英語フォント	ミリアド R
画像サイズ (pixel)	128×128, 64×64, 32×32 16×16
フォントサイズ (pt)	108, 54, 27, 13
矢印	JIS で定められている矢印 + 45 度ずつ回転させた矢印 (合計 8 種類)
ピクトグラム	JIS で定められているピクトグラムの内駅で使用されているもの (合計 19 種類)



(a) 0 度の矢印の原画像　(b) 135 度回転させた矢印の原画像　(c) 180 度回転させた矢印の原画像　(d) 225 度回転させた矢印の原画像　(e) 315 度回転させた矢印の原画像

図 22　生成した矢印の原画像の一例

は ImageNet で学習されたものを使用する．VGG16 は，畳み込み層 13 層と総結合層 3 層の計 16 層で構成されている．VGG16 の層の一覧を表 7 に示す．表 7 の dense_3 は使用する文字種の数である．VGG16 は 1000 クラスを分類するモデルだが，生成した学習画像を使用することにより今回使用する文字種すべて識別することが可能で

(a) ピクトグラム「情報コーナー」の原画像 (b) ピクトグラム「お手洗い」の原画像 (c) ピクトグラム「きっぷうりば/精算所」の原画像 (d) ピクトグラム「エレベーター」の原画像 (e) ピクトグラム「鉄道/鉄道駅」の原画像像

図 23　生成した矢印の原画像の一例

表 6　想定される歪みと生成に使用した数式

レンズと看板の相対位置	想定される歪み	式番号	学習画像
	右下に湾曲した歪み	(1),(4)	図 24(b)
中央より右	右に寄せた歪み	(1),(6)	図 24(e)
	右側のみ湾曲した歪み	(3),(4)	図 24(h)
	左下に湾曲した歪み	(2),(5)	図 24(c)
中央より左	左に寄せた歪み	(2),(6)	図 24(f)
	左側のみ湾曲した歪み	(3),(5)	図 24(i)
中央	歪みなし	なし	各原画像

ある．

5.4.3　学習済みデータを用いた文字認識

文字認識には，セグメンテーション済みの文字画像と学習済みデータを用いる．CNN の文字認識では，学習済みデータのノードの重みから算出される各ラベルの信頼度が出力できる．この信頼度が一番高いラベルをその文字画像の認識結果として，正答率を算出する．また，上位 3 位までに正しい認識結果が含まれる割合である，累積認識率も算出する．

5.5　文字認識結果を訂正する辞書の利用

5.5.1　辞書の作成

文字認識結果を訂正する辞書は，「駅データ.jp」[39] と「Wikipedia (日本語版)」[40] から入手できる情報から作成する．作成する辞書は，路線や駅名が含む辞書と駅周辺のランドマークや出口情報などを含む辞書の 2 つを作成する．

駅データ.jp は，新幹線駅やロープウェイ駅などの一部駅を除く殆どの駅名や路線名

(a) 文字の原画像「あ」 歪みなし (b) 文字の原画像「あ」右下に湾曲した歪み (c) 文字の原画像「あ」左下に湾曲した歪み

(d) 文字の原画像「ア」 歪みなし (e) 文字の原画像「ア」右に寄せた歪み (f) 文字の原画像「ア」左に寄せた歪み

(g) 文字の原画像「亜」 歪みなし (h) 文字の原画像「亜」右側のみ湾曲した歪み (i) 文字の原画像「亜」左側のみ湾曲した歪み

図 24　生成した学習画像の一例

表 7　使用した VGG16 の層

層の種類	出力のシェープ
input_1 (InputLayer)	(128, 128, 3)
block1_conv1 (Conv2D)	(128, 128, 64)
block1_conv2 (Conv2D)	(128, 128, 64)
block1_pool (MaxPooling2D)	(64, 64, 64)
block2_conv1 (Conv2D)	(64, 64, 128)
block2_conv2 (Conv2D)	(64, 64, 128)
block2_pool (MaxPooling2D)	(32, 32, 128)
block3_conv1 (Conv2D)	(32, 32, 256)
block3_conv2 (Conv2D)	(32, 32, 256)
block3_conv3 (Conv2D)	(32, 32, 256)
block3_pool (MaxPooling2D)	(16, 16, 256)
block4_conv1 (Conv2D)	(16, 16, 512)
block4_conv2 (Conv2D)	(16, 16, 512)
block4_conv3 (Conv2D)	(16, 16, 512)
block4_pool (MaxPooling2D)	(8, 8, 512)
block5_conv1 (Conv2D)	(8, 8, 512)
block5_conv2 (Conv2D)	(8, 8, 512)
block5_conv3 (Conv2D)	(8, 8, 512)
block5_pool (MaxPooling2D)	(4, 4, 512)
flatten_1 (Flatten)	(8192)
dense_1 (Dense)	(4096)
dropout_1 (Dropout)	(4096)
dense_2 (Dense)	(4096)
dropout_2 (Dropout)	(4096)
dense_3 (Dense)	(n)

を CSV 形式で公開しているサイトである．使用したデータは，2020 年 6 月 19 日の「路線データ」と「駅データ」である．「路線データ」は「路線名称 (一般)」，「駅データ」は「駅名称」と運用中もしくは廃止の「状態」のデータを用いて駅一覧のデータを作成する．

Wikipedia のデータはデータベース・ダンプ [41] の「20201101」の「jawiki-latest-pages-articles.xml」を用いる．このデータは，2020 年 11 月 01 日の日本語版の Wikipedia のすべてのページが 1 つの XML ファイルに格納されている．「jawiki-latest-pages-articles.xml」内にある新宿駅のページのデータの一部を図 25 に示す．図 25 のように，各ページの各情報にはタグやスタイルがあるため，必要な情報が取得しやすい．本研究で用いたタグと活用方法の一覧を表 8 に示す．表 8 のタグやスタイルを用いて，各駅の駅周辺情報を取得する．

本提案手法では，駅データ.jp の駅一覧データと Wikipedia のページ名が一致するものを駅のページデータとして使用する．駅一覧データを使用しない手法として，Wikipedia のカテゴリ一覧から現在運用されている駅のページのみを使用する方法が考えられる．Wikipedia のページ名やカテゴリにブラックリスト[*7]・ホワイトリスト[*8]の単語を設定することにより，一部の駅を抽出することは可能である．しかし，表記ゆれやカテゴリの追加ミスなどによるカテゴリの完全な統一がされていないため，運用されている駅だけを完全に抽出することは困難である．上記の手法で駅ごとに分割した Wikipedia の駅一覧データは 7547 駅[*9]取得できた．しかし，Wikipedia の駅一覧データ内には，現在使われていない駅や改名・移転した駅のデータなどの使用できないデータが一部残存している．カテゴリ名や駅名から判定することは可能だが，Wikipedia 内の全駅データを調査する必要があるため困難である．そのため，駅データ.jp と Wikipedia のデータを使用する．

駅データ.jp と Wikipedia のデータから抽出した，ほとんどの駅のページデータには，「== 駅周辺 ==」(図 25 の 22 行目) という見出しが含まれている．この見出しが含まれていない駅は，駅周辺情報は使用しない．この見出しは以下の 2 つのパターンがある．

1. 日本語の文章で記述された駅周辺の環境と箇条書きによる駅周辺のランドマーク
2. 箇条書きによる駅周辺のランドマークのみ

日本語で記述された文章は，Wikipedia のタグやリンクなどのインライン要素を考慮した自然言語処理を行うことで，駅周辺のランドマークや環境等は利用可能であると考える．しかし，箇条書き内にランドマーク等がほとんど含まれる点と，その項目の編集者によって記述されているランドマークや環境などの内容の傾向が変わる点の 2 点から，本研究では日本語の文章で記述された文章は使用しない．生成したランドマークの辞書の一部を図 26 に示す．この辞書を用いて認識結果を訂正する．

[*7] ”Wikipedia:削除依頼”や，”ファイル:”，”廃止”，”Template”，”貨物”など

[*8] ”日本の鉄道駅”，”東日本旅客鉄道の鉄道駅”，”西日本旅客鉄道の鉄道駅”，営鉄道事業者名など

[*9] 日本の駅数の 81.979%

```
<page>
  <title>新宿駅</title>
  <ns>0</ns>
  <id>1855047</id>
  <revision>
    <id>80165651</id>
    <parentid>80008108</parentid>
    <timestamp>2020-10-27T07:32:55Z</timestamp>
    <contributor>
      <username>Keita.Honda</username>
      <id>1254939</id>
    </contributor>
    <model>wikitext</model>
    <format>text/x-wiki</format>
    <text bytes="207721" xml:space="preserve">{{Otheruses||[[西武新宿線]]
    の駅|西武新宿駅}}
(省略)
== 駅周辺 ==
周辺は[[繁華街]]・[[歓楽街]]・[[オフィス街|ビジネス街]]・[[ゲイ・タウン|ゲイタウ
ン]]などさまざまなものが集中し、東京駅周辺などと比較して雑多で華やいだ雰囲気を持
つ。昼夜を問わず人の波が途絶えることはない。乗り入れる路線も[[東京都]]多摩地域、
[[神奈川県]]北西部、[[埼玉県]]南部といった[[ベッドタウン]]とを結んでいるため、終
日利用者は絶えない。
(省略)
* [[新宿駅西口地下広場]]
** 新宿西口[[献血]]ルーム
* [[京王百貨店]] 新宿店 - 京王線の駅と一体
* [[小田急百貨店]] 新宿店 - 小田急とJR東日本の西口と一体
* [[小田急電鉄直営事業#その他のビル経営|小田急ハルク]]
** [[ビックカメラ]] 新宿西口店
* [[西新宿|新宿高層ビル群]]・[[新宿新都心]]
(省略)
[[Category:新宿駅|*]]
[[Category:新宿区の鉄道駅]]
[[Category:渋谷区の鉄道駅]]&lt;!--都営大江戸線のみ渋谷区代々木に位置--&gt;
[[Category:日本の鉄道駅 し|んしゆく]]
[[Category:東日本旅客鉄道の鉄道駅]]
[[Category:日本国有鉄道の鉄道駅]]
[[Category:山手線]]
[[Category:埼京線]]
[[Category:中央・総武緩行線]]
[[Category:中央線快速]]
[[Category:日本鉄道]]
[[Category:甲武鉄道]]
[[Category:京王電鉄の鉄道駅]]
[[Category:小田急電鉄の鉄道駅]]
[[Category:東京地下鉄の鉄道駅]]
[[Category:東京都交通局の鉄道駅]]
[[Category:1885年開業の鉄道駅]]
[[Category:日本のギネス世界記録|しんしゆくえき]]
[[Category:新宿|しんしゆくえき]]
[[Category:代々木|しんしゅくえき]]&lt;!--都営大江戸線のみ渋谷区代々木に位置--&gt;
[[Category:1880年代日本の設立]]</text>
  </revision>
</page>
```

図 25 「jawiki-latest-pages-articles.xml」内の新宿駅のページのデータの一部

```
新宿駅西口地下広場
新宿西口献血
京王百貨店
小田急百貨店
小田急ハルク
ビックカメラ
新宿高層ビル群
東京都庁舎
モード学園
工学院大学
ヨドバシカメラ新宿西口本店
新宿郵便局
ユニクロ
新宿西口駅
青梅街道
新宿ステーションスクエア
新宿アルタ
ビックロ
ヨドバシカメラ　マルチメディア新宿東口
新宿高野
紀伊國屋書店
伊勢丹
丸井
新宿マルイアネックス・新宿WALD9
新宿マルイワン
新宿マルイカレン
ルミネエスト新宿
メトロプロムナード
ヤマダ電機
新宿サブナード
東京都道430号新宿停車場前線
新宿二丁目
みずほ銀行
ムサシノ通
新宿区役所
花園神社
新宿ゴールデン街
西武新宿駅
新宿プリンスホテル
西武新宿PePe
靖国通り
東京都健康プラザハイジア
東京都保健医療公社大久保病院
東新宿駅
角筈ガード
Flags
ルミネ
大塚家具
ルミネ新宿　ルミネ1
小田急ミロード
モザイク通り
ヤマダ電機
甲州街道
```

図 26　生成された新宿駅のランドマークの辞書の一部

表 8　辞書作成に用いたタグと活用方法の一覧

タグ	該当箇所	活用方法
page タグ	図 25 の 1 行目と 39 行目	各 Wikipedia のページの分割
title タグ	図 25 の 2 行目	各 Wikipedia のページ名の取得
id タグ	図 25 の 4 行目	該当ページへの URL 作成
「駅周辺」見出し	図 25 の 22 行目	駅周辺情報を抽出
「Category」リンク	図 25 の 39 から 59 行目	情報の取捨選択

5.5.2　辞書と認識結果の信頼度を用いた訂正

認識結果の誤り訂正には前節の手法で作成した辞書と，文字認識の第 3 位累積認識率まで含む認識結果のラベルと信頼度の 3 つを用いて行う．誤り訂正には JR 八王子駅でシミュレーションした時に作成した以下の数式を用いて評価する．

$$w_{eval_i} = \frac{1}{Lv(w, dict_i) + 1} \prod_{k=0}^{len(w)} C_{acc_k} \tag{7}$$

w は認識結果後の各ラベルを結合した単語，i は添字 $(0 \leq i < 3^{len(w)})$[*10]，w_{eval_i} は i 番目の結合した単語の評価値，$dict$ は辞書内の単語リスト，i は $0 < i < len(dict)$ を範囲とする添字，C_{acc_k} は k ラベル目の信頼度，Lv はレーベンシュタイン距離を求める関数，len はリストの要素数を求める関数である．この数式を用いて，各ラベルを結合した単語と辞書内の単語とのレーベンシュタイン距離と，各ラベルの信頼度の総乗を計算し誤り訂正を行う．

[*10] 添字 i は認識結果の各ラベルの順列のリストの添字である．添字 i の最大値 $3^{len(w)}$ の底が 3 の理由は第 3 位累積認識率まで計算するためである．

6　事前実験

事前実験では，事前調査と調整に使用した JR 八王子駅と JR 立川駅の写真を用いて提案手法の有効さを検証した．

目視で内容を読み取れた看板領域の抽出は，表 9 の通り，84.6% の精度で正しく抽出できた．抽出できなかった案内サインはなかったが，照明の影響による案内サインの分割や，内部に照明を持たない黒の案内サインの文字部分のみの抽出などが確認できた．

文字セグメンテーションは，抽出された案内サイン内の文字と 1 文字に分割できた文字は表 10 の通り，22.8% となった．これは，床設置型案内サインの背景色特定が失敗し，文字色まで削除したためである．吊り下げ型案内サインのみに限定した場合 71.4% の文字のセグメンテーションを行うことができた．

文字認識の精度は，前フェーズでセグメントされた文字を用いた場合，JR 八王子駅の学習済みデータでは 83.3%，JR 立川駅の学習済みデータでは 78.6% である．本提案手法の認識器は日本語を正しく認識できたが，特徴量の少ない記号や，画像サイズが極端に小さい画像の一部を正しく認識しなかった．正しく認識した場合でも信頼度が極端に低い結果となった．

誤り訂正は表 11 の通り，78.6% の精度で正しく誤り訂正が行えた．辞書内にある単語は，文字認識の精度が高かったため高精度で訂正が行えたが，セグメンテーションに失敗して単語内の一部が欠損した単語や辞書にない単語では誤った訂正を行う傾向が確認された．

これらの予備実験で判明した，看板抽出の色相範囲や文字セグメンテーションの調整に使用する値，文字認識の学習に使用する生成型学習法の生成時のパラメータを本実験に使用した．使用した各パラメータは，第 5 章に記載されている値である．

表 9　事前実験の看板抽出の精度

駅名	取得枚数	案内サイン枚数	精度 (単位:%)
JR 八王子駅	9	11	81.8
JR 立川駅	13	15	86.7
合計	22	26	84.6

表 10　事前実験の文字セグメンテーションの精度

タイプ	分割成功文字数	全文字数	精度 (単位:%)
全て	126	552	19.2
吊り下げ型案内サイン	105	147	71.4

表 11　事前実験の誤り訂正の精度

正しい訂正結果の単語数	全体の単語数	精度 (単位:%)
21	31	67.7

7　実験

本研究の実験内容は，JR 八王子駅で撮影された全天球画像で調整した提案手法を，JR 新宿駅の全天球画像に使用して各フェーズの精度を算出する．実験に使用した全天球画像の内訳を表 12，使用した画像を図 30 から図 34，使用した画像の撮影箇所を図 27 から図 29 に示す．

本章では，以下の 4 つのフェーズの精度を記述する．

1. 全天球画像を用いた看板抽出の精度
2. 文字セグメンテーションの精度
3. CNN を用いた文字認識の精度
4. 文字認識結果を訂正する辞書の利用の修正率と精度

7.1　全天球画像を用いた看板抽出の精度

看板抽出のフェーズで抽出された画像枚数とその内訳を表 13 に示す．表 13 の「抽出枚数」の括弧内は色相範囲の色名もしくは，全ての色相範囲で抽出した枚数である．また，抽出できた看板画像の内訳と看板抽出の精度を表 14 に示す．案内サインと看板画像は，案内サインの画像と案内サインを除く看板画像を計数を行う．また，各画像は，文字・ピクトグラム・矢印・ロゴなどから，どの看板から抽出できたかが判別できる画像のみ計数を行う．表 14 の「ノイズ付」は，看板や床などの看板以外の領域と看板の領域を抽出した画像を計数した値である．また，「画像内の看板 (目視)」は，全天球画像から目視で案内サインを計数した値，「可読のみ」は案内サイン内の文字・ピクトグラム・矢印が読み取れた画像のみ計数した値である．

表 12　本実験に使用した全天球画像の内訳

画像名	実験画像 1	実験画像 2	実験画像 3	実験画像 4	実験画像 5
使用した写真	図 30	図 31	図 32	図 33	図 34
場所	図 27(s1)	図 28(s2)	図 29(s3)	図 27(s4)	図 27(s5)
看板数 (総数)	13 枚	38 枚	19 枚	30 枚	9 枚
看板数 (可読)	4 枚	8 枚	15 枚	17 枚	6 枚

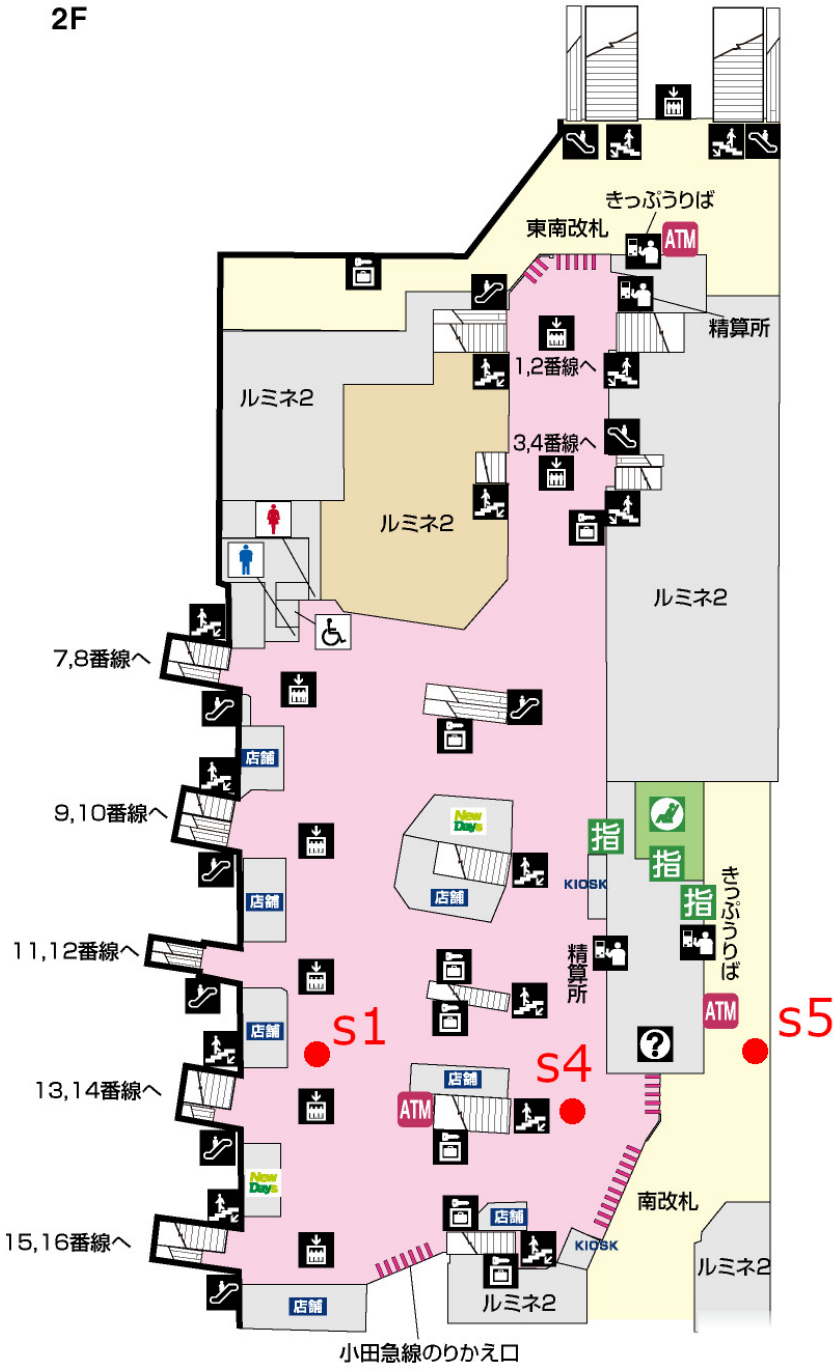


図 27　新宿駅構内図 2 階 [42]

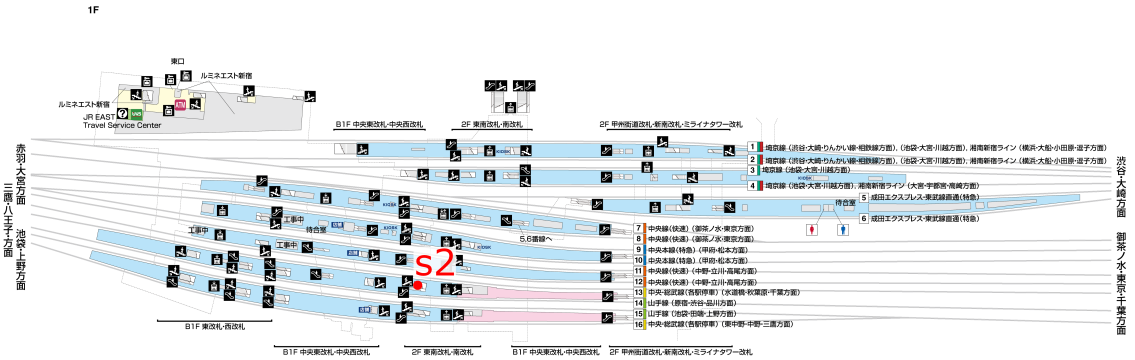


図 28　新宿駅構内図 1 階 [42]

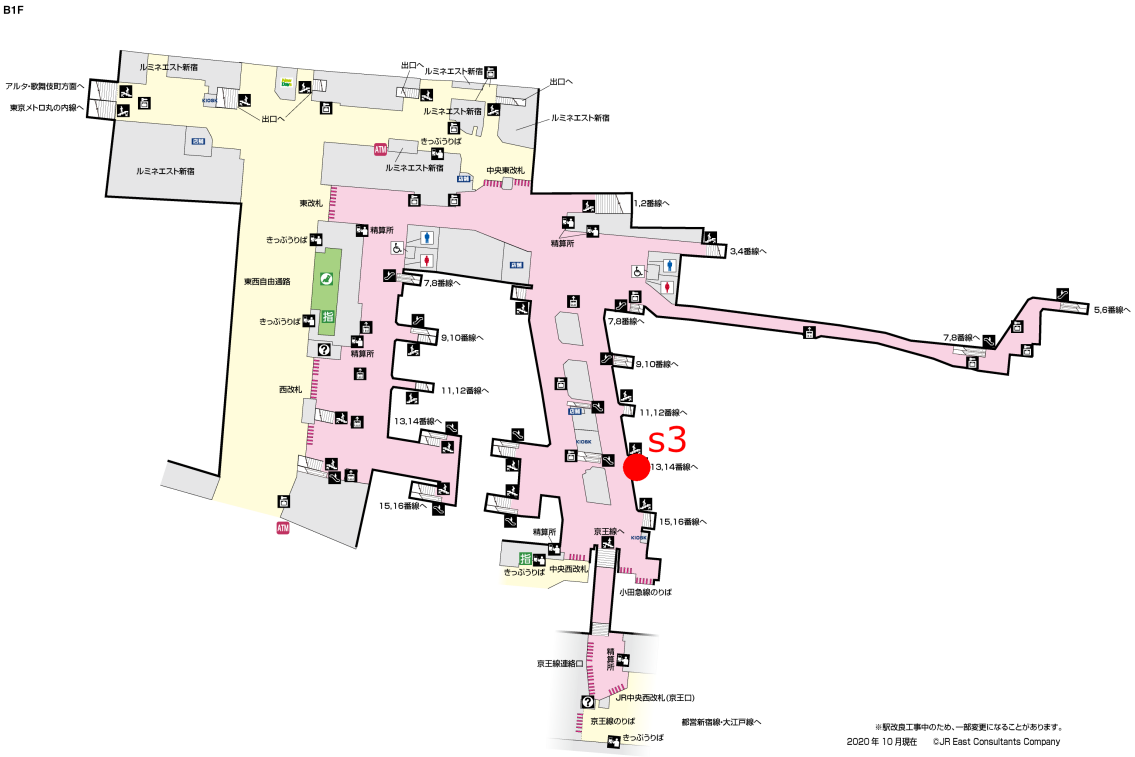


図 29　新宿駅構内図 地下 1 階 [42]

図 30　実験に使用した実験画像 1

図 31　実験に使用した実験画像 2

図 32　実験に使用した実験画像 3

図 33　実験に使用した実験画像 4

図 34　実験に使用した実験画像 5

表 13　看板の抽出の画像の内訳 (単位:枚)

画像名	実験画像 1	実験画像 2	実験画像 3	実験画像 4	実験画像 5
抽出枚数 (赤)	100	46	17	35	14
抽出枚数 (緑)	363	116	40	127	38
抽出枚数 (青)	0	0	0	0	3
抽出枚数 (白)	45	26	47	56	65
抽出枚数 (黒)	131	227	70	99	112
抽出枚数 (総数)	640	415	174	317	232

表 14　抽出された看板画像の内訳 (単位:枚) と看板抽出の精度

画像名	実験画像 1	実験画像 2	実験画像 3	実験画像 4	実験画像 5
案内サイン (一部)	11	34	31	31	18
案内サイン (ノイズ付)	4	3	3	4	0
案内サイン (一部 + ノイズ付)	1	0	1	0	0
案内サイン (完全)	0	5	0	3	1
看板 (一部)	7	6	6	2	2
看板 (ノイズ付)	17	1	3	2	7
看板 (一部 + ノイズ付)	0	0	1	0	0
看板 (完全)	2	6	2	1	0
抽出できた看板	9	13	12	12	4
画像内の看板	18	41	19	21	6
抽出精度	50.0%	31.7%	63.2%	57.1%	66.7%
抽出できた看板 (可読のみ)	8	7	10	9	4
画像内の看板 (可読のみ)	10	10	17	15	6
抽出精度 (可読のみ)	80.0%	70.0%	58.8%	60.0%	66.7%

7.2　文字セグメンテーション

全天球画像を用いた看板抽出結果の画像を用いた文字のセグメンテーションの結果を表 15 に示す．なお，入力画像に複数文字を含んだ画像から正しくセグメンテーションできた枚数は 0 枚だった．

7.3　CNN を用いた文字認識の精度

前フェーズにセグメンテーションした画像を用いた文字認識の精度を表 16 に示す．また，全天球画像の文字を手動でセグメンテーションした画像を用いた精度を表 17 と表 18 に示す．表 16 の「認識した枚数」は前フェーズの結果の画像を入力画像とした認識結果である．表 17 は JR 新宿駅に使用されている文字のみ学習した学習済みデータ，表 18 は駅名と路線名の文字種を学習した学習済みデータを用いた認識結果である．CNN を用いた文字認識の第 3 位累積認識率は表 16 の精度と変わらなかった．手動でセグメンテーションした画像の文字認識の第 3 位累積認識率は表 17 では 86.567%，表 18 では 49.253% だった．

7.4　文字認識結果を訂正する辞書の利用した修正率と精度

前フェーズの認識結果と辞書を用いた訂正結果と，手動でセグメンテーションした画像の認識結果を用いた訂正結果を表 19 に示す．表 19 の「正しい結果」にある括弧はセグメンテーションに失敗して認識できなかった文字,「辞書内の単語の有無」は,「正しい結果の単語の有無/間違った結果の単語の有無」である．

表 15　文字セグメンテーションの結果 (単位:枚)

画像名	画像 1	画像 2	画像 3	画像 4	画像 5
1 文字にセグメンテーションできた枚数	0	2	8	7	2

表 16　CNN を用いた文字認識の結果

画像名	画像 1	画像 2	画像 3	画像 4	画像 5
認識した枚数	0	1	0	0	1
精度	0%	50%	0%	0%	50%

表 17　手動でセグメンテーションした画像を用いた新宿駅の学習データの文字認識の結果

文字種	使用した枚数	認識した枚数	精度
英語	9 枚	3 枚	33.3%
矢印	8 枚	7 枚	87.5%
ピクトグラム	9 枚	6 枚	66.7%
数字	3 枚	0 枚	0.0%
ひらがな	6 枚	5 枚	83.3%
カタカナ	3 枚	3 枚	100.0%
漢字	29 枚	28 枚	96.6%
総合	67 枚	52 枚	77.6%

表 18　手動でセグメンテーションした画像を用いた全駅の学習データの文字認識の結果

文字種	使用した枚数	認識した枚数	精度
英語	9 枚	0 枚	0.0%
矢印	8 枚	7 枚	87.5%
ピクトグラム	9 枚	6 枚	66.7%
数字	3 枚	0 枚	0.0%
ひらがな	6 枚	1 枚	16.7%
カタカナ	3 枚	0 枚	0.0%
漢字	29 枚	6 枚	20.7%
総合	67 枚	20 枚	29.9%

表 19　文字認識結果を訂正する辞書の結果

認識結果の種類	認識結果	正しい結果	訂正結果	辞書内の単語の有無
前フェーズ	口	(東) 口	複数	なし/あり
手動	内田	成田	内田	あり/あり
手動	新宿御苑	新宿御苑	新宿御苑前	なし/あり
手動	バスタ新宿	バスタ新宿	バスタ新宿	あり/-

8　考察

8.1　全天球画像を用いた看板抽出

8.1.1　正しく抽出できた看板

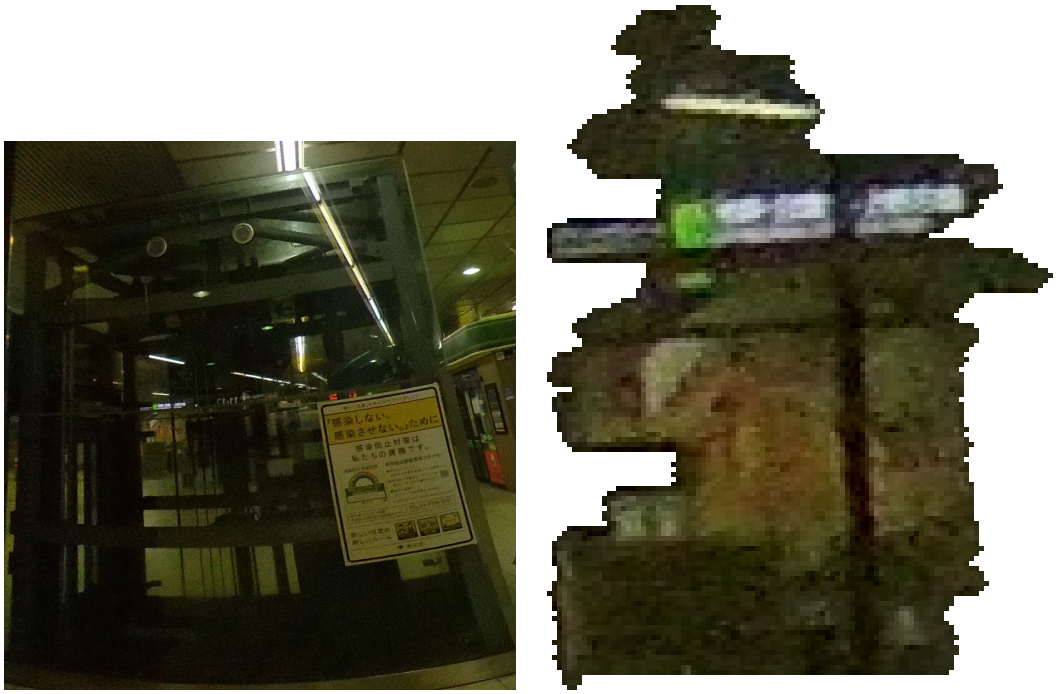
正しく抽出できた案内サインの内訳と精度は表 14 のような結果になった．抽出された案内サインは，1 文字のみを含む画像と看板内で分割された画像の 2 種類が確認できた．図 35 のような 1 文字のみを含む画像は，案内サインの背景色と文字色の色相範囲が異なるため抽出したと考えられる．本提案手法では，複数の色相範囲を使用しているため，文字のみを抽出することは必然であると考える．そのため，1 文字のみの抽出を許容し文字認識を行うことで，提案手法全体の精度が向上すると考えられる (表 20 改善案 1)．看板内で分割された画像も同様に，複数文字を含む画像であるため，セグメンテーションを行えば文字認識に利用できる．ただし，両者とも本提案手法では外形を利用して歪み緩和やセグメンテーションを行っているため，現在の手法では正しくセグメンテーションすることはできない．そのため，外形を利用しない手法を用いて正しくセグメンテーションを行う必要がある．

実験画像 1 では，駅利用者が見逃す可能性がある案内サインを抽出することができた．駅利用者が見逃す可能性がある案内サインと場所を図 36 に示す．図 36 は，実験画像 1(図 30) にある透明なエレベーターの奥にある案内サインである．この案内サインは，透明なエレベーターを透過した奥にあるため，見逃す可能性が高い[*11]．図 36(b) は，目視でも内容は読み取れないが，場所から推測すると 15 番線の山手線であると考えられる．この案内サインは，実験画像 1 内では唯一撮影された案内サインである．この案内サイン内の文字が読み取ることができれば，目視で見落とす可能性がある案内サインの情報を抽出できるため，駅構内のナビゲーションの情報の 1 つとして有効に利用できると考える．

案内サイン以外の看板抽出では，様々な看板や掲示物などが抽出できた．抽出できた案内サイン以外の看板を図 37 に示す．これらは，本研究の対象外である看板や掲示物である．しかしこれらの情報を活用することで，誘導路に沿ったナビゲーションや，店舗情報の取得や案内，掲示物内の情報提供などが行えると考える．

[*11] 撮影者である筆者は，どの案内サインが撮影できるかを確認しながら撮影を行っていたが見落としていた．

図 35　1 文字に分割された案内サインの一部



(a) 実験画像 1(図 30) の中央のエレベーター　(b) 駅利用者が見逃す可能性がある案内サイン

図 36　目視では見逃す可能性がある案内サイン

8.1.2　看板の枚数以上を抽出した原因

実験に使用した全天球画像の内訳 (表 12) と看板の抽出の画像の内訳 (表 13) から，案内サインの枚数以上の画像が抽出されたことがわかる．これは，以下のパターンの要因が考えられる．

1. 看板以外の床・天井・店舗のペイント・照明などが写り込んだ事による影響
2. 看板以外の利用者誘導用のペイント・点字ブロックなどが写り込んだ事による影響

(a) 改札内にある店舗の看板 (図 30)

(b) 改札内の掲示物 (図 30)

(c) ホームと通路をつなぐエスカレータの看板 (図 33)

(d) ホームの床に設置された誘導の矢印 (図 31)

図 37　抽出された案内サイン以外の看板

3. 照明が床や天井に反射して本来の色ではないが色相範囲に入った事による影響
4. 看板や案内サインが内部もしくは，外部の照明で複数の色相範囲に別れた事による影響
5. 看板や案内サインが外部の照明の反射により分割された事による影響
6. 文字のみ抽出できてしまう条件による影響

1・2・3 は，各条件で案内サイン用に設定した HSV 色空間のパラメータの範囲内に入ってしまったことが原因である．これは，事前に調整で使用した JR 八王子駅や JR 立川駅を用いて事前実験を行った時から確認されている．この影響は，文字セグメンテーション時の文字の外形取得の失敗や，文字認識の信頼度の異常な低下などで，案内サインもしくは文字を含む看板のみ絞り込むことができるため，問題はないと考える．しかし，本来取得したい看板の画像の 10 倍以上の枚数を取得し，後のフェーズまで処理をしてしまうので，実行時間が多くかかってしまう．最終目標である「駅構内のナビゲーション」を達成するためには，看板以外の画像の抽出枚数を減らし，実行時間をより短くしなければならない．現段階で考えられる対処法は，取得する看板の範囲を制限すること (表 20 改善案 2) や，HSV 色空間の範囲をより狭める事 (表 20 改善案 3)，看板と看板以外の HSV 色空間のパラメータの差を広げる事 (表 20 改善案 4)，色相範囲以外の抽出方法を行う事 (表 20 改善案 5) が考えられる．取得する看板の範囲を制限することは，全天球カメラ内の加速度センサーから水平が取得できるため，点字ブロックやノイズ画像の除去ができる．しかし，壁・床設置型の案内サインを撮影した場合，制限した範囲内の看板を認識することができない問題が生じる．この問題は案内サインのタイプの判定や，もっとも近い案内サインとの距離の取得などを行うことにより回避できると考える．HSV 色空間の範囲をより狭める事は，本来抽出しなければならない看板の画像が欠落する可能性があるため，容易ではない．看板と看板以外のパラメータの差を広げるには，補正もしくはカメラの設定による調整が必要である．この補正やカメラの設定は，輝度調整や色温度調整，ホワイトバランス，ISO 感度などが考えられるが，各駅の撮影条件によって必要な補正が変わるため，すべての駅に対応する補正は困難であると考える．色相範囲以外の抽出方法は，物体検出が考えられるが，案内サインを学習させなければいけないため，困難であると考える．

4 は，図 38 のように，案内サインの一部が内部の照明により白飛びしてしまい，取得できた色相範囲が複数に分かれてしまうことが原因である．この原因は，HSV 色空間の範囲を広げることにより対処は可能であるが，看板以外の画像も取得してしまう

問題がある．また，事前に調査した JR 八王子駅や JR 立川駅では発生しなかったため，JR 新宿駅の撮影環境が調査した駅と大きく異なる可能性がある．この問題は，撮影に使用する ISO 感度やシャッタースピードなどのカメラ設定を，案内サイン用に複数用意し使用すること (表 20 改善案 6) で改善できると考えられる．

5 は，看板付近の照明が看板に反射してしまい，白もしくは，看板の背景色の輝度を高めた色に変色したように撮影できてしまうことが原因だと考える．これは，撮影位置と看板の材質によるものだと考えられる．最終目標である「駅構内のナビゲーション」では歩きながらを想定しているため，撮影時に照明が反射しても次回の撮影では同じ看板を違う角度で撮影できるので，問題ないと考える．また，看板の材質によって反射の起こりやすさは，図 1 のタイプや材質によって変化すると考える．吊り下げ型であり，かつ，内部に照明を備えた図 1(a)・図 38 のような案内サインと図 1(b) の案内サインのような案内サインでは前者のほうが反射しやすいと考える．そのため，案内サインのタイプによってカメラ設定 (表 20 改善案 6) や撮影方法を変えることにより対策ができると考える．近年のスマートフォンやカメラでは LiDAR や深度計測が可能なカメラが開発されている [43, 44] が，カメラと案内サインの距離が遠いため，正しく撮影することは困難であると考える．

6 は，案内サイン内に背景色と文字色の複数の色があることが原因である．現在の提案手法では，看板の範囲を抽出するため，看板が抽出されれば中の文字も一緒に抽出することができる．しかし，中の文字のみを看板とは違う色相範囲で抽出できた場合，そのまま使用する．そのため，看板と文字のセットと文字のみの 2 つの情報が取得できてしまう．この問題は，処理時間が増加してしまうが，両方の情報が活用できれば，片方が正しく処理されなかった場合でも補完できるため有効に利用できると考える．

(a) 色相範囲・赤の画像　　(b) 色相範囲・白の画像

図 38　実験画像 2 の白飛びした案内サインの画像

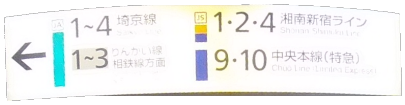
8.1.3　抽出できなかった画像

撮影した全天球画像内の目視で文字を読み取れる案内サインは，ほとんど抽出できた．しかし，黒色の内部に照明を搭載していない看板の精度は，他の色相範囲の抽出枚数と比較した場合，低いことがわかった．これは，色相範囲の指定では白・黒が一意に定まらないためである．色相範囲が一意に定まらないため，色相範囲の範囲や彩度・明度も広範囲に取得する必要があり，細かい範囲指定が困難である．しかし，広範囲に指定した場合，看板以外の床や天井などの必要ない部分も取得してしまうため，調整が困難である．また，図 32 のように，柱の内部に埋め込まれた乗り場番号や，電車の到着時間が表示されている電光掲示板の上部の乗り場案内，図 34 の文字部分のみ発光する乗り場案内など，さまざまなタイプの黒の案内サインがあるのも原因の一つだと考える．白の内部に照明を搭載していない看板も同じ問題が発生する可能性があるが，今回の実験では対象となる看板が設置されていないため，全駅を対象とした場合にこの問題が発生すると考えられる．

今回使用した「RICOH THETA Z1」 [37] の性能もしくは撮影時の設定が不適切だった可能性がある．「RICOH THETA Z1」は 4K 解像度で撮影可能だが，駅構内の通路や改札外の出入り口などの広い空間の遠くの案内サインは，文字や案内サインが潰れたり，前節の照明の影響を強く受けるなどにより情報が欠落してしまうことがある．カメラのホワイトバランスや ISO 感度，シャッタースピードを駅の環境ごとに調整すること (表 20 改善案 6) で，この問題は回避可能である．しかし，各駅によって案内サインの抽出に最適な設定を行うのはカメラの専門的な知識が必要であるため，一般人には困難であると考える．カメラの性能が向上することにより，カメラの知識が無い撮影者でも，もっと多くの看板が抽出できると考える．

8.1.4　看板画像の外形の取得

抽出できた看板画像のほとんどは，看板画像の外形通り抽出できていない．外形通り抽出できている案内サインとできていない案内サインを図 39 に示す．これは，看板の周辺の背景が看板の照明で滲んでしまう問題や，第 8.1.2 節で述べた影響で案内サイン内で分割されてしまう問題，同じ色相範囲内に看板と背景が入ってしまう問題が起こり正しく抽出できていないと考えられる．外形通り抽出できなければ，歪みの緩和や文字セグメンテーションが正しく行えないため，改善を行う必要がある．



(a) 成功例 1: 下部に背景が入っているが歪み緩和可能

(b) 失敗例 1: 内部照明が滲み外形周辺が色相範囲内

(c) 失敗例 2: 中央下から色相範囲外

(d) 失敗例 3: 左側の上下の外形部分が内部照明で照らされていない

図 39　外形通り抽出できている案内サインとできていない案内サイン

8.1.5　撮影時間による抽出精度の変化

本実験に使用した実験画像は，COVID-19 の感染対策のため利用者が少ない早朝に撮影したものである．事前実験や提案手法の各フェーズに使用した JR 八王子駅の画像も早朝に撮影した．そのため，太陽光の影響による抽出精度の変化を想定した実験は行えなかった．JR 新宿駅は，太陽光が直接案内サインに当たることがないため，精度や撮影環境が著しく変化することはないと考えられる．ただし，太陽光が直接内サインに当たる駅では，抽出精度の低下や分割されて抽出される枚数の増加が予想されるため，対象駅を調査し実験する必要がある．

8.2　全天球カメラによる歪みの緩和

本提案手法では，文字セグメンテーションのために，全天球カメラによる歪みの緩和を行った．しかし，第 8.1.2 節で述べたとおり，看板の外形が正しく取得できていないため，正しく歪み緩和が行えなかった．図 39 を歪み緩和行った画像を図 40 に示す．図 40(a) は，2 番乗り場の上部が欠けていたため一部正しく補正できなかったと考えられる．図 40(b) は，図の外形が波打ってたため内部が正しく補正できなかった．図 40(c)・図 40(d) は，外形が正しく取得できていないため，一部もしくは全体を正しく補正できていない．本提案手法は，外形が正しく取得できていない場合，正しく補正ができない．しかし，文字のセグメンテーションを考慮した場合，図 40(a) と図

40(b) と図 40(d) の一部文字は目視で読み取ることができるため，次の処理に及ぶ影響は少ないと考える．改善案としては，歪み緩和を行や文字・ピクトグラムから歪みを推定すること (表 20 改善案 7) で，看板の外形が正しく取得できない場合でも補正できると考える．

8.3　文字セグメンテーション

文字のセグメンテーションは，ほとんどの画像に対して正しく分割できなかった．主な原因は背景色の範囲の特定の失敗である．背景色の範囲の特定には，第 5.1 節の看板抽出のフェーズで特定できた色相範囲の情報と看板画像の背景を元に HSV 色空間の範囲を決めて背景のみを削除する手法になっている．この手法は，看板抽出のフェーズの色相範囲が正しく抽出できる範囲でなければ，その範囲内の看板画像内の色の詳細な範囲を取得するため正確に背景色を取り除くことが難しい．背景色が正しく削除できなかった場合，その後の文字の外形取得が正しく認識できないため，文字の抽出が行えない．この問題は，案内サイン用に複数の設定したカメラで撮影すること (表 20 改善案 6) や，看板抽出時に使用する色相範囲を細かく分割し，それを使用すること (表 20 改善案 3) で解決できると考える．

8.4　CNN を用いた文字認識

8.4.1　提案手法の文字セグメンテーションの画像を用いた文字認識

本提案手法で用いて全天球画像から文字をセグメンテーションした画像の文字認識結果は，総数が少ない上に精度が低い結果となった．精度が低い原因は，以下の 2 点

(a) 成功例 1: ほとんど補正できている

(b) 失敗例 1: 波状の外形通り補正してしまっている

(c) 失敗例 2: 画像が破綻している

(d) 失敗例 3: 左側が破綻している

図 40　図 39 を歪み緩和行った画像

が考えられる．

1. 歪み緩和と文字のセグメンテーションの精度の低さ
2. 文字セグメンテーション時の背景色特定率の低さ

前節の通り，歪み緩和と文字のセグメンテーションは多くの画像で正しく分割できない結果となった．本提案手法の文字認識は 1 文字の画像しか識別できない．そのため，複数文字の画像に対して特徴量が似た 1 文字の認識結果を出力するため，正しい認識結果を出力することはできない．文字認識より前の精度を向上させることによりこの問題は解決できると考える．

8.4.2　手動で文字のセグメンテーションを行った画像を用いた文字認識

提案手法の文字セグメンテーションでは，文字認識の精度を正しく評価できないため，全天球画像から手動でセグメンテーションを行った画像の認識率を算出した．

表 17 と表 18 の文字認識結果は，大きな差が確認できた．これは，識別するラベル数の差だと考えられる．表 17 はラベル数が 200 に対し，表 18 はラベル数が 1631 と大きくかけ離れている．事前実験に使用した JR 八王子駅のラベル数は 94，JR 立川駅のラベル数は 153 である．このように，1 つの駅で使用される文字は，全駅と全路線名より少ないことがわかる．ラベル数が少ない場合，比較する特徴量の数が減るため誤認識の確率が減る．そのため，表 17 のラベルを使用した文字認識の精度が高い結果になったと考えられる．また，矢印とピクトグラムの精度に変化がない原因は，他の文字種と比較した場合，矢印とピクトグラムのラベル数の差が少ない事と，似た特徴量のものが少ないためだと考えられる．

案内サインにおけるアルファベットは，日本語や矢印，ピクトグラムより小さく表示されている．そのため接写しない限り 1 文字の解像度は日本語の半分未満になることが多い．数字は，比較的大きく記述されていることが多いが，どの数字も特徴量が少ないため，VGG16 のような比較的大きいモデルでは，畳込みの過程で特徴量がなくなってしまっている可能性がある．

英語と数字の精度を向上させるためには，学習画像の生成方法を全角文字と同様に横方向に伸ばす処理を行う手法や，本研究ではできなかったが，日本語と英語の位置関係やサイズなどを用いて言語を特定し学習済みデータを切り替える手法 (表 20 改善案 8) が挙げられる．

また，日本語と日本語以外の文字が記載されている案内サインもある．このような

案内サインは，同じ情報を多言語で表示しているため，一方の言語の情報が正しく認識できなくても，もう一方の言語が認識できれば情報の補完ができるため，翻訳することで案内サインの情報取得の精度が向上すると考えられる．

8.5 辞書を用いた認識結果の訂正

本研究では，駅名辞書と路線名辞書とランドマーク辞書を作成し，その辞書と認識結果の訂正ができるかを検証した．提案手法で文字のセグメンテーションを行った画像の認識結果は，単語の中の 1 文字しか認識できなかったため，候補が絞り込めずに修正できなかった．手動でセグメンテーションを行った画像の文字文字結果でも，3 単語中 1 単語しか正しい結果にならなかった．原因は，以下の 2 点である．

1. 生成した辞書が不完全だったこと
2. 訂正に使用した式では正しく訂正できないこと

生成した辞書は前述の通り，駅名辞書と路線名辞書とランドマーク辞書の 3 つである．表 19 の 2 行目の「正しい答え」の「東口」という単語は含まれていない．これは，上記 3 つの辞書に含まれていないからである．解決方法は，駅に使用されるような単語辞書を作成することである．例えば，東西南北の出口や改札，きっぷうりばやみどりの窓口といった駅構内の施設名を含む辞書である．また，ランドマーク辞書の出典元の Wikipedia には「東口」という単語があるが，今回対象とした箇条書きには「東口」単体の単語はなかった．そのため，箇条書き以外にも文章で書かれた説明から単語抽出や，見出しや駅周辺以外の項目から取得することにより解決することもできると考える (表 20 改善案 9).

訂正に使用した数式は，各文字の信頼度を乗算した値にレーベンシュタイン距離の逆数を乗算したものを使用した．この計算式では，辞書に含まれていない単語は，辞書に含まれている単語に訂正されてしまう．そのため，表 19 の 4 行目の「新宿御苑」は駅名辞書内にある「新宿御苑前」駅に訂正した．この問題の解決方法は，辞書内に正解の単語が含まれていない場合でも正しい評価を行うことができる数式に変更すること (表 20 改善案 10) である．レーベンシュタイン距離を求める関数，len はリストの要素数以求める関数にも認識結果の文字と辞書内の単語の距離を計測する方法はあるので，そちらも検証するべきである．

8.6　各手法の改善案

本実験から導かれる提案手法の改善案を表 20 に示す．表 20 を改善することにより，提案手法全体の精度が向上すると考えられる．

表 20　抽出された看板画像の内訳 (単位:枚) と看板抽出の精度

No.	対象手法名	改善案
改善案 1	看板抽出	1 文字のみの抽出の許容
改善案 2	看板抽出	取得する看板の範囲制限
改善案 3	看板抽出	色相範囲の制限または分割
改善案 4	看板抽出	入力画像を編集し看板と背景の差を広げる
改善案 5	看板抽出	物体検出の使用
改善案 6	撮影時	案内サイン用のカメラ設定を複数用意
改善案 7	歪みの緩和	外形以外の情報から歪みを推定
改善案 8	文字認識	文字種ごとに学習済みデータを切り替える
改善案 9	誤り訂正	辞書作成方法の改善
改善案 10	誤り訂正	誤り訂正に使用した評価式の改善

9　結論

9.1　結論

本研究では，全天球カメラで撮影した駅構内にある案内サインの文字認識を行う手法を提案した．全天球カメラで撮影した全天球画像から看板を抽出する手法は，一部の黒色の看板以外ほとんどの案内サインを含む看板を抽出することができた．全天球カメラによる歪みを緩和する手法は，看板の外形を利用するため，外形が正しく取得できている看板の画像には有効だったが，外形が正しく取得できていない画像に関しては，歪みを緩和することができなかった．歪みを緩和した看板画像から文字をセグメンテーションする手法は，背景色が正しく削除できない場合があるため，改善する余地がある．看板内の文字を認識する OCR は，正しくセグメンテーションされている文字に対しては有効であった．辞書を用いる誤り訂正を行う手法は，辞書に記載されている単語は正しく訂正し，辞書に記載されていない単語は，誤った訂正を行うことがあった．

9.2　今後の課題

今後の課題として，以下の 4 つが挙げられる．

1 つ目は，すべての駅で看板を外形を正しく抽出することである．事前実験や提案手法のパラメータの調整で使用した JR 八王子駅では，高精度で看板の抽出を行えた．しかし，調整に使用していない JR 新宿駅では，看板を抽出することはできたが，看板の一部や看板周辺の背景まで抽出してしまった．後のフェーズである，歪み緩和や文字のセグメンテーションの精度向上のためにも，正しく外形を取得する必要がある．

2 つ目は，外形以外を使用した歪み緩和を行うことである．現在は，外形を利用しているため，前フェーズの看板抽出で外形が崩れた場合，この手法は，有効ではなくなる．そのため，行の歪み情報を用いた歪み緩和や，全天球画像を複数枚に分割し，カメラのレンズからキャリブレーションを行うなどの，外形が取得できない場合の手法が必要だと考える．

3 つ目は，生成型学習法を用いた文字認識で精度が低かった英語と数字の認識率の改善である．半角という共通点があるため，学習画像の精製方法の改善や使用するモデルの変更などにより改善できると考える．

4 つ目は，誤り訂正を行う辞書と訂正に用いた計算式の改良である．辞書は駅構内にある単語をなるべく記載することにより正しく訂正でき，計算式は辞書に記載されていない単語を誤った結果に訂正しないように改良するべきである．

以上の 4 つの課題を改善する表 20 を適用し，最終目標である「駅構内のナビゲーション」を達成したい．

10　本研究の意義

本研究では，「駅構内の情報を利用せずに駅構内のナビゲーション」という最終目標を達成するために，全天球カメラで撮影した全天球画像の文字認識を行う手法を提案した．建造物内のナビゲーションに多く用いられる建造物の構造解析・調査や，BLE ビーコンや QR コードなどを用いた自己位置推定などの技術からのアプローチではなく，本研究は既存の案内サインとコンピュータビジョンの技術で目標を達成しようとした．新しいアプローチを提案することにより，事前情報を使用しない建造物内のナビゲーションという大きな問題を解決する一歩になると考える．

参考文献

[1] 日本政府観光局, ”訪日外客統計の集計・発表”, <https://www.jnto.go.jp/jpn/statistics/data_info_listing/index.html>, (参照 2021/01/15).

[2] 三井 UFJ リサーチ&コンサルティング, ”外国人観光客の首都圏交通インフラ利用調査結果のお知らせ”, <https://www.murc.jp/wp-content/uploads/2014/06/cr_140613.pdf>, (参照 2021/01/15).

[3] NPO 法人 まちの案内推進ネット, ”「外出と交通の案内について」アンケート調査”, <http://www.annai.or.jp/project/enquete.pdf>, (参照 2021/01/15).

[4] Yahoo!JAPAN, ”Yahoo!MAP”, <https://map.yahoo.co.jp/promo/>, (参照 2021/01/15).

[5] NAVITIME, サービス紹介ページ 駅構内ルート, <https://products.navitime.co.jp/function/2535.html> (参照 2021/01/15).

[6] 市川裕介, 中村幸博, 中村泰治, 手塚博久, 瀬下仁志, 深田聡, 三井英毅, ”2020 Airport/Station˜ 訪日外国人向け空港˜ 駅でのおもてなし˜”, 映像情報メディア学会誌 Vol. 71 No. 3, pp.185–191, 2017.

[7] 国土地理協会, ”緯度経度付き全国沿線・駅データベース”, <http://www.kokudo.or.jp/database/004.html> (参照 2021/01/15).

[8] Daily Portal, 案内サインは必死に呼びかける, <https://dailyportalz.jp/kiji/141002165297> (参照 2021/01/15).

[9] Google, CLOUD VISION API, <https://cloud.google.com/vision/> (参照 2021/01/15).

[10] Adobe, Adobe Document Cloud, <https://acrobat.adobe.com/jp/ja/acrobat.html> (参照 2021/01/15).

[11] Evernote, Evernote, <https://evernote.com/intl/jp> (参照 2021/01/15).

[12] Google, Google ドライブ, <https://www.google.com/intl/ja_ALL/drive/> (参照 2021/01/15).

[13] 荒木禎史, 関海克, 小島啓嗣, ”製本原稿スキャン画像の歪み補正技術”, Ricoh Technical Report, No.29, pp.31–40, 2003.

[14] 志久修, 手島裕詞, 内田誠一, ”傾斜文字認識のための正規化方法”, 電子情報通信

学会論文誌 D, Vol.J100-D, No.10, pp.902–906, 2017.

[15] 成田了, 大山航, 若林哲史, 木村文隆. ”3 次元回転不変カメラベース文字認識”, 電気学会論文誌 C (電子・情報・システム部門誌), 133 巻, 4 号, pp.876–882, 2013.

[16] 村瀬洋, ”画像認識における生成型学習法”, 情報処理学会研究報告コンピュータビジョンとイメージメディア, 91(2004-CVIM-145) 号, pp.211–218, 2004.

[17] 石田皓之, 高橋友和, 井手一郎, 目加田慶人, 村瀬洋, ”携帯カメラ入力型文字認識におけるぼけやぶれに対処するための生成型学習法”, 電子情報通信学会論文誌 D, Vol.J89D, No.9, pp.2055–2064, 2006.

[18] Kenichiro Ofusa, Tomo Miyazaki, Yoshihiro Sugaya, Shinichiro Omachi, ”Glyph-Based Data Augmentation for Accurate Kanji Character Recognition”, 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), Kyoto, pp.597–602, 2017.

[19] 比留川翔哉, 丸山一貴, ”紙面の色と見開きの歪みを考慮した光学文字認識の実装と評価”, 第 18 回情報科学技術フォーラム, No. 3, pp. 25–30, 2019.

[20] Shoya Hirukawa, Kazutaka Maruyama, ”Optical Character Recognition for Navigation Signs in Japanese Stations”, Document Analysis Systems Short Paper, 2020.

[21] Sari Dewi Budiwati, J. Haryatno and Eddy Muntina Dharma, ”Japanese character (Kana) pattern recognition application using neural network”, Proceedings of the 2011 International Conference on Electrical Engineering and Informatics, Bandung, pp.1-6, 2011.

[22] J. Bai, Z. Chen, B. Feng and B. Xu, ”Chinese Image Text Recognition on grayscale pixels,” 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Florence, pp. 1380-1384, 2014.

[23] Kai Zhou, Yongsheng Zhou, Rui Zhang, Xiaolin Wei, ”An Improved Convolutional Block Attention Module for Chinese Character Recognition”, document Analysis Systems, pp.18–29, 2020.

[24] 野村松信, ”情景画像における看板抽出アルゴリズムの開発に関する研究”, 秋田大学, 博士論文, 2015.

[25] Xuebo Liu, Ding Liang, Shi Yan, Dagui Chen, *et al.*, ”FOTS: Fast oriented text spotting with a unified network”, The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.5676–5685, 2018.

[26] Qin Siyang, Bissacco Alessandro, Raptis Michalis, Fujii Yasuhisa and Xiao, Ying, ”Towards unconstrained end-to-end text spotting”, Proceedings of the IEEE International Conference on Computer Vision, pp.4704–4714, 2019.

[27] Dongjin Lee, Hosub Yoon, Myung-Ae Chung, Jaehong Kim, ”Robust Sign Recognition System at Subway Stations Using Verification Knowledge”, ETRI Journal Volume 36 Number 5, pp.696–703, 2014.

[28] Sayyan N. Shaikh, Neelakantha V. Londhe, ”OCR Based Mapless Navigation Method of Robot”, International Journal of Inventive Engineering and Sciences, Vol. 1, No. 8, pp. 6–12, 2013.

[29] 石井雄飛, 栗原徹, ”全天球画像の円筒平面における直線検出を用いた水平補正”, 情報処理学会 第 79 回全国大会講演論文集, Vol. 2017, No. 1, pp.259–260, 2017.

[30] Takeshi Yashiro, Haruhisa Hirayama, Ken Sakamura, ”An Indoor Localization Service using 360 Degree Spherical Camera”, 2020 IEEE 2nd Global Conference on Life Sciences and Technologies, pp.17–18, 2020.

[31] 山中優太朗, 粥川青汰, 髙木啓伸, 長岡雄一, 平塚義宗, 栗原 聡, ”One-Shot Wayfinding System: 360 度スマートフォンカメラと矢印分析を用いた視覚障碍者のための公共施設における方向決定支援システム”, 第 28 回インタラクティブシステムとソフトウェアに関するワークショップ (WISS2020), pp.7–12, 2020.

[32] João Guerreiro, Dragan Ahmetovic, Daisuke Sato, Kris Kitani, *et al.*, ”Airport Accessibility and Navigation Assistance for People with Visual Impairments”, Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, pp.1–14, 2019.

[33] João Guerreiro, Daisuke Sato, Saki Asakawa, Huixu Dong, *et al.*, ”CaBot: Designing and Evaluating an Autonomous Navigation Robot for Blind People”, The 21st International ACM SIGACCESS Conference on Computers and Accessibility, pp.68–82, 2019.

[34] 三部裕史, 大森健児, ”言語情報にもとづく候補文字補完を用いた文字認識後処理”, 情報処理学会論文誌 Vol. 36, No. 4, pp.840–848, 1995.

[35] 三部裕史, 大森健児, ”信頼性の低い文字認識結果に対する言語情報を用いた誤認識文字の訂正”, 情報処理学会論文誌 Vol. 34, No. 10, pp.2117–2124, 1993.

[36] Guillaume Chiron, Antoine Doucet, Mickaël Coustaty, Jean-Philippe Moreux, ”ICDAR2017 Competition on Post-OCR Text Correction”, 14th IAPR

International Conference on Document Analysis and Recognition (ICDAR), pp.1423–1428, 2017.

[37] RICOH, ”製品紹介 RICOH THETA Z1”, <https://theta360.com/ja/about/theta/z1.html>, (参照 2021/01/15).

[38] Simonyan, Karen, and Andrew Zisserman, ”Very deep convolutional networks for large-scale image recognition”, arXiv preprint arXiv:1409.1556, 2014.

[39] 駅データ.jp, 駅データ 無料ダウンロード サービス 『駅データ.jp』, <https://www.ekidata.jp/> (参照 2021/01/15).

[40] Wikipedia, メインページ, <https://ja.wikipedia.org/wiki/%E3%83%A1%E3%82%A4%E3%83%B3%E3%83%9A%E3%83%BC%E3%82%B8> (参照 2021/01/15).

[41] Wikipedia, Index of /jawiki/, <https://dumps.wikimedia.org/jawiki/> (参照 2021/01/15).

[42] JR 東日本, 新宿駅の構内図 <https://www.jreast.co.jp/estation/stations/866.html> (参照 2021/01/15).

[43] Apple, iPhone 12 Pro, <https://www.apple.com/jp/iphone-12-pro/> (参照 2021/01/19).

[44] intel REALSENSE, Intel®RealSense™Depth Camera D455, <https://www.intelrealsense.com/depth-camera-d455/> (参照 2021/01/19).

謝辞

指導教員である丸山一貴准教授には，研究全体の見通しや本論文に関する様々なアイデアやご指摘をいただきました．心よりお礼申し上げます．多くの助言・指摘を頂いた丸山研究室・横山研究室の皆様や，国士舘大学中村研究室・東海大学大西研究室の皆様，14TH IAPR INTERNATIONAL WORKSHOP ON DOCUMENT ANALYSIS SYSTEMS (DAS2020)，第 225 回コンピュータビジョンとイメージメディア研究発表会の参加者の皆様に感謝の意を表します．

発表一覧

- 比留川翔哉, 丸山一貴, ”紙面の色と見開きの歪みを考慮した光学文字認識の実装と評価”, 第 18 回情報科学技術フォーラム, No. 3, pp. 25–30, 2019.
- Shoya Hirukawa, Kazutaka Maruyama, ”Optical Character Recognition for Navigation Signs in Japanese Stations”, Document Analysis Systems Short Paper, 2020.
- 比留川翔哉, 丸山一貴, ”駅構内の案内サインを対象とした全天球画像による光学文字認識”, 研究報告コンピュータビジョンとイメージメディア (CVIM), Vol. 2021-CVIM-225, No. 42, pp. 1–8, 2021.

付録 A　実験に用いたプログラム

実験に用いたプログラムの一部を図 A.1, 図 A.2 に示す．図 A.1 は提案手法の，看板抽出，歪み緩和，文字のセグメンテーションまでを行うプログラムである．図 A.2 は生成型学習法による学習画像の生成プログラムである．

```
import os
import shutil
import numpy as np
import cv2
from datetime import datetime


class SignSegment:
    def __init__(self, exportPath):
        self.exportPath = exportPath
        self.NOT_SIGN_AREA_PAR = 0.3
        self.MIN_NOT_SIGN_AREA_PIXEL = 500
        self.SQUARE_RATIO = 0.2
        self.BACKCOLOR_MAGINE = 10

        r_minS = 210
        r_maxS = 255
        r_minV = 100
        r_maxV = 255
        self.MIN_RED_RANGE1 = np.array([0, r_minS, r_minV])
        self.MAX_RED_RANGE1 = np.array([45, r_maxS, r_maxV])
        self.MIN_RED_RANGE2 = np.array([150, r_minS, r_minV])
        self.MAX_RED_RANGE2 = np.array([179, r_maxS, r_maxV])

        g_minS = 60
        g_maxS = 255
        g_minV = 40
        g_maxV = 255
        self.MIN_GREEN_RANGE = np.array([30, g_minS, g_minV])
        self.MAX_GREEN_RANGE = np.array([105, g_maxS, g_maxV])

        bl_minS = 90
        bl_maxS = 255
        bl_minV = 230
        bl_maxV = 255
        self.MIN_BLUE_RANGE = np.array([90, bl_minS, bl_minV])
        self.MAX_BLUE_RANGE = np.array([165, bl_maxS, bl_maxV])

        w_minS = 0
        w_maxS = 15
        w_minV = 230
        w_maxV = 255
        self.MIN_WHITE_RANGE = np.array([0, w_minS, w_minV])
        self.MAX_WHITE_RANGE = np.array([179, w_maxS, w_maxV])

        bk_minS = 0
        bk_maxS = 255
        bk_minV = 0
        bk_maxV = 40
        self.MIN_BLACK_RANGE = np.array([0, bk_minS, bk_minV])
        self.MAX_BLACK_RANGE = np.array([179, bk_maxS, bk_maxV])

```

```
53          self.RANGE_NAMES = ["red", "green", "blue", "white", "black"]
54          self.dGCScnt = 0
55          self.GBC_PAR = 0.05
56
57      def rgb2hsv(self, ccode):
58          if len(ccode) != 3:
59              raise Exception(f"rgb2hsv Error: {str(ccode)}")
60          return cv2.cvtColor(np.uint8([[ccode]]), cv2.COLOR_RGB2HSV)[0][0]
61
62      def hsv2rgb(self, ccode):
63          if len(ccode) != 3:
64              raise Exception(f"rgb2hsv Error: {str(ccode)}")
65          return cv2.cvtColor(np.uint8([[ccode]]), cv2.COLOR_HSV2RGB)[0][0]
66
67      def getMask(self, img, name, reverse=False):
68          name = name.lower()
69          hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
70          if name == "red":
71              redMask1 = cv2.inRange(hsv,
72                                     self.MIN_RED_RANGE1,
73                                     self.MAX_RED_RANGE1)
74              redMask2 = cv2.inRange(hsv,
75                                     self.MIN_RED_RANGE2,
76                                     self.MAX_RED_RANGE2)
77              redMask = redMask1 + redMask2
78              if reverse:
79                  redMask = cv2.bitwise_not(redMask)
80              return cv2.bitwise_and(img, img, mask=redMask)
81
82          elif name == "green":
83              greenMask = cv2.inRange(hsv,
84                                      self.MIN_GREEN_RANGE,
85                                      self.MAX_GREEN_RANGE)
86              if reverse:
87                  greenMask = cv2.bitwise_not(greenMask)
88              return cv2.bitwise_and(img, img, mask=greenMask)
89
90          elif name == "blue":
91              blueMask = cv2.inRange(
92                  hsv, self.MIN_BLUE_RANGE, self.MAX_BLUE_RANGE)
93              if reverse:
94                  blueMask = cv2.bitwise_not(blueMask)
95              return cv2.bitwise_and(img, img, mask=blueMask)
96
97          elif name == "white":
98              whiteMask = cv2.inRange(
99                  hsv, self.MIN_WHITE_RANGE, self.MAX_WHITE_RANGE)
100             if reverse:
101                 whiteMask = cv2.bitwise_not(whiteMask)
102             return cv2.bitwise_and(img, img, mask=whiteMask)
103
104         elif name == "black":
105             blackMask = cv2.inRange(
106                 hsv, self.MIN_BLACK_RANGE, self.MAX_BLACK_RANGE)
107             if reverse:
108                 blackMask = cv2.bitwise_not(blackMask)
109             return cv2.bitwise_and(img, img, mask=blackMask)
110
111         elif name == "all":
112             redMask1 = cv2.inRange(
113                 hsv, self.MIN_RED_RANGE1, self.MAX_RED_RANGE1)
114             redMask2 = cv2.inRange(
115                 hsv, self.MIN_RED_RANGE2, self.MAX_RED_RANGE2)
```

```
            redMask = redMask1 + redMask2
            if reverse:
                redMask = cv2.bitwise_not(redMask)
            redImg = cv2.bitwise_and(img, img, mask=redMask)
            greenMask = cv2.inRange(
                hsv, self.MIN_GREEN_RANGE, self.MAX_GREEN_RANGE)
            if reverse:
                greenMask = cv2.bitwise_not(greenMask)
            greenImg = cv2.bitwise_and(img, img, mask=greenMask)
            blueMask = cv2.inRange(
                hsv, self.MIN_BLUE_RANGE, self.MAX_BLUE_RANGE)
            if reverse:
                blueMask = cv2.bitwise_not(blueMask)
            blueImg = cv2.bitwise_and(img, img, mask=blueMask)
            whiteMask = cv2.inRange(
                hsv, self.MIN_WHITE_RANGE, self.MAX_WHITE_RANGE)
            if reverse:
                whiteMask = cv2.bitwise_not(whiteMask)
            whiteImg = cv2.bitwise_and(img, img, mask=whiteMask)
            blackMask = cv2.inRange(
                hsv, self.MIN_BLACK_RANGE, self.MAX_BLACK_RANGE)
            if reverse:
                blackMask = cv2.bitwise_not(blackMask)
            blackImg = cv2.bitwise_and(img, img, mask=blackMask)
            return redImg, greenImg, blueImg, whiteImg, blackImg
        else:
            raise Exception(f"getMask Error: {name}")

    def getSigns(self, img):
        return self.getMask(img, "all")

    def signImg2signs(self, rawImg, signImg):
        # img, orgImg = im
        height, width, _ = signImg.shape
        gray = cv2.cvtColor(signImg, cv2.COLOR_RGB2GRAY)
        contours, _ = cv2.findContours(
            gray, cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)
        signs = []
        for j, c in enumerate(contours):
            area = cv2.contourArea(c)
            if width * height * self.NOT_SIGN_AREA_PAR < area or\
                    area < self.MIN_NOT_SIGN_AREA_PIXEL:
                continue
            ctrX = [x for x in c[:, 0, 0]]
            ctrY = [x for x in c[:, 0, 1]]
            minX = min(ctrX)
            maxX = max(ctrX)
            minY = min(ctrY)
            maxY = max(ctrY)
            newH = maxY - minY + 1
            newW = maxX - minX + 1
            newImg = np.zeros((newH, newW, 4), np.uint8)
            c = sorted(
                list(map(lambda point: point[0].tolist(), c)),
                key=lambda x: (x[1], x[0]))
            fst = c[0]
            row = 0
            size = []
            for ind, k in enumerate(range(len(c))):
                if k >= len(c) - 1 or c[k] > c[k + 1]:
                    kk = [[*x, 255] for x in rawImg[c[k][1], fst[0]:c[k][0]]]
                    if len(kk) == 0:
                        continue
```

```
                    newImg[row, fst[0] - minX:c[k][0] - minX] = kk
                    size.append(len(kk))
                    if k < len(c) - 1:
                        fst = c[k + 1]
                        row += 1
            signs.append(newImg)
        return signs

    def fixDstList(self, dst):
        spl = int(len(dst) / 2)
        for i in range(spl, -1, -1):
            if dst[i] != -1:
                continue
            for j in range(i, spl + 1):
                if dst[j] == -1:
                    continue
                dst[i] = dst[j]
                break
        for i in range(spl + 1, len(dst)):
            if dst[i] != -1:
                continue
            for j in range(i, spl, -1):
                if dst[j] == -1:
                    continue
                dst[i] = dst[j]
                break
        return dst

    def hosei(self, signImg, thr=0.1, dExportPath=None):
        if dExportPath is not None:
            if os.path.isdir(dExportPath):
                debug = True
        gray = cv2.cvtColor(signImg, cv2.COLOR_BGR2GRAY)
        h, w = signImg.shape[:2]
        _, thresh = cv2.threshold(gray, 0, 255, 0)
        contours, _ = cv2.findContours(
            thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)
        if len(contours) == 0:
            return None
        epsilon = 0.1 * cv2.arcLength(contours[0], True)
        map_x = np.zeros((h, w), dtype=np.float32)
        map_y = np.zeros((h, w), dtype=np.float32)
        tateCtr = {}
        yokoCtr = {}
        for i, cn1 in enumerate(contours[0]):
            x = cn1[0][0]
            y = cn1[0][1]
            if x in tateCtr:
                tateCtr[x].append(y)
            else:
                tateCtr[x] = [y]
            if y in yokoCtr:
                yokoCtr[y].append(x)
            else:
                yokoCtr[y] = [x]
        tateSize = []
        yokoSize = []
        for i in tateCtr.items():
            mn = min(i[1])
            mx = max(i[1])
            tateCtr[i[0]] = (mn, mx)
            tateSize.append(mx - mn)
        for i in yokoCtr.items():
```

```
242             mn = min(i[1])
243             mx = max(i[1])
244             yokoCtr[i[0]] = (mn, mx)
245             yokoSize.append(mx - mn)
246         tateSizeAve = int(round(sum(tateSize) / len(tateSize), 0))
247         yokoSizeAve = int(round(sum(yokoSize) / len(yokoSize), 0))
248         if tateSizeAve == 0:
249             return None
250         topDst = [-1 for _ in range(w)]
251         bottomDst = [-1 for _ in range(w)]
252         leftDst = [-1 for _ in range(h)]
253         rightDst = [-1 for _ in range(h)]
254         for i, item in enumerate(tateCtr.items()):
255             if item[1][1] - item[1][0] >= h * thr or True:
256                 topDst[item[0]] = item[1][0]
257                 bottomDst[item[0]] = item[1][1]
258         for i, item in enumerate(yokoCtr.items()):
259             leftDst[item[0]] = item[1][0]
260             if item[1][1] - item[1][0] >= w * 0.9:
261                 rightDst[item[0]] = item[1][1]
262 
263         topDst = self.fixDstList(topDst)
264         bottomDst = self.fixDstList(bottomDst)
265         map_x = np.zeros((h, w), dtype=np.float32)
266         map_y = np.zeros((h, w), dtype=np.float32)
267         np.set_printoptions(suppress=True, precision=0)
268         topDstMax = max(topDst)
269         topDst = [x for i, x in enumerate(topDst)]
270         kdTop = -1
271         kdBottom = -1
272         for i in range(1, h - 1):
273             try:
274                 if bottomDst[i - 1] - topDst[i - 1] == 0:
275                     continue
276                 if bottomDst[i] - topDst[i] == 0:
277                     continue
278                 if bottomDst[i + 1] - topDst[i + 1] == 0:
279                     continue
280             except IndexError:
281                 continue
282 
283             if kdTop == -1 and\
284                     not (topDst[i - 1] >= topDst[i] >= topDst[i + 1]):
285                 kdTop = i + 1
286             if kdBottom == -1 and\
287                     not (bottomDst[i - 1] <=
288                          bottomDst[i] <=
289                          bottomDst[i + 1]):
290                 kdBottom = i + 1
291 
292         preSignSize = None
293         for i in range(h):
294             try:
295                 signSize = bottomDst[i] - topDst[i]
296             except IndexError as e:
297                 if preSignSize is not None:
298                     signSize = preSignSize
299                 else:
300                     print(e)
301                     exit(1)
302             par = i / (h - 1)
303             for j in range(w):
304                 map_x[i, j] = j
```

```
                dst = topDst[j] * (1 - par) + \
                    (bottomDst[j] - tateSizeAve) * par
                map_y[i, j] = signSize / h + (i * signSize * 1.0 / h) + dst - 1
            preSignSize = signSize

        remapImg = cv2.remap(signImg.copy(), map_x, map_y, cv2.INTER_CUBIC)
        remapImg = cv2.resize(remapImg, (w, tateSizeAve), cv2.INTER_LINEAR)
        return remapImg

    def getCharShape(self, orgImg, name):
        hsv = cv2.cvtColor(orgImg.copy(), cv2.COLOR_BGR2HSV)
        print(self.getBGColor2(orgImg, name))
        minRange, maxRange = self.getBGColor2(orgImg, name)
        mhsvRange = np.array([(int(x + y) / 2)
                              for (x, y) in zip(minRange, maxRange)])
        rgbBcColor = self.hsv2rgb(mhsvRange)
        h, w = hsv.shape[:2]
        mask = cv2.inRange(hsv, minRange, maxRange)
        mask = cv2.bitwise_not(mask)
        img = cv2.bitwise_and(orgImg, orgImg, mask=mask)
        cfnt = len(os.listdir(self.debugEPath))
        self.dGCScnt += 1
        h, w, _ = orgImg.shape
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        contours, _ = cv2.findContours(
            gray, cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)
        points = []
        for j, c in enumerate(contours):
            area = cv2.contourArea(c)
            if w * h * self.NOT_SIGN_AREA_PAR * self.NOT_SIGN_AREA_PAR > area:
                continue
            ctrX = [x for x in c[:, 0, 0]]
            ctrY = [x for x in c[:, 0, 1]]
            minX = min(ctrX)
            maxX = max(ctrX)
            minY = min(ctrY)
            maxY = max(ctrY)
            newH = maxY - minY + 1
            newW = maxX - minX + 1
            newImg = np.zeros((newH, newW, 4), np.uint8)
            c = sorted(
                list(map(lambda point: point[0].tolist(), c)),
                key=lambda x: (x[1], x[0]))
            fst = c[0]
            row = 0
            for k in range(len(c)):
                if k >= len(c) - 1 or c[k] > c[k + 1]:
                    kk = img[c[k][1], fst[0]:c[k][0] + 1]
                    newImg[row, fst[0] - minX: c[k][0] - minX + 1, :] = kk
                    if k < len(c) - 1:
                        fst = c[k + 1]
                        row += 1
            point = (minX, minY, newH, newW)
            points.append([newImg, fst, point])
        if len(points) == 0:
            print("len(points) == 0")
            return None

        ip = sorted(points, key=lambda x: (x[2][1] * x[2][0]))

        thrH = int(h * 0.05)
        thrW = int(w * 0.025)

```

```
368             imgGp = [[ip[0]]]
369             imgGpCnt = 1
370             imgAddFlag = False
371             for i, (img, contour, pt) in enumerate(ip):
372                 if i == 0:
373                     continue
374                 for j in range(imgGpCnt):
375                     _x, _y = imgGp[j][0][2][:2]
376                     nx, ny = pt[:2]
377                     bx = nx - thrW < _x < nx + thrW
378                     by = ny - thrH < _y < ny + thrH
379                     if bx and by:
380                         imgGp[j].append(ip[i])
381                         imgAddFlag = True
382                         break
383 
384                 if not imgAddFlag:
385                     imgGp.append([ip[i]])
386                     imgGpCnt += 1
387                 imgAddFlag = False
388 
389             chars = self.imgGpSave(imgGp, rgbBcColor.tolist())
390             return chars
391 
392         def imgGpSave(self, ipgp, bgColor):
393             bgColor.append(255)
394             sumImgs = []
395             for i, ips in enumerate(ipgp):
396                 sumImgH = 9999
397                 sumImgW = 9999
398                 iapsH = 0
399                 iapsW = 0
400                 for j in ips:
401                     minX, minY, newH, newW = j[2]
402                     if sumImgW > minX:
403                         sumImgW = minX
404                     if sumImgH > minY:
405                         sumImgH = minY
406                 for j, idp in enumerate(ips):
407                     minX, minY, newH, newW = idp[2]
408                     snW = minX - sumImgW + newW
409                     snH = minY - sumImgH + newH
410                     if iapsH < snH:
411                         iapsH = snH
412                     if iapsW < snW:
413                         iapsW = snW
414                 if iapsH < iapsW:
415                     lsize = iapsW
416                     lsaH = round((iapsW - iapsH) / 2)
417                     lsaW = 0
418                 else:
419                     lsize = iapsH
420                     lsaH = 0
421                     lsaW = round((iapsH - iapsW) / 2)
422                 sumImg = np.zeros((lsize, lsize, 4), np.uint8)
423 
424                 sumImg[:, :] = [bgColor[2], bgColor[1], bgColor[0], bgColor[3]]
425                 for j, ip in enumerate(ips):
426                     minX, minY, newH, newW = ip[2]
427                     _w = int(minX - sumImgW + lsaW)
428                     _h = int(minY - sumImgH + lsaH)
429                     sumImg[_h: _h +
430                            newH, _w: _w +
```

```
                        newW] = self.paste(ip[0], sumImg[_h: _h +
                                                         newH, _w: _w +
                                                         newW])
            sumImgs.append(sumImg)
        return sumImgs

    def paste(self, fg, bg, x=0, y=0):
        img1 = bg.copy()
        img2 = fg.copy()
        w1, h1 = img1.shape[:2]
        w2, h2 = img2.shape[:2]
        roi = img1[x:x + w2, y:y + h2]
        mask = img2[:, :, 3]
        _, mask_inv = cv2.threshold(
            cv2.bitwise_not(mask),
            0, 255, cv2.THRESH_BINARY
        )
        img1_bg = cv2.bitwise_and(roi, roi, mask=mask_inv)
        img2_fg = cv2.bitwise_and(img2, img2, mask=mask)
        dst = cv2.add(img1_bg, img2_fg)
        img1[x:x + w2, y:y + h2] = dst
        return img1

    def getRange(self, name):
        if name == "red":
            return [[self.MIN_RED_RANGE1, self.MAX_RED_RANGE1],
                    [self.MIN_RED_RANGE2, self.MAX_RED_RANGE2]]
        elif name == "green":
            return [[self.MIN_GREEN_RANGE, self.MAX_GREEN_RANGE]]
        elif name == "blue":
            return [[self.MIN_BLUE_RANGE, self.MAX_BLUE_RANGE]]
        elif name == "white":
            return [[self.MIN_WHITE_RANGE, self.MAX_WHITE_RANGE]]
        elif name == "black":
            return [[self.MIN_BLACK_RANGE, self.MAX_BLACK_RANGE]]

    def getBGColor2(self, img, name, par=None):
        cimg = img.copy()
        if par is None:
            par = self.GBC_PAR
        hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        ww, hh = img.shape[:2]
        step = ww / hh
        sw = 0
        crange = self.getRange(name)
        minh, mins, minv = 999, 999, 999
        maxh, maxs, maxv = 0, 0, 0
        kcnt = 0
        for r in crange:
            for i in range(ww):
                for j in range(hh):
                    h = hsv[i, j, 0]
                    s = hsv[i, j, 1]
                    v = hsv[i, j, 2]
                    if r[0][0] < h < r[1][0] and\
                        r[0][1] < s < r[1][1] and\
                            r[0][2] < v < r[1][2]:
                        if minh > h:
                            minh = h
                        if mins > s:
                            mins = s
                        if minv > v:
                            minv = v
```

```
                        if maxh < h:
                            maxh = h
                        if maxs < s:
                            maxs = s
                        if maxv < v:
                            maxv = v
                        kcnt += 1
        return np.array([minh, mins, minv]), np.array([maxh, maxs, maxv])

    def getBGColor(self, img, margin=0.1, ccnt=10):
        w, h = img.shape[:2]
        mgW = int(w * margin)
        mgH = int(h * margin)

        l = [img[0 + mgW, 0 + mgH][:-1],
             img[0 + mgW, h - 1 - mgH][:-1],
             img[w - 1 - mgW, 0 + mgH][:-1],
             img[w - 1 - mgW, h - 1 - mgH][:-1],
             img[int(w / 2), int(h / 2)][:-1]
             ]
        r = 0
        g = 0
        b = 0
        cnt = 0
        for i, ii in enumerate(l):
            q, qq, qqq = ii
            if int(q) + int(qq) + int(qqq) == 0:
                continue
            b += q
            g += qq
            r += qqq
            cnt += 1
        if cnt == 0:
            if ccnt == 0:
                return None
            import random
            while True:
                r = random.random()
                if r < 0.4:
                    break
            return self.getBGColor(img, margin=r, ccnt=ccnt - 1)

        return [int(b / cnt), int(g / cnt), int(r / cnt)]
```

図 A.1　実験に使用したプログラムの一部

```
import os
import re
import sys
import shutil
import time
import cv2
import numpy as np
from PIL import Image, ImageDraw, ImageFont
from joblib import Parallel, delayed


def pp(b):
    for x in b:
        for y in x:
            print("{:5.1f}".format(y), end=" ")
        print()


def rgb2bgr(htmlColor):
    if "#" not in htmlColor:
        print("error rgb2bgr()")
        sys.exit(1)
    ht = htmlColor[0]
    if len(htmlColor) == 7:
        ht += htmlColor[5:7] + htmlColor[3:5] + htmlColor[1:3]
    elif len(htmlColor) == 9:
        ht += htmlColor[5:7] + htmlColor[3:5] + htmlColor[1:3] + htmlColor[7:9]
    else:
        print("error rgb2bgr() not 7 or 9")
        sys.exit(1)
    return ht


def html2color(htmlColor, type="np"):
    if "#" not in htmlColor:
        print("error html2color()")
        sys.exit(1)
    if len(htmlColor) == 7:
        if type == "tuple":
            return int(htmlColor[1:3], 16), \
                int(htmlColor[3:5], 16),\
                int(htmlColor[5:7], 16)
        elif type == "np":
            return(np.array([int(htmlColor[1:3], 16),
                             int(htmlColor[3:5], 16),
                             int(htmlColor[5:7], 16)]))
    elif len(htmlColor) == 9:
        if type == "tuple":
            return int(htmlColor[1:3], 16), \
                int(htmlColor[3:5], 16), \
                int(htmlColor[5:7], 16), \
                int(htmlColor[7:9], 16)
        elif type == "np":
            return(np.array([int(htmlColor[1:3], 16),
                             int(htmlColor[3:5], 16),
                             int(htmlColor[5:7], 16),
                             int(htmlColor[7:9], 16)]))
    else:
        print("error html2color() not 7 or 9")
        sys.exit(1)


def pil2cv(image):
```

```
    ''' PIL型 -> OpenCV型 '''
    new_image = np.array(image, dtype=np.uint8)
    if new_image.ndim == 2:
        pass
    elif new_image.shape[2] == 3:
        new_image = new_image[:, :, ::-1]
    elif new_image.shape[2] == 4:
        new_image = new_image[:, :, [2, 1, 0, 3]]
    return new_image


def cv2pil(image):
    ''' OpenCV型 -> PIL型 '''
    new_image = image.copy()
    if new_image.ndim == 2:
        pass
    elif new_image.shape[2] == 3:
        new_image = new_image[:, :, ::-1]
    elif new_image.shape[2] == 4:
        new_image = new_image[:, :, [2, 1, 0, 3]]
    new_image = Image.fromarray(new_image)
    return new_image


def chkRange(img, bgColor, fontsize, type="pil"):
    if type == "pil":
        _bgColor = html2color(bgColor, type="tuple")
        w, h = img.size
        f = False
        for x in range(w):
            if img.getpixel((x, 0)) != _bgColor:
                f = True
                break
            if img.getpixel((x, h - 1)) != _bgColor:
                f = True
                break
        if not f:
            for y in range(h):
                if img.getpixel((0, y)) != _bgColor:
                    f = True
                    break
                if img.getpixel((w - 1, y)) != _bgColor:
                    f = True
                    break
    elif type == "cv2":
        _bgColor = html2color(bgColor)
        w, h, _ = img.shape
        f = False
        for x in range(w):
            if all(img[x, 0] != _bgColor):
                f = True
                break
            if all(img[x, h - 1] != _bgColor):
                f = True
                break
        if not f:
            for y in range(h):
                if all(img[0, y] != _bgColor):
                    f = True
                    break
                if all(img[w - 1, y] != _bgColor):
                    f = True
                    break
```

```
127     return f
128
129
130 def sumImage(img, bgColor):
131     width, height, _ = img.shape
132     bg = cv2.cvtColor(np.full((height, width, 4),
133                               html2color(bgColor),
134                               dtype=np.uint8),
135                       cv2.COLOR_BGRA2RGBA)
136     w1, h1 = bg.shape[:2]
137     w2, h2 = img.shape[:2]
138     x = 0
139     y = 0
140
141     roi = bg[x:x + w2, y:y + h2]
142     mask = img[:, :, 3]
143
144     for x in range(w2):
145         for y in range(h2):
146             if mask[y, x] != 255:
147                 mask[y, x] = 0
148             else:
149                 break
150
151     mask_inv = cv2.bitwise_not(mask)
152     img1_bg = cv2.bitwise_and(roi, roi, mask=mask_inv)
153     img2_fg = cv2.bitwise_and(img, img, mask=mask)
154
155     dst = cv2.add(img1_bg, img2_fg)
156     return dst
157
158
159 def exportPgFile(filepath, svPath, cpath):
160     path = os.path.sep.join(cpath.split(os.path.sep)[:-1])
161     # print(f"path: {filepath}\n\twrite: {path}\n")
162     f = open(filepath, "w")
163     f.write(path)
164     f.close()
165
166
167 def makeImg2(svPath, char, nImg, imgSize, fontsize,
168              jpnFP, engFP, numFP, markFP, fgColor,
169              bgColor, dp, op, cpath, doPgExportPath=None):
170     if doPgExportPath is not None:
171         exportPgFile(doPgExportPath, svPath, cpath)
172     time.sleep(1)
173
174     if not os.path.isdir(svPath):
175         os.mkdir(svPath)
176
177     if pEng.fullmatch(char):
178         fontPath = engFP
179     elif pNum.fullmatch(char):
180         fontPath = numFP
181     elif pMark.fullmatch(char):
182         fontPath = markFP
183     else:
184         fontPath = jpnFP
185
186     chfontsize = fontsize
187     while True:
188         img = Image.new("RGBA", imgSize, bgColor)
189         draw = ImageDraw.Draw(img)
```

```
190         font = ImageFont.truetype(fontPath, chfontsize)
191         fontH, fontW = draw.textsize(char, font=font)
192         fw = abs((imgSize[1] - fontW) / 2)
193         fh = abs((imgSize[0] - fontH) / 2)
194         draw.text((fh, fw), char, font=font, fill=fgColor)
195         if chkRange(img, bgColor, chfontsize) or \
196             fw > imgSize[0] or \
197                 fh > imgSize[1]:
198             print("\"{}\"({}) Warning: change fontsize".format(chr, char))
199             chfontsize -= 1
200         else:
201             break
202 
203     img.save(os.path.join(svPath, "{}_0.png".format(ord(char))), "PNG")
204     cv2Img = pil2cv(img)
205     width, height, _ = cv2Img.shape
206     noiseNum = 20
207     chn = 1.6
208     remap = []
209     for i in range(1, nImg + 1):
210         xcnt = i
211         xtm = xcnt / noiseNum / chn / chn
212         ycnt = (i - nImg // 2)
213         ytm = ycnt * chn
214 
215         map_x1 = np.fromfunction(lambda y, x:
216             (x**(1 + xtm)) / (height**(xtm)),
217             (width, height),
218             dtype=np.float32)
219         map_x2 = np.fromfunction(lambda y, x:
220             abs(((height - x)**(xtm + 1)) / (height**xtm) - height + 1),
221             (width, height),
222             dtype=np.float32)
223         map_x3 = np.fromfunction(lambda y, x:
224              x, (width, height), dtype=np.float32)
225 
226         map_y1 = np.fromfunction(lambda y, x:
227             y - x * x / width / (nImg - ytm),
228             (width, height),
229             dtype=np.float32)
230         map_y2 = np.fromfunction(lambda y, x:
231             y - (width - x) * (width - x) / width / (nImg - ytm),
232             (width, height),
233             dtype=np.float32)
234         map_y3 = np.fromfunction(lambda y, x:
235             y, (width, height),
236             dtype=np.float32)
237 
238         remap = []
239         remap.append(sumImage(cv2.remap(cv2Img,
240                               map_x1,
241                               map_y1,
242                               cv2.INTER_AREA),
243                               bgColor))
244         remap.append(sumImage(cv2.remap(cv2Img,
245                               map_x2,
246                               map_y2,
247                               cv2.INTER_AREA),
248                               bgColor))
249         remap.append(sumImage(cv2.remap(cv2Img,
250                               map_x1,
251                               map_y3,
252                               cv2.INTER_AREA),
```

```
                                bgColor))
        remap.append(sumImage(cv2.remap(cv2Img,
                                map_x2,
                                map_y3,
                                cv2.INTER_AREA),
                                bgColor))
        remap.append(sumImage(cv2.remap(cv2Img,
                                map_x3,
                                map_y1,
                                cv2.INTER_AREA),
                                bgColor))
        remap.append(sumImage(cv2.remap(cv2Img,
                                map_x3,
                                map_y2,
                                cv2.INTER_AREA),
                                bgColor))

        for j in range(len(remap)):
            saveFileName = os.path.join(svPath, f"{char}_{i + nImg * j}.png")
            if chkRange(remap[j], bgColor, chfontsize, type="cv2"):
                print("Warning: out of range {}", saveFileName)
            print(saveFileName)
            cv2.imwrite(saveFileName, remap[j])


pEng = re.compile("[a-zA-Z]")
pNum = re.compile("[0-9]")
pMark = re.compile('[!"#$%&\'\\\\()*+,-./:;<=>?@[\\]^_‘{|}~「」'
                   '〔〕“”〈〉『』【】＆＊・（）＄＃＠。、？！｀＋￥％]')
tl = []
pgPath = ""
cnt = -1


def setting(svPath, clPath):
    global tl, pgPath, cnt
    f2iPath = os.path.dirname(__file__)
    savePdir = os.path.abspath(svPath)
    chrListPath = os.path.abspath(clPath)

    if not os.path.isfile(chrListPath):
        print("chrList not found")
        exit(1)

    if os.path.isdir(savePdir):
        while True:
            print(f"{savePdir} を削除してよろしいですか？ [Yn]", end=" ")
            yn = input().lower()
            if yn in ["y", "yes"]:
                print(f"削除中: {savePdir}... ", end="")
                shutil.rmtree(savePdir)
                print("完了")
                break
            elif yn in ["n", "no"]:
                print("終了します")
                exit(0)

    os.mkdir(savePdir)

    # 進捗具合
    pgPath = os.path.join(f2iPath, "Progress")
    if os.path.isdir(pgPath):
        shutil.rmtree(pgPath, ignore_errors=True)
```

```
316     os.mkdir(pgPath)
317
318     fontdataPath = os.path.join(f2iPath, "fonts")
319
320     print("生成条件チェック...", end="")
321
322     colorDict = {
323         "White": "#ffffffff",
324         "Black": "#000000ff",
325         "Green": "#009821ff",
326         "Yellow": "#ffd900ff"
327     }
328
329     imgConds = {
330         "nMakeImg": [x for x in range(50, 201, 50)],
331         # imgSize, fontsize
332         "size": [[128, 108], [64, 54], [32, 27], [16, 13]],
333         "fontPath": [{
334             "jpn": os.path.join(fontdataPath, "ShinGoPro-M.otf"),
335             "eng": os.path.join(fontdataPath, "MyriadPro-R.otf"),
336             "num": os.path.join(fontdataPath, "MyriadPro-R.otf"),
337             "mark": os.path.join(fontdataPath, "MyriadPro-R.otf")
338         }],
339         # back, fore
340         "bfColor": [[colorDict["White"], colorDict["Black"]],
341                     [colorDict["Yellow"], colorDict["Black"]],
342                     [colorDict["Green"], colorDict["White"]],
343                     [colorDict["Black"], colorDict["White"]]],
344         "bfColorName": ["W-B", "Y-B", "G-W", "B-W"],
345         "distPar": [1],
346         "oofPar": [0]
347     }
348     if len(imgConds["bfColor"]) != len(imgConds["bfColorName"]):
349         print("\nError: imgCondsのカラーとカラーネームが不正です")
350         exit(1)
351     print("OK")
352
353     print("check fontPath")
354     for i in imgConds["fontPath"]:
355         for k, v in i.items():
356             _b = os.path.isfile(v)
357             print(f"\t{k}: {v}: {'OK' if _b else 'NG'}")
358             if not _b:
359                 print(f"error: {v} not found")
360                 exit(1)
361     print("check fontPath... OK")
362
363     print("文字一覧取得中... ", end="")
364     chrList = [x for x in open(chrListPath).read().split("\n") if x]
365     print(f"OK\n文字数:{len(chrList)}文字")
366
367     print("make folder... ", end="")
368     makeDirCpath = [os.path.join(savePdir,
369                                  f"n{a}_i{b}_f{d}_c{e}_d{str(f).replace('.','')}"
370                                  f"_o{str(g).replace('.','')}",
371                                  str(ord(c)))
372                     for a in imgConds["nMakeImg"]
373                     for b, _ in imgConds["size"]
374                     for d in range(len(imgConds["fontPath"]))
375                     for e in imgConds["bfColorName"]
376                     for f in imgConds["distPar"]
377                     for g in imgConds["oofPar"]
378                     for c in chrList
```

```
                    ]

    print("OK")

    # debug
    for i in makeDirCpath:
        os.makedirs(i, exist_ok=True)
        # print(i)
        # pass

    makeImgCondList = [(c, a, b, d, e, f, g)
                       for a in range(len(imgConds["nMakeImg"]))
                       for b in range(len(imgConds["size"]))
                       for d in range(len(imgConds["fontPath"]))
                       for e in range(len(imgConds["bfColorName"]))
                       for f in range(len(imgConds["distPar"]))
                       for g in range(len(imgConds["oofPar"]))
                       for c in chrList
                       ]

    tmp = len(makeDirCpath) != len(makeImgCondList)
    print("check Fol and Clist... ", end="")
    if tmp:
        print("Error: 保存先リストと条件リストの数が合いません")
        exit(0)
    print("OK")

    print("設定生成")
    tl = []
    # bar = tqdm(total=len(makeDirCpath))
    cnt = -1
    for path, (c, n, s, fp, bf, dp, op) in zip(makeDirCpath, makeImgCondList):
        cndN = imgConds["nMakeImg"][n]
        cndS = imgConds["size"][s]
        cndFP = imgConds["fontPath"][fp]
        cndBF = imgConds["bfColor"][bf]
        cndDP = imgConds["distPar"][dp]
        cndOP = imgConds["oofPar"][op]
        # print(path)
        if os.path.basename(path) != str(ord(c)):
            print(path, str(ord(c)))

        if c == chrList[0]:
            cnt += 1

        writeFlag = None
        writeCPath = None
        if len(tl) == cnt:
            tl.append([])
            # print(path)
            pgFileName = f"pgEnd{cnt}.txt"
            if cnt != 0:
                writeCPath = tl[cnt - 1][-1][0]
                writeFlag = os.path.join(pgPath, pgFileName)

        tl[cnt].append((path,
                        c,
                        cndN,
                        (cndS[0], cndS[0]),
                        cndS[1],
                        cndFP["jpn"],
                        cndFP["eng"],
                        cndFP["num"],
```

```
                        cndFP["mark"],
                        cndBF[0],
                        cndBF[1],
                        cndDP,
                        cndOP,
                        writeCPath,
                        writeFlag
                        ))

    print("設定生成 end")
    return len(tl)


def run():
    tmp = 0
    for i in tl: tmp += len(i)
    print(f"tasks: {len(tl)} total:({tmp})")
    for ind, tasks in enumerate(tl):
        Parallel(n_jobs=-2, verbose=0)([
            delayed(makeImg2)(a, b, c, d, e, f, g, h, i, j, k, l, m, n, o)
            for a, b, c, d, e, f, g, h, i, j, k, l, m, n, o in tasks])
        print(f"font2img: end {ind+1}/{len(tl)} {(ind+1)/len(tl)*100:7.3f}%")

    # def exportPgFile(filepath, svPath, cpath):
    exportPgFile(os.path.join(pgPath,
                              f"pgEnd{cnt+1}.txt"),
                 tl[-1][-1][0], tl[-1][-1][0])
    open(os.path.join(pgPath, "pgEndALL.txt"), "w").write("allend")
```

図 A.2　学習画像プログラムの一部