明るさの影響を考慮した道路の俯瞰画像と航空写真のマッチング手法の検討

向井 聖人[1,a)]　栗 達[1,b)]　小野 晋太郎[1,c)]

概要：本研究では，カーブミラ内の道路を俯瞰画像へ変換するための射影変換行列の計算と，航空写真と俯瞰画像のマッチングを行う．
まず，3 つの手法で射影変換行列の計算に取り組んだ．3 つの手法のうち 1 つを用いて，実際に俯瞰画像への変換を行った．その後，変換した俯瞰画像を用い，航空写真とのマッチングを行った．この時，3 つの手法でマッチングを行い，それぞれの手法のマッチング精度の比較を行った．

Consideration of a Matching Method for Overhead Road Images and Aerial Photographs Considering the Influence of Brightness

1. はじめに

1.1 背景

近年，交通事故の削減や防止，運転手不足対策のための自動運転の実現など，社会的課題の解決に向け研究開発が進められている．このような課題を解決するためには，交通環境の安全性向上が必要不可欠である．

しかし，見通しの悪い交差点では周囲の状況を把握することが困難である．したがって，交通環境の安全性向上には，このような環境での視認性を改善する必要がある．

1.2 先行研究

これまで，視認性を改善するためにカーブミラーを用いた先行研究が行われている．

宮柱ら [1] は，車載カメラ画像からカーブミラーを検出するために，運転支援システムに適した物体検出モデルと最適化アルゴリズムの組み合わせを検討している．その中で，最適化アルゴリズムに SGDM を使用した YOLOv4 という物体検出モデルが適している，と結論付けている．

百崎 [2] は，エッジによる輪郭検出アルゴリズムを用いて，カーブミラーの輪郭を楕円として検出をしている．その後，検出した楕円の縦横比から，カーブミラーの角度推定をしている．輪郭検出率は 4 割程度，ミラーの角度は高い精度で推定可能だが，小さい角度 (0°〜10°) では誤差が大きい，と結論付けている．

1.3 目的・目標

見通しの悪い交差点にはカーブミラーが設置されている．しかし，カーブミラーには，ミラー越しに見た対象を実際の距離より遠く認識してしまったり [3]，対象のサイズやミラーの設置条件によって，接近してくる対象の視認距離が変化する [4] などの問題がある．

そこでこの研究では，図 1 の手順のように，カーブミラーに映った道路を俯瞰画像に変換し，地図とマッチングすることで死角となっている道路の状況を把握することを最終的な目的としている．本研究では，斜め視点から撮った道路を俯瞰画像に変換し，道路と航空写真のマッチング手法の検討を目標としている．

[1] 福岡大学
Fukuoka University
[a)] tl211237@cis.fukuoka-u.ac.jp
[b)] lida@fukuoka-u.ac.jp
[c)] onoshin@fukuoka-u.ac.jp

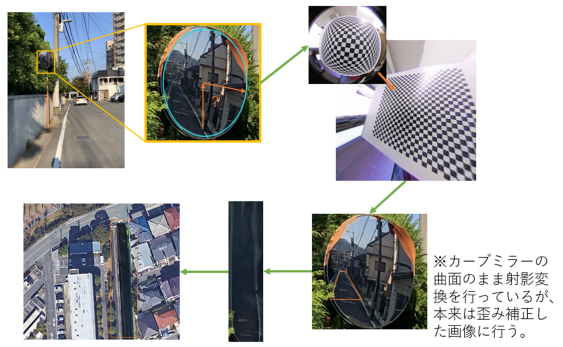

図 1 道路を地図とマッチングするまでの手順

2. 道路の俯瞰画像への変換

この章は，道路を俯瞰画像へ変換するために使用した技術の紹介と，変換手法の検討を行う．

2.1 斜め視点の道路画像の撮影

図 1 のように本来は，
(1) 車載カメラでカーブミラーを検出
(2) ミラー曲面のキャリブレーション（歪み補正）
(3) 道路画像を俯瞰画像化
という手順で俯瞰画像の作成を行う．しかし，本研究は道路と航空写真のマッチングを主題としているので，単純な条件で道路画像の撮影を行った．

実際の道路画像の撮影は，図 2 と図 3 のように斜め視点カメラの位置から行った．図 3 のように，カメラの撮影位置は道路の中央，カメラの向きは道路の進行方向の路面上になるように撮影を行った．

図 7 が実際に撮影した斜め視点の道路画像で，これを俯瞰画像へ変換する．

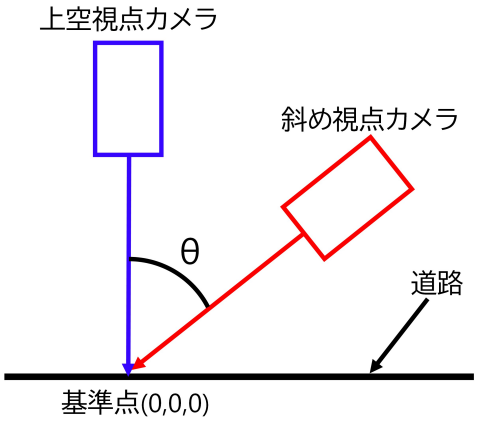

図 2 横から見た図

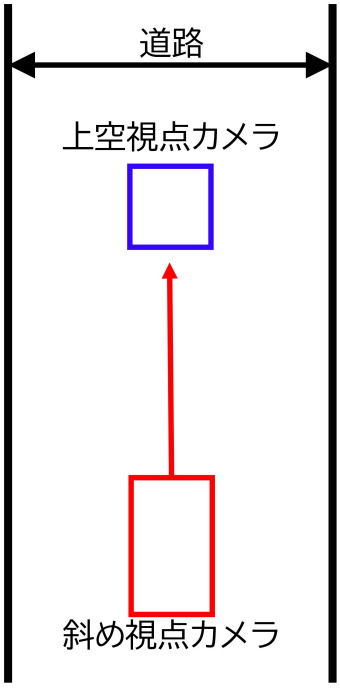

図 3 上空から見た図

2.2 射影変換

射影変換とは，任意の平面画像の形状を，異なる任意の平面画像の形状に変形する変換手法の 1 つである．例えば，図 4 のように斜めから撮影された道路の画像を，上空から見下ろした俯瞰画像に変換することができる．

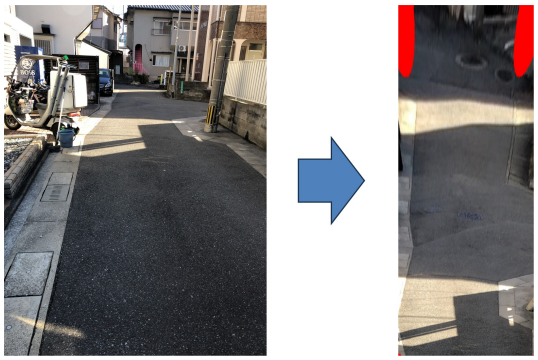

図 4 射影変換の例

Python の OpenCV ライブラリには，射影変換行列を計算する cv2.getPerspectiveTransform() 関数，射影変換を実行できる cv2.warpPerspective() 関数がある．

- cv2.getPerspectiveTransform()[8]
  入力画像上と出力画像上の少なくとも 4 組の対応点から射影変換行列を求める
  src - 入力画像上の四角形の頂点座標
  dst - 出力画像上の対応する四角形の頂点座標
  次の式で 3 × 3 の射影変換行列を求める．

$$\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix} = \text{map\_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \quad (1)$$

$$dst(i) = (x'_i, y'_i), src(i) = (x_i, y_i), i = 0, 1, 2 \quad (2)$$

- cv2.warpPerspective()[8]
  射影変換行列を用いて，入力画像を変換する
  src - 入力画像
  dst - src と同じタイプの出力画像
  M - 3 × 3 の変換行列
  次の式で，指定された変換行列を用いて入力画像を変換する．

$$\begin{aligned} &\text{dst}(x, y) \\ &= \text{src}\left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}\right) \end{aligned} \quad (3)$$

2.3 射影変換行列の計算

本研究では，射影変換行列を計算するためにいくつかの手法を検討した．

2.3.1 では 4 点の座標を指定する手法，2.3.2 では roll,pitch,yaw を用いた手法，2.3.3 では 7 自由度パラメータを用いた手法で計算した．

2.3.1 4 点の座標を指定する手法

まずは，射影変換したい道路の範囲をマウスイベントで指定する．この時，4 点の座標を指定する順番は，変換後の画像の左上に位置する座標から反時計回りに指定する．

指定した 4 点の座標と，2.2 で説明した cv2.getPerspectiveTransform() 関数を用いて射影変換行列を計算する．この射影変換行列を用いて，図 5 のように道路画像を俯瞰画像に変換する．

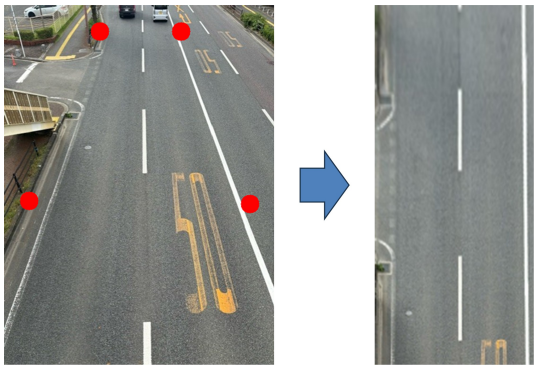

図 5 4 点の座標を指定した道路画像の射影変換

2.3.2 roll,pitch,yaw を用いた手法

カメラの回転姿勢を表現する方法として，roll,pitch,yaw という表現がある．この方法は，直交する 3 つの軸それぞれに回転角を設定することによって，カメラの回転姿勢を表現する．

OpenCV では図 6 のように右手系の座標系である．カメラの位置を原点とした場合，右方向に x 軸，下方向に y 軸，光の進む方向に z 軸をとる．斜め視点のカメラ姿勢を上空から見下ろすカメラ姿勢になるように 3 つの軸に回転角を設定することで，道路画像を俯瞰画像に変換する射影変換行列を計算できる．

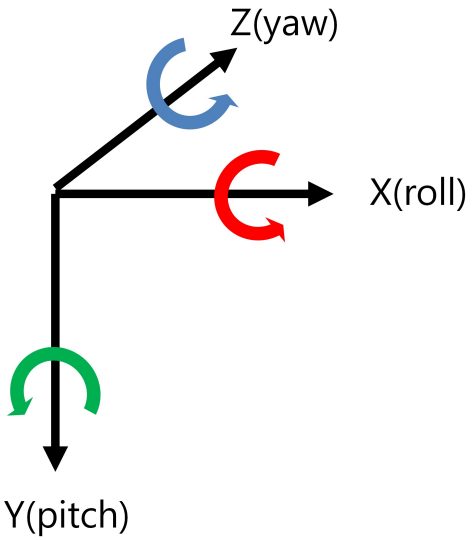

図 6 OpenCV の右手系の座標系

射影変換行列を計算するために，次の 3 つの行列を用意する．

- カメラ行列 K
  linalg.inv() 関数を用いて，カメラ行列 K の逆行列も求める．

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (4)$$

- 回転行列 R
  α に roll 角，β に pitch 角，γ に yaw 角を設定して，回転行列 R を求める．

$$R = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \quad (5)$$

- 平行移動行列 T
  射影変換を行うと変換画像が出力画像外に出ることがある．そこで，変換画像を出力画像内に収めるために

変換した俯瞰画像を移動する．

$$T = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

次の式で，用意した 3 つの行列から射影変換行列 H を計算する．

$$H = T \cdot K \cdot R \cdot K^{-1} \tag{7}$$

2.3.3 7 自由度パラメータを用いた手法

この手法は，7 つのパラメータを用いて射影変換行列の計算を行う．7 つのパラメータを列挙すると，

- カメラの画角
- 入力画像の横幅
- 上空視点カメラの姿勢と位置
- 斜め視点カメラの姿勢と位置
- 平行移動量

となる．

射影変換行列の計算には，カメラ行列 K，上空視点カメラと斜め視点カメラの 2 つの外部パラメータ行列 P，平行移動行列 T の 4 つの行列が必要である．それぞれの行列を必要なパラメータを用いて用意していく．

- カメラ行列 K
  カメラの画角 θ と入力画像の横幅 w を用いて，次の式から焦点距離 f を求める．

$$f \tan\left(\frac{\theta}{2}\right) = \frac{w}{2} \tag{8}$$

  求めた焦点距離を用いてカメラ行列 K を求める．

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

  求めたカメラ行列 K を 4×4 行列に拡張し, linalg.inv() 関数を用いて逆行列も求める．

$$K = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

- 外部パラメータ行列 P
  カメラの姿勢と位置を表す行列である．図 2 と図 3 から，2 つのカメラの姿勢は上空視点カメラの姿勢を基準に 3×1 の回転ベクトル，2 つのカメラの位置は基準点をもとに 3×1 の位置ベクトルで表す．
  続いて，cv2.Rodrigues() 関数を用いて，2 つの回転ベクトルを 3×3 の回転行列 R に変換する．回転行列 R と位置ベクトルを合わせることで，上空視点カメラと斜め視点カメラをそれぞれ表す，2 つの 4×4 の外部パラメータ行列 P が求まる．

$$P = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \tag{11}$$

  斜め視点カメラの外部パラメータ行列 P は, linalg.inv() 関数を用いて逆行列に変換する．
- 平行移動行列 T
  2.3.2 と同様に，出力画像内に収めるために，変換した俯瞰画像を移動する．(式 6)

次の式で，射影変換行列 H を計算する．

$$H = T \cdot K \cdot P_T \cdot P_C^{-1} \cdot K^{-1} \tag{12}$$

2.4 用いる手法と俯瞰画像への変換

本研究では，2.3.3 で述べた “7 自由度パラメータを用いた手法” で俯瞰画像への変換を行った．

次のような手順で俯瞰画像作成までを実装した．

(1) 斜め視点から道路を撮影し，道路画像として入力する
(2) カメラ行列 K と，その逆行列を求める
(3) 上空視点カメラの外部パラメータ行列 P_T を求める
(4) 斜め視点カメラの外部パラメータ行列の逆行列 P_C^{-1} を求める
(5) 出力画像内に収めるための調整量から，平行移動行列 T を求める
(6) 射影変換行列を計算し，道路画像を cv2.warpPerspective() 関数を用いて射影変換

まず，図 2 と図 3 の斜め視点カメラの位置から，θ が 45 ° になるカメラ姿勢で撮影を行った．実際に歩道橋の上から撮影した道路画像が図 7 であり，この道路画像を射影変換する．

図 7 実際に撮影した斜め視点の道路画像　481px × 640px

続いて，手順 2 のカメラ行列 K を求める．

焦点距離 f は式 8 から，カメラの画角 θ と入力画像の横幅 w で求める．この時，カメラの画角 $\theta = 60$，入力画像の横幅 $w = 481$ から焦点距離は $f = 416$ となり，式 9 の 3 × 3 の行列が求まる．これを拡張することで 4 × 4 のカメラ行列 K が求める．

その後，linalg.inv() 関数を用いて 4 × 4 のカメラ逆行列 K^{-1} も求めた．

続いて，手順 3 の上空視点カメラの外部パラメータ行列 P_T を求める．

本研究では上空視点カメラを基準としているので，上空視点カメラの姿勢を表す回転ベクトルは $(0, 0, 0)$ となる．よって，これを回転行列 R に変換すると 3 × 3 の単位行列となる．上空視点カメラの位置を表す位置ベクトル t は，$(0, 0, -1000)$ としている．

回転行列 R と位置ベクトル t を合成することで，4 × 4 の外部パラメータ行列 P_T を求めた．

続いて，手順 4 の斜め視点カメラの外部パラメータ行列の逆行列 P_C^{-1} を求める．

図 2 の θ が 45 になる位置で道路画像を撮影した．これは斜め視点カメラの姿勢が上空視点カメラの姿勢から x 軸を 45° 回転した位置になる．よって斜め視点カメラの姿勢を表す回転ベクトルは $(-\pi/4, 0, 0)$ となる．これを cv2.Rodrigues() 関数を用いて 3 × 3 の回転行列 R に変換する．

斜め視点カメラの位置ベクトル t は，2 つのカメラの位置関係から $(0, 707, -707)$ となる．回転行列 R と位置ベクトル t を合成することで 4 × 4 の外部パラメータ行列の行列 P_C が求まる．

linalg.inv() 関数を用いて，外部パラメータ行列の逆行列 P_C^{-1} を求めた．

続いて，手順 5 の平行移動行列 T を求める．

OpenCV では画像の座標を (x, y) としたとき，画像の左上が原点となり，原点から右方向が x 軸の正方向，下方向が y 軸の正方向になる．x 軸と y 軸それぞれ移動量を調整することで，俯瞰画像を出力画面内に収める．

今回は，$x = 2000$，$y = 3000$ として，平行移動行列 T を求めた．

最後に，手順 6 の式 12 で射影変換行列を計算し，道路画像を cv2.warpPerspective() 関数を用いて射影変換を行った．図 8 が道路画像を射影変換した俯瞰画像である．

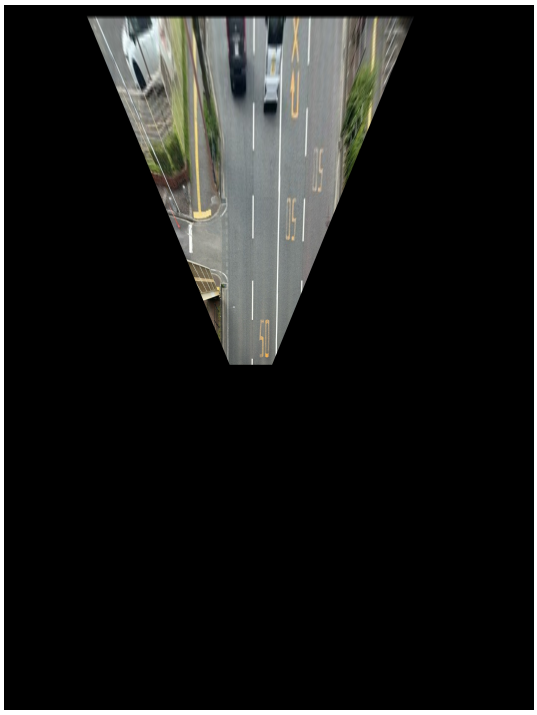

図 8 俯瞰画像

3. 俯瞰画像と航空写真のマッチング

この章では，2.4 で変換した俯瞰画像（図 8）を用いて，俯瞰画像と航空写真のマッチングを行うための手法関する説明を行う．

マッチングにはテンプレートマッチングを用いて，マッチング精度を比較するために 3 つの手法を検討した．

3.1 テンプレートマッチング

テンプレートマッチングとは，図 9 のように入力画像の中からテンプレート画像と類似する場所を探索する手法である．

類似する場所を探索する方法は，テンプレート画像と同じサイズの検索窓を，入力画像の左上から 1px ずつスライドしていく．1px ずつスライドするごとに，「入力画像内の検索窓の範囲」と「テンプレート画像」の各位置での類似度を計算していく．

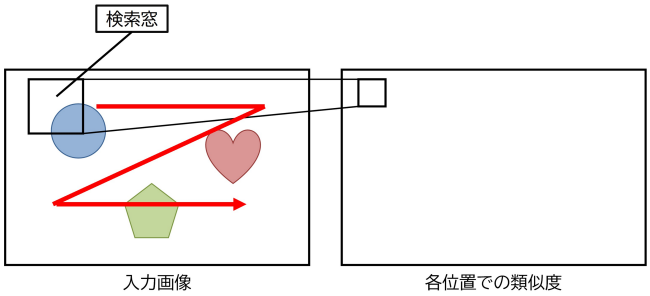

図 9 テンプレートマッチングの探索方法

テンプレートマッチングには，Python の OpenCV ライブラリに含まれる，cv2.matchTemplate() 関数を用いた．類似度の計算方法を指定する method は，cv2.TM_CCOEFF_NORMED（正規化相関係数）を用いて計算した．出力範囲は [-1,1].

- cv2.matchTemplate()[9]
  image - 入力画像
  templ - テンプレート画像
  method - 類似度の計算方法

3.2 テンプレート画像の作成

航空写真とマッチングするためのテンプレート画像を，図 10 に示す手順で作成する．

(1) 2.4 で変換した俯瞰画像（図 8）から，赤枠部分で切り取る
(2) 航空写真のマッチングしたい範囲に合わせてサイズを調整

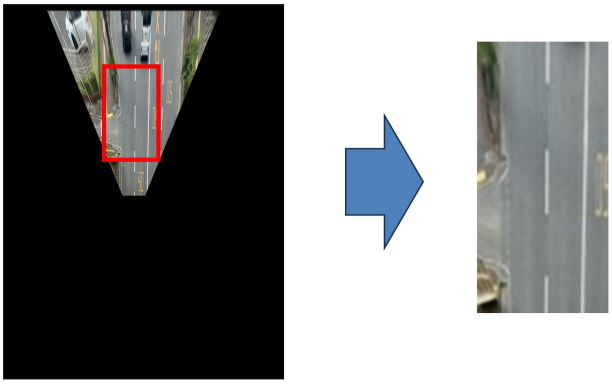

図 10 テンプレート画像の作成手順

3.3 明るさの影響を考慮したマッチング手法

図 11 の航空写真と，作成したテンプレート画像でテンプレートマッチングを行うための手法を説明する．

3.3.1 では BGR 色空間でのマッチング，3.3.2 では HSV 色空間でのマッチング，3.3.3 では明度に統一の固定値を設定した HSV 色空間でのマッチングを検討した．

図 11 航空写真

3.3.1 BGR 色空間

BGR 色空間とは，画像を青 (Blue)，緑 (Green)，赤 (Red) の数値の組み合わせで表現する方法である．画像の各ピクセルごとに (青，緑，赤) の順番で画素値が格納されている．各値は，画像の型が符号なし 8 ビットのため 0-255 の範囲で表される．

OpenCV では，cv2.imread() 関数で画像を読み込んだ場合，BGR 色空間がデフォルトとなっている．一般的に用いられる RGB 色空間と比較して，青と赤の順番が逆で格納される．

実験では以下の手順で行う．

(1) テンプレート画像と航空写真を cv2.imread() 関数で読み込む
(2) cv2.matchTemplate() 関数でテンプレートマッチングを行う

3.3.2 HSV 色空間

HSV 色空間とは，画像を色彩 (Hue)，彩度 (Satulation)，明度 (Value) の数値の組み合わせで表現する方法である．

BGR 色空間画像を HSV 色空間画像に変換する

ために，Python の OpenCV ライブラリに含まれる cv.cvtColor() 関数を使用した．この関数の引数である色空間変換コードの cv2.COLOR_BGR2HSV，または cv2.COLOR_BGR2HSV_FULL のどちらかを使うことで，HSV 色空間画像に変換できる．どちらの色空間変換コードを使っても，彩度と明度は 0-255 の範囲で表現されるのは共通である．cv2.COLOR_BGR2HSV で変換される HSV 色空間の色相は 0-179 の範囲，それに対して cv2.COLOR_BGR2HSV_FULL で変換される HSV 色空間の色相は 0-255 で表現される違いがある．

- cv.cvtColor()[11]
  src - 8or16 ビット符号なし整数，または単精度浮動小数型の入力画像
  dst - srt と同じサイズ，同じ型の出力画像
  code - 色空間変換コード

本研究では，cv2.COLOR_BGR2HSV の色空間変換コードを用いて HSV 色空間画像に変換した．

実験では以下の手順で行う．

(1) テンプレート画像と航空写真を cv2.imread() 関数で読み込む
(2) 色空間変換コード cv2.COLOR_BGR2HSV を使用した cv.cvtColor() 関数で，2 つの画像を HSV 色空間画像に変換
(3) cv2.matchTemplate() 関数でテンプレートマッチングを行う

3.3.3 明度 (Value) に統一の固定値を設定した HSV 色空間

テンプレート画像と航空写真を HSV 色空間画像に変換するまでの手順は同じである．ここでは色彩，彩度，明度のうち，テンプレート画像と航空写真両方の，明度が格納されている部分に統一の固定値を設定する．そのため，色彩と彩度によるテンプレートマッチングを行うことになる．

実験では以下の手順で行う．

(1) テンプレート画像と航空写真を cv2.imread() 関数で読み込む
(2) 色空間変換コード cv2.COLOR_BGR2HSV を使用した cv.cvtColor() 関数で，2 つの画像を HSV 色空間画像に変換
(3) 2 つの画像の明度が格納されている部分に統一の固定値を設定
(4) cv2.matchTemplate() 関数でテンプレートマッチングを行う

3.3.4 課題

テンプレート画像を作成する手順において，俯瞰画像からのテンプレート画像範囲の切り取り，テンプレート画像のサイズ調整など全てが手作業であり，改善の余地がある．

切り取る範囲指定とサイズ調整の自動化，またはサイズ調整を必要としない手法の考案が課題である．

4. 実験

4.1 実験内容

第 3 章で説明したテンプレート画像の作成手順と 3 つのマッチング手法を用いて，それぞれの手法のマッチング精度の比較を行う．4.2 では俯瞰画像と航空写真のマッチング精度の比較，4.3 ではマッチング手法ごとの類似度ヒートマップの比較を行った．

4.2 俯瞰画像と航空写真のマッチング精度の比較

第 2 章で変換した俯瞰画像（図 8）から，異なる範囲（道路範囲のみ，道路横を含めた場合，車両が映り込んだ場合 etc.）のテンプレート画像を 10 枚作成した．これを用いて 3 つの手法のマッチング精度を比較する．

4.2.1 実験

作成したテンプレート画像 10 枚を，以下の 3 つの手法で図 11 の航空写真とテンプレートマッチングを行った．

- BGR 色空間
- HSV 色空間
- 明度 (Value) に統一の固定値を設定した HSV 色空間
  Value = 100 を設定

図 11 の航空写真上の正解位置で検出できた枚数で精度を比較する．

4.2.2 結果

表 1 の結果より，BGR 色空間では 10 枚中 3 枚，HSV 色空間では 10 枚中 6 枚，HSV 色空間（明度固定）では 10 枚中 5 枚，正確に検出できた．

4.3 マッチング手法ごとの類似度ヒートマップの比較

3.1 で説明したようにテンプレートマッチングでは，入力画像とテンプレート画像の各位置での類似度が計算される．この類似度のヒートマップを 3 つのマッチング手法ごとに生成し比較する．

4.3.1 結果

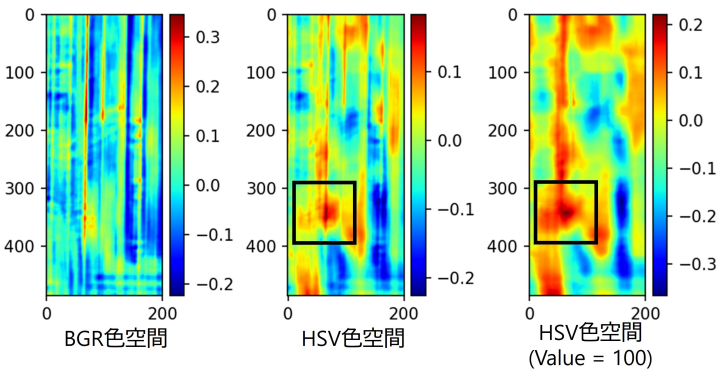

図 12 類似度ヒートマップ

図 12 の結果より，HSV 色空間同士のヒートマップを比

表 1 マッチング結果

テンプレート画像	1	2	3	4	5	6	7	8	9	10	正解枚数
BGR 色空間	○	×	○	×	×	○	×	×	×	×	3
HSV 色空間	○	○	×	×	×	○	○	○	○	×	6
HSV 色空間（明度固定）	×	○	×	○	○	×	○	○	×	×	5

較した．黒枠で囲まれた範囲の最も類似度の高い位置とその周辺を比較して，明度を固定しなかった HSV 色空間の方が類似度の高い位置が際立っている．

4.4 考察

4.2，4.3 から得られた実験結果から，HSV 色空間の 2 つの手法は，明度の固定の有無に関わらず BGR 色空間よりマッチング精度が優れていると言える．しかし，HSV 色空間のそれぞれの正解枚数は，10 枚中 6 枚，10 枚中 5 枚と高いとは言えないので，マッチング精度を高める必要があるだろう．

HSV 色空間同士のヒートマップを比較すると，明度固定なしの最も類似度の高い位置が，その周辺と比較して類似度の高い位置が際立っていた．よって，明度を固定しない方がマッチング精度が優れていると考えた．

これらの結果をまとめて，本研究で検討した 3 つのマッチング手法の中では，明度を固定しない HSV 色空間の手法のマッチング精度が優れていると考えた．

5. おわりに

5.1 結論・まとめ

本研究では，斜め視点から撮影した道路画像を俯瞰画像に変換し，航空写真と俯瞰画像のマッチング手法の検討を行った．実験の結果から，明度を固定しない HSV 色空間でのマッチングが優れていると判断した．

5.2 今後の課題

本研究では大きく分けて，道路画像の射影変換と，俯瞰画像と航空写真のマッチングを行った．

道路画像の撮影では，本来の手順ではミラー曲面のキャリブレーションが必要だが，条件を単純化するために実装していない．そしてテンプレート画像の作成手順では，全ての手順が手作業であり，改善の余地がある．

カーブミラー内の道路を俯瞰画像へ変換，地図とのマッチングを行うまでの全体の手順の内，キャリブレーションの実装や手作業部分の自動化が課題である．

謝辞 研究ならびに生活面においてご指導を賜りました福岡大学工学部電子情報工学科小野晋太郎准教授に心より感謝致します．また，本研究の遂行にあたり，有益な御助言を賜りました栗達助教に謹んで感謝の意を表します．同学部同期学生の皆様，並びに私を見守って頂いた家族に深く感謝致します．

参考文献

[1] 宮柱太一, 羽倉輝, 栗達, 河合由起子, 小野晋太郎. (2023) 車載カメラ画像からカーブミラーを検出する深層学習モデルの性能比較. 第 85 回全国大会講演論文集, 2023(1), 259-260.
[2] 百崎友哉. (2024) 安全運転支援のためのカーブミラーの向き推定. 卒業論文.
[3] 島崎敢, 中村愛, 高橋明子, 石田敏郎.(2013). カーブミラーを利用した交差車両の距離認知. 交通心理学研究. 29(1). 25-31.
[4] 森みどり, 久保登, 堀野定雄. (2011). 実路テストコース実車実験に基づく交差点カーブミラーの視認性評価. 日本人間工学会第 52 回大会. 47(特別号). 112-113.
[5] 森みどり, 堀野定雄, 久保登, 西村洋, 野口絵理香. (2010). 3DCG シミュレーション法による交差点カーブミラーの視認性評価. 日本人間工学会第 51 回大会. 46(sp). 344-345.
[6] 久保登, 森みどり, 堀野定雄. (2010). 鏡像シミュレーションによる交差点カーブミラー視認性向上のための設置条件検討 (機械要素，潤滑，設計，生産加工，生産システムなど). 日本機械学会論文集 (C 編). 76(768). 2154-2159.
[7] 小野晋太郎, 日野裕介, 須田義大, 板垣紀章. (2022). 走行中の車載カメラとカーブミラーによる死角の危険予知. 生産研究. 74(1). 123-128.
[8] 画像の幾何学変換 - opencv v2.1 documentation. http://opencv.jp/opencv-2.1/cpp/geometric_image_transformations.html (2025/01/27 閲覧)
[9] Object Detection - OpenCV. https://docs.opencv.org/4.x/df/dfb/group__imgproc__object.html (2025/01/27 閲覧)
[10] OpenCV - matchTemplate でテンプレートマッチングを行う方法. https://pystyle.info/opencv-template-matching/#index_id3 (2025/01/28 閲覧)
[11] その他の画像変換 - opencv v2.1 documentation. http://opencv.jp/opencv-2.1/cpp/miscellaneous_image_transformations.html (2025/01/27 閲覧)