

# 囲碁における盤面識別システムの検討

橋本 燐 (指導教員 井上 優良)

令和3年1月13日

## A Study of System for Extracting Go Stone Positions

RYO HASHIMOTO (ACADEMIC ADVISOR YUSUKE INOUE)

**概要：**本研究では、コンピュータと現実での囲碁の対局場面との関係をより親密なものにするために、囲碁盤の画像から碁石の配置を識別するシステムを検討する。

システムは、指定した四隅の座標をもとに碁盤を切り抜き、グレースケール変換を行った後に、石を設置できる座標の色情報から石の検出を行う。

**キーワード：**画像処理、OpenCV、射影変換

### 1. 緒言

囲碁は、2人のプレイヤーが、「碁石」と呼ばれる白黒の石を、通常  $19 \times 19$  の格子が描かれた「碁盤」と呼ばれる板へ交互に配置するボードゲームである。他のボードゲームと比較すると、ルール上の制約が極めて少ないといった特徴を持ち、他のボードゲームよりも可能な局面の数は膨大になる（約  $2.081681994 \times 10^{170}$  通り [1]）。

日本棋院によると、2014年時点での、国内における囲碁人口は推定350万人、世界の囲碁人口は推定4000万人とされている。

コンピュータの世界においては、囲碁の研究は1960年代から行われていた。最も古い研究は、1962年のRemusによる研究 “Simulation of a Learning Machine for Playing GO”[2] とされている。1969年には、Zobristが世界で初めて囲碁が動作するコンピュータプログラムを作った[3]。2005年には、モンテカルロ木探索を実装した囲碁プログラム “Crazy Stone”[4] が様々な大会で優秀な成績を記録し、以降のコンピュータ囲碁プログラムでも同様のアルゴリズムを採用するなど、コンピュータ囲碁の研究開発に大きな進歩をもたらした。2008年には、モンテカルロ木探索を採用した囲碁プログラム “MoGo” が初めて、公の場でプロ棋士に対して9路盤（ $9 \times 9$  の囲碁盤で行われる対局）で勝利を収めた[5]。

近年では、2015年にGoogle DeepMind社が開発した、ディープラーニングを実装した囲碁プログラム “AlphaGo”[6] が、初めてプロ棋士に対しハンデ無しの対局で勝利した。コンピュータが人間に打ち勝つのは難しいとされていた分野で勝利を果たしたことは、人工知能の有用性を広く知らしめるものとなった。以降にも様々な人工知能が登場し、これらとプロ棋士との対局が行われてきたが、これらの対局の様子を見ると、プロ棋士の置いた石の配置をコンピュータに入力する役割を持った人

間が存在していた。

そこで本研究では、コンピュータと現実での囲碁の対局をより親密なものにするために、対局中の囲碁の画像から碁石の配置を識別するシステムの検討を行う。具体的には、盤面を含む画像から射影変換を用いて盤面を切り抜き、ノイズ処理を適用して盤面上の線を曖昧なものにした後に、輝度値をもとに碁石の配置を識別するシステムの構築を行う。その後、複数の画像に対してシステムを試し、結果を考察する。

本論文の構成は次の通りである。第2章では先行研究についての説明を行う。第3章ではシステムで用いた射影変換とメディアンフィルタ、しきい値の評価で用いた感度と特異度についての説明を行う。第4章ではシステムが行う処理についての説明、具体的には射影変換を利用した画像の変形、メディアンフィルタを用いたノイズ除去、石を置くことができる座標の色情報をもとにした碁石の検出を行う。第5章では複数の画像に対してシステムを適用し、結果を考察する。最後に第6章で本論文をまとめ、今後の課題について述べる。

### 2. 先行研究

棋譜の自動生成に関する既存研究に着目すると、1手前の着手と比較することで碁石の配置を検出する研究がある。芝らは、着手ごとのグレイ画像と1手前のグレイ画像との、碁盤領域内の差分をとることで、碁盤上の碁石の位置（碁石座標）を検出する手法を提案した[7]。しかし、この手法では、途中から対局を記録しようとした時に、そのターンを記録できない問題が存在する。

そこで本研究では、記録しようとした時点のターンから記録できるように、直前の着手の情報を使用せず、1つの着手の画像だけで碁石の位置を検出することを目的とする。

### 3. 理論

#### 3.1 変換

##### 3.1.1 同次座標

座標  $(x, y)$  に対し、その要素を 1 つ増やした座標  $(\xi_1, \xi_2, \xi_3)$  を、以下の関係式を満たすように定義する。

$$\begin{aligned} x &= \frac{\xi_1}{\xi_3} \\ y &= \frac{\xi_2}{\xi_3} \end{aligned} \quad (3.1.1)$$

ただし、 $\xi_1, \xi_2, \xi_3$  のうち、少なくとも 1 つは 0 ではないとする。このように定義される座標を同次座標と呼ぶ [8]。

同次座標においては、 $\lambda \neq 0$  なる任意の  $\lambda$  に対して、 $(\xi_1, \xi_2, \xi_3)$  と  $(\lambda\xi_1, \lambda\xi_2, \lambda\xi_3)$  は、通常の座標に直したとき、ともに  $(\xi_1/\xi_3, \xi_2/\xi_3)$  となるため、同じ点を表している。つまり、同次座標による表現では、定数倍をしても変わらないとみなすことができる。このような表現を同値であるとよび、これを式では以下のように表す。

$$\begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix} \sim \begin{pmatrix} \lambda\xi_1 \\ \lambda\xi_2 \\ \lambda\xi_3 \end{pmatrix} \quad (3.1.2)$$

ここで、記号  $\sim$  が同値関係を表し、定数倍の違いを許して等しいことを意味する。

##### 3.1.2 射影変換

同次座標を利用することにより、一般的な変換を表現することができる。これは以下の式で表現されるもので、射影変換と呼ばれている [8]。

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \sim \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad (3.1.3)$$

あるいは、これをベクトルと行列の記号を用いて、以下のように表現することもできる。

$$\vec{x}' \sim H \vec{x} \quad (3.1.4)$$

$H$  は任意の  $3 \times 3$  の行列である。これを変換行列という。

式 (3.1.1) の関係を用いて、(3.1.3) から座標  $(x', y')$  を求めると以下のように表すことができる。

$$\begin{aligned} x' &= \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}} \\ y' &= \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}} \end{aligned} \quad (3.1.5)$$

一意に射影変換の変換行列を表すため、行列のどれか 1 つの値を固定する。 $h_{33} = 1$  とし、(3.1.5) に代入、式変形を行うと  $(x', y')$  は以下のように表すことができる。

$$\begin{aligned} xh_{11} + yh_{12} + h_{13} - xx'h_{31} - yx'h_{32} &= x' \\ xh_{21} + yh_{22} + h_{23} - xy'h_{31} - yy'h_{32} &= y' \end{aligned} \quad (3.1.6)$$

(3.1.6) をベクトル演算で表記すると、以下のように表記できる。

$$\begin{aligned} (x, y, 1, 0, 0, 0, -xx', -yx')(h_{11}, h_{12}, \dots, h_{31}, h_{32})^T \\ (0, 0, 0, x, y, 1, -xy', -yy')(h_{11}, h_{12}, \dots, h_{31}, h_{32})^T \end{aligned} \quad (3.1.7)$$

2 つの四角形の頂点座標をそれぞれ  $((x_{11}, y_{11}), (x_{12}, y_{12}), (x_{13}, y_{13}), (x_{14}, y_{14}))$ 、 $((x_{21}, y_{21}), (x_{22}, y_{22}), (x_{23}, y_{23}), (x_{24}, y_{24}))$  とする、すべての頂点を (3.1.7) で表すと以下の式になる。

$$\begin{pmatrix} x_{11}, y_{11}, 1, 0, 0, 0, -x_{11}x_{21}, -y_{11}x_{21} \\ 0, 0, 0, x_{11}, y_{11}, 1, -x_{11}y_{21}, -y_{11}y_{21} \\ x_{12}, y_{12}, 1, 0, 0, 0, -x_{12}x_{22}, -y_{12}x_{22} \\ 0, 0, 0, x_{12}, y_{12}, 1, -x_{12}y_{22}, -y_{12}y_{22} \\ x_{13}, y_{13}, 1, 0, 0, 0, -x_{13}x_{23}, -y_{13}x_{23} \\ 0, 0, 0, x_{13}, y_{13}, 1, -x_{13}y_{23}, -y_{13}y_{23} \\ x_{14}, y_{14}, 1, 0, 0, 0, -x_{14}x_{24}, -y_{14}x_{24} \\ 0, 0, 0, x_{14}, y_{14}, 1, -x_{14}y_{24}, -y_{14}y_{24} \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = \begin{pmatrix} x_{21} \\ y_{21} \\ x_{22} \\ y_{22} \\ x_{23} \\ y_{23} \\ x_{24} \\ y_{24} \end{pmatrix} \quad (3.1.8)$$

よって、(3.1.8) の左辺にある  $8 \times 8$  行列の逆行列を計算することで、射影変換の変換行列の各要素が求まる [9]。

$$\begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = \begin{pmatrix} x_{11}, y_{11}, 1, 0, 0, 0, -x_{11}x_{21}, -y_{11}x_{21} \\ 0, 0, 0, x_{11}, y_{11}, 1, -x_{11}y_{21}, -y_{11}y_{21} \\ x_{12}, y_{12}, 1, 0, 0, 0, -x_{12}x_{22}, -y_{12}x_{22} \\ 0, 0, 0, x_{12}, y_{12}, 1, -x_{12}y_{22}, -y_{12}y_{22} \\ x_{13}, y_{13}, 1, 0, 0, 0, -x_{13}x_{23}, -y_{13}x_{23} \\ 0, 0, 0, x_{13}, y_{13}, 1, -x_{13}y_{23}, -y_{13}y_{23} \\ x_{14}, y_{14}, 1, 0, 0, 0, -x_{14}x_{24}, -y_{14}x_{24} \\ 0, 0, 0, x_{14}, y_{14}, 1, -x_{14}y_{24}, -y_{14}y_{24} \end{pmatrix}^{-1} \begin{pmatrix} x_{21} \\ y_{21} \\ x_{22} \\ y_{22} \\ x_{23} \\ y_{23} \\ x_{24} \\ y_{24} \end{pmatrix} \quad (3.1.9)$$

射影変換においては、線分の直線性は保たれるものの、平行性は失われる。別の言い方をすると、任意の四角形を別の任意の四角形に移すような変換であるといえる。

#### 3.2 空間フィルタ

入力画像の対応する画素値だけでなく、その周囲の画素も含めた領域内の画素値を用いて計算する処理のことを空間フィルタリングといい、この処理で用いるフィルタを空間フィルタといいう [8]。

空間フィルタは、線形フィルタと非線形フィルタの 2 つに分類できる。

線形フィルタは、入力画像を  $f(i, j)$ 、出力画像を  $g(i, j)$  とするとき、以下の式で表すことができるフィルタのことといいう。

0	1	1
2	3 (median)	5
8	13	21



図 3.1 メディアンフィルタの例

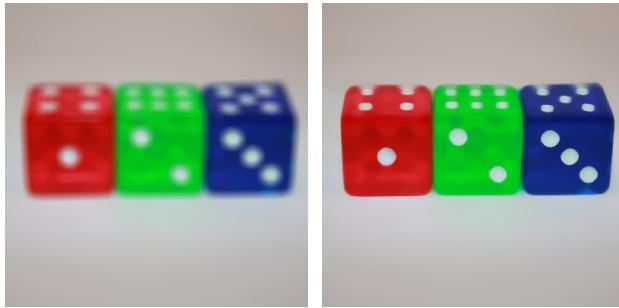


図 3.2 平均化フィルタ

図 3.3 メディアンフィルタ

$$g(i, j) = \sum_{n=-W}^W \sum_{m=-W}^W f(i+m, j+n) h(m, n) \quad (3.2.10)$$

$h(m, n)$  はフィルタの係数を表す配列であり , フィルタの大きさは  $(2W + 1) \times (2W + 1)$  である .

非線形フィルタは , (3.2.10) 式に当てはまらない処理を伴うフィルタである .

### 3.3 メディアンフィルタ

フィルタ範囲内のピクセルの画素値を昇順か降順に並び替え , 中央値を出力するフィルタをメディアンフィルタと呼ぶ [8] . これは非線形フィルタに属する .  $3 \times 3$  の画素に対しメディアンフィルタを適用した際の例を図 3.1 に示す .

このフィルタは , コントラストの差がある輪郭部分がぼやけにくい特徴を持つ . 同じ画像に対して , 領域内の画素値の平均を出力するフィルタ ( 平均化フィルタ ) とメディアンフィルタを施した例を , それぞれ図 3.2 , 図 3.3 に示す .

### 3.4 感度

検査で検出したい信号や疾患有するもののうち , 検査が正しく陽性と判断したものの割合を指す . 真陽性率 ( TPF: True Positive Fraction ) とも呼ぶ .

感度は

### 3.5 特異度

検査で検出したい信号や疾患有さないもののうち ,

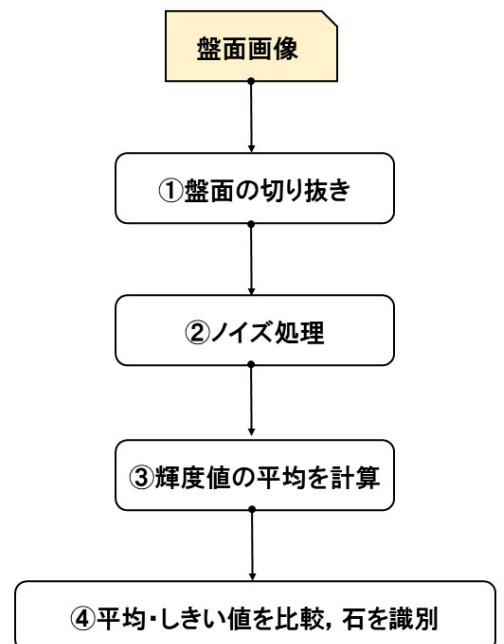


図 4.1 フロー図

検査が正しく陰性と判断したもののが割合を指す . 真陰性率 ( TNF: True Negative Fraction ) とも呼ぶ .

## 4. 盤面識別システム

本研究で検討したシステムでは , 盤面を含む画像から , 碁石の位置を識別することを目的とする .

実際には , 盤面を含む画像から盤面を切り抜き , 碁石上の線を黒石と誤検知しないようノイズ処理を施した後に  $19 \times 19$  の領域を付与する . その後 , 付与した領域内における輝度値の平均を計算し , 黒石・白石のしきい値とを比較 , 結果をもとに石の識別を行う . システムの流れを図 4.1 に示す .

システム内で行う処理には , プログラミング言語である Python と , オープンソースのコンピュータビジョン用ライブラリである OpenCV を用いる . 途中で行う射影変換の処理には , 異なる画像処理ライブラリである Pillow や scikit-image を用いても実装は可能であるが , 複雑な計算を実装する必要がある . OpenCV であればこれを計算する関数が用意されており , 処理を簡単に実装できるといった理由から OpenCV を選択した .

### 4.1 切り抜き

盤面を含む画像では , 盤面以外の部分は不要であるため , 盤面を含む画像から , 盤面以外の不要な部分を除去する . 変換前と変換後の図を図 4.2 , 図 4.3 に示す .

OpenCV を用いた射影変換には , getPerspectiveTransform() 関数と , warpPerspective() 関数を用いる . 処理の

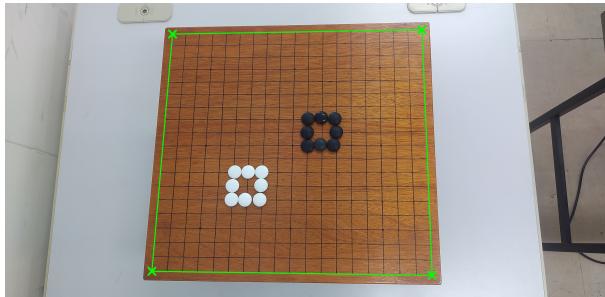


図 4.2 射影変換前

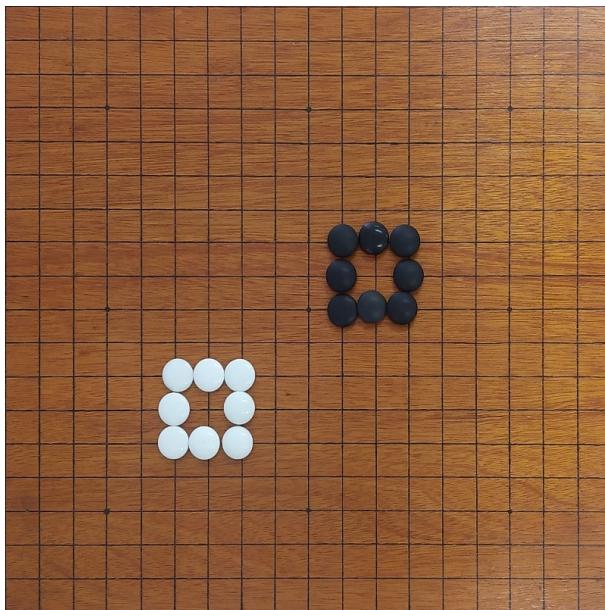


図 4.3 射影変換後

手順を以下に示す。

はじめに, `getPerspectiveTransform()` 関数に, 第一引数には変換前の四角形の頂点座標リスト, 第二引数には変換後の四角形の頂点座標リストを与えることで, 変換行列の計算を行う。ここで, 変換前の四角形の頂点座標リストとして, 盤面の四隅の座標を格納したリストを与える。盤面部分を正方形に変換するため, 変換後の四角形の頂点座標リストには, 正方形の四隅の座標を格納したリストを与える。

次に, `warpPerspective()` 関数に, 第一引数には画像情報を格納した変数, 第二引数には求めた変換行列, 第三引数には出力画像のサイズを与えることで, 画像に対する射影変換を行う。

今回の実験では, 変換行列の計算に用いる変換後の正方形の頂点と, 出力する画像のサイズに  $800 \times 800$  の正方形を与えた。

## 4.2 ノイズ除去

後述する識別の方法では, 盤面上の線が交差する点を誤って黒石と識別してしまうことがある。これを減らすため, ノイズ処理を適用することで, 盤面上の線がある

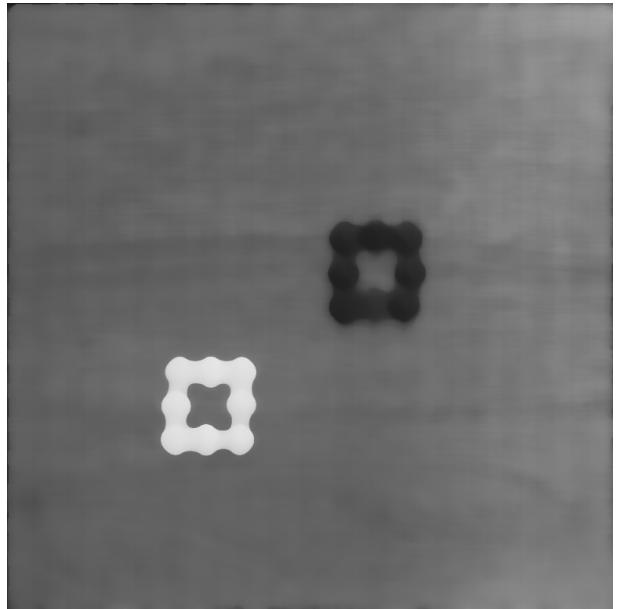


図 4.4 ノイズ除去後の盤面

程度無視できるようにする。4.1 小節で生成した画像(図 4.3)に対し, フィルタを用いてノイズ処理を行う。フィルタには, 輪郭部分がぼやけにくいといった特徴を持つメディアンフィルタを使用する。

OpenCV を用いたメディアンフィルタには, `medianBlur()` 関数に, 第一引数に画像情報を格納した変数, 第二引数にフィルタのカーネルサイズを与えることで実装できる。ここで, カーネルサイズの指定には奇数を用いる。これはメディアンフィルタの理論上, 偶数のカーネルサイズを与えると中央値が求められないためである。今回の実験では, カーネルサイズに 25 を与える。処理結果を図 4.4 に示す。

## 4.3 領域の付与

後述する識別の際に, 情報を取得するための,  $19 \times 19$  個の小さな領域を付与する。4.2 小節で生成した図(図 4.4)に対し, 領域を可視化した図を図 4.5 に示す。

## 4.4 識別

4.3 小節で与えた領域内における輝度値の平均(0~255)を取得し, 白黒それぞれのしきい値をもとに, 各領域を

- 平均値が黒石のしきい値より小さい = 「黒石がある」
- 平均値が白石のしきい値より大きい = 「白石がある」
- 「それ以外」

の 3 種類に分類することで, 碁石の識別を行う。盤面の画像(図 4.3)上に, 実際に識別した結果を重ねた図を図 4.6 に示す。

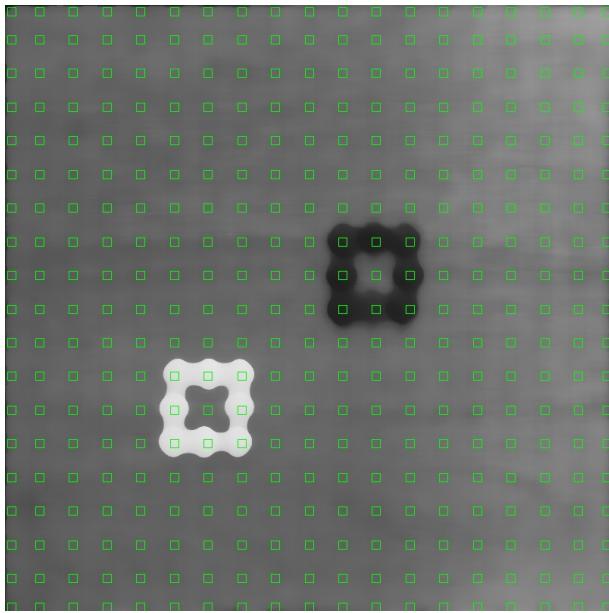


図 4.5 領域を可視化した盤面

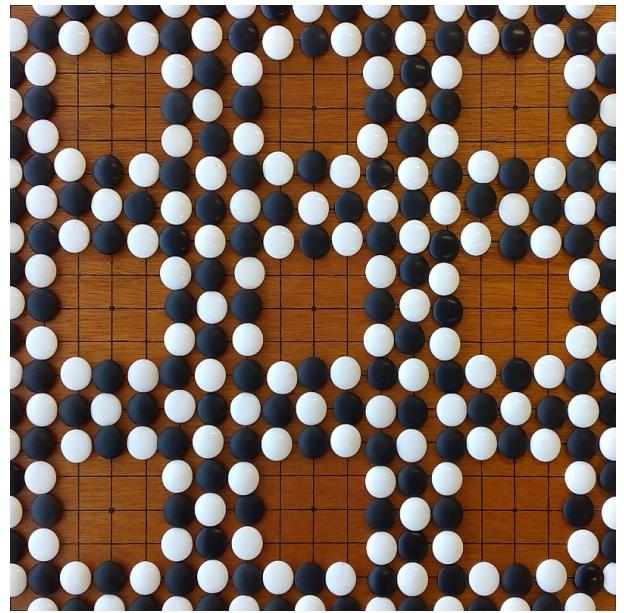


図 5.1 しきい値の決定に用いた盤面 1

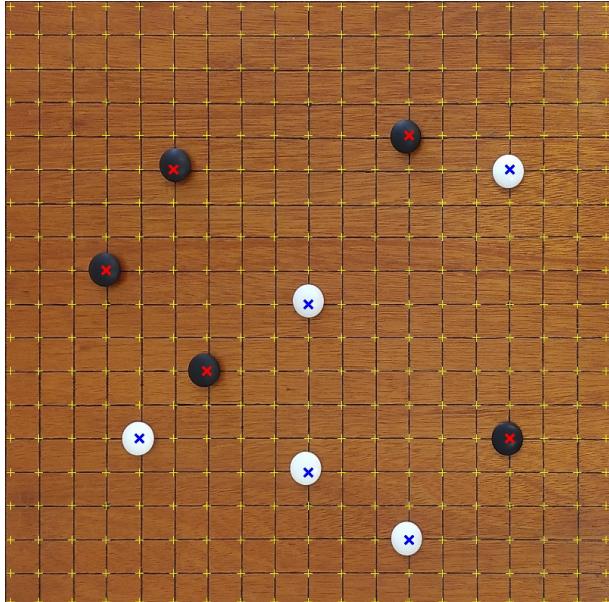


図 4.6 識別結果を重ねた盤面

## 5. 実験と考察

本章では、4章で検討したシステムの有効性を検証するため、実際に盤面を含む画像を使ってシステムを適用し、結果を考察する。

本章1節では、最適なしきい値を決定するために行った予備実験について示す。2節では、決定したしきい値のもとで、複数の事例に対し、実際にシステムで碁石の識別を行った結果について示す。

### 5.1 予備実験

どのしきい値で最も正確に識別を行えるか確認するた

めに、2枚の盤面画像（図5.1、図5.2）に対し、システムを適用して碁石の識別を行う。しきい値を変化させながらシステムを適用し、しきい値の変動に対応して変動した以下の4項目によって、最適な値を決定した。

- 白石を、白石と識別した割合（白石の感度）
- 黒石を、黒石と識別した割合（黒石の感度）
- 白石でないものを、正しく白石でないと識別した割合（白石の特異度）
- 黒石でないものを、正しく黒石でないと識別した割合（黒石の特異度）

図5.1、図5.2における白石の感度と特異度をそれぞれ図5.3と図5.4に、黒石の感度と特異度をそれぞれ図5.5と図5.6に示す。

図5.3、図5.4より、図5.1と図5.2の両方に対して、白石のしきい値は156～160の範囲で感度100%，特異度100%を記録したため、今回の実験ではその範囲の中央値である158を使用した。

図5.5と図5.6より、図5.1と図5.2の両方に対して、黒石のしきい値は52で感度100%，特異度100%を記録したため、今回の実験では52を使用した。

### 5.2 実際の識別結果

本小節では、3件の画像に対してシステムを適用した結果と、その考察を述べる。具体的には、すべて正常に識別できた場合、石の位置がずれていることで誤った識別をした場合、反射光を白石と誤った場合について述べる。

#### 5.2.1 事例 1

この盤面画像（図5.7）に対し、システムを適用した結果、全ての碁石を正確に識別できた。

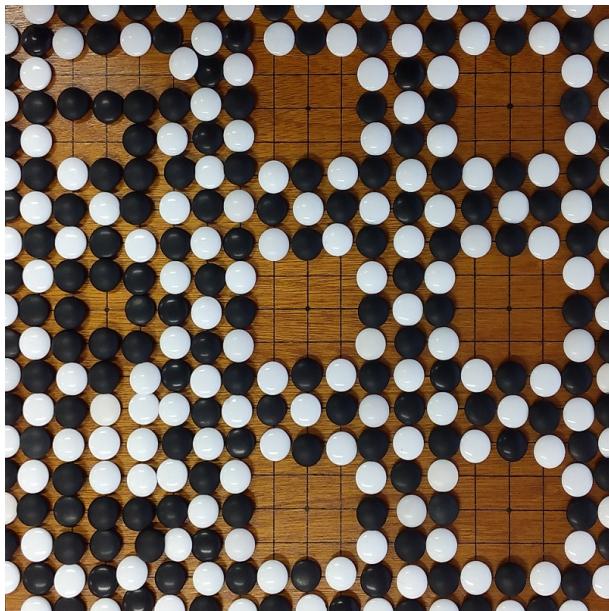


図 5.2 しきい値の決定に用いた盤面 2

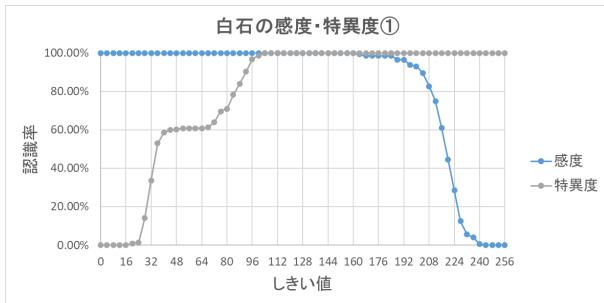


図 5.3 白石の感度・特異度 1

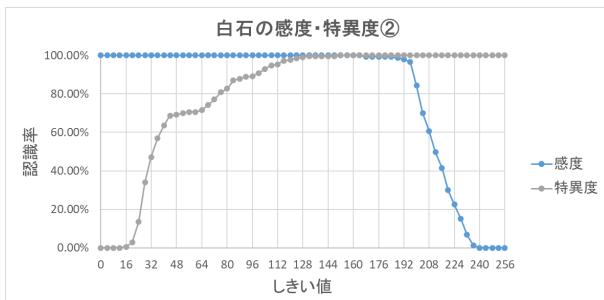


図 5.4 白石の感度・特異度 2

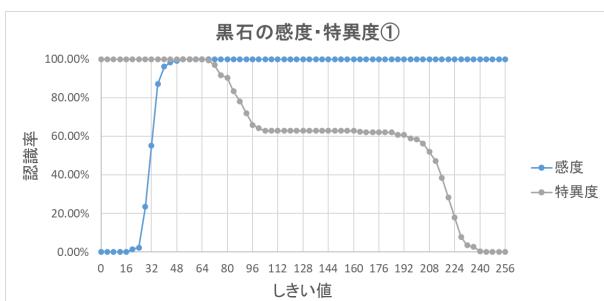


図 5.5 黒石の感度・特異度 1

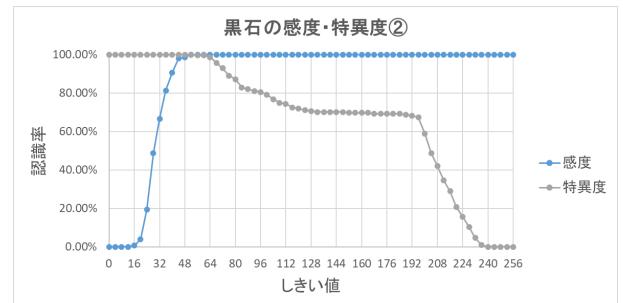


図 5.6 黒石の感度・特異度 2

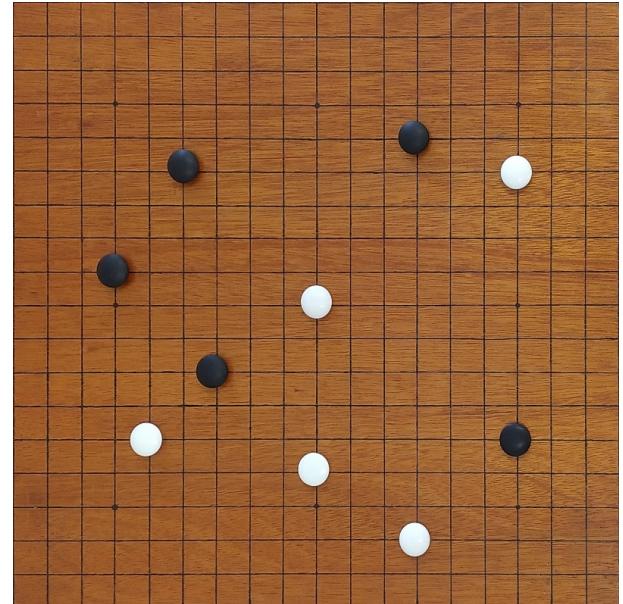


図 5.7 事例 1 の盤面

### 5.2.2 事例 2

この盤面画像（図 5.8）に対し、システムを適用した。識別結果に誤りがあった部分にマークを付け、その周囲をトリミングしたものを図 5.9 に示す。

図 4.4 と同様に、輝度値の平均を取得する領域を可視化すると、中央の石の大部分が領域からはみ出ているのが分かる。このことから、石が正しく盤面の交点上に配置されていない場合、システムは正常に石を識別できないと考える。図 5.9 の範囲に対し、領域を可視化した図を図 5.10 に示す。

### 5.2.3 事例 3

この盤面画像（図 5.11）に対し、システムを適用した。識別結果に誤りがあった部分にマークを付け、その一部をトリミングしたものを図 5.12 に示す。

図 5.12 の範囲に対し、白石の識別に用いた閾値よりも暗い部分を 0、明るい部分を 1 として `inRange()` 関数で二値画像を生成すると、白石が無い部分も白石があるとみなしていることが分かる。このことから、盤面や黒石に反射光が含まれていると、システムは誤って白石と識別してしまうと考える。

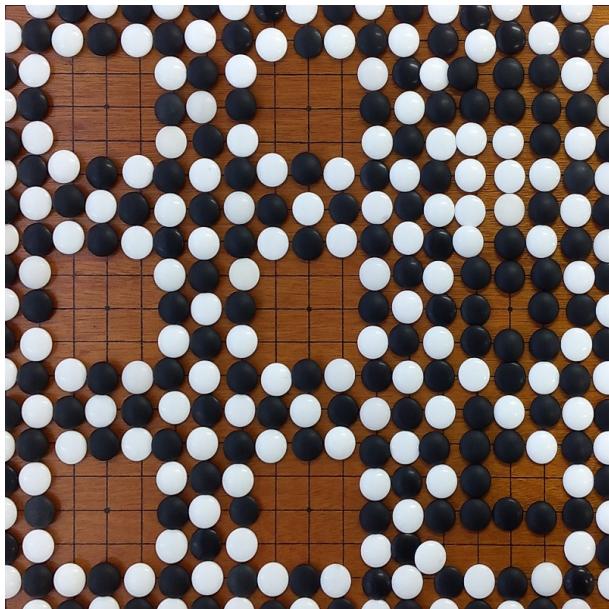


図 5.8 事例 2 の盤面

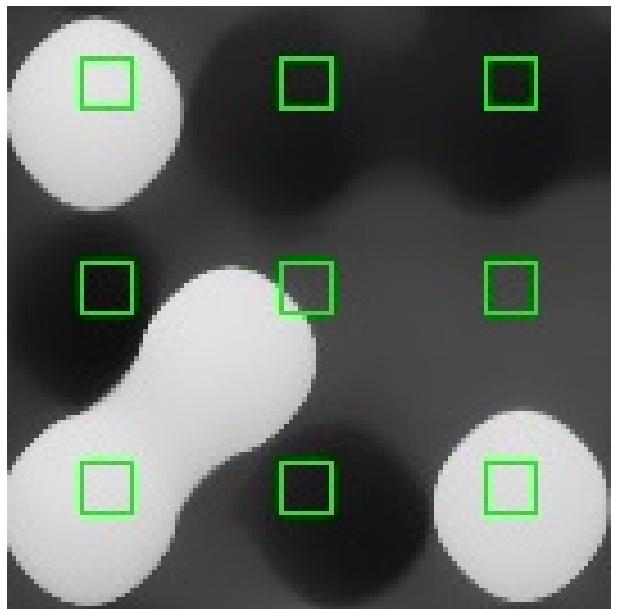


図 5.10 誤り部分の領域を可視化した図

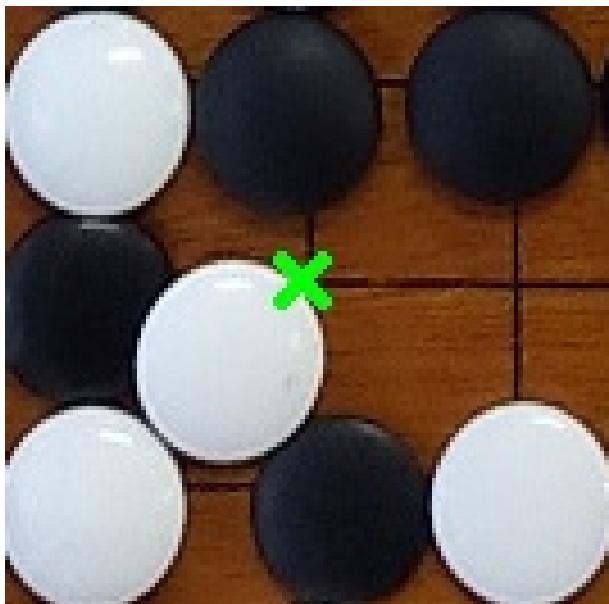


図 5.9 事例 2 の誤り部分

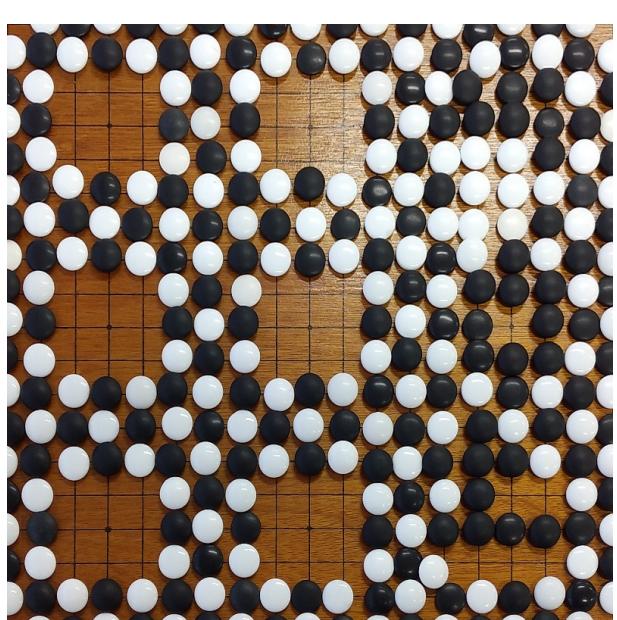


図 5.11 事例 3 の盤面

## 6. 結言

本研究では、囲碁の盤面を含んだ画像から、碁石の配置を識別することを目的として、画像上から碁石の配置を識別するシステムの検討を行い、数件の画像に対してシステムを適用して有効性を確認した。

5.2 節より、石の位置に大きなずれが無い、かつ反射光が写り込んでいない場合に限り、碁石を正常に識別することができた。石のずれに対応できない問題には、画像に付与する領域（碁石座標）の位置を等間隔ではなく、碁盤の交点や石の形状をもとに検出することでこれを解決できると考える。反射光を白石と識別してしまった場合には、しきい値を固定するのではなく、画像の状況に応

じて最適なしきい値を算出することで、誤った識別をする数を減らすことができると思われる。

今後の課題として、石の座標を検出する手法と、画像の状況に応じて適切なしきい値を算出する手法の模索が挙げられる。

## 謝辞

本研究を進めるにあたり、様々なご指導を頂きました井上優良助教に深謝いたします。また、この研究の機会をくださった情報工学科の先生方、そして多くの知識やご指摘を下さいました同級生の皆様に厚く御礼申し上げます。

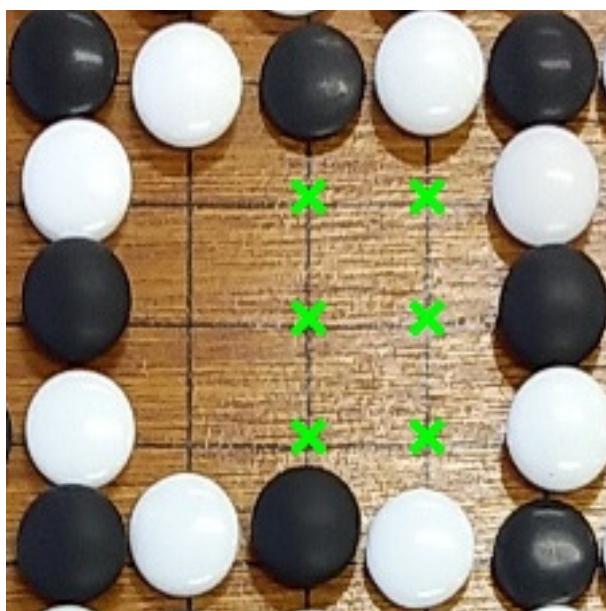


図 5.12 誤り部分の一部



図 5.13 誤り部分をしきい値で二値化

## 参考文献

- [1] John Tromp 「Number of legal Go positions」(最終閲覧日:令和3年1月13日)  
<https://tromp.github.io/go/legal.html>
- [2] Remus, H. : *Simulation of a Learning Machine for Playing Go*, Information Processing, pp. 192-194 (1962).
- [3] Zobrist, A. L.: *A Model of Visual Organisation for the Game of Go*, Proceedings of AFIPS Spring joint Computer Conference, Vol. 34, pp. 103-112 (1969).
- [4] Rémi Coulom 「Crazy Stone」(最終閲覧日:令和3年1月13日)  
<https://www.remi-coulom.fr/CrazyStone/>
- [5] 美添一樹(2008)「モンテカルロ木探索 コンピュータ囲碁に革命を起こした新手法」,『情報処理』, 49(6), pp.686-693
- [6] DeepMind 「AlphaGo — DeepMind」(最終閲覧日:令和3年1月13日)  
<https://deepmind.com/research/case-studies/alphago-the-story-so-far>

和3年1月13日)

<https://deepmind.com/research/case-studies/alphago-the-story-so-far>

[7] 芝 浩二郎・古屋 保・西 省吾・森 邦彦(2006)「画像処理による囲碁棋譜自動生成システム」,『電気学会論文誌C(電子・情報・システム部門誌)』, 126(8), pp.980-989.

[8] ディジタル画像処理 [改訂第二版], 松阪 喜幸(2020)

[9] 宮崎 大輔「四角形から四角形への変換」(最終閲覧日:令和3年1月13日)

<http://www.info.hiroshima-cu.ac.jp/~miyazaki/knowledge/tech0115.html>