1　テーマ

制御ゼミナールのレポートについて直線を検出するアルゴリズムについて調べ C++で実装した.

2　直線検出のアルゴリズム

直線検出のアルゴリズムには大きく分けて次の 2 つあると考えられる.

1. Hough 変換
2. LSD(Line Segment Detector)

2.1　Hough 変換

Hough 変換は各点を通る直線が無限個あり，それが様々な方向を向く．それらの直線の中で画像中の点を最も多く通るものを決定する．直線の式は,

$$y = ax + br = x \cdot cos(\theta) + y \cdot sin(\theta) \tag{1}$$

の 2 つがある．この時，a, b は任意の定数.x, y を座標，r を座標 (x, y) を通る直線との距離，θ を座標 (x, y) を通る直線と垂直に交わり r を通る直線と x 軸の成す角とする．また，それぞれの式は図 1,2 のように表される.

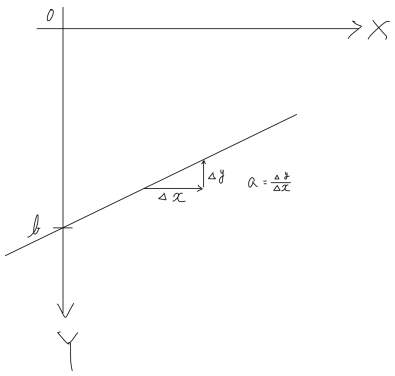

図 1: $y = ax + b$ の図

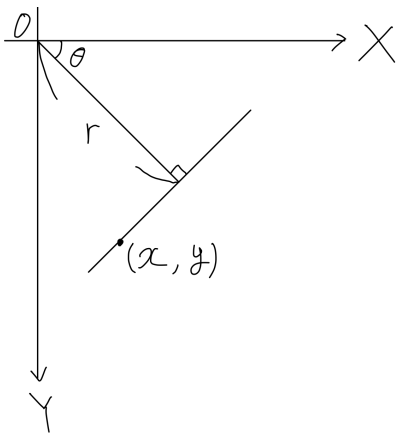

図 2: $r = x \cdot cos(\theta) + y \cdot sin(\theta)$ の図

Hough 変換では式 (1) を用いて画像内の点を通る直線を r と θ で表す．最も多く使用された r と θ のペアによって作られる直線が直線であると考えられる.

2.2 LSD(Line Segment Detector)

LSD は画像上の局所的な直線の輪郭を検出することを目的としている．輪郭とは明暗の変化が十分に早く変化している場所を指している．

3 実装

今回は Hough 変換を実装した．入力はすでに 2 値化された ASCII PBM ファイルを使用し，出力は最も多く使用された r と θ のペアとする．

3.1 ソースコード

ヘッダファイルをソースコード 1 に，Hough 変換の主要部をソースコード 2 に示す．

ソースコード 1: header

```
#include <fstream>
#include <iostream>
#include <cmath>
#include <sstream>
#include <string>

using namespace std;

#define SITA 180
```

ソースコード 2: Hough Transration

```
  int row = ceil(sqrt(xmax * xmax + ymax * ymax));
  int k = 0, distance = 0, rmax, sitamax;
  int** rsita;

  rsita = new int*[row];
  for (i = 0; i < row; i++) {
    rsita[i] = new int[SITA];
    // cout << i << endl;
  }

  for (j = 0; j < row; j++) {
    for (i = 0; i < SITA; i++) {
      rsita[j][i] = 0;
    }
  }

```

```
17  for (j = 0; j < ymax; j++) {
18    for (i = 0; i < xmax; i++) {
19      if (matrix[j][i] == 1) {
20        for (k = 0; k < SITA; k++) {
21          distance = round(i * cos(k * M_PI / 180) + j * sin
                (k * M_PI / 180));
22          if (distance < 0) {
23            distance *= -1;
24          }
25          rsita[distance][k] += 1;
26        }
27      }
28    }
29  }
30
31  int max_rsita = 0;
32
33  for (j = 0; j < row; j++) {
34    for (i = 0; i < SITA; i++) {
35      cout << rsita[j][i] << "␣";
36      if (max_rsita < rsita[j][i]) {
37        max_rsita = rsita[j][i];
38      }
39    }
40    cout << endl;
41  }
42
43  for (j = 0; j < row; j++) {
44    for (i = 0; i < SITA; i++) {
45      if (max_rsita == rsita[j][i]) {
46        cout << "r␣=␣" << j << ",␣θ␣=␣" << i << "°
              " << endl;
47      }
48    }
49  }
```

このとき，ymax と xmax はそれぞれ画像の縦と横の最大の値である．

3.2 評価方法

このソースコードを評価するために図 3 から 6 を用意した．

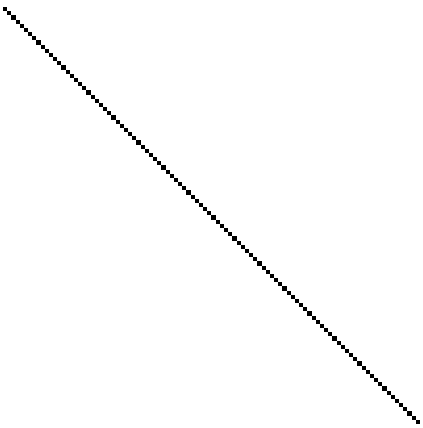

図 3: $y = x$ の直線のある図

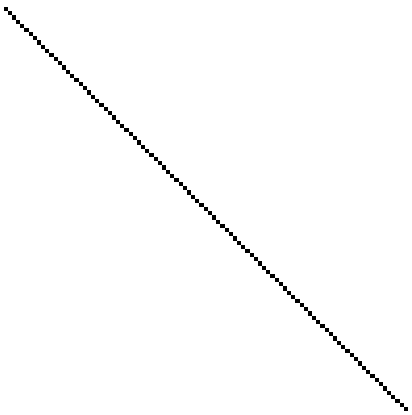

図 4: $y = x - 3$ の直線のある図

図 5: $y = 50$ の直線のある図

図 6: $x = 50$ の直線のある図

画像はすべて 100×100 の大きさである．これらの画像を自作の Hough 変換のソースコードに入力し，正しい値が出ているか評価する．

3.3 結果

図 3 から 6 の結果を表 1 に示す.

表 1: 画像から検出した直線のパラメータ

	r	θ °
図 3	0	135
図 4	2	135
図 5	50	90
図 6	50	0

このように出てきている値が概ね問題ないため，ソースコード 2 は正しいものであると言える．

4　考察・感想

4.1　考察

今回作成した Hough 変換では，$O(N^3)$ になるため計算量が大変多くなる．また，最も多くの点を含んでいる直線のみしか検出していないため，画像に一本長い線が入ってしまうとそれしか表示されなくなってしまう．そのため完璧なものとは言い難い．

これらの改善点として次のようなものが挙げられる．

1. 確率モデルを用いて計算する点の量を減らす．
2. 上位数%までの直線は表示する．

またこれ以外にも，もう一つの直線検出方法である LSD を用いてみると良いのではないかと考える．

4.2　感想

直線を検出するアルゴリズムについて調査し，C++で実装を行った．結果として C++に対する理解と直線検出のアルゴリズムが深まったので良かった．ただ，当初考えていた直線検出のアルゴリズムとやや違う形になってしまったので，力不足で実装する時間を十分に作れなかったことを反省したい．

参考文献

1. 2022 資格情報処理研究室ゼミ資料
2. ハフ変換と LSD による直線検出の比較
   (https://data-analysis-stats.jp/python/ハフ変換と lsd による直線検出の比較/)
3. C++の基礎: 数学関数の使い方を学ぶ
   (https://docwiki.embarcadero.com/Support/ja/C++の基礎: 数学関数の使い方を学ぶ)