情報理工学特別研究報告書

題目

NDT Scan Matching による 3 次元点群の位置合わせに関する研究

学生証番号　953229

氏名　梅原　綜馬

提出日　令和 5 年 1 月 26 日

指導教員　蚊野　浩

京都産業大学
情報理工学部

要約

3 次元点群の位置合わせは，自動車やロボットの自動運転で，地図に対する自己位置の推定に利用される．点群マッチングアルゴリズムの NDT Scan Matching は，計算を高速化するために，地図側の点群を格子状のグリッドセルで分割しグリッドセル内部の点群をガウス分布で近似する．そして，ガウス分布で近似した参照点群に入力点群をマッチングさせる．NDT は C++と MATLAB での実装が存在するだけであった．本研究では，NDT を Python で実装することで NDT の性能や性質を解明することを目的とした．

NDT の性質として評価関数にガウス分布が使用されており，評価関数を最適化する座標変換パラメータを求めることによりマッチングさせる．実用的に NDT を使用するためにガウス分布に一様分布を組み合わせた分布を新たな評価関数とし，外れ値の対策をすることがわかった．

実装した NDT では，勾配法を用いて NDT の評価関数を最適化することによって，参照点群全体を 1 つのグリッドセルとした場合のマッチングに成功した．参照点群を複数のグリッドセルに分割した場合のマッチングでは，参照点群によって失敗する場合と成功する場合があった．これは，グリッドセル内の近似されたガウス分布の形状が円状で，マッチングのための特徴量が少ないグリッドセルが存在する場合にマッチングに影響を与えていると考えた．課題として，マッチングに失敗してしまう参照点群で，ガウス分布が円状に分布しているグリッドセルに対する対策の実装が残った．

目次

1章 序論

立体物の3次元情報を,物体表面の3次元座標を離散的にサンプリングした点の集合で表現したものを3次元点群（以下，単に点群と記述）とよぶ．点群を取得するには，LiDAR（Light Detection and Ranging）[4]などの3次元計測技術を用いる．LiDARで市街地などの広域をスキャンし,その全体形状を点群で表現したものを3次元点群地図とよぶ．図1.1は3次元点群地図の例である．

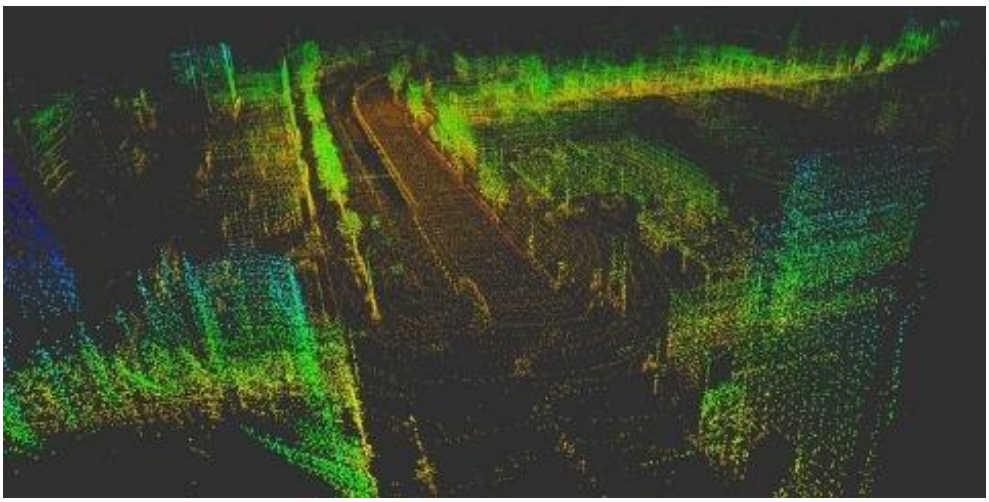


図1.1 LiDARでスキャンされた3次元点群地図

(https://www.macnica.co.jp/business/maas/columns/134751/)

3次元点群地図を用いた位置合わせ技術は，自動車やロボットの自動運転で地図に対する自己位置の推定に利用される.屋外で自己位置を求める主な手法としてGPSがあるが，GPSは精度が数mであり，トンネルなどの衛星からの電波が届かないところでは利用できない.GPSだけでは姿勢を決定することができない.などの欠点がある.そこで，3次元点群地図を用いた位置決め技術は，GPSを補う有力な方法の1つとなる.

点群地図に対する自己位置の決定は，LiDARなどを用いて現在位置を中心としてローカルな点群を取得し,それをグローバルな点群地図に対してマッチングさせることで可能になる．ここで点群と点群のマッチングが必要になる．点群マッチングのアルゴリズムの代表例としてICP(Iterative Closest Point)がある[3]．しかし，ICPは計算コストが高いため,自動運転のようなリアルタイムに処理を行う必要がある用途には適さない.

NDT(Normal Distribution Transform) Scan Matching(以下，単にNDTと記述）は，ICPよりも新しい点群マッチングアルゴリズムである[1]．マッチングを高速化するために，地図側の点群(参照点群)を格子状のグリッドセルで分割し，グリッドセルの内部の点群をガウス分布で近似する．そして，ガウス分布で近似した参照点群に入力点群を

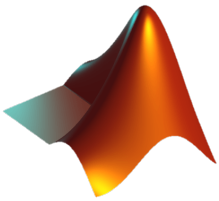
マッチングさせる．また，ICP はPCL（Point Cloud Library）[5]やOpen3D[6]などにC++とPython の実装が存在するが，NDT はPCL にC++の実装とMATLAB での実装が存在するだけであった．そこで，本研究では，NDT を Python で実装することで NDT の性能や性質を解明することを目的とした．以下，2 章ではNDT のアルゴリズムについて詳しく記述する．3 章では NDT の実装を行う．4 章では実装した NDT の評価を行い，5 章で結論を記述する．

2 章　3 次元点群の位置合わせと NDT Scan Matching

本章では点群処理ライブラリと NDT のアルゴリズムを説明する.

2.1　点群処理ライブラリ

点群処理を行うプログラミング言語・ライブラリとして，MATLAB，C++で利用できる PCL(Point Cloud Library)，C++と Python で利用できる Open3D がある(図 2.1)



(a) MATLAB

(MathWorks の Web サイトの写真を利用)

(b) Point Cloud Library

(PCL の Web サイトの写真を利用)

(c) Open3D

(Open3D の Web サイトの写真を利用)

図 2.1 点群処理を行えるプログラミング言語・ライブラリの例

Python は少ないコードでプログラムを書くことができ，習得難易度も低く，人気のプログラミング言語である．そのことから，点群処理をこれから学んでみようと思う初学者にとって，Python で利用できる Open3D などが利用しやすいといえる.

2.2　NDT Scan Matching のアルゴリズム

NDT は，参照点群をボクセルやグリッドといった格子状の領域に分割し，各領域内に含まれる参照点群のデータをガウス分布で近似し，ガウス分布と入力点群の間でマッチングを行う手法である.

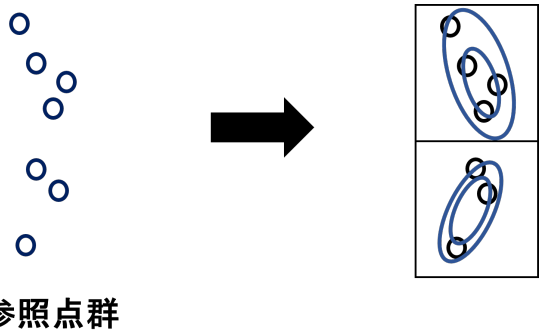


図 2.2　参照点群をガウス分布で近似する([1]を参考に作成)

図 2.2 で，図中の楕円がガウス分布を表しており，中心にいくほど確率が高くなる．ガウス分布で近似するには各グリッドセル内に含まれる参照点群の平均，分散共

分散行列を用いる．各グリッドセル内に含まれる参照点群数を n，各点の座標を $\boldsymbol{X}_i\ (x_i,\ y_i)$とすると，平均$\boldsymbol{q}$，分散共分散行列Σは以下の式で算出する．

$$\boldsymbol{q} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{X}_i \tag{1}$$

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n} (x_i - q_x)\left(y_i - q_y\right)^T \tag{2}$$

次に，参照データと入力点群の両方が含まれるグリッドセルのみ，マッチングの計算を行う．入力点群の位置が計算された座標変換パラメータによって移動することで，参照データと入力点群の両方が含まれるグリッドセルが修正され，演算に利用されるグリッドの状態が変化する．参照データをガウス分布で近似してマッチングを行うことにより，参照点群の座標データを保持する必要がなく，各グリッドセルの平均，分散共分散行列の値のみ保持できれば良いので，参照データのデータ量が少なくて済むのが利点の一つである．

参照データと入力点群のマッチング度を表す評価関数 E を以下に示す[1]．評価関数を求めるための変数を以下のものとする．

- $\boldsymbol{p} = (p_n)_{n=1,2,3}^{t} = \left(t_x,\ t_y,\ \theta\right)^t$：入力点群の座標変換パラメータ
- $\boldsymbol{x}_i = (x, y)^t$ ： 入力点群の i 番目のデータ
- $\boldsymbol{x}_i'$ ： 座標変換パラメータ $\boldsymbol{p}$ により変換された$\boldsymbol{x}_i$
- $\boldsymbol{q}_i, \Sigma_i, N$ ： $\boldsymbol{x}_i'$に対応するグリッドセルに含まれる参照点群の平均座標，分散共分散行列，点の数

$$E = \sum_{i=0}^{N-1} \exp\left(\frac{-(\boldsymbol{x}_i' - \boldsymbol{q}_i)^t\ \Sigma_i^{-1}\ (\boldsymbol{x}_i' - \boldsymbol{q}_i)}{2}\right) \tag{3}$$

また，$\boldsymbol{x}_i$を座標変換パラメータ $\boldsymbol{p}$ によって$\boldsymbol{x}_i'$に変換する式は，

$$\boldsymbol{x}_i' = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{4}$$

式(3)の exp()内は，$\boldsymbol{x}_i'$と$\boldsymbol{q}_i$のマハラノビス距離を求める式になっている．マハラノビス距離は，データの相関関係を考慮した距離のことで，多変量解析に用いられる．それにマイナス記号が付き，2 で割っている．これは，多次元ガウス分布の式で，基準となる値（この場合，平均座標$\boldsymbol{q}_i$）から遠くなればなるほど値が小さくなる．

NDT では，収束演算によって式(3)の評価値を最大化するような座標変換パラメータ $\boldsymbol{p}$ を求める．NDT の原論文では収束演算をニュートン法で行うが，本研究では収束演算

を勾配法で行う．ここで収束演算とは，同じ計算を繰り返すことで求める値を収束させる計算法である．

勾配法の計算を行うために式(3)を微分し，一次導関数を求める．式(3)内の，$\boldsymbol{q}_i$, Σ_i は，参照データとしてあらかじめ計算されたものである．マッチング計算の過程で変化することがないため，定数として扱うことができる．したがって，式(3)の微分は，$\boldsymbol{x}_i'$を求める式(4)に含まれる座標変換パラメータ$\boldsymbol{p} = (t_x, t_y, \theta)$の各パラメータについて偏微分し，3 つの偏微分値をベクトル化したもの，すなわち勾配である．これらの計算は式(5)から(7)のように進む．なお，以下より$\boldsymbol{q}_i = \boldsymbol{x}_i - \boldsymbol{q}_i$ とおく．

$$E' = -\boldsymbol{q}^t \Sigma_i^{-1} \frac{\partial \boldsymbol{q}_i}{\partial \boldsymbol{p}} \exp\left(\frac{-\boldsymbol{q}_i \Sigma_i^{-1} \boldsymbol{q}_i}{2}\right) \tag{5}$$

また，$\boldsymbol{q}_i$についての偏微分値はヤコビアン行列 J として求められる．

$$\mathrm{J} = \begin{pmatrix} 1 & 0 & -\mathrm{x}\sin\theta - \mathrm{y}\cos\theta \\ 0 & 1 & \mathrm{x}\cos\theta - \mathrm{y}\sin\theta \end{pmatrix} \tag{6}$$

式(5), (6)より，式(3)の一次導関数は以下となる．

$$E' = -\boldsymbol{q}^t \Sigma_i^{-1} J \exp\left(\frac{-\boldsymbol{q}_i \Sigma_i^{-1} \boldsymbol{q}_i}{2}\right) \tag{7}$$

なお，$-\boldsymbol{q}^t$は 1×2 ベクトル，Σ_i^{-1}は 2×2 行列，Jは 2×3 行列 $\exp\left(\frac{-\boldsymbol{q}_i \Sigma_i^{-1} q_i}{2}\right)$ はスカラ値であるため，最終的に求められる値は 1×3 のベクトルとなり，座標変換パラメータ$\boldsymbol{p}$の各要素に対応した偏微分値が得られる．

2.3　勾配法による最適化

勾配法は，関数の最大値や最小値を求めるために用いるアルゴリズムである．以下では，図 2.3 の関数 $f(x) = -x^2$の最大値を求めることを例に説明する．

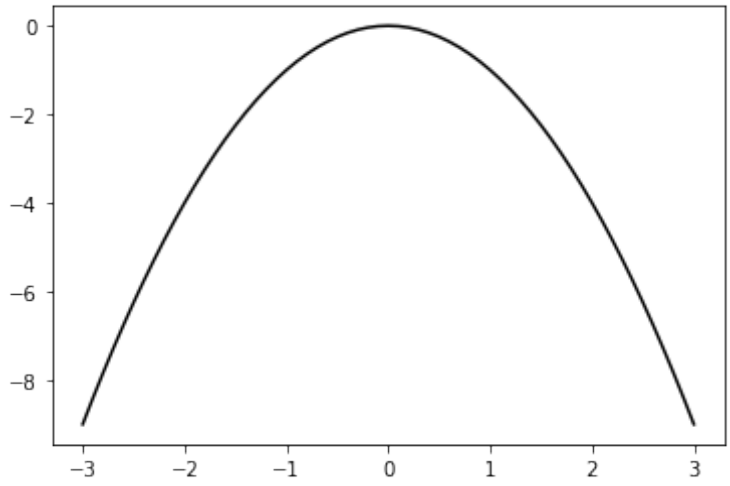


図 2.3 $f(x) = -x^2$

この関数は x = 0 の時に最大値をとる．これを，勾配法を用いて求めていく．勾配法では，まず，適当な初期位置をx_0として定義する．次に，x_0における一次導関数$\frac{df}{dx}$を求める．x_0の位置を以下の式により更新し，x_1とする．

$$x_i = x_{i-1} + n\frac{df}{dx} \tag{8}$$

式(8)中の n を学習率と呼ぶ．学習率の値によって収束の速度が変化する．学習率が小さすぎると収束に時間がかかる．学習率が大きすぎた場合，x_iの位置が最大値付近で反復してしまい，収束に時間がかかるようになる．これらの工程を，勾配が設定された閾値以下になるまで繰り返し行う．処理が終了した地点での$f(x_i)$を最大値とする．

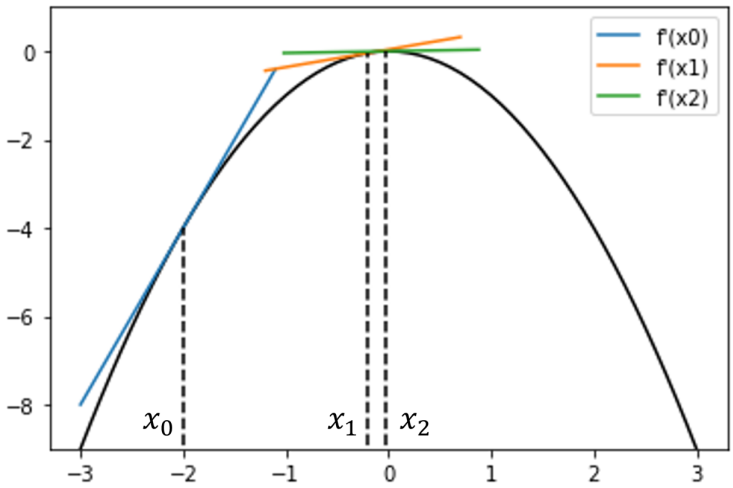


図 2.4 勾配法により関数の最大値を求める様子

図 2.4 では，初期位置 $x_0 = -2$ ，閾値を 0.1，学習率を 0.45 として，勾配法を行った様子を示している．傾き$f'(x_i)$ の値をもとに，x_{i+1} の値が求められ，$f'(x_2)$の値が閾値以下となったため，処理を終了している．

多変数で勾配法を行う場合，各変数について偏微分を行い，その偏微分値を用いて各変数の位置を更新し，勾配の大きさが閾値以下になるまで繰り返し計算する．

2.4　k-d 木を用いた最近傍探索

k-d 木は，k 次元の空間を分割する木構造である．2 次元空間を分割する場合には，1 つのノードが 2 つの子を持つ 2 分木になる．一般には，多次元空間を各座標軸に垂直に分割し続けることによって構築される．

本研究のような 2 次元空間で k-d 木を構築するには，空間の分割を x 軸→y 軸→x 軸…のように交互に行っていく．まず点群全体を x 軸の値でソートし，中央の点を中心に分割する．この時の中央の点が構築される木のルートノードとなる．次に，左の子ノードを構築するために，ソート結果でルートノードの値より小さい点群を y 軸の値でソートし，中央となる点を中心に 2 分割する．ルートノードから見て左のノードの構築が終われば，右側のノードの構築を行う．例として次の座標群 $\{(4.1),(1,3),(1,1),(6,4),(2,4),(5,2),(3,5)\}$を用いて k-d 木を構築すると，対象の 2 次元空間は図 2.5 のように分割される．

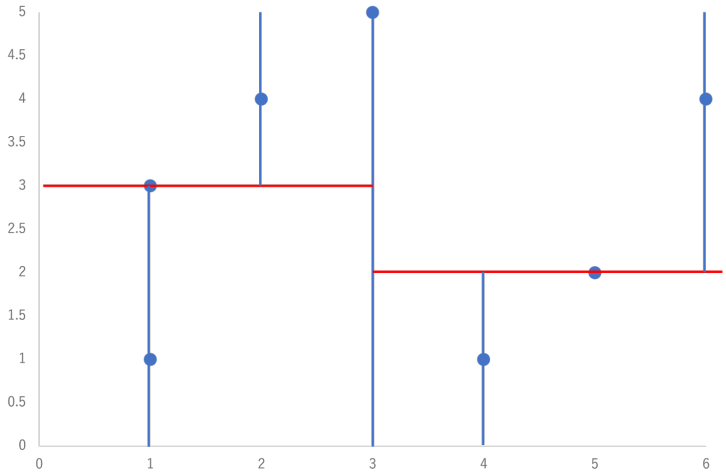


図 2.5　k-d 木により分割された 2 次元空間

図 2.5 をみると，ルートノードとなる $(3,5)$ を中心に縦に空間が分割されている．ルートノードから見た左側の空間は y 軸の値でソートし，中央となる $(1,3)$ の点で空間が分割されている．このように構築された k-d 木を利用することで，最近傍探索を高速化することができる．

k-d 木を用いた最近傍探索は，入力された点をクエリとして，各節を分割した軸についてクエリと節のデータの大小関係を比較しながら葉まで探索する．葉に到達した直前の節を暫定解としてクエリとの距離を計算する．その後，バックトラックを行いながら k-d 木を登る動作を行い，クエリと各節の距離を計算し暫定解よりも距離が近い場合，暫定解を更新していく．このようにしながらルートノードまで戻ってきた場合，探索を終了する[7]．

k-d 木を利用した最近傍探索の計算量は，グリッド数を M として，最初の 2 分探索の計算量が$O(\log M)$である．それに加えてバックトラックの計算が必要であるが，全体として，総当たりの計算量よりも小さくなると考えられる．

本研究では，この k-d 木の構築と最近傍探索を，SciPy モジュールを用いて実装した．SciPy モジュールで使用した関数を図 2.6 に示す．

```
import scipy.spatial as ss # モジュールのインポート
tree = ss.KDTree(array) # k-d 木の構築 array = 元の座標群
d, i = tree.query(x) # 最近傍点を探索 x=クエリ, d=最近傍点との距離 I=最近傍点のインデックス
```

図 2.6　SciPy モジュールで使用した関数

3 章　NDT Scan Matching の実装

実際の地図情報を表現した点群を NDT Scan Matching させると，参照点群と入力点群の位置が正確にマッチングしない場合が多かった．その理由は，これらの点群の分布がガウス分布から大きく外れるからである．3 章では，この問題の解決法や点群をグリッド分割した場合の NDT Scan Matching について説明する．

3.1　外れ値への対応

NDT Scan Matching では，評価関数を最大化することで座標変換のパラメータを求める．このような計算における評価関数はパラメータを求めるための尤度関数とよばれる．尤度関数の最大化において，尤度関数の対数を最大化しても，求めることができる座標変換パラメータは同じである．尤度関数の対数を対数尤度関数とよび，尤度関数を最大化する代わりに，対数尤度関数を最大化する場合がある．

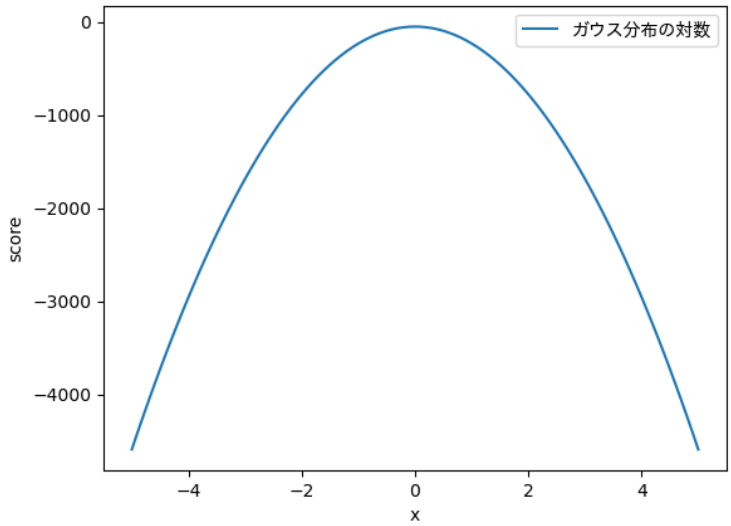


図 3.1　評価関数（ガウス分布）の対数尤度関数のグラフ

NDT Scan matching の評価関数はガウス分布である．図 3.1 は，ガウス分布の対数をとり，対数尤度関数にしたものである．図 3.1 は上凸の 2 次関数になる．対数尤度関数に，関数が最大値を取る x=0 から大きく離れた値を与えると，評価値への影響が際限なく大きくなることがわかる．つまり，対数尤度関数を用いてマッチングさせると，平均から大きく離れた値（外れ値）が存在した時に，評価値に大きな影響を与えてしまう．点群地図の座標値の分布はガウス分布から外れており，対数尤度関数を用いたマッチングでは，正確にマッチングすることが少ない．

実用的に NDT を使用するための外れ値への対応として，ガウス分布と一様分布を組

み合わせた式(9)の評価関数を使用する[3].

$$E = \sum_{i=0}^{N-1} c_1 \exp\left(\frac{-(\boldsymbol{x}_i' - \boldsymbol{q}_i)^t \, \Sigma_i^{-1} \, (\boldsymbol{x}_i' - \boldsymbol{q}_i)}{2}\right) + c_2 \tag{9}$$

式(9)内のc_1, c_2は定数であり，c_2が一様分布の大きさを決定し，c_1は区切られたグリッドの範囲で評価値の積分値が 1 になるように設定される. 式(9)の対数尤度関数を図 3.2 に示す.

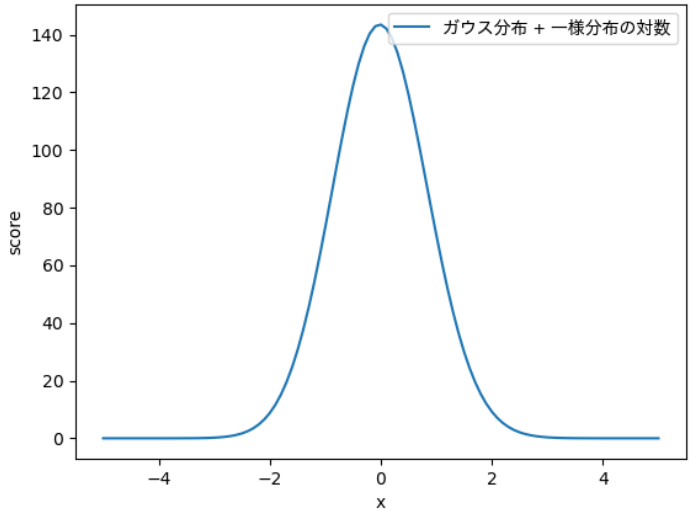


図 3.2　ガウス分布と一様分布の組み合わせの対数尤度関数

図 3.2 を見ると，ガウス分布と一様分布を組み合わせることにより，最大値を取る x=0 から離れた値が対数尤度関数に与えられたときの影響が制限されていることがわかる．式(9)の対数尤度関数は単純な一次微分を持たず，収束演算を利用して最大値を求めることが難しい. 図 3.2 のグラフを見ると，一般的なガウス分布のグラフに酷似している．そこで，この対数尤度関数をガウス分布で以下のように近似する[3].

$$\log(E) = \sum_{i=0}^{N-1} -d_1 \exp\left(\frac{-d_2(\boldsymbol{x}_i' - \boldsymbol{q}_i)^t \, \Sigma_i^{-1} \, (\boldsymbol{x}_i' - \boldsymbol{q}_i)}{2}\right) \tag{10}$$

式(10)内のd_1, d_2は定数であり，以下のように求める [3].

$$d_3 = -\log(c_2)$$
$$d_1 = -\log(c_1 + c_2) - d_3 \tag{11}$$

$$d_2 = -2\log\left(\frac{\left(-\log\left(c_1\, exp\left(-\frac{1}{2}\right)+c_2\right)-d_3\right)}{d_1}\right.$$

式(10)を新たな評価関数とし，収束演算により最大値を取る座標変換パラメータを求める．

3.2　参照点群のグリッド分轄

本研究では，以下の手順で参照点群を格子状のグリッドセルに分轄した(図 3.3)．

① 参照点群の（x，y）座標の最小値・最大値を求める．

② グリッドセルの幅と高さを決め，x=min，y = min の地点から x 軸方向に移動しながら，グリッドの最下列について，グリッドセルの位置を確定させていく．

③ 右側のグリッドセルに達したら，グリッドセルの位置を一段上げて，左側からグリッドセルの位置を確定させていく．

④ ③の処理を上端に達するまで繰り返す．

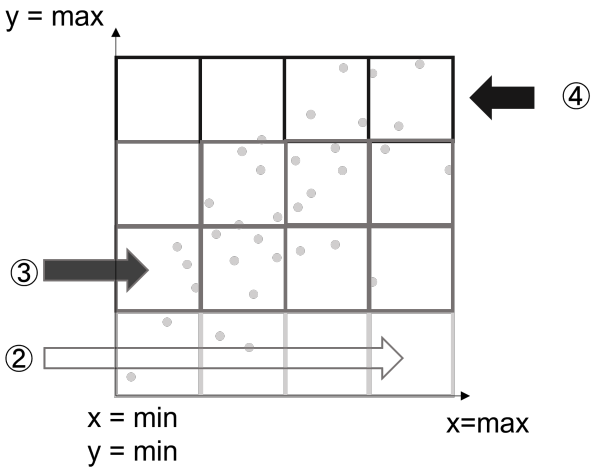


図 3.3　参照点群をグリッドに分轄する様子

作成したグリッドの各セルに参照点群が含まれていれば格納していく．各セルに格納された参照点群の平均・分散共分散行列を求める．本研究では，点の数が 10 以下のセルは，無効なセルとしてマッチング処理には使用しない．

2 章で説明したように，NDT のマッチング処理は，入力点と参照データの両方が含まれるグリッドセルのみ収束演算に使用する．この時，各入力点がどのグリッドセルに含まれているかをグリッド全体で探索する必要がある．グリッドセル数を M とすると計算量は最大で $O(M)$ となり，グリッドセル数が多くなるほど計算量が大きくなる．そのため本研究では，各グリッドセルの中心座標を用いて k-d 木を作成し，最近傍探

索を行い，各入力点と距離の近いグリッドセルを探索し，入力点がグリッドセルに含まれているかを判定している．

4 章　NDT Scan Matching の評価

4 章では，3 章のアルゴリズムと NDT の性質を利用して，実装した NDT を評価していく．

4.1　予備実験：ガウス分布とガウス分布に従う点群とのマッチング

NDT は，参照点群をガウス分布で近似し，入力点群とマッチングを行う．指定したガウス分布を参照データとし，同じガウス分布に従う点群を入力点群とし，ガウス分布同士でのマッチングを行った．マッチング前の点群を図 4.1 に示す．なお，今回は参照点群全体を 1 つのグリッドセルとして扱い，NDT による位置合わせを行った．

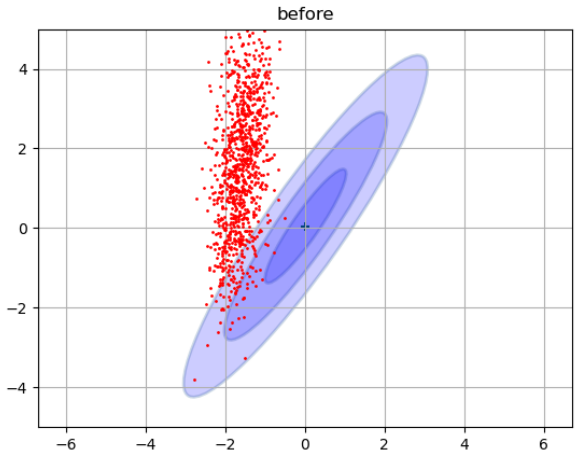


図 4.1　マッチング前の点群

図 4.1 の青い楕円が参照データで，平均[0,0]，分散共分散行列[[1,1.3]，[1.3,2]]に従うガウス分布を表す．赤色の点群が参照データと同じ平均，分散共分散行列に従って分布する 1000 個の点を，平行・回転移動させたものである．参照データと入力点群をマッチングさせた結果を図 4.2 に示す．

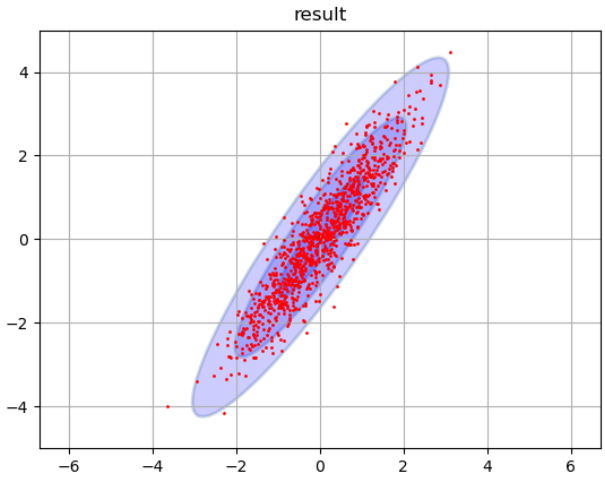
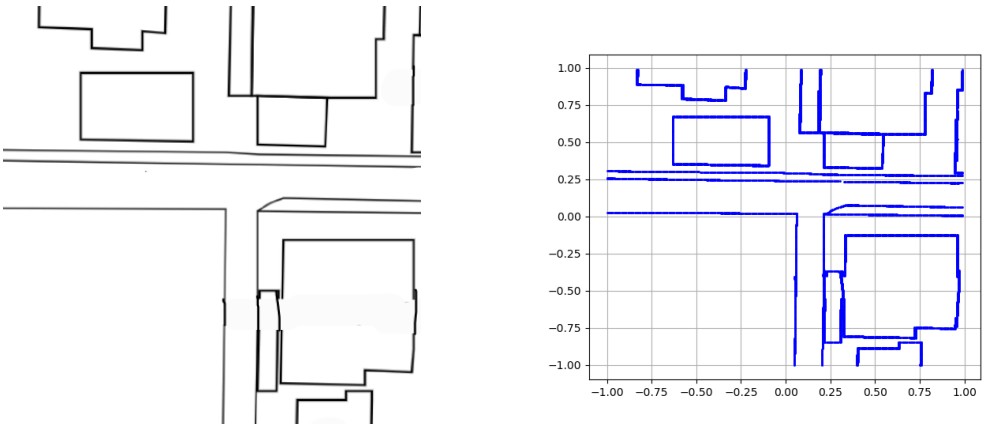


図 4.2　マッチング結果

図 4.2 を見ると，参照データのガウス分布に合致するように，入力点群が平行・回転移動し，マッチングができている．この予備実験によって，ガウス分布を参照データとした場合のマッチングができることがわかった．

4.2　地図データを使ったマッチング

NDT Scan Matching は自動運転における自車両の位置決めに利用されている．これに近い環境での動作確認を行うため，地図を表す点群を図 4.3 のように作成した．図 4.3 左は地図画像を二値化した画像である．二値画像の黒色部分の座標を抽出し，地図を表現する参照点群とした．これを図 4.3 右に示す．



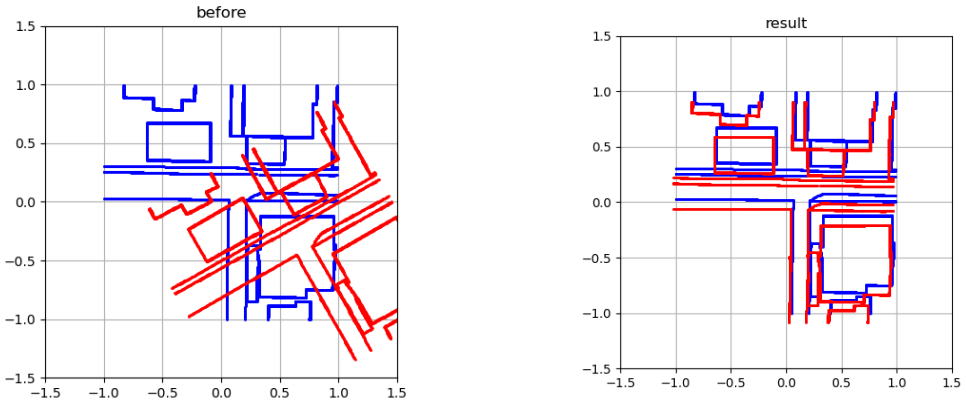
(a) 地図を表す画像　　(b) 画像から作成した参照点群

図 4.3　使用する参照点群

参照点群に用いる地図の座標値の範囲はさまざまである．その範囲によって，収束の

速度が大きく異なる．これを解決するために，参照点群の x 軸方向の座標値の範囲を−1〜1 に正規化した．それに伴って，y 軸を x 軸の縮小率に合わせて縮小した．地図を表す画像から作成したものを参照点群，参照点群を平行・回転移動したものを入力点群としてマッチングを行った．マッチング前の点群とマッチング結果を図 4.4 に示す．

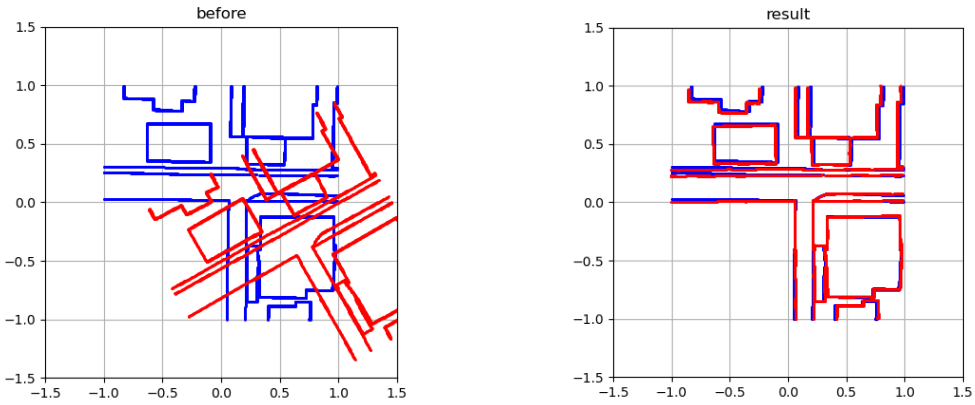
図 4.4 で，青色点群が参照点群，赤色の点群が入力点群を表す．この実験でも参照点群全体を一つのグリッドセルとして扱った．また，この実験では 3.1 章の外れ値への対策を施さずにマッチングを行なった．(a)の初期位置の入力点群がマッチング処理によって(b)のように参照点群に近づいているが，誤差が生じている．



(a)マッチング前　　(b)マッチング結果

図 4.4　地図を利用した点群でのマッチング(外れ値への対策前)

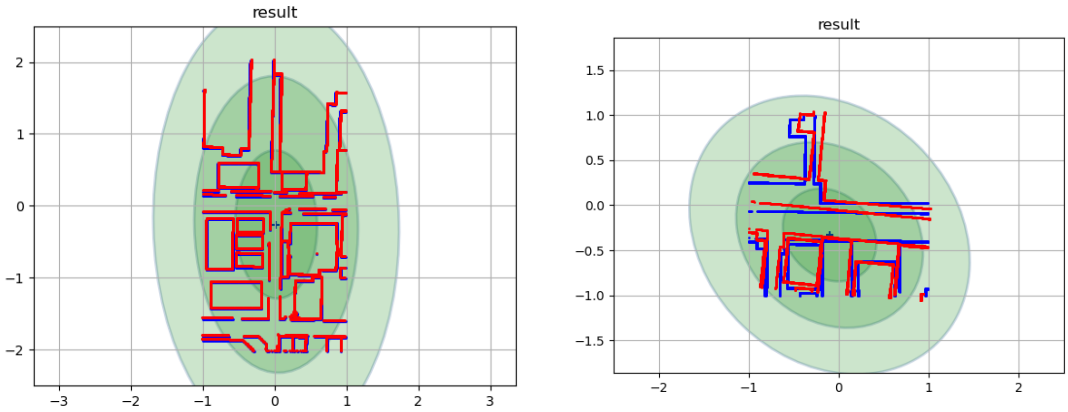
外れ値への対策を施してマッチング処理を行なった．マッチング前の点群とマッチング結果を図 4.5 に示す．(a)の初期位置の入力点群がマッチング処理と外れ値への対策によって(b)のように図 4.4 より微小な誤差で，参照点群とマッチングができている．このことから，3.2 章の外れ値への対策は有効であるとわかる．以降の実験では，外れ値への対策を施してマッチングを行う．



(a)マッチング前　　(b)マッチング結果

図 4.5　地図を利用した点群でのマッチング(外れ値への対策後)

別の地図を表す画像を参照点群として使用した場合のマッチング結果を図 4.6 に示す．図 4.6 の緑の楕円は，参照点群をガウス分布で近似した分布である．(a)の場合のマッチングは正確だが，(b)の場合ではある程度の誤差が生じた．これは，ガウス分布の形が(a)は縦長であったのに対して，(b)は円に近くなっており，角度を合わせるための特徴が少なかったのだと考えられる．



(a)マッチング結果の例
(分布の方向性が強い場合)

(b)マッチング結果の例
(分布の方向性が弱い場合)

図 4.6　分布の方向性によるマッチング結果の比較

4.3　グリッドに分轄してのマッチング

NDT の本来の手法通りに，参照点群をグリッド分轄してマッチングの処理を行った．

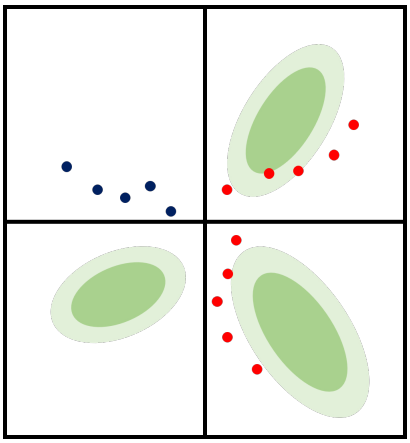


図 4.7　参照点群をグリッド分割したイメージ図

図 4.7 の緑の楕円はガウス分布で近似された参照点群を表し，赤と青の点はどちらも入力点群を表す．NDT では，入力点が含まれるグリッドセルに参照データが存在した場合のみ計算を行うので，図 4.7 の時点では赤色の 10 個の点で計算を行い，青色の 5 個の点については計算を行わない．計算を行う 10 個の各点が，現在自身が含まれているグリッドセルの正規分布を利用して評価関数の勾配を算出する．図 4.7 の場合，10 個の勾配が算出される．算出された全ての勾配の平均が入力点群全体の勾配となる．入力点群全体の勾配を使用し，勾配法で座標変換パラメータを求める．求められた座標変換パラメータに従って入力点群全体を平行・回転移動をする．そうした工程を，勾配法が収束するまで行い，マッチングを行う．

図 4.4 と同じ参照点群を，グリッドセルの 1 辺の長さを 0.5 として分轄した様子を図 4.8 に示す．

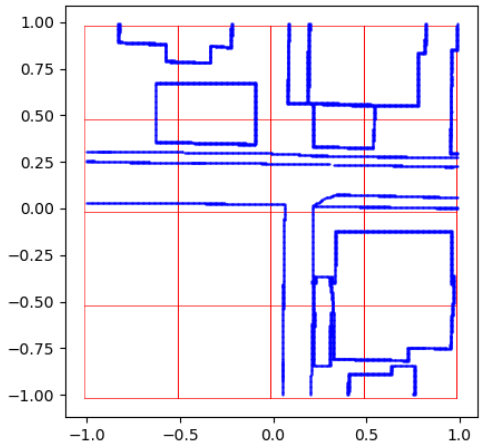
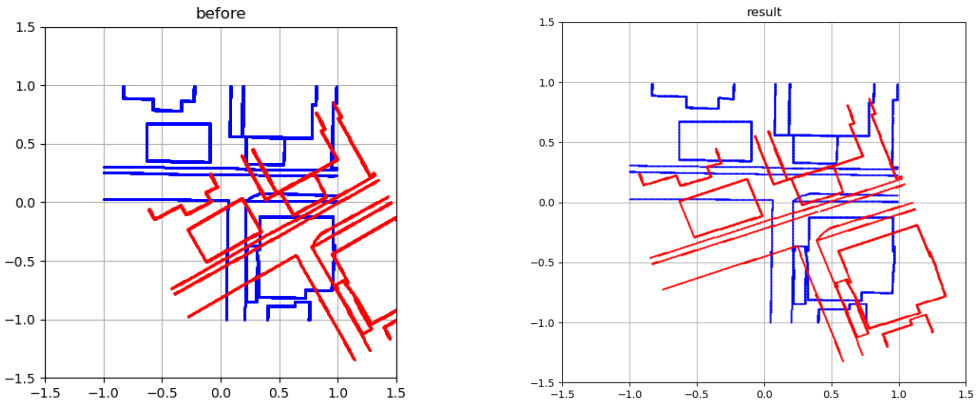


図 4.8　1 辺の長さ 0.5 のグリッドセルに分轄した参照点群

図 4.8 の赤色の線で分割した空間が 1 つのグリッドセルになり，各セルに含まれる参照点群の平均と分散共分散行列を求めて，ガウス分布で近似する．グリッド分轄した参照点群と入力点群をマッチングさせた結果を図 4.9 に示す．



(a)マッチング前　　(b)マッチング結果

図 4.9　参照点群をグリッド分轄したマッチング

図 4.9 を見ると，4.2 章で行ったマッチングの結果と比較して誤差が大きく，マッチングに失敗している．入力点群とその最近傍のグリッドセルを対応付け，マッチング処理を行っているが，入力点群の初期位置が参照点群と離れている場合，最近傍のグリッドセルが対応するグリッドセルではない場合が多い．対策として，対応させるグリッドセルの候補として，半径 r 内に存在する複数のグリッドセルを選択することにした．それら複数のグリッドセルに対して評価値を算出し，最も評価値の大きいグリッドセルと

入力点群を対応付けた. この考え方に従ってマッチングを行った結果を図 4.10 に示す.

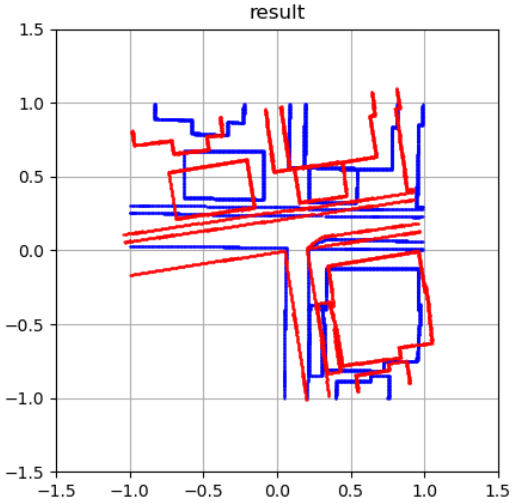
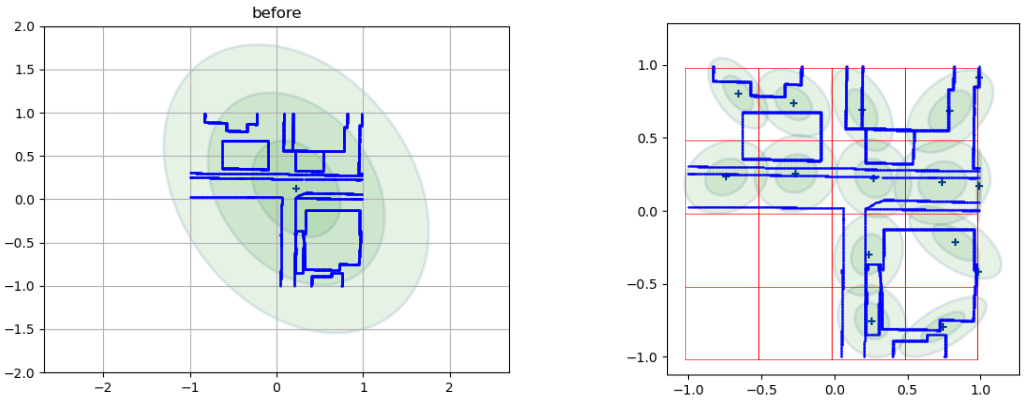


図 4.10　グリッドセルの探索手法を変更した場合の結果

図 4.10 は図 4.9 と比べて x, y 軸方向の平行移動量に関するマッチングの精度が向上している. これは, 入力点と目標となるグリッドが正しく対応付けられるようになったからだと考えた. しかし, 角度についてのマッチングは失敗している.

グリッドに分割する前と後の参照データのガウス分布の様子を図 4.11 に示す.

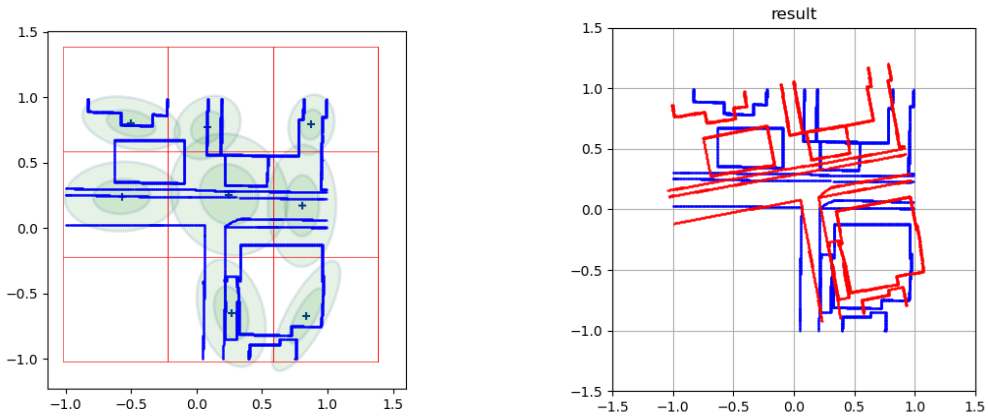


(a) グリッドに分割する前　　(b) グリッドに分割した後

図 4.11　ガウス分布の比較

図 4.11 をみると, (a) のガウス分布は角度に方向性がみられる. (b) のガウス分布は方向性が見られるガウス分布もあるが, 中心付近のグリッドセルのガウス分布は円状で方向性がない. 1 つのグリッドセルに含まれる参照点群の数も少なく, 特徴量が減り, 処理に影響を与えたと考えられる. そこで, グリッドセルの 1 辺のサイズを 0.5 から

0.8 に拡大して，1 つのグリッドセルに含まれる参照点群の数が多くなるようにし，マッチングを行った．参照点群のガウス分布とのマッチング結果を図 4.12 に示す．

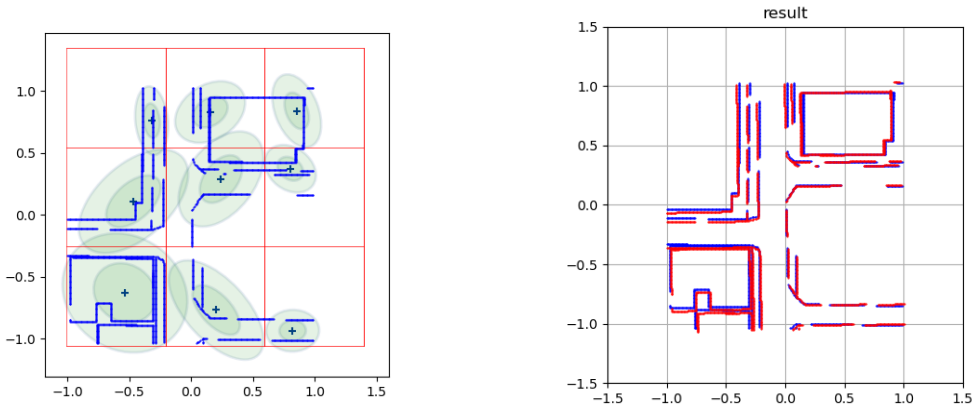


(a)参照点群のガウス分布　　(b)マッチング結果

図 4.12　グリッドサイズを拡大させた場合のマッチング結果

グリッドのサイズを拡大し，1 つのグリッドに含まれる参照点群の数を増やしても角度のマッチングが失敗したことがわかる．これは，参照点群の中央付近のガウス分布が円状であり，マッチングに影響を与えていると考えられる．

比較として別の地図画像を用いてマッチングを行った．参照点群のガウス分布とマッチング結果を図 4.13 に示す．



(a)参照点群のガウス分布　　(b)マッチング結果

図 4.13　別の地図画像を用いたマッチング結果

(a)のガウス分布は，円状に分布しているグリッドセルがなく，全グリッドセルに角度的特徴がある．(b)をみると，マッチングが成功していることがわかる．

これらのことから，実装した NDT がマッチングに成功するには，参照点群のガウス分

布に角度的な特徴が必要だと考えた．グリッド分割した場合は，全グリッドセルの中に円状に分布するグリッドセルが存在すると，マッチングに影響を及ぼして仕舞うと考える．

5章　結論

本研究では，Pythonを用いてNDT Scan Matchingを実装した．道路地図を想定したサンプルの点群を用意し，マッチングさせた．NDTは，点群とガウス分布をマッチングさせる手法であり，評価関数としてガウス分布を用いている．入力点群の各点と，参照データの平均位置とのマハラノビス距離をガウス分布で変換し，その積算を最大化することが基本となっている．また，近似するガウス分布からの外れ値を考慮するために，ガウス分布と一様分布を組み合わせた分布を使用することで，高精度化が可能になっている．

実装したNDTでは，外れ値を考慮した評価関数を最大化させる座標変換パラメータを，勾配法を用いて求めた．今回の実験では，参照点群全体を1つのグリッドセルとして扱った場合に，入力点群とのマッチングに成功した．参照点群を複数のグリッドセルに分割した場合，マッチングに成功する地図画像と，失敗する地図画像があった．これは，グリッドの中でガウス分布の形状が円状に分布しているグリッドセルが存在する場合，マッチングに影響を与えていると考えた．

課題として残ったこととして，参照点群を複数のグリッドセルに分割した場合，影響を与えてしまうグリッドセルの分布に対しての対応ができなかった事である．対応策として，分散共分散行列で使用される共分散の値が小さく，ガウス分布が円状に分布するグリッドセルについては角度のマッチングを行わない手法が考えられる．

参考文献

[1] P. Biber and W.Straßer, “The Normal Distributions Transform: A New Approach to Laser Scan Matching,” IROS 2003, pp.2743-2748.

[2] 田窪 朋仁, 上撫 琢也, 前 泰志, 新井 健生, 大原 賢一 「高解像度 NDT グリッドマップを用いた環境地図生成」 2012 年

[3] Martin Magnusson, “The Three-Dimensional Normal-Distributions Transform – an Efficient Representation for Registration, Surface Analysis, and Loop Detection,” Orebro University Thesis for Ph.D, December 2009.

[4] 「LIDAR（ライダーとは？）」, MathWorks の Web サイト, https://jp.mathworks.com/discovery/lidar.html, 2023 年 1 月 18 日確認.

[5] Point Cloud Library の Web サイト, https://pointclouds.org/, 2023 年 1 月 18 日確認.

[6] Open3D の Web サイト, http://www.open3d.org/, 2023 年 1 月 18 日確認.

[7] 和田　俊和,「最近傍探索の理論とアルゴリズム」, 情報処理学会研究報告, Vol.2009-CVIM-169, No.13.

謝辞

本論文を作成にあたり,丁寧な御指導を賜りました蚊野浩教授に感謝いたします.

付録　開発したプログラムの説明

[ファイル名]

oneGridNDT.py

[内容]

2 次元の NDT Scan Matching を実装したファイル．画像データを 2 値化し，座標を抽出し参照点群とする．勾配法を用いた収束演算を行って，評価値を最大化する座標変換パラメータを算出する．

[関数]

表 1　関数リスト

関数名	内容
main()	実行用の関数
moove()	点群の平行・回転移動を行う
score()	評価値の総和を算出する
E()	評価値を返す
grad()	勾配を返す

[ファイル名]

multipleGridNDT.py

[内容]

oneGridNDT.py に参照点群をグリッドに分割する機能を付け加えたもの．

[関数]

oneGridNDT.py と重複する関数は省略

表 2　関数リスト

関数名	内容
add_noise()	点群にノイズを加える
create_grid()	参照点群をグリッドに分割する
pairing()	入力点群と各グリッドを対応づける

[ファイル名]

grid.py

[内容]

multipleGridNDT.py で利用する．分割した各グリッドを表す．

[関数]

表 3 関数リスト

関数名	内容
__init__()	コンストラクタ
inside_grid()	与えられた点がグリッド内に存在するかを判定する
pairing()	入力点群と各グリッドを対応づける
get_ave()	平均座標を返す
get_icov()	分散共分散行列の逆行列を返す
get_cov()	分散共分散行列を返す
count_points()	点群の総数を返す
get_center()	中心座標を返す
add_point()	点を追加する
calc_param()	平均と分散共分散行列を求める