タイトル詐欺回避のためのおまけ

@ogiwara_oekaki

December 17, 2021

初めまして、ogiwara(@ogiwara_oekaki) です。このたびは本書をお手に取っていただきありがとうございます。これをあなたが見ているということは、私は無事に同人活動をスタートできたということです。ちょうど一年前の例大祭である、2020 年 10 月 18 日の秋例大祭に一般参加した際に感動してから同人活動の準備を進めて来ました。

「僕もそっち側 (サークル側) に回りたいなあ…でも絵は一度たりとも描いたことないし…なら、今からでも遅くはないから絵を描き始めよう！」と決心して、その数日後から絵の描き方の勉強を初めました。

本書の「幻想郷に情報革命を」は長編の漫画としてシリーズ化していくつもりです。筆者が遅筆なせいもあり、今回の作品は毎日 2,3 時間の作業でなんと制作に 8ヶ月もかかってしまいました！次は 1 からストーリー構成・漫画の描き方を勉強し直し、より短時間でもっと面白い作品を仕上げられるようになりたいですね。

さて、「情報革命」なんてタイトルにデカデカと入っている以上、ストーリーだけじゃなくて IT 技術関連の解説も入れないとまずいなと思い、「幻想郷に情報革命を」シリーズではストーリーの後にオマケとしてキャラが IT 技術について解説する、技術同人誌ぽい要素も入れようと考えています。ただ今回は締め切りに間に合いそうになかったので、教科書みたいに普通に淡々と説明する形になっています。

今回のテーマは、流行りの「自動運転」です (本編のストーリーと全く関係無いじゃん…)。筆者が自動運転関連の研究を大学で行っているという事もあり取り上げさせていただきました。

文章を左のページから右のページに向かって読む、というちょっと気持ち悪い構成になっていますがご容赦下さい。

1　自動走行ロボットや自動運転における自己位置推定

自動運転における問題の一つに、「ある目的地が与えられ、その目的地まで移動する」というものがあります。さて、それではこの問題は私たち人間は解けるのでしょうか？おそらく読者のほとんどの方にとっては簡単にクリアできる問題だと思われます。ではそれはなぜでしょうか？

まず、私たち人間は地図を確認します。最近なら外出先でも地図アプリなどで簡単に地図が見れますし、馴染みの道であればわざわざ地図を見る必要もなく、自分の頭の中の地図を参照するでしょう。次に、GPS を用いたり周りの建物を確認することによって自分が地図のどこに居るのかを確認します。最後に、目的地が地図上のどこにあるのかを確認し、その目的地の方向に向かって歩き出して障害物があれば回り道などをし、最終的に目的地に到達できます。まとめると、

1. (地図がなければ) 作る
2. GPS や周りの景色を元に、今自分がどこにいるのかを把握する
3. 目的地まで障害物を避けながらなるべく最短ルートを選ぶ

という手順を踏めばいいわけですね。同じ問題をコンピューターを用いて解く場合にも、同じ手順を踏めば良いわけです！このようにコンピューターである問題を解くときには、同じ問題を人間ならどう解くか、と言う直感さえ分かれば問題の見通しがかなり良くなります。難しいのは、この「直感」をどうプログラムで表現するかと言う点です。

上記三つの手順は自動運転においてはそれぞれ

1. SLAM(Simultaneous Localization and Mapping)
2. 自己位置推定
3. 経路生成

という用語と対応します。ここでは自動走行ロボット・自動運転車における（そうです、自動運転車はロボットであると捉えることができます！）自己位置推定問題について扱います。

1.1　自動走行ロボットや自動運転車における自己位置推定

さて、地図が与えられている状況下で自分の位置を確認するにはどうすれば良いでしょうか？ここでまた、人間の「直感」からまた考えてみましょう。

まず目を瞑って耳を塞いだ状態で自分の場所を確認できるでしょうか？目が見えないので周りの景色は確認できませんし、耳を塞いでいるので周りの音も聞こえないので、うまくいきそうにありません。スタート地点は地図上のどこにいるかわかった状態で、スタート地点で目と耳を塞ぎ、自分の歩数だけを頼りにして何 m 進んだかを計算し、自分の位置を確認する方法はどうでしょうか？この方法もうまく行かないことは、皆さんはスイカ割りの経験からよくわかるかと思われます。つまり、歩くたびにどんどんと自分の予想と実際の位置にズレが生じていくのです！しかしながら、例えばスタート地点から半径 5m 以内といった限られた範囲であればこの方法は有効でしょう。

では今度はちゃんと目と耳を使いましょう。周りに建物が多い環境ならば目を使えば周りの建物が確認でき、自分が地図上のどこに居るのかわかるでしょう。しかしながら、例えば砂漠や森の中にいる時には周りに特徴的な物がないため、自分の居場所がわかりません。

今度はスマートフォンに載っている文明の利器である GPS を使ってみましょう (ちなみに GPS は衛星と地表の間が非常に離れているために生じるミリ秒の時間のズレを解消するために、相対性理論を使っています。かっこいいね)。これならば砂漠でも自分の位置が確認できますね！(森の中だと専用の GPS じゃないと厳しいけど…) しかしながら、建物やトンネルの中だと GPS の電波が届かなくて使えないですね。また、高い建物が近くにある環境では GPS に 10m 前後のズレが生じることが知られています。まとめると、私たち人間は自分の居場所を把握す流ために、大まかに以下三つの「センサー」に頼っています。

1. 足の歩数の感覚
2. 目や耳
3. GPS

各「センサー」は先ほど説明したように一長一短ではありますが、私たち人間はこれらを組み合わせて使うことで自分の居場所を確認して来ました。足の感覚を使えば目や GPS が使えない環境でもある程度は自分の居場所が確認できますし、目を使えば周りに特徴的な建物や自然物がある場合には非常に大きな地図の中から自分の居場所を特定できますし、GPS があれば周囲の建物や自然物を確認することなく自分の居場所がわかります。

1.2 ロボットにおけるセンサー

ロボットの自己位置推定においても同じように、「複数のセンサーを組み合わせて使う」と言う方法が取られています。上記三つの「センサー」は、それぞれ次のように対応します

1. オドメトリ
2. カメラや超音波センサー、LiDAR
3. GPS(ここはそのままですね)

ロボットの足とも言える車輪の回転数を元に移動距離を計算する手法をオドメトリと呼び、これが人間の足の歩数の感覚に対応します。カメラや超音波センサー、そして近年スマートフォンなどにも搭載されることによって有名になった LiDAR(Light Detection And Ranging) が目や耳に対応します。LiDAR センサーは次のように機能します。

図 1 のような建物内の地図において、LiDAR センサーを搭載したロボットが図 2 のように大広間に居るとします。ロボットの形は円で表現され、ロボットの向いている方向は円の中の線で表現されます。ここではロボットの頭は斜め 45° の教室 2 の方向に向いています。

まず、LiDAR センサーはいろんな方向にレーザーを飛ばします。ここでは LiDAR センサーは左右それぞれ 45°、合計で前方 90 度の範囲でレーザーを飛ばすと仮定します。図 3 はロボットの LiDAR センサーから各方向に向かってレーザーが射出されている様子を表しています。

次に、このレーザーは周囲の障害物や壁にぶつかることによってロボットに跳ね返ってきます。図 4 では壁に当たって跳ね返ってきたレーザーを表しています。

最後に、ロボットにレーザーが跳ね返ってくるまでの時間を元にして各角度方向に何 m 先に障害物があるかがわかります (レーザーが跳ね返ってこない場合にはその角度方向には障害物が無いと考えることができます)。図 5 は跳ね返ってきたレーザーの情報を元に各角度方向で何 m 先に障害物があるかを丸印で表しており、これを元に地形を把握することができます。

現時点で周囲の環境を把握するのに最もよく使われているのが LiDAR センサーとなります。もちろん LiDAR センサーも周囲に障害物がない状況では使えないため、ロボットは人間と同じようにこれら複数センサーを組み合わせて使います。初期位置や大まかな位置決めには GPS を使い、周囲の地形を元に LiDAR を用いて自己位置を予測し、オドメトリを用いて細かなズレを修正します。

2 地図と NDT matching

さて、自己位置推定に必要なセンサーについて説明しましたが、では与えられた地図データとセンサーから得られた観測値からどのようにして自己位置を推定するのでしょう？そもそも、地図はどのようなデータで与えれば良いでしょうか？

ここで、ロボットは屋外だけではなく屋内の環境の中でも自動走行すると言う点に注意しなくてはなりません。私たち人間がよく地図として使っているような屋外用の地図には建物の名前等が書いてあり、私たちはそれをランドマークとして自己位置を推定することができます。しかしながら、屋内の物に名前が割り当てられていることは少ないためランドマークとしては使えず、同じ方法では自己位置を推定することができません。私たち人間は、実は屋内ではなんとなくの壁や障害物の形状を「地図」として捉えているのです。つまり屋内でも動作する地図を作るには壁や障害物の形状がわかるような地図を採用する必要があります。

ここではLiDARセンサーからの点群を扱いやすくするため、点群地図を用います。例えば、図1の建物に対応する点群地図は図6のように表せます。これはLiDARセンサーを使って次のようにして作ることができます。まず、図5の状態で点群を保存した後、図7の位置・向きに移動してもう一度LiDARセンサーでスキャンを行います。すると、図5の位置では確認できなかった点群 (薄い灰色の点) が確認できます。新しく観測した点群を既存の点群と組み合わせると、だんだんと点群地図ができ上がっていきます。この処理を繰り返すことによって、図6のような点群地図を作ることができるのです。

鋭い読者であれば、地図がなければ自己位置が推定できないのに、この方法ではそもそも地図を生成するのにも自己位置の推定が必要になる、というジレンマに気がつくかと思われます。つまり「地図が与えられていない状況下で自己位置を推定しながら、同時に地図を生成する」という問題を解く必要があります。今回はこの問題については扱いませんが、これは冒頭で説明したSLAM(Simultaneous Localization and Mapping)[1]と言う技術で取り扱っています。

点群地図が得られた状態でどのようにすれば自己位置推定ができるのでしょうか？例えば自己位置がわからない状態で、図8のスキャン結果が得られたとします。この大きな角が見られるのは教室3の部分だけであり、点群地図とこのスキャン結果を照らし合わせる・マッチングを行うことによって図9のような位置・角度にロボットがいることがわかります。このマッチングは「点群マッチング」と呼ばれます。

では、この点群マッチングの作業をどうコンピューター上で動作するプログラムに落とし込むことができるか？いよいよここからが本題です。ここまでの流れで「自己位置推定」というざっくりとした問題を人間の「直感」を頼りにすることによって問題をよりクリアにしていき、「点群同士の照らし合わせ」という問題に単純化することができました。これなら数学的な処理によって表現することができ、それを実現するアルゴリズムの一つが次に説明するNDT matching[2]となります。

2.1 NDT matching

NDT matching[2]は同じ場所で観察された二つの点群A,Bが与えられた時に、Bの点群をどれだけ並行移動・回転すればAの点群と最も近くなるか、それだけマッチするかを算出するアルゴリズムです。自己位置推定においてはこのアルゴリズムを用いてA=地図の点群、B=特定地点で観測された点群とし、最もマッチする並行移動・回転量を自己位置とします。以下、地図の点群データは地図点群、位置が未知の場所で観測された点群データは観測点群と呼称します。

まず、NDT matchingにおいては正規分布を使って地図点群を大まかに表現します。図10のように地図点群をグリッドで分けた後、各正方形 (セル) の中の点群のばらつき度合いを元に正規分布で近似します (図11)。これにより、図12のように正規分布で近似された地図点群が得られます。

この一連の流れは数学的には次のように表記します。

1. 地図点群を長さ l のセルで分割し、セルの合計数を K とする
2. 各セル内の n 個の点群を2次元ベクトル $\mathbf{x}_{i=1..n}$ とする
3. 各セル内の点群の平均 $\mathbf{q} = \frac{1}{n}\sum_i \mathbf{x}_i$ を計算する
4. 各セル内の点群の分散 $\Sigma = \frac{1}{n}\sum_i (\mathbf{x}_i - \mathbf{q})(\mathbf{x}_i - \mathbf{q})^t$ を計算する

これにより、各セル内の点群は

$$p(\mathbf{x}) = N(\mathbf{x}|\mathbf{q}, \mathbf{\Sigma}) \propto \exp\left(-\frac{(\mathbf{x}-\mathbf{q})^t \sum^{-1}(\mathbf{x}-\mathbf{q})}{2}\right) \tag{1}$$

と表現できます。ここで、正規分布の平均ベクトルは $\mathbf{q}_{k=1..K}$、正規分布の分散ベクトルは $\mathbf{\Sigma}_{k=1..K}$ という風にそれぞれ K 個あることに注意してさい。これにより、各セル中の確率分布は $N(\mathbf{x}|\mathbf{q}_1, \mathbf{\Sigma}_1)$, $N(\mathbf{x}|\mathbf{q}_2, \mathbf{\Sigma}_2), N(\mathbf{x}|\mathbf{q}_k, \mathbf{\Sigma}_k), \ldots N(\mathbf{x}|\mathbf{q}_K, \mathbf{\Sigma}_K)$ と表します。

次に、点群マッチングを数学的に表現します。先ほどの節の点群マッチングの例は、図 13 のように観測点群を回転・並行移動して地図点群の一部との「距離」が小さくなるようにして行うことができます。

元の点群の位置を $(x, y)^t$、回転量を ϕ、並行移動量を $(t_x, t_y)^t$、回転・並行移動後の点群の座標を $(x', y')^t$ とすると、点群の回転・並行移動をする関数 T は次のように表現されます。

$$T : \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{2}$$

(1) の正規分布によって表現された地図点群と回転・並行移動して得られた観測点群との「距離」を表現するために、次の表記を導入します。

- 回転・並行移動量のパラメータをまとめて $\mathbf{p} = (pi)^t_{i=1..3} = (t_x, t_y, \phi)^t$ と表記する
- 観測点群中の D 個の点群を 2 次元ベクトル $\mathbf{x}_{i=1..D}$ とする
- $\mathbf{p}$ によって変換された $\mathbf{x}_i$ を $\mathbf{x}'_i = T(\mathbf{x}_i, \mathbf{p})$ とする
- $\mathbf{x}'_i$ が属するセルの正規分布の平均、分散を $\mathbf{q}_i$、$\mathbf{\Sigma}_i$ とする

観測点群中の各点群 $\mathbf{x}'_i$ についてそれが属する正規分布の確率を計算し、それを足し合わせた結果が最も大きくなる場合に「一番点群同士がマッチングしている」という事ができます。これは

$$\text{score}(\mathbf{p}) = \sum_i \exp\left(-\frac{(\mathbf{x}'_i - \mathbf{q}_i)^t \sum_i^{-1}(\mathbf{x}'_i - \mathbf{q}_i)}{2}\right) \tag{3}$$

と表す事ができ、NDT matching ではこの関数を最大化する事によって最適な回転・並行移動量 $\mathbf{p}$ を求めます。この関数の負 (つまり $-\text{score}(\mathbf{p})$) を取れば地図点群と回転・並行移動して得られる観測点群との「距離」として表す事ができ、この「距離」を最小化する問題に置き換える事ができます。NDT matching では、次に説明するニュートン法によってこの最適化問題を解きます。

2.2 ニュートン法による最適化

$\mathbf{p} = (p_1, p_2, p_3)^t$ についての関数 f を最小化する問題は、ニュートン法では以下のようにして解きます。まず、パラメータ $\mathbf{p}$ をランダムに選びます。次に $g_i = \frac{\partial f}{\partial p_i}$、$H_{ij} = \frac{\partial f}{\partial p_i \partial p_j}$ とし、

$$\mathbf{H}\mathbf{\Delta p} = -\mathbf{g} \tag{4}$$

を解きます。ここで得られた $\mathbf{\Delta p}$ を使って $\mathbf{p} \leftarrow \mathbf{p} + \mathbf{\Delta p}$ によってパラメータ $\mathbf{p}$ を更新します。更新された $\mathbf{p}$ を用いて再び $\mathbf{H}$ と $\mathbf{g}$ を計算し、(4) 式の計算、$\mathbf{p}$ の更新を行います。$f(\mathbf{p})$ が一定以下の値になるまでこの反復計算を繰り返します。

$\mathbf{H}$ はヘッセ行列なので、これが正定値行列である時には $f(\mathbf{p})$ が減少する方向に $\mathbf{\Delta p}$ が算出されます (二回微分が「正」であるから、と雑に捉えることもできます)。もし正定値行列でない場合には $\mathbf{H} \leftarrow \mathbf{H} + \lambda\mathbf{I}$ とし、更新後の $\mathbf{H}$ が正定値行列となるように λ を選びます。

このニュートン法を用いて、$-\text{score}(\mathbf{p})$ を最小化します。ここで簡単のために $\mathbf{q} = \mathbf{x}'_i - \mathbf{q}_i$ とし、最小化する目的関数 s を

$$s = -\exp\frac{-\mathbf{q}^t\mathbf{\Sigma}^{-1}\mathbf{q}}{2} \tag{5}$$

とします。まず $\mathbf{g}$ は以下のように求められます。

$$\begin{aligned} g_i &= -\frac{\partial s}{\partial p_i} = -\frac{\partial s}{\partial \mathbf{q}}\frac{\partial \mathbf{q}}{\partial p_i} \text{ (連鎖律)} \\ &= \mathbf{q}^t\mathbf{\Sigma}^{-1}\frac{\partial \mathbf{q}}{\partial p_i}\exp\frac{-\mathbf{q}^t\mathbf{\Sigma}^{-1}\mathbf{q}}{2} \text{ (合成関数の微分法)} \end{aligned} \tag{6}$$

二次元ベクトル $\mathbf{q}$ を三次元ベクトル $\mathbf{p}$ で微分した時の結果は q_i が微分によって消えるので T を $\mathbf{p}$ で微分した結果と同じになり、それは次のヤコビ行列で表されます。ベクトルをベクトルで微分すると行列になるのですが、詳しい導出は Matrix Cookbook[3] が参考になります。

$$\mathbf{J}_T = \begin{pmatrix} 1 & 0 & -x\sin\phi - y\cos\phi \\ 0 & 1 & x\cos\phi - y\sin\phi \end{pmatrix} \tag{7}$$

続いて、$\mathbf{H}$ は以下のように求められます。

$$\begin{aligned} H_{ij} &= -\frac{\partial s}{\partial p_i \partial p_j} = \frac{g_i}{\partial p_j} = -\exp\frac{-\mathbf{q}^t\mathbf{\Sigma}^{-1}\mathbf{q}}{2} \\ &\left((-\mathbf{q}^t\mathbf{\Sigma}^{-1}\frac{\partial \mathbf{q}}{\partial p_i})(-\mathbf{q}^t\mathbf{\Sigma}^{-1}\frac{\partial \mathbf{q}}{\partial p_j}) + (-\mathbf{q}^t\mathbf{\Sigma}^{-1}\frac{\partial^2 \mathbf{q}}{\partial p_i \partial p_j}) + (-\frac{\partial \mathbf{q}^t}{\partial p_j}\mathbf{\Sigma}^{-1}\frac{\partial \mathbf{q}}{\partial p_i})\right) \text{ (合成関数の微分)} \end{aligned} \tag{8}$$

二次元ベクトル $\mathbf{q}$ を三次元ベクトル $\mathbf{p}$ で二回微分した時の結果は q_i が微分によって消えるので T を $\mathbf{p}$ で二回微分した結果と同じになり、それは次の行列で表されます。

$$\frac{\partial^2 \mathbf{q}}{\partial p_i \partial p_j} = \begin{cases} \begin{pmatrix} -x\cos\phi + y\sin\phi \\ -x\sin\phi - y\cos\phi \end{pmatrix} & i = j = 3 \\ \begin{pmatrix} 0 \\ 0 \end{pmatrix} & \text{それ以外} \end{cases} \tag{9}$$

これで (4) 式に NDT matching の (3) 式を導入できたので、あとは (6),(7),(8) 式を計算して (4) 式に代入して $\mathbf{\Delta p}$ を計算し、$\mathbf{p} \leftarrow \mathbf{p} + \mathbf{\Delta p}$ でパラメータを更新し、再び (6),(7),(8) 式を計算し…という手順によって地図点群と回転・並行移動された観測点群の「距離」を徐々に小さくしていくことができます。「距離」を表す $-\text{score}(\mathbf{p})$ の値がある一定以下になったところでこの反復計算を終了し、得られた $\mathbf{p}$ を解とします。

実際にはあとはこの数式を C,C++などのプログラミング言語で実装してようやく実際に動くプログラムとなり、これは自動運転用のオープンソフトウェアである Autoware[4] のソースコード [5] 等が参考となります。

3　おわりに

いかがでしたでしょうか？今回はタイトル詐欺・サークルカット詐欺にならないように (本編とは全く関係が無いですが) ゴリゴリに大学レベルの数学を使うレベルで自動運転についての技術を解説してみました。実際に使われているキモい数式を見ていただいてロマン・厨二病感を味わってもらえれば、と思い概要の説明だけではなく数式レベルでの解説をしてみました。

今回は時間の都合上教科書のような形式での解説となってしまいましたが、次回作以降では漫画形式でキャラが解説する、という本来描きたかった内容にするつもりです。また次回作で取り上げてほしいテーマがありましたら、気軽に Twitter 等でお知らせください！

References

[1] Simon J. Julier and Jeffrey K. Uhlmann "Building a million beacon map", Proc. SPIE 4571, Sensor Fusion and Decentralized Control in Robotic Systems IV, (4 October 2001)

[2] Biber, Peter, and Wolfgang Straßer. "The normal distributions transform: A new approach to laser scan matching." Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on. Vol. 3. IEEE, 2003.

[3] Matrix Cookbook: https://www.math.uwaterloo.ca/ hwolkowi/matrixcookbook.pdf

[4] The Autoware Foundation: https://www.autoware.org

[5] lidar_localizer: https://github.com/Autoware-AI/core_perception/tree/master/lidar_localizer