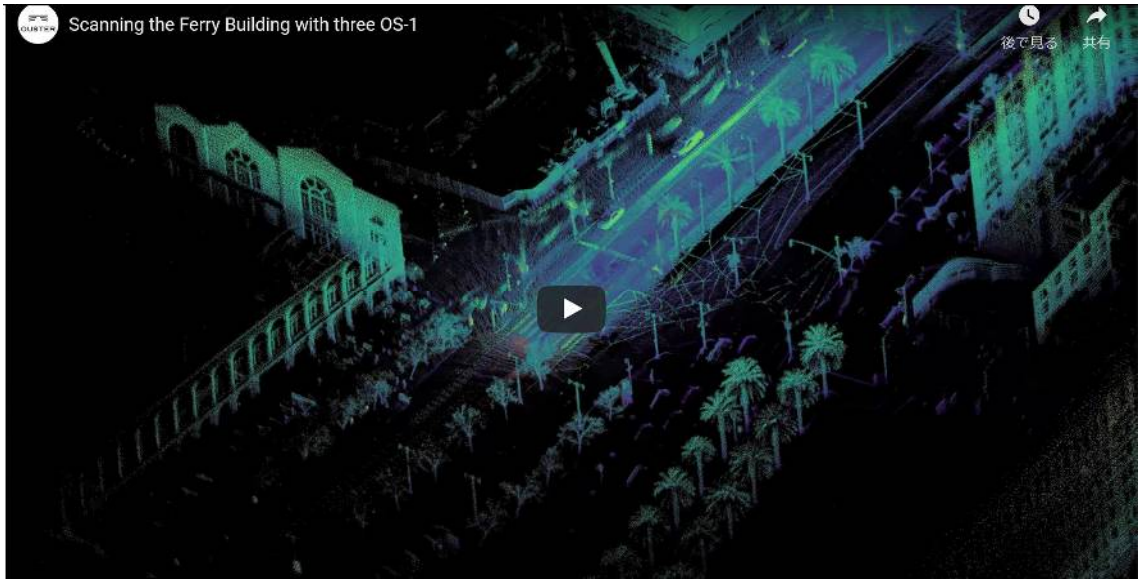
Ouster 3D センサーを用いたライダーマッピング

2019年4月1日　DANIEL LU

ライダーセンサーを用いてHDマップを作成する

ライダー技術を使った最良の使用例の１つはマッピングです。ライダーがあれば、身の回りのあらゆる物から 3D モデルを作り出すことが出来ます。



３台の OusterＯＳ－１を使って象徴的なサンフランシスコのフェリービルディングをスキャンする

（ビデオは、https://www.ouster.io/blog-posts/2019/3/29/lidar-mapping-with-ouster-3d-sensors）

ＳＬＡＭ(Simultaneous localization and mapping)を使ってＨＤマップを作成する

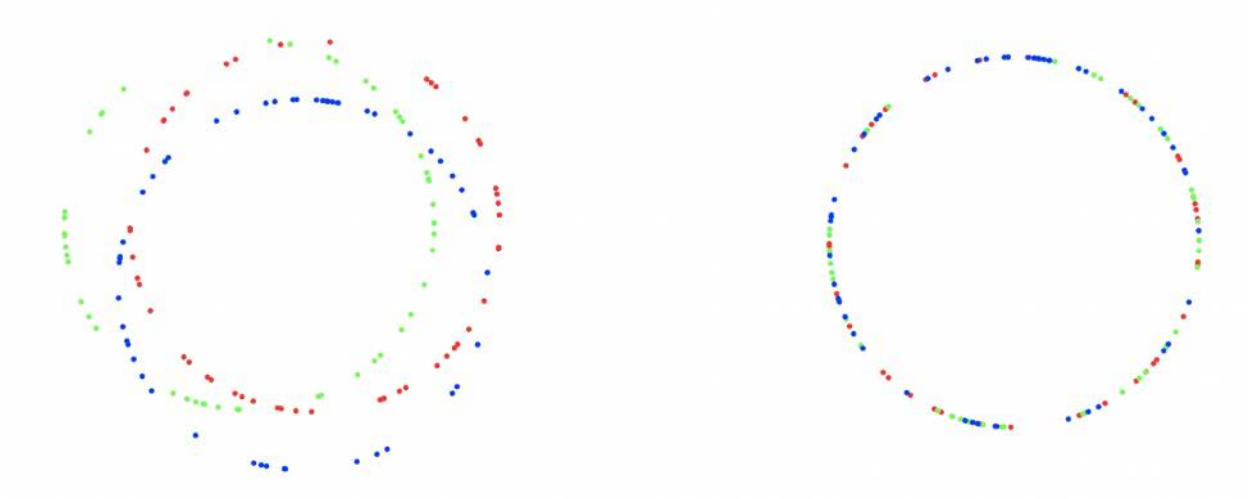
世界を 3D マップする際に必要なことはすべてにおいて、異なる場所で撮られたライダースキャン・データを合成（位置合わせ）することです。しかし、スキャンの位置合わせのプロセスは、それほど容易ではありません。これは、ライダーが、車のような移動するプラットフォーム上に設置されている場合に、特に難しいと言えます。

ライダーマッピングは過去、高精度のＧＰＳ慣性航行システム（INS）を含む高額なリグ（装置）に依存していました。GPS INS で計測された位置および方向を用いて、ライダー点群の位置合わせをおこなってきました。最近の GPS INS システムは非常に優秀です。--リアルタイムキネマティクス（RKT）付きの地上局修正信号のような技術を用いて、GPS INS システムは位置を数㎝の精度で正確に示すことが出来ます。これは、数十メーターでオフになってしまうかもしれない携帯電話のような安い GPS ユニットに比べて、はるかに優秀と言えます。残念ながら良い GPS INS システムは非常に高価です。しばしば、数百万円することがあります。

Ouster では、高価な GPS INS システムをマッピングには使用しません。OusterＯＳ－１の何倍もする GPS INS システムを購入することは、導入の際のＯＳ－１のコスト上の有利性を無駄にしてしまいます。Ouster はそのかわりに、ライダーデータ自体を位置合わせに使います。これは、*simultaneous localization and mapping* (SLAM)と呼ばれる技術です。これまでのコンピュータでは、ライダーデータを SLAM で実行さ

せるのに十分に速く処理できませんでした。しかし、SLAM 技術の最近の発展により、コンピュータハードウェアの高速化と同様に、現在これを行うことが可能になりました。

SLAM の原理は以下の通りです。センサーデータの最良の条件は、最もシンプルなものです。そして、センサーデータの最もシンプルな解釈は、全てが位置合わせされた時です[1]。



どっちがよりシンプルでしょうか：３つのサークル、あるいは １つのサークル？

どのように、ライダーデータの位置合わせを行なうか？

ポイントの位置合わせ

ほとんどのロボティクスアルゴリズムは、２つのステップに要約できます。まず、損失関数（loss function）あるいは、目的関数（objective function）と呼ばれる関数を定義します。これは、うまくいっていない時に、大きな値を取り、うまくいっている場合は小さな値を取ります。次に、上述の関数が最小化されるように状況を調整します。

ライダーの ２ つの点群データを位置合わせするために、以下のように目的関数を定義します。動かない現場シーン（static scene）を計測した点群Ｓに対して位置合わせを行うために、動きのある現場シーンを計測した点群Mを移動することを考えます。Mの全てのポイントに対して、Sの最近接のポイントを見出すことが出来ます。そして、目的関数は、Mの各ポイントとその対応するポイントの距離の２乗の総和です。点群Mを回転したり、並進（平行移動）させたりして、目的関数を最小化することを試みます。目的関数は、Levenberg-Marquardt [2]のような非線形最小自乗ソルバーの一種を用いて最小化できることがあります。ここで、最適化変数は、点群Mの回転と並進です。3D 上では、回転と並進は、合計で６自由度を持ちます。

コスト関数を最小化した後、点群Mは移動されます。これにより、対応する最近接ポイントが変更になるかもしれません。これに対処するため、最近接ポイントが変わらなくなるまで、この手順を繰り返します。このアルゴリズムは、iterative closest point (ICP) と呼ばれます。

移動式車両やロボットでは、スムースに移動し、どこにいるかをうまく予測できるので、closest point をうまく適用できると期待できます。初期の最近接ポイントは、最適化の後でも、最近接ポイントのまま残る可能性が高いのです。

実際には、ライダーのポイントは厳密に、互いに、直接的に相対しているわけではありません。そのかわり、ライダーは、内在する物理的サーフェスからサンプルポイントをスキャンします。これを考慮しながら、point-to-plane ICP を使用できます。1 つの最近接ポイントを探す代わりに、幾つかを探し、それらのポイントを平面フィッティングします。そして、ポイントペア間の距離を最小化する代わりに、ポイントと平面間の距離を最小化します。

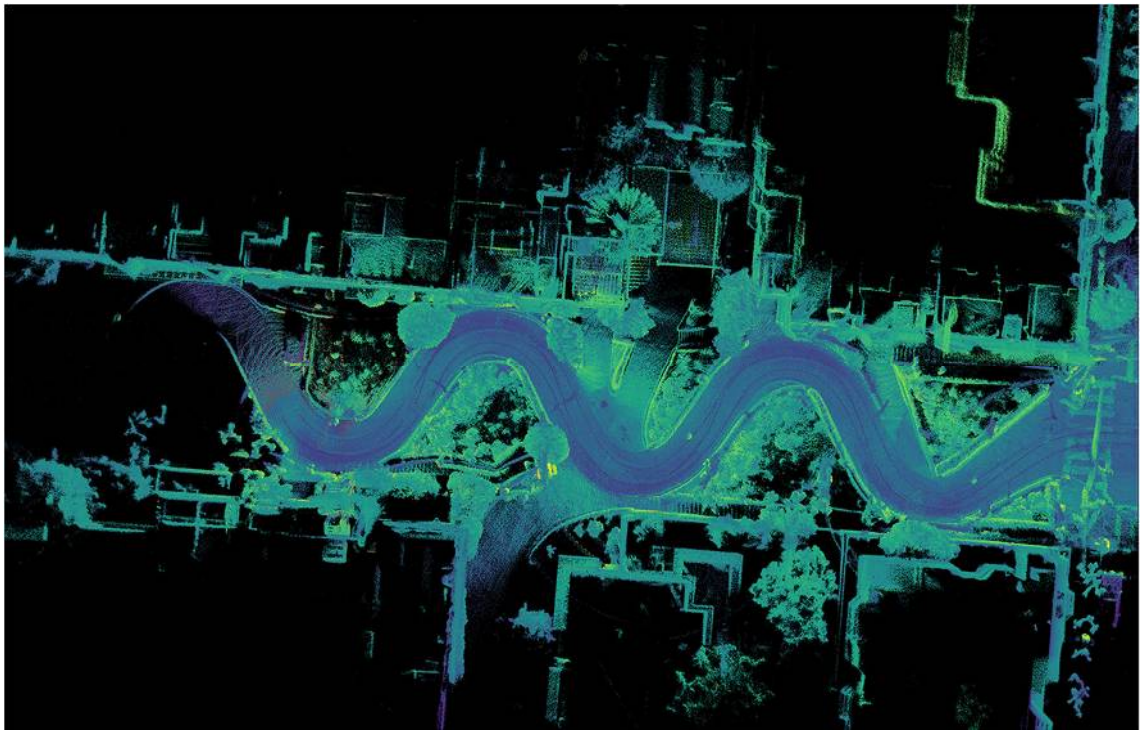
一般化した ICP、surfels を用いた方法、NDT（Normal Distributions Transform）のような他の進んだ方法も存在します。これら 3 つは、ポイント近傍のジオメトリを記述するのに多変量正規分布を用います。

他のセンサーの取り込み

全ての OusterＯＳ-１は、内蔵慣性計測装置（ＩMU）が装備されます。これは、どのスマートフォンにも内蔵しているような類似した低価格のセンサーです。高精度 INS ほど精度はありませんが、にもかかわらず、信じられないほど役立ちます。

ポイントの位置合わせと同様に、同じ方法で、IMU データを取り込むことが出来ます。まず、目的関数を作成して、慣性計測間（回転速度と並進加速度）の違いを評価します。次に、この目的関数を最小化するために、状態を最適化します。

最終的には、これら様々な目的関数を 1 つの大きなものに統合します。これは、tight coupling と呼ばれます。一般的に、センサーの数が多ければ、その分良好になります。



３つのＯＳ-１ライダーセンサーを使ってサンフランシスコのロンバード街をスキャンする

実際のパフォーマンス

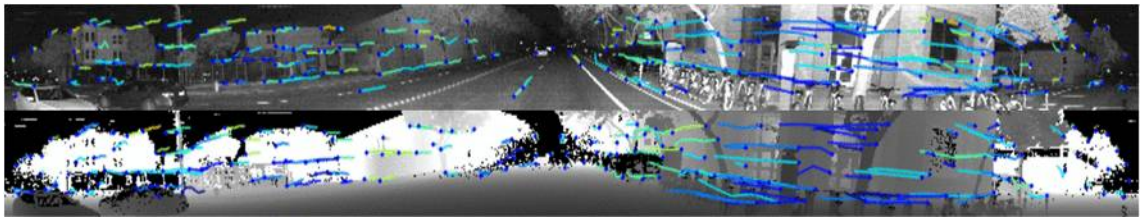
OusterOS-1 は 100 万ポイント/秒以上の出力を行い、最速のセンサーの 1 つです。これは、マルチビームフラッシュライダーデザインを採用したため実現しています。残念なことに、解像度を大きくすると、計算上の複雑さの問題が大きくなります。

我々の SLAM アルゴリズムは、1 台のデスクトップコンピュータ CPU で、同時に、1 つだけではなく、3 つもの OusterOS-1 デバイスをリアルタイムで操作できます。これは特筆に値します。

その秘密は、目的関数に対して全てのポイントを考慮せず、最も考慮すべきものの中から一部のみを考慮することです。これは、特徴抽出と呼ばれ、最も考慮すべきポイントは特徴点と呼ばれます。

特徴を抽出する 1 つの方法は、点群の最も平坦な部分のポイントを見つけることです。前述のとおり、1 つの目的関数の方策として、point-to-plane ICP があります。直感的に、平面に良くフィットするポイントは、平面自体にあるとわかります。ポイントの局所領域において、主成分分析を行うことで、全てのポイントに対する平坦さの指標を計算することが出来ます。そして、均一なポイントの分布を得るために、互いに近接しすぎる 2 つのポイントが存在しないという前提で、およそで言えば、千程度の最良の平坦なポイントを保持することが出来ます。

形状的な特徴抽出とは対照的に、画像ベースの特徴抽出法と併せて、ライダー強度を使用することも可能です。Ouster OS-1 が、2D カメラ似の画像を取得できるという事実を利用して、スーパーポイントのような深層学習（ディープラーニング）に基づく特徴抽出を利用できます。これにより我々のアルゴリズムはより汎用的になります。例えば、形状面の特徴抽出は、全ての平面が同じ方向を向くスムースなトンネルで失敗するかも知れません。しかし、トンネルの壁のテクスチャに起因する視覚的な特徴は存在します。



SLAM に対する特徴抽出としてスーパーポイントが使用されることがあります。

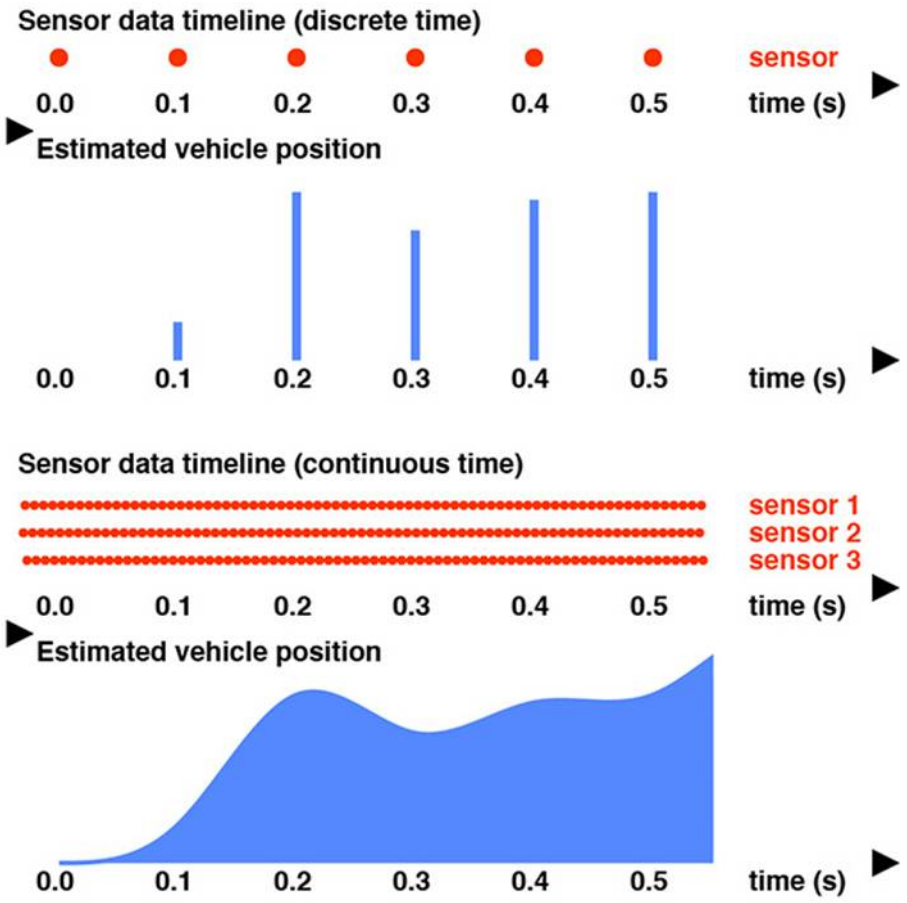
連続時間

従来の SLAM アルゴリズムは、1 つのフレームを次のフレームと位置合わせを行います。しかし、実世界ではほとんどのセンサーは、適切な速度では、個々のフレームを出力しません。例えば、慣性計測ユニットは、1000Hz でデータを出力するかも知れませんが、1 秒に 1000 回も、車両の位置を更新することは現実的ではありません。Ouster OS-1 は、より速い速度で出力を行います：1 秒の間に、20,480 回ごと、64 ポイントのグループを出力します。言い換え

れば、各 2048×64 のフレームが、1/10 秒の間に広がっていきます。ここでは、ピクセルの各コラムはわずかに異なるタイムスタンプを持ちます。

ライダーベースの SLAM に対する以前の方法は、単に、データの 0.1 秒の時間差を使ってフレーム間での点群位置合わせをおこなってきました。高速道路の速度では、この時間では、車両は 3m 進み、点群に歪みを生じるかもしれません。このため、この種の方法では、位置合わせの前に点群を“de-warp（平坦化処理）”するため、ホイールオドメトリのような外部センサーを使用することがあります。ホイールオドメトリは、ライダー計測ほど正確ではないため、その平坦化は決して完全ではありません。そして、不正確な結果になりがちです。

Ouster では、異なる速度で何とか稼働するマルチ高周波センサーに対応するめに、*連続時間的手法*を採用しています。点群の全てのポイントを同時に収集したかのように見せるため点群全体を平行移動したり回転したりする方式に代わって、Ouster は、車両の軌跡を時間の連続関数として扱います。



連続する軌跡を扱うには、2 つの主要な方法があります。最初のものは、ガウス過程（ここではその詳細には触れません）のような、ノンパラメトリックな方法です。第二は、より一般的なもので、軌跡をスプライン関数でパラメータ化するものです。よく使われるのが B-スプライン 3 次関数で、コンピュータ

グラフィクス業界において多くの用途が見受けられます[3]。B-スプライン 3 次関数を用いて、およそ 0.1 秒間隔離れた時系列的に連続するスプラインノットを保存することが出来ます[4]。各ノットの値は 6 次元になります：回転と並進の組み合わせです。そして、任意の時刻のなす軌跡は、最近接の 4 ノットの重み付け総和になります[5]。この方法を用いて、1 秒ごとに数千ではなく、10 の変数のみによる合理的な変数最適定量化を行うことが出来ます。同時に運動による歪みは問題でなくなります。

最適化に関しては、2 つのフレームを互いに位置合わせするのではなく、例えば、0.5 秒の間で“移動するウィンドウ”内の全てのポイントを考慮します[6]。ウィンドウは 0.1 秒ごとに前進（スライド）します。直前の 0.5 秒の全てのポイントのことを考慮してください。それぞれに対して、空間的な最近接点を幾つか見出すことが出来ます。ただし、時間的にあまり近接しているポイントはないとします。これにより依然と同様に最小化できる距離のセットを得ることが出来ます。現在では、すべての点群の回転と並進を更新する代わりに、関連する各ノットを代用し更新しています。そのメリットとして、B-スプライン 3 次関数は微分可能なので、当然、IMU 目的関数を導き出すことが出来ます。車両軌跡の微分不可能な数式表現を用いる場合、これはチャレンジング（有効になりうる）かも知れません。

ループを閉じる

上記では、直近の 0.5 秒のスライドウィンドウを用いた軌跡の評価について説明してきました。しかし、高精度センサーを用いた高機能 SLAM アルゴリズムであっても、結果として生じるランダムな不確実性の累積の影響を受けやすいと言えます。例えば、この方法で見出された推測軌跡は、車両の真の軌跡から、どうしても僅かにドリフトを生じます。結果として、非常に大きなループで移動する場合、車両が実際に正確にスタート地点に戻ってきたとしても、車両の推測軌跡は同じ地点で終了しない可能性があります。これは、ループ閉じ込み問題として知られています。

Loop closure の問題はまだ活発に研究の行われている分野です。Ouster では、位置推定とラフな位置合わせに高速フーリエ変換に基づく技術を道いて Loop closure の最先端のソリューションを提供できるよう取り組んでいます。その場合、大きな点群の位置合わせのバッチ最適化処理が実行され、複数の車両から、シームレスで、1 つの 3D モデルへ、データのフュージョンが行われます。バッチ最適化は完全にコンテナ化され、クラウド・インフラ上で実行され、拡張性に富むよう設計されていきます。

その結果として、簡潔明瞭な詳細な 3D マップが得られます。より多くの車両が同じ領域を走査すれば、より鮮明で高精度なデータになります。

結論

大規模な 3D マッピングは容易ではありません。Ouster は、より大きなスケールで、より

高い解像度で、そして、どこよりもより低コストで、マッピングを実現します。Ouster のソフトウェア重視のマッピング戦略は、高精度 GPS システム、ホイールオドメトリ、高価なジャイロスコープの必要性を否定します。さらに、Ouster OS-1 のマルチビームフラッシュライダーデザインは、高密度点データを出力できる他のどのセンサーよりも非常に低価格、小型、軽量です。これは、Ouster のマッピングシステムが、ドローンであれ、車両であれ、ロボットであれ、あらゆるプラットフォームに、容易かつ廉価に展開可能なことを意味します。

---

[1] A point cloud registration objective function may be shown to be equivalent to entropy, as seen in section 2.2 of Tsin, Y., & Kanade, T. (2004, May). A correlation-based approach to robust point set registration. In European conference on computer vision (pp. 558-569). Springer, Berlin, Heidelberg.

[2] https://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm

[3] Piecewise linear functions and Hermite splines are also popular

[4] ノットの間隔は、車両の運動状態に依存します。車両の速度が、0.1 秒の間に大きく変わることはありそうもありません。また、ノットを追加することは、計算上もその分高価になります。一般に、ノット数の 3 乗に比例して時間も必要になります。

[5] この場合、“総和”には、リー群の数学機械を必要とします。3D のリジッド変換は、非ユークリッドである特殊ユークリッド群である からです。

[6] 最適化ウィンドウが長くなるほど、結果は高精度になります。ロボティクス用途に関しては、低遅延の車両の位置、方向評価も重要になります。