

# 3DLiDAR を活用したナビゲーションシステムの構築

中村 勇太<sup>1</sup>, 吉田 侑樹<sup>1</sup>, 青木 修平<sup>1</sup>, 村上 雄飛<sup>1</sup>

<sup>1</sup>Abudori Lab.

## 1 はじめに

つくばチャレンジは 2024 年のロボット大賞に選出 [1] された。つくばチャレンジは 18 年間、課題や取り組みを変えながら日本のロボット技術の進歩に大きく貢献してきた。このつくばチャレンジで完走しているチームの技術力は非常に高い。

この高い技術力を学ぼうと思っても、初学者にとって超えるべきハードルも高い。自律ナビゲーションシステムはたくさんの機能が有機的につながっており、一つの機能を理解しても達成できない。複数の機能を、精度よく、再現性の高い状態に持っていくことは、単なるこの分野の理解だけでなく、経験も必要であると感じた。

屋外環境のナビゲーションで、3DLiDAR によるセンシングが効果的であることは、感覚的にわかる。実際にやってみようと思うと、ROS のチュートリアルや解説記事などでは、2DLiDAR によるナビゲーションを扱われていることが多いが、3DLiDAR の記述は多くないことがわかった。そうなると、現場で実際に手を動かして知見を得る方法しかない。このつくばチャレンジ 2024 で 3DLiDAR を使ったナビゲーションシステムを作ることで、学ぶこととした。

また、現在の日本語の Web 上では、3DLiDAR を用いたナビゲーションシステムの知見が少ない。このつくばチャレンジ 2024 で構築したノウハウを各分野ごとに公開することで、少しだけでも貢献できると考えた。必ずしも、正解ではないが、失敗も含めてブログに綴り公開している。詳しくは <https://www.abudorilab.com/entry/2024/07/14/132220> を参照していただきたい。

Abudori Lab. チームでは、以下の観点でつくばチャレンジ 2024 に参加した。

- 3D 地図の構築方法を知る
- 3D 地図による自己位置推定方法を知る
- 3DLiDAR による障害物検知を知る

- 3DLiDAR 使用しながら 2D ナビゲーションの方を知る

Abudori Lab. チームの本走行では確認走行区間をクリアできなかった。しかし、決められたルートを走行することだけであれば、自己位置を喪失することなく、障害物も発見し走行することができた。

このレポートはゼロからつくばチャレンジに参加する人に必要なことがわかるレポートにしたいと考える。Abudori Lab. が用意したロボットやソフトウェア構成を網羅的に紹介する。

## 2 システム

### 2.1 ハードウェア

自律走行を行う機体で必要な要素は大きく分けると下記 5 点となる。

- センサ
- 走行装置
- 計算機
- バッテリー
- シャーシ

この章では、搭載したハードウェアがどのように使用されたか説明する。

#### 2.1.1 自律走行ロボット:Penguin

今回、自律走行を実現する基本的な構成を持つつ、不整地の走行も可能な機体とすることをコンセプトに機体を作成した。作成した機体とその搭載した機器を、図 1 に記載する。この章では、各要素について詳細を説明する。

#### 2.1.2 センサ

Penguin には、エンコーダ、IMU、LiDAR の 3 種類のセンサが搭載されている。

エンコーダはモータの回転数を計測するもので、どのくらいモータが回転したか累積を数えることでロボットのホイールオドメトリを計算する。ただ

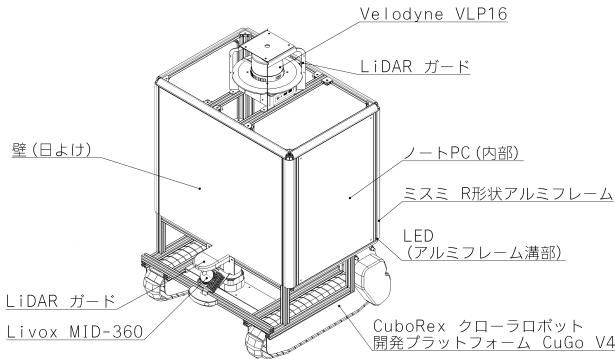


図1 Penguin

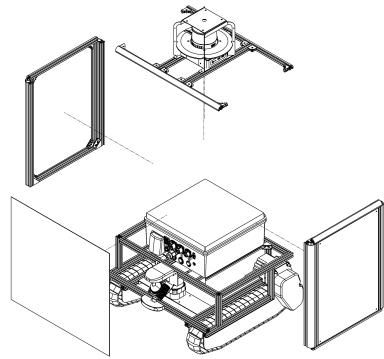


図2 展開図

し、スリップや量子化誤差などで誤差が蓄積するためこの値をそのまま使用することはできない。後述の LiDAR を使って外界の情報を得ることで補正して使用した。

IMU はロボットの X, Y, Z 方向の加速度と Roll, Pitch, Yaw 方向の角加速度を取得できる。この値を累積することで、ジャイロオドメトリを計算することができる。ホイールオドメトリのようにスリップすることはないが、段差などの振動の影響を強く受ける。今回は、SLAM アルゴリズムでの位置補正を使用した。

LiDAR は距離を知ることができるセンサである。今回搭載した LiDAR は 3DLiDAR で、ロボット周辺の形状を 3 次元の点群として得ることができる。3DLiDAR は自己位置推定用のものと、障害物認識用のものの 2 つを搭載した。

自己位置推定を行うための LiDAR センサとして、Velodyne 社の VLP-16[2] を搭載した。VLP-16 は 100m という長距離の測定が可能であるため、つくば市庁舎やつくばエクスプレスの建物を常に観測できる。ロボット近傍で人だからりができるて周囲の景気が見えづらくなても、遠くの景色を用いることで自己位置推定を維持しやすくなる。なるべく周囲の物体によって LiDAR の光を阻害されないように機体の最上部に搭載した。また、非常に高額なセンサであるため、転倒したり、ものが当たって傷つかないように金剛製のガードも合わせて設置した。

障害物を検出する LiDAR として、Lixox 社の MID-360[3] を搭載した。MID-360 は VLP-16 のような長距離の物体を測定することはできないが、非常に密な空間測距ができる。また、FoV も 60 度と広く、ロボットに衝突しそうな近くの物体を精細に捉えることができる LiDAR である。

障害物検知用の LiDAR は、接近する物体を検知す

るため、機体の下部に設置し、35 度下向きに傾けて設置した。これにより、ロボットの前方 250mm の地面を測距できるため、衝突しそうになるギリギリまでの距離の物体を測定できる。また、今回の機体は後進を行わないため、障害物の検出範囲は前方向のみとした。障害物検出用の LiDAR も、光学窓への傷を防ぐため金属製のガードを上下に設置した。

### 2.1.3 走行装置

走行装置には、CuboRex 社のクローラロボット開発プラットフォーム CuGo V4[4] を使用した。不整地走行に適したクローラを有するプラットフォームを使用する事で、不整地の走破性向上と開発の高速化を図った。

### 2.1.4 計算機

ロボットの制御を行う PC として、ASUS 社の ROG Strix SCAR 15 G532LWS[5] を搭載した。ノート PC を搭載することにより、ロボットを駆動する電源系列と PC を駆動する計算機の電源分割を実現した。

モータの制御には、CuboRex 社製モータドライバに付属している Raspberry Pi Pico[6] を使用した。

### 2.1.5 バッテリー

機体下部に、24V20Ah の LiFePO4 バッテリーを搭載した。本機体は不整地走行の可能性を考慮し、LiPo バッテリーより安全性の高い LiFePO4 バッテリーを採用した。

バッテリーは機体下部に設置することで、重心を下げ、機体の安定性を向上させた。バッテリーから供給される電源は、ロボットに搭載した電源分配基板により適切に切替・分配・変圧し、PC を除く各機器に接続した。

### 2.1.6 シャーシ

屋外でのデバッグ作業を想定し、日よけの壁を搭載した。ロボットは図 2 のような 5 ユニットで構成し、各部を計 10 本のボルトで分解できる形とした。分解箇所には付当てを設置し、複数回の分割、組立を実施しても再現性のあるシャーシとなる形とした。

構造部材にはミスミ社 R 形状アルミフレーム [7] を使用することで、衝突しても対象に危害を加えにくい形状を実現した。突起部・巻き込みが発生しうる箇所には、樹脂製のカバーを搭載した。

ロボットの状態表示のため、アルミフレームの溝部に LED を搭載した。この LED の色でロボットが自律走行状態か、操作者による操作を実施している状態かを一般通行者に対して示した。

## 2.2 ソフトウェア

ロボットを走行させるためのソフトウェア構成は以下の通りである。

- 地図作成
- 自己位置推定
- 障害物認識
- 経路計画
- 経路追従

自律走行を実現するために、上記のタスクを同時に満たすナビゲーションシステムを構築した。図 4 にナビゲーションシステムの概要を示す。このシステムを構築するために OSS である ROS 2[8] を使用した。このレポートでは、上記のそれぞれのタスクについてどのようなアプローチをしたか述べる。

本システムで自律走行をするためには、まず、走行する範囲の地図を作成する。現実の場所とリンクした地図をロボットに持たせることでロボットが目的地と現在地の相対位置を把握することができる。このロボットに持たせる地図の精度が後述の自己位置推定の精度に大きな影響を与えるため、効率的で精度の高い地図を作成することが大事である。この地図作成は第 3 章で説明する。

次に、ロボットの現在位置を推定する。LiDAR などのセンサ情報から自分自身が地図上のどの位置にいるのかを推定する。現在位置が分かれば、目的までの向かう経路を計算することができる。正確な位置情報を維持し続けるために回転式 3DLiDAR のセンサ値とオドメトリ情報を使用して自己位置推定を行

う方法を第 4 章で説明する。

自己位置推定ができるても、実際には障害物があつてたどり着かないことがほとんどである。向かいたい場所までの経路上に障害物がどのようにあるのか反映させる。ロボット直近の広範囲で高密度な点群を取得できる 3DLiDAR を使用することで、2DLiDAR では見逃しがちな細長い物体を見逃さず、坂道などの検知したくない物体を除外することができた。この障害物検知手法は第 5 章で説明する。

ロボットの現在位置と障害物を知ることができると、目的地までの障害物を避けた経路を計算することができる。実際の自律走行では、スタート地点から本走行のゴール地点までの経路を 1 度に計算しない。数メートルおきに小さなゴール地点を作成し、そこに到達することを繰り返す。経路を計算した後には、ロボットがこの経路を正確にトレースするようにモータの出力を制御する。この時、計算した理想的な経路を忠実に走行すれば良いとは限らない。ロボットハードの都合で急停止や急旋回が物理的に実現できなかったり、とっさに現れた障害物を回避する必要があるためである。障害物を回避しながら、必要十分に経路をなぞる制御を経路追従という。第 6 章では、経路計画と経路追従を行うナビゲーションについて説明する。

最後に、本走行の結果と今後の展開について述べる。

## 3 地図作成

現代の多くの自律走行ロボットは、行動範囲分の環境地図を必要とする。ロボットが環境地図を持つことにより、既知の環境内で自由に行動することを目指す。

物流倉庫やオフィスビルなどの環境では 2D 地図を使用しても充分に効果を発揮する。しかし、つくばチャレンジのようなひらけた場所であったり、交通の往来があつたりする屋外環境であると 2D 地図の使い方によっては、性能が不十分になることがある。このつくばチャレンジ 2024 では、回転式 3DLiDAR を利用し 3D 地図を作成した。3D 地図は道路や建物など特徴が異なる環境や、人だからでロボット周辺に障害物がたくさんあるような環境であってもロバストに自己位置推定を続けることができた。

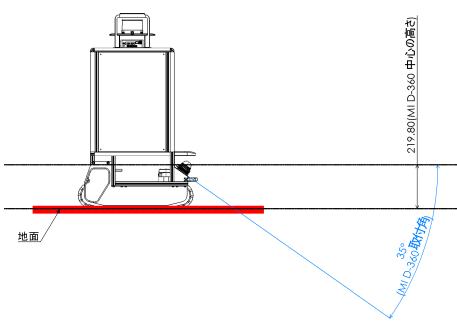


図 3 障害物検知用の LiDAR 取り付け位置

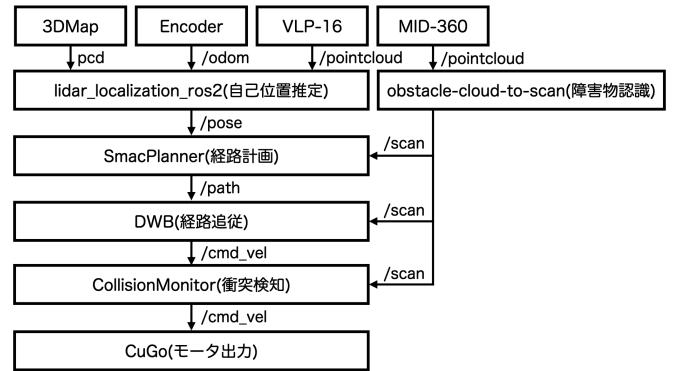


図 4 ナビゲーションシステムの構成

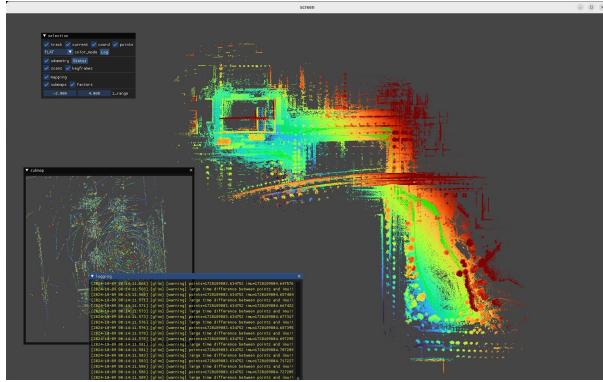


図 5 作成した 3D 地図

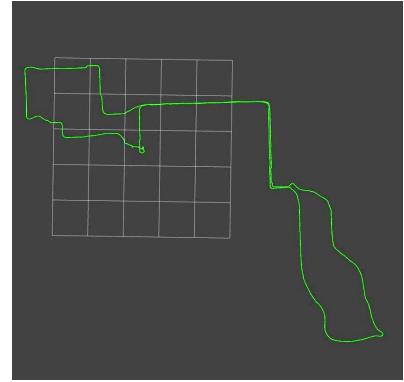


図 6 自己位置推定した軌跡

### 3.1 GLIM

地図作成には、回転式 3DLiDAR で、GLIM[9] を用いて作成した。GLIM はさまざまな 3DLiDAR に対応した SLAM 手法である。IMU やオドメトリの値と LiDAR の値を密結合することにより、正確な SLAM を行うことができる。回転式やソリッドステート式の 3DLiDAR だけでなく、深度付きカメラにも対応する。IMU やオドメトリの値の有無も選ぶことができるため、利用の敷居はとても低い。

作成したつくばチャレンジの走行範囲全域の地図は図 5 の通りである。つくばチャレンジの現場でロボットのセンサを起動し、スタート地点からゴールまで走行した時のセンサデータを入力し地図を作成した。

### 3.2 パラメータチューニング

GLIM のチューニングとしては以下の点を変更した。

- *k\_correspondences*
- *voxel\_resolution*

*k\_correspondences* は一つのサブマップを構築す

る際に利用する点群のスキャンの数である。回転式 LiDAR の 16 ラインのものだと点群が疎であるため、この数を上げないとマッチングに必要な点群数に至らないことがある。32 ラインの LiDAR はデフォルトから変更なし、16 ラインの LiDAR は数を 30 程度に設定するとうまく地図が構築されることを確認した。

*voxel\_resolution* は 3 次元空間の点群を配置する解像度である。つくばチャレンジ 2024 のようなキロメートルオーダーでは、デフォルトの 0.1m 四方だと、GPU メモリ 8GB が飽和した。今回は 0.25m 四方に設定したら、研究学園駅前公園を含むすべての範囲で地図を作成することができた。

最後に GLIM のオフラインツールでループクローズを設定する。SLAM を実施した後、オフラインツールで再度開くと、各サブマップ同士の位置関係をグラフィカルに確認することができる。このツールで、スタート地点、往路と復路が重なるパイルオン地帯、横断歩道の待機場所で再度ループクローズをかけることで、より正確な地図に修正された。

これらの作業を通じて、つくばチャレンジ全域で自己位置推定ができる地図ができた。つくばの 3D 地図は GoogleDrive の <https://x.gd/22G04> で公開し

ているので、来年同じ構成で使用したい方は使うと良い。

## 4 自己位置推定

前章で作成した 3D 地図に対して、ロボットの自己位置を推定をする。3D 地図を利用することで、つくばチャレンジ 2024 のコース全域でほとんど破綻することなく位置を推定することができた。

自己位置推定手法には、OSS の lidar\_localization\_ros2[10] を利用した。

### 4.1 lidar\_localization\_ros2

lidar\_localization\_ros2 は ROS 2 で動作する自己位置推定アルゴリズムである。3DLiDAR の他に IMU やオドメトリのセンサ値を利用してよりロバストに推定することができる。前章で作成した 3D 地図を pcd ファイルに変換して読み込み、コンフィグで有効にしているセンサを起動すれば、ロボットの自己位置を維持してくれる。

### 4.2 パラメータチューニング

lidar\_localization\_ros2 のチューニングとしては以下の点を変更した。

- *ndt\_resolution*
- *use\_odom*
- *use\_imu*

*ndt\_resolution* を小さく設定すると、自己位置が非常に滑らかにプロットされる傾向があった。一方で、マッチングが失敗する確率が高くなり全く別の位置に吹っ飛んで破綻することが多かった。初期値が 2.0 であったが、5.0 にすることで、破綻を防ぐことができた。ただし、プロットが荒く 0.1 0.3m ほど一瞬ずれることがあった。ずれることがあっても、すぐに復帰し戻すことと、ロボットの走行速度が遅いことから悪影響なしと判断し、安定性を重視したパラメータに設定した。

*use\_odom* と *use\_imu* でホイールオドメトリと IMU の値を自己位置推定に反映できる。

ホイールオドメトリを有効にすると前述の急激な自己位置推定変化を抑制することができる。モータの回転数は突然極大な値を出力することがなく安定した値を出力するためである。今回はホイールオドメトリの値を有効に設定した。

IMU はホイールオドメトリのような段差などの不

整地によるスリップの影響を受けにくい。今回は安価な IMU を使用していたため、ある方向のバイアスが強く作用した。LiDAR によるスキャンマッチングと誤マッチングを抑制するホイールオドメトリが働いていたが、IMU が常に強い横移動を示し続けていて、自己位置に悪影響を与えていたため IMU の値を無効に設定した。

つくばチャレンジ期間中、走行中に自己位置推定を喪失し、次にどこに向かえば良いかわからない状態が発生することなく安定して動作し続けることができた。

## 5 障害物認識

自己位置とゴール座標が分かれればロボットが進むべき経路を計算できる。ただし、これだけでは現実にある衝突してはいけない障害物を考慮していない。LiDAR などのセンサを使って障害物を認識し、それを地図に反映させることで障害物を考慮した経路を計算することができる。

多くのロボットでは、2DLiDAR を障害物認識に使用している。2DLiDAR を使用すると、ロボットに設置した高さの平面でしか障害物を認識することができない。それによって、背の低い障害物が認識できなかったり、パイロンなどのように地面に向かって太くなる形状のものを本来より小さな障害物として認識したりすることがある。また、坂道のようにロボットが傾いている時は、地面を障害物として認識することがある。2DLiDAR は工夫しないと、認識したい障害物を見逃したり、地面のような障害物でないものも障害物として認識することができる。

Abudori Lab. チームでは、3DLiDAR を使うことで、小さな障害物を認識し、坂道などの地面はアルゴリズムで除外することで、上記の問題を回避した。このアルゴリズムは OSS で obstacle-cloud-to-scan[11] として公開している。

### 5.1 obstacle-cloud-to-scan

このパッケージは、3DLiDAR でロボット直近の 3 次元点群を取得し、障害物のみを抜き出し 2DLiDAR の scan として出力する。

簡単な手順としては以下の通りである。

- 点群をダウンサンプリングする
- LiDAR 点群の傾きを設置角度分回転し水平にする
- 法線を各点ごとに計算する

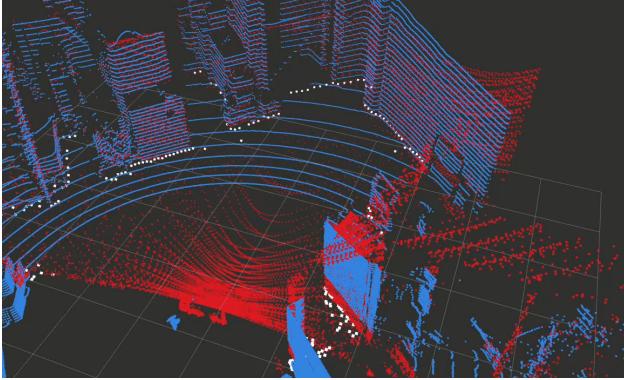


図 7 3DLiDAR から認識した障害物の scan. 赤い点群から障害物を認識し、白い 2DLiDAR の点群に変換した。

- ・法線ベクトルが水平に近い点群を抜き出す
- ・抜き出した点群を scan に変換する

法線ベクトルに注目することで、坂道や乗り越えられる小さな段差は障害物と認識しないという判断ができた。壁や大きな段差は、法線ベクトルが水平方向に向くため障害物として認識することができた。

障害物と坂道を判断する計算を単純にするために、法線の Z 方向で閾値で区切った。つくばチャレンジでは、斜度 25 度に傾いた時の Z 方向の高さを閾値に、それよりも Z の値が小さいものを障害物が映った点群と判断した。この閾値処理のみで、細い鉄パイプやパイロンの根本、パイロンにかかった棒も障害物として判定できた。苦手な物体として、背のひくい水平な障害物は検出することが難しい。身近にあるものとしては、平台車や高さが 20cm 程度のダンボールなどが挙げられる。幸い、つくばチャレンジの環境にはなかったため、障害物認識で苦戦することはなかった。

出力した点群を OSS の `pointcloud_to_laserscan`[12] パッケージに入力し、2DLiDAR の scan トピックに変換した。この scan トピックを Navigation2 に入力することで、広く使われる 2DLiDAR のサンプルをそのまま動かすことができた。

## 6 ナビゲーション

ROS 2 では、自己位置推定、経路計画、経路追従の手法が詰め合わせとなっているパッケージとして Navigation2[13] がある。この Navigation2 パッケージを利用することで、それぞれのタスクを処理するアルゴリズムを利用できる。今回は 3 次元地図で自己位置推定を行なっているので、経路計画と経路追従のアルゴリズムのみを利用した。

### 6.1 経路計画

経路計画は現在の位置座標からゴールとなる座標までの区間でロボットが到達可能な経路を計算するタスクである。経路計画は 2 次元の占有格子地図上で計算した。ロボットに持たせている 3D 地図は、単純に地図上の座標を知ることのみで利用している。この 3 次元の自己位置座標から、2 次元空間で利用するために、X,Y,Yaw の値をそのままコピーするだけで 2D に転写した。

経路計画を行う時は、いきなり最終目的地への経路を計画するのではなく、数メートルおきに小さなゴールを用意し、到達したらまた次の小さなゴールに向かって経路計画を行うようにした。これはゴールへの距離を短くすることにより経路計画にかかる計算時間を削減するためである。この小さなゴールを Waypoint と呼ぶ。

Waypoint をロボットの状況に合わせて通知する `penguin_nav` パッケージ [14] を作成した。`penguin_nav` は Waypoint を記述した CSV ファイルを読み込み、Navigation2 の SimpleCommanderAPI を利用して Waypoint を送信するパッケージである。Waypoint は、X, Y, Yaw の姿勢情報とゴールでの振舞いとして表現される。ゴールでの振舞いには「一時停止」を指定でき、これにより交差点手前での停止を実現する。また、経路計画の計算時間削減のため、経路計画に使用される Global Costmap の範囲を現在のロボット位置とゴールが含まれる矩形に狭める処理を行っている。

`penguin_nav` から受け取った Waypoint までの経路を計算する。Navigation2 では、さまざまな経路計画アルゴリズムを利用することができます。今回は SmacPlanner[15] を利用した。SmacPlanner は Navigation2 の Plugin に実装されているため、Config ファイルで SmacPlanner を選択するだけで利用できる。

SmacPlanner はロボットの形状を考慮した経路を計算する。したがって、自ロボットの投影面積であるポリゴンを指定するだけで、無茶な経路は選ばれなくなる。また、最小回転半径も指定できるため、なめらかな円弧の走行経路を利用できる。クローラは非常に抵抗が大きく、その場旋回がとても大変な駆動方法である。そのため、最小半径を 0.4m に設定した。これにより、走り出しのぎこちなさを抑えつつも、必要に応じて急旋回もできる曲率と

なった。

また、Waypoint を記述した CSV ファイルは別途用意する必要があるが、これを人手で作成するのは非常に手間がかかる。そのため、Waypoint 作成を支援するツールとして waypoint-resampler[16] と waypoint-viewer[16] を作成した。waypoint-resampler は、自己位置推定による軌跡を自動でダウンサンプリングするツールである。ダウンサンプリングの例を図 8 に示す。waypoint-viewer は、Waypoint の可視化を行うツールで、Waypoint の位置を確認することができる。可視化の例を図 9 に示す。これらのツールを使用することで Waypoint の作成を効率化した。

## 6.2 経路追従

前節の SmacPlanner で計算された経路をロボットがどのように再現するかを計算するタスクを経路追従という。Navigation2 では、経路追従アルゴリズムもすでに実装されている手法から選んで使用することができます。今回は、DWB を利用した。

DWB はロボットが再現可能な速度や加速度の範囲の中で最も良い速度ベクトルを選択する。このとき、自ロボットの最高速度と最低速度と加速度を設定できる。クローラの制御の場合、抵抗が非常に大きく初動に大きなエネルギーを必要とするため、最低速度を大きく設定し、すぐに停止できるように減速加速度を大きく設定した。

また、障害物が急に現れた場合の停止や、道が狭い場合に減速するために Navigation2 の CollisionMonitor を使用した。これは障害物との距離をチェックし、速度制限をかけるアルゴリズムである。

以上のアルゴリズムによって、ロボットのモータをどのように駆動するかを計算することができた。この速度ベクトルの通り、モータを回すことでロボットをより目標地点に近づくように走行することができる。センシング→自己位置推定→障害物検知→経路計画→経路追従→移動→センシング…と、この一連の操作を繰り返していくことで、Waypoint までの移動を達成する。最後の Waypoint は到達したいゴールとなり、長距離の自律走行を実現することができる。

## 7 本走行

つくばチャレンジ 2024 の本走行では、市庁舎裏の細い通路でロボットを回避し、元の経路に再計画

できずに 180m 地点でリタイアした。それまでの走行は、自己位置を見失うことなく、障害物を未検出になることなく動作した。

障害物からは距離を 1m 程度離れるように設定していたため、狭路で非常にゆっくり進んだ。そのため、図 10 のように後続のロボットが渋滞してしまった。

その後、図 11、12 のように、ロボットが追い越し、追い越され、経路上で詰まってしまった。このロボットは設定したルートに忠実になぞる機能しか搭載していないかったゆえに、迂回することができず、図 13 のように障害物を回避したあと復帰できなくなってしまった。

本年は確認走行区間内でリタイアしたが、決められた座標の通りに走行する機能までは 3DLiDAR を使って実現することができた。

## 8 結論

つくばチャレンジ 2024 では、3DLiDAR を利用したナビゲーションシステムを構築することができた。3DLiDAR のナビゲーションシステムは文献が少なく構築することに苦労したが、最低限の機能を揃えることはできた。

このレポートでは、ハードウェアから自律ナビゲーションシステムの各機能のおおよその仕組みを述べた。来年初参加を考えている人の参考になれば幸いである。環境構築の方法や実際の詳細の使用方法などは引き続きブログで解説していく予定である。

次年度では、決められたルートを走行するまでの意思決定機能を追加で実装する。完走するためには、不測の事態でも、認知判断行動することが必須である。完走にむけてナビゲーションシステムをブラッシュアップしていく予定である。

## 参考文献

- [1] 日本機械工業連合会. 「第 11 回 ロボット大賞」受賞一覧, 2024. <https://robotaward.jp/winning/>.
- [2] OUSTER. VLP 16. <https://ouster.com/products/hardware/vlp-16>.
- [3] Livox. MID-360. <https://www.livoxtech.com/jp/mid-360>.
- [4] CuboRex. クローラロボット開発プラットフォーム CuGo V4, 2023. <https://cuborex.com/product/?id=9>.
- [5] ASUS. Rog Strix SCAR 15 G532LWS, 2020. [https://jp.store.asus.com/store/asusjp/ja\\_JP/pd/ThemeID.4850018000/productID.5425083800](https://jp.store.asus.com/store/asusjp/ja_JP/pd/ThemeID.4850018000/productID.5425083800).

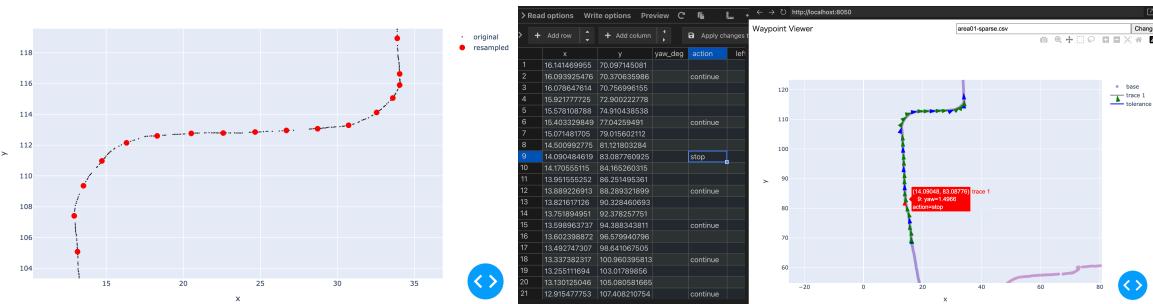


図 8 waypoint\_resampler によりダウンサンプリング。黒点が入力の自己位置推定結果で、赤点がダウンサンプリングした点列



図 10 狹路走行中、走行と停止を繰り返した。後続のロボットが渋滞となってしまった



図 12 追い越したロボット同士で接触しそうになりしばらく停止

図 9 waypoint\_viewer. 左の画面は外部の CSV エディタ



図 9 waypoint\_viewer. 左の画面は外部の CSV エディタ



図 11 狹路が終わった段階で一時停止した時に、後続のロボットが追い越しを始める

図 13 経路上にいるロボットを避けようとして壁に接近。復帰の経路作成できず断念

- [6] Raspberry Pi Foundation. Raspberry pi pico. <https://www.raspberrypi.com/products/raspberry-pi-pico/>.
- [7] MiSUMi. アルミフレーム 5 シリーズ R 形状 20x20mm. <https://jp.misumi-ec.com/vona2/detail/110302685570/?CategorySpec=00000173364%3A%3Ac>.
- [8] CuboRex. ROS - Robot Operating System. <https://www.ros.org/>.
- [9] Kenji Koide, Masashi Yokozuka, Shuji Oishi, and Atsuhiko Banno. Glim: 3d range-inertial localization and mapping with gpu-accelerated scan matching factors. **Robotics and Autonomous Systems**, 2024.
- [10] rsasaki0109. lidar\_localization\_ros2. [https://github.com/rsasaki0109/lidar\\_localization\\_ros2](https://github.com/rsasaki0109/lidar_localization_ros2).
- [11] Abudori. obstacle-cloud-to-scan. <https://github.com/AbudoriLab-TC2024/obstacle-cloud-to-scan>.
- [12] Open Robotics. pointcloud\_to\_laserscan. [http://wiki.ros.org/pointcloud\\_to\\_laserscan](http://wiki.ros.org/pointcloud_to_laserscan).
- [13] Steven Macenski, Francisco Martin, Ruffin White, and Jonatan Ginés Clavero. The marathon 2: A navigation system. In **2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**, 2020.
- [14] nonanonna. penguin\_nav. [https://github.com/AbudoriLab-TC2024/penguin\\_nav](https://github.com/AbudoriLab-TC2024/penguin_nav).
- [15] Steve Macenski, Matthew Booker, and Josh Wallace. Open-source, cost-aware kinematically feasible planning for mobile and surface robotics. **Arxiv**, 2024.
- [16] nonanonna. waypoint-modifier. <https://github.com/AbudoriLab-TC2024/waypoint-modifier>.