

明石工業高等専門学校専攻科

専攻科特別研究論文

遠隔型自動運転システムの
ユーザビリティに関する研究

The Research of Usability of Remote and Autonomous
Driving System

ME2208 高橋 尚太郎
(機械・電子システム工学専攻)

指導教員 野村 隼人

目 次

| | |
|--|----------|
| 第1章 序論 | 1 |
| 1.1 研究背景 | 1 |
| 1.2 本研究の目的と特徴 | 2 |
| 1.3 本論文の構成 | 3 |
| 第2章 FPV 車両操作の体感速度変化率の定式化 | 4 |
| 2.1 はじめに | 4 |
| 2.2 従来研究 | 5 |
| 2.3 体感速度のパラメータの模索 | 7 |
| 2.3.1 原理 | 8 |
| 2.3.1.1 体感速度とスケールスピードについて | 8 |
| 2.3.2 DS・RC カーの走行映像の解析 | 9 |
| 2.3.2.1 実験方法 | 9 |
| 2.3.2.2 実験結果 | 14 |
| 2.3.3 体感速度のパラメータの選定 | 15 |
| 2.3.4 映像のクロップによって体感速度変化を促す方法 | 15 |
| 2.3.4.1 映像のクロップ手法 | 15 |
| 2.3.5 映像クロップによる体感速度の変化の仮説 | 17 |
| 2.3.6 体感速度のパラメータの導入 | 18 |
| 2.4 体感速度モデルの提案 | 18 |
| 2.4.1 原理 | 18 |
| 2.4.2 体感速度モデルの定式化 | 19 |
| 2.5 提案モデルの検証実験 | 25 |
| 2.5.1 定式化モデルと実測値の比較 | 25 |
| 2.5.2 実験条件 | 26 |
| 2.5.3 定式化モデルの検証方法 | 26 |
| 2.5.4 体感速度を変化させる効果の検証 | 27 |
| 2.5.5 実験結果 | 28 |
| 2.5.5.1 定式化モデルと実測値の比較 | 28 |
| 2.5.5.2 体感速度を変化させる効果の検証 | 30 |
| 2.6 考察 | 30 |
| 2.6.1 定式化モデルと実測値の比較に関して | 30 |
| 2.6.2 体感速度を変化させる効果の検証 | 31 |

| | |
|---|-----------|
| 2.7 おわりに | 31 |
| 第3章 LiDARのみの SLAM で作成された環境地図の評価 | 32 |
| 3.1 はじめに | 32 |
| 3.2 従来研究 | 33 |
| 3.2.1 SLAM の概要 | 33 |
| 3.2.1.1 SLAM の効果 | 33 |
| 3.2.1.2 SLAM の実行形態 | 36 |
| 3.2.1.3 SLAM の種類 | 36 |
| 3.2.2 SLAM の原理 | 37 |
| 3.2.2.1 地図構築: ランドマーク位置の推定 | 37 |
| 3.2.2.2 ロボット位置の推定 | 40 |
| 3.2.2.3 ロボット位置とランドマーク位置の同時推定 | 41 |
| 3.2.2.4 外界センサが一度に大量に得られる場合 | 42 |
| 3.2.3 要素技術 | 43 |
| 3.2.3.1 不確実性の扱い | 43 |
| 3.2.3.2 データ対応付け | 44 |
| 3.2.3.3 センサ融合 | 45 |
| 3.2.3.4 ループ閉じ込み | 46 |
| 3.2.4 SLAM の処理形態: 一括処理と逐次処理 | 47 |
| 3.2.5 Gmapping | 48 |
| 3.2.5.1 事前推定 | 49 |
| 3.2.5.2 観測更新 1 | 50 |
| 3.2.5.3 地図更新 | 50 |
| 3.2.5.4 観測更新 2 | 50 |
| 3.2.6 Hector SLAM | 53 |
| 3.2.7 Cartographer | 53 |
| 3.2.7.1 分岐限定法によるスキャンマッチング | 54 |
| 3.2.7.2 分岐限定法によるスキャンマッチングアルゴリズム | 55 |
| 3.2.8 SLAM による環境地図の評価 | 57 |
| 3.2.8.1 Hector SLAM および Gmapping の性能分析 | 57 |
| 3.2.8.2 Grand truth データと LiDAR を用いた 2D SLAM アルゴリズムの地図比較 | 58 |
| 3.3 提案 | 59 |
| 3.3.1 実装 | 59 |
| 3.3.1.1 ハードウェア | 59 |
| 3.3.1.2 ソフトウェア | 60 |
| 3.3.1.3 Gmapping, Cartographer | 60 |
| 3.3.1.4 laser scan matcher | 63 |
| 3.3.1.5 laser scan matcher の原理 | 63 |
| 3.3.1.6 Hector SLAM | 65 |
| 3.3.1.7 パラメータの調整 | 67 |

| | | |
|---|----------------------------|-----------|
| 3.3.1.8 | RViz による環境地図の可視化 | 69 |
| 3.4 検証 | | 70 |
| 3.4.1 実験 | | 70 |
| 3.4.2 結果 | | 72 |
| 3.5 考察 | | 75 |
| 3.6 おわりに | | 76 |
| 第 4 章 自動運転システムの開発 | | 77 |
| 4.1 はじめに | | 77 |
| 4.2 従来研究 | | 78 |
| 4.2.1 つくばチャレンジの屋外ナビゲーションシステム | | 78 |
| 4.2.2 中之島チャレンジ | | 78 |
| 4.2.3 Autonomous Driving of Competition Robot | | 79 |
| 4.2.4 Driverless Car のデザインと実装: 三重県南伊勢町における実証実験 | | 80 |
| 4.3 提案 | | 80 |
| 4.3.1 システム構成 | | 80 |
| 4.3.2 システムインターフェース | | 81 |
| 4.3.2.1 ステアリングコントローラによる遠隔操作 | | 81 |
| 4.3.2.2 カメラ映像の取得 | | 82 |
| 4.3.2.3 SLAM | | 83 |
| 4.3.2.4 自動運転機能 | | 83 |
| 4.3.2.5 waypoint の設定 | | 83 |
| 4.3.2.6 自動運転・手動運転の引継ぎ機能 | | 84 |
| 4.4 実装 | | 84 |
| 4.4.1 システムの実装 | | 84 |
| 4.4.2 ROS について | | 84 |
| 4.4.2.1 ピアツーピア設計方式 | | 85 |
| 4.4.2.2 多言語に対応 | | 85 |
| 4.4.2.3 機能の細分化と集約 | | 86 |
| 4.4.2.4 補助ツールが豊富 | | 86 |
| 4.4.2.5 オープンソースのプラットフォーム | | 86 |
| 4.4.2.6 コンピューティンググラフ | | 87 |
| 4.4.3 遠隔操作機能の実装 | | 90 |
| 4.4.3.1 コントローラ入力値解析 | | 90 |
| 4.4.3.2 ステアリング・前進後退機能 (AT・MT) | | 90 |
| 4.4.3.3 ROS マスタースレーブ (Laptop to Raspberry Pi) | | 90 |
| 4.4.3.4シリアル通信 (Raspberry Pi to Arduino) | | 90 |
| 4.4.3.5 カメラ映像機能 | | 90 |
| 4.4.4 SLAM の実装 | | 91 |
| 4.4.4.1 urg node | | 91 |
| 4.4.4.2 Gmapping | | 91 |
| 4.4.4.3 laser scan matcher | | 91 |

| | | |
|---------|-------------------------|------------|
| 4.4.5 | 自動運転機能の実装 | 91 |
| 4.4.5.1 | Navigation Stack | 91 |
| 4.4.5.2 | Transform Tree | 92 |
| 4.4.5.3 | パーティクルフィルタについて | 95 |
| 4.4.5.4 | AMCLにおけるパーティクルフィルタ | 97 |
| 4.4.5.5 | Teb local planner | 100 |
| 4.4.5.6 | アッカーマンステアリングジオメトリ | 105 |
| 4.4.5.7 | コストマップ | 107 |
| 4.4.5.8 | 地図の保存と再読み込み | 110 |
| 4.4.6 | Waypoint Navigation の実装 | 110 |
| 4.4.6.1 | waypoint の設定・保存・可視化 | 110 |
| 4.4.6.2 | 自動運転機能 | 110 |
| 4.4.7 | 自動運転と手動運転の引継ぎ機能の実装 | 111 |
| 4.5 | 検証 | 114 |
| 4.5.1 | 遠隔操作の操作性の評価 | 114 |
| 4.5.2 | 手動運転から自動運転への切り替え | 115 |
| 4.5.3 | 方向転換機能の評価 | 115 |
| 4.5.4 | 結果 | 117 |
| 4.5.4.1 | 遠隔操作の操作性の評価 | 117 |
| 4.5.4.2 | 手動運転から自動運転への切り替え | 118 |
| 4.5.4.3 | 方向転換機能 | 118 |
| 4.6 | 考察 | 120 |
| 4.6.1 | 遠隔操作の操作性の評価 | 120 |
| 4.6.2 | 手動運転から自動運転への切り替え | 120 |
| 4.6.3 | 方向転換機能の評価 | 120 |
| 4.7 | おわりに | 120 |
| | 第5章 結論 | 122 |
| | 付録A 付録があるときは | 126 |

第1章

序論

1.1 研究背景

近年、様々な自律移動ロボットの研究開発が行われ、徐々に人間の生活環境に進出し始めている。従来は、工場における産業用無人搬送車が主体であったが、活躍の場がオフィス、家庭、さらには屋外へと広がりつつある。今後、ロボットを用いた省力化や危険な仕事の代替を進めるためには、自律移動ロボットが活動できる領域が人間の生活環境へと拡大していくことが期待される。

一例として、自律移動の応用技術として交通課題の解決策として自動運転システムの実用化が進んでいる。現在は法定上遠隔地からの監視・運用を必要とする遠隔型自動運転システムが想定されている。遠隔型自動運転では、FPV(First Person View)における車速の制御に関して、ドライバの体感速度変化を促すための視覚効果に関する研究が行われている。視覚効果は、定式化に関して未知数であり、体感速度変化の評価はドライバ自身の知覚の個人差を含めた主観で考えられるため、定性的解析に委ねられている。

自律移動ロボットの実現に関して、工場のように限定された環境では、環境をロボットのために整備することが可能であった。しかしロボットの活動領域を広げるためには、環境をロボットに合わせて整備するのではなく、ロボットが様々な環境に適応することが求められる。人間の生活環境は整備された工場と比較して複雑であり、不確実性が高くなる環境である。この不確実性への対処が、重要な課題となる。特に、ロボットに必要となる知覚、計画、制御のうち、不確実性に最も深く関わるのが知覚（計測と認識）である。移動ロボットを対象として考えると、重要な知覚の機能として自己位置推定と地図生成 (SLAM: Simultaneous Localization and Mapping) がある。自己位置推定は、ロボットが自身の位置（方位を含む）を認識しながら目的地まで移動するために必要となる。また地図は、自己位置推定や経路計画に必要である。人手による地図の生成は多大な工数が必要なため、ロボットが自動的に生成することが望ましい。移動ロボットの自己位置推定と地図生成において不確実性に対処するには、確率論に基づく手法が有効であることが知られている。確率論に基づくことで、センサデータやコンピュータが持つモデルの不確実性を明示的に表現すると共に、事前確率の情報を用いてより確実性の高い推定を行うことが可能となる。一般に、SLAMでは、外界センサであるLiDARに加えて、ロータリーエンコーダ、慣性測定ユニット(IMU)等の内界センサによるロボットの移動量の内部情報(オドメトリ)を用いることで、マッピングを行うことが主流とされているが、オドメトリを必要としない、オドメトリフリーのSLAM手法も提案されている。

自律移動ロボットの開発プロセスにおいて、実環境における実証実験が数多く行われている。特に、屋外における自律移動においては、シミュレーションと実環境では、制約や環境条件が全く異なるため、実環境のみでしか得ることのできない知見が多く、自律移動の実用化に向けては、実機を用いた実証実験が必要とされている。実証実験に代表されるつくばチャレンジでは、自律移動ロボットの設計において、要素技術である LiDAR を中心とした、センサ系の使用が主となり、複数のセンサとの融合（センサフェュージョン）による環境認識の高精度化が図られている。それと同時に、自律移動ロボットの実用化に向けて、使用されるセンサの選定、用途に応じて最適化する試みが行われている。特に、ロボットに搭載するセンサの使用数を少なくすることは、ロボットに課される制約・条件、ソフトウェア開発の工数減、保守性の向上のために、考慮すべき点と述べられている。自律移動チャレンジの参加車両の形態は多様であり、差動二輪型が一般的であるが、パーソナルビークル等の使用としては、4輪車両型の自律移動等も見られる。自律移動ロボットは、歩行者等と同じ空間に共存するロボットとしての位置づけであり、比較的低速であることが特徴であり、LiDAR による環境認識の比重が高い。対して、自動運転では、リアルタイム性が重視され、カメラや深度センサ等のイメージセンサ等のコンピュータビジョンと LiDAR センサによる SLAM がそれぞれ同じ割合での併用が標準となり、センサ融合に関する技術について言及されている。

1.2 本研究の目的と特徴

本研究では、ROS(Robot Operating System) を用いた4輪車両型自律移動ロボットシステムの開発を目的として、一つ目に、LiDAR のみによる SLAM によって生成された環境地図の評価と、二つ目にステアリングコントローラによる、ハンドルフットペダル型インターフェースによる遠隔操作を用いた、手動運転自動運転の切り替えシステムの開発を行う。ROS に実装された SLAM メタパッケージである、slam_gmapping による環境地図構築と Navigation メタパッケージである、Navigation Stack を用いて、Waypoint Navigation により、中継目的地を経由して、最終目的地まで自律移動を行い、最終目的地に到着した時点で、手動運転に切り替えることで、運転操作の引継ぎを実現するシステムを構築する。三つ目に、遠隔型自動運転システムの FPV 操作に着目した、自動車の映像における、運転視野角や、映像画角の変化による体感速度のモデルの定量化を目的として、体感速度モデルの定式化を提案する。自律移動ロボットの要素技術として用いられている LiDAR のメーカーとして有名な北陽電機株式会社は、中之島チャレンジに参加し、自律移動ロボットの実機を開発して、実際にフィールドで議論を行うことで、新製品の開発を行っている。自律移動ロボットの要素技術である LiDAR の開発においても、実証実験が不可欠である。筆者は、中之島ロボットチャレンジを観戦することで、将来的に自律移動ロボットや LiDAR の開発に携わりたいと考えた。今回は、縁あって北陽電機株式会社様より LiDAR を貸与頂き、有効活用して筆者自らの開発アイデアを実現することを目標として研究に取り組むこととした。本研究の位置付けは、LiDAR の使用技術の勉強並びに、北陽電機株式会社様への就職を目的としている。

1.3 本論文の構成

本論文は全 5 章から構成される。以下に各章の概要を述べる。

第 1 章では、本研究の背景と目的及び各章の構成を述べた。

第 2 章「FPV 車両操作の体感速度変化率の定式化」では、ドライバの体感速度変化を促すための視覚効果に関する仮説を提案する。具体的には、体感速度の変化率を支配するパラメータに着目し、走行映像の視聴環境ドライバの視野角と、走行映像のクロップ率をパラメータとしたときの単位時間当たりの映像ピクセルの移動量を体感速度の変化率として、クロップ率・視野角の減少と増加の特性を一つのモデルとして定式化した。

第 3 章「LiDAR のみの SLAM で作成された環境地図の評価」では、内界センサによるオドメトリを用いず、ROS で実装されている各 LiDAR SLAM パッケージを用いた LiDAR のみによる環境地図作成システムを開発し、オドメトリフリーの手法に対して、オドメトリが必要な手法に LiDAR から得るレーザオドメトリを用いて比較することで、LiDAR のみによる SLAM の性能を明らかにした。

第 4 章「自動運転システムの開発」では、測距センサ (LiDAR) による測距データのみを用いた SLAM によって作成された環境地図を用いて、4 輪車両型 RC カーの LiDAR のみによる自己位置推定と Waypoint Navigation の有用性の検証を行った。特に、4 輪車両特有の動作である、切り返しの実現性、自動運転システムとして機能をする場合の遠隔操作のユーザビリティと、手動・自動運転のタスクの切り替え機能の動作に着目した評価を行った。

第 5 章「結論」では、本研究で得られた結果とその成果について述べる。

第2章

FPV車両操作の体感速度変化率の定式化

2.1 はじめに

体感速度の変化とは、基準となる速度に対して、観測環境における何らかのパラメータの作用により、視覚的な体感情報から得られる速度感覚が基準と比べて変化する現象のことである。自動車を運転する際、車速の増加によってドライバの有効視野角が狭まり、ドライバの体感速度が減少する現象が報告されている。また、低速状態においても、ドライバの視野角の変化、運転視野映像のクロップによって体感速度が変化する現象が一般的に知られている。また、自動車等を、カメラ映像を確認しながら遠隔操作 (FPV:First Person View 操作) する際に、カメラの設定によって同様の現象が起こる。これはカメラ映像による視覚情報が、遠隔操作を行うための操作者への体感情報に影響を与えるためだと考えられている。具体的には、カメラの画角の設定などが挙げられる。FPVは、操作対象から見た視点であり、操作者が操作対象から周囲の景色を見る目的としている。自動車の運転操作と、遠隔型自動運転や、FPVによるRCカーの操作では、スケールの違いや、カメラの画角の設定等から、視覚情報から得られる体感情報において矛盾が生じる。一例として、操作対象自身の視点から体感する感覚的な移動速度(体感速度)を錯覚する。一般に、自動車のドライバが利用する情報のおよそ90%が視覚情報であると報告されており、ドライバが受け取る情報の大部分を占めているため、体感速度の錯覚が、適切な運転に影響を与え、思わぬ事故を引き起こす恐れがある。体感速度の錯覚は、走行環境と自動車の運転速度の変化によっても生じる。より広い道路を高速で移動すると、人間の視野における周辺視の減少により、自動車の体感速度を実際の速度よりも低く感じるよう錯覚すると報告されている。高速道路等で体感速度を低く感じると、ドライバは、車速を増加させる傾向にある。車速の増加が自動車事故の原因になる。また、逆に低速域においても視野角の減少によって体感速度が減少することが報告されている。また、ゲーム等の演出で使用されているものとしては、視野を広げると速く見せることができる。一方、運転映像や運転中の視野角の変化による体感速度の比率の定量化に関しては行われていない。視野角を狭めることによる効果や、視野の増加によって得られる体感速度の定量化(モデル化)はされていない。また、視野角の減少と増加の両方における体感速度の変化を一つの特性として表した例はない。体感速度を定式化することができれば、映像から得られる視覚効果の影響を光学的に応用し、FPV操作や、ヘッドアップディスプレイ等によって、ドライバの体感情報の補正手法としての応用が期待できる。そこで我々は、自動車の映像における体感速度のモデルの定量化を目的として、体感速度モデルの定

式化を提案する。DS の走行映像において観測者の体感速度の減少を促す映像クロップと視野角の特性を求め、クロップ率-体感速度と、視野角-体感速度の幾何学モデルを定式化する。本研究では、クロップ率・視野角の減少と増加の特性を一つのモデルとして表し、視野角の変化に対する体感速度の変化を定式化することで、映像から得られる体感速度を定量的に定義した。

2.2 従来研究

淺田ら [1] は、人間の視覚情報を用いた体感速度変化を利用した研究として、ドライビングシミュレータの操作視点にバーチャルパターンを投影することによって、ドライバに適切な速度制御を促す手法が提案されている。運転者の速度制御をモデル化することにより、ドライバが適切な速度を制御できない要因として「環境からの速度認知に関する誤差が大きいこと」、「目標速度設定が適切でないこと」の2点が挙げられることを明らかにした。前者の要因を改善するバーチャルパターンとして、一定速度で移動するバーチャルパターン、流れる風景を誇張表現するバーチャルパターンを挙げ、バーチャルパターン投影映像の視聴実験の被験者の主観評価により、これらの有効性を確認した。これらは、目標速度制御のための定性的な効果であり、体感速度の変化に寄与するパラメータについては言及されておらず、体感速度の変化を示すものは、被験者の主観による評価であるため、視覚効果の定量的な評価はされていない。

大前ら [2] は、遠隔操縦に基づく資格情報の影響を評価し、遠隔操縦において、カメラ条件や映像条件が運転操作に与える影響を評価している。評価結果に基づいて遠隔操縦車両および、その制御系を構築し、遠隔操縦と直接運転を比較することで、本研究で構築した遠隔操縦システムの構成により、直接運転と近い運転操作が実現できることを明らかにした。視覚情報の影響評価においては、カメラの視野角の影響が大きく、フレームレートや解像度などの影響が相対的に小さいことが報告されている。以上より、遠隔操縦における視覚情報の運転への影響として、視野角による影響が大きいことが述べられている。

Speed Management [3] によると、速度が上がるとドライバの視野が狭くなることが報告されている。時速 40 km の場合、ドライバの視野は 100 度であり、路傍の障害物やその他の潜在的な危険を確認できる。図 2.1 に視野角と体感速度の関係を示す。時速 130 km では、視野は約 30° にまで減少し、ドライバが潜在的な危険を判断する能力が大幅に低下するとされている。

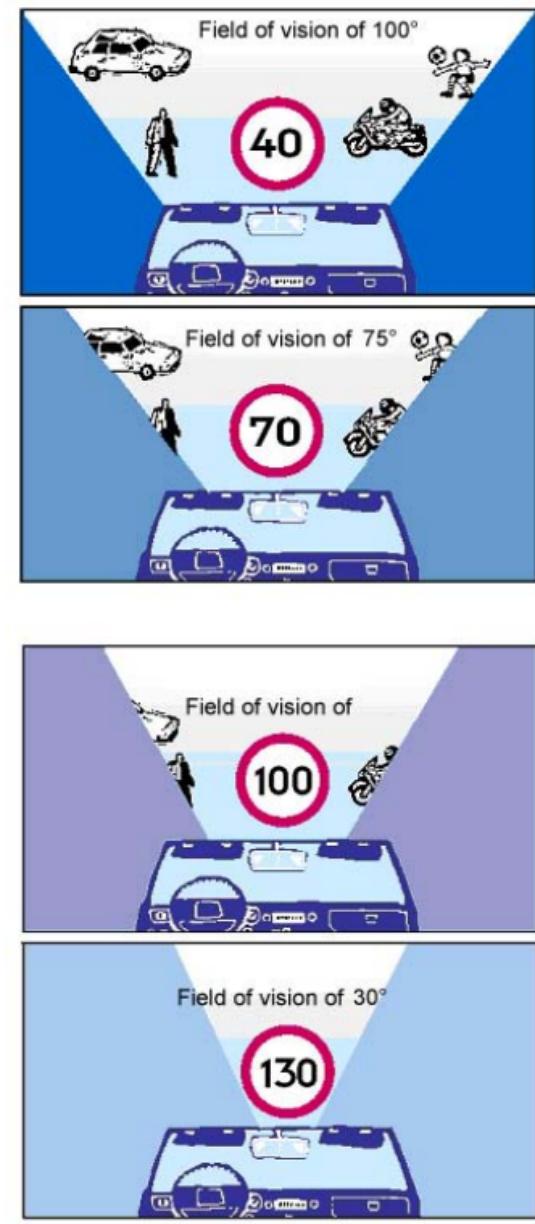


図 2.1: 視野角と体感速度の関係 [3]

体感速度を支配するパラメータとして、運転者の速度に影響を与えるパラメータに関する研究をいくつか述べる。

AR 技術を用いて運転者の体感速度を変化させる試みとして東井ら [4] は単純な線や四角形からなるパターンの速度を実際よりも速く表示することで運転者の体感速度を変化させることに成功していた。しかし、実際にどのような速度制御が行われるかや他の表示方法で表示した場合の効果は未知数である。

走行速度と道路環境の関係についての調査として Gitelman[5] らは運転者が速度超過する原因は道路環境から人々が適切と感じる速度と制限速度があつてないこととし、どういった要素が適切と感じる速度に影響を与えているかを実際の道路環境の要素の調査と運転者への意識調査によって検証した。結果として視野的狭窄や歩行者の活動、道路のレイアウト等が要素として挙げられることが分かった。

Jo ら [6] は、運転速度がドライバの視覚的注意に及ぼす影響を、ドライバが処理できる視覚情報の最大量とのバランスがとれる最大視野という観点から評価を行った。運転速度の増加により、処理する視覚情報の量が増加するため、ドライバが処理する視覚情報の量が、自身の取得できる最大量と釣り合う点までしか視野を広げることができないとした。この点を超えるとドライバは、不安、ストレスの増加によりドライバが取得できる最大の視覚情報が減少し、視野がさらに狭くなつて心理的圧迫感を感じる現象(トンネル効果)が起きる可能性があると述べている。具体的に、ドライバは、潜在的な危険があった場合に対処するための時間を確保するために、運転速度が上がるにつれて遠くを見る傾向がある。これは、運転速度の増加による流体刺激の増加だけでなく、遠くを見ることでの視野の増加により、ドライバが処理する視覚情報の量が増加し、ドライバの精神的負担が増大することを意味する。ドライバは、安全を確保するために、取り扱える視覚情報の量が許される限り、可能な限り多くの視覚情報を取得する傾向にある。しかしながら、前述したように、運転者が取り込める最大の視覚情報は、運転の専門知識のレベルに応じて個人差がある。走行速度が取り込める視覚情報の許容量が釣り合う点を超えた場合、ドライバは自動的に視野を狭めることで視覚的情報の許容が釣り合う点を自己維持しようとし、処理する視覚情報の量を減らす。一部のドライバにとっては、バランス点を超える不安ストレスが非常に高くなり、バランス点を超えた後に処理できる最大の視覚情報がさらに減少している。

2.3 体感速度のパラメータの模索

RC カーと実車の速度が視覚的に一致すること(体感速度が一致すること)を目的として、体感速度の変化を及ぼすパラメータを、RC カー・DS の走行映像の比較により調査した。体感速度の比較について、車両模型の迫力・臨場感の表現として用いられるスケールスピードとの比較も行った。これにより体感速度の一致の目標が、走行映像中における単位時間当たりの描画の移動ピクセル数の一一致であることが分かった。また、この目標を達成するためのパラメータを明らかにした。

2.3.1 原理

図 2.2 に、体感速度の原理を示す。一般に、トラックなど乗用車よりも車高が高くなる車両を運転する場合、体感速度は遅くなり、逆に、レースカート等の車高の低い車両を運転すると体感速度は速く感じる。RC カーを運転する場合は後者に相当する。その原理を幾何学的に説明する。

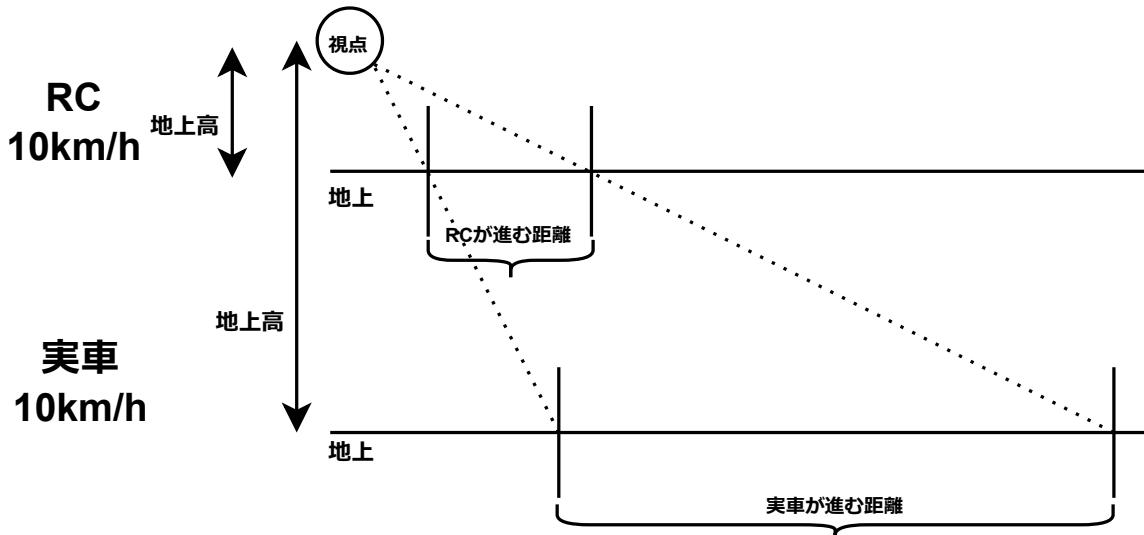


図 2.2: 体感速度の原理

体感速度の変化は上下・左右方向の視野それぞれの影響が存在する。一例として上下方向の視野の場合を説明する。操作視点が高い時の視点は、低い時の視点の延長線上にある。ある点から他方の点まで進んだ時も同様になる。実車あるいは RC カーが、体感的と同じ速度で移動しているための条件は、画面上のピクセルの移動速度(実際の線を移動する時間)が、実車と RC カーで同じになることである。RC カーと実車で速度が同じでも、風景が移動する速度が異なるために体感速度が異なる。この RC カーの実速度に対して体感速度が同じになる実車の速度との比を求めることで、体感速度の一致を求めることができる。つまり、体感速度の一致を走行映像中の単位時間当たりの移動ピクセル数の一致とする。

2.3.1.1 体感速度とスケールスピードについて

スケールスピードとは、模型車両の実際の移動速度にスケール比の平方根を乗じたものを、実際の車両のスケールでの速度としたものである。実際の速度にスケール比を乗ずるのではなく、力学的な相似効果から迫力の表現として使用されている。スケールスピードの導出には、流体分野における流れの相似の指標とするフルード数が用いられ、以下のようにして導出する。

フルード数 Fr の定義は、式 (2.1) で表される。

$$Fr = \frac{U}{\sqrt{Lg}} \quad (2.1)$$

ここで、 U : 体感速度 [m/s]、 L : 代表長さ [m]、 g : 重力加速度 [m/s²] である。実車の特性速度、代表長さをそれぞれ、 U_1 、 L_1 、RC カーの特性速度、代表長さを U_2 、 L_2 とすると、それぞれが力学的に相似であることから、フルード数が一致するという条件のもとで、立式すると式 (2.2)、(2.3) のようになる。

$$\frac{U_1}{\sqrt{L_1 g}} = \frac{U_2}{\sqrt{L_2 g}} \quad (2.2)$$

$$U_2 = \frac{U_1}{\sqrt{\frac{L_1}{L_2}}} \quad (2.3)$$

$\frac{L_1}{L_2}$ は、実車と RC カーのスケール比であるため、特性速度、 U_2 が RC カーのスケールスピードであり、実車の速度 U_1 をスケール比の平方根で除すると求めることができる。

2.3.2 DS・RC カーの走行映像の解析

ここでは、DS と RC カーの走行映像の比較について述べる。DS 上の実速度と RC カーのスケールスピードが一致する場合 RC カーと実車の体感速度が等しいと仮定して、RC カーと DS の走行映像の単位時間あたりの移動ピクセル数の一致を目指す。またスケールスピードではなく、スケール比を乗じた速度での走行映像との比較も行った。

2.3.2.1 実験方法

1. RC カーの走行環境を作成する。図 2.3(a) に、RC カーの走行環境を示す。今回は移動ピクセル数の比較のために、コース車両の脚部分に黄色い装飾を施した。RC カーに設置したカメラで定速走行映像を撮影し、始点・終点間の動画の再生時間で除して実速度を求め、スケール比の平方根との積によりスケールスピードを計算する。RC カーの移動速度は、RC カーの DC モータの PWM 制御によって行われているため、PWM 周期の入力値と出力値を実測する。この出力値を用いて RC カーの移動速度を計算している。
2. DS 上で実車スケールの RC カーの走行環境を作成する。図 2.3(b) に DS 上で再現した RC カーの走行環境を示す。RC カーの現物の環境と視覚的な条件を等しくするために、RC カーの走行実験で使用した環境の壁面(机)の寸法を測定し、走行路面をテクスチャマッピングによって再現した。
3. DS の走行環境において RC カーのスケールスピードで走行した映像を撮影する。DS と RC カーの走行映像を見比べて、単位時間当たりに通過するポールの数を目視で確認し、スケールスピードと体感速度の一致・不一致を評価する。

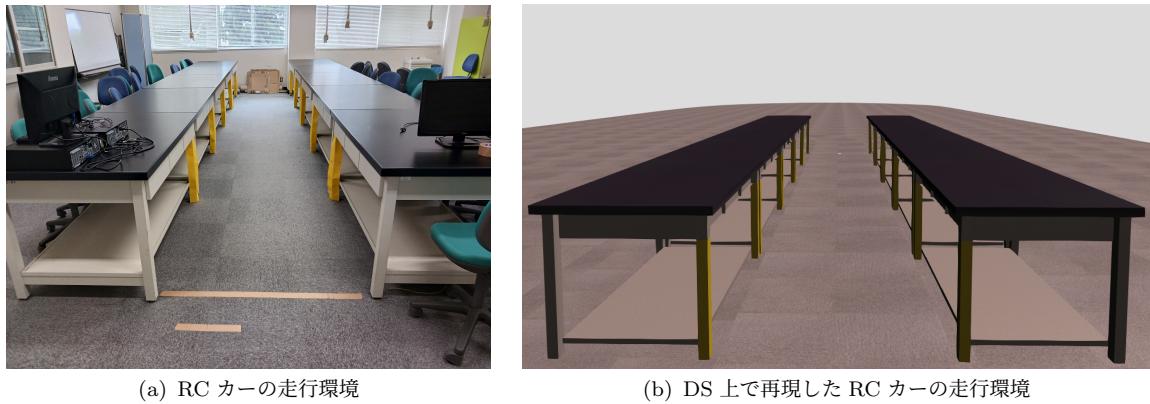


図 2.3: 体感速度の実験環境

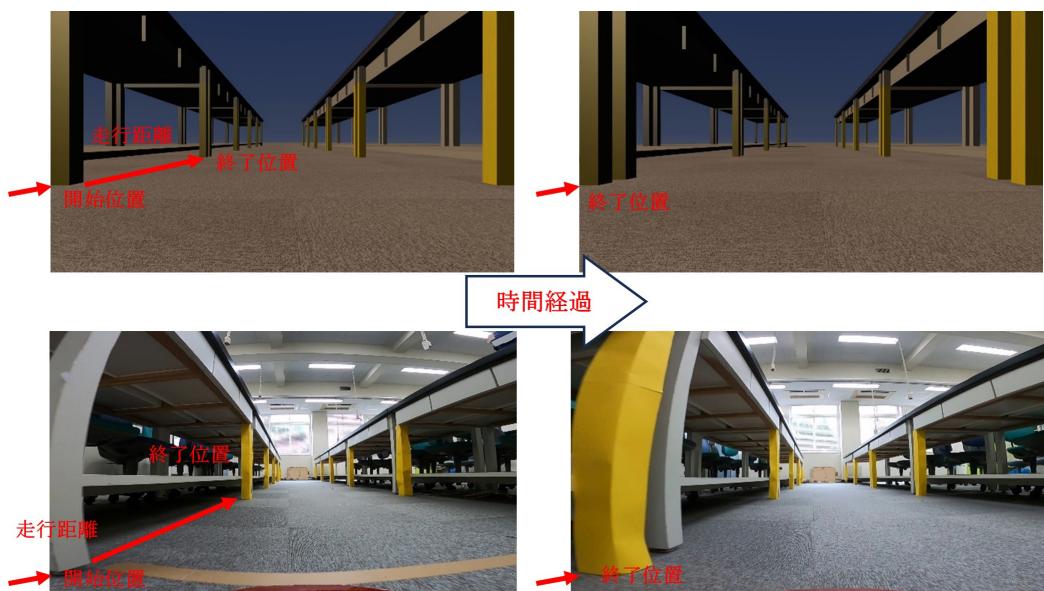


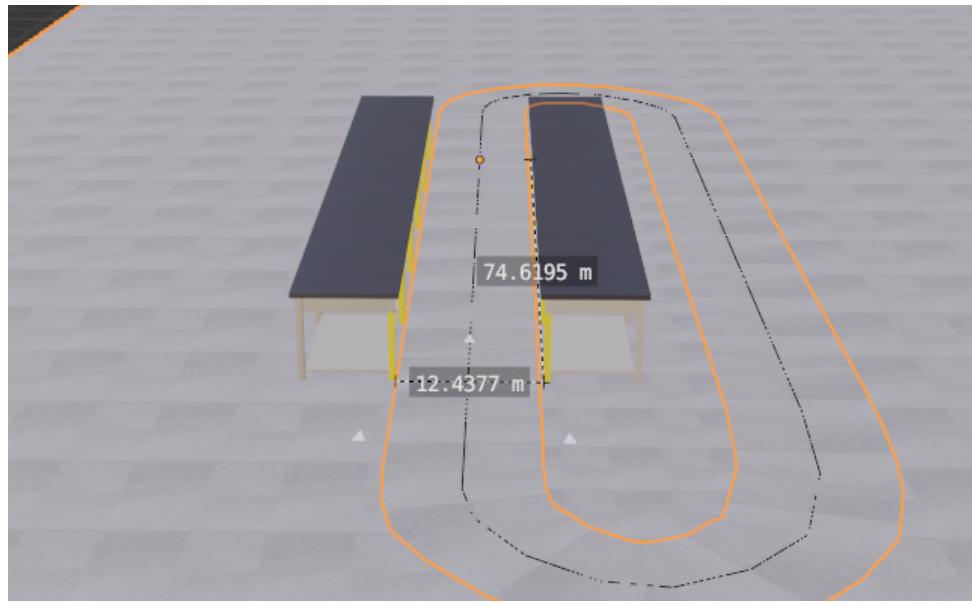
図 2.4: 単位時間当たりに通過するポールの数の確認

4. RC 映像にスケール比を乗じた速度の RC カーの走行映像を取得する。
5. スケールスピードでの比較とスケール比をかけた速度を比較する。
6. 体感速度が一致しない場合、その原因を検討し、パラメータとする。

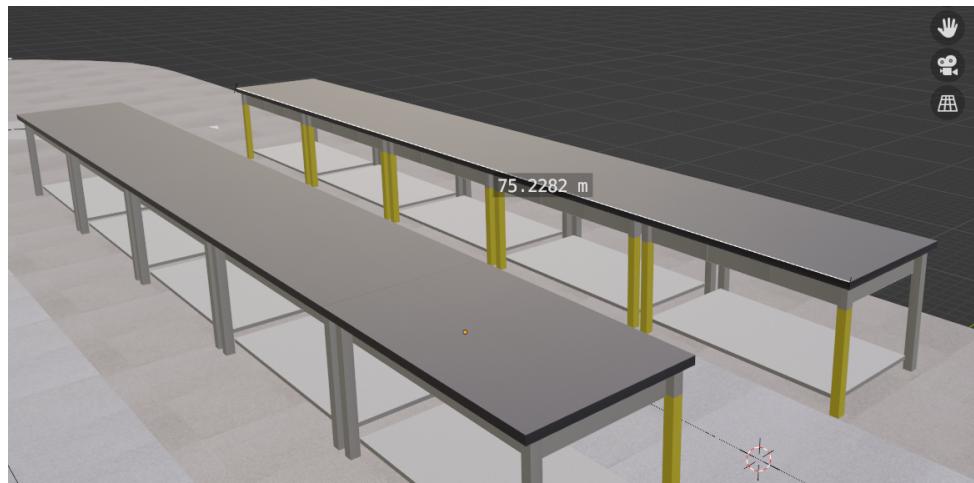
なお、本実験で作成した DS 環境の寸法を図 2.5、2.6 に示す。DS 上の環境に関しては、RC カーのスケール比に合わせて作成した。今回用いた RC カーのスケールが $\frac{1}{10}$ であったため、実際の RC カーの走行環境を 10 倍した DS 上の走行環境とした。

実験の様子を図 2.7 に示す。今回のスケールスピードの比較は、表に示す 32~48 [km/h] の 6 パターンで行った。一例として、RC カーのスケールスピード 32 [km/h](実際の速度は 10 [km/h]) と DS 上の速度 32 [km/h] の比較を行う。

スケール比での比較は、RC カーの実際の速度 10 [km/h] と DS の 100 [km/h] の映像で



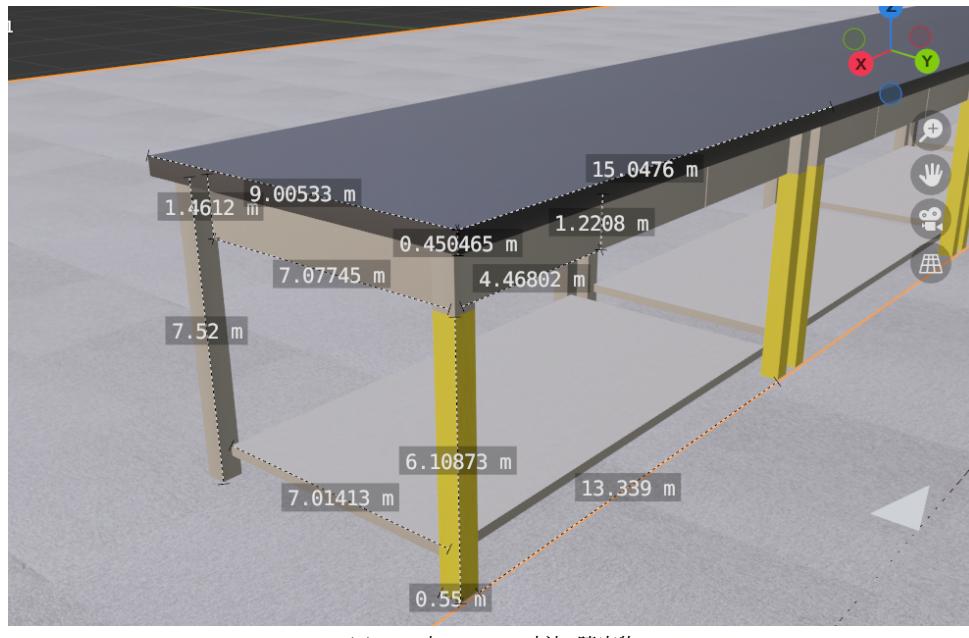
(a) DS 上コースの寸法: 全体



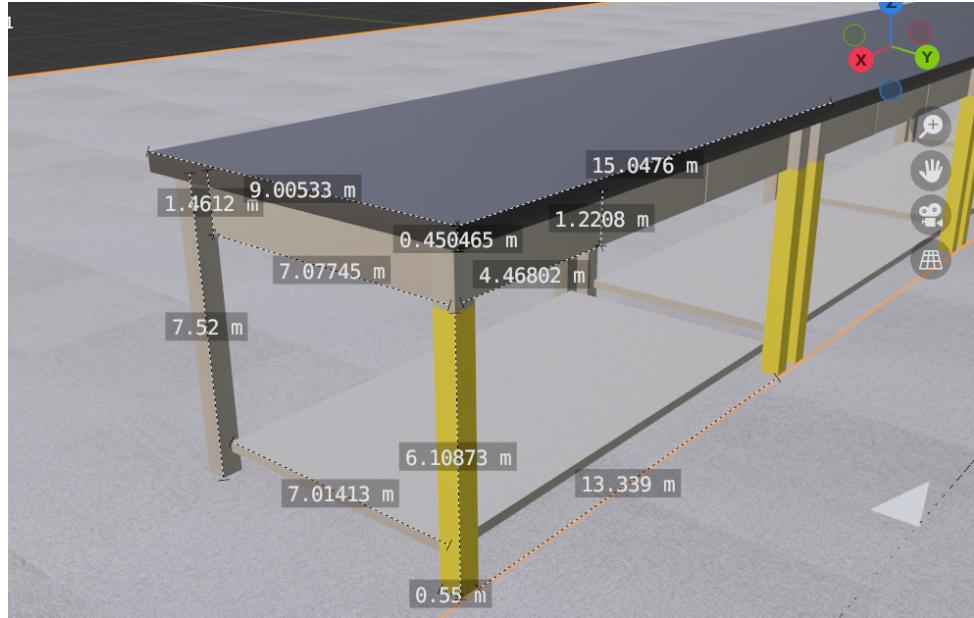
(b) DS 上コースの寸法: 全長

図 2.5: DS 環境の寸法: 全体

行った。体感速度の評価を目的として DS と RC カーの走行映像を PC ディスプレイ上に並べて視聴する。視聴者の主観によって体感速度の遅速・一致を評価する。体感速度が一致しない映像を観察することで、体感速度の変化を示すパラメータを模索する。



(a) DS 上コースの寸法: 障害物



(b) DS 上コースの寸法: 道幅

図 2.6: DS 環境の寸法: 詳細

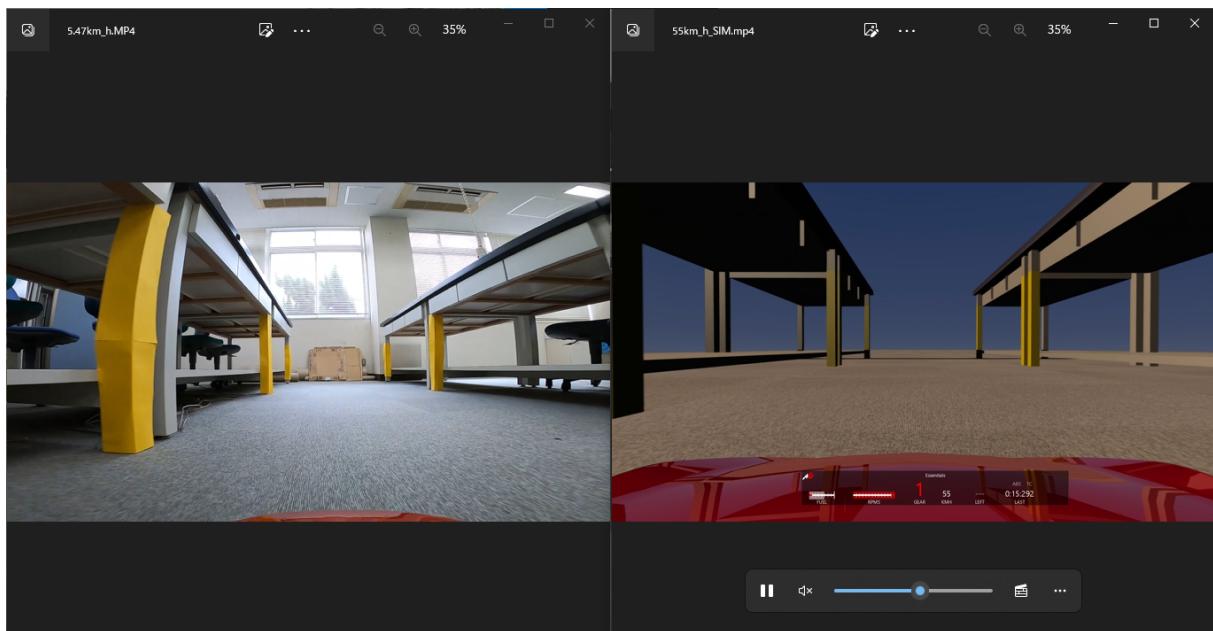


図 2.7: 実験の様子

2.3.2.2 実験結果

実測値とスケールスピードの計算結果を表 2.1 に示す。RC カーのスピードはモータの回転数を回転数計で計測し、計算を行って、時速への変換を行っている。なお、モータの回転数は 10 回計測した際の平均値を採用している。

表 2.1: 実速度とスケールスピードの計算結果

| パルス入力値 [μs] | パルス測定値 [μs] | パルス誤差 | 回転数 [rpm] | 実速度 [km/h] | スケールスピード [km/h] |
|-------------|-------------|-------|-----------|------------|-----------------|
| 1348 | 1329 | 19 | 2194 | 24.81 | 78.47 |
| 1402 | 1386 | 16 | 1896 | 21.44 | 67.81 |
| 1438 | 1418 | 20 | 1589 | 17.97 | 56.83 |
| 1458 | 1440 | 18 | 1337 | 15.12 | 47.82 |
| 1480 | 1460 | 20 | 1161 | 13.13 | 41.52 |
| 1502 | 1482 | 20 | 854.6 | 9.666 | 30.57 |

モータの回転数を時速に変換する式を式 (2.4) に示す。

$$v_{km} = d \times r \times 0.01885 \quad (2.4)$$

ここで、 v_{km} : 時速、 d : ホイールの直径 (6 [cm])、 r : 回転数 [rpm]、

0.01885: 係数成分: $\frac{3600}{1000} \times \frac{2\pi}{60 \times 100}$ である。

RC カーのスケールスピード 32 [km/h] (実際の速度は 10 [km]) と DS 上の速度 32 [km/h] で比較した場合は、体感速度が一致しなかった。RC カーの速度 10 [km/h] と DS 上の速度 100 [km/h] を比較した場合は、体感速度が一致した。実験結果より、RC カーの操作視点カメラ映像の方が、DS の操作映像よりも体感速度が速いと評価されるため、RC カーのスケールスピードは体感速度と一致しないことを確認した。また、スケール比で走行させた場合に体感速度が一致したことから、RC カーと自動車における移動ピクセル比は、スケール比の平方根ではなく、スケール比と同じ値をとることが検証された。つまり、体感速度は、スケールスピードによる力学的相似ではなく、単純なスケール比を一致させた場合に一致すると言える。次に、スケールスピードと体感速度が一致しない原因として検討したパラメータを挙げる。

1. 映像中の風景線が流れるスピード (映像ピクセルの移動速度)

撮影範囲の道路における白線や、壁面と空との境界線等の風景線が単位当たりに移動する映像上のピクセル数に注目することで、体感速度が異なることを確認した。

2. 地上高

RC カーの方が、実車と比べて操作視点の地上高が低いため、近くを見るようになる。景色は遠くより近くの方が速く流れるため、体感速度が上がる。これは、電車に乗って外を見ると、近くの建物は速く動くが、遠くの山などは遅く動くことと同じ現象である。RC カーの操作視点カメラの取り付け位置を変更し、変更前の走行映像と比較したが、体感速度に変化は見られなかった。

3. 水平線に対する地面の割合

地上高が低くなり、近くを見るようになると、水平視点に対して、地面が見える割合が増えることによって体感速度が上がると推測した。水平線に対する地面の割合を変えるために、カメラの下半分を段ボールで隠し、水平線以上の景色のみしか撮影できないようにすることで、早く流れる近くの景色を遮断し、遠くの景色のみを写す。これによって体感速度を変化させる効果は得られなかった。

4. 操作映像のクロップ・カメラの画角

RC カーの操作視点カメラ映像を拡大クロップすることで、RC カーの体感速度が減少し、DS の映像とおおよそ一致したことを確認した。カメラ映像の画角体感速度の抑制に最も大きな効果を与えた。また、映像のクロップは、カメラ映像の画角を狭くする（カメラの種類を広角・狭角と変更する）ことと同じ効果を得ることができた。

2.3.3 体感速度のパラメータの選定

前で述べた体感速度のパラメータのなかで影響度が高いものを選定する。映像のクロップ率を変化させることで、他の全てのパラメータが変化し、体感速度の中で最も影響の高いパラメータであることが明らかになった。

2.3.4 映像のクロップによって体感速度変化を促す方法

本実験で使用したカメラは水平視野角が 120 度である広角レンズを使用している。このため、体感速度が実際よりも早く感じることがあった。カメラの種類（狭角・広角）を変えることは容易ではないが、撮影映像を編集することで体感速度を変化させる方法について検討した。その結果、映像をクロップ（映像の一部を拡大表示）して画面全体に表示させることで、元の映像よりも体感速度を減少させる効果を得た。

2.3.4.1 映像のクロップ手法

図 2.8、2.9 に映像クロップの概要を示す。以下にクロップの手順を示す。

1. FPV 映像をクロップし、元の解像度の比となるように拡大する。
2. 拡大クロップした RC カーの車体が見える部分（下部）を削除する。
3. 拡大クロップした映像の上部 1 を、DS の映像における水平消失点の流れと合わせるように切り取る。
4. 拡大クロップした映像の残った部分を削除し、流れを合わせるように切り取った上部映像を、下部と繋がるように位置を下げる。
5. 1 の上部と連続している映像 2 を下の FPV 映像から切り取る。
6. 1 の上部に 2 を繋ぎ合わせる。

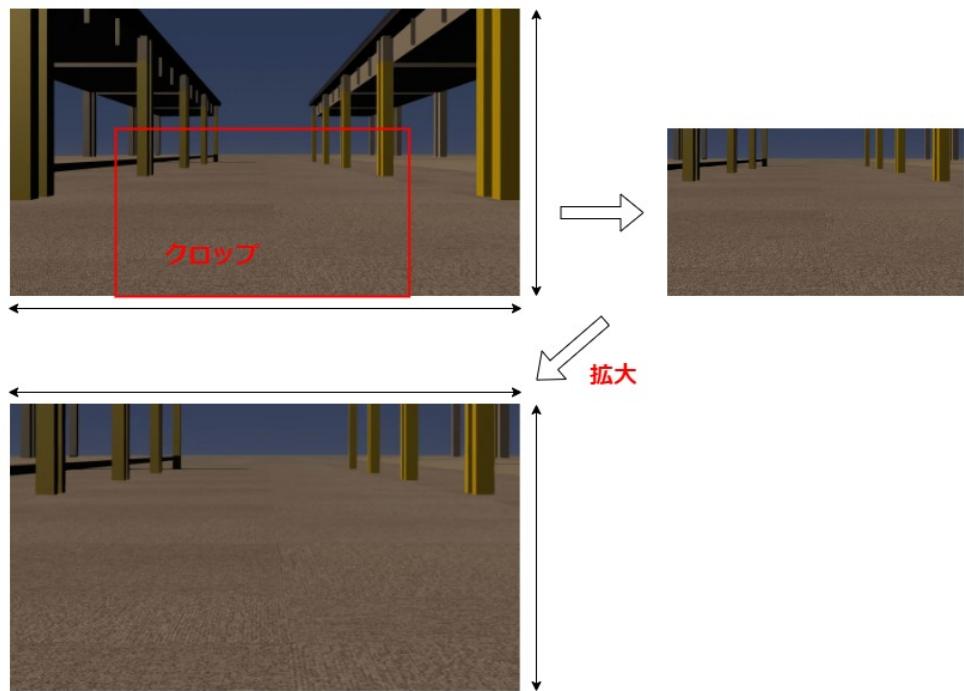


図 2.8: クロップ手順 1

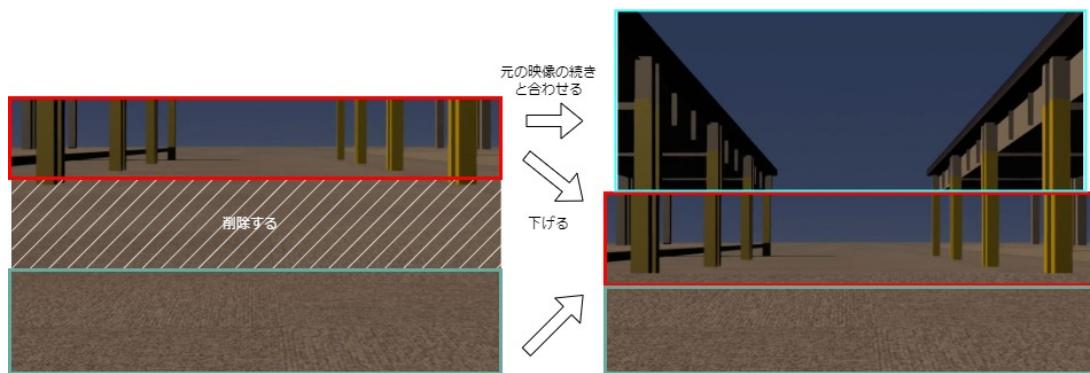


図 2.9: クロップ手順 2

この実験によって、変化したパラメータについて述べる。クロップ率を増加させることで、映像の縦横の長さが減少し、視野角が減少することが明らかになった。また、そのほかのパラメータも同時に変化し、クロップ率・視野角が体感速度の変化を制御するパラメータであることを定性的に示した。この時点では定量的な解析はできない。

2.3.5 映像クロップによる体感速度の変化の仮説

クロップ率の変化によって、体感速度が補正される根拠を以下に示す。

1. 地上高

手元の景色を排除し、遠くの景色を拡大したため、遠くの景色の流れが遅くなることから、体感速度が遅くなる。

2. ピクセル

映像のクロップにより、近くに見える映像がより遠い位置になるため、水平面よりも下の景色の流れが遅くなることで、ピクセルの移動速度が遅くなり、体感速度が遅くなる。

3. 水平面より下の景色

図 2.10 にクロップ前後の見え方の変化について示す。

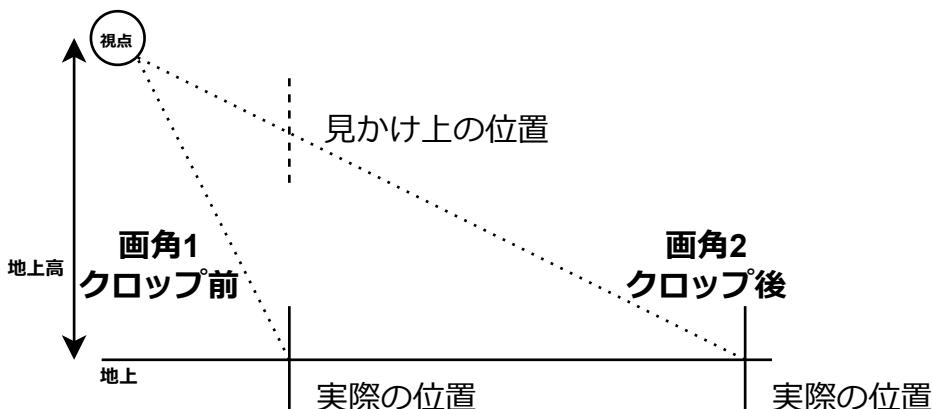


図 2.10: クロップ前後の見え方の変化

映像のクロップにより、クロップ前に見えていたものが前進する。それにより、見かけ上の位置がクロップ前と比べて、近くに見える映像がより遠い位置になるため、水平面よりも下の景色の見え始めが遠くなる。よって、水平面より下の景色の流れが遅くなることで、体感速度が遅くなる。

本研究以外にも同様の報告として、論文によると、映像のクロップによる体感速度の増強効果が、報告されている。また、レース用ゲーム等で用いられる手法として取り入れられていることを確認している。

2.3.6 体感速度のパラメータの導入

体感速度変化で最も影響力のあるパラメータが映像のクロップとカメラの画角であることが分かった。一般的に、走行映像を投影するカメラの画角に加えて、車両のドライバの視野角の変化が体感速度変化に寄与すると考えられている。また、映像のクロップによってカメラの画角の変化が可能であることが共通の見識として共有されている。つまり、映像のクロップによって、容易に車両中のドライバの視点の視野角を変化させることができ、体感速度を変化させることができる。しかし、映像のクロップでは、ドライバの視野角を縮小する場合のみでしか検証ができない。DS の視野角変更機能を用いることで、カメラの画角・ドライバの視野角を拡大する場合の検証を行うことができる。そこで、本研究では、画角(視野角)・クロップというパラメータを導入し、映像のクロップを拡張し、ドライバの視野角の拡大における体感速度の変化に関する検討を行う。

2.4 体感速度モデルの提案

これまで、スケールスピードと実速度を比較することで、体感速度はスケールスピードと一致せず、単なるスケール比に一致することが明らかになった。さらに、その過程において体感速度を変化させるパラメータを発見した。列挙した中で最も影響力のあるパラメータが、走行映像のクロップ率やカメラの画角・ドライバの視野角であることが明らかになった。そこで、走行映像のクロップ率とドライバの視野角を体感速度変化のパラメータとして、体感速度を定式化する手法を提案する。本研究では、実測値との比較として、視野角・ディスプレイ比等、DS 環境におけるパラメータを定数とした幾何学計算による幾何学モデルを提案する。

2.4.1 原理

実車が同じ速度で走行する場合でも、視野角が広い方が、体感速度が速く感じるという現象が生じる基準視野角と、ある視野角での画面上の一方から他方の点にかけてのピクセルの移動距離の比が体感速度の比となる。図 2.11 に、ピクセルの移動距離の考え方を示す。同様の原理として、交通事故で取り上げられるコリジョンコース現象が挙げられる。複数の走行映像において、ピクセルの移動速度が一致する時、体感速度の比と一致していることが望ましい。



図 2.11: ピクセルの移動距離の考え方

2.4.2 体感速度モデルの定式化

体感速度を求めるために、映像中のポール間の移動の際に映像中の移動ピクセル数と、環境によって決まるパラメータを定式化する。ここでは、体感速度 v_{sense} を、視野角の関数 $f(h_{fov_v})$ で表し、 h_{fov_v} を、クロップ率の関数 $g(n)$ で表すことを目標とする。実際の速度（実速度とする）を v_2 とすると、式 (2.5)、(2.6) に示す関係式を求める。

$$v_{sense} = f(h_{fov_v}) \cdot v_s = f(g(n)) \cdot v_s \quad (2.5)$$

$$h_{fov_v} = g(n) \quad (2.6)$$

まず、ディスプレイのアスペクト比とサイズ（ディスプレイの対角線の長さ）をディスプレイの縦・横の長さに変換する式を式 (2.7)、(2.8) に示す。図 2.12 にディスプレイ長さの概要と諸条件を示す。

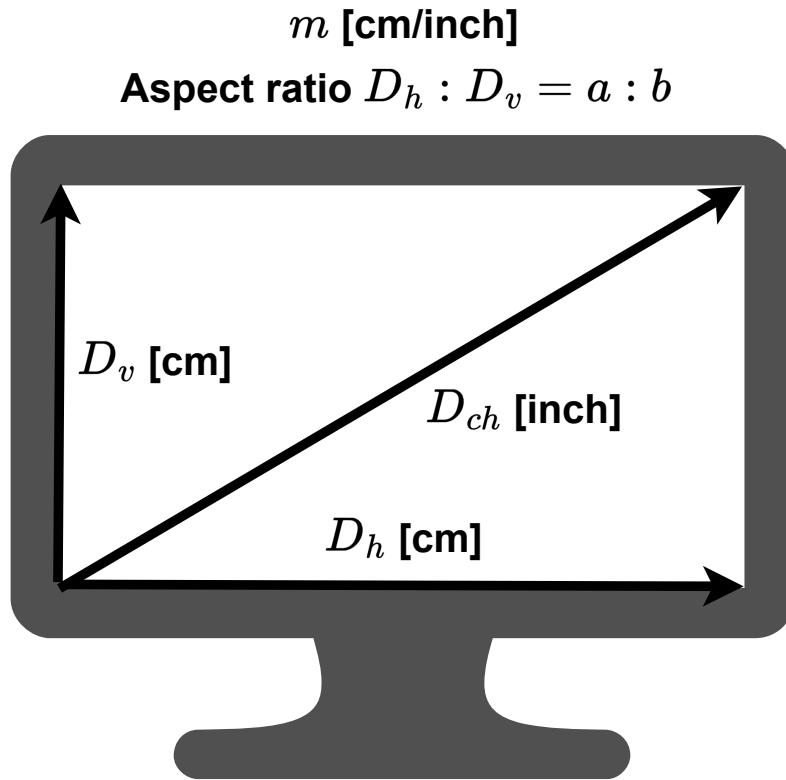


図 2.12: ディスプレイ長さの概要と諸条件

$$D_v = D_{ch} \cdot m \cdot \frac{b}{\sqrt{a^2 + b^2}} \quad (2.7)$$

$$D_h = D_{ch} \cdot m \cdot \frac{a}{\sqrt{a^2 + b^2}} \quad (2.8)$$

D_v : ディスプレイの縦の長さ [cm]、 D_h : ディスプレイの横の長さ [cm]、 D_{ch} : ディスプレイの対角線の長さ [inch]、 $m: 2.4[\text{cm}/\text{inch}]$ 、 a : ディスプレイの横のアスペクト比:16、 b : ディスプレイの縦のアスペクト比:9とする。

次に、ディスプレイの縦・横の長さとディスプレイとの視点距離を DS 上での水平・垂直方向の基準視野角に式 (2.9)、(2.10) を用いて変換する。図 2.13 に基準視野角とその導出の諸条件を示す。

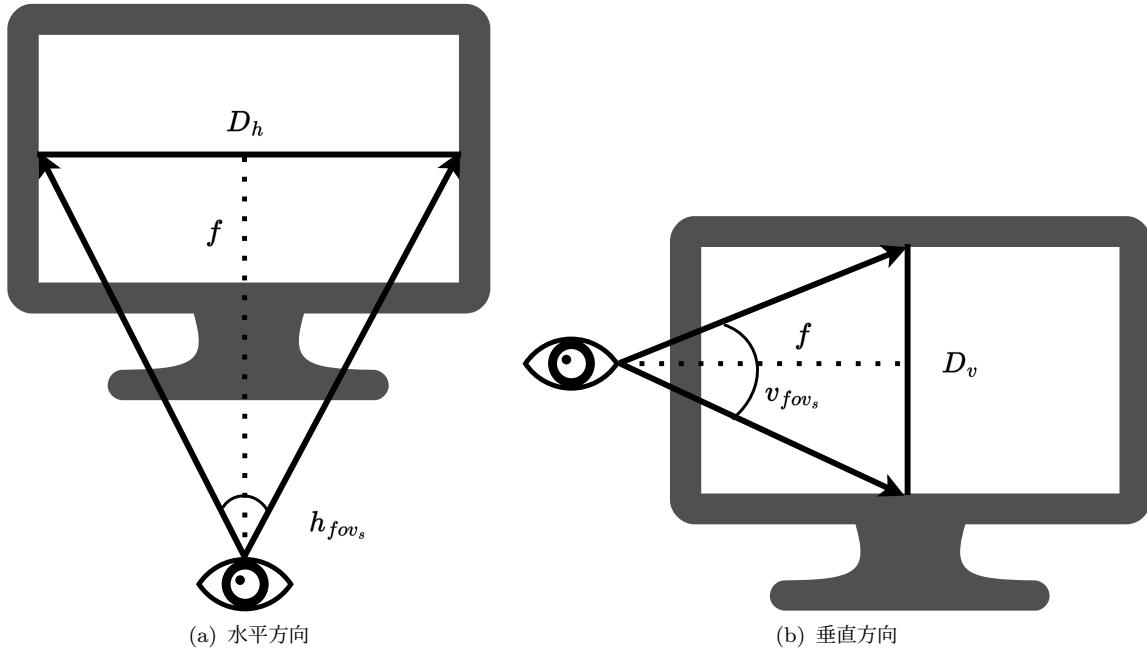


図 2.13: 基準視野角とその導出の諸条件

$$h_{fov_s} = 2 \arctan \frac{D_h}{2f} \quad (2.9)$$

$$v_{fov_s} = 2 \arctan \frac{D_v}{2f} \quad (2.10)$$

h_{fov_s} : 基準水平視野角 [degree]、 v_{fov_s} : 基準垂直視野角 [degree]、 f : 視点距離 [cm] とする。次に、基準視野角を映像クロップによる視野角に変換する。クロップ率 n の値域は、 $1 \leq n \leq 3.5$ とする。 $n = 1$ の時 h_{fov_s} 、 v_{fov_s} (基準視野角) とする。式 (2.11)、(2.12) は、映像のクロップにより、視野範囲が変わることを意味する。クロップは縦横比を一定とし、映像の中心は変化させない。図 2.14 に映像クロップによる視野角とその導出の諸条件を示す。

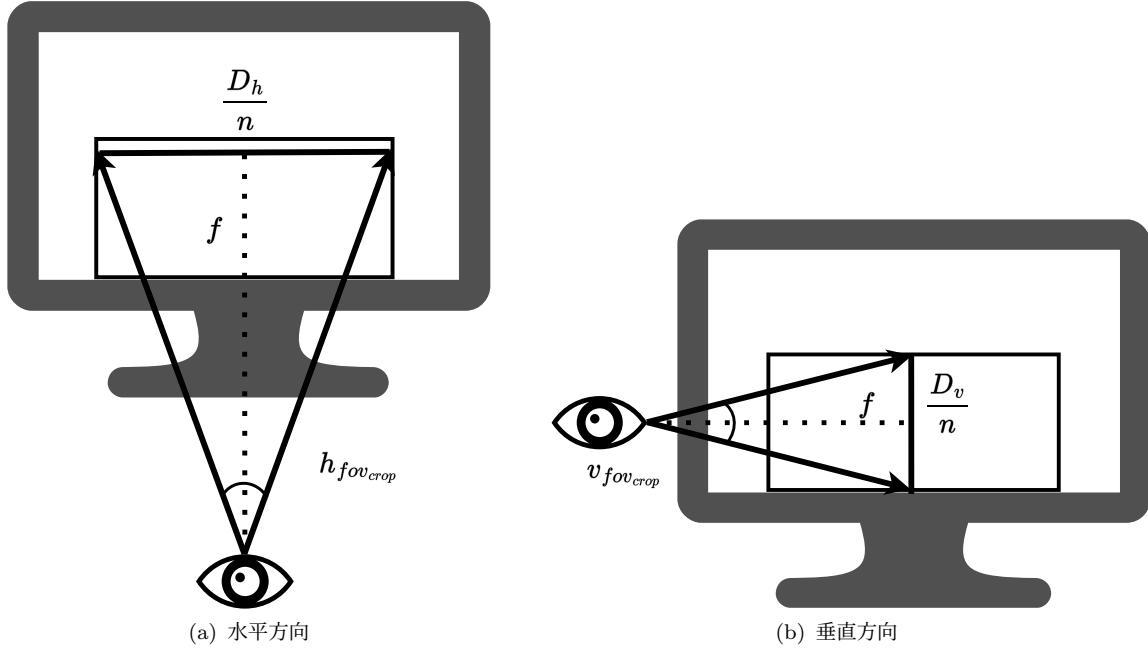


図 2.14: 映像クロップによる視野角とその導出の諸条件

$$h_{fov_{crop}} = 2 \arctan \frac{D_h}{2f \cdot n} \quad (2.11)$$

$$v_{fov_{crop}} = 2 \arctan \frac{D_v}{2f \cdot n} \quad (2.12)$$

$h_{fov_{crop}}$: 映像クロップによる水平視野角 [degree]、 $v_{fov_{crop}}$: 映像クロップによる垂直視野角 [degree]、 n : クロップ率の関数

次に、変数の視野角を基準よりも大きくとる場合に拡張する。つまり、 $n < 1$ を考える。 $n = 0.2 \sim 1$ として拡張し、得られた視野角を変数 h_{fov_v} 、 v_{fov_v} とおく、これを体感速度のパラメータとする。 n は、クロップ率を基準視野角から拡大または縮小するための値として拡張した、視野角拡大率とする。 n は、 h_{fov_v} 、 v_{fov_v} のパラメータである。つまり、体感速度 $v_{sense} = f(h_{fov_v}) = f(g(n))$ とし、 $h_{fov_v} = g(n)$ である。これらの式を、式 (2.13)、(2.14) に示す。

$$h_{fov_v} = 2 \arctan \frac{D_h}{2f \cdot n} \quad (2.13)$$

$$v_{fov_v} = 2 \arctan \frac{D_v}{2f \cdot n} \quad (2.14)$$

h_{fov_v} : 水平視野角変数 [degree]、 v_{fov_v} : 垂直視野角変数 [degree]、 n : 視野角拡大率とする。次に、体感速度は、基準視野角と水平視野角変数におけるそれぞれの映像のピクセル数の比を、実速度に乘ずることで式 (2.15) により求められる。

$$v_{sense} = \frac{px_v}{px_s} v_s = m_{sense} \cdot v_s \quad (2.15)$$

$px_v:h_{fov_v}$ におけるポール間ピクセル数 [px] (実測値)、 $px_s:h_{fov_s}$ におけるポール間ピクセル数 [px] 実測値、 v_{sense} : 体感速度 [km/h]、 v_s : 実速度 [km/h]、 m_{sense} : ポール間ピクセル数の比とする。

図 2.15 に視野角の違いによるポール間ピクセル数の違いを示す。

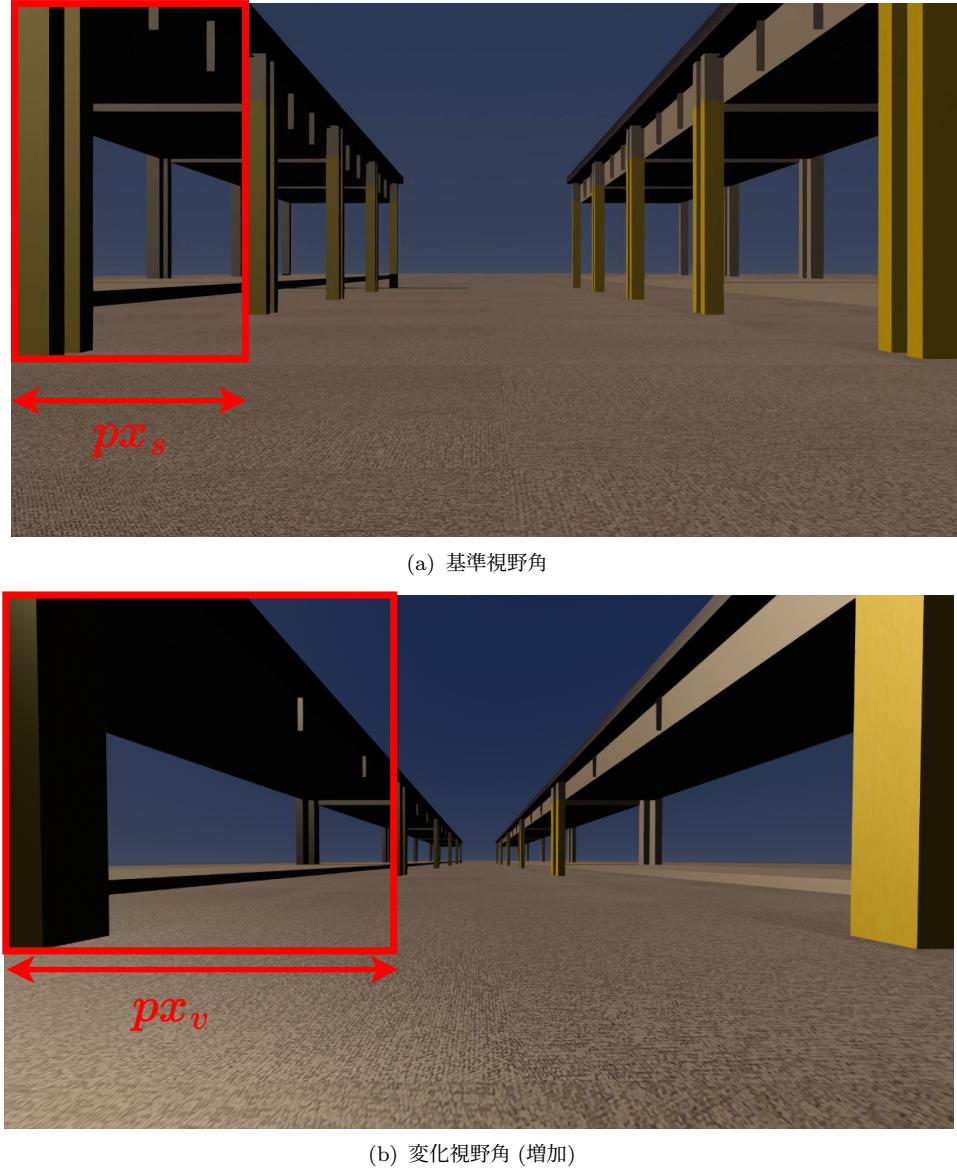


図 2.15: 視野角の違いによるポール間ピクセル数の違い

次に、 $v_{sense} = f(h_{fov_v})$ となる f を導きたい。 v_s は一定として考えているため、 $m_{sense} = f(h_{fov_v})$ を導く。ここで、視野角変数 h_{fov_v} において、映像中のポールが画面の端にあるとする。(視点とディスプレイの短辺は h_{fov_v} の角度をなす。) そこから、次のポールが画面の端になるように縦横比一定でクロップした時のクロップ率を視野辺ピクセル倍率 $n_{p \rightarrow p}$ と

する。基準視野角においても同様に、 $n_{p \rightarrow p_s}$ (基準視野角の場合の $n_{p \rightarrow p_s}$) を求める。なお、遠近法より、 $n_{p \rightarrow p_s} = 2$ であることが一般的に知られている。すると、 m_{sense} は、図 2.16 に示す視野角に関する幾何学を用いて、式 (2.16)～(2.18) のように求めることができる。

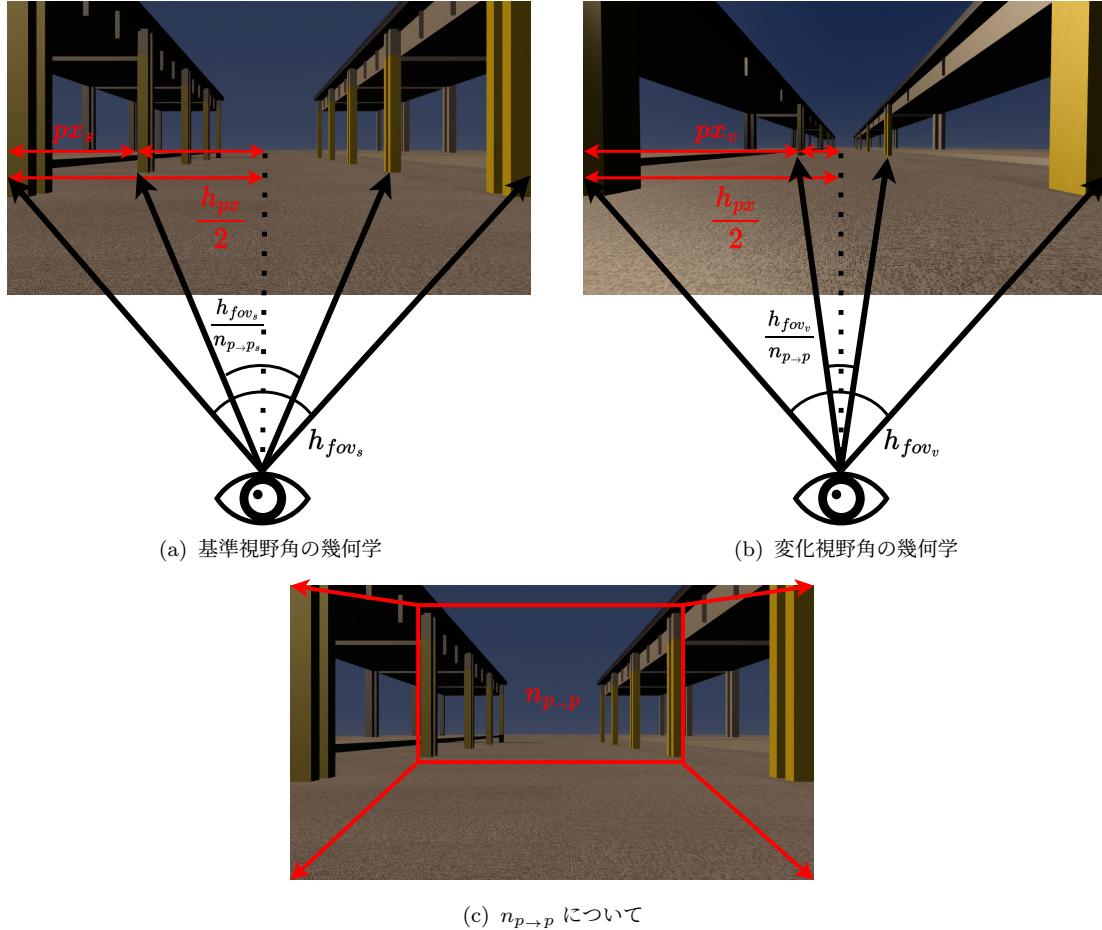


図 2.16: 視野角に関する幾何学

$$px_s = \frac{h_{px}}{2} - \frac{h_{px}}{2} \left(\frac{\tan \frac{h_{fov_s}}{2 \cdot n_{p \rightarrow p_s}}}{\tan \frac{h_{fov_s}}{2}} \right) = \frac{h_{px}}{2} \left(1 - \frac{\tan \frac{h_{fov_s}}{4}}{\tan \frac{h_{fov_s}}{2}} \right) \quad (2.16)$$

$$px_v = \frac{h_{px}}{2} - \frac{h_{px}}{2} \left(\frac{\tan \frac{h_{fov_v}}{2 \cdot n_{p \rightarrow p_v}}}{\tan \frac{h_{fov_v}}{2}} \right) = \frac{h_{px}}{2} \left(1 - \frac{\tan \frac{h_{fov_v}}{2 \cdot n_{p \rightarrow p}}}{\tan \frac{h_{fov_v}}{2}} \right) \quad (2.17)$$

$$m_{sense} = \frac{px_v}{px_s} = \frac{\left\{ \frac{h_{px}}{2} \left(1 - \frac{\tan \frac{h_{fov_v}}{2 \cdot n_{p \rightarrow p}}}{\tan \frac{h_{fov_v}}{2}} \right) \right\}}{\left\{ \frac{h_{px}}{2} \left(1 - \frac{\tan \frac{h_{fov_s}}{4}}{\tan \frac{h_{fov_s}}{2}} \right) \right\}} = \frac{1 - \frac{\tan \frac{h_{fov_v}}{2 \cdot n_{p \rightarrow p}}}{\tan \frac{h_{fov_s}}{2}}}{1 - \frac{\tan \frac{h_{fov_s}}{4}}{\tan \frac{h_{fov_s}}{2}}} = f(h_{fov_v}) \quad (2.18)$$

h_{px} : ディスプレイの横のピクセル比 [px](1920 [px]) $、n_{p \rightarrow p}:h_{fov_v}$ における視野辺ピクセル倍率(実測値) $、n_{p \rightarrow p_s}:h_{fov_s}$ における視野辺ピクセル倍率(定数値:2)。

ここで、幾何学的に求められた $f(m_{sense})$ を、体感速度 v_{sense} の理論式とする。 h_{fov_v} を v_{sense} に変換する。式 (2.18) を整理すると、式 (2.19) のようになる。

$$\begin{aligned} m_{sense} &= \frac{px_v}{px_s} = \frac{1 - \frac{\tan \frac{h_{fov_v}}{2 \cdot n_{p \rightarrow p}}}{\tan \frac{h_{fov_v}}{2}}}{1 - \frac{\tan \frac{h_{fov_s}}{2 \cdot n_{p \rightarrow p_s}}}{\tan \frac{h_{fov_s}}{2}}} = \frac{1 - \frac{\tan \frac{h_{fov_v}}{2 \cdot n_{p \rightarrow p}}}{\tan \frac{h_{fov_v}}{2}}}{1 - \frac{\tan \frac{28}{2 \cdot 1.986}}{\tan \frac{28}{2}}} = 1.923 \cdot \left(1 - \frac{\tan \frac{h_{fov_v}}{2 \cdot n_{p \rightarrow p}}}{\tan \frac{h_{fov_v}}{2}} \right) \\ &= f(h_{fov_v}, n_{p \rightarrow p}) = 1.923 \cdot \left(1 - \frac{\tan \frac{h_{fov_v}}{2 \cdot 1.078415307 \cdot e^{0.011995591 \cdot h_{fov_v}}}}{\tan \frac{h_{fov_v}}{2}} \right) = f(h_{fov_v}) \end{aligned} \quad (2.19)$$

よって、式 (2.19) は、式 (2.20) のようになる。

$$v_{sense} = 19.30 \left(1 - \frac{\tan \frac{h_{fov_v}}{2 \cdot 1.078415307 \cdot e^{0.011995591 \cdot h_{fov_v}}}}{\tan \frac{h_{fov_v}}{2}} \right) \quad (2.20)$$

$n_{p \rightarrow p}$ は、式 (2.21) による回帰曲線とした。

$$n_{p \rightarrow p} = 1.078415307 \cdot e^{0.011995591 \cdot h_{fov_v}} \quad (2.21)$$

この式を基準視野角における倍率をクロップ倍率を m として、正規化すると、式 (2.22)

$$m_{sense} = \frac{px_v}{px_s} = \frac{1 - \frac{\tan \frac{h_{fov_v}}{2 \cdot n_{p \rightarrow p}}}{\tan \frac{h_{fov_v}}{2}}}{1 - \frac{\tan \frac{h_{fov_s}}{2 \cdot n_{p \rightarrow p_s}}}{\tan \frac{h_{fov_s}}{2}}} = \frac{1 - \frac{\tan \frac{h_{fov_v}}{2 \cdot n_{p \rightarrow p}}}{\tan \frac{h_{fov_s}}{2}}}{1 - \frac{\tan \frac{h_{fov_s}}{2 \cdot 1.986}}{\tan \frac{h_{fov_s}}{2}}} = \frac{1 - \frac{\tan \frac{h_{fov_v}}{2 \cdot n_{p \rightarrow p}}}{\tan \frac{h_{fov_v}}{2}}}{1 - \frac{\tan \frac{h_{fov_s}}{2 \cdot 1.986 \cdot m}}{\tan \frac{h_{fov_s}}{2}}} \quad (2.22)$$

$$m = 1/1.986$$

よって、以上をまとめると、式 (2.23)、(2.24) となる。

$$v_{sense} = 19.30 \cdot \left(1 - \frac{\tan \frac{h_{fov_v}}{2 \cdot n_{p \rightarrow p}}}{\tan \frac{h_{fov_v}}{2}} \right) = f(h_{fov_v}) \cdot v_s = f(g(n)) \cdot v_s \quad (2.23)$$

$$h_{fov_v} = 2 \arctan \frac{D_h}{2f \cdot n} \quad (2.24)$$

以上により、視野角、クロップ率を体感速度の倍率に変換する以下の式を導いた。

$$v_{sense} = f(h_{fov_v}) \cdot v_s = f(g(n)) \cdot v_s$$

$$h_{fov_v} = g(n)$$

この定式モデルを用いることで、基準視野角に対して、任意の視野角における体感速度を求めることができる。また、視野角はクロップ率の関数であるため、任意のクロップ率に対しての体感速度を求めることもできる。この定式もでるでは、パラメータを視野角としたが、その他にも、体感速度を決める環境のパラメータとして、ディスプレイ長さや、インチ数等も用いることができる。したがって、走行映像の取得環境に応じた変数と固定値を入れ替えることで、あらゆる環境の幾何学的なパラメータに対応した定式もでると見える。このモデルの大まかな特性を求めるために、 m_{sense} を累乗で近似すると、式 (2.25) となり、体感速度は、式 (2.26) で表すことができる。

$$m_{sense} = 0.086537715 \cdot h_{fov_v}^{0.615630604} \quad (2.25)$$

$$v_{sense} = 0.86537715 \cdot h_{fov_v}^{0.615630604} \quad (2.26)$$

従って、 v_{sense} は、視野角の n 乗根に比例する軌跡となる。その概形は、緩やかに上昇する軌跡である。

2.5 提案モデルの検証実験

これまで、体感速度の定式化を提案した。ここでは、提案したモデルが正しいことを実証するための検証実験に関して述べる。前述の体感速度のモデルを実証するために、以下の手順で実験を行った。

2.5.1 定式化モデルと実測値の比較

前と同様の RC カーの走行環境を模した DS 走行環境において走行実験を行う。基準視野角における速度と映像のクロップ率を変化させた時の体感速度との比率をポール間ピクセル数の計測により求める。ポール間をディスプレイの横幅まで拡大し、ディスプレイの解像度をその拡大倍率で割ることでポール間ピクセル数が導出される。今回、定式化したモデルは、水平視野角をパラメータとして用いているが、DS 環境設定では、垂直視野角を設定値とする仕様であるため、水平視野角を垂直視野角に変換した値を DS 環境の視野角設定に用いる。基準視野角(垂直視野)を 20 度として、DS 環境設定の視野角変更機能により、視野角を変化させる。図 2.17 に DS の視野角設定を示す。

DS 乗で定速走行した映像を解析し、基準視野角とポール間ピクセル数を実測する。基準視野角と、ある視野角の関係から幾何学的にポール間ピクセル数を計算し、定速度値に乘じることで体感速度を実測する。幾何学的な倍率は一部実測値を含む。その後、実測値と幾何学モデルとの一部実測値を含む。実測値と幾何学モデルとの一致を確認する。基準視野角で実速度 10 [km/h]、体感速度 10 [km/h] の映像と、ある視野角を変えて実速度 10 [km/h]、体感速度 17 [km/h](単位時間のポール間の移動ピクセル数が 1.7 倍になったもの)

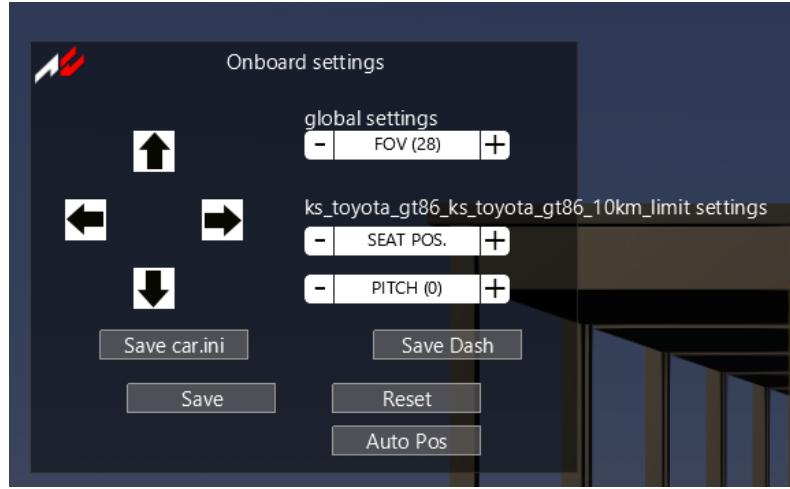


図 2.17: DS の視野角設定

の映像と、基準視野角で実測度 17 [km/h]、体感速度 17 [km/h]、体感速度 17 [km/h] (単位時間に通過する柱の数が 1.7 倍になったもの) の映像を比較して、この仮説を検証する。

一例として、基準視野角に対する視野角での見え方を図 2.18 に示す。赤線の横辺のピクセル数を求める。

2.5.2 実験条件

今回の実験で用いた環境と実験条件を示す。本研究で定式化したモデルは、水平視野角をパラメータとして用いているが、DS 環境設定では、垂直視野角を設定値とする仕様であるため、水平視野角を垂直視野角に変換した値を DS 環境の視野角設定に用いる。その指定値を v_{fov} と、crop として、表 2.2 に示す。本研究では、クロップ率をパラメータとして定めると、 v_{fov} を求めることができるために、クロップ率 crop に対応する視野角を用いる。

表 2.2: 水平視野角の指定値 v_{fov} とクロップ率 crop との対応

| v_{fov} [degree]: | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 19 | 20 | 21 | 23 | 25 | 28 | 30 | 34 | 39 | 44 | 52 | 63 | 78 | 102 |
|---------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|-----|-----|-----|-----|-----|-----|
| crop: | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 | 1.6 | 1.7 | 1.8 | 2 | 2.2 | 2.4 | 2.6 | 2.8 | 3.1 | 3.5 |

また、本実験で使用した環境について表 2.3 に示す。

2.5.3 定式化モデルの検証方法

DS 走行環境においてピクセル倍率を実測する。まず、動画編集ソフトを用いて、映像中のポール間距離を画面上の左右のサイズになるように拡大し、左右のサイズを拡大率で割ることで、走行映像におけるポール間ピクセル数 $n_{p \rightarrow p}$ を求める。これを基準視野角

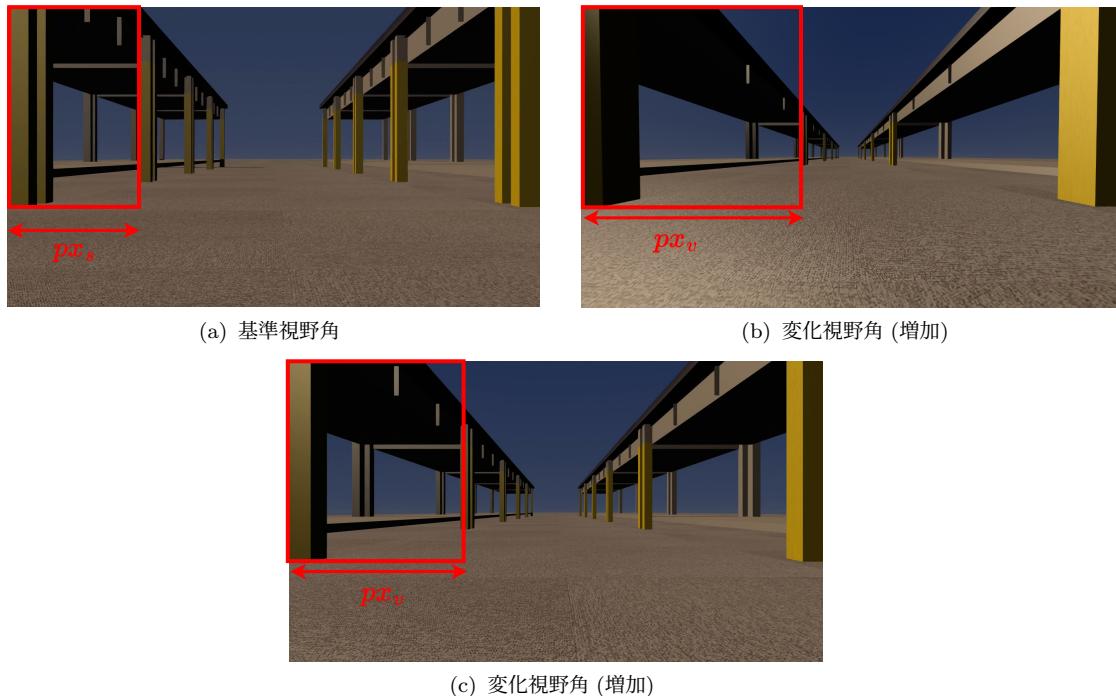


図 2.18: 視野角の違いによるポール間ピクセル数の違い

表 2.3: 本研究で使用した環境

| 環境 |
|--|
| DS:Asetto Corsa 1.16 |
| ディスプレイ:iiyama ProLite P2474HS 23.6 インチ full HD 1920x1080 60 Hz |
| MOD:ACCContentManager 0.8.2569.39678 |
| カメラ:gopro hero8 |
| CG ソフト:Blender |

と視野角を変化させた時で測定し、基準視野角に対する、ある視野角でのピクセル数の比を求めて、体感速度比とする。次に、体感速度比と実際の速度を掛けて体感速度を導出する。以上の手順で体感速度を求めて、視野角との特性をグラフ上にプロットする。定式モデルに、実験条件で示した視野角と先に求めた拡大率を代入し、体感速度を導出する。定式モデルの視野角と体感速度の特性をグラフ上にプロットする。定式モデルでの拡大率は、視野角の変数で表すことが困難であるため、実測値を用いる。

2.5.4 体感速度を変化させる効果の検証

定式化モデルが体感速度を変化させる効果を与えることの検証として、以下の手順を行う。

1. 基準視野角(垂直 28 度)、実速度が基準速度と等しく 10 [km/h]、体感速度が基準速度と等しい映像を取得する。

2. ある視野角(垂直 102 度)、実速度 10 [km/h]、体感速度 17 [km/h] での走行映像を取得する。
3. 基準視野角(垂直 28 度)、実速度 17 [km/h]、体感速度 17 [km/h] での走行映像を取得する。
4. 基準視野角・実速度 17 [km/h]、体感速度 17 [km/h] の映像と、ある視野角(垂直 102 度)、実速度 17 [km/h]、体感速度 17 [km/h](単位時間のポールの移動ピクセル数が 1 の時の 1.7 倍になったもの)の映像を取得する。

2.5.5 実験結果

2.5.5.1 定式化モデルと実測値の比較

実験結果を図 2.19、2.20 に示す。

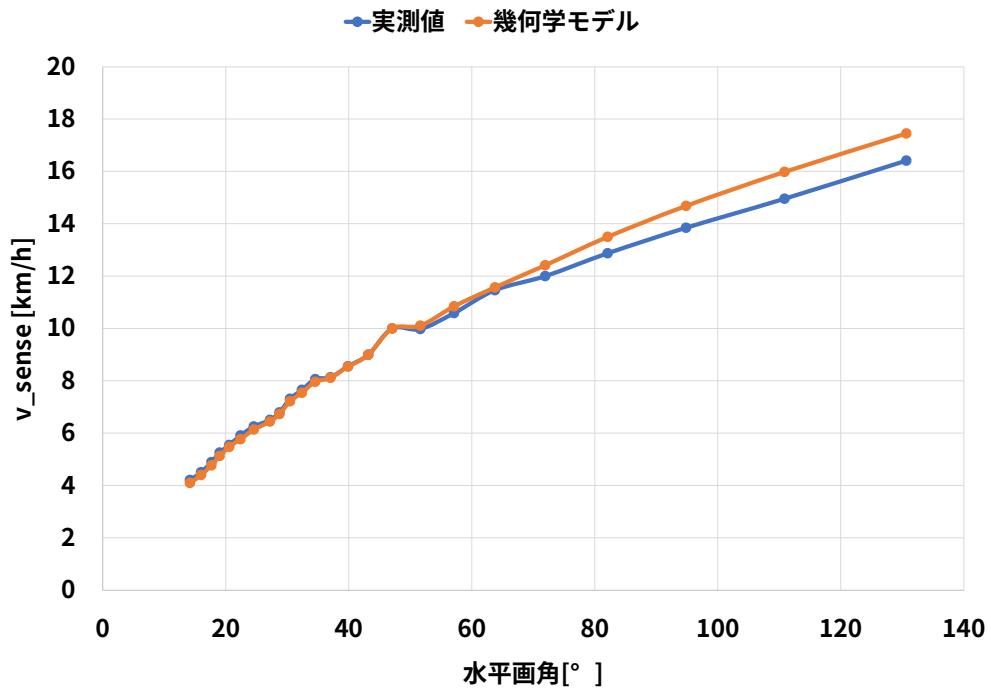


図 2.19: 水平視野角と体感速度の特性

幾何学モデルの理論式を式 (2.27)～(2.30) に示す。 $n_{p \rightarrow p}$ は、実測である。

$$v_{sense} = 19.30 \left(1 - \frac{\tan \frac{h_{fov}}{2 \cdot n_{p \rightarrow p}}}{\tan \frac{h_{fov}}{2}} \right) \quad (2.27)$$

$$h_{fov_v} = 2 \arctan \left(\frac{D_h}{2f \cdot n} \right) \quad (2.28)$$

$$n = \frac{D_h}{2f \tan \frac{h_{fov_v}}{2}} \quad (2.29)$$

$$v_{fov_v} = 2 \arctan \frac{D_v}{2f \cdot n} \quad (2.30)$$

以上の式 (2.31)、(2.32) を用いて、 v_{fov_v} を、 h_{fov_v} に変換し、 h_{fov_v} を変数とする。

$$h_{fov_v} = 2 \arctan \frac{D_h}{2f \cdot n} \quad (2.31)$$

$$v_{sense} = 19.30 \cdot \left(1 - \frac{\tan \frac{h_{fov_v}}{2 \cdot n_{p \rightarrow p}}}{\tan \frac{h_{fov_v}}{2}} \right) \quad (2.32)$$

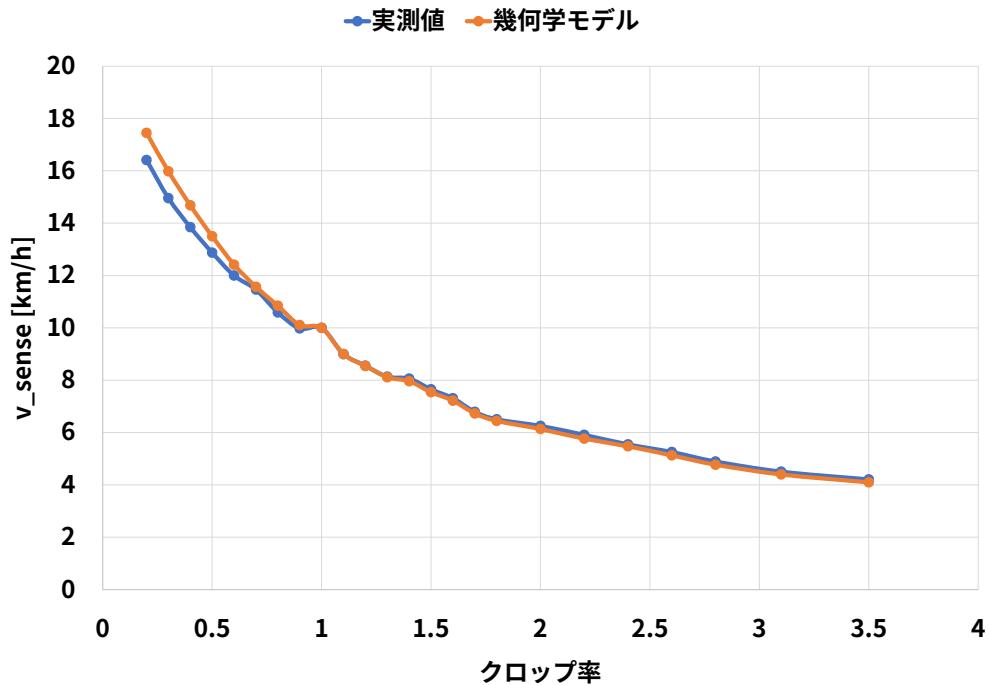


図 2.20: クロップ率と体感速度の特性

2.5.5.2 体感速度を変化させる効果の検証

実験結果を以下に示す。

1. ピクセル比と実速度比の一致の確認

基準視野角(垂直 28 度)、実速度が基準速度と等しく 10 [km/h]、体感速度が基準速度と等しい映像に関して

ポール間移動時間:5.25 秒、ポール間ピクセル数:480 [px]

ある視野角(垂直 120 度)、実速度 10 [km/h]、体感速度 17 [km/h] での走行映像に関して

ポール間移動時間:5.25 秒、ポール間ピクセル数:788 [px]

時間比: $5.5 \div 3.05 = 1.7213$

ピクセル比: $788 \div 480 = 1.64$

体感速度: 時間比 \times 実速度 $= 17.213 \times 10 = 17.213$ [km/h]

誤差: $\frac{17.2 - 16.4}{16.4} \times 100 = 4.878 = 4.9\%$

2. 体感速度比(時間比)とピクセル比の一致の確認

ある視野角(垂直 102 度)、実速度 10 [km/h]、体感速度 17 [km/h] での走行映像に関して、

ポール間移動距離:5.25 秒、ポール間ピクセル数:788 [px]

基準視野角(垂直 28 度)、実速度 17 [km/h]、体感速度 17 [km/h] での走行映像に関して

ポール間移動距離:2.96 秒、ポール間ピクセル数:463 [px]

時間比: $5.25 \div 2.96 = 1.7736$

ピクセル比: $788 \div 463 = 1.7019$

体感速度: $1.7736 \times 10 = 17.7$ [km/h]

誤差: $(1.7736 - 1.7019) \div 1.7019 \times 100 = 4.2129 = 4.2\%$

よって、これらの定式化が実測と同程度であることを確認した。

2.6 考察

2.6.1 定式化モデルと実測値の比較に関して

視野角-体感速度特性において、視野角の増加に伴って体感速度は緩やかに増加する。この結果は定説と一致している。特性の概形は、おおよそ視野角の n 乗根に比例すると考えられる点が仮定と一致していた。基準視野角よりも小さな視野角値である時は、比例的に増加するが、基準視野角以上の値をとると、理論値、実測値ともに体感速度の変化率は緩やかになる。また、視野角が 70 度以上の値をとると、理論値と、実測値との差が大きくなる。視野角が基準視野角よりも大きな値である映像は、基準視野角よりも小さな視野角の映像と比べて、映像の奥行きが大きくなり、映像自体が歪んでいることが分かる。このことから、定式化モデルとして、ひずみに関するパラメータを考慮する必要があることが分かる。なお、今回の定式化モデルでは、理想的な 2 次元の映像として考えていたため、

このひずみに関しては考慮していない。考慮するためには、3次元の立体映像に関する知見が必要である。クロップ率-体感速度特性について、クロップ率の増加に伴い体感速度は急激に減少する。クロップ率を増加させると視野角が減少するため、視野角の減少により体感速度が減少することが定説通りであることを確認した。クロップ率が1よりも大きな値をとる場合は、視野角を増加させていることと考えると、先に述べた点と一致している。

2.6.2 体感速度を変化させる効果の検証

ピクセル比と実速度比の一致の確認に関して、時間比と速度比は、定義通り一致していたが、単位時間当たりの移動ピクセル数と実速度比に関しては、4.9 % 程の誤差がみられた。DSでは小数点以下の速度は表示されないため、この誤差は、DS上における実速度の誤差であると考えられる。このことから、体感速度に関する仮説として、視野角の変化によって、走行映像中のポール間ピクセル数が変化していることが体感速度の変化を表していることが実証された。体感速度比とピクセル比の一致に関して、視野角をパラメータとして求めた体感速度と、その体感速度値と一致する実速度の映像では、単位時間当たりに進行するピクセルの比率がほとんど一致するため、単位時間当たりに移動するポール数は一致しないが、単位時間当たりに進行するピクセル数が一致することが確認された。この結果より、体感速度を支配するパラメータとして提案されていた、映像中の風景線が流れるスピードが一致していることが分かる。また検証により、視野角やクロップ率をパラメータとして、幾何学的に単位時間当たりに移動するピクセル比を導出する手法を用いた体感速度のモデルが正しいことが明らかになった。

2.7 おわりに

本章では、走行映像における体感速度を支配するパラメータを明らかにし、そのパラメータを用いて、体感速度の変化率のモデルを定量化した。映像のクロップ率と映像の視野角(撮影画角)が最も影響力の大きいパラメータであることが明らかになった。体感速度のモデルとして、映像の視聴環境とクロップ率と視野角を用いて幾何学的な計算によって導出したモデル(幾何学モデル)を提案した。幾何学モデルが正しいことを検証するために、移動ピクセル数の実測値との比較により、検証実験により、実速度が異なる映像でも、体感速度が一致する視野角を設定することで、単位時間当たりの移動ピクセル数が一致し、映像中の風景線が流れるスピードが一致したことから、幾何学モデルが正しいことを明らかにした。提案した幾何学モデルに一部実測値を用いている点や、視野角を大きくしたときに生じた立体的なひずみが生じることが明らかになり、立体視野角を考慮した幾何学が必要であることに関して、新たな知見を得た。この知見を基にモデルの改良に関する検討の余地があり、今後の課題とする。本研究は、人間の視野が走行映像全体に均一に焦点しているという仮定の上で行っていた。今後は、人間の集中点・集中視野によって変化する体感速度のモデルに関して研究を行う予定である。

第3章

LiDARのみのSLAMで作成された環境地図の評価

3.1 はじめに

SLAMは、GPS、GNSS等による絶対的な位置座標を得ることができない環境での自律移動ロボットへの導入ができる点で注目されている。LiDARのSLAMを行う場合は、LiDARによる位置計測(スキャン)と、ロータリーエンコーダ等の内界センサによる移動履歴(デッドレコニング、オドメトリ)を組み合わせることが一般的かつ、内界センサによるオドメトリとの併用が前提とされたアルゴリズムが主流である。

実用の現状において、つくばチャレンジ等をはじめとする屋外自律移動ロボットコンテストでは、GNSSを組み合わせている場合が多い。LiDARのSLAMにおける内界センサやGNSS等との併用のメリットは、位置情報の補正が優位であるからである。しかし、オドメトリの有無に関して、ロータリーエンコーダを用いる場合は、タイヤのスリップ等による誤差の蓄積による地図の精度の誤差をもたらし、精度の良いIMU、GNSS等の内界センサを用いる場合は、導入コストがかかるといったデメリットが考えられる。LiDAR導入のメリットは、カメラを用いる場合での欠点であった、測定精度や暗闇であっても精度に影響がなく、夜間や雨天時に利用できることである。SLAMにおける、LiDARのハードウェア的な要求スペックに関しては、2次元の地図作成のみの機能に限定するならば、ほとんど限定されない。むしろ、LiDARを用いたSLAMアルゴリズムの性能に依存する。

LiDARを用いたSLAMのアルゴリズムに関しては、いくつかの種類がある。LiDARによって計測された点群のノイズの処理手法に関するフィルタの種類(ガウシアンフィルタ・ベイズフィルタ)や、オドメトリの有無、ループクローズ機能等の有無等によって分類される。各々のSLAMアルゴリズムの作者による予想は立てられているが、作者による評価は行われておらず、ユーザーによる実機ロボットを用いた評価自体は行われていていることが確認できている。その評価手法(実験環境)に関しては、LiDARのSLAMアルゴリズムにおいて、内界センサとの併用が前提とされていて、LiDAR単体のみでのSLAMアルゴリズムの性能指標に関しては、着目されていない。LiDARのみでのSLAMアルゴリズムの性能を保証することができれば、SLAMアルゴリズムにおけるLiDAR単体での性能を把握することで、LiDARのスペックに応じたSLAMアルゴリズムの適性を把握することができ、その適性により、内界センサ等に含まれる誤差(スリップ等)を無効化や、自律移動ロボットの制作コスト削減に寄与することができる。そこで、LiDARのみでのSLAM

アルゴリズムの性能の妥当性を明らかにすることを目的として、内界センサによるオドメトリを用いず、ROSで実装されている各LiDAR SLAMパッケージを用いたLiDARのみによる環境地図作成システムを提案する。オドメトリフリーの手法に対して、オドメトリが必要な手法にLiDARから得るレーザオドメトリを用いて比較することで、各種SLAMにおけるLiDAR単体での性能指標の評価を行う。内界センサによるオドメトリを無しにすることによって、地図の精度が悪化することが知られているため、LiDARそのものからオドメトリをとる手法を用いることで、LiDARのみでひずみの少ない地図を作成することを目指す。

3.2 従来研究

3.2.1 SLAMの概要

SLAM (Simultaneous Localization And Mapping) とは、ロボットが地図を作る方法の1つで、同時に自己位置推定 (Localization) と地図構築 (mapping) を行うことである。両者には相互依存関係があり、同時に行う必要がある。SLAMを用いなくても、地図構築は可能であるが、その場合、ロボットの位置を別途推定する仕組み(既知のランドマークやGPSなど)が必要である。ロボットに搭載したセンサだけを用いて地図構築を行う場合、SLAMの考え方は本質になる。

3.2.1.1 SLAMの効果

SLAMでは、ロボットが移動しながらセンサで周囲を計測し、移動軌跡に沿って地図を作る。SLAMの入力はセンサデータであり、出力はロボットの移動軌跡と地図である。図3.1は、後述のレーザスキャナというセンサで得たデータ(スキャンと呼ぶ)から地図ができる段階を示したものである。ロボットの位置が分からない場合、センサデータだけを並べても、図3.1のように地図とは言えない。後述のオドメトリを用いることで、SLAMによらず、ロボットの位置を求めることができる。その軌跡に沿って周囲形状を表すセンサデータを並べると、図3.2のようになる。一般にオドメトリだけだと、移動量が多くなる度にロボットの位置の誤差は大きくなっていく。そのためこの地図は歪んでいる。

SLAMを用いることで、この歪みを減らすことができる。SLAMには大きくレベルが2段階あり、ループ閉じ込みという技術を用いないレベルでは図3.3のように局所的なぶれや歪みは小さくなるが、全体的な歪みは残る。ループ閉じ込みを行うレベルでは図3.4のように全体形状の歪みも小さくなった地図をえることができる。このようにSLAMは、ロボット位置と地図の推定を同時に行うことで、地図の精度を向上させる技術である。

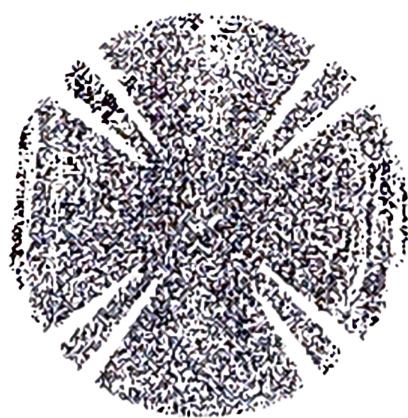


図 3.1: 全スキャンを同じ位置においた結果

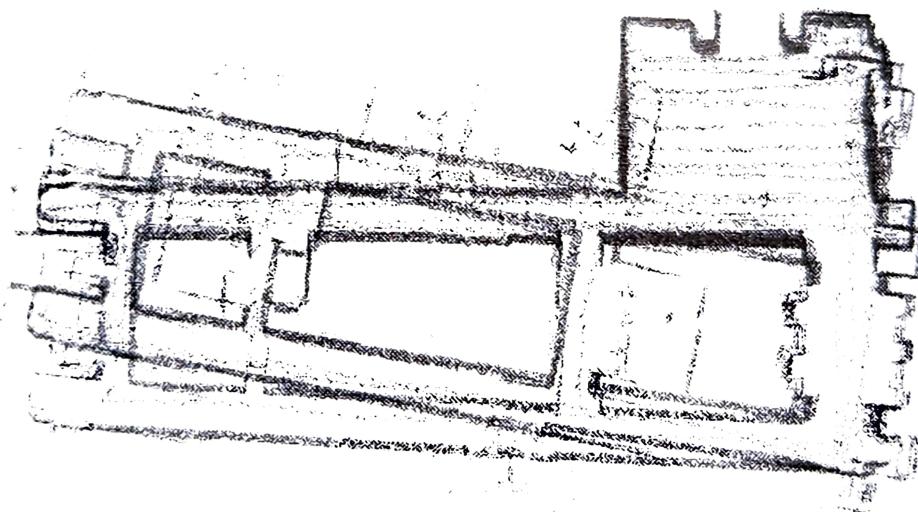


図 3.2: オドメトリによる地図

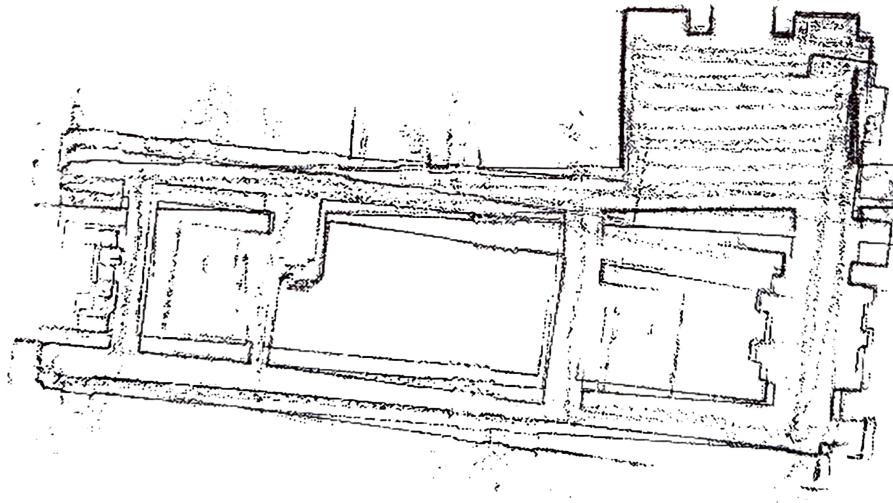


図 3.3: SLAMによる地図(ループ閉じ込みなし)

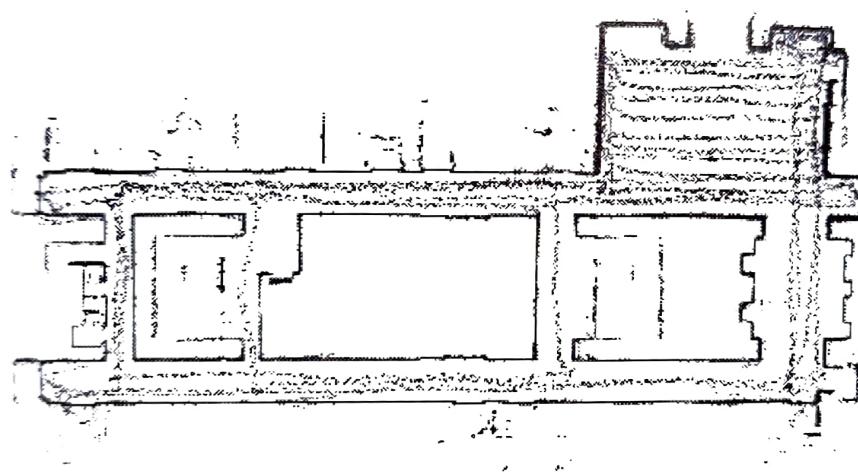


図 3.4: SLAMによる地図(ループ閉じ込みあり)

3.2.1.2 SLAMの実行形態

SLAMを行う際にまず問題となるのは、センサデータ取得のための走行経路をどのように決めるかである。ロボットは地図を持っていないので、どういう経路を通るのが安全で効率が良いか、あらかじめ知ることはできない。そこで、以下の2つの方法が考えられる。

1. 人間がセンサデータの取得経路を与える

対象環境をよく知っている人間がロボットにセンサデータ取得経路を与える。たとえば、人間がロボットを操縦してセンサデータを集める場合がこれにあたる。地図の生成は、操縦と同時に行う場合も、データ収集後にオフラインで行う場合もある。目的地への自律走行(ナビゲーション)を行う場合は、この方法で地図をつくることが多い。ナビゲーションでは、あらかじめ作成した地図の上で目的地への経路計画と自己位置推定を行うため、その場で地図を作る必要はないからである。ただし、ナビゲーションの最中に障害物や環境変化に対処するために、局所的な地図をその場で作って対処することは良くある。

2. ロボットがセンサデータの取得経路を自律的に決める

ロボットが自律走行してセンサデータを取得しつつ、リアルタイムに地図をつくる。その際、それまでに生成した地図と現在地周辺の局所的な地図を自律走行のために使う。ロボットは障害物を回避しながら走行し、次にどこに行くか自分で決める必要があるため、SLAMだけでなく、探査、経路計画障害物回避なども含んだ総合的なシステムが必要になる。このようなシステムは、人間が操縦や遠隔操作をするのが難しい環境、あるいは、地図構築の知識をもったオペレータを用意できないアプリケーションなどで重要になる。理想はロボットが自律で地図を作ることであるが、実用においては目的に応じて使い分ければ良い。

3.2.1.3 SLAMの種類

センサや地図には多くの種類があり、それに応じてSLAMの手法も変わる。ここではSLAMの大きな分類軸として、2次元か3次元かを考える。ロボットの位置(運動)と地図が、それぞれ2次元か3次元かで、原理的には4通りの分類ができる。ここでは、ロボット位置/地図の次元によって、SLAMを2D/2D型、2D/3D型、3D/3D型に分けたものを表3.1に示す。(3D/2D型は事例が少ないので除外する。)

表3.1: SLAMの種類

| ロボット位置 | 地図 | センサ |
|--------|---------|------------------------------------|
| 2D/2D型 | 2次元3自由度 | 2次元 2Dレーザスキャナ、オドメトリ、ジャイロ |
| 2D/3D型 | 2次元3自由度 | 3次元 2D/3Dレーザスキャナ、カメラ、オドメトリ、ジャイロ |
| 3D/3D型 | 3次元6自由度 | 3次元 3Dレーザスキャナ、カメラ、IMU |

これらのどの型を選ぶかは、ロボットの種類、タスク、コスト等によって変わる。一般に2D/2D型、2D/3D型、3D/3D型の順に機能が高いと言えるが、コストもかかる機能的には3D/3D型が最も望ましいが、処理量が多いため、多くの場合、高性能コンピュータが必要になる。ただし、最近は、地図の規模が小さければ、携帯端末レベルのコンピュータでも、3D/3D型SLAMが可能になっている。移動機能を持ったロボットには、車輪型ロボット、クローラー型ロボット、脚型ロボット、ヒューマノイド、飛行ロボット等がある。例えば、ヒューマノイドや飛行ロボットのように3次元の運動をするロボットには、3D/3D型が必要である。車輪型ロボットは、地面や床面等の平面を走ることが多いので、2D/2D型が良く使われている。ただし、障害物の検出には2D/3D型が有利であり、もし坂道や段差等を考慮するのであれば、車輪型ロボットでも3D/3D型SLAMが必要であると考えられる。

現在のSLAMシステムは、レーザスキャナを使ったものとカメラを使ったものに大別できる。カメラはさらに、単眼カメラ、ステレオカメラ、距離画像カメラに分かれる。2D-SLAMはレーザスキャナを用いるものが多く、また、古くから研究がおこなわれていたので、現在では実用レベルになっている。代表的なものにGmapping[7]やCartographer[8]等がある。3D-SLAMには、3Dレーザスキャナかカメラ、あるいはその両方が用いられる。

3.2.2 SLAMの原理

本節では、SLAMの原理を模式的に述べる。前提として、ロボットに搭載されたセンサのみを使う。GPSなど、絶対位置が直接得られるセンサや装置は使わない。簡単のため、ロボットにおけるセンサの設置個所は既知とし、両者の座標系は同一視する。ロボット位置 x は方向も含めて $(x, y, \theta)^T$ と表す。 x, y は、床面等の2次元座標系での位置(並進成分)、 θ はロボットが向いている方向である。なお、右肩のTは、ベクトルの転置を表す。

図3.5にロボットで地図を作る様子を示す。地図はランドマークだけで構成されているとする。ランドマーク位置は点 q で表す。ランドマークに向きはない。この点では点がまばらにあるだけなので、障害物等の形状を表せず、地図としては不十分であるが、ロボットが位置を知るために使える。ここでも、ロボットは各ランドマークを区別できると仮定する。

3.2.2.1 地図構築: ランドマーク位置の推定

前述のように、ロボット位置が分かれればランドマーク位置は計算できる。図3.6～図3.7に例を示す。この図では、図3.5のロボット位置 x_1 からランドマーク q_2 を見る部分に注目している。 q_2 の地図座標系での位置を $q_2 = (q_{2,x}, q_{2,y})^T$ また、センサの計測データ(センサ座標系の値)を $z_2 = (z_{2,x}, z_{2,y})^T$ とする。ここで、ロボット位置 $x_1 = (x_1, y_1, \theta_1)^T$ が分かれれば、 q_2 は式(3.1)のように計算できる。

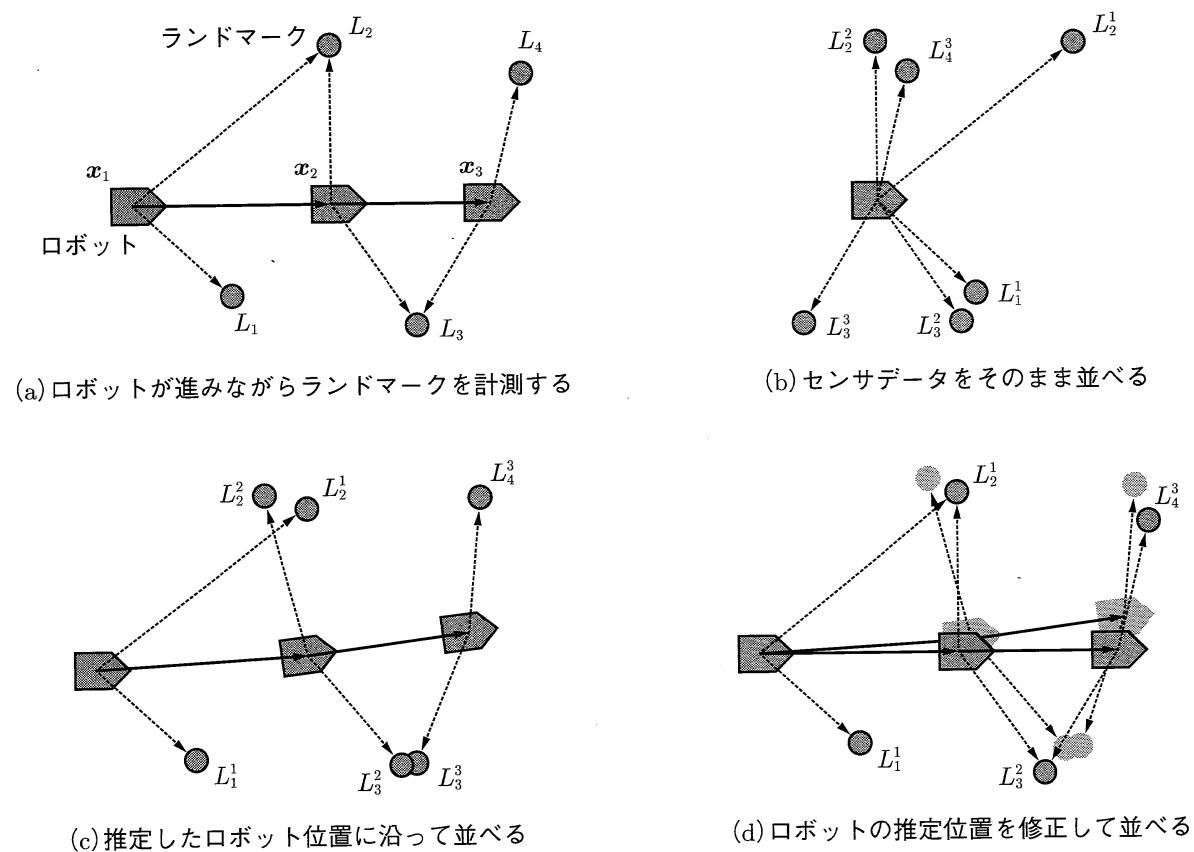


図 3.5: ロボットで地図を作る様子

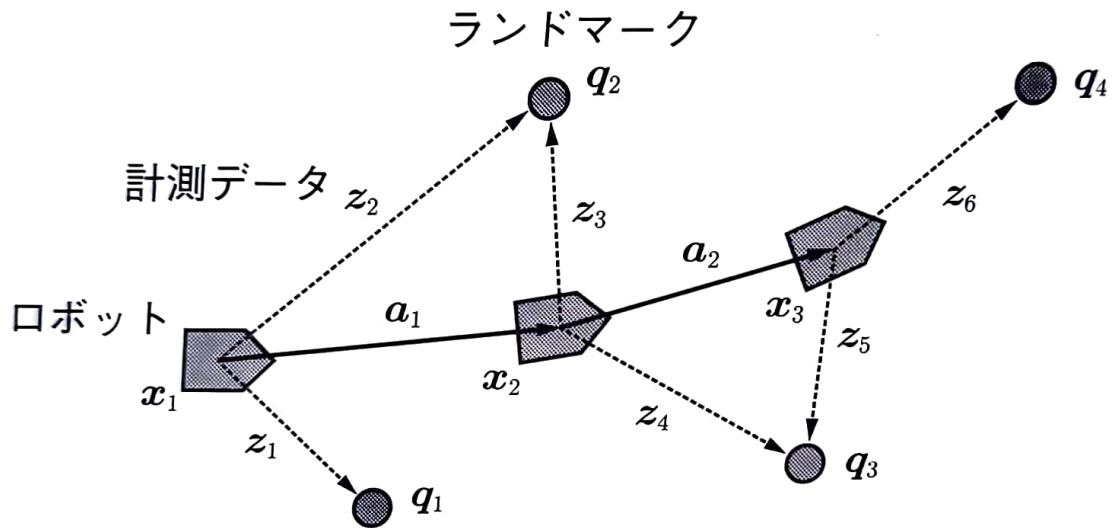


図 3.6: ロボットとランドマークの位置推定

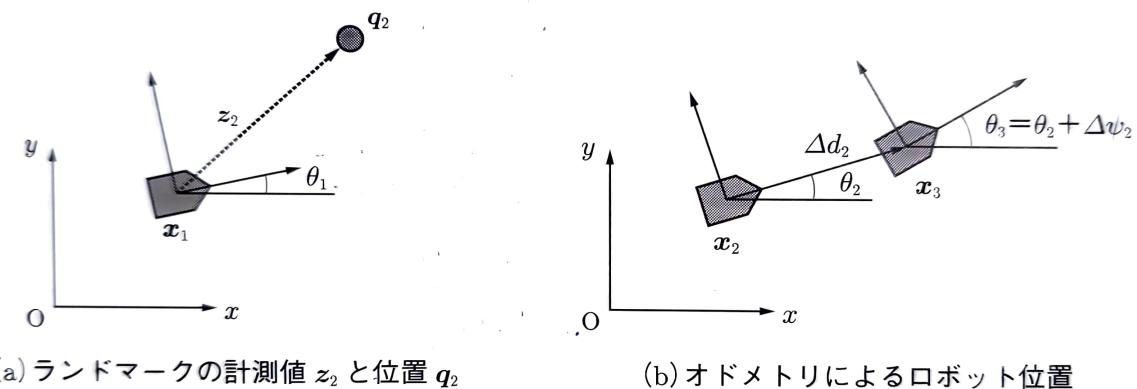


図 3.7: ランドマーク位置の推定とロボット位置の推定

$$\begin{pmatrix} q_{2,x} \\ q_{2,y} \end{pmatrix} = \begin{pmatrix} \cos \theta_1 & -\sin \theta_1 \\ \sin \theta_1 & \cos \theta_1 \end{pmatrix} \begin{pmatrix} z_{2,x} \\ z_{2,y} \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \quad (3.1)$$

ここで、式を整理すると、

$$\mathbf{R}_1 = \begin{pmatrix} \cos \theta_1 & -\sin \theta_1 \\ \sin \theta_1 & \cos \theta_1 \end{pmatrix}, \mathbf{t}_1 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \quad (3.2)$$

なお、式(3.2)は、式(3.3)のように表すことができる。

$$\mathbf{q}_2 = \mathbf{R}_1 \mathbf{z}_2 + \mathbf{t}_1 \quad (3.3)$$

なお、センサがレーザスキャナの場合、センサ中心からランドマークまでの距離 d_2 と方向 ϕ を計測する。すると、 $\mathbf{z}_2 = (d_2 \cos \phi_2, d_2 \sin \phi_2)^T$ となる。

3.2.2.2 ロボット位置の推定

ロボット位置を推定する有力な方法の1つにオドメトリ(Odometry)がある。オドメトリは、与えられた初期位置から微小変位を積分して現在位置を求める仕組みである。例えば、車輪ロボットでは、車輪の回転数から移動量を求める「車輪オドメトリ」が良く用いられる。オドメトリによるロボット位置の計算方法はいくつかあるが、ここでは、図3.7を例に、簡単な方法を紹介する。この図では、図3.6のロボットが x_2 から x_3 に移動する様子を表している。オドメトリで得た移動量(ロボット座標系での並進と回転)を $\mathbf{a}_2 = (\Delta d_2, 0, \Delta \psi_2)^T$ とする。オドメトリで得る移動量は短時間の微小量なので、瞬間的にはロボットは直進しているとみなし、 y 成分は0にする。すると、ロボット位置は x_3 は次の式(3.4)のように計算できる。

$$\begin{pmatrix} x_3 \\ y_3 \\ \theta_3 \end{pmatrix} = \begin{pmatrix} \cos \theta_2 & -\sin \theta_2 & 0 \\ \sin \theta_2 & \cos \theta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \Delta d_2 \\ 0 \\ \Delta \psi \end{pmatrix} + \begin{pmatrix} x_2 \\ y_2 \\ \theta_2 \end{pmatrix} \quad (3.4)$$

なお、図3.6の \mathbf{a}_2 は、微小量である。この計算は、回転成分を含めた座標変換といえるため、compounding演算子 \oplus で表すことがある。compounding演算子を用いると、式(3.4)は次のように表すことができる。

$$\mathbf{x}_3 = \mathbf{x}_2 \oplus \mathbf{a}_2 \quad (3.5)$$

オドメトリで位置を求めるには、最初に初期位置 x_1 を与え、そこを起点として式(3.4)によって微小な移動量を次々に加算していく。そうすると、ロボットの位置を時間経過ごとに計算することができる。しかし、オドメトリを移動量を加算していく積分計算なので、移動量に含まれる誤差も加算される。このため、ロボット位置の誤差が累積してどんどん大きくなる。

どん増えていくという問題がある。そのため、図3.2に示すように、オドメトリによる推定位置は走行するにつれてずれていく。人間に例えると、目をつぶって歩数で距離を推測するようなものであり、歩いているうちに位置がずれる。

3.2.2.3 ロボット位置とランドマーク位置の同時推定

オドメトリには「走行するにつれて誤差が累積する」という問題がある。そのため、図3.5のロボット位置のずれは、これを推定したものである。この累積誤差を減らすには、センサデータを地図に登録されたランドマークと照合して、ロボット位置を修正する方法が有効的である。目をつぶって歩きながら、時々薄目を開けてランドマークを見て、自分がどこにいるか確認するようなものである。この時重要なのは、同じランドマークを複数回計測することである。例えば、計測データ z_3 からも、式(3.6) ロボット位置 x_2 に関する計算式が得られる。

$$q_2 = R_2 z_3 + t_2 \quad (3.6)$$

既に計測したランドマークからロボット位置を逆算して修正するために、式(3.3)によって q_2 が既知になったと考え、その q_2 を使って、式(3.6)を用いて x_2 を逆算すれば、ロボット位置を修正できる。ただし、この式だけでは、制約が足りず逆算できないので、より多くの計算式を立式する。図3.6において、オドメトリとランドマーク計測の計算式を全て立式すると、以下のような連立方程式となる。

$$\begin{aligned} q_1 &= R_1 z_1 + t_1 \\ q_2 &= R_1 z_2 + t_1 \\ q_2 &= R_2 z_3 + t_2 \\ q_3 &= R_2 z_4 + t_2 \\ q_4 &= R_3 z_1 + t_3 \\ x_2 &= x_1 \oplus a_1 \\ x_3 &= x_2 \oplus a_2 \end{aligned}$$

これらの式の変数は、ロボット位置 x_i とランドマーク位置 q_j である。ただし、地図がない時点でのロボットの初期位置 x_1 は適当な定数(多くの場合、地図座標系の原点)にする。そのため、変数としてのロボット位置は2個になる。そうすると、ロボット位置は3次元、ランドマーク位置は2次元なので、これらの式の変数は全てで $3 \times 2 + 2 \times 4 = 14$ 個、式は、 $2 \times 6 + 3 \times 2 = 18$ 個となり、変数より式の方が多い連立方程式となる。変数より式の方が多い連立方程式には、多くの場合、厳密な解が存在しないので、最小二乗法を用いて解くのが一般的である。ただし、これらの式は非線形関数である三角関数を含むので、非線形最小二乗問題となる。一般的に、センサデータには誤差があり、ロボット位置や地図の推定値にも誤差が波及する。連立方程式を最小二乗問題として解くことは、推定値の

誤差を減らすことにもなる。

3.2.2.4 外界センサが一度に大量に得られる場合

外界センサによる 1 回の計測で 1 回の計測で多くのランドマークデータを取得できれば、オドメトリ無しでもロボット位置を決められる。図 3.8 を例として考える。レーザスキャナやカメラを用いる場合にこれが可能になる。

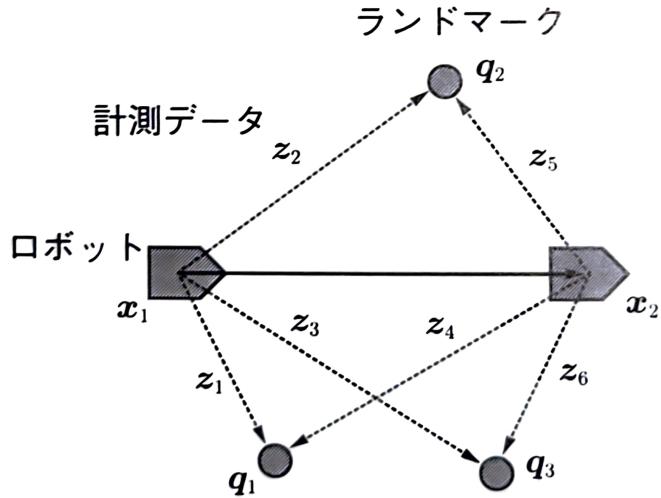


図 3.8: 外界センサデータが一度に大量に得られる場合

この図より得られる式は、式 (3.7) に示す。

$$\begin{aligned}
 q_1 &= R_1 z_1 + t_1 \\
 q_2 &= R_1 z_2 + t_1 \\
 q_2 &= R_1 z_3 + t_1 \\
 q_3 &= R_2 z_4 + t_2 \\
 q_3 &= R_3 z_5 + t_2 \\
 q_4 &= R_3 z_6 + t_2
 \end{aligned} \tag{3.7}$$

ロボット位置は 3 次元、ランドマーク位置は 2 次元なので、これらの式の変数は全部で $3 \times 1 + 2 \times 3 = 9$ 個、式は $2 \times 6 = 12$ 個となる。変数より式の方が多いので、前と同様に最小二乗法により解を求めることができる。この連立方程式にはオドメトリの制約が含まれていないが、このように外界センサデータだけでロボット位置を求めることができる。上記の例はランドマークまでの距離と方向を計測できるレーザスキャナを用いた場合である。レーザスキャナのデータだけからロボット位置を求める方法には、例えば「スキャンマッチング」が挙げられる。

3.2.3 要素技術

ここではSLAMで必要となる要素技術について説明する。具体的にはこれまでの原理を実現する方法について述べる。

3.2.3.1 不確実性の扱い

センサデータには多くの誤差が含まれており、自己位置推定や地図構築に不確実性をもたらす。ここでは、偶然誤差の扱い方について述べる。偶然誤差はランダムに発生する確率で、原因が特定できないため、確率的に扱う必要がある。前節の連立方程式には誤差が明示的に考慮されておらず、方程式自体はいわば誤差の無い理想的な状態を表しているが、現実ではそうはならない。そこで、不確実性を扱うために、誤差を明示的に表す。式(3.3)を再喝する。

$$\mathbf{q}_2 = \mathbf{R}_1 \mathbf{z}_2 + \mathbf{t}_1$$

これを次のように変形して、ロボット(センサ)座標系で見たランドマーク位置を考える。

$$\mathbf{z}_2 = \mathbf{R}_1^{-1}(\mathbf{q}_2 - \mathbf{t}_1) \stackrel{\text{def}}{=} \mathbf{h}(\mathbf{x}_1, \mathbf{q}_2) \quad (3.8)$$

ここで、 $\stackrel{\text{def}}{=}$ は定義を表す。 \mathbf{z}_2 は、実際の計測値、 \mathbf{h} は、ロボット(センサ)座標系でのランドマーク位置を計算する関数である。この式では等号を結んでいるが、実際は計測値 \mathbf{z}_2 に誤差があるため左の等式は成り立たない。そこで誤差を導入する。誤差は計測値と真値の差である。 \mathbf{x}_1 と \mathbf{q}_2 は変数であるが、一旦それらを真値とみなすと、 \mathbf{z}_2 の誤差 \mathbf{v}_2 は次のようになる。

$$\mathbf{v}_2 = \mathbf{z}_2 - \mathbf{h}(\mathbf{x}_1, \mathbf{q}_2) \quad (3.9)$$

多くの誤差は正規分布に従うことで知られており、SLAMでも正規分布が最もよく使われる。そこで、ここでの誤差 \mathbf{v}_2 も正規分布に従うと仮定することにする。正規分布は平均と分散(多次元の場合は共分散行列)の2つのパラメータで定義できるので、 \mathbf{v}_2 も正規分布に従うと仮定する。正規分布は平均と分散(多次元の場合は共分散行列)の2つのパラメータで定義できるので、 \mathbf{v}_2 の平均を $\boldsymbol{\mu}_2$ 、共分散行列を Σ_2 とすると、 \mathbf{v}_2 は式(3.11)の正規分布で表される。

$$p(\mathbf{v}_2) = \frac{1}{\sqrt{|2\pi\Sigma_2|}} \exp\left\{-\frac{1}{2}(\mathbf{v}_2 - \boldsymbol{\mu}_2)^T \Sigma_2^{-1} (\mathbf{v}_2 - \boldsymbol{\mu}_2)\right\} \quad (3.10)$$

$$p(\mathbf{v}_2) = \frac{1}{\sqrt{|2\pi\Sigma_2|}} \exp\left\{-\frac{1}{2}(\mathbf{z}_2 - \mathbf{h}(\mathbf{x}_1, \mathbf{q}_2))^T \Sigma_2^{-1} (\mathbf{z}_2 - \mathbf{h}(\mathbf{x}_1, \mathbf{q}_2))\right\} \quad (3.11)$$

これを計測モデル(観測モデル)という。オドメトリの誤差も同様に式を例にすると、関

数 \mathbf{g} を使って、次のように表す。

$$\mathbf{x}_3 = \mathbf{x}_2 \oplus \mathbf{a}_2 \stackrel{\text{def}}{=} \mathbf{g}(\mathbf{x}_2, \mathbf{a}_2)$$

これに、誤差 \mathbf{u}_2 を導入し、次のように表す。

$$\mathbf{u}_2 = \mathbf{x}_3 - \mathbf{g}(\mathbf{x}_2, \mathbf{a}_2)$$

上と同様に、誤差 \mathbf{u}_2 を導入して、式(3.12)のように表す。

$$p(\mathbf{u}_2) = \frac{1}{\sqrt{|2\pi\Sigma_3|}} \exp\left\{-\frac{1}{2}(\mathbf{x}_3 - \mathbf{g}(\mathbf{x}_2, \mathbf{a}_2))^T \Sigma_2^{-1} (\mathbf{x}_3 - \mathbf{g}(\mathbf{x}_2, \mathbf{a}_2))\right\} \quad (3.12)$$

と表す。これを運動モデルと呼ぶ。前の連立方程式の全誤差の確率密度 p は、 u_i, v_j の確率密度 p を掛け合わせたものになる。

$$p = \prod_{i=1}^2 p(\mathbf{u}_i) \prod_{j=1}^6 p(\mathbf{v}_i) \quad (3.13)$$

この式は SLAM の構造を完全に表しており、この確率密度 p を推定する問題は完全 SLAM 問題と呼ばれている。この式は、最終的には、非線形最小二乗問題に帰着される。

3.2.3.2 データ対応付け

データ対応付け (data association) とは、「別々に計画されたセンサデータで、同じものを対応づけること」である。SLAM では、とくに、現在計測したセンサデータと、地図に登録されたランドマークの対応付けが重要になる。「ロボットはランドマークを区別できる」という前提で SLAM の原理を説明した。例えば、IC タグやビーコン等で ID(識別子) が付いたランドマークであれば、ID をもとに一意に区別することが可能である。このような装置を用いてロボットが場所や物体を認識する研究もなされており、目的・応用によっては非常に有効である。一方 SLAM では、もともと存在する地形・風景・物体等を (ID のない) ランドマークとして用いることがよく行われている。ID の無いランドマークを区別してデータ対応付けを行うのは、そう簡単ではない。ID に変わる手がかりとして、ランドマーク位置と特徴量がよく使われる。

1. 位置制約による対応づけ

自己位置がある程度の精度で推定できるならば、その周辺の地図に登録されたランドマーク位置も予測できる。「自分は今地点 A にいるので、この方向にランドマーク B が存在するはず」という具合である。そうすると、周辺のランドマーク位置を予想して、現在取得したセンサデータに最も近いものを対応づけるという方法が考え付く。非常によく用いられる。

2. 特徴量による対応づけ

センサデータとランドマークになんらかの特徴量を付加して、その特徴量によって

両者の対応付けを行う方法である。特微量はセンサによって多種多様で、レーザスキャンなら局所記述子やレーザの反射強度、カメラ画像ならば局所記述子や少領域の色・形状等がある。強力な特微量ならば、それだけでかなりの対応づけができる。単純な特微量では、それだけで対応づけるのは難しいが、候補の絞り込みに使うことができる。

一般には、位置制約だけで対応付けを行うのは十分ではない。同じぐらいの位置に2つのランドマークがあった時に、どちらに対応づけるのが良いか判断するのが難しいからである。逆に、特報量を用いる場合も、それだけで対応付けを完結するのは危険である。形状や外見が良く似たランドマークは区別が難しいからである。このため、特微量を用いて対応候補を減らした上で位置の探索範囲が狭く、しかも、一度に多くのセンサデータを計測できる場合は、位置制約だけで対応づけが上手いく可能性が高くなる。探索範囲が狭いために、対応付けの候補が少なく、しかも、同時に計測したデータ間の位置制約も使えるからである。「スキャンマッチング」という手法では、この性質を利用する。

3.2.3.3 センサ融合

前のように、大量データを一度に取得できる外界センサがあれば、基本的には、オドメトリ無しでSLAMを行うことが可能である。しかし、実際には、そのような外界センサを用いても状況によっては大量データを得られないことがあり、センサデータに十分な情報が含まれていない場合もある。例えば、極端な場合、センサデータにランドマークが1つも含まれていない場合もある。例えば、極端な場合、センサデータにランドマークが1つも含まれていなければ、ロボット位置は推定できない。この場合は、オドメトリを使うしか方法はない。さらにセンサデータは取れているが、情報が不足している場合等が考えられる。オドメトリを使わない場合、一度に得たセンサデータにランドマークが1個しか含まれていなければ、それだけではロボット位置を確定できない。前節でのSLAMの定式化では、オドメトリを用いていたので、ランドマークが1個でも、連立方程式の一部として、ロボット位置の推定に活かすことができた。しかし、オドメトリを使わない場合、センサデータにランドマークが1個しか含まれないとロボット位置は不定になってしまう。例えば、式(3.1)では、ロボット位置が変数3個に対して、方程式は2個しかないので、解は不定になる。一般には、センサデータにランドマークが多く含まれているほど、安定して解が得られる。ところが、センサデータにランドマークが多数含まれても、上手くいかない場合がある。例えば、長い廊下にロボットがいる場合を考えてみる。今、外界センサによって壁を計測してロボット位置を推定することを考える。廊下の2つの壁は平行なので、その地図は計測点が2本の平行な直線上に並んだようになる。同様に、センサデータも2本の平行な直線になる。壁に模様がないと仮定すると、この2本の直線を使ってロボット位置を一意に確定することはできない。地図とセンサデータは、これらの直線に垂直な方向には位置が決まるが、平行な方向にはどこの位置でも合うからである。このような状況を退化という。このように一度に大量のデータを計測できるセンサを用いたとしても、状況によっては情報が不足することがあり、ロボット位置を確定できなくなる。このため、ロボットが安定して動作するためには、1つの外界センサだけでなく、オドメトリのような他のセンサを併用することが重要である。

3.2.3.4 ループ閉じ込み

SLAMにおいて重要な概念にループがある。図3.9にロボットの軌跡のループを示す。ループとは周回路のことと、以前通った場所を再び通る経路である。SLAMによってランドマークを複数回計測してオドメトリによるロボット位置の累積誤差を減らした。これによって累積誤差は減るが、完全には消せない。それは、ランドマーク計測にも誤差があるからである。ランドマーク位置が外部からあたえられたもの(既存地図など)であれば、誤差は累積しない。しかし、SLAMにおいては、ランドマーク位置も自分で推定したものなので、ランドマーク位置に誤差が入り、そのランドマークを基準に自己位置すいているとまた誤差が生じて、というように、誤差は累積していく。SLAMでロボット位置の累積誤差をオドメトリ単体の場合よりも減らせるのは、その誤差の増加が、たいていはオドメトリ単体の場合より小さいからである。

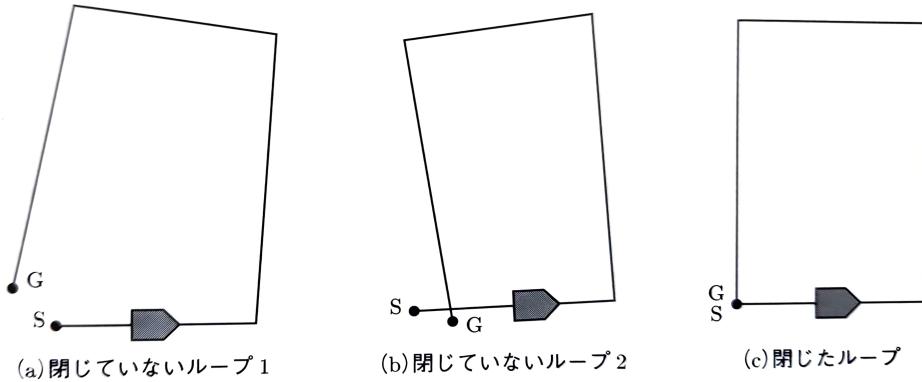


図3.9: ロボットの軌跡のループ

このように、SLAMにも累積誤差があるので、ループは多くの場合、閉じない。図3.9に例を示す。ロボットはS(スタート)から出発して1周し、Sと同じ位置にあるG(ゴール)に行くとする。この図で(a)は開いてしまった例、(b)は重なってしまった例である。このように地図が歪んで実世界との違いが大きくなれば、ロボットが壁や障害物に衝突するかもしれない。さらに困るのは、地図が歪んで実世界との違いが大きくなれば、ロボットが壁や障害物に衝突するかもしれない。さらに困るのは、地図が歪んで経路の接続関係が変わることである。たとえば、経路がつながらなかったり、偽の交差点ができたりすると目的地までの経路を上手く見つけられなくなる。このため、ループを閉じることがSLAMの重要な処理になる。これをループ閉じ込みという。(c)が、ループが閉じ状態である。ループ閉じ込みは次の手順で行う。

1. ループ検出

ロボットが同じ場所に戻ったことを検出する。ただし、まったく同じ座標に戻ることはめったになく、たいていは少しずれる。図3.10にループ閉じ込みの様子を示す。(a)は、ループがまだ閉じていない状態である。位置 x_k で、 x_1 と同じランドマーク q_1, q_2 を計測したとする。 x_k は誤差によりずれているので、それにもとづいて配置したランドマーク位置 q'_1, q'_2 もずれている。しかし、本来は同じランドマークなの

で、 q'_1, q'_2 と、 q_1, q_2 が同じになるようなロボットの位置 \hat{x}_k を求める。(a) ではループは閉じておらず、本来同じはずのランドマーク q'_1, q'_2 と、 q_1, q_2 がずれている。(b) ではループ閉じ込みにより正しいロボット位置 \hat{x}_k を見つけ、 q'_1, q'_2 は、 q_1, q_2 と一致している。

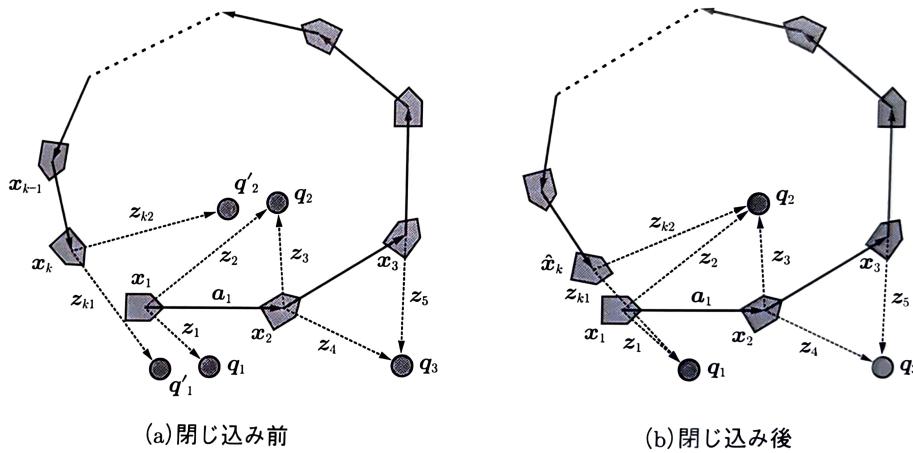


図 3.10: ループ閉じ込みの様子

2. ロボット軌跡と地図の修正

図 3.10 で、ループを閉じると、 x_k は、 \hat{x}_k の位置に来るはずである。しかしそうすると、 \hat{x}_k と x_{k-1} のずれが大きくなってしまうので、 x_{k-1} のずれが大きくなってしまうので、 x_{k-1} も修正する。こうして、逆向きにロボットの位置を修正していく。これに伴い、各ロボット位置で計測したランドマーク位置も修正する。

この手順に従うと、軌跡を逆向きに修正していくことになるが、より効率的で精度が良いのは、連立方程式を一度に解くことである。ループ閉じ込みは、上で述べた SLAM の累積誤差を解消するのに有効である。ループが大きい程累積誤差は大きくなるので、それをループ閉じ込みで解消すれば、大きな誤差が一気に修正されることになる。以上の通り、ループ閉じ込みは SLAM の重要な要素技術であり、多くの研究が行われてきた背景にある。

3.2.4 SLAM の処理形態: 一括処理と逐次処理

SLAM を連立方程式として考えると、センサデータを十分にそろえて、多くの方程式を立てた上で、一括で計算することになる。前述の完全 SLAM 問題やループ閉じ込みは、基本的には一括処理である。人間がロボットを操縦してセンサデータを集め場合は、このような一括処理で地図を作ることができる。一方、ロボットが走行している最中に地図を作りたいことも良くある。ロボットが自律的にセンサデータを集め場合は、センサデータを集めている最中にも地図が必要になる。また、既存の地図を用いてナビゲーションを行っている場合でも、地図構築時とは環境が変化している可能性があるため、安全の

ために周囲の局所地図をリアルタイムで作り続けることが重要である。このような場合は、連立方程式が全てそろうのを待たずに、それまでに収集された小数のセンサデータで作った部分的な連立方程式を解いて、逐次的に地図を構築していくことになる。逐次的な地図構築は前節で述べたように、ロボット位置と地図を交互に求めていくことで行う。すなわち、時刻 $t - 1$ の地図に統合して、地図 t の地図を求める。以上のように、SLAMの実行形態には、一括処理と逐次処理が考えられる。それぞれの特徴を以下に示す。

1. 一括処理の特徴

- センサデータが十分多く揃ってから行うため、リアルタイム処理に向かない。
- 多くのセンサデータを用いるので、処理時間は長いが、地図の精度は良い。

2. 逐次処理の特徴

- リアルタイムで逐次的に地図を構築する。このため、未知環境や変化する環境で行動する場合に有用である。
- 一部のセンサデータしか用いないので、処理時間は短いが、地図の精度は高くない。

なお、一括処理においても、必ずしも全センサデータがそろっている必要はない。十分多くのセンサデータがあれば、一括処理する意味がある。実用上の一つの区切りは、ループを見つけるまでである。このように、両者は補完し合う性質を持つため、組み合わせて使うのが望ましい。

3.2.5 Gmapping

ここでは、SLAMの実証例であるGmappingについて述べる。Gmappingのアルゴリズムは、Rao-Blackwellized Particle Filter (RBPF)によるGrid-Based SLAMである。RBPFはベイズフィルタ系のパーティクルフィルタ(PF)の一種であり、SLAMにおいては、ロボット位置をPFで、地図を他の何らかのフィルタで推定するものと捉える。ここでは、各パーティクルが現在のロボット位置だけでなく過去の走行軌跡も保持できる性質を利用している。すなわち、パーティクルごとの走行軌跡に基づいて、各パーティクルで独立に地図を推定する。パーティクルの数だけ、地図の仮説、ここでは占有格子地図が生成される。占有格子地図の推定にはバイナリベイズフィルタによって、確率的に求められる。Gmappingは、文献[7]に示された格子ベースFast SLAMを実装したものである。以下にFastSLAMの流れを示す。

1. 事前推定

- 動作モデルに従ってサンプリング
- 観測に従ってサンプルを更新(作成途中の地図情報を利用)

2. 観測更新 1

- 重みを計算
3. 地図更新
 - 地図を更新する (事前推定による状態値を利用)
 4. 観測更新 2
 - 重みが最大となる粒子の状態値と地図を現時刻の推定値とする。
 - 必要ならリサンプリングする。

なお、FastSLAM にも 1.0 と 2.0 が存在し、Gmapping では FastSLAM2.0 が実装されている。FastSLAM の流れについて、FastSLAM1.0 から述べる。前提として時刻 t でロボットが移動している。今使える地図は、時刻 $t - 1$ までに作成された中途半端なものである。この状態でロボットの位置と地図を更新することを考える。図 3.11 に各々の時刻で作成された地図を示す。

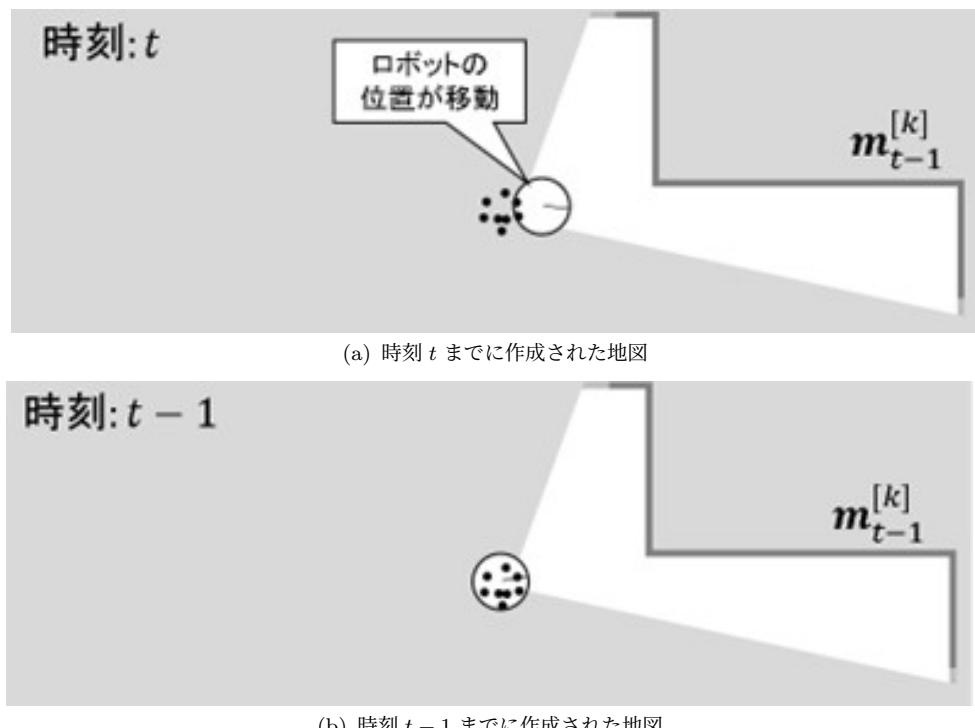


図 3.11: 各時刻において作成された地図

3.2.5.1 事前推定

今の時刻フレームでの地図を作成する前に、ロボットの動作モデル(制御指令に対する応答の予測体系: オドメトリ等)に従って、今の時刻フレームにおける自己位置の候補点(位置と姿勢)に粒子を撒き、粒子の観測を行う。動作モデルがホイールオドメトリならば、あらかじめホイールオドメトリに生じる誤差とその確率が明らかであれば、それに従って点の撒き方が決定される。図 3.12 にサンプルの撒き方の決定を示す。

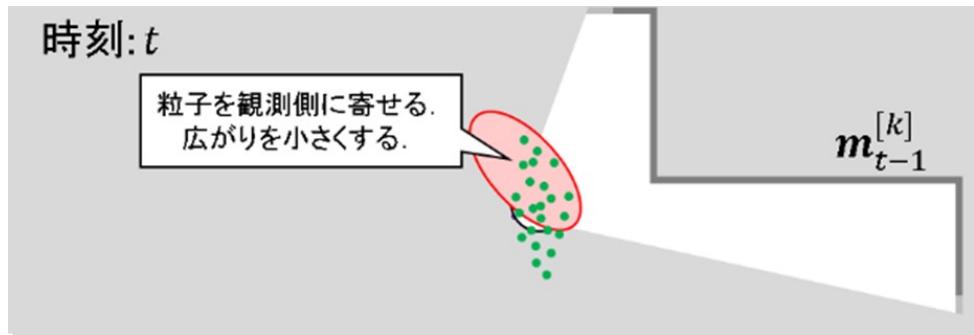


図 3.12: サンプルの撒き方の決定

3.2.5.2 観測更新 1

事前推定において撒いた各候補点から、既知の地図を観測した場合にどのように見えるかという予測と、現在センサの実観測値を照合し、各候補点の尤度を計算し、各粒子の重みを更新する。図 3.13、3.14 に、尤度による重みの調整と事前推定を考慮した尤度の調整を示す。

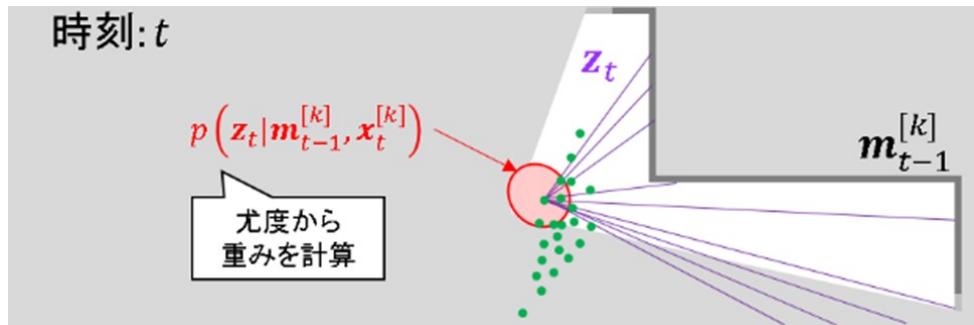


図 3.13: 尤度による重みの計算

3.2.5.3 地図更新

各候補点それぞれを自己位置・姿勢であると仮定して、リサンプリングの前に地図を更新する。図 3.15、3.16 に自己位置の仮定と、地図の更新を示す。

3.2.5.4 観測更新 2

地図を更新した後に、現時刻の推定値を確定する。最終的に、各候補点から「重みが最大」の粒子を選択し、現在における自己位置推定結果として確定する。図 3.17 に自己位置推定結果の確定を示す。また、図 3.18、3.19 に、粒子のサンプリングと粒子の多様性の考慮を示す。

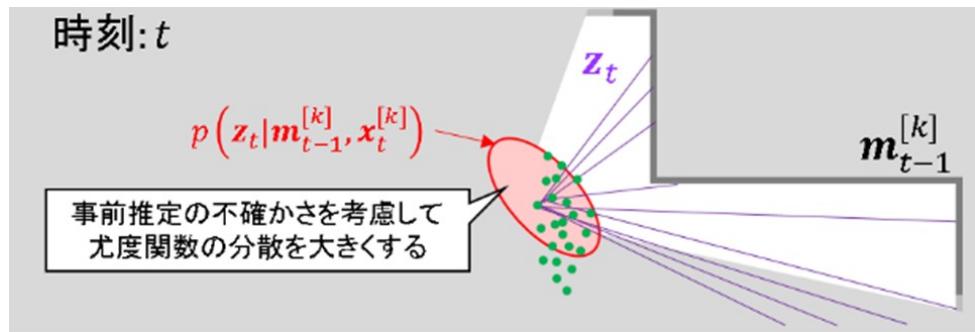


図 3.14: 事前推定を考慮した尤度の調整

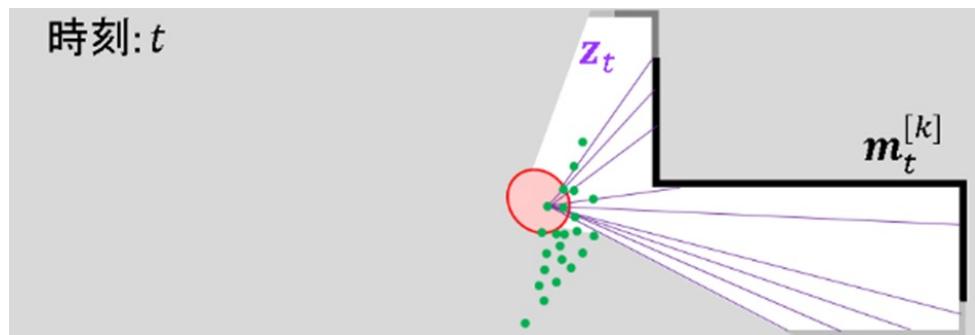


図 3.15: 自己位置の仮定

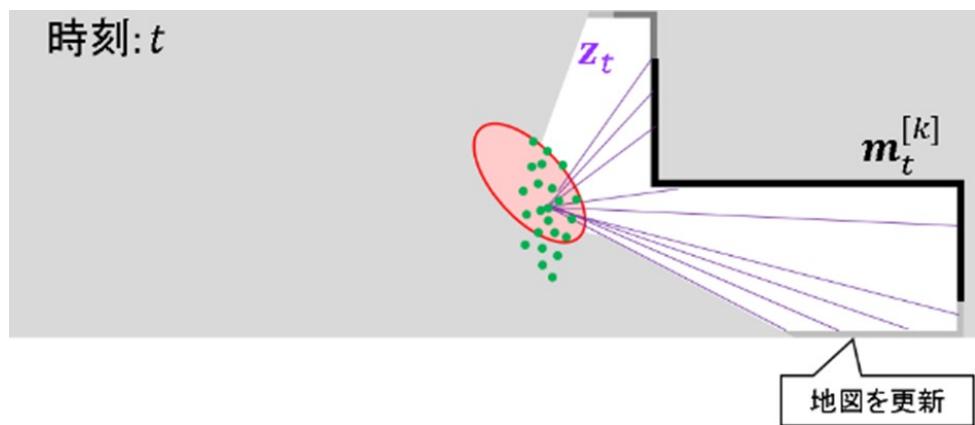


図 3.16: 地図の更新

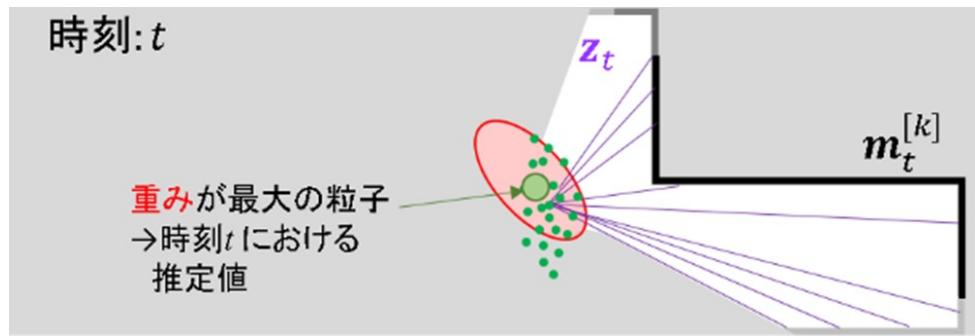


図 3.17: 自己位置推定結果の確定

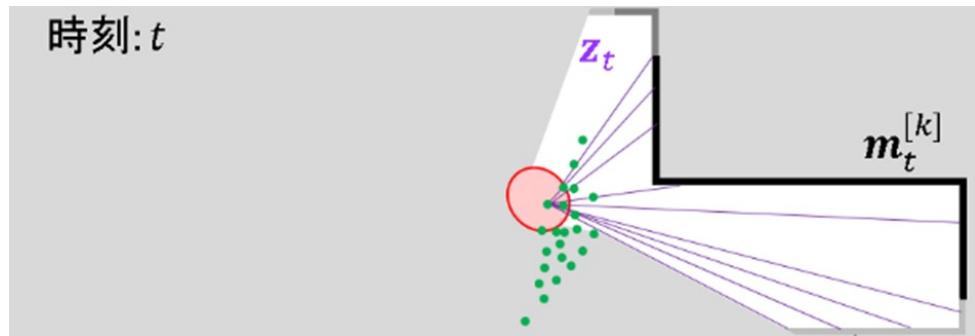


図 3.18: 粒子のリサンプリング

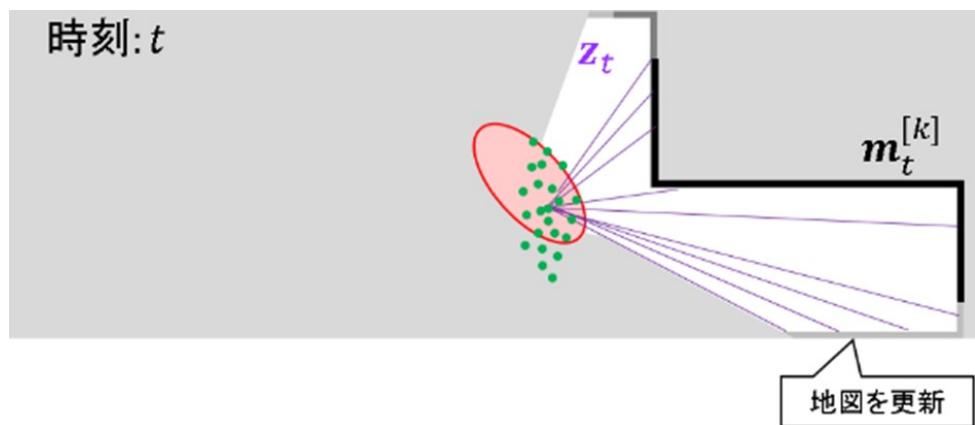


図 3.19: 粒子の多様性の考慮

3.2.6 Hector SLAM

ここでは、SLAMの実装例であるHector SLAMについて述べる。Hector SLAMは、スキャンマッチングで環境地図を構築している。スキャンマッチングは、LiDARから刻々と得られる測定データを逐次重ね合わせていき、その際に生じる測定データの移動量からロボットの移動量を推定する手法である。なお、スキャンマッチングに良く用いられる手法がICP(Iterative Closest Point)アルゴリズム[9], [10]である。Hector SLAMの場合も環境地図の構築にICPアルゴリズムの手法を取り入れている。Hector SLAMの場合、Gauss-Newton法で最適化を行い、最適されたものを式(3.14)で剛体変換している。

$$\xi^* = \arg \min \sum_{i=1}^n [1 - M(S_i(\xi))]^2 \quad (3.14)$$

ここで、 $\xi = (p_x, p_y, \psi)^T$ と、 $S_i(\xi)$ は、グローバル座標系におけるスキャンされた測定データ*i*の端点を表し、ある連続したマップの座標 P_m における占有値 $M(P_m)$ は、サブ占有格子の精度を与るためにバイリニアフィルタリングを用いて計算される。式3.15は、Gauss-Newton法を用いた $\Delta\xi$ についての最小化させる式を表し、式(3.16)はそのヘッセ行列である。式(3.15)、(3.16)は、占有値 $M(P_m)$ すなわち、ある座標 P_m におけるマップ勾配に依存する。

$$\Delta\xi = H^{-1} \sum_{i=1}^n \left[\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi} \right]^T [1 - M(S_i(\xi))] \quad (3.15)$$

$$H = \left[\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi} \right] \left[\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi} \right] \quad (3.16)$$

3.2.7 Cartographer

Cartographerは、グリッドベースのスキャンマッチングによる逐次SLAMと一括処理のグラフベースSLAMの二種類で構成されている。スキャンマッチングは、2Dの占有格子地図(Occupancy Grid Map)と、2D-LiDARから取得したスキャンデータを重ね合わせてロボットの自己位置を推定する処理である。占有格子地図とは、環境を格子状に区切って、それぞれの格子に障害物が存在する確率(占有確率)を割り当てるものである。例えば、黒い格子ほど占有確率が1に近く、白い格子ほど占有確率が0に近くなる。また、LiDARは、センサから周囲の環境に向けてレーザ光を照射し、その反射光を受光素子で検知することで、センサから見た障害物までの距離と方向を取得する。レーザ光が障害物に当たって跳ね返り、センサ中心まで戻ってくるまでの時間を計測することで、障害物までの距離を計算できる。LiDARセンサが回転しながら、あらゆる方向に対しレーザ光を照射するので、周囲の様々な障害物までの、距離と方向のペアが幾つも得られる。LiDARセンサ1周分のデータをスキャンといい、周囲360度にある障害物(家具や壁など)の形状を反映

した点群となる。スキャン同士、スキャンと地図、地図同士など、重ね合わせの対象にはいくつかの種類があるが、Scan-to-scan Matching, Scan-to-map Matching, Map-to-map Matchingなどと呼んで区別する。ここでは、スキャンと占有格子地図の重ね合わせを扱うが、スキャン同士の場合と比べると誤差が小さいとされている。ICP(Iterative Closest Point)は、スキャン同士(Scan-to-scan)の場合と比べると誤差が小さいとされている。ICP(Iterative Closest Point)とその派生、スキャン同士のマッチング手法、NDT(Normal Distributions Transform)は、スキャンと地図のマッチング手法に分類できる。(NDTは占有格子地図ではなく、各格子が正規分布を表す格子地図を使う)。スキャンマッチングは、SLAMの核となる重要な処理である。SLAMでは、ロボットの自己位置(ロボットが辿った軌跡)と、環境地図(占有格子地図)の2つを推定するが、これらの精度は、どのようなスキャンマッチングの手法を採用するかによって大きく左右される。各手法には長所と短所があるので、計算量、メモリ消費、精度などの様々な観点から、最適なものを1つ選んで用いたり、あるいは複数の手法を組み合わせて用いたりすることが重要である。グラフベースSLAMでは、明示的に再訪点を検出する。具体的には、現在地点の形状が過去の地図のある地点の形状と一致しているかを探索し、その地点が再訪点か判断する。再訪点と現在値が一致するという拘束をかけ、ロボット軌跡の最適化を行う。

3.2.7.1 分岐限定法によるスキャンマッチング

2D-LiDAR SLAMの最先端であるCartographerでは、ループ検出に、分岐限定法(Branch-and-bound)ベースのスキャンマッチング手法を用いている。ループ検出(Loop Detection)とは、ロボットが以前訪れた場所に、再び戻ってきたことを検出するための処理で、スキャンマッチングにより実現される。SLAMでは、直近のいくつかのLiDARスキャンをもとに占有格子地図を作成し、上記のような直近の観測データと最新の観測データとのマッチングだけを繰り返していくと、ロボットの位置現在位置には、誤差が蓄積していく、本来の正しい位置から大きく外れる。ループ検出では、古い観測データと最新の観測データとのマッチングを行う。これより、ある場所を訪れてから、再びそこを訪れるまでの間に溜まった誤差を一気に解消し、ロボットの現在位置を本来の正しい位置に戻すことができる。

ループ検出では、(以前その場所を訪れたときに取得した)古いスキャンを含む地図と、最新のスキャンとのマッチングを行う。直近のスキャンとのマッチングによって、ロボットの現在位置は一応得られている。しかし、本来の位置からは大きくずれているので、ループ検出によって大幅に修正される。つまり、初期値と最適解が離れているということである。ガウス・ニュートン法(Gauss-Newton法)やレーベンバーグ・マーカート法(Levenberg-Marquardt)、山登り法(Hill-Climbing)のような逐次的なマッチング手法では、初期値が最適解にある程度近いことが要求される。言い換えると、ロボットの現在位置が真値とかなり近く、誤差が少ない状態である(地図とスキャンとが既にある程度重なり合っている)ことが求められるが、ループ検出での前提とは異なる。したがって、逐次的なマッチング手法は利用できず、初期値に依存しない頑健な手法が求められる。分岐限定法によるスキャンマッチングは、効率が良く、しかも頑健な手法であるため、ループ検出に利用できる。

3.2.7.2 分岐限定法によるスキャンマッチングアルゴリズム

最初に、アルゴリズムの入出力について述べる。入力とパラメータは次の通りである。

1. 探索方向のサイズ: (W_x, W_y, W_θ) 、 (W_x, W_y) は、単位 [m]、 W_θ は [rad]
2. θ 方向のステップサイズ: δ_θ [rad]
3. ノードの高さ: h_0
4. スキャンデータ: $\mathcal{S} = \{z_1, \dots, z_N\} = (r_1, \theta_1), \dots, (r_N, \theta_N)$
5. 占有格子地図: \mathcal{M} : 解像度 r
6. 探索領域の中心に対応する姿勢: $\xi_0 = [\xi_{0,x}, \xi_{0,y}, \xi_{0,\theta}]^T$
出力は次の通りである。
7. 最大のスコア: s
8. スコアを最大化する姿勢: ξ^*
アルゴリズムは以下のようにまとめられる。
9. 探索領域サイズ(半径) w_x, w_y, w_θ を計算し、探索領域 $\overline{\mathcal{W}}$ と \mathcal{W} を定める。
 $w_x = \lceil \frac{W_x}{r} \rceil, w_y = \lceil \frac{W_y}{r} \rceil, w_\theta = \lceil \frac{W_\theta}{\delta_\theta} \rceil$
 $\overline{\mathcal{W}} = \{0, \dots, 2w_x\} \times \{0, \dots, 2w_y\} \times \{0, \dots, 2w_\theta\}$
 $\mathcal{W} = \{\xi_0 + [r(-w_x + j_y), r(-w_y + j_y), \delta_\theta(-w_\theta + j_\theta)]^T | (j_x, j_y, j_\theta) \in \overline{\mathcal{W}}\}$
10. 与えられた地図 \mathcal{M} を基に、 h_0 個の地図 $\mathcal{M}_{\text{precomp}}^1, \dots, \mathcal{M}_{\text{precomp}}^{h_0}$ を計算する。

$$\mathcal{M}_{\text{precomp}}^1(i, j) = \max_{\substack{i \leq i' < i + 2^h \\ j \leq j' < j + 2^h}} \mathcal{M}(i', j')$$

11. 空のスタック(あるいは優先度付きキュー) \mathcal{C} を \mathcal{C}_0 で初期化する。 \mathcal{C}_0 はノードの集合であり、次のように定義される。

$$\overline{\mathcal{W}}_{0,x} = \{2^{h_0} j_x | j_x \in \mathbb{Z}, 0 \leq 2^{h_0} j_x \leq 2w_x\}$$

$$\overline{\mathcal{W}}_{0,y} = \{2^{h_0} j_y | j_y \in \mathbb{Z}, 0 \leq 2^{h_0} j_y \leq 2w_y\}$$

$$\overline{\mathcal{W}}_{0,\theta} = \{j_\theta | j_\theta \in \mathbb{Z}, 0 \leq j_\theta \leq 2w_\theta\}$$

$$\mathcal{C}_0 = \overline{\mathcal{W}}_{0,x} \times \overline{\mathcal{W}}_{0,y} \times \overline{\mathcal{W}}_{0,\theta} \times \{h_0\}$$

各ノード $c = (2^{h_0} j_x, 2^{h_0} j_y, j_\theta, h_0) \in \mathcal{C}_0$ について、 $\mathcal{M}_{\text{precomp}}^{h_0}$ を使って、スコアの上界 $\bar{s}(c)$ が最も高いノードが、 \mathcal{C} の先頭に来るよう、 \mathcal{C}_0 に含まれる全ノードを \mathcal{C} に追加していく(上界も一緒に追加)。各 j_θ について、 $I_{i,x}^0, I_{i,y}^0$ は一度だけ計算すれば良い。

$$\bar{s}(c) = \sum_{i=1}^N \mathcal{M}_{\text{precomp}}^{h_0}(I_{i,x}^0 + 2^{h_0} j_x, I_{i,y}^0 + 2^{h_0} j_y)$$

$$I_{i,x}^0 = \left\lfloor \frac{\xi_{0,x} + r_i \cos(\xi_{0,\theta} + \delta_\theta(-w_\theta + j_\theta) + \theta_i)}{r} \right\rfloor - w_x$$

$$I_{i,y}^0 = \left\lfloor \frac{\xi_{0,y} + r_i \sin(\xi_{0,\theta} + \delta_\theta(-w_\theta + j_\theta) + \theta_i)}{r} \right\rfloor - w_y$$

12. 最大のスコア s^* を $-\infty$ で、また最適解 $(j_x^*, j_y^*, j_\theta^*) \in \overline{\mathcal{M}}$ を $(0,0,0)$ で初期化する。

13. キュー \mathcal{C} が空になるまで、以下を繰り返す。

- (a) \mathcal{C} の先頭から、ノード $c = (c_x, c_y, c_\theta, h)$ と上界 $\bar{s}(c)$ を取り出す。
- (b) 枝刈り: 上界 $\bar{s}(c)$ が、現在の最大値 s^* 以下であれば、ノード c と、その子ノードの探索は不要であるから、
上に戻って次のノードを試す。
- (c) ノード c の高さが $h = 0$ 、言い換えると葉ノードである場合は、現在の最大スコア s^* を $\bar{s}(c)$ 、また最適解 $(j_x^*, j_y^*, j_\theta^*)$ を (c_x, c_y, c_θ) で更新し、上に戻る。
- (d) 分岐: ノードが葉ノードでなければ、4つの子ノード c_1, c_2, c_3, c_4 に分割する。

$$c_1 = (c_x, c_y, c_\theta, h - 1)$$

$$c_2 = (c_x + 2^{h-1}, c_y, c_\theta, h - 1)$$

$$c_3 = (c_x, c_y + 2^{h-1}, c_\theta, h - 1)$$

$$c_4 = (c_x + 2^{h-1}, c_y + 2^{h-1}, c_\theta, h - 1)$$

14. 各ノード c_i について上界 $\bar{s}(c_i)$ を計算する。以下は c_3 の場合の計算式である。

$$\bar{s}(c_3) = \sum_{i=1}^N \mathcal{M}_{precomp}^{h-1}(I_{i,x}^0 + c_x + 2^{h-1}, I_{i,y}^0 + c_y + 2^{h-1})$$

$$I_{i,x}^0 = \left\lfloor \frac{\xi_{0,x} + r_i \cos(\xi_{0,\theta} + \delta_\theta(-w_\theta + c_\theta) + \theta_i)}{r} \right\rfloor - w_x$$

$$I_{i,y}^0 = \left\lfloor \frac{\xi_{0,y} + r_i \sin(\xi_{0,\theta} + \delta_\theta(-w_\theta + c_\theta) + \theta_i)}{r} \right\rfloor - w_y$$

上界の最も大きな子ノードが先頭に来るよう、4つの子ノードを \mathcal{C} に追加していく（上界も一緒に追加）。

15. 上記の手続きによって最適解 $(j_x^*, j_y^*, j_\theta^*) \in \overline{W}$ が得られたので、最適な姿勢 ξ^* を次のように計算する。

$$\xi^* = [\xi_{0,x} + r(-w_x + j_x^*), \xi_{0,y} + r(-w_y + j_y^*), \xi_{0,\theta} + \xi_\theta(-w_\theta + j_\theta^*)]^T$$

16. 最大のスコア s^* と、最適な姿勢 ξ^* を返す。

3.2.8 SLAMによる環境地図の評価

ここでは、LiDARを用いたSLAMによって生成された環境地図の評価に関する研究について述べる。

3.2.8.1 Hector SLAMおよびGmappingの性能分析

文献[?]によると、モバイルロボットのナビゲーションを目的としたHector SLAMおよびGMappingアルゴリズムの性能分析が行われている。研究結果によると、Hector SLAMアルゴリズムの性能がGMappingよりも優れていることが示されている。ここでは、2Dレーザスキャンデータと、Hector SLAMとGMappingアルゴリズムを使用して、2次元の占有格子地図を生成した。生成されたマップの精度を評価するために、マップ長をRVizの測定ツールで測定を行っている。測定の結果Hector SLAMアルゴリズムは、オドメトリを使用せずに地図を生成し、平均誤差が7.86 cmであった。一方、GMmappingアルゴリズムはオドメトリデータとスキャンデータを組み合わせて使用し、平均誤差が10.86 cmであることが示されている。したがって、Hector SLAMアルゴリズムは、精度の面でGMmappingアルゴリズムよりも優れていると結論付けられている。文献での評価に用いられたハードウェア構成を以下に示す。

1. Raspberry Pi: スキャンデータの処理、地図作成、通信、Arduinoとの通信(速度データ)
2. Arduino:Raspberry Piとの通信、DCモータの制御、ロータリーエンコーダによるオドメトリの取得
3. LiDAR:RPLiDAR-A1

ここで、この文献上では、環境地図生成のためのオドメトリ情報としてロータリーエンコーダを用いており、LiDARのみでのSLAMの性能の比較は行われていない。Hector SLAMアルゴリズムは、環境地図生成のためのオドメトリが必要でないアルゴリズムである。一方Gmappingは、オドメトリが必要である。しかし、比較を行うならば、GmappingをLiDARのみで動作させるべきである。また、これらのSLAMのパッケージでは、パラメータの値により処理のパフォーマンスが変わる可能性があるため、それぞれのSLAMのパッケージに応じたパラメータの与え方を調整する必要がある。

3.2.8.2 Grand truthデータとLiDARを用いた2D SLAMアルゴリズムの地図比較

文献[?]によると、ROSのGmapping、Cartographer、Hector SLAMの3つのSLAMアルゴリズムによって、作成された地図と、レーザートラッカーによって示されたGrand truth(AIモデルの出力の教師あり学習で使われる正解データ)の比較が行われた。地図は、静的な屋内環境で行われ、Grand truthモデルは非常に精度が高いものが使用されている。地図空間上の点座標をマイクロメートル精度で計測する高精度レーザートラッカーFAROによって構築されたGrand truthを用いて、SLAM地図の品質を評価する手法が提案された。実験は、移動ロボットの動作について、以下の異なる条件で行われた。(1)滑らかなターンで素早く、(2)急旋回を伴う高速移動、(3)スムーズな旋回を伴う低速移動。(4)スムーズなターンでゆっくり、ただしループは閉じない。結果として、おおよそほとんどの条件でCartographerが、レーザートラッカーが提示するGrand truthに対して、最小の誤差で地図を構築することが明らかになった。このアルゴリズムは、様々なタイプの移動ロボットの動きに対して、十分に強固であるとした。Gmappingの地図は、Cartographerの地図の品質と同等であった。その理由はGmappingは、2D地図構築において、ループ閉じ込みがなくても、自己位置の地図の補正にオドメトリを使用しているからである。これは、Cartographerについても同様である。Hector SLAMの最先端であるCartographerではLiDARデータのみを使用し、ループ閉じ込みを行わない。そのため、Hector SLAMの結果は精度が悪いとしている。このようにCartographerが、LiDARを移動ロボットに搭載し、2Dの地図を生成するのに最適なアルゴリズムであると結論づけた。文献[?]の実験で使用された差動駆動移動ロボットプラットフォームについて示す。

1. オンボードコンピュータ Jetson TX1
2. バンパー
3. ソナー
4. Troyka IMU モジュール
5. Hokuyo URG-04LX-UG01 二次元 LiDAR
6. Velodyne WLP-16 三次元 LiDAR
7. 車輪のオドメトリを計算するためのロータリーエンコーダ
8. ROS Kinetic Ubuntu 16.04

ここでも文献上では、ロータリーエンコーダやIMU等の内界センサによるオドメトリによって自己位置と地図の補正を行っていて、LiDARのみでのSLAMの地図精度の評価は行われていない。また、先に述べた文献と比較すると、Hector SLAMよりも、Gmappingのアルゴリズムの方が優れているとした、異なる結果が得られている。これらの違いは、RVizによる手動測定ではなく、レーザートラッカーによるGrand truthデータを用いている点、地図生成GPUを搭載した高性能なコンピュータを用いている点である。前述のように、SLAMアルゴリズムや、各アルゴリズムで指定するパラメータの値によって、計

算量が異なるため、搭載するロボットの開発上の制約により、コンピュータの性能が限られる場合等によって、コンピュータの性能差が地図精度に与える影響等が予想されるが、今回の研究では、コンピュータはロボットの開発上の制約で決められたものを使用する。

3.3 提案

ROS で実装されている各 SLAM パッケージを用いた LiDAR のみによる環境地図作成システムを提案する。LiDAR の SLAM において、LiDAR のレーザオドメトリによるオドメトリ有りの手法とオドメトリフリーの手法の地図を比較することにより、内界センサを必要としない SLAM の環境地図の妥当性を検証することを目指す。

3.3.1 実装

本研究で使用した環境地図作成システムのプロトタイプの設計について、ソフトウェアとハードウェアに関して述べる。ソフトウェアでは、LiDAR のスキャンを、ROS を介して、レーザオドメトリを用いて地図を作成する方法と、各 SLAM アルゴリズムに対応した ROS の座標変換パッケージ tf による座標変換について述べる。その後、ROS の可視化ツール RViz アプリケーション上でプロトタイプが生成するマップシミュレーションのプロセスについて述べる。

3.3.1.1 ハードウェア

ハードウェア図と配線について説明し、使用される各入出力に基づいて分割する。環境地図作成システムの概要を以下に示す。図 3.20 のように、構成するハードウェアは入力、通信、演算・出力のブロックに分かれている。LiDAR は、このシステムの入力として機能する。LiDAR は、ロボットの周囲の状況をスキャンし、Raspberry Pi で処理される。

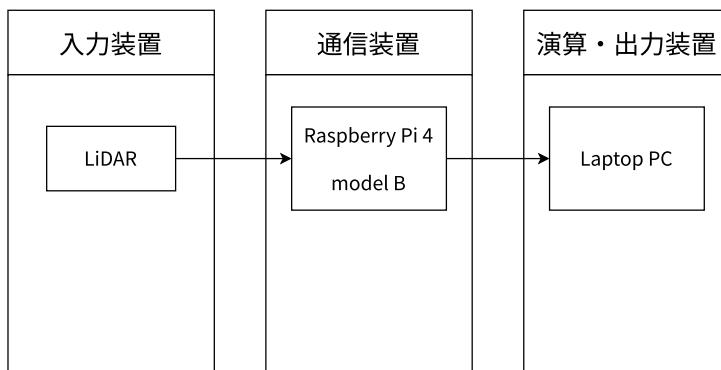


図 3.20: 環境地図作成システムの概要

通信装置では、Raspberry Pi が ROS システム上のマスターノードとして機能してスレーブノードである Laptop PC と通信を行う。演算・出力装置では、Laptop PC がマッピ

ングの処理を行う。ROS のシステムをマスターとスレーブに分割することで、分散処理により、マッピング等の演算機能を Raspberry Pi よりも処理速度が速いノート PC で行うことができる。また、通信装置に使用している Raspberry Pi は、将来的に自律移動ロボットに組み込むことを想定し、アクチュエータ制御を担う Arduino との通信を行うことも目的としており、ROS の通信機能によって自律移動の情報を Laptop と Arduino に送受信する。Raspberry Pi に接続した LiDAR の点群取得の実行ファイルを以下に示す。LiDAR は、北陽電機社製の URG-04LX-UG01 を用いた。ROS 上で北陽電機社製の LiDAR を扱うためのドライバとして urg_node[?] を用いる。LiDAR による点群取得の実行ファイルをソースコード 3.1 に示す。

ソースコード 3.1: LiDAR による点群取得の実行ファイル

```

1 <launch>
2   <node pkg="urg_node" name="urg_node" type="urg_node" output="screen" >
3     <param name="serial_port" value="/dev/ttyACM0"/>
4     <param name="serial_baud" value="115200"/>
5     <param name="frame_id" value="base_link"/>
6     <param name="calibrate_time" value="true"/>
7     <param name="publish_intensity" value="false"/>
8     <param name="publish_multiecho" value="false"/>
9     <param name="angle_min" value="-1.5707963"/>
10    <param name="angle_max" value="1.5707963"/>
11  </node>
12 </launch>

```

3.3.1.2 ソフトウェア

LiDAR のスキャンによって得られるランドマークの位置情報を、自己位置を示すロボット座標系と、地図の座標系に変換する。座標変換は ROS の tf[?] パッケージを用いて行い、それぞれの SLAM アルゴリズムの実装に合わせて必要な座標系を取得するように実装する。

次にそれぞれの SLAM アルゴリズムにおいて、地図作成に使用した座標系の取り扱いの違いについて説明する。

3.3.1.3 Gmapping, Cartographer

Gmapping と Cartographer に関しては、LiDAR のスキャンをロボット座標系に変換し、laser_scan_matcher[?] パッケージを用いてオドメトリとマッチングさせることで、地図を作成する。それぞれの座標変換の概要 tf ツリーを図 3.21、3.22 に示す。

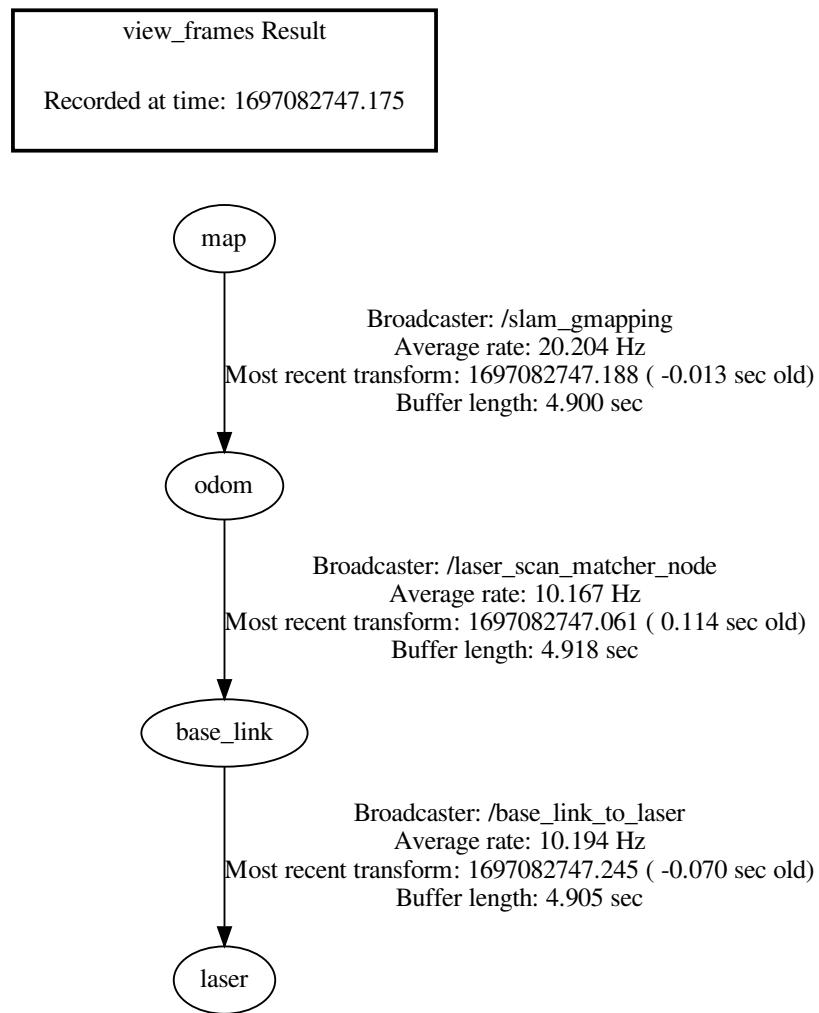


図 3.21: Gmapping の座標変換 tf ツリー

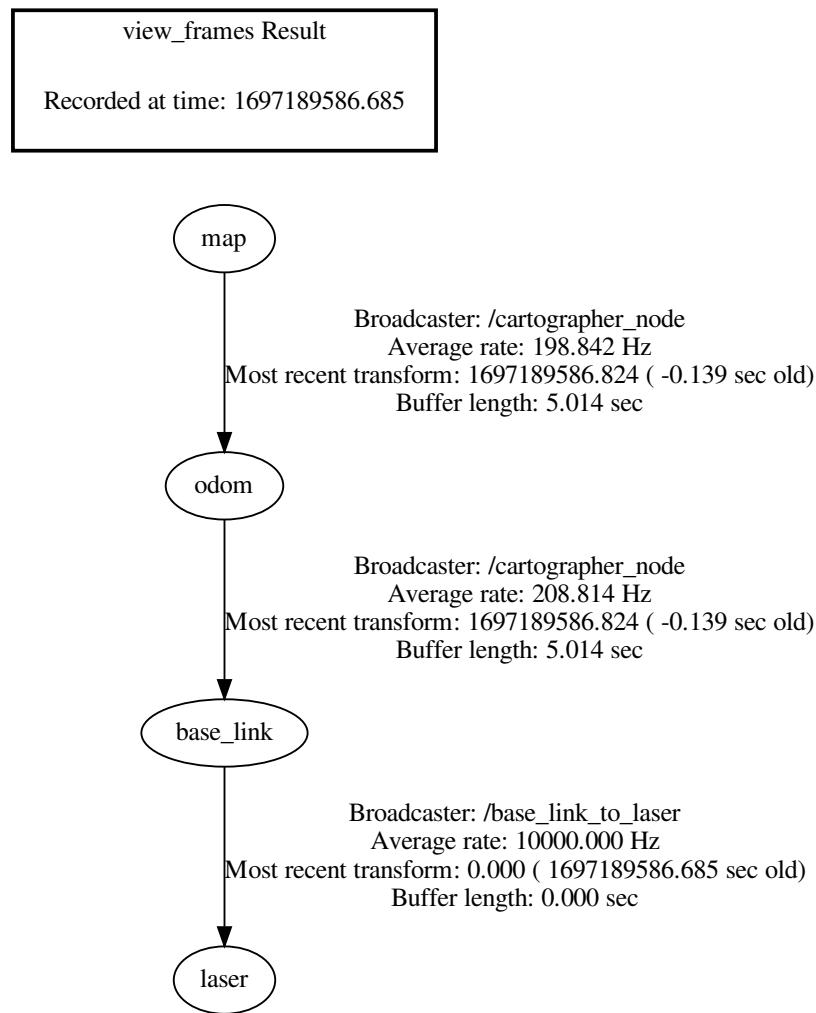


図 3.22: Cartographer の座標変換 tf ツリー

3.3.1.4 laser scan matcher

`laser_scan_matcher` は、2つの点群に関して、その位置関係を推定して合成することを行う ROS のパッケージであり、レーザスキャンの移動の増分を取得する機能を有する。このパッケージでは、`sensor_msgs/LaserScan` メッセージ間のスキャンマッチングを可能にし、レーザの推定位置を `geometry_msgs/Pose2D` または `tf` のメッセージとして取得する。このパッケージは、LiDAR 以外のセンサによって得られるオドメトリ無しで使用することができる。また、複数の種類のオドメトリを追加することで、LiDAR 自身のレーザスキャンの取得速度と精度を向上させることができる。

3.3.1.5 laser scan matcher の原理

`laser_scan_matcher` は、点群のスキャンマッチングとして有名な ICP アルゴリズムの派生版である PLICP(Point to Line ICP) が実装されている。

まず、ICP アルゴリズム (Iterative Closest Point) について述べる。ICP アルゴリズムでは、2つの点群が整合するように、その位置姿勢の関係を調整する方法の1つである。ICP では、図 3.23 に示す。ようやく繰り返し計算に基づいて、段階的に位置姿勢を調整する。

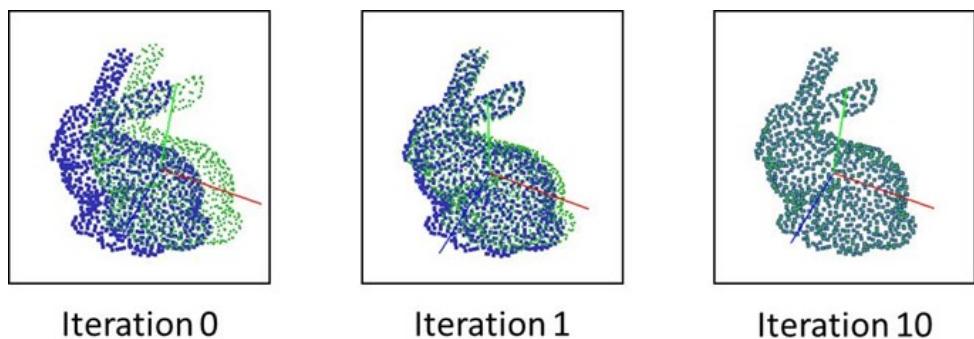


図 3.23: ICP アルゴリズムの繰り返しの様子

次に ICP の処理の手順を説明する。ICP のアルゴリズムの繰り返しの様子を図 3.24 に示す。

1. 片側の点群 A の各点から、もう片方の点群 B で最近傍の点を探索し、対応づける。
2. 対応づけた点の差を最小化するように、点群の座標系の位置姿勢を調整する。

処理 1 の時点で 2つの点群のずれが大きいと、対応付けは間違いを含みやすくなる。この場合、処理 2 の調整結果はずれが残る場合がある。そこで、繰り返し計算による精度向上を考える。まず処理 2 の後、2つの点群のずれは初期の状態より小さくなっていると仮定する。その状態で再度処理 1 を行えば、初期の状態より正確に対応付けができることが期待できる。そして、対応付けが正確にできれば処理 2 の位置姿勢も正確に調整できる。つまり、繰り返し計算を行うことで、徐々に精度を向上させることができる。

ICP 方式には次のような欠点がある。

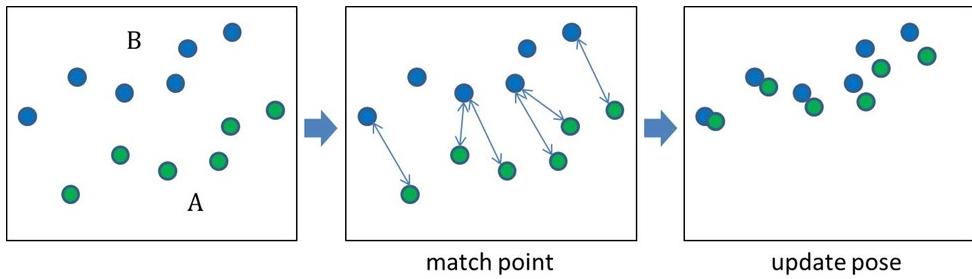


図 3.24: ICP アルゴリズムの繰り返しの様子

1. 初期値に依存し、初期値が悪いと反復回数が増加し、初期誤差が大きい場合は、反復結果を誤ることがある。
2. ICP は一次収束であり、収束速度が遅い。(探索効率を上げるために kd-tree を使うことになる。)
3. 異常値やノイズが発生する。

PLICP は、ICP アルゴリズムを改良し、標準の点と点のマッチングではなく、点と線のマッチングを行う。PLICP を図 3.25 に示す。PLICP の処理は、ICP と同じである。違いは、ICP が最近傍点を見つけて点間の距離を誤差として使用するのに対し、PLICP は 2 つの最近傍点を見つけることである。2 つの点を接続する場合、点から線までの距離が誤差として使用されるため、PLICP のマッチングエラーは ICP のマッチングエラーよりもはるかに小さくなる。

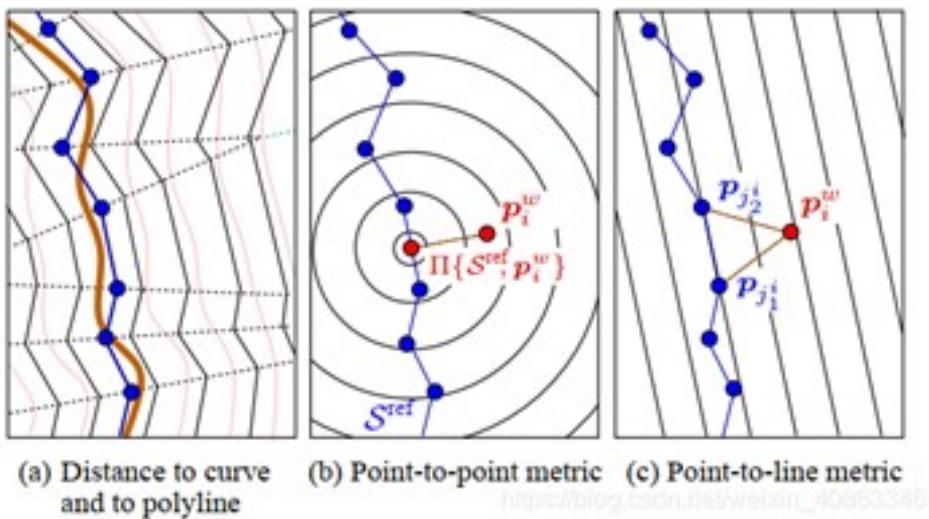


図 3.25: PLICP

図 3.26 に PLICP による P1 と P2 の位置関係を示す。青い点は $t - 1$ 時間のレーザ点、茶色の線は t(壁など)、赤い点は時点実際のオブジェクト図中の同心円の実践は点間の距

離誤差を表す。点 P1 を知るある点からの距離 D によって別の点 P2 の位置が決まる。P2 の位置は、P1 を中心とし、D を半径とする円上にある。

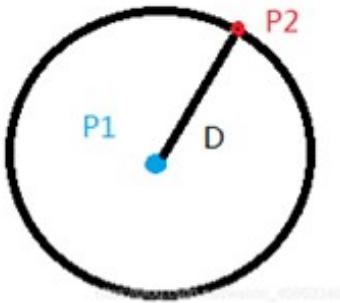


図 3.26: PLICP による P1 と P2 の位置関係

文献 [11] によると、PLICP が ICP、IDC、MBICP 等のアルゴリズムよりも精度が高く、反復回数が少なくて済むことが示されている。また、アルゴリズムの最適化技術により、PLICP は高速に動作することが述べられている。

3.3.1.6 Hector SLAM

図 3.27 に Hector SLAM の座標変換の tf ツリーを示す。Hector SLAM に関しては、LiDAR のスキャンをロボットの座標系に変換した後に、ロボット座標系の移動履歴の座標系を tf のみでオドメトリを生成し、ロボット座標系に変換する前の LiDAR のスキャンとマッチングさせることで、地図を作成する。

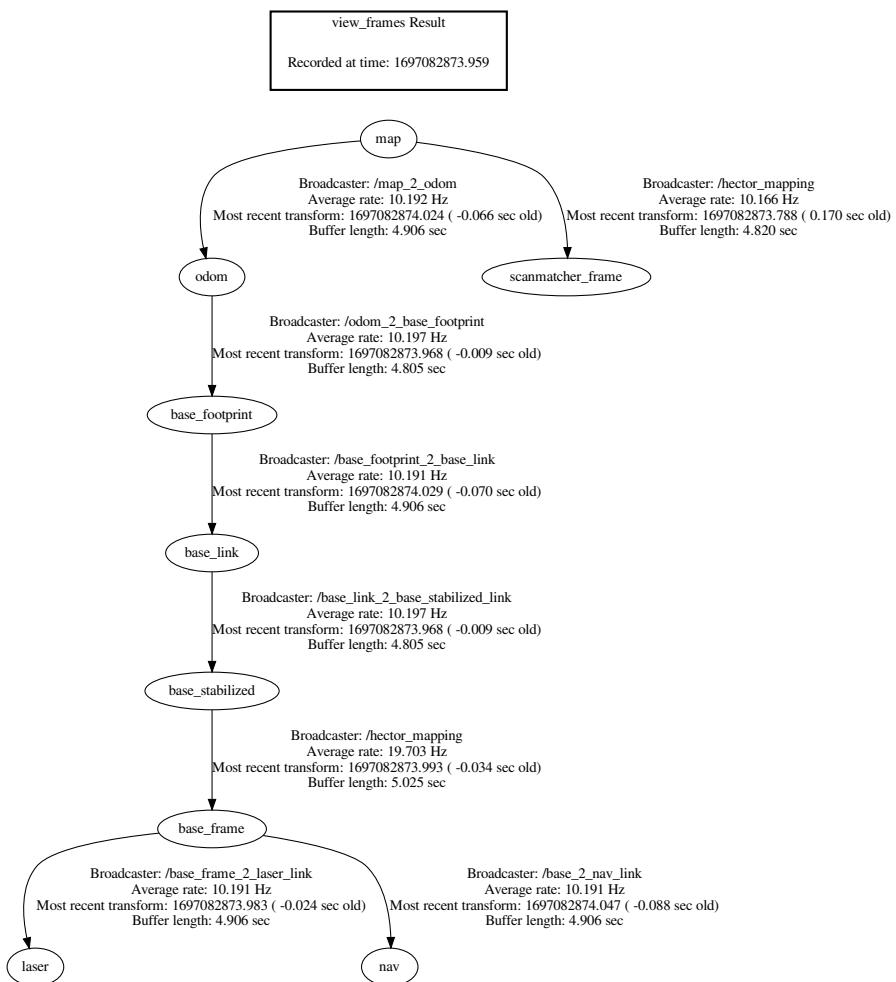


図 3.27: Hector SLAM の座標変換 tf ツリー

3.3.1.7 パラメータの調整

各SLAMパッケージのパラメータの値を示す。

ソースコード 3.2: Gmapping のレーザオドメトリのパラメータ

```

1 <?xml version="1.0"?>
2 <launch>
3   <node pkg="tf" type="static_transform_publisher" name="base_link_to_laser"
      args="0.0 0 0 0.0 0.0 0.0 /base_link /laser 100" />
4 //スキャンLiDAR(laser)をロボット座標(base_link)に変換
5   <node pkg="laser_scan_matcher" type="laser_scan_matcher_node" name="laser_scan_matcher_node">
6     <param name="fixed_frame" value = "/odom地図座標"/>//
7     <param name="base_frame" value = "/base_linkロボット座標"/>//
8     <param name="use_cloud_input" value="false"/>
9     <param name="publish_tf" value="true"/>
10    <param name="publish_odom" value="true"/>
11    <param name="use_odom" value="false"/>
12    <param name="use_imu" value="false"/>
13    <param name="use_alpha_beta" value="true"/>
14    <param name="max_iterations" value="10"/>
15  </node>
16 </launch>
```

ソースコード 3.3: Gmapping の座標系の設定

```

1 <launch>
2   <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" clear_params
      ="true">
3     <rosparam command="load" file="$(find urg_gmapping)/config/gmapping.yaml"
            />
4     <!-- <remap from="scan" to="front_laser/scan" /> -->
5     <param name="base_frame" value="/base_link" />
6     <param name="odom_frame" value="/odom" />
7     <param name="map_frame" value="/map" />
8   </node>
9   <node pkg="rviz" type="rviz" name="rviz" args="-d $(find urg_gmapping)/
      rviz_cfg/rviz_cfg.rviz"/>
10 </launch>
```

ソースコード 3.5: Cartographer の座標系とレーザオドメトリのパラメータの設定

```

1 include "map_builder.lua"
2 include "trajectory_builder.lua"
3
4 options = {
5     map_builder = MAP_BUILDER
6     trajectory_builder = TRAJECTORY_BUILDER
7     map_frame = "map"
8     tracking_frame = "base_link"
9     published_frame = "base_link"
10    odom_frame = "odom"
11    provide_odom_frame = true
12    publish_frame_projected_to_2d = true
13    use_odometry = false
14    use_nav_sat = false
15    use_landmarks = false
16    num_laser_scans = 1
17    num_multi_echo_laser_scans = 0
18    num_subdivisions_per_laser_scan = 1
19    num_point_clouds = 0
20    lookup_transform_timeout_sec = 0.2
21    submap_publish_period_sec = 0.3
22    pose_publish_period_sec = 5e-3
23    trajectory_publish_period_sec = 30e-3
24    rangefinder_sampling_ratio = 1.
25    odometry_sampling_ratio = 1.
26    fixed_frame_pose_sampling_ratio = 1.
27    imu_sampling_ratio = 1.
28    landmarks_sampling_ratio = 1.
29 }
30
31 MAP_BUILDER.use_trajectory_builder_2d = true
32
33 TRAJECTORY_BUILDER_2D.min_range = 0.5
34 TRAJECTORY_BUILDER_2D.max_range = 8.
35 TRAJECTORY_BUILDER_2D.missing_data_ray_length = 8.5
36 TRAJECTORY_BUILDER_2D.use_imu_data = false
37 TRAJECTORY_BUILDER_2D.use_online_correlative_scan_matching = true
38 TRAJECTORY_BUILDER_2D.real_time_correlative_scan_matcher.linear_search_window
= 0.1
39 TRAJECTORY_BUILDER_2D.real_time_correlative_scan_matcher.
    translation_delta_cost_weight = 10.
40 TRAJECTORY_BUILDER_2D.real_time_correlative_scan_matcher.
    rotation_delta_cost_weight = 1e-1
41 TRAJECTORY_BUILDER_2D.motion_filter.max_angle_radians = math.rad(0.2)
42 -- for current lidar only 1 is good value
43 TRAJECTORY_BUILDER_2D.num_accumulated_range_data = 1
44
45 POSE_GRAPH.constraint_builder.min_score = 0.65
46 POSE_GRAPH.constraint_builder.global_localization_min_score = 0.65
47 POSE_GRAPH.optimization_problem.huber_scale = 1e2
48 POSE_GRAPH.optimize_every_n_nodes = 35
49
50 return options

```

3.3.1.8 RViz による環境地図の可視化

RViz アプリケーション上でロボットプロトタイプが生成する環境地図の可視化のプロセスについて述べる。SLAM のプログラムによって作成された環境地図は、ROS ツールである map_server によって、pgm と yaml の形式で保存される。Rviz アプリケーションでこれらの形式を読み込むことができる。Rviz アプリケーションでは、読み込んだ地図の測定ツールがある。この機能は 2 点間の距離を計算することができる機能である。計算したい 2 つの値を取得することで、測定結果をメートル単位で表示する。図 3.28 に LiDAR のスキャン結果を示す。図 3.29 に環境地図の読み込み画面を示す。urg_node で LiDAR の scan トピックを生成し、SLAM プログラムがスキャンデータの変換に tf パッケージを使用して、自己位置と地図の演算を行う。Rviz アプリケーションはマッピングの結果をリアルタイムで表示する。

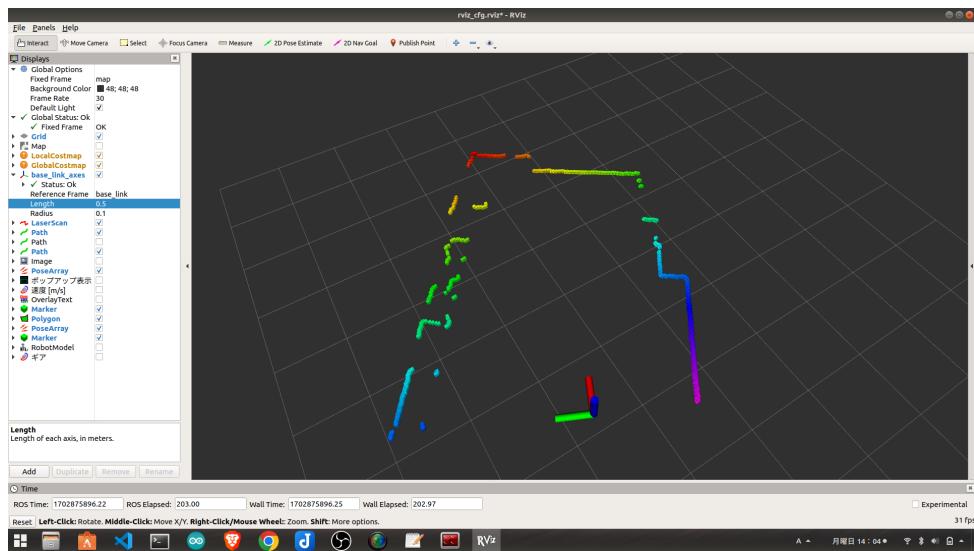


図 3.28: LiDAR のスキャン結果

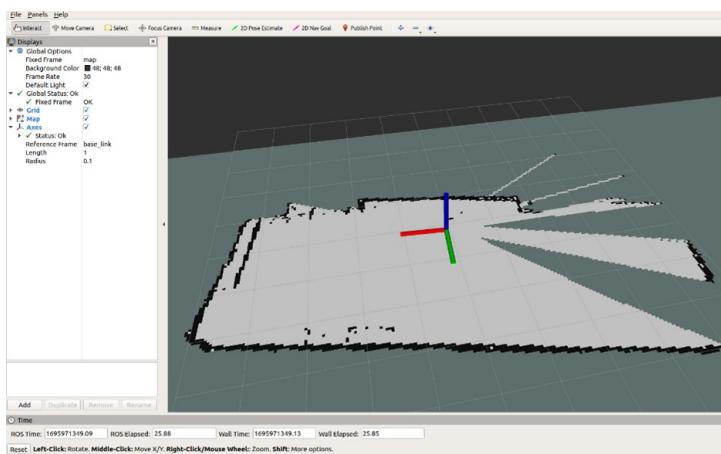


図 3.29: 環境地図の読み込み

3.4 検証

3.4.1 実験

本研究で提案した環境地図作成システムを用いて、Gmapping、Hector SLAM、Cartographer の SLAM アルゴリズムパッケージを用いて、屋内の一室を用いた環境地図の作成実験を実施した。今回の屋内環境は一つの部屋のみであり、仕切りなどは存在せず、部屋の基礎部分となる支柱が特徴点として検出しやすいため、SLAM の環境に適している。部屋の詳細は、図 3.30 の平面図に示している。平面図には、部屋の設計寸法から値を使用している。図 3.31 は、環境地図中の辺に番号を付けたものである。本実験では、Gmapping、Hector SLAM、Cartographer のアルゴリズムによって生成された環境地図の寸法と実際の寸法との対応の誤差を評価する。方法として Rviz 測定ツールによる手動測定値と実際の寸法の値を比較する。人が部屋の中で LiDAR からなるシステムを持って歩くことで地図を作成する。この際、LiDAR の方向を変える際は、ゆっくり行い、人が歩く動作をなるべく慎重に行った。

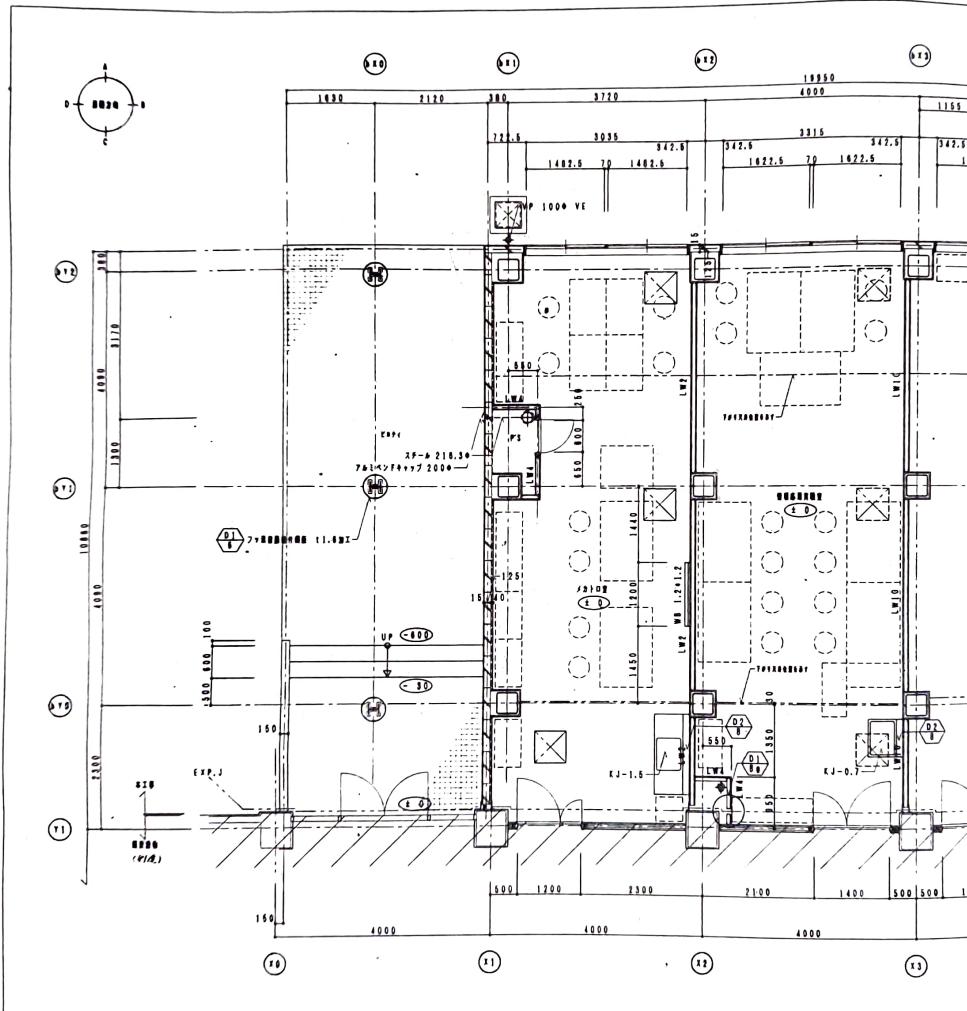


図 3.30: 実際の部屋の寸法図

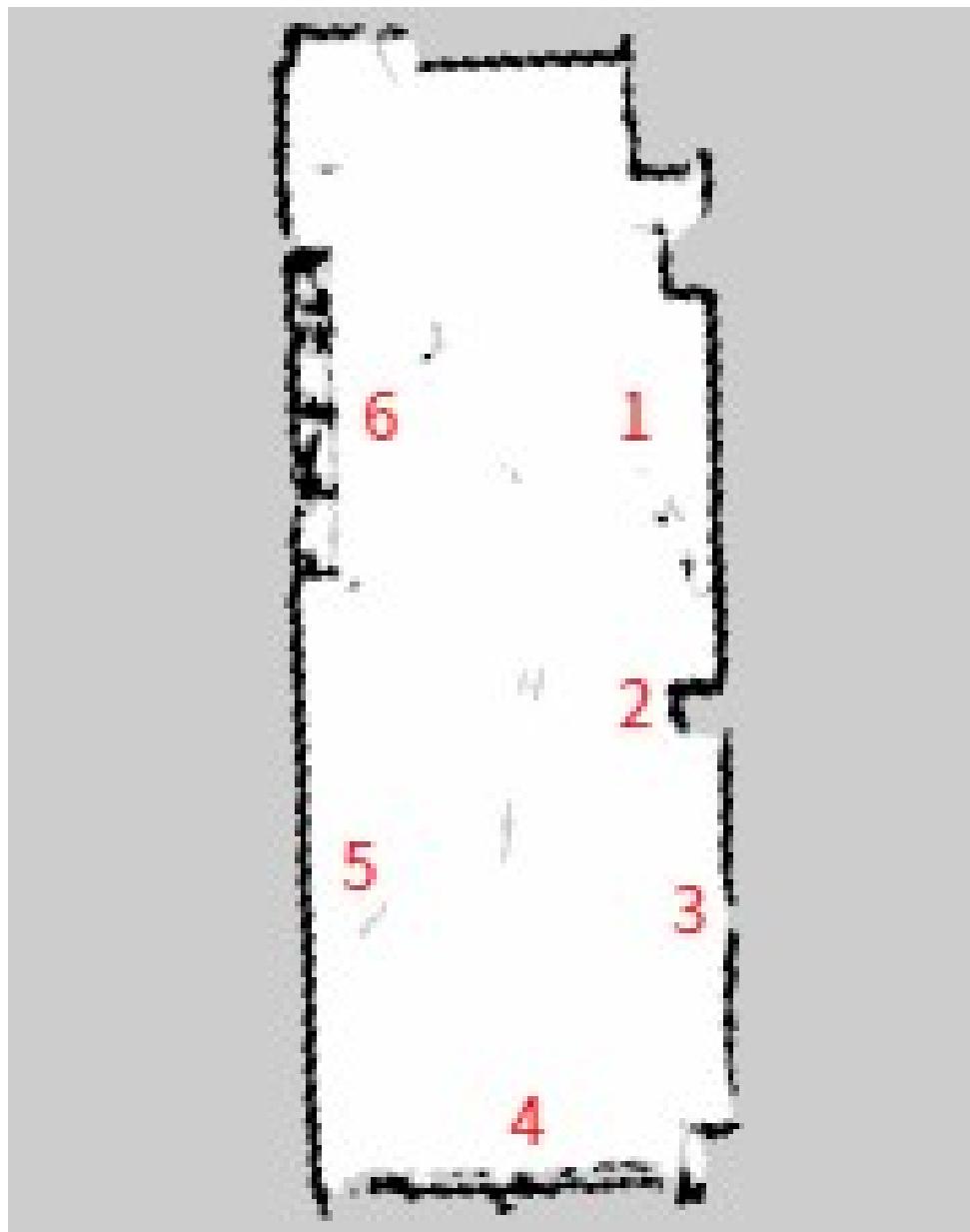


図 3.31: 寸法の測定箇所

3.4.2 結果

Gmapping、Hector SLAM、Cartographer とともに RViz の画面上に環境地図が描画された。図 3.32～3.34 に各アルゴリズムによって生成された環境地図を示す。内界センサによるオドメトリを用いず、LiDAR のみで SLAM を行っているが、部屋の特徴を的確にとらえており、実際の部屋と同様に支柱の構造と収納棚が確認され、非対称な形状が環境地図に反映されていることが分かる。平面地図の計測の結果を表 3.2 に示す。

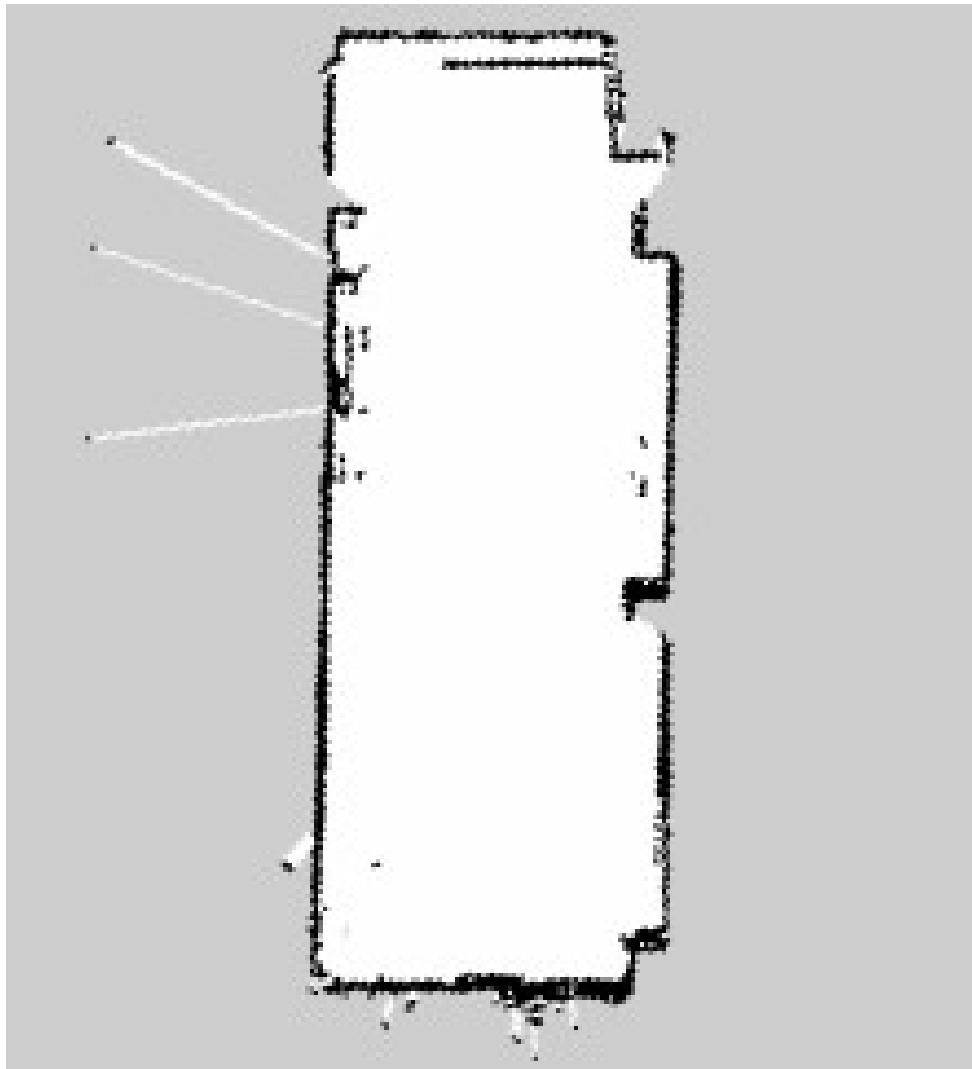


図 3.32: Gmapping

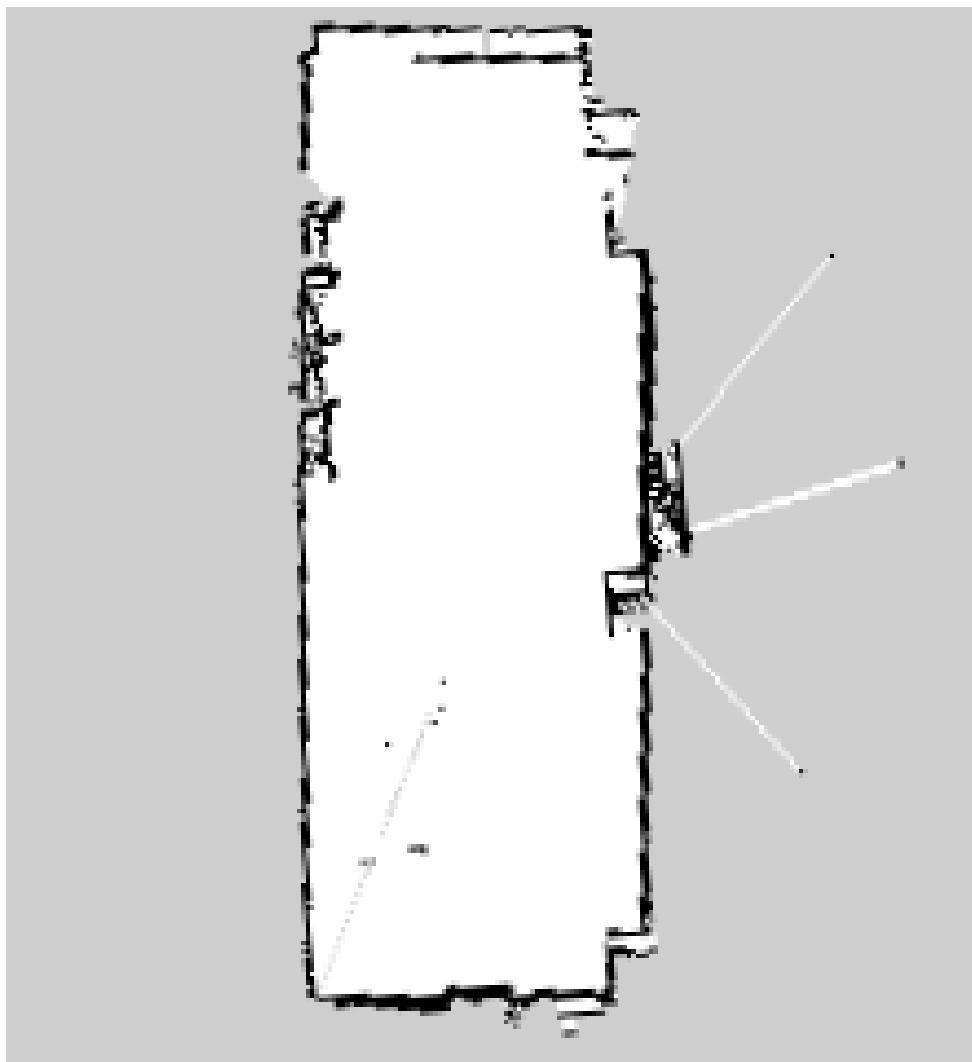


図 3.33: HectorSLAM

表 3.2: RViz の測定ツールによる測定結果

| No | 測定値 (cm) | | | 実寸 | 誤差 (cm) | | |
|--------|----------|------------|--------------|--------|----------|------------|--------------|
| | Gmapping | HectorSLAM | Cartographer | | Gmapping | HectorSLAM | Cartographer |
| 1 | 361.531 | 356.567 | 359.158 | 356.5 | 5.031 | 0.067 | 2.658 |
| 2 | 47.8064 | 59.3467 | 51.7346 | 52.5 | -4.6936 | 6.8467 | -0.7654 |
| 3 | 362.027 | 333.141 | 359.743 | 356.5 | 5.527 | -23.359 | 3.243 |
| 4 | 338.146 | 331.471 | 333.615 | 331.5 | 6.646 | -0.029 | 2.115 |
| 5 | 1067.5 | 1083.04 | 1073.81 | 1074.3 | 5.845 | 4.427 | 3.108 |
| 6 | 309.045 | 307.627 | 306.308 | 303.2 | 5.845 | 4.427 | 3.108 |
| 平均二乗誤差 | | | | | 5.6051 | 19.150 | 0.90262 |

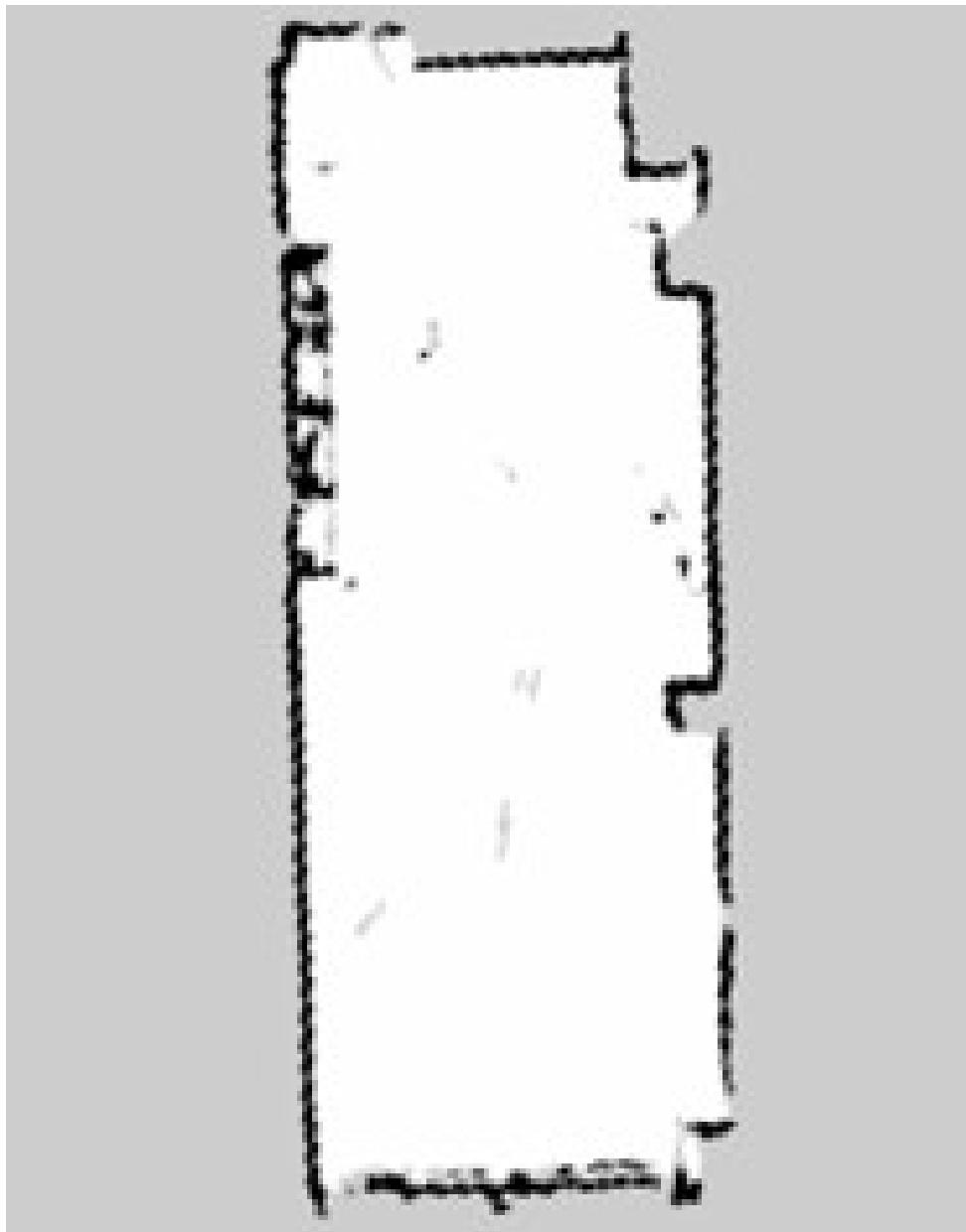


図 3.34: Cartographer

実験番号1、2の実際の寸法が356.5 [cm]、52.50 [cm]であることに対して、Gmappingの測定値が361.531 [cm]、47.8064 [cm]であったためRviz測定ツールによる測定値と実際の寸法との誤差は5.031 [cm]、-4.6936 [cm]となる。実験番号2～7で示された異なる場所で同様に測定したところ、平均二乗誤差は5.605 [cm]であった。このことからlaser_scan_matcherを使用するGmappingでは5.605 [cm]の誤差で部屋の地図を作成できることが分かる。Hector SLAMによって生成された地図の場合、実験番号1、2の測定値が356.567 [cm]、59.3467 [cm]の測定値が得られる。他も同様に359.743 [cm]、333.615 [cm]、1073.81 [cm]、306.308 [cm]の結果が得られた。これらの平均二乗誤差は、19.15 [cm]であり、Hector SLAMによる地図の寸法にはばらつきがあることが分かる。また、Hector SLAMアルゴリズムを使用した場合、実験番号1の上面と、2の横面、下面に点群のずれが生じている。Hector SLAMやCartographerは、地図作成の際のLiDARの方向転換や、前進速度等が少しでも早くなると、地図に大きな歪みが生じた。Hector SLAMは、地図が歪むと修正されない。Cartographerや、Gmappingは、限度があるものの、歪みは修正される。Gmappingは、地図作成の際のLiDARの方向転換や、前進速度等が速くの影響をほとんど受けなかった。Cartographerアルゴリズムによって生成された地図の場合、どの実験番号でも誤差が高々3[cm]程度であり、平均二乗誤差が0.90 [cm]と非常にばらつきが少なく、誤差が少ないことが明らかになった。

3.5 考察

実験の結果より、LiDAR以外の内界センサを用いず、LiDARのみによって地図の生成を行うことが可能であると言える。特に、平均二乗誤差が最も少ない、Cartographerが他のアルゴリズムと比べて、精度の良い地図であると考えられる。次にGmappingの地図の精度が高く、Hector SLAMが最も精度が低い地図となった。しかし、測定の際の動作を丁寧に行う必要があるアルゴリズムと、測定の際の外乱に強いアルゴリズムが存在することが明らかになった。今回の実験では、LiDAR計測方向の転換や、歩きながら地図を作成する動作を慎重に実施した、Hector SLAMは、これらの動作を慎重に行わなかった場合、正確な地図を作成することが不可能であった。これは、LiDARの筐体が移動する速度が速い場合、現在の測定データの端点が、それまでに読み込んで生成した地図のデータ(過去の測定データ)と離れすぎてしまい正しくスキャンマッチングができなくなつたためだと考えられる。また、移動ロボットの現在位置が生成した環境地図から大きくずれてしまい、自己位置を見失う現象が生じた。その後、復帰することがなかつたため、正しいスキャンマッチングができず、別の地形と判断されたためだと考えられる。Cartographerは、平均二乗誤差が最も少ないと、LiDARの筐体を移動させる速度を速くすることにより、地図が歪む現象が起きた、その後、地図は修正されるが、歪みの分と、元の地図との間の位置へと修正されるが、歪みの量が大きいと、復帰が不可能である。Gmappingは、移動速度を速くしても、地図の歪みが比較的小さく、現在のスキャンに歪みが含まれても、元の地図の位置とマッチングするように修正が行われる。これは、他のSLAMアルゴリズムと比べて計算量が少なく、復帰に要する時間が少なかつたためだと考えられる。よって、ロボットに搭載して、移動させながら地図を作成する場合、Gmappingを用いることが、妥当であると考えられる。

3.6 おわりに

LiDAR(URG-04LX-01)のみでの地図生成が可能であることを明らかにした。特に、Hector SLAM よりも Cartographer や Gmapping で生成した地図の精度が 70.7% ~ 95.3% 程良いことが明らかになった。また、LiDAR による計測に要する応答性は、Gmapping が、他のアルゴリズムと比べて優れていることが明らかになった。これらの観点から、比較的、並進速度や角速度が速いロボットの動作において、URG-04LX-01 のみで、SLAM を行う場合は、Gmapping を用いることが適切である。今後は、Gmapping を用いて、4輪自律移動ラジコンカーの開発を行う。

第4章

自動運転システムの開発

4.1 はじめに

自律移動ロボットの開発プロセスにおいて、実環境における実証実験が数多く行われている。特に、屋外における自律移動においては、シミュレーションと実環境では、制約や環境条件が全く異なるため、実環境のみでしか得ることのできない知見が多く、自律移動の実用化に向けては、実機を用いた実証実験が必要とされている。実証実験に代表されるつくばチャレンジでは、自律移動ロボットの設計において、要素技術である LiDAR を中心とした、センサ系の使用が主となり、複数のセンサとの融合（センサフュージョン）による環境認識の高精度化が図られている。それと同時に、自律移動ロボットの実用化に向けて、使用されるセンサの選定、用途に応じて最適化する試みが行われている。特に、ロボットに搭載するセンサの使用数を少なくすることは、ロボットに課される制約・条件、ソフトウェア開発の工数減、保守性の向上のために、考慮すべき点と述べられている。自律移動チャレンジの参加車両の形態は多様であり、差動二輪型が一般的であるが、パーソナルビークル等の使用としては、4輪車両型の自律移動等も見られる。自律移動の要素技術は、自動運転にも使用されているが、つくばチャレンジでは、人と共存するロボットとして比較的低速であることが特徴であり、LiDAR センサによる環境認識の比重が高い。対して、自動運転では、カメラや深度センサ等のイメージセンサ等のコンピュータビジョンと LiDAR センサによる SLAM がそれぞれ同じ割合での併用が標準となり、センサ融合に関する技術について言及されている。自律移動ロボットの要素技術として用いられている LiDAR のメーカーとして有名な北陽電機株式会社は、中之島チャレンジに参加し、自律移動ロボットの実機を開発して、実際にフィールドで議論を行うことで、新製品の開発を行っている。自律移動ロボットの要素技術である LiDAR の開発においても、実証実験が不可欠である。筆者は、中之島ロボットチャレンジを観戦することで、将来的に自律移動ロボットや LiDAR の開発に携わりたいと考えた。今回は、縁あって北陽電機株式会社様より LiDAR を貸与頂き、有効活用して筆者自らの開発アイデアを実現することを目標として研究に取り組むこととした。本研究の位置付けは、LiDAR の使用技術の勉強並びに、北陽電機株式会社様への就職を目的としている。

4.2 従来研究

4.2.1 つくばチャレンジの屋外ナビゲーションシステム

つくばチャレンジ[?]は、自律移動ロボットのナビゲーション技術の発展を目的とした実験走行会である。この期間は実験走行の機会が定期的に用意され、最終日に本走行（最終実証走行）が行われる。課題として、つくば市の屋外環境において約1[km]の自律走行コースが設定される。コースには、遊歩道、公園、広場、車道に面した一般歩道、湾曲路、陸橋、狭いゲート、自動ドアなど多様な環境を含む。ロボットのために環境に手を加えることは許されず、歩行者や自転車などが存在する実環境のままで行われる。ただし、安全確保のため、一部の交差点で一時停止を行うことが規定されている。つくばチャレンジのコースを自律走行するには、目的地と現在地を認識する自己位置推定、歩行者や他のロボットとの衝突を防ぐ障害物検出・回避が高い確実性で求められる。

友納ら[?]は、屋外での自律走行が現状でどのくらい可能かを実環境で見極めることを目的として、本走行における完走を第一とし、新規技術の研究とは異なるレベルの問題分析と知見を重視した取り組みを行っている。文献では、自己位置推定と障害物回避のためのシステム設計方針について、ナビゲーション、障害物対応、センサ、システム規模について述べていた。ナビゲーションにおいては、正確な地図上で自己位置推定に重点をおく方法と、大まかな地図上でオンラインの環境認識を重視する方法の二つを挙げた。後者は、人間が取っている戦略に近く、柔軟性が高い反面、非常に高い認識能力を必要とする。そこで、自己位置推定を基とした方法を取った。障害物回避について、つくばチャレンジでは、一般的市街地で遭遇するほぼすべての移動障害物が存在し、他のロボットや見物客など、つくばチャレンジ特有の障害物も存在する。そこで、本質的な困難さを避けるため、最低限の方策として、障害物回避は静止障害物にだけ行い、移動障害物に対しては一時停止して相手に避けてもらうという方針をとった。センサについては、システムの能力に大きく影響する。主力であるレーザスキャナと、有望であるカメラを両方とも検討した結果、レーザスキャナを用いて確実性の高いシステムを作成した。そのうえでカメラを追加した。GPSについては、つくばチャレンジのコースは樹木や建物の近くを通ることに加え、屋内も走行することを想定すると、安定利用ができないため、使用しなかった。システム規模として、サイズやコストの制約からあまり多くのセンサやPCを搭載できない。また、多くのセンサを搭載すると、ソフトウェア開発や保守の工数も大きくなる。そのため、できるだけ少ないセンサを有効に使うことを考えた。

4.2.2 中之島チャレンジ

中之島ロボットチャレンジ[?]は、人々の往来する実環境において自律移動ロボットが問題無く行動できる技術開発の公開実験を大学や企業向けに提供する大会である。様々な研究開発機関が参加する技術交流の場を設けることで、ロボット開発技術のレベルを向上させることを目的としている。中之島ロボットチャレンジでは、具体的な社会課題の解決に向けての新たなテーマとして、2025年万博でのデモンストレーションを目指した自動ゴミ回収ロボット実現のためのマイルストーンを掲げている。前述のつくばチャレンジの

ルールに準拠し、大阪中之島公園の中央公会堂周辺の歩行者天国エリアにて、定められたコースをロボットに自律走行させる課題と、コースの自律走行を行いながら、人物発見/ゴミ発見/ゴミ回収の課題を行う。

4.2.3 Autonomous Driving of Competition Robot

Cotrim, Sousa ら [?] は、Sociedade Portuguesa de Robotica (SPR) 主催の Portuguese Robotics Open 2018 に参加する目的で、1/8 スケールの RC カーモデルを用いた、4 輪車両型 RC カーのナビゲーション技術の研究を行った。自律移動 RC カーには、LiDAR センサ (Hokuyo URG-04LX) と、イメージセンサ (Microsoft Kinect One、Stereolabs ZED(RGB-D)、IDS グローバルシャッター) が用いられ、LiDAR による SLAM とイメージセンサによるコンピュータビジョンを組み合わせたセンサフュージョンの方式がとられている。自律移動 RC カーのための経路計画と動作決定モジュールを提案し、実機での適用性を評価した。具体的には、レーンコリドー、クロソイド、スプラインに基づく 3 つの軌道生成手法のロボットオペレーティングシステム (ROS) の実装を行い、ROS のローカルプランナーである `teb_local_planner` に適用した。コンテストでは、6 種類の課題が課され、障害物なしの規定コースの周回、障害物ありの規定コースの周回、工事等による車線減少エリアの通過、障害物なしの縦列駐車、障害物ありの縦列駐車、切り返し操作を含めた駐車場での駐車車両の隣への駐車が実施される。課題の最中に自律移動に失敗すると、失敗の種類により、ペナルティが与えられる。ペナルティは、コースアウト (一部)、コースアウト (全て)、コース設置物への軽度の衝突、コース配置が変わるほどの設置物への衝突、ローカルミニマムによる planner のスタック、経路計画の失敗、姿勢が挙げられる。特に、切り返し操作を含めた駐車の課題では、コースアウトが 1 回と、不規則な駐車動作が 3 回、コース設置物への軽度の衝突が 1 回、経路計画の失敗が 1 回生じる結果が得られた。ここでは、駐車のための切り返し動作による目標地点への駐車を、自律移動の経路計画を 2 つの目標地点に分けて行っている。また、センサフュージョンによる自律移動車両の評価を行っており、LiDAR のみの SLAM での、自律移動の評価に関しては、挙げられていない。

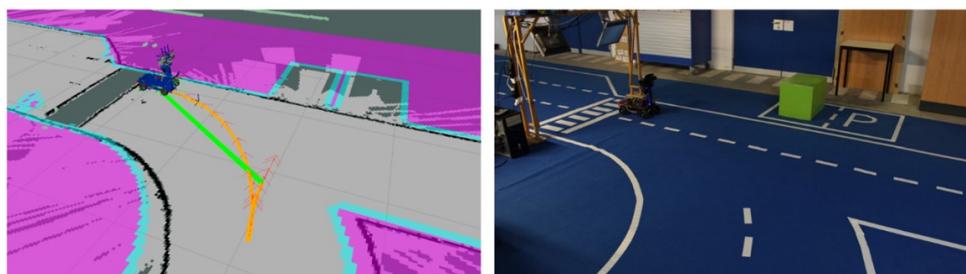


図 4.1: 後退動作

4.2.4 Driverless Car のデザインと実装: 三重県南伊勢町における実証実験

浦瀬ら [?] は、Last One Mileにおいて利用者が自在に移動できるようにする Driverless Car のデザインと設計・実装について、歩行者空間も走れるような自律走行車両を Last One Mileにおいて提供する、設計コンセプトの実証を行った。三重県南伊勢町で行われたフィールドワークをもとに Driverless Car のコンセプトを設計し、設計したコンセプトから実装を行い、これを用いて、南伊勢町における実証実験を行い、その有効性を実証した。文献では、LiDAR センサと IMU を用いたパーソナルビークルの自律移動システムを開発している。ここでは、プロトタイプの開発として、1/10 スケールの RC カーを用いた手動走行、地図作成、自己位置推定、経路計画のシステムを構築している。AMCL による自己位置推定が実装されており、環境地図を用いた位置推定の精度を評価している。椅子やゴミ箱などの物体が存在する環境で、地図に特徴のあるエリアでは、自己位置の推定精度が高く、平坦で長い廊下等の特徴量の少ない経路では、自己位置推定の精度が下がる結果が報告されている。しかし、自己位置推定の実装では環境地図中での手動動作における AMCL の精度のみを評価しており、RC カーの自律移動の精度に関しては、評価が行われていない。また、車型の自律移動特有の切り返し動作においても、同様であったかは、不明である。

4.3 提案

これまでの記述により、自律移動ロボットは、センサ数を少なくする方針で設計されている。また、4 輪車両型の自律移動システムにおいては、比較的少ないセンサ数でのシステム評価、特に、LiDAR のみによる SLAM、ナビゲーションシステムの評価は行われていない。LiDAR とイメージセンサ等によるセンサフュージョン技術による駐車や車庫入れのような複数の移動の組み合わせの評価は行われているが、LiDAR のみによる自律移動を実現することが理想と考えられる。LiDAR のみによる自律移動が実現できれば、制作コストの削減や、保守性の向上が期待される。

そこで、本研究では、遠隔操作と自律移動の機能の切り替え機能を有する、自動運転システムを提案する。遠隔操作機能として、車載カメラ映像をモニタで確認しながら、ハンドル・フットペダルゲームコントローラの入出力によって RC カーの遠隔操作を実現する。遠隔操作の際には、LiDAR SLAM による環境地図を作成し、プログラムを切り替えることで、作成した環境地図中の自律移動を実現する。自動運転機能の評価として、自動運転から手動運転への切り替え機能の評価と、後退機能と旋回機能を組み合わせた、切り返しの機能の検証を行う。

4.3.1 システム構成

以下と、図 4.2 に、本研究で使用するハードウェアとソフトウェアとシステム全体の構成を示す。

1. ハードウェア

Arduino、Raspberry Pi 4 model B、URG-04LX-UG01、モータドライバ(pca9685)、ノートPC

2. ソフトウェア

Ubuntu 18.04 LTS(OS)、ROS Melodic

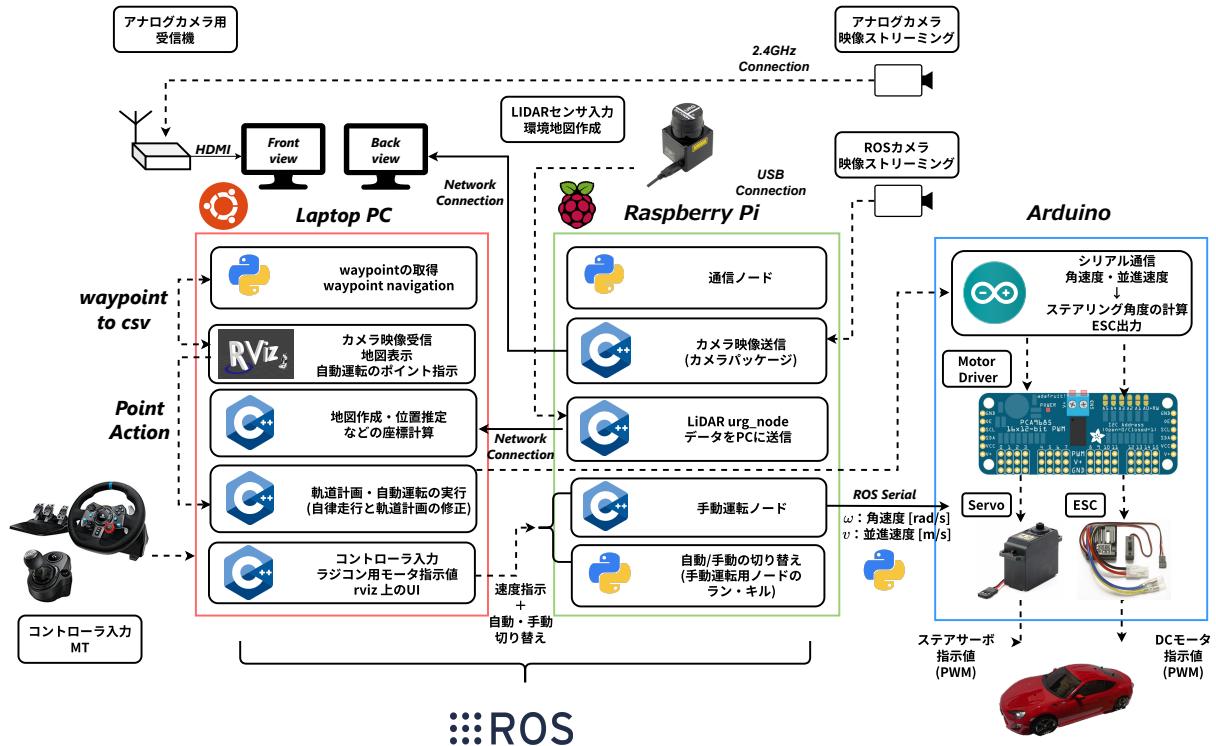


図 4.2: システム構成

4.3.2 システムインターフェース

4.3.2.1 ステアリングコントローラによる遠隔操作

ステアリングコントローラの入力に対するRCカー動作について、ハンドルフットペダルによるモータ制御を図4.3、4.4に示す。ハンドル部分の回転角値(アナログ値)をRCカーの角速度値に変換し、その後さらにサーボモータの制御パルス値に変換し、サーボモータに出力する。サーボモータに連動して、ステアリング機構が駆動する。アクセルペダル部の踏み込み値(アナログ入力)を並進速度値に変換し、その後モータドライバの入力値(DCモータの制御パルス値)に変換し、DCモータに出力する。DCモータに連動して、タイヤ部が回転することで前進・後退動作を実現する。ブレーキペダルを踏みこむことでDCモータの停止値が出力され、タイヤの駆動が停止する。ステアリングコント

ローラのハンドル部とペダル部に回転角を検出するセンサが搭載されている。それぞれのセンサが示す入力値を解析することができれば、入力値に対応させた電圧パルス値の制御の処理を行うことができる。解析したセンサの入力値を RC カー側のコンピュータへと送信し、RC カーの制御用マイコンが RC カーを制御する。Wi-Fi による無線通信をノート PC と RC カー側のコンピュータで行う。制御信号の伝送を RC カー側のコンピュータとマイコンのシリアル通信により行う。ノート PC がステアリングコントローラの入力値を解析して角速度と並進速度値に変換し、Wi-Fi 経由で RC カー側のコンピュータに送信する。RC カー側のコンピュータが、並進速度値と、角速度値をシリアルポート上に送信する。マイコンが並進速度値と角速度値を受信し、モータの制御パルス値に変換することで、RC カーを動作させる。また、本研究では、AT(Automatic Transmission) の他に、MT(Manual Transmission) による操作を行う。前進・後退動作を行う際、MT 操作のパドルシフトとして、ステアリングコントローラに搭載された 2 つのパドルスイッチの入力により、シフトアップとシフトダウンを行うことで、モータの回転速度値を変更する機能を実現する。加えて、MT 操作の H パターンシフトとして、クラッチペダルとシフトレバーの両方の入力を検知することで、RC カーの前進速度の变速と、前進・後退のモード切り替えを行う。

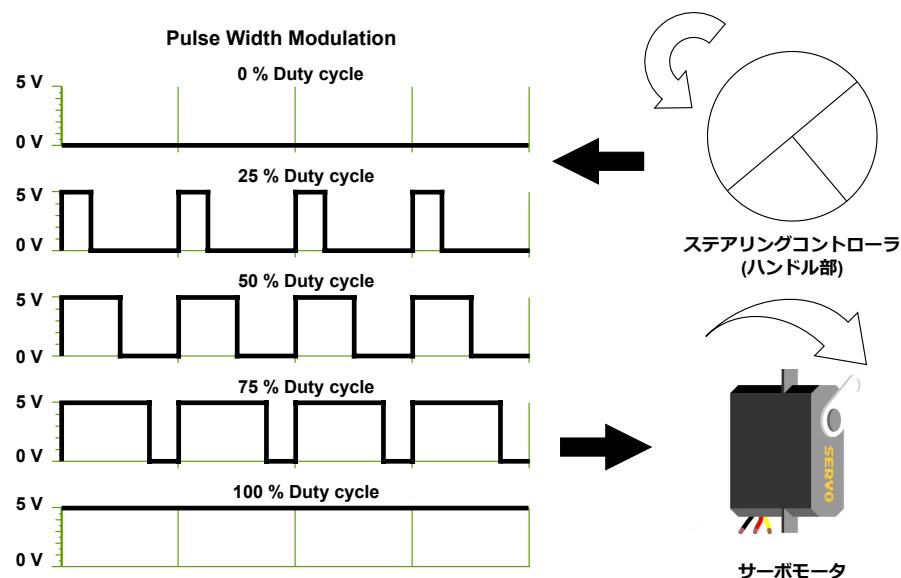


図 4.3: ハンドルによるサーボモータの制御パルス値の取得

4.3.2.2 カメラ映像の取得

RC カーの遠隔操作で、車両側周囲の状況を把握するためにカメラ映像をリアルタイムで送信するシステムを導入する。RC カー側に取り付けられたカメラの映像を Wi-Fi や、高周波帯の電波により運転者側に送信し、受信した後に、ディスプレイに出力する。

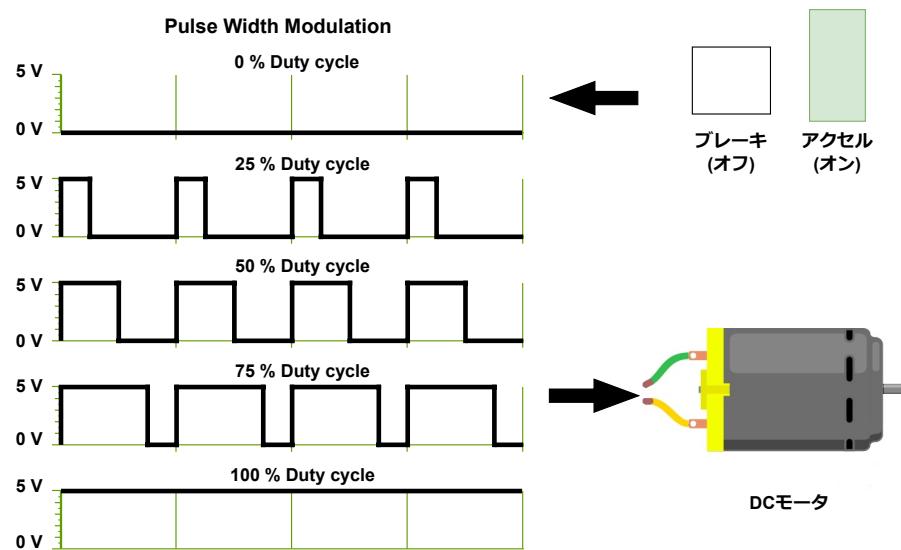


図 4.4: ペダルによる DC モータの制御パルス値の取得

4.3.2.3 SLAM

RC カーに LiDAR センサを搭載し、LiDAR によって取得した、点群データを用いて環境認識を行う。環境認識によって、RC カーの自己位置の推定と、RC カーが走行した周辺の環境地図の構築を行う。LiDAR は、RC カー側に搭載したコンピュータに接続し、点群データを Wi-Fi 通信により、ノート PC へと送信する。また、自己位置・地図構築に関する計算は、ノート PC で行う。全体の流れとして、SLAM と遠隔操作のプログラムを同時に起動し、環境地図を取得する。取得した地図は、ファイル形式を指定して保存をする。

4.3.2.4 自動運転機能

SLAM によって取得した地図を読み込み、地図中で RC カーの初期位置を設定し、目的地点を設定すると、目的地点の座標と、初期位置の座標、地図情報(障害物の位置)等から、最適な経路計画を行う。経路計画に基づいて、並進速度と、角速度を出力し、その値をモータの制御パルス値に変換して出力する。

4.3.2.5 waypoint の設定

環境地図の座標系を下に、自律移動の目的地をコースの形状に沿って複数の点に分割して設定した中継地点を waypoint と呼ぶ。図に waypoint の例を示す。周回コースに設定した waypoint に到着すると、次の waypoint へと自律移動を開始する。設定した waypoint にはマージンを設定し、waypoint から半径数 [m] 程度の閾値を設定し、ロボットが閾値以内に到着すると、次の設定点への経路計画を行う。

4.3.2.6 自動運転・手動運転の引継ぎ機能

AMCL による自己位置推定をしながら、環境地図中を遠隔操作し、地図中の経路の途中でコントローラのボタンを入力すると、遠隔操作のプログラムが終了し、自律移動のプログラムが起動する。自律移動のプログラムが起動している最中に、もう一度コントローラのボタンを入力すると自律移動のプログラムが終了し、遠隔操作のプログラムが再び起動する。

4.4 実装

4.4.1 システムの実装

システムのハードウェアが行う機能を以下に示す。

1. Arduiono:Raspberry Pi とのシリアル通信、モータの制御パルスの出力、モータドライバへの入力
2. Raspberry Pi: ノート PC との通信(ROS マスター)、LiDAR のデータとカメラ映像を PC に送信、Arduino に並進速度と角速度のデータを送信
3. LiDAR:Raspberry Pi とのシリアル通信、環境認識・点群データの取得
4. ノート PC:Raspberry Pi との通信(ROS スレーブ)、地図構築・経路計画の計算、経路計画・環境地図の可視化(RViz)、カメラ映像の受信

4.4.2 ROSについて

ROS(Robot Operating System)[?] は、知能ロボットの研究開発についてのソースコードの汎用性、移植性とモジュール化への要求等の需要を満たすために、2010 年に Willow Garage 社が発表したミドルウェアである。ROS のスタック構成を図 4.5 に示す。ROS は、ロボット開発を行うためのソフトウェアの集まりであり、プロセス間通信のためのライブラリと、プログラムをコンパイルするためのビルドシステムを提供している。ROS の特徴として、アプリケーション側から見た時には、ハードウェアの抽象化、各種ドライバの管理、共有機能の実行管理、ジョブ管理、プロセス間のメッセージパッキングなど、基本 OS の管理機能に類似する上位機能を提供しており、ロボット向けの基本 OS になれるように設計されている。本来の基本 OS は ROS によりカプセル化され、ロボットアプリケーションからはシームレスとなっている。ROS の設計目標の最も重要な要素の一つは、ソースコードのユーザビリティを改善することである。ROS は分散処理システムであり、各種ロボット向けの基本機能のうち、汎用性のある部分をノード(nodes) と呼ばれる機能モジュールに表現されている。そして、ノードを組み合わせることにより、より複雑な制御機能を分散的に実現する。これらのノードモジュールをパッケージ(package) にまとめることができるので、開発済みの機能モジュールの共有と配布が簡単に行えるようになっている。さらに、ROS のファイルシステムでは、統合開発環境をサポートしており、ファ

イールレベルからライブラリレベルまでの設計、開発、管理を独立的に行うことができる。ROSは、分散型処理システムであり、各ノードは通信モジュール間のピアツーピアネットワークで関係づけられている。このネットワーク上で同期型リモートプロシージャコール(remote procedure calls, RPC)を基にしたノード間の通信や、トピック(topic)の配布/購読により実現される非同期型データ通信、パラメータサーバでのパラメータ(グローバル変数)の分散的格納と取り出し機能を提供されるが、厳密なリアルタイム性を持たない。

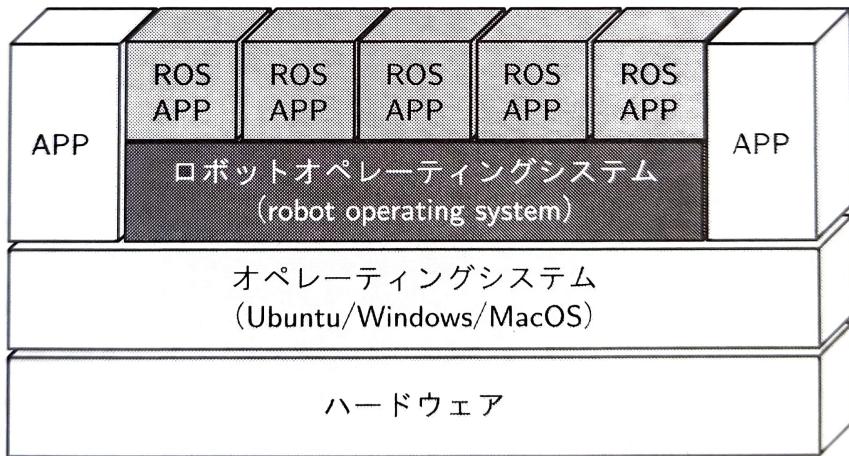


図 4.5: ROS のスタック構成

4.4.2.1 ピアツーピア設計方式

ROS の実行ジョブは、分散された一連のプロセスで構成される。図 4.6 に、ROS のピアツーピア設計例を示す。これらのプロセスは同じホスト、または、別々のホスト上に分散させることができ、プロセス間ではピアツーピアネットワークを構成する。このようなピアツーピア設計を利用することにより、サービス管理やノード管理等を、ビジュアル処理部または音声処理部等のロボットの補助処理部から、計算負荷を分散させることができる。

4.4.2.2 多言語に対応

様々な開発者のニーズに対応するために、ROS では、C++、Python、Java 等、複数のプログラミング言語をサポートしている。複数の言語が混在する環境では、ROS はプログラミング言語に独立する簡単なインターフェース定義言語を提供し、各モジュール間でのメッセージ通信を表現する。インターフェース定義言語で構成されるテキストファイルでは、各ノードのメッセージの構造を表現している。これにより、オブジェクトコードを生成する際に、対応するコードをノードごとに生成することができる。

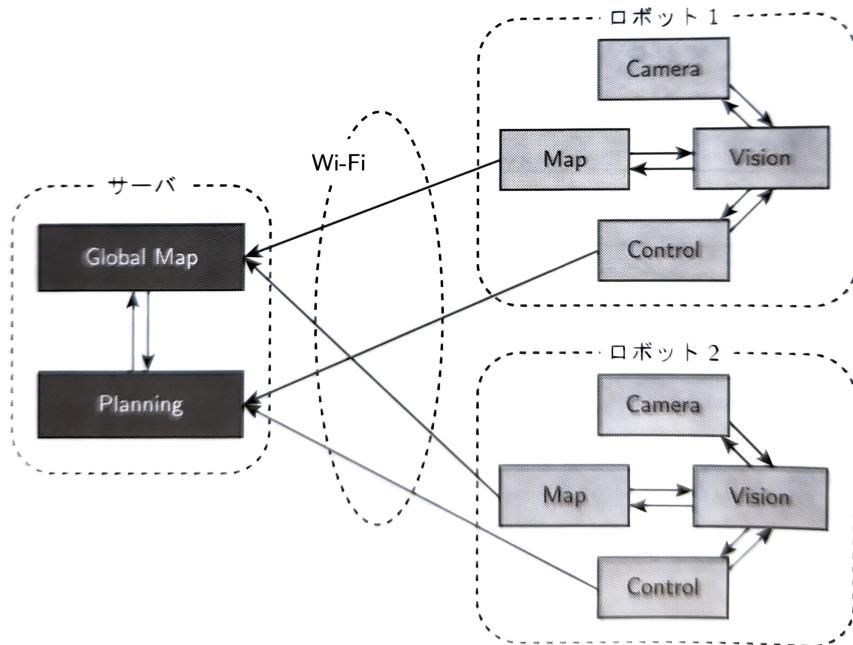


図 4.6: ROS のピアツーピア設計例

4.4.2.3 機能の細分化と集約

既存の手法で知能ロボット向けのソフトウェアを開発する場合、重複作業が多く、本来共有すべきドライバ等のソースコードは、各ロボットのミドルウェアの正弦で抽出しにくく、他のロボットへ採用することが困難であった。ROSの場合、汎用性のある機能、または計算処理部を独立性のあるモジュールにしており、各モジュールはCMakeにより単独にコンパイルすることができる。さらに細分化された汎用性のある機能モジュールをライブラリに集約することにより、より複雑な機能が実現しやすくなり、ユーザビリティの改善が期待できる。

4.4.2.4 補助ツールが豊富

複雑になっていくROSパッケージを管理するために、ROSではさまざまなツールを用意して、ROSパッケージのビルト管理や、新しい機能モジュールの開発を比較的簡単にしている。ROSのコア部は、知能ロボットを制御する基本的な部分しかもっておらず、モジュールを追加することにより、様々なバリエーションをもたらすことができる。これらの補助ツールは、全てのロボットの共通機能を組み合わせるために方法を提供している。

4.4.2.5 オープンソースのプラットフォーム

ROSの全てのソースコードが一般公開されており、それらのほとんどがBSDライセンスのもとで、使用できる。つまり、非商用/商用での使用が許可されている。

4.4.2.6 コンピューティンググラフ

ROS のコンピューティンググラフはデータの処理場の依存関係を表すもので、ポイントツーポイントネットワークで示される機能グラフのことである。ROS のコンピューティングレベルでは、ノード、マスター、パラメータサーバ、メッセージ、サービス、トピック、バッグという機能単位があり、これらを機能的に表現することにより、ネットワークグラフが構成される。

1. ノード

処理機能の基本単位で、処理プロセスを表す。図??にROSのコンピューティングレベルを示す。ROSでは、プロセス間のデータ通信を実現するために、各プロセスをノードで表現する必要がある。通常では、ROSシステムは様々なノードより提供されている機能で構成されているが、ノードの設計方針として、複雑な機能表現、あるいは、たくさんの機能を持たせるのではなく、できるだけ単一機能にする、ということがある。そして、このように単純化されたノードを組み合わせることにより、より複雑な機能を実現する。ノードはrosCPPやrospy等、ROSのクライアントライブラリにより生成することができる。

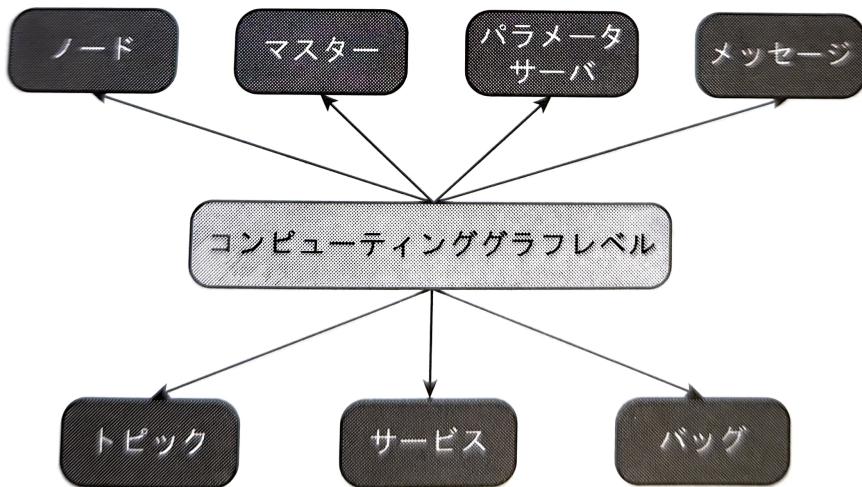


図 4.7: ROS のコンピューティングレベル

2. マスター

ROS のノードやトピック、サービスの管理を行う機能単位である。ROSシステムの中でマスターが起動していないければ、ノード間の通信ができない。マスターを利用すれば、異なるコンピュータの中のノード間の通信も実現することができる。マスターの仕組みは、roscore というプロセスにより提供されている。roscore は、ノードやトピック、サービスなどの名前の登録、ノードのURIの検索、トピックの検索、トピックに対する購読者の参加通知等を提供する。ROSシステムの中で、roscore は必ず一つ立ち上げておく必要がある。

3. パラメータサーバ

ROSにはトピックの他に、数値等のデータを送受信する仕組みとしてパラメータ通信機構がある。パラメータは、パラメータサーバにグローバル変数として保持しておくことができる。パラメータは本来、ノードをコンフィグレーションする目的で利用される。パラメータサーバを利用すれば、ほかのノードと協調しながらノードのダイナミックコンフィグレーションを実現することができる。

4. メッセージ

前述のとおり、ノード間で通信を行うデータ単位の一つであり、様々なデータタイプを持つ。

5. トピック

ROS ネットワーク中で転送される各メッセージは識別子を持つ必要がある。図 4.8 に ROS トピックを利用した情報共有を示す。配布者が提供するデータは、トピックの形で ROS ネットワーク中に公開される。購読者は、自分の関係にあるトピックを常に監視していて、自分のメッセージタイプでトピックのデータを受け取って利用する。つまり、トピックはメッセージのインスタンスであると考えてよい。メッセージのインスタンスを生成する側は配布者、利用する側は購読者である。例えば、カメラ映像で取得した画像を利用する複数のノードがあるとすると、カメラノードは画像メッセージをトピックとして発行し、他のノードはそのトピックを監視して、カメラ情報を分散的に利用することができる。なお、トピック名は、ROS システム中で動作しているノード間での唯一性を保証する必要がある。

6. サービス

ノード間で通信を行うための一つの手段である。サービスは、ROS ネットワークの中でただ一つの名前で識別される。

7. バッグ

ROS メッセージの記録と再生に使われる。バッグは、収集しにくいセンサデータを処理するアルゴリズムを開発・テストする際に、データを収納する重要なメカニズムであり、複雑なロボットを制御するロボットのプログラムを開発する際に大変重要な役割を果たす。

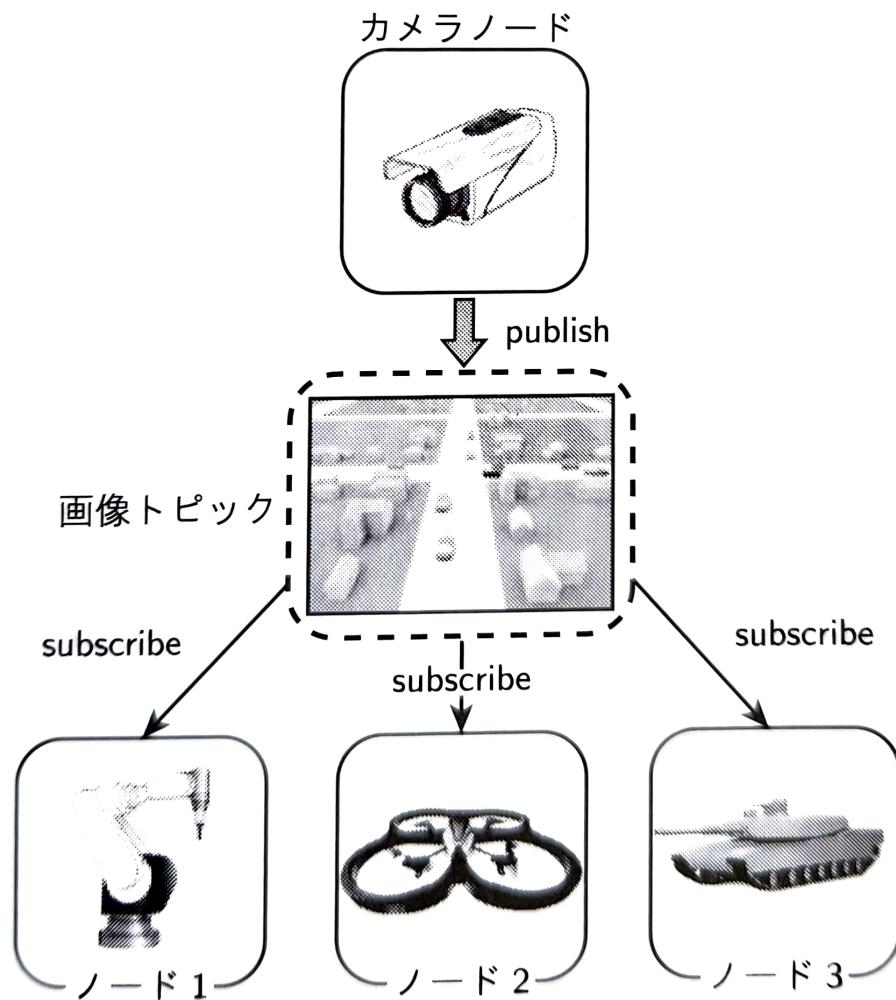


図 4.8: トピックを利用した情報共有

4.4.3 遠隔操作機能の実装

4.4.3.1 コントローラ入力値解析

ステアリングコントローラはゲーム用のコントローラや、ジョイスティック全般と共に、ROS の joy パッケージを用いて、解析を行う。joy パッケージは、ゲーム用のコントローラのボタン全てにディジタル入力とアナログ入力を与える。

4.4.3.2 ステアリング・前進後退機能 (AT・MT)

コントローラ入力値解析によって、得られた入力値のうち、ハンドル入力値を角速度(単位:[rad/s])の値に変換する。アクセル・フットペダルの入力値を並進速度(単位:[m/s])の値に変化する。なお、角速度、並進速度等のトピック通信には、ROS の定義済みメッセージタイプである geometry_msg(float32) を用いている。

4.4.3.3 ROS マスタースレーブ (Laptop to Raspberry Pi)

ROS システム内の全てのホストは、互いに到達できることが要求される。それぞれのホストは同じネットワークに所属しなくても良いが、ルータ等を経由して必ずたどりつけるようにしなければならない。(DNS などで IP アドレスが解決できることと、宛先への経路が確保できる)ROS システムのホスト(マスター)を Raspberry Pi に、スレーブをノート PC に設定する。設定は、ROS システムのホストを決める環境変数にマスターとするマシンの IP アドレスを登録すれば、プログラム起動時に自動的にマスターを検索して接続する。

4.4.3.4 シリアル通信 (Raspberry Pi to Arduino)

Raspberry Pi と Arduino 間の通信として、ROS のシリアル通信用パッケージ rosserial を用いる。rosserial は、主に、python ノードや、Arduino をクライアントとすることを前提とされており、シリアルポート、ボードレート等を指定し、通信ノードとして起動させることで、ホストとクライアント間でトピック通信を行うことができる。

また、Arduino 側に ros_libAPI を用いることで、Arduino が ROS のシリアルクライアントとして ROS のマシンとトピックの通信を行うことができる。

4.4.3.5 カメラ映像機能

カメラ映像には、5.8 [GHz] 電波帯と VTX(Video Transmitter) を用いることで、リアルタイム映像を受信することができる。また、ROS 側でも、USB カメラを Raspberry Pi に接続することで、可視化ツール RViz 上に映像を送信する。

4.4.4 SLAM の実装

4.4.4.1 urg node

北陽電機社製の LiDAR は、北陽電機株式会社が開発した C++ ドライバソフトウェア urg_node を ROS 環境にインストールすることで点群データの取得に使用できる。urg_node で使用するシリアルポート、検知角度等のパラメータを設定する。

4.4.4.2 Gmapping

ROS で既に実装されている SLAM メタパッケージの 1 つとして、Gmapping が挙げられる。Gmapping は、LiDAR のデータとオドメトリ情報(必須) を用いて SLAM を行う。ROS を用いた SLAM で最も多く使用されているパッケージである。

4.4.4.3 laser scan matcher

今回の開発では、ロータリーエンコーダや、IMU 等の内界センサを用いないため、Gmapping パッケージを用いるための内界センサ以外からのオドメトリ情報が必要である。その一つとして、レーザオドメトリを使用する。レーザオドメトリとして、LiDAR のスキャンデータの差分から、オドメトリデータを生成する laser_scan_matcher パッケージを用いる。

4.4.5 自動運転機能の実装

自動運転機能は、Gmapping による SLAM で生成された地図上の経路計画を行う Navigation Stack[?] と、自己位置推定を行う AMCL[?] を用いる。以下に Navigation Stack と AMCL について述べる。

4.4.5.1 Navigation Stack

ROS では、自律移動を目的としたメタパッケージ Navigation Stack[?] が構成されている。図 4.9 に Navigation Stack の概要を示す。Navigation Stack とは、オドメトリ情報(ホイールエンコーダ等の内界センサによる推定自己位置及び現在速度)、および、LiDAR の情報(障害物までの距離情報)を入力値として、与えられた目標地点/姿勢に到達するための安全な駆動(速度)命令を出力するソフトウェアである。LiDAR の情報は、平面レーザによる 2D レーザスキャンまたは 3D デプスカメラ等によるポイントクラウドを利用可能である。Navigation Stack を扱うためには、TF(Transform tree) と呼ばれる座標軸の相対位置情報を与える必要があり、オプションで、Navigation Stack に対して地図を与えることができる。地図を与えることで、Navigation Stack はより効率的に移動経路の探索を行うことができる。



図 4.9: Navigation Stack の概要

4.4.5.2 Transform Tree

Transform Tree(tf) とは、ロボットが存在する空間及びロットの構成要素がそれぞれ持っている座標軸ごとのずれ (x,y,z の相対位置及び相対姿勢) がどの程度であるかを、親子関係のツリー構造で表現したものである。例えば、障害物からの距離情報は、ロボット上に設置されたレーザから取得されるが、ロボットの障害物回避を考える場合は、ロボットの基準面の位置で考える必要がある。これを求めるために、基準面の座標軸とレーザ座標軸とのずれを Navigation Stack に入力する必要がある。

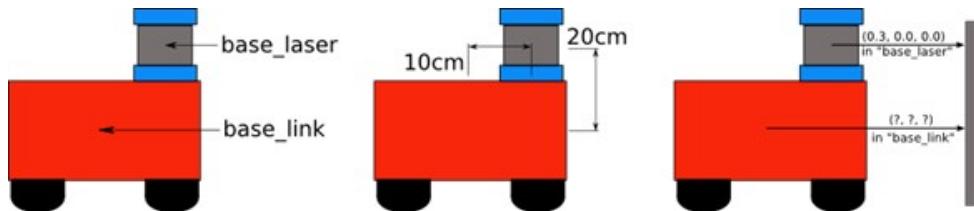


図 4.10: Transform Tree の説明

各座標軸は、`frame_id` と呼ばれる文字列により識別される。Navigation Stack が動作するには、基本的な構成の場合は、map フレーム、odom フレーム、base_link フレーム、laser フレームの 4 つの構成要素の相対位置・姿勢 (3 つの Transform Tree 情報) が必要である。map フレームは、Transform Tree の頂点座標軸で、ロボットが移動する空間で固定となる絶対座標系である。odom フレームは、ロボットのオドメトリ (内界センサーによる推定自己位置) を表すための座標軸である。map フレームと同じ空間固定座標軸であり、車輪の空転やロボットの横滑り、オドメトリの計算誤差がないならば、map フレームとのずれは理論上常にゼロとなる。(実際には計算誤差やスリップ等があるため、map フレームとのずれは時間と共に拡大していく。) base_link フレームは、ロボット基準面の中心を原点とする座標軸である。ロボットの移動に伴い、odom フレームとの相対位置・姿勢が変化していく。laser フレームは、ロボット上の LiDAR の基準点を原点とする座標軸である。一般的には、LiDAR はロボット上に固定されるため、base_link フレームとの相対位置・姿勢は固定値となる。

1. LiDAR 情報 (レーザスキャン)

レーザスキャンデータは、laser 座標系における、スキャン範囲内の各スキャン谷における、障害物までの距離を通知する。図 4.11 にレーザスキャンのパラメータ値を示す。例えば、測定範囲が 180 度で、測定単位が 5 度のレーザセンサであれば、36 個の距離データ (ranges) を通知する。測定範囲内に何もない部分については、無効値 (range_max 値または無限大) を入れておく。測定角度は、前方正面が 0 ラジアンで、右側がマイナス値、左側がプラス値となる。

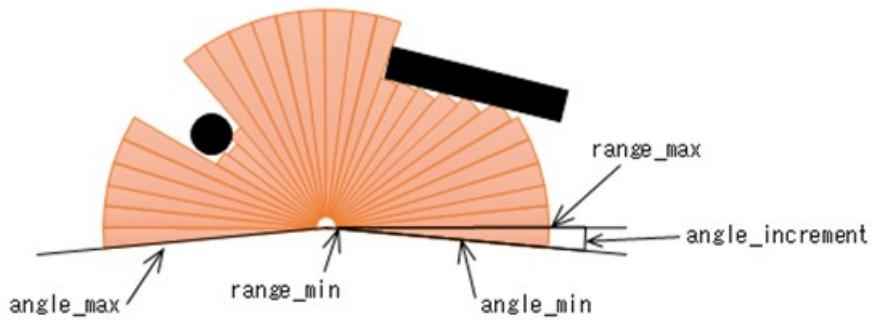


図 4.11: レーザスキャンのパラメータ値

2. オドメトリ情報

オドメトリ情報は、odom 座標軸におけるロボット (base_link 座標系) の推定自己位置及び現在速度 (並進速度および回転速度) を通知する。図 4.12 にオドメトリの座標系を示す。推定自己位置は、多くの場合、車輪駆動を制御するノードが、車輪回転量から推定される自己位置情報を通知する。(IMU 等、車輪回転量とは別の何等かの手段を組み合わせて、オドメトリ情報の精度を上げることも可能である。) 現在速度は、2D のナビゲーションでは、x 軸並進速度及び z 軸回転速度が設定される。座標系の考え方は、下図の通りで、ロボット前方が x 軸正方向、ロボット左方向が y 軸正方向となる。z 軸回転速度は、左回転が正方向である。odom と base_link の相対位置については、TF でも同じ内容が通知される。Navigation Stack は、TF で通知される位置情報を見ており、オドメトリ情報側の自己位置情報は、実際には参照されていない。位置・姿勢の共分散について、x,y,z 軸位置および x,y,z 軸姿勢の 6 つの要素について、推定値と実際の値との掛け合わせの相関関係を、 6×6 の行列データとして設定する。速度の共分散についても、同様に、x,y,z 軸並進速度および x,y,z 軸回転速度の 6 つの要素について、推定値と実際の値の掛け合わせの相関関係を、 6×6 の行列データとして設定する。

3. 地図

オプションとして、Navigation Stack に地図を与えることで、LiDAR で見えていない後方や遠方、死角の情報も加味して経路検索を行うことができる。地図は、OccupancyGrid というデータ形式で表現される。5cm 四方など、決まったサイズのセル

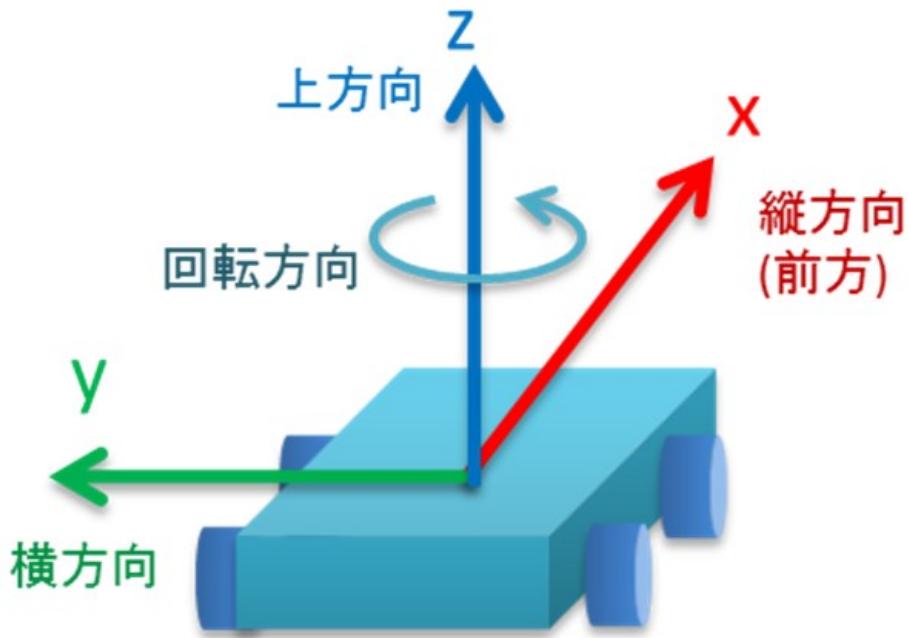


図 4.12: オドメトリの座標系

で地図を区切り、各セルを「占有されたセル(100)」、「占有されていないセル(0)」、「未知のセル(-1)」の3つで区分して地図を表現する。地図情報は、解像度(セルの大きさ)、地図の大きさ(縦横のセル数)、オリジナル(map座標系におけるセル(0,0)の位置)、その一つとして各セルの占有率を保持した配列データからなる。

4. 駆動(速度)命令

安全な経路を検索した結果として、Navigation Stack から駆動命令が出力される。速度は、並進速度(x,y,z)および回転速度(x,y,z)で表現され、一般的な2輪差動型のロボットの場合は、 x 軸並進速度と z 軸回転速度のみ指定される。座標系の考え方は、オドメトリ情報の現在速度と同じである。

5. MoveBase アクション

Navigation Stack へ、目標位置・姿勢を指定してロボットの移動を指示し、フィードバック(現在位置)および結果を受け取る。

6. AMCL

AMCL(Adaptive Monte Carlo Localization)は、2Dで移動するロボットのための確率的な位置推定システムである。パーティクルフィルタ(またはKLDサンプリング)を使用したモンテカルロ法を用いて、既知のマップに対するロボットの姿勢の追跡を行う。

7. パーティクルフィルタ

パーティクルフィルタは、逐次モンテカルロ法とも呼ばれ、逐次ベイズ推定の一種

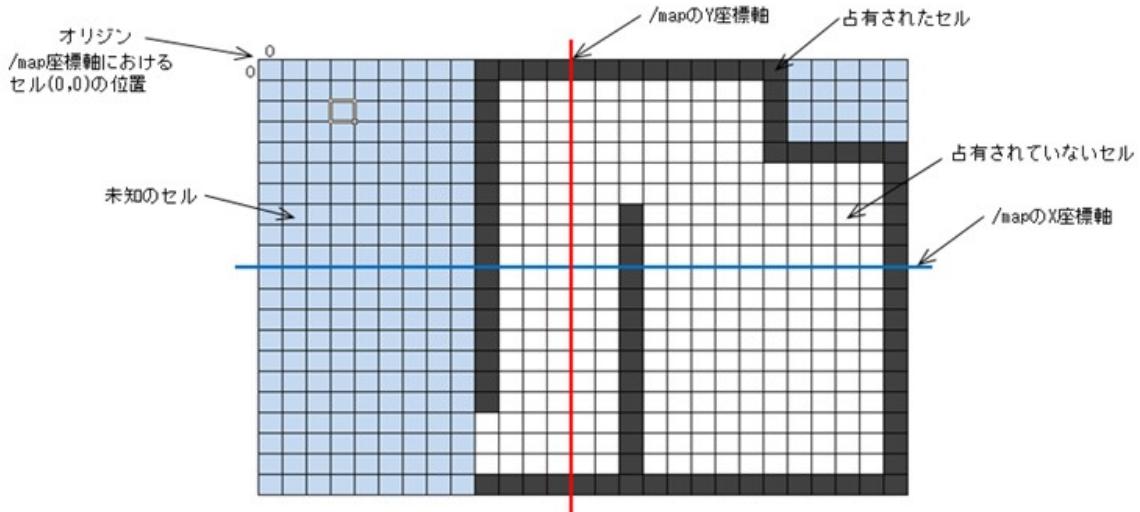


図 4.13: 占有格子地図

で、現在の状態から想定される多数の次状態を、多数(数百か数千)のパーティクルに見立て、全パーティクルの尤度に基づいた重み付き平均を次状態として予測しながら追跡を行っていくアルゴリズムである。パーティクルの移動の予測や外れてしまったパーティクルのリサンプリングを行うことで、ガウス性のないノイズにも強い手法となっている。パーティクルフィルタは基本的に以下3つのサイクルを繰り返し実行することで推定を行う。予測を何回か行うごとにリサンプリングを行う。

4.4.5.3 パーティクルフィルタについて

1. 予測

図4.14に、パーティクルを用いた予測を示す。動作モデルから求めた運動量を基に、それぞれのパーティクルを次のタイミングのパーティクルの位置に移動させる。各パーティクルで任意のノイズを仮定して、もとあった領域よりも広い領域に粒子をばら撒く。各パーティクルは、「運動モデルによって想定されるロボットの位置」という状態値の仮説を表している。

2. 尤度の計算

観測データにより、各パーティクルにおける尤度を計算する。図4.15に尤度の計算を示す。それぞれのパーティクルがどのくらい正解に近いかを観測データで決定し、実際の位置の重み付き平均やリサンプリング時の優先度等に使用する。レーザセンサの場合、各パーティクルでのレーザセンサの位置と壁の位置によってそのパーティクルの尤度を計算する。

3. リサンプリング

図4.16 予測を繰り返すとパーティクルが広がってしまうため、尤度の高

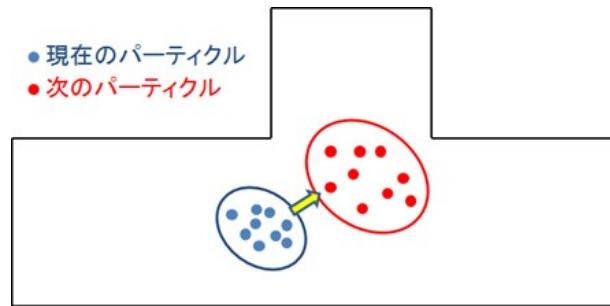


図 4.14: パーティクルを用いた予測

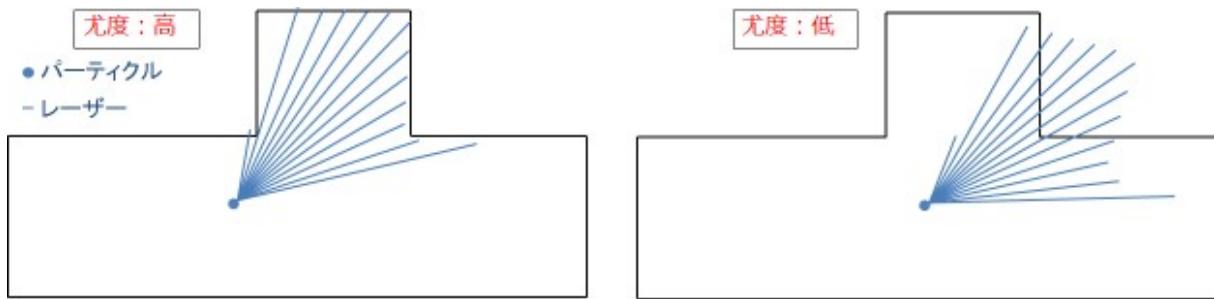


図 4.15: 尤度の計算

いパーティクルはそのまま残して、尤度の低いパーティクルを尤度の高いパーティクルの周りに再配置する。全てのパーティクルを尤度の高いパーティクルの周囲に配置した場合、局所解に陥り、のノイズ等に対応しにくくなるため、一部パーティクルはランダムで配置する。

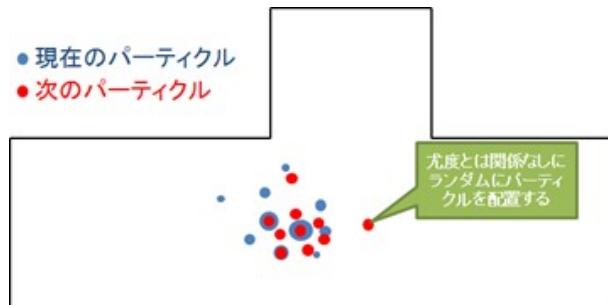


図 4.16: リサンプリング

4.4.5.4 AMCL におけるパーティクルフィルタ

AMCL のパーティクルフィルタの工夫について述べる。

1. オドメトリによる予測

一つ前のパーティクルの位置姿勢に対し、現在のオドメトリ情報により、現在のパーティクルの位置・姿勢を予測する。運動モデルのタイプは diff、omni、diff-corrected、omni-corrected の 4 つがある。今回は、実際に使用した diff モデルについて述べる。diff モデルでは、オドメトリの並進速度に対する。各パーティクルへの回転速度・並進速度へのノイズ、オドメトリの回転速度に対する。各パーティクルへの回転速度・並進速度へのノイズが設定できる。前後にのみ移動するロボットに対するモデルになっている。具体的な計算方法は以下のようになる。

- 前回の AMCL でのヨー角に対する現在のオドメトリの移動方向の角度の差分を始点角度差分とする。
- オドメトリの移動方向の角度と現在のオドメトリの回転速度の差分を終点角度差分とする。
- 始点角度差分・終点角度差分・並進速度に対してガウシアンノイズを追加する。
- 各パーティクルにおいて、パーティクルのヨー角に始点角度差分を足した角度を移動方向とする。
- 移動方向に対して並進速度の移動量を足して、現在のパーティクルの位置姿勢を求める。

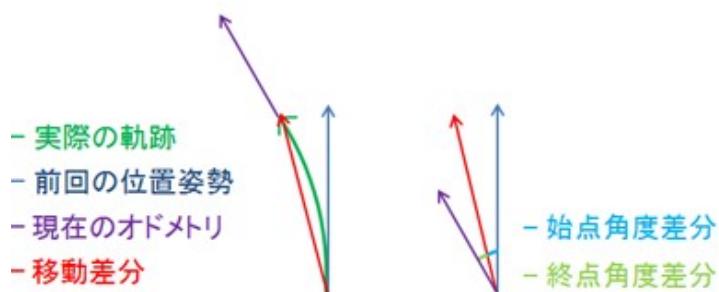


図 4.17: diff モデル

2. レーザによる尤度計算

外れたパーティクルの尤度が低すぎる場合は、リサンプリング時に外れたパーティクルが削除されすぎるため、パーティクルフィルタの安定性がなくなる。そのため、ガウシアンモデル以外のノイズも用いてレーザの値が壁と離れている場合でも尤度を持たせている。amcl ではレーザの尤度計算にビームモデルと尤度モデルを用いた推定の 2 種類が選択できる。ビームモデルと尤度モデルについて説明を行う。ビームモデルでは、「ガウシアンノイズ」と「障害物対策ノイズ」「ランダムノイズ」の組み合わせで使用している。それぞれのパーティクルの位置で各レーザ方向に伸ばし

ていき、マップ上で壁に当たる距離を計算し、その距離と実際のレンジの値によって尤度を計算する。そのレーザが返ってくると期待される値によって尤度を求めていく。壁に当たる位置はレーザごとに異なるため、「パーティクル数」×「レーザ数」分、マップ上で壁に当たる位置を計算する必要がある。ガウシアンノイズは、マップ上で壁に当たる距離(レーザで期待される値)の周辺では尤度が高くなるようになる。障害物対策ノイズは、地図上のレーザが当たる地点より前に、地図にない障害

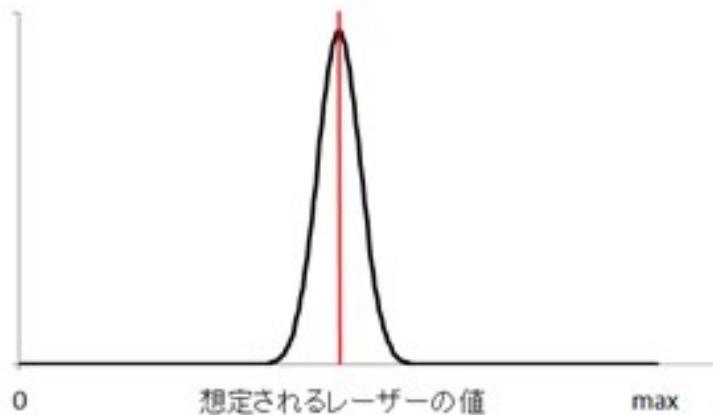


図 4.18: ガウシアンノイズモデル

物がありレンジの値が小さくなる場合がある。その際の尤度を大きくするため、障害物対策のノイズでは、想定されるレンジより小さな値に尤度を少し上げるようなノイズを追加する。センサの最大距離ノイズは、乱反射等によりレーザがセンサに

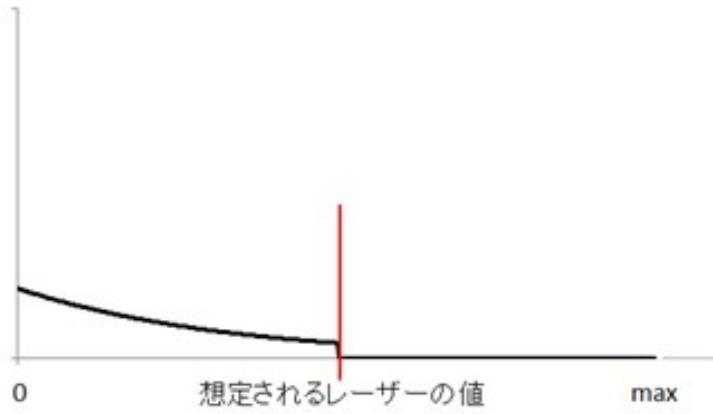


図 4.19: 障害物対策モデル

帰ってこなかった場合、レンジの距離はセンサの最大値が設定される場合がある。その際に尤度を大きくするため、センサの最大距離ノイズでは、センサの最大距離部分では尤度が高くなるノイズを追加する。パーティクルフィルタで安定して動作させるため、ある程度外れてしまっているパーティクルが全て削除されないように、想定されるレンジの値に関わらず尤度を持たせる。ランダムノイズでは、

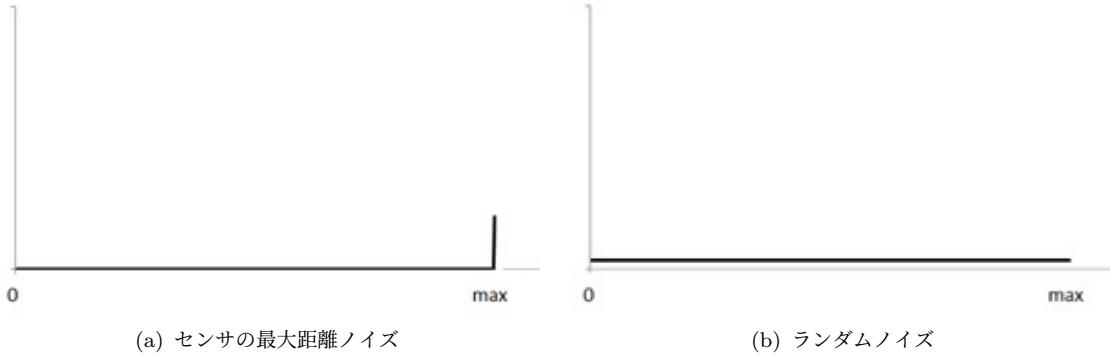


図 4.20: センサノイズの分類

全ての距離で一律の尤度を持たせるようなノイズを追加する。これらを総合した尤度のモデルは以下図 4.21 のようになる。各レーザに以下のモデルで尤度を計算していく。各ノイズの重みは、パラメータにより設定できる。実行する状況に合わせてノ

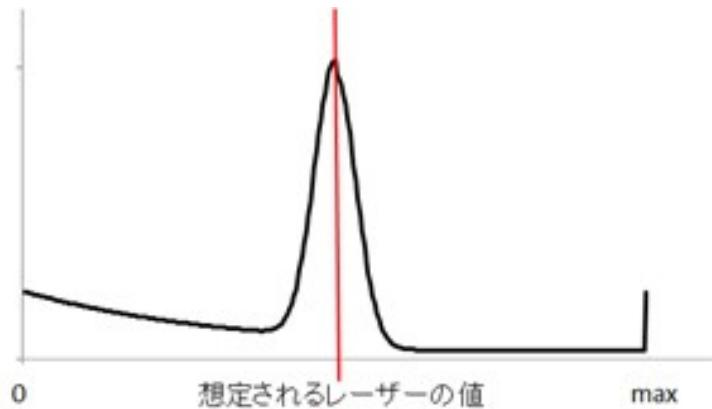


図 4.21: 融合尤度モデル

イズの重みを変更することができる。各重みの和が 1 になるようにする必要がある。尤度モデルを用いたレーザは、「ガウシアンノイズ」と「ランダムノイズ」の組み合いで使用している。マップの座標ごとにガウシアンノイズが決定するため、ビームモデルと異なり、各レーザごとに壁に当たる位置を判定する必要がなく、レーザの当たっている位置を求めるだけでよくなる。ガウシアンノイズは、レーザモデルとは異なり、壁からの距離で求める。壁に近い方が、尤度が高くなる。レーザが当たった座標の尤度を使用する。ランダムノイズは、ビームモデルと同じで、パーティクルフィルタで安定して動作させるため、ある程度外れてしまっているパーティクルが全て削除されないように、想定されるレンジの値に関わらず尤度を持たせる。ランダムノイズでは、全ての距離で一律の尤度を持たせるようなノイズを追加する。

ビームモデルは各レーザに計算を行い処理に時間がかかるため、レーザ数・パーティクル数が多い状態でリアルタイムに実行したい場合は尤度モデルを使用するのが良

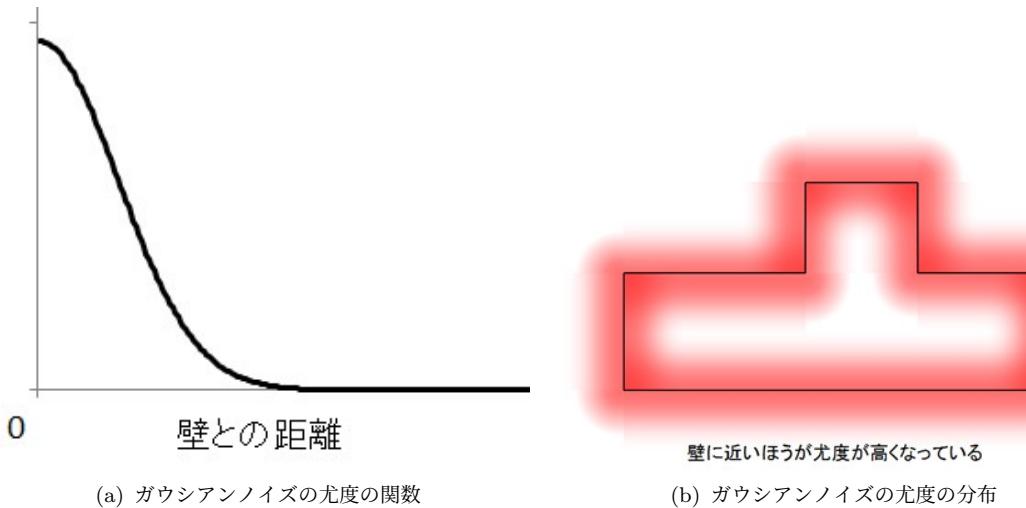


図 4.22: ガウシアンノイズ

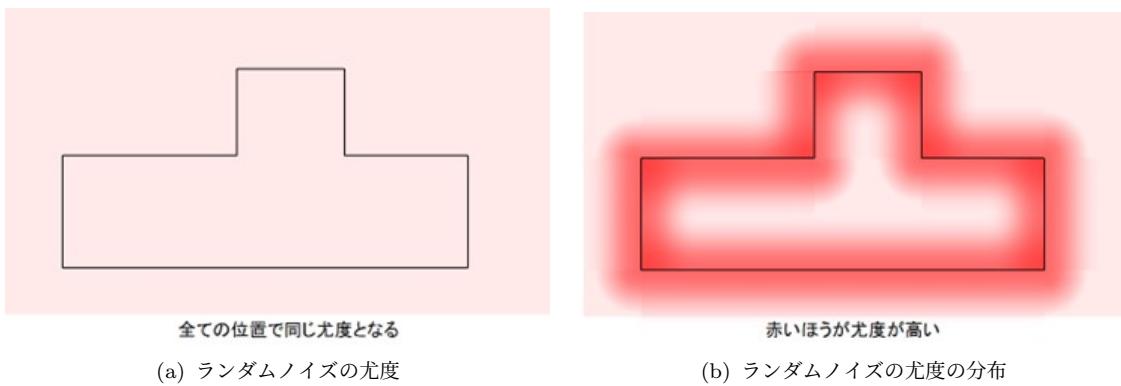


図 4.23: ランダムノイズ

い。また、尤度モデルでは、障害物対策ノイズ等が使用されていないため、人が多い等でマップと異なる障害物が多い場合は、ビームモデルが良いと思われる。

4.4.5.5 TEB local planner

`teb_local_planner[?],[?]` は、前述の Navigation Stack の経路計画の内、Timed Elastic Band を用いた、局所計画 (local planner (controller)) のアルゴリズムの一つである。DWA(Dynamic Window Approach) に比べて複雑な回避軌道を生成できるが、計算コストが比較的高いという特徴がある。計算コストを抑えつつ意図した挙動を実現するためには、アルゴリズムを理解して、パラメータ調整を行う必要がある。

ナビゲーションモジュールにゴールが与えられると、以下のような流れを繰り返して、ゴールまでの移動を実現する。`teb_local_planner` は、軌道計画と速度・角速度制御を扱う。

1. 自己位置および障害物の観測・推定

2. 経路計画: ゴールまでの間の最適な Pose 列を計算する。単純な場合、ゴールが与えられるたびに一度だけ
3. 軌道計画: ゴールまでの間の最適な各時刻の Pose を計算する。
4. 速度・角速度制御(目標速度・角速度を計算する。)

1. Timed Elastic Band

Timed Elastic Band は、軌道を「Pose(位置と角度)の列と Pose 間の移動に掛かる時間の列」で表現するモデルである。 i 番目の Poses $_i(x_i, y_i, \beta_i)$ から $i + 1$ 番目の Pose $s_{i+1}(x_{i+1}, y_{i+1}, \beta_{i+1})$ までの移動にかかる時間が ΔT_i の時、図 4.24 に示す、TEB の模式図のように表される。次に TEB の最適化について述べる。

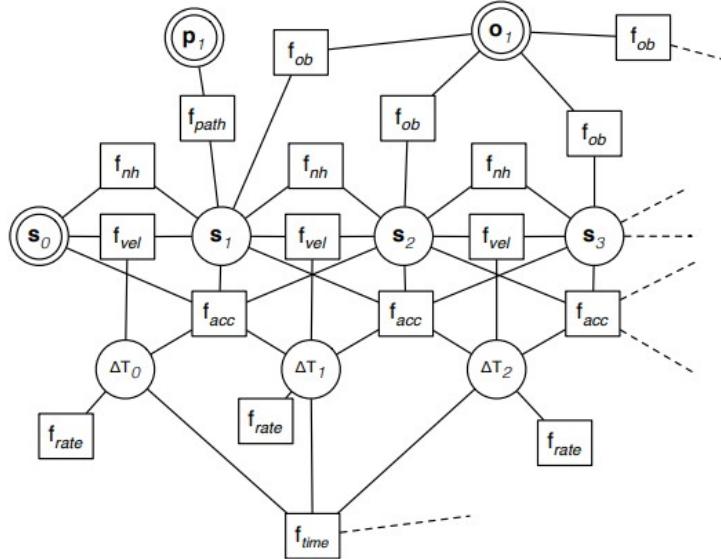


図 4.24: TEB の模式図

2. 制約・目的関数

`teb_local_planner` は以下の制約・目的関数を満たすように最適な軌道を計算する。

- (a) 速度制約(2つの連続した Pose と Pose 間の移動にかかる時間)
- (b) 加速度制約(3つの連続した Pose と 2つの Pose 間の移動にかかる時間)
- (c) 障害物/waypoint(障害物/waypoint の近くの($k = 3\sim 5$ 個の)Pose)
- (d) 非ホロノミック制約
- (e) 最短時間
- (f) 最短距離

隣接する Pose 間の関係によって定義される制約・目的関数は、疎行列(成分のほとんどが零である行列)で表現することができるので、疎行列最適化ライブラリを使用して、計算を行う。

3. 疎行列最適化

上記の制約・目的関数を Hyper-Graph(エッジが複数のノードをつなぐグラフ)で表現して、g2oで最適化する。図 4.25 に疎行列最適化を示す。s は Pose、p は waypoint、

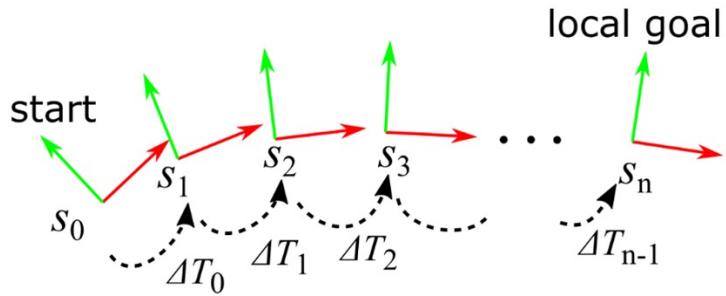


図 4.25: 疎行列最適化

○は障害物、 ΔT が Pose 間の移動にかかる時間のノードを表している。これらのノードを制約・目的関数のエッジで接続することによって最適化の対象をグラフで表現する。

4. auto resizing

Timed Elastic Band を最適化した後、サンプリング間隔が不適切になる場合がある。例としては、以下が挙げられる。

- (a) 障害物を回避するために経路が最適化され、サンプリングが粗くなる。
- (b) ゴールが近づいて、サンプリングが過剰に細かくなる。サンプリングが粗いと軌跡の表現能力が落ち、サンプリングが過剰に細かいと計算コストがかかる。auto resizing は、Pose および Pose 間の移動にかかる時間を挿入したり、削除したりすることによって、この問題を解消する。以下図 4.26～4.27 にサンプリングが粗くなる場合の処理の流れを示す。最適化前はほぼ等間隔にサンプリングされた状態である。最適化後は、サンプリングの間隔が粗くなることがある。図 4.28 に示す、auto resizing によって、Pose および Pose 間の移動にかかる時間を挿入して、サンプリング間隔を調整する。

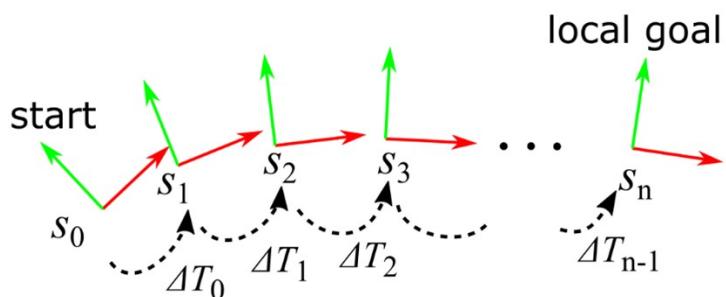


図 4.26: 最適化前

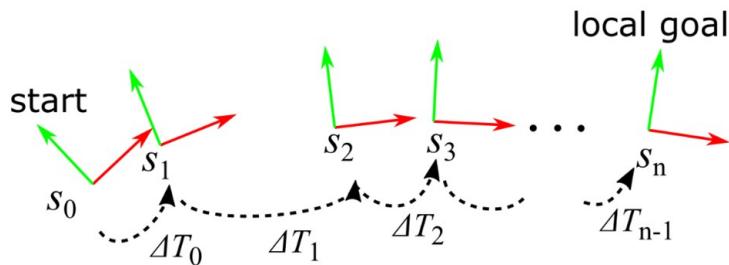


図 4.27: 最適化後

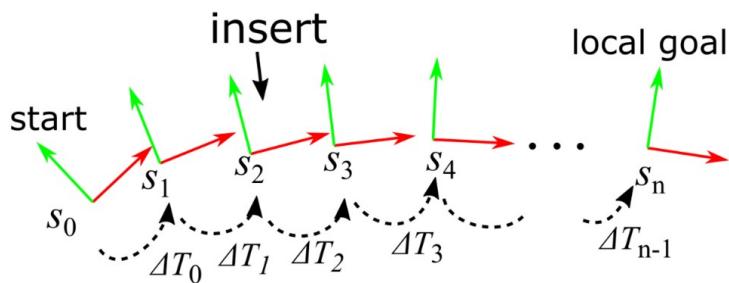


図 4.28: auto resizing 後

5. TEB の最適化処理の流れ

Navigation Stack から並進速度・角速度の計算要求があるたび、デフォルトでは、auto resizing(Insert/delete TEB states)、Hyper-graph の最適化(5回の Levenberg-Marquardt の反復)という流れを4回反復する。

6. 初期軌道生成

TEB では軌道を連続的に変形するため、障害物をまたぐような変形が行われない。そこで、複数の軌道を扱えるようにするために、HomotopyClassPlanner が使われる。Homotopy Class には以下のようない定義がある。

- (a) 定義 1:(Homotopy) 同じ始点と終点を結ぶ二つの軌道は、一方が障害物と交差することなく他方に連続的に変形できる。
- (b) 定義 2:(Homotopy Class) 互いに Homotopy である全ての軌道の集合

複数の軌道を扱う際、同じ Homotopy class に属する軌道は1つあれば充分であるため、Homotopy class の判定を行って、フィルタリングを行う。

7. 複数の軌道を扱う際、同じ Homotopy class に属する軌道は1つあれば充分であるため、Homotopy class の判定を行って、フィルタリングを行う。

8. H-signature による Homotopy class の判定

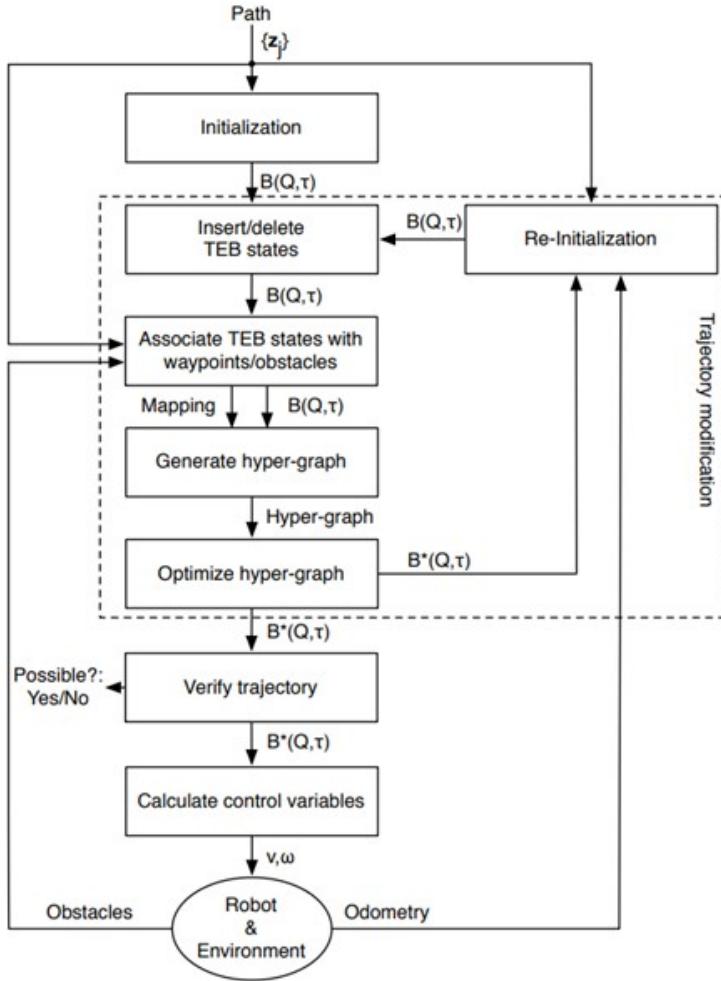


図 4.29: TEB の最適化プロセス

Homotopy class に属する軌跡 τ に対して不变な量を以下のように定義する。

$$\mathcal{H}(\tau) = \int_{\tau} \mathcal{F}(z) dz$$

$$\mathcal{F}(z) = \frac{f_0(z)}{(z - \xi_1)(z - \xi_2) \dots (z - \xi_R)}$$

ここで、 ξ_k は、障害物の代表点である。この不变量 H-signature を数値的に(離散的な軌跡に対して)計算すると、2つの軌跡の間の H-signature を比較することによって、同じ Homotopy class に属するか否かの判定を行うことができる。(実部と虚部でそれぞれ差の絶対値を計算して、閾値より小さければ同じ Homotopy class に属す) 以下の図 4.30 に初期軌道生成から最適化までの処理を示す。

- (a) 頂点のサンプリングを行う。
- (b) スタートの頂点から、ゴールに近づき、かつ線分が障害物と交差しないようにエッジを作成する。さらに、スタートの頂点とエッジで接続された頂点から、

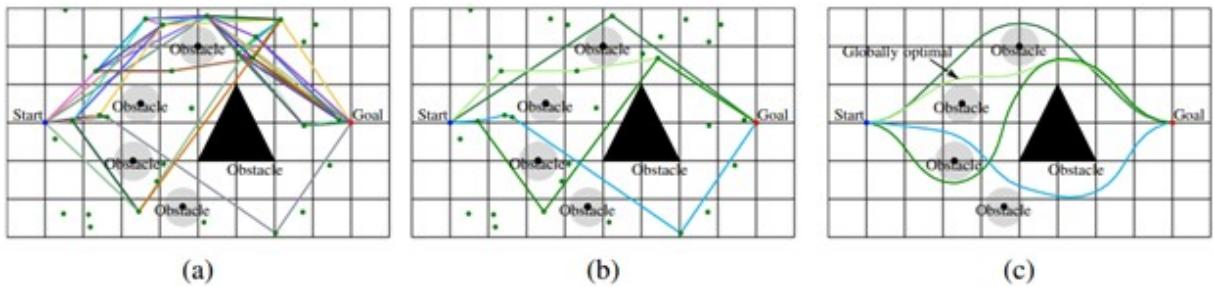


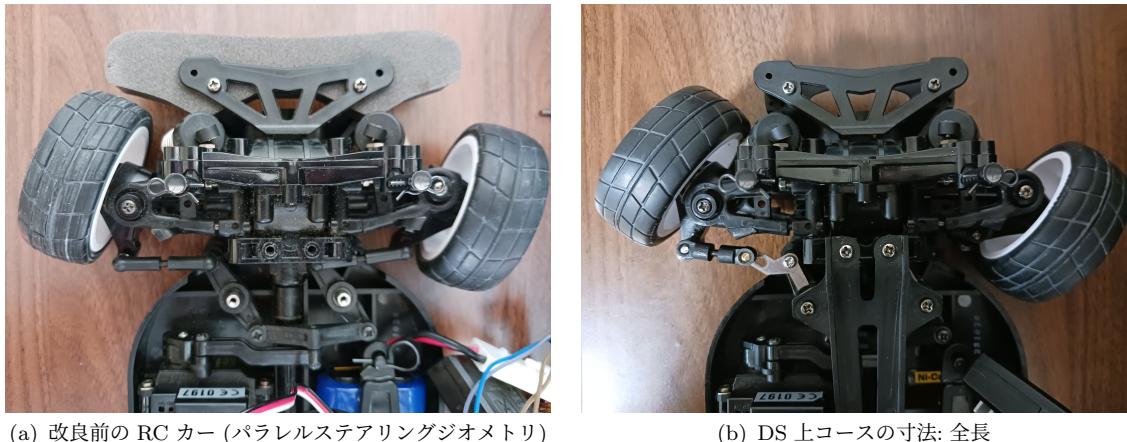
図 4.30: 初期軌道生成から最適化までの処理

ゴールに近づき、かつ線分が障害物と交差しないようにエッジを作成するという処理を繰り返す。(図 (a))

- (c) スタートからゴールまでの各パスに対して、H-signature を計算して、新しい Homotopy class であれば、軌跡と H-signature を保存する。
- (d) 各軌跡を TEB で最適化する。

4.4.5.6 アッカーマンステアリングジオメトリ

Navigaton Stack の軌道計画に与える制約条件を求めるために、本研究で用いるシステムの旋回角度を求める。図 4.32 のようにすることで、サーボモータの制御パルス値と旋回角度(切れ角)の対応値を取得する。本研究の自動運転システムに用いた RC カーのステアリング機構は、図 4.31(a) パラレルステアリングジオメトリである。軌道計画に運動学が簡単なアッカーマンステアリングジオメトリが採用されているため、ステアリング機構を図 4.31(b) に示すアッカーマンステアリングジオメトリへの改造を行った。図 4.33、式 (4.1) にアッカーマンステアリングジオメトリと、その運動学を示す。



(a) 改良前の RC カー (パラレルステアリングジオメトリ)

(b) DS 上コースの寸法: 全長

図 4.31: 本研究で使用した RC カーの改造



図 4.32: 切れ角とサーボモータの制御パルス値の特性取得

$$\phi = \arctan \frac{wheelbase}{r} = \arctan \frac{\omega \cdot wheelbase}{v} \quad (4.1)$$

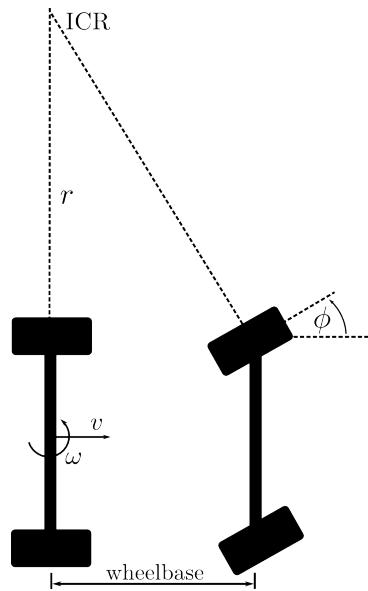


図 4.33: アッカーマンステアリングジオメトリの運動学

アッカーマンステアリングでは、式(4.1)に基づいて、角速度の値からステアリング角度を計算する。ステアリング角度をサーボモータの制御パルス値へと変換した際に取得した特性を図4.34に示す。また、前進・後退の速度値は適当な定数を掛け合わせて、DCモータの制御パルス値に変換しているため、速度の精度は厳密ではない。前進・後退時の速度は前進を0.06 [m/s]、後退を0.05 [m/s]として、DCモータの制御パルスとの特性を一次関数的に近似した。

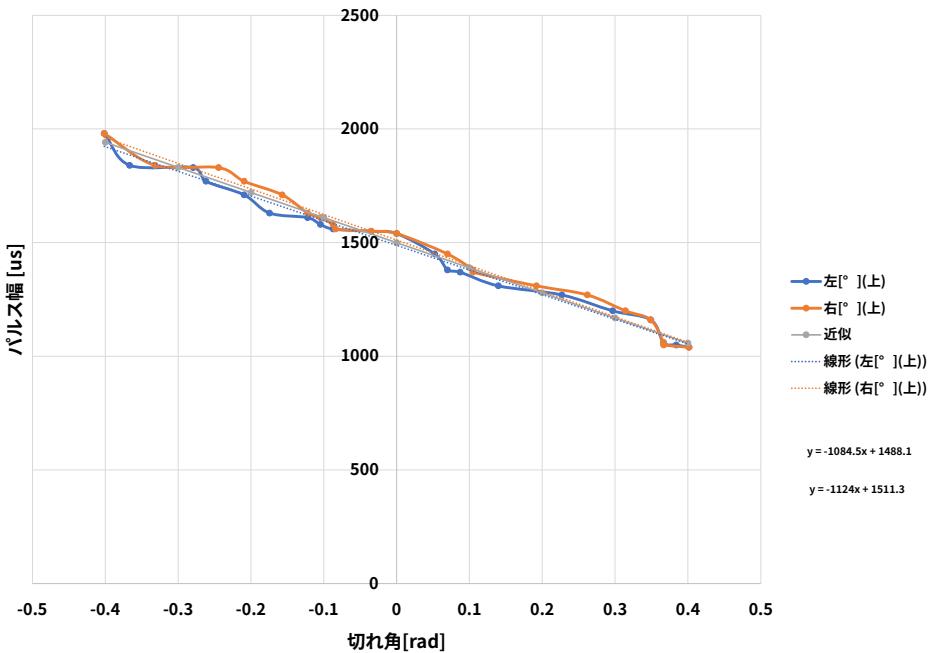


図 4.34: ステアリング角度とモータの制御パルス値へと変換した特性

4.4.5.7 コストマップ

`costmap_2d` パッケージ??は、ロボットが占有格子の形式でナビゲーションを行う場所に関する情報を提供する。コストマップは、制約マップのセンサデータと情報を使用して、`costmap_2d::Costmap2DROS` オブジェクトを介して、空間の障害物に関する情報を保存および更新する。つまり、障害物の判定は“列”単位でのみ行うことができる。たとえば、XY 平面で同じ位置にあるが Z 位置が異なるテーブルと靴の場合 `costmap_2d::Costmap2DROS` オブジェクトのコストマップの対応するセルのコスト値は同じになる。これは、平面空間での経路探索がしやすいように設計されている。

1. マークとクリア

コストマップは、ROS を介してセンサトピックを自動的にサブスクライブし、それに応じて更新させる。各センサは、マーク (障害物情報をコストマップに挿入)、クリア (障害物情報をコストマップから削除)、またはその両方に使用される。マーク操作は、セルのコストを変更するための配列への単なるインデックスである。しかしながらクリア操作は、レポートされる各観測について、センサの原点から外側に向かってグリッドを通過するレイトレーシングで構成される。3 次元構造を使用して障害物情報を保存する場合、各列からの障害物情報は、コストマップに配置されるときに 2 次元に投影される。

2. 占有・空き・未知スペース

コストマップの各セルには 255 の異なるコスト値のいずれかを持つことができるが、使用する基本構造は 3 つの状態のみを表すことができる。具体的に、この構造の各セ

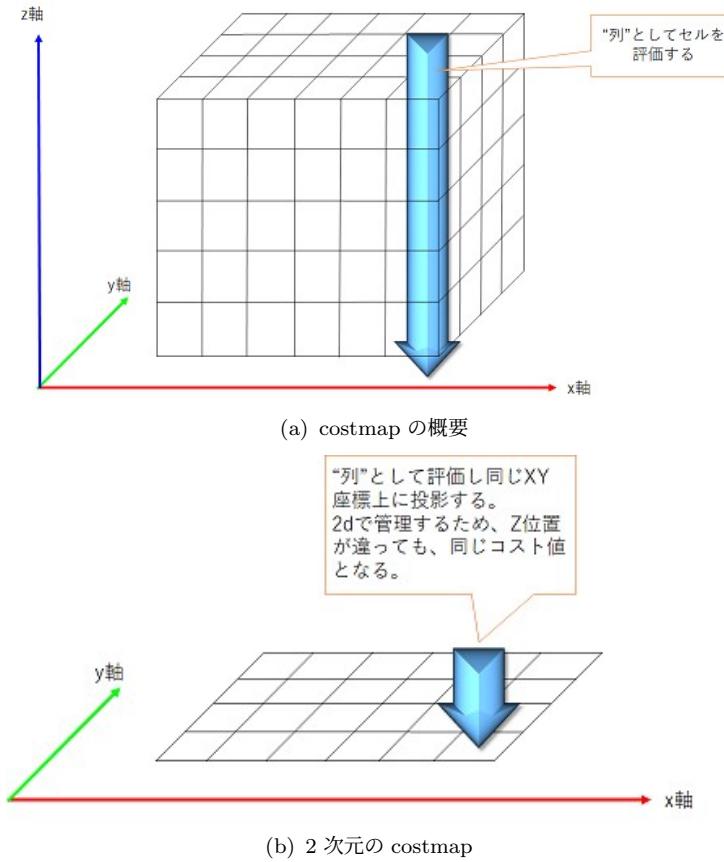


図 4.35: コストマップ

ルは、空き、占有、未知スペースのいずれかである。各ステータスには、コストマップへの投影時に割り当てられる、特別なコスト値がある。一定数の占有セルがある列には `costmap_2d::LETHAL_OBSTACLE` コストが割り当てられ、一定数の未知のセル (`unknown_threshold` パラメータを参照) がある列には `costmap_2d::NO_INFORMATION` コストが割り当てられ、その他の列には `costmap_2d::FREE_SPACE` コストが割り当てられる。

3. マップの更新

コストマップは、`update_frequency` パラメータで指定された周期でマップ更新サイクルを実行する。周期ごとに、センサデータが入力され、マークとクリアの操作がコストマップの基本的な占有構造で実行される。この構造は上記のように適切なコスト値が割り当てられるコストマップに投影される。その後、障害物のインフレーションは `costmap_2d::LETHAL_OBSTACLE` コストを持ったそれぞれのセルで実行される。これは、占有されている各セルからユーザ定義のインフレーション半径までコスト値を外側に伝播することで構成される。このインフレーション処理の詳細を以下に示す。

4. tf

センサソースからコストマップデータを挿入するために、`costmap_2D::Costmap2D`

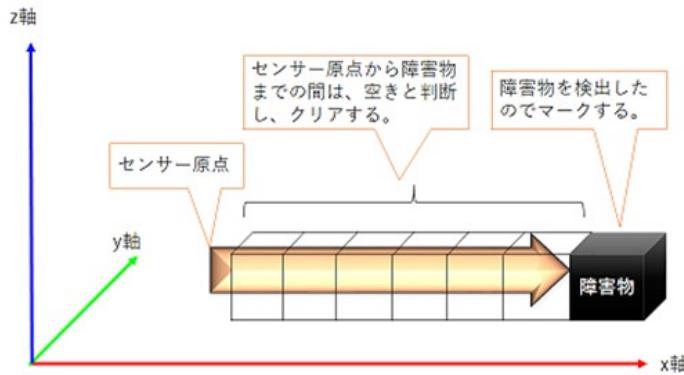


図 4.36: マークのクリア

ROS オブジェクトは、tf を広範囲に使用する。具体的に、global_frame パラメータ、センサーソースが接続され、最新である想定で使用している。transform_tolerance パラメータは、これらの変換間で許容されるレイテンシーの最大量を設定する。tf ツリーがこの予想されるレートで更新されない場合、Navigation Stack はロボットを停止する。

5. インフレーション

インフレーションは、距離とともに減少する占有セルからコスト値を伝播するプロセスである。この目的のために、ロボットに関連するコストマップ値に 5 つの特定のシンボルを定義する。

- (a) 「致命的」コスト (cost_lethal) とは、セルに障害物があることを意味する。そのため、ロボットの中心がそのセルにある場合、ロボットは間違いなく衝突している。
- (b) 「内接」コスト (cost_inscribed) とは、セルと実際の障害物の距離が、ロボットの内接半径より小さいことを意味する。従って、ロボットの中心が内接コスト以上のセル内にある場合、ロボットは確実に障害物と衝突する。
- (c) 「おそらく外接」コスト (cost_possibly_circumscribed) は「内接」に似ているが、ロボットの外接半径を限界距離として扱う。したがって、ロボットの中心がこの値以上のセル内にある場合、障害物と衝突するかどうかはロボットの向きによって決まる。「おそらく」という言葉を用いるのは、実際には、障害物ではなく、ユーザ設定値によりこのコスト値となっているセルの可能性があるためである。例えば、ユーザが、ロボットが建物の特定の領域を回避するように表現したい場合、障害物に関係なく、その領域のコストマップに独自のコスト値を挿入する場合がある。
- (d) 「空き」コスト (freespace cost) はゼロであると想定される。これは、ロボットがそこに行くのを妨げるものが無いことを意味する。
- (e) 「未知」コストは、特定のセルに関連する技術がないことを意味する。コストマップのユーザは適切に解釈することができる。

(f) 他の全てのコストには「致命的」セルからの距離と、ユーザが提供する減衰関数に応じて、「空き」と「おそらく外接」の間の値が割り当てられる。

図 4.37 にコストマップの設定変数を示す。

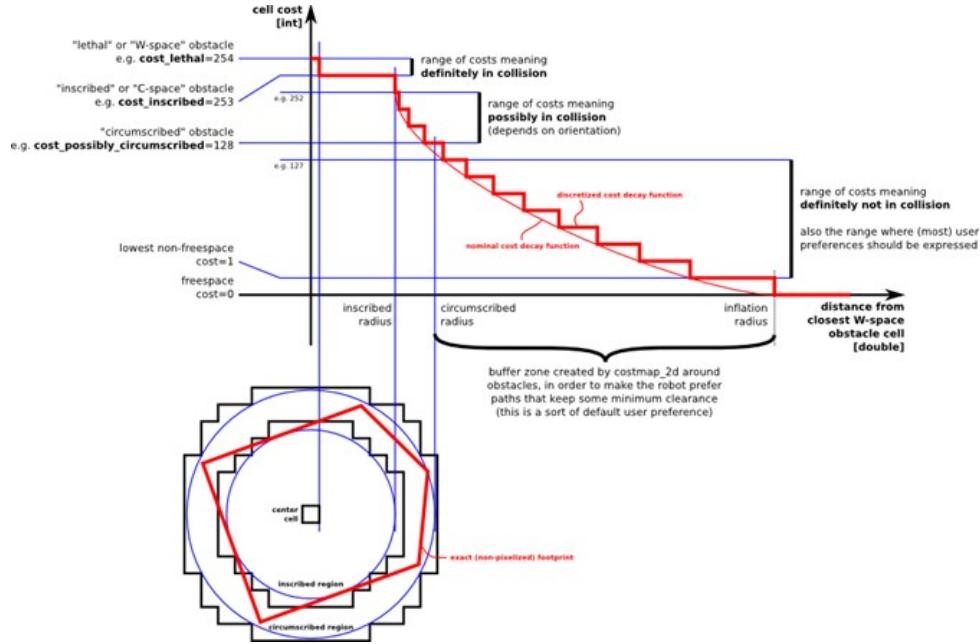


図 4.37: コストマップの設定変数

4.4.5.8 地図の保存と再読み込み

地図の保存には ROS の `map_server`、`map_saver` を使用する。これらにより、LiDAR によって取得した地図をコマンド入力により、`pgm` 形式で保存する。

4.4.6 Waypoint Navigation の実装

4.4.6.1 waypoint の設定・保存・可視化

図 4.38 に、waypoint の保存と可視化を示す。RViz 上で waypoint を設定し、csv 形式で保存する。Navigation を行う際に、RViz 上に地図の読み込みと waypoint の csv データを読み込んでマーカで表示する。これらの機能を python により実装する。

4.4.6.2 自動運転機能

自動運転切り替え機能のノードを図 4.39、図 4.40 に示す。保存した地図と waypoint のデータを読み込む。初期位置から Waypoint までのナビゲーションを実行する。Waypoint Navigation をする上で、次の waypoint へ到着する閾値を設定し、waypoint から半径 1m 以内に到着したら、次の waypoint への Navigation を開始する。最後の waypoint を読み込んだならば、Navigation を終了する。これらを C++ 言語により実装する。

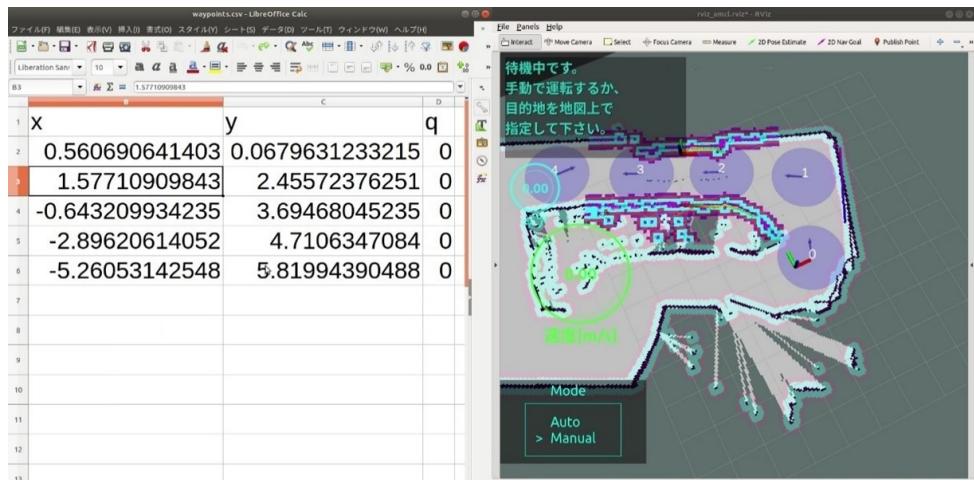


図 4.38: waypoint の保存と可視化

4.4.7 自動運転と手動運転の引継ぎ機能の実装

手動運転中、自動運転に切り替える機能において、手動操作と自動運転でトピックを共通にし、コントローラのボタンを入力することで、手動運転のトピックを発行するノードをシャットダウンし、自動運転の Waypoint Navigation を起動するようにする。また、自動運転から手動運転に切り替える際は、Waypoint Navigation をシャットダウンし、手動運転のノードを起動する。これらを Python により実装する。以上の機能を図 4.41、4.42 に示す。

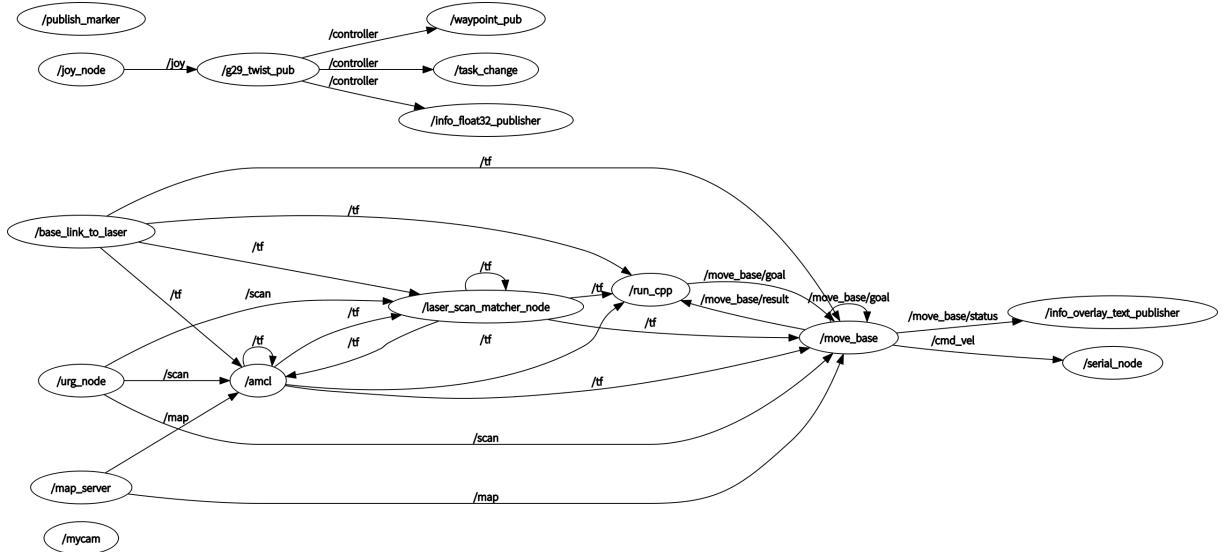


図 4.39: 自動運転切り替え機能のノード (トピック)

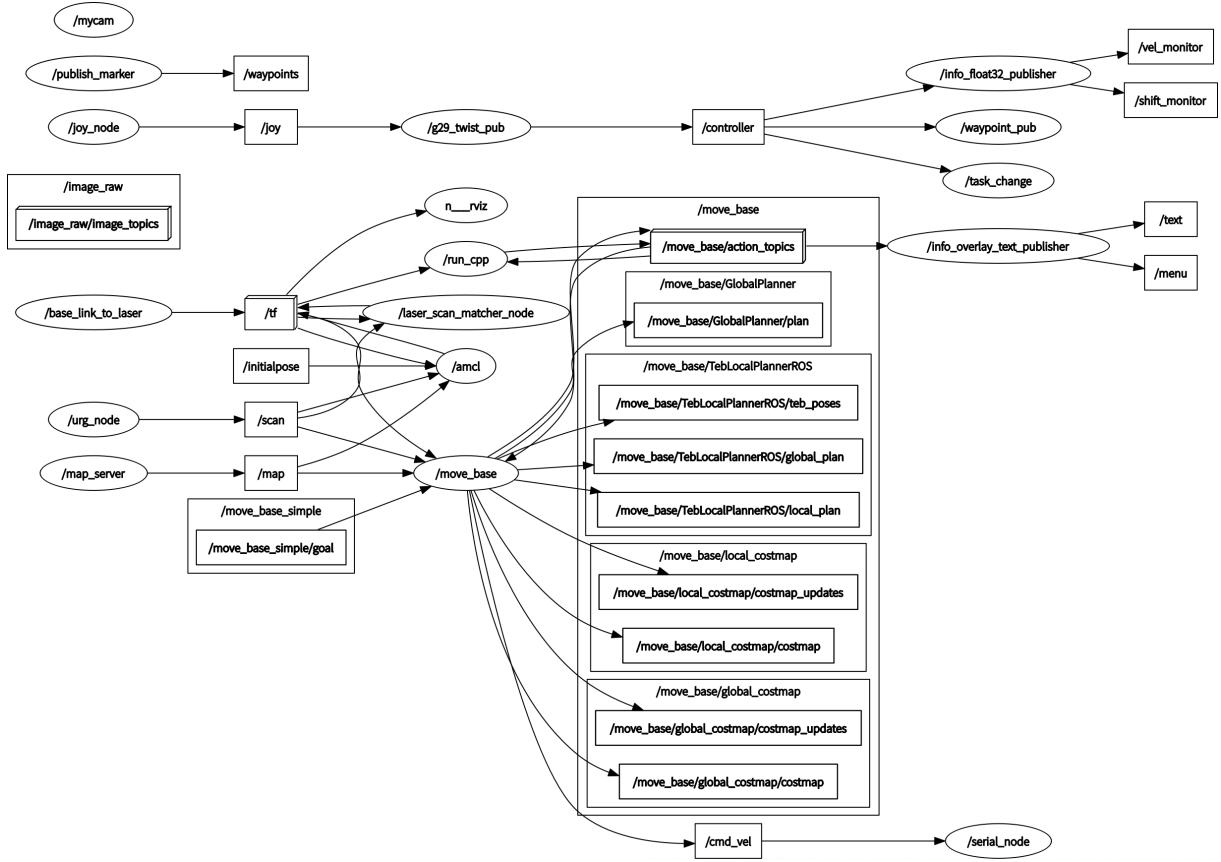


図 4.40: 自動運転機能の全体ノード

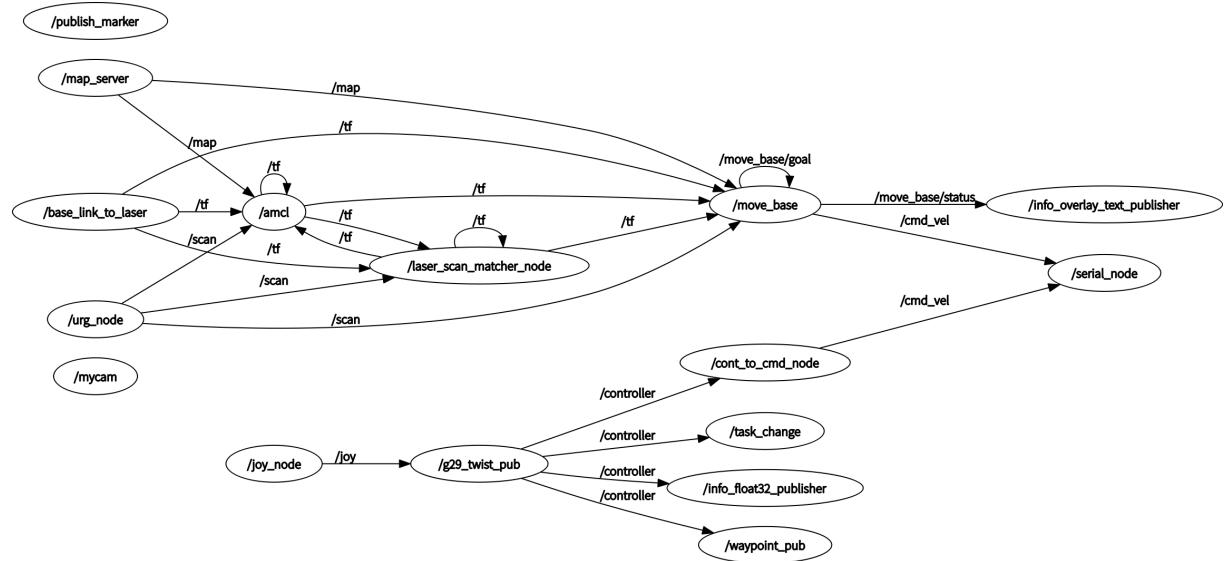


図 4.41: 手動操作時のノード (トピック)

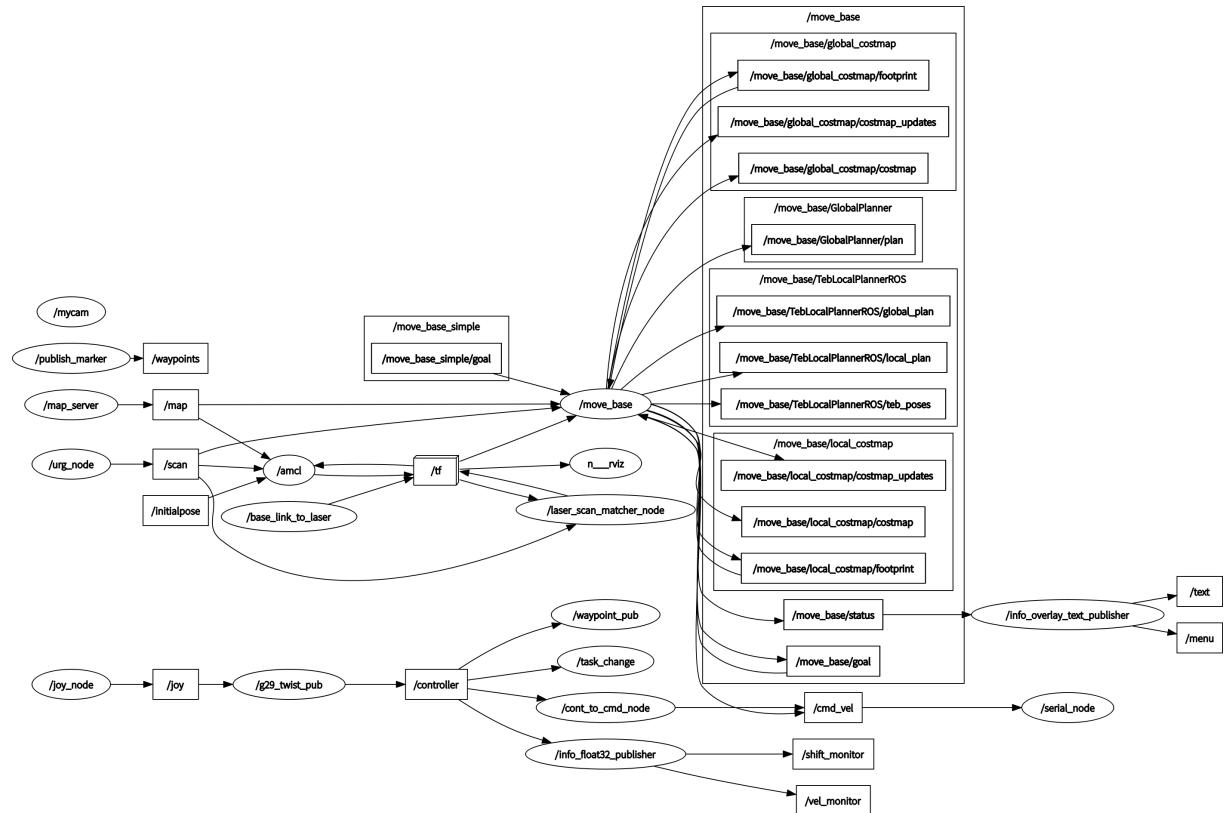


図 4.42: 手動操作時の全体ノード

4.5 検証

本研究では、以下に示す3つの実験を行った。

4.5.1 遠隔操作の操作性の評価

運転免許不所持者も含めた、運転操作のインタラクションの分析を目的とした遠隔運転の実験を行った。図4.43にしめす、仕切りの無い走行コースの遠隔操作の体感情情報の記録に加えて、ブレーキ操作の記録を行った。実験の方法として、操作インターフェース(AT/MT/セミAT)を自由に一つ選択し、コースを走行する。その後、図4.44ブレーキテスト用コースにて、ブレーキテストを実施した。

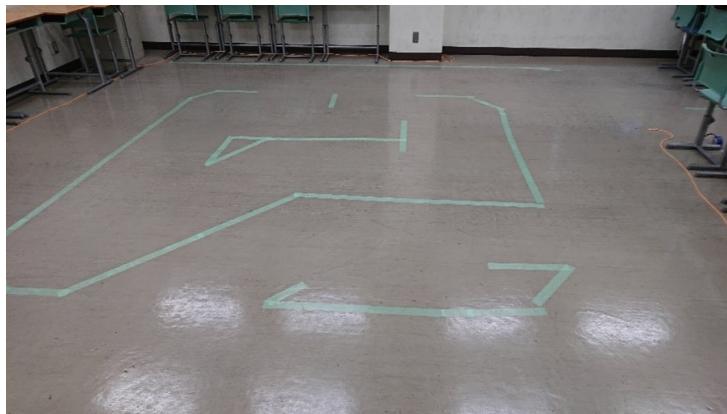


図 4.43: 仕切りの無い走行コース

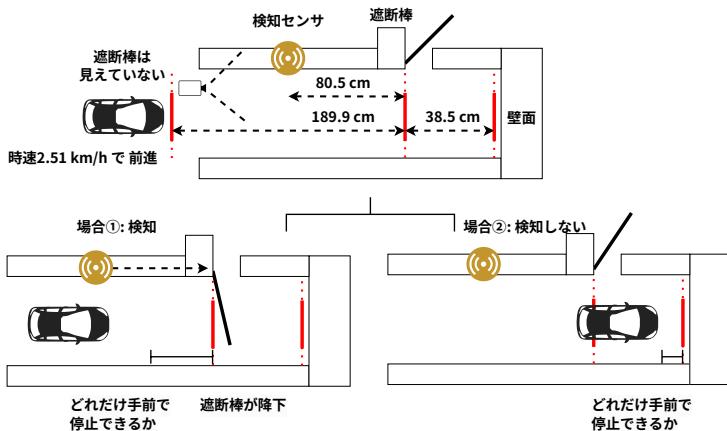


図 4.44: ブレーキテストコース

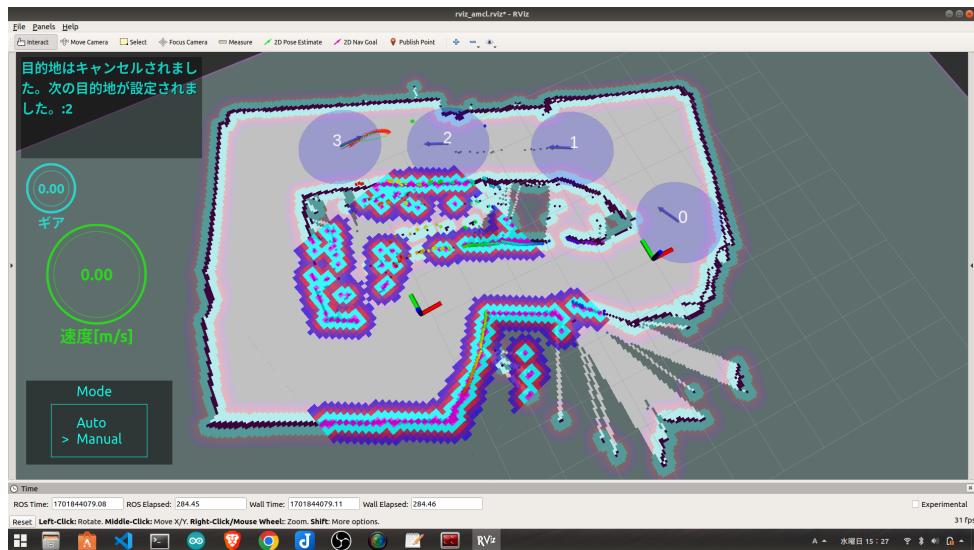


図 4.45: 設定した Waypoint

4.5.2 手動運転から自動運転への切り替え

以下の図 4.45 に取得したコースの占有格子地図と waypoint の情報を基に、手動運転から自動運転への切り替えを行う。具体的には、手動運転から自動運転、それから最後に手動運転に切り替える走行実験を行った。あらかじめ SLAM により地図を取得して、waypoint が設定されている地図上を AMCL による自己位置推定を行いながら、手動操作を行う。地図上では、あらかじめ手動から自動に切り替える地点(マップ中の番号 0)と、自動から手動に切り替える地点(マップ中の番号 3)が明示され、所定の位置に移動した際に、ステアリングコントローラのモード切り替えスイッチにより、自動運転・手動運転を切り替える操作をして、コースを周回して、目的値へと操作を行う。

4.5.3 方向転換機能の評価

以下の図 4.46(a) に示すコースにおいて、手動操作によって、初期位置から目的値に向けて手動操作を行う。次に図 4.47 のように、自動運転モードで初期位置から目的地に向けて自律移動を行う。走行のパターンは、図 4.46(b)、4.46(c) に示す 2 種類で行う。

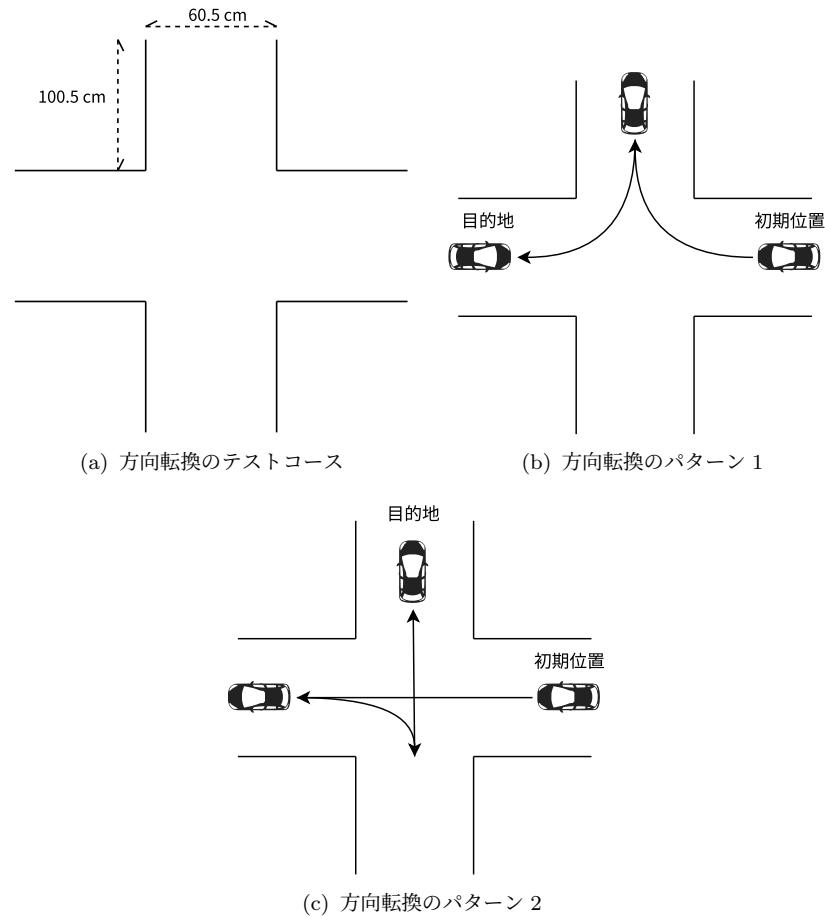


図 4.46: 方向転換のテストコース

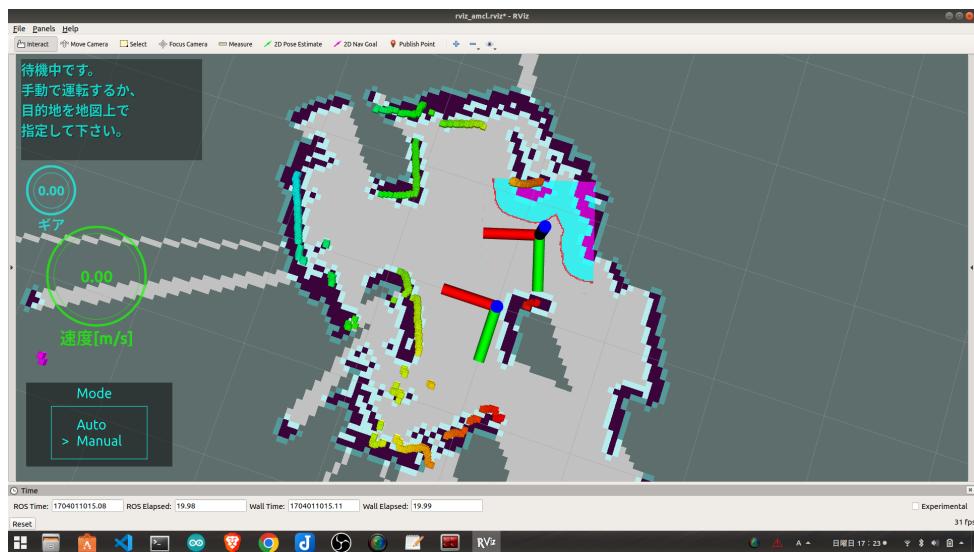


図 4.47: 方向転換の自律移動画面

4.5.4 結果

4.5.4.1 遠隔操作の操作性の評価

図 4.48(a)～4.49(d) に操作性評価に関する実験結果を示す。図 4.48(a) の項目に注目すると、被験者が体感した操作が AT、MT、セミ AT の内いずれが多いかを示す。アンケートの回答者は 46 人で、高専祭での一般の来場者に向けたものである。ブレーキテストは、被験者の内 33 人がアンケートに回答した。比較的簡単な AT 操作を選択した被験者が 67% を占め、その内 79% が操作を難しいと回答した。また、回答者のうち、若年層(未就学児・小中学生)の被験者が多い。(全体の 6～7 割)

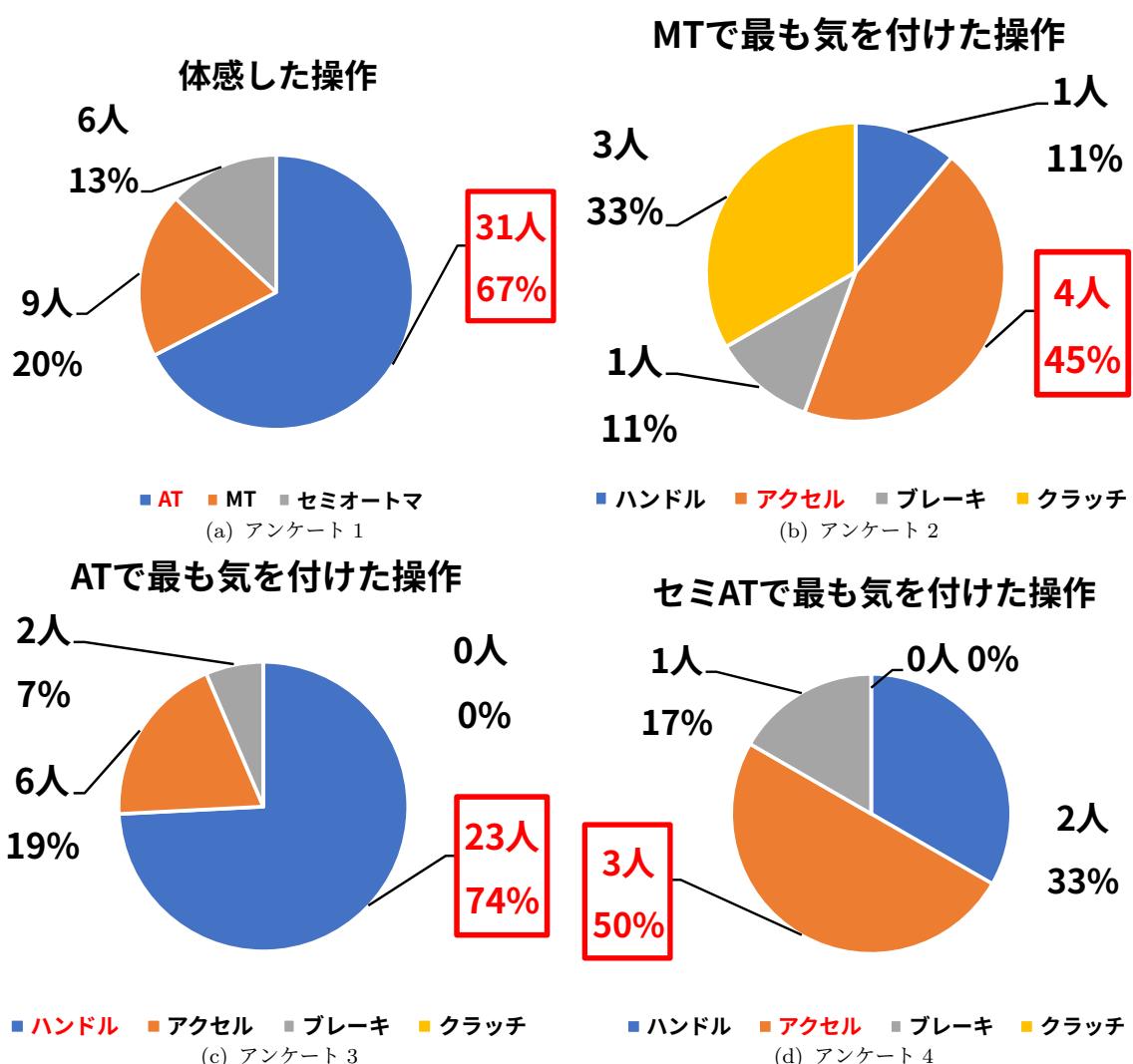


図 4.48: アンケート

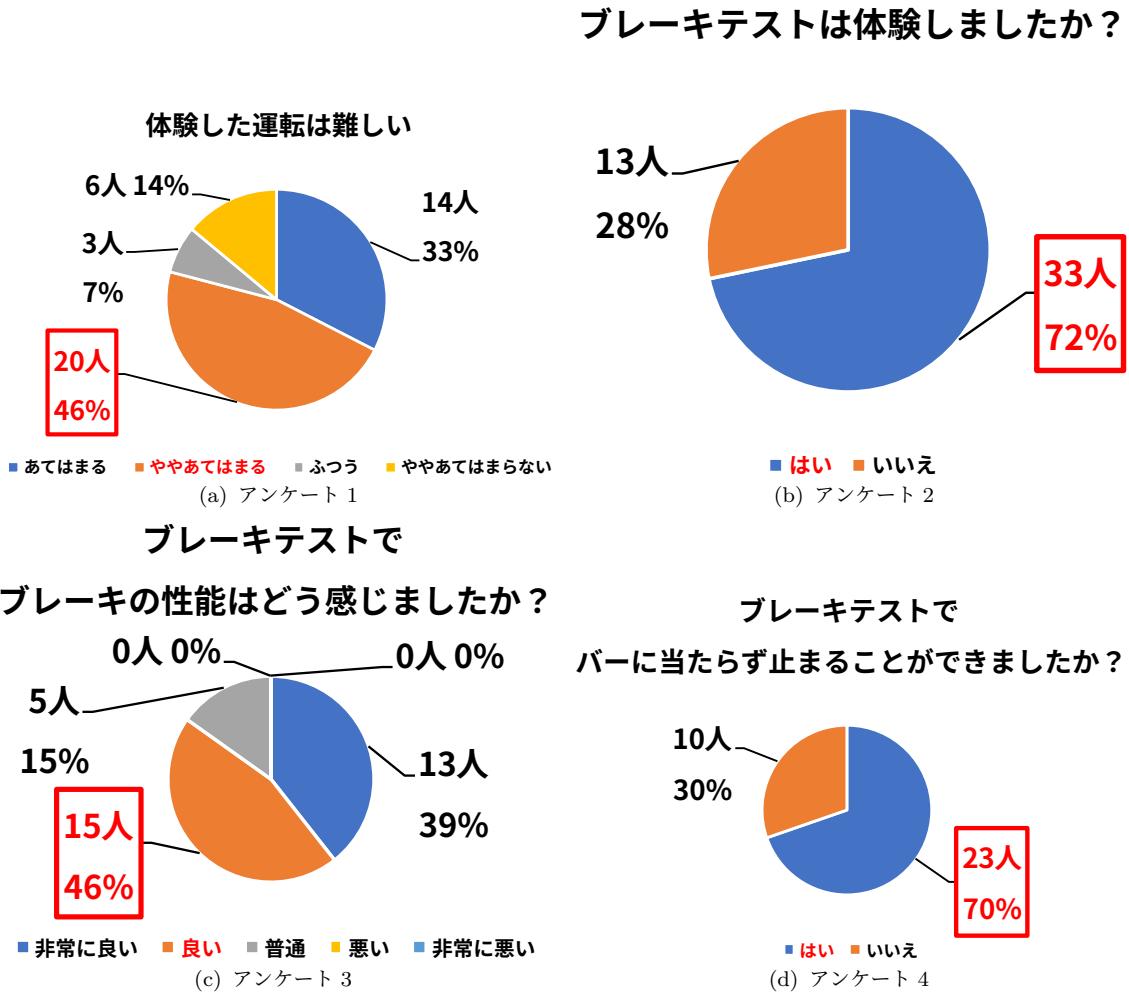


図 4.49: アンケート

4.5.4.2 手動運転から自動運転への切り替え

手動運転から自動運転への切り替えの実験結果について述べる。手動運転と自動運転の双方への切り替えは問題なく実施できた。しかし、自律移動の際に、後退状態になると、平坦な直線の後、自己位置が並進方向にずれて見失う現象であるセンサ退化が起こった。これは、直線が平坦であるため、LiDAR のスキャンデータが、地図のどの位置にでもマッチングするためである。前進状態の場合にも起こるが、自動で補正される場合もあった。しかし、後退時には必ず生じた。

4.5.4.3 方向転換機能

図 4.51 に方向転換機能の評価の結果を示す。図 4.46(b)、4.46(c) に示す通りに移動を行うと自己位置が破綻し、目的地まで到達できない結果であった。具体的には、後退のまま一度に目的地への移動を試みると、最適な経路が得られない。または、経路の移動中に障害物付近に到達することで停止する。そこで、以下の図 4.51(a)、4.51(b) のように、自律移動のための経路の分割数を増やし、経路を直線的にすること、さらに、切り返しの分割

静止後の障害物との距離

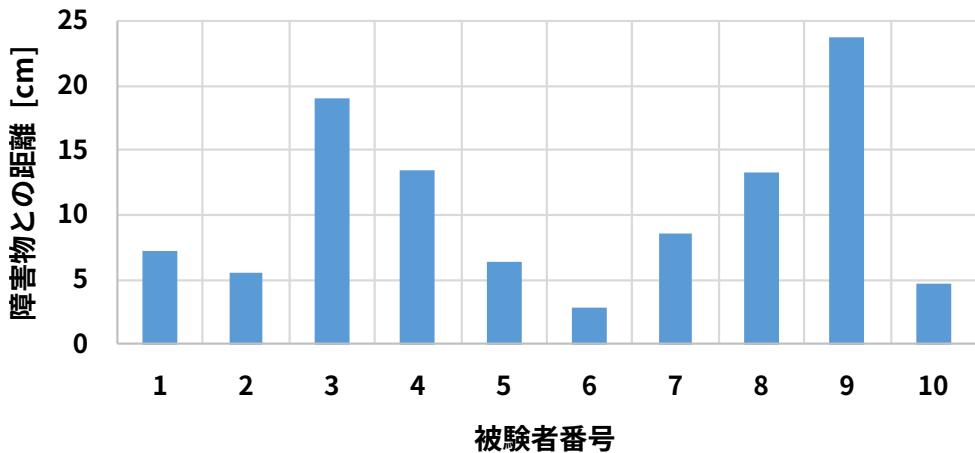


図 4.50: ブレーキテストの結果

の際に、転換の方向からやや斜めの姿勢として、一度に切り返す量を少なくすることで、最終目的値への移動を可能とした。自律移動シミュレーションでは、切り返しの量を減らす必要がなく、一度で切り返しを行うことができるが、実機では、異なる結果が得られている。

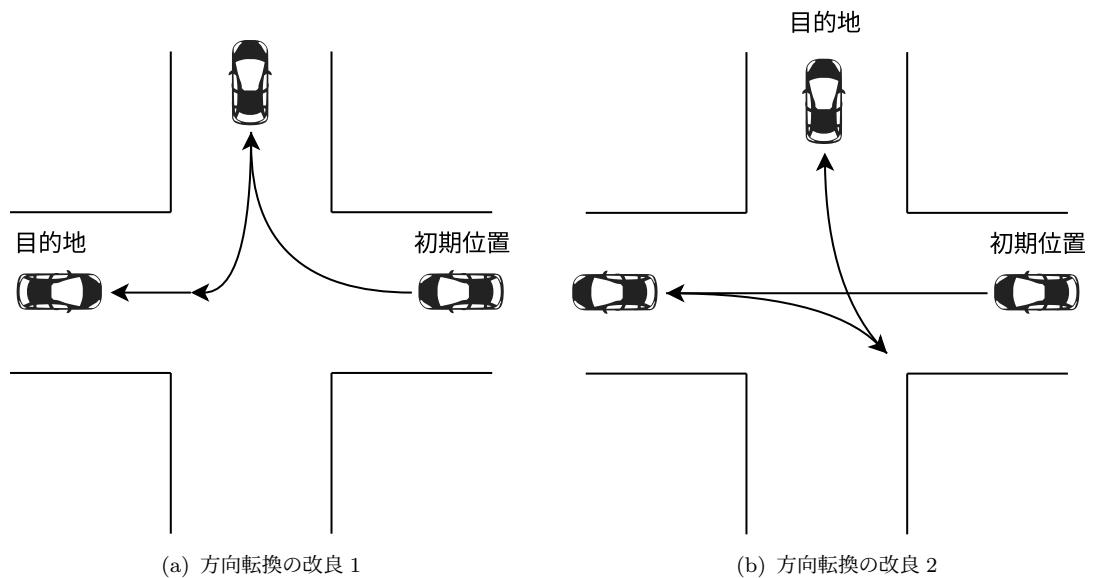


図 4.51: 方向転換のパターン改良

4.6 考察

4.6.1 遠隔操作の操作性の評価

若年層の6~7割を占め、AT操作を選択し、同時にハンドル操作に注力したことから、ハンドル操作に慣れていないことが分かる。特に、若年層の操作を観察すると、カウンター操作(曲がる方向にハンドルを切った後に元に戻す操作)ができないことが明らかになった。一方、大人や、AT操作と比べて比較的操作の難易度が高い、MT/セミAT操作の選択者は、難なくハンドルを操作した。このことから、操作経験者や熟練者にとっては、実車よりも操作が簡単であることが推測される。

4.6.2 手動運転から自動運転への切り替え

手動運転から自動運転に切り替える際の待ち時間が1秒程度であり、その間停止する。手動から自動への切り替えは機能した。今回は問題なく機能したが、手動操作の間隔には遅延が生じたため、応答速度が速い場合に影響が生じる可能性があると考えられる。

4.6.3 方向転換機能の評価

LiDARのみのSLAMで作成した地図は、歪んでいたが、特徴量に矛盾が無かったため、LiDARのレーザオドメトリとのスキャンマッチングが上手く機能することで、自己位置を見失いにくく、自律移動が実施できたと考えられる。切り返しの分割の際に、転換の方向からやや斜めの姿勢として、一度に切り返す量を少なくすることで、最終目的地への移動を可能とした。これは、実際の人による車両操作に近く、最も衝突のリスクが少ない方法と考えられる。このことから、LiDARのみのSLAMで作成した環境地図は、精度が低くとも、特徴量に矛盾が無ければ、目的地への自律移動は、中継地点を複数設定することで実現されることが明らかになった。また、自律移動シミュレーションでは、切り返しの量を減らす必要がなく、一度の目的地設定により、最適な切り返しを行うことができるが、実機では、目的地を複数設定し、切り返しの頻度を増やす必要がある点については、地図に含まれる測定誤差や、LiDARの測定誤差等による影響で、自己位置推定に影響を受けているためだと考えられる。

4.7 おわりに

本研究では、RCカーの遠隔操作と自律移動を切り替える機能を有する、自動運転システムを開発し、その使用性を評価した。遠隔操作は、ステアリングコントローラのハンドルの左右の入力による、旋回機能、アクセルペダルの押下による前進機能、シフトレバーによるトランスミッションのモード切り替えによる後退機能と、ブレーキペダルの押下による減速機能を実現した。また、AT操作のみではなく、MT操作やセミオートマ機能を搭載し、実車と同等の操作感覚で操作を行うことができるシステムを実現した。自

動運転システムは、LiDAR のみによる SLAM と、Navigation Stack を用いた Waypoint Navigation を実現した。手動運転・自動運転の切り替え機能は、遠隔操作と、自律移動のタスクの切り替えをステアリングコントローラから行うことで実現した。使用性の評価において、遠隔操作においては、個々の運転技能に応じたインターフェースによって、実機でのインタラクションを実現し、若年層に向けたステアリング操作の補正が必要であることが明らかになった。また、自動運転機能を用いた手動・運転引継ぎの実験では、問題なく引継ぎが行えることが明らかになった。また、従来の車型自律移動システムでは、検証されなかった、後退機能に関する切り返し動作の評価を行った。その結果、一度に目的地に到達するために、切り返す回数を少なくするのではなく、切り返す回数を増やすことで、LiDAR のみでのナビゲーションが可能であることが明らかになった。自律移動の際に、特徴量の少ない経路を通る際に、LiDAR のみによるオドメトリではセンサの退化により、自己位置を見失う現象が生じたが、特徴量が多い点であれば、スキャンマッチングが機能したことから、特徴量の多い地図ならば、LiDAR のみによる SLAM はある程度可能であることが明らかになった。

本研究では、屋内環境での実験であり、屋外での利用を想定していない。具体的には、屋外では、地図が膨大になるため、複数の地図を用意する必要があり、自律移動の度に、切り替える機能が必要となる。今後は、屋外自律移動を目的として、地図の切り替え機能の実装を検討する。

第5章

結論

本研究は、遠隔型自動運転システムのユーザビリティに関するシミュレーションによる遠隔型自動運転における体感速度のモデルの提案と、LiDARのみのSLAMによる自律移動システムとして、手動・自動運転引継ぎシステムの実機開発と、開発したシステムの評価をまとめたものである。図5.1、5.2に本論文で提案した手動・自動運転引継ぎシステムとその構成要素の概略、体感速度モデルを示し、本論文の内容を総括する。

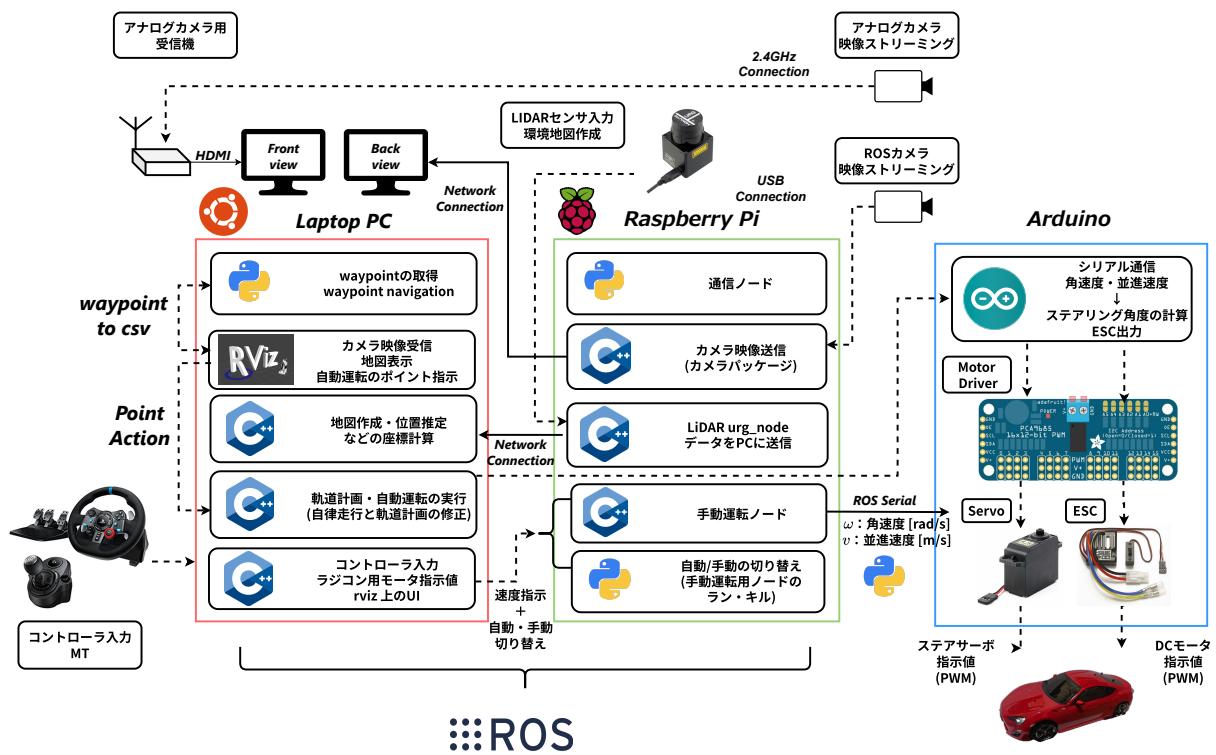


図 5.1: 手動・自動運転引継ぎシステム

第1章「序論」では研究背景として近年の自動運転システム・自律移動ロボットの研究開発を俯瞰し、本研究が目指す遠隔型自動運転システムのユーザビリティの立ち位置、自律移動ロボットにおける実証実験の必要性、自律移動ロボットの開発方針と、著者自身の研究の位置づけ・目標設定について述べた。

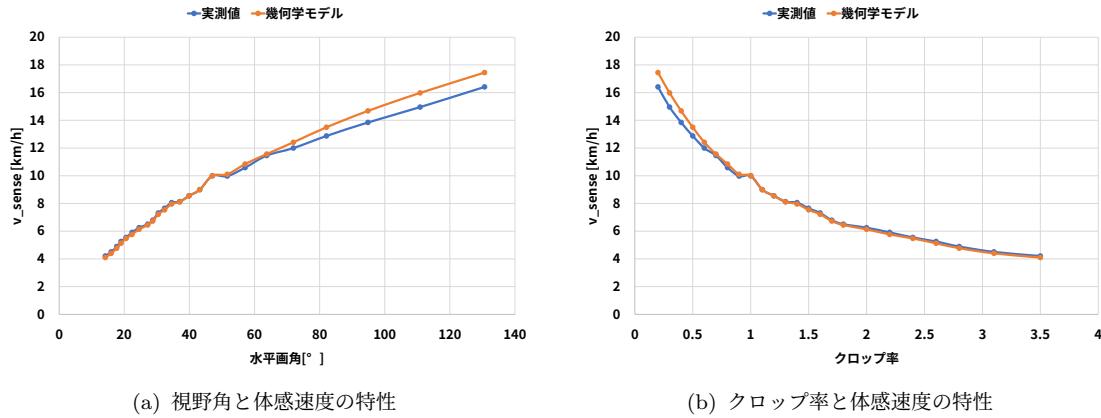


図 5.2: 体感速度モデル

第2章「FPV車両操作の体感速度変化率の定式化」では、体感速度の関連研究において、定性的な体感速度の変化について議論されている一方、体感速度の定量化がなされていない点を取り上げ、独自に、遠隔型自動運転システムのFPV操作に着目した、自動車の映像における運転視野角や、映像画角の変化による体感速度のモデルの定量化を目的として、体感速度モデルの定式化を提案した。成果として、体感速度はおよそ視野角の根に比例する特性に近く、クロップ率に対して単調に減少する特性であるモデルを提案した。

第3章「LiDARのみのSLAMで作成された環境地図の評価」では、ROSに実装されたSLAMメタパッケージである、GmappingとHector SLAMとCartographerによる環境地図構築システムを開発し、LiDARのみによるSLAMによって作成された自律移動用環境地図の精度を比較した。成果として、CartographerによるSLAMが最も歪みが少なく、追ってGmapping、Hector SLAMの順に精度が高いことが明らかになった。しかし、Gmappingがノイズに強い点で、以降の自動運転システムに採用した。

第4章「自動運転システムの開発」では、ステアリングコントローラによる、ハンドルフットペダル型インターフェースによる遠隔操作の手動運転のユーザビリティの分析と、手動運転自動運転の切り替えシステムの開発を行った。ROS(Robot Operating System)を用いた遠隔操作による手動運転と、SLAMとWaypoint Navigationを実装した手動運転・自動運転の切り替えシステムを開発した。第3章で得た知見として、SLAMパッケージとして、Gmappingを採用し環境地図構築とNavigationメタパッケージである、Navigation Stackを用いて、4輪車両型RCカーの自律移動システムを実現した。Waypoint Navigationにより、中継目的地を経由して、最終目的地まで自律移動を行い、最終目的地に到着した時点で、手動運転に切り替えることで、運転操作の引継ぎを実現するシステムを実現した。成果として、一つ目に、若年層の手動操作におけるステアリングのカウンター操作の補助の必要性が明らかになった。二つ目に、LiDARのみの自律移動ロボットシステムにおいて、手動・自動運転の切り替え操作が可能であった。三つ目に、自律移動による切り返し操作が目標地点を分割することによって可能であり、目標地点を1点のみの指定で切り返しに成功したシミュレーションとは異なる結果であることが明らかになった。

本研究の特徴は、遠隔型自動運転システムの体感速度に関する特性を体系化している点

と、シミュレーションと実機による実験評価を実施し、シミュレーションと実機の両方による実証実験の必要性を提言している点である。

従来の研究では、定性的な体感速度の効果に関する記述と、非常に高価な内界センサを用いたものである。対して、我々の研究では、自律移動に関しては、LiDARのみと、1/10スケールのRCカーを用い、比較的小規模で、安価でハードウェアの制約がある中で、シミュレーションと実機の両者を比較して新たな手法へと発展させている研究は非常に少ない。

本研究は、遠隔型自動運転における体感速度のモデル化と、安価なハードウェアにおける自律移動の妥当性を検証するための方法を提案することで、移動ロボットの能力向上に大きく貢献したと信ずる。

謝辞

私の研究のモチベーションは、中之島ロボットチャレンジを観戦することで、将来的に自律移動ロボットや LiDAR の開発に携わりたいと考えたことです。今回は、縁あって北陽電機株式会社様より LiDAR を貸与頂き、自らの開発アイデアを研究活動で実現することができました。本研究は、LiDAR の使用技術の勉強並びに、北陽電機株式会社様への就職を目的として取り組みました。北陽電機株式会社 R&D 室 嶋地様、和田様、木元様、岡本様には大変お世話になりました。北陽電機株式会社様への報告を以て、専攻科の研究を総括とさせて頂きます。

第A章

付録があるときは

プログラム文とかを書いてページ数を稼ぎたいときは、以下のようにしてみます。

```
#include <iostream>
using namespace std;
int main() {
for(int i = 1; i <= 5; i++) {
cout << "こんにちは、C++ の世界! " << i << endl;
}
return 0;
}
```

\usepackage{ascmac}して screen 環境を使うと、枠がつきます。

```
#include <iostream>
using namespace std;
int main() {
for(int i = 1; i <= 5; i++) {
cout << "こんにちは、C++ の世界! " << i << endl;
}
return 0;
}
```

参考文献

- [1] 淩田樹生, 藤本雄一郎, 澤邊太志, 神原誠之と加藤博一, 「自動車運転者の適切な速度制御を促す拡張現実感」, 研究報告コンピュータビジョンとイメージメディア (CVIM) , vol. 2023-CVIM-232, no. 13, pp. 1-6, 1月 2023.
- [2] 大前学, 「自動車の遠隔操縦における視覚情報の運転への影響評価と影響補償の為の車両制御の開発」, 科学研究費補助金研究成果報告書, 2010, 参照: 2023年3月26日. [Online]. Available at: <https://koara.lib.keio.ac.jp/xoonips>
- [3] speed management. [Online]. Available at: <https://www.itf-oecd.org/sites/default/files/docs/06speed.pdf>
- [4] 東井隼斗, 北原格, 龜田能成と大田友一, 「ドライバの体感速度変化を促すバーチャルパターン」, 電子情報通信学会論文誌 D, vol. J99-D, no. 1, pp. 45-55, Jan 2016.
- [5] V. Gitelman, F. Pesahov, R. Carmel, 「Speed perception by drivers as dependent on urban street design; a case-study」, Transactions on Transport Sciences, vol. 11, pp. 5-18, Sep 2020.
- [6] D. Jo, S. Lee, Y. Lee, 「The Effect of Driving Speed on Driver's Visual Attention: Experimental Investigation」, Engineering Psychology and Cognitive Ergonomics, D. Harris, Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 174-182.
- [7] G. Grisetti, C. Stachniss, W. Burgard, Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters, IEEE Transactions on Robotics, vol. 23, no. 1, pp. 34-46, Feb 2007.
- [8] W. Hess, D. Kohler, H. Rapp, D. Andor, Real-time loop closure in 2D LIDAR SLAM, 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden: IEEE, Mar 2016, pp. 1271-1278.
- [9] 増田健, 「ICP アルゴリズム」, 研究報告コンピュータビジョンとイメージメディア (CVIM) , vol. 2009-CVIM-168, no. 23, pp. 1-8, Aug 2009.
- [10] S. Rusinkiewicz, M. Levoy, 「Efficient variants of the ICP algorithm」, Proceedings Third International Conference on 3-D Digital Imaging and Modeling, Mar 2001, pp. 145-152.

[11] 点云配准方法 PLICP CSDN 博客. 参照: 2023 年 12 月 27 日. [Online]. Available at:
https://blog.csdn.net/weixin_40863346/article/details/102731212