修士論文

倒立振子を搭載した自律移動ロボットの経路計画に向けた

ダイクストラ法と A*アルゴリズムの比較検討

Comparison of Dijkstra's algorithm and A* algorithm
for path planning of an autonomous mobile robot
with the cart-pole control

報　告　者

学籍番号:　　1245060

氏名:　　西 祐希

指　導　教　員

星野 孝総 准教授

令和４年２月18日

高知工科大学大学院工学研究科

基盤工学専攻電子・光工学コース

目 次

第1章　序論

1.1.　研究背景

近年、人の日常生活支援を目的としたロボットの研究開発が盛んに進められ、ロボットの活動範囲は広がりを見せている。特に、人にサービスを提供するロボットは、安全で親近感のある自律移動機能が求められている。ロボットが自律移動するためには自己位置認識、空間認識が必要不可欠な要素となる。一般には、エンコーダなどの内界センサや、LiDAR、カメラなどの外界センサを搭載することになる。

移動ロボット工学の分野では、自律移動ロボットにおいて、ロボットが任意の環境で安全に動き回ることを可能にする技術が研究されており、その主な問題の 1 つとして経路計画が挙げられる。経路計画は、グローバル計画とローカル計画に分けられる。ローカル計画は、途中でロボットの視界に現れたり、事前にマッピングしている場所以外の障害物を回避するためにロボットに使用される。一方、グローバル計画は、地図情報のみに基づいて最適な経路を計算する。本研究では、こちらのグローバル計画を実験の対象とし、実験を進めていく。

倒立振子は、それ自体が不安定なシステムである。この種のシステムの安定性を保ちつつ、目的地に向かって走行させることは、制御アルゴリズムやコントローラの構成などの面から、ロボット工学において、興味深いテーマである。従来より、山藤、松本らはこの問題について精力的な研究を行い、移動型倒立振子の直立姿勢安定化および平行走行を実現している [1][2]。

倒立振子を搭載した移動体を制御することは、車輪の揺れや動きをコントロールすることであり、高度な無人搬送車の制御や自動車の乗り心地の向上にもつながる実用的な応用の可能性が考えられる。また、経路計画を組み合わせることで、任意の環境でさらなる安全性を追求することが可能であると考えた。

これらを実現するためのロボットのシステムを Robot Operating System(ROS) を用いて作成する。ROS とは、オープンソースで提供される、ロボット向けのメタ・オペレーティングシステムであり、世界中の開発者が生み出したプログラムやノウハウが蓄積されており、産業用ロボットだけでなく、サービスロボットや研究用ロボットにも利用されている。例えば、パナソニック社が開発したトマト収穫の産業用ロボットに、ROS が搭載されている。センサーや AI を使ってトマトの色や形を認識し、熟したトマトを傷つけないように収穫できるロボットである。作業員よりも動作は遅いものの、ロボットは昼夜問わず可動できるため、生産量はほとんど落とさずに人件費の最適化が行われた。また、SONY 社が 1999 年に販売を開始し、2018 年に大幅なリニューアルをして話題を呼んだペットロボット『aibo』にも ROS 搭載されている。

人感センサーで人を識別したり、タッチセンサーがなでられているか、叩かれているのかを判断した上で、行動に移したりしており、そうした情報を処理し、どのような動作を行うのか演算しているのが ROS である。

ROS (Robot Operating System) には、自己位置推定、SLAM (Simultaneous Localization and Mapping)、経路計画，障害物回避など、移動ロボットのナビゲーションに必要な機能を実装したソフトウェアパッケージが一通り揃っており、広く利用されている。たとえばロボットを屋外で自律走行させる実験プロジェクトであるつくばチャレンジでは、2017 年大会において、参加チームの 2/3 以上が ROS を利用したことが油田によって報告されている [3]。また、RoboCup の一部門である@Home リーグでは、2016 年大会の出場 24 チーム中 23 チームが ROS を利用したことが出村らによって報告されている [4]。移動ロボットのための膨大なソフトウェアすべてを自前で実装することは現実的ではない。そのため、ROS を通じた既存のソフトウェアの利用は、これらのコミュニティーにおいては半ば常識化している。本稿では、移動型倒立振子にナビゲーション機能を実装していく。

本稿で想定する OS、ROS のバージョンはそれぞれ、Ubuntu 16.04、Kinetic Kame である。また、ROS のビルドシステムとして catkin の使用を想定している。

1.2. 研究目的

本研究は、ロボットオペレーティングシステム（ROS）のフレームワークを使用して、差動ロボットでの動作をよりよく理解することにより、倒立振子を搭載した自律移動ロボットにおいて、グローバル軌道計画の問題の解決策である 2 つの手法、ダイクストラ法および A *アルゴリズムの比較研究を行うことを目的としている。本稿の構成は次の通りである。第 2 章では、本研究を行うために使用した開発環境である ROS とそのデバッグツールの概要について述べる。そして、第 3 章では倒立振子システムの数式モデル化や制御方法、それらを用いて行う MATLAB/simulink や Gazebo でのシミュレーションとその結果について述べる。第 4 章では、経路計画を行う際に使用したロボットや 2 つのグローバル経路計画の内容、経路計画に必要な SLAM、そして Gazebo でのシミュレーションとその結果について述べる。第 5 章では、倒立振子を搭載した自律移動ロボットのシステムについて述べる。最後に第 6 章では本稿についてのまとめを述べる。そして、今後の研究課題と展望について述べる。

第2章　開発環境

2.1.　ROS

ROS についての概要を述べる。ROS とは、オープンソースで提供される、ロボット向けのメタ・オペレーティングシステムであり、複雑なロボットの動作を作成するタスクを容易にするなど、ソフトウェア開発者のロボット・アプリケーション作成を支援するライブラリとツールを提供している。

ROS が広まるまでは、メーカーや研究者が独自のロボットシステムを開発していたため、開発ノウハウが一般化せず、技術的なブレイクスルーが生まれないことや開発コストの高騰などが課題となっていた。

ROS の目標は、ロボットソフトウェアの共同開発を世界的に進めることであり、多くのロボットソフトウェアプラットフォームの中でも、ロボットの研究、開発で使用されるソースコードの再利用性を重視している。そのため、ROS は以下のような特徴を持っている。

・プログラムの再利用性
・コンポーネント化されたプログラムと通信
・豊富な開発ツール
・活発なコミュニティー活動
・エコシステムの構築

現在は、全世界に公開されている ROS のおかげで、過去に開発された技術やノウハウを誰でも活用できるようになり、ロボットシステムの開発や進歩速度が速くなっている。

2.2.　Turtlebot

TurtleBot は、ROS の標準のロボットプラットフォームである。Turtlebot は ROS をはじめて学ぶ人にもわかりやすいように設計されており、現在でも ROS 開発初心者や学生などで最も使用されているものである。そのため、外部コンポーネントを追加することなく、複数のタスクを作成して実行できる。例えば、TurtleBot を使用してリアルタイムで自律的にナビゲートしながら、同時に自己位置推定および地図作成を行うアルゴリズム（SLAM）を実行して環境のマップを作成することができる [5]。本研究において、シミュレーション用に選択したロボットは、TurtleBot3-Burger である。

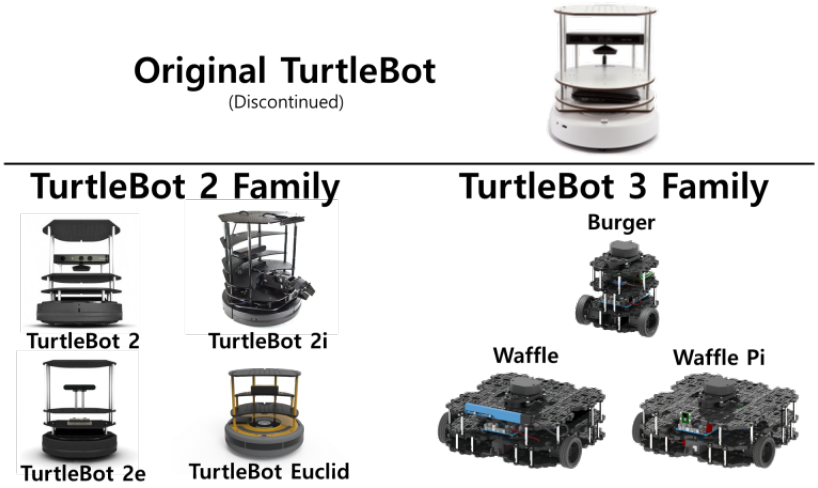

図 2.1: 歴代の Turtlebot

2.3. Rviz

Rviz（ROS 視覚化）は、センサーデータと ROS 状態情報を表示することを主な目的とする 3 次元ビューアーである。 Rviz を使用すると、現在の仮想ロボットモデルの構成を表示できる。他の多くのアプリケーションの中でも、カメラデータ、赤外線距離測定、ソナーデータ、レーザーセンサー測定など、ROS トピックに関するセンサー値のライブ表現を表示することもできる [6]。 以下の図 2.2 に、Rviz の GUI 画面の例を示す。

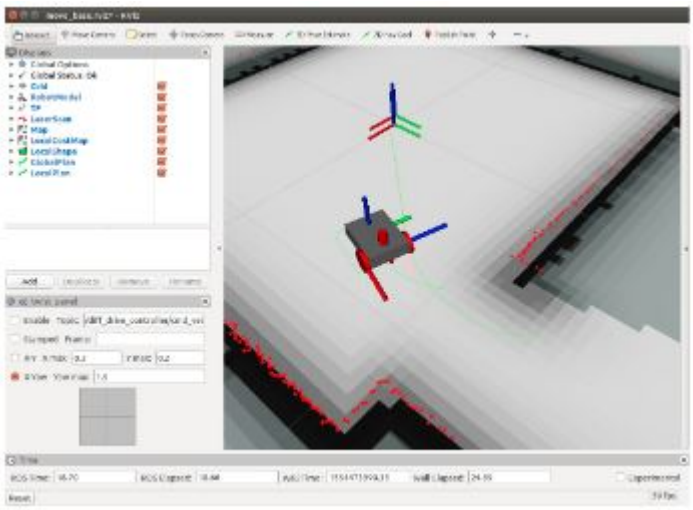

図 2.2: RViz の GUI

2.4. Gazebo

Gazebo は、物理エンジンを搭載し、接触や力のバランスなどの動力学的な変化を伴う実世界を再現し、そこにロボットやセンサなどのモデルを置くことで 3 次元でのシミュレーションを実行するオープンソースのロボットシミュレーターである。デフォルトでは、物理エンジンに Open Dynamics Enigine(ODE)[7] が使用されているが、バージョン 3.0 以降は、使用用途に応じて、他に Bullet、Simbody、DART などの物理演算方式を選択することができる。ODE は、Smith.R らがオープンソースで開発しているフリーの動力学計算エンジンであり、Gazebo のデフォルトの物理エンジンということもあり、多くの研究で採用されているため、本研究でも、デフォルトの ODE を使用するように設定した。

ロボットシミュレーションは、ロボット工学の分野で働きたい人にとって基本的な要素である。Gazebo はロボット分野におけるオープンソースシミュレータとして高い評価を受け、米国で開催された DARPA Robotics Challenge の公式シミュレータに採用されたことをきっかけに、さらなる高性能化が図られている。現在、ROS の開発と普及の中心となっている OpenRobotics が Gazebo の開発も行っているため、ROS との連携も強力なものになっている。

適切に設計されたシミュレーターを使用すると、アルゴリズムのテストやロボットの設計などを行うことができ、RViz と比べて、ロボットの衝突、センサの計測値（エンコーダ、IMU センサ）、カメラによる画像の取得などを行うことができる。 Gazebo ソフトウェアを使用したのは、ROS システムとの互換性に加えて、複雑な屋内および屋外環境で TurtleBot3 を含むさまざまなロボットを妥当な精度と効率でシミュレートできるためである。 また、高品質のグラフィックスと便利なグラフィカルおよびプログラムインターフェイスが含まれている [8]。今回は、バージョン 8.0 を使用している。

以下の図 2.3 に、Gazebo の GUI 画面の例を示す。

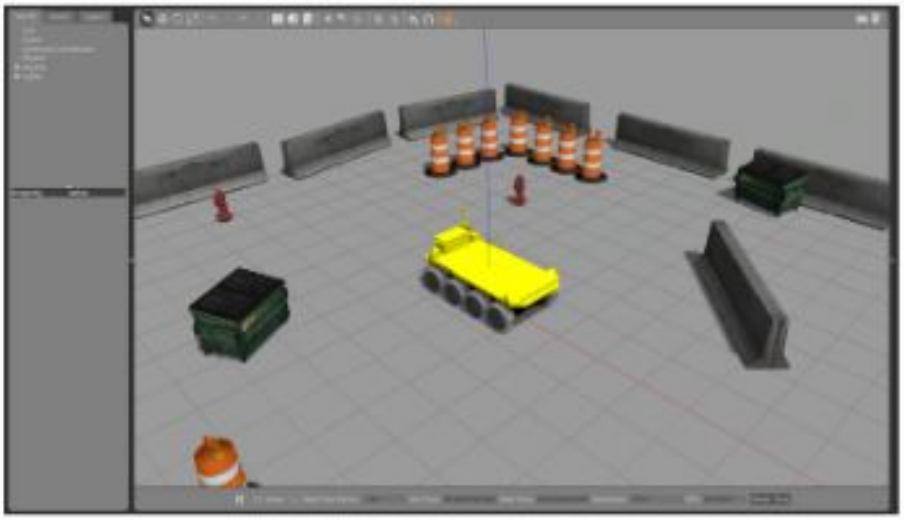

図 2.3: Gazebo の GUI

第3章　倒立振子の線形制御

3.1.　設計した倒立振子システム

本章では、今回使用する倒立振子モデルとその制御方法について述べる。倒立振子とは、支点よりも重心が高い位置にある振り子のことであり、台車には、振子の支点が取り付けられている。直線のレール上をカートが移動することにより、カート上の振子を倒さずに鉛直にすることを目的としている。なお、振子は抵抗なく回転できるものとする。倒立振子は、代表的な教材であり、非線形で不安定な制御対象として、ソフトコンピューティングなど各種制御系設計法の有効性を検証するためによく用いられている。図 3.1 は、倒立振子システムの概念図である。

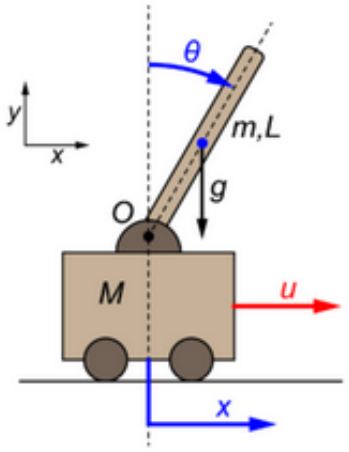

図 3.1: 倒立振子システム

また、そのパラメーターは表 3.1 で定義されている。

x は台車の位置、θ は振り子の傾き、u は台車にかかる力を表している。また、振り子の重心は振り子の中心に位置するものとする。振子の長さ L、振子の重さ m は後述する実験の条件により変更するパラメータとなるため詳細は後述する。

表 3.1: 倒立振子における各パラメータ

記号	パラメータ	値
g	重力	$9.81m/s^2$
M	台車の重さ	$1.0kg$

3.2. 状態方程式の導出

次式のラグランジュの運動方程式より運動方程式を導出する。

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial}{\partial \dot{q}}KE\right)+\frac{\partial}{\partial q}PE=F \tag{3.1}$$

KE は運動エネルギー、PE は位置エネルギー、F は外力を表している。倒立振子のモデルでは、変数に台車の変位 x と振り子の傾き θ が含まれているため、ラグランジュ方程式の一般式に含まれる変数 q は

$$q=\{x,\theta\} \tag{3.2}$$

となる。
また、外力 F については、台車にかかる力 u が存在し、振り子には力がかかっていないため

$$F=\{u,0\} \tag{3.3}$$

となる。
よって、倒立振子の数学モデルは、次の非線形方程式で表される。

$$\begin{cases} (M+m)\,\ddot{x}+\frac{1}{2}mL\ddot{\theta}\cos\theta-\frac{1}{2}mL\dot{\theta}^2\sin\theta=u \\ \frac{1}{2}mL\ddot{x}\cos\theta-\frac{1}{2}mL\dot{x}\dot{\theta}\sin\theta+\frac{1}{3}mL^2\ddot{\theta}-\frac{1}{2}mgL\sin\theta=0 \end{cases} \tag{3.4}$$

また、この非線形方程式に対して、線形化を行う。振り子の傾き θ が十分に小さい場合は、次のような近似をすることができる。

$$\begin{cases} \cos\theta\approx 1 \\ \sin\theta\approx\theta \\ \dot{\theta}^2\approx 0 \\ \theta\dot{\theta}\approx 0 \end{cases} \tag{3.5}$$

よって、平衡点近傍で線形近似して展開すると次式が導出される。

$$\begin{cases} (M+m)\,\ddot{x}+\frac{1}{2}mL\ddot{\theta}=u \\ \frac{1}{2}mL\ddot{x}+\frac{1}{3}mL^2\ddot{\theta}-\frac{1}{2}mgL\theta=0 \end{cases} \tag{3.6}$$

式3.6から、状態ベクトルxと入力uを含む線形状態方程式を求めると次式で表すことができる。

$$\dot{x} = Ax + Bu \tag{3.7}$$

$$\begin{aligned} A &= \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & a_1 & 0 \\ 0 & 0 & a_2 & 0 \end{pmatrix} \\ B &= \begin{pmatrix} 0 \\ 0 \\ b_1 \\ b_2 \end{pmatrix} \\ x &= \left(x, \theta, \dot{x}, \dot{\theta}\right)^T \\ a_1 &= -\frac{3mg}{4M+m} \\ a_2 &= \frac{6\left(M+m\right)g}{4ML+mL} \\ b_1 &= \frac{4}{4M+m} \\ b_2 &= -\frac{6}{4ML+mL} \end{aligned} \tag{3.8}$$

3.3. コントローラ設計

3.3.1. 状態フィードバック制御

制御方法としては、状態フィードバック制御を用いる。状態フィードバック制御とは、現在の系の状態量をフィードバックし、制御量の値と目標値との比較を行い、それらを一致させるように訂正動作を行う制御であり、閉ループ制御とも呼ばれる。フィードバックというのは閉ループを形成して出力側の信号を入力側に戻すことである。制御すべき系の対象の特性が完全に分かっていなくても、少々特性が変化しても、所期の目的を達することができ、外乱の影響を打消すことができるなど開ループ制御に比べすぐれた特長を有している。ここで、本研究で用いるフィードバック制御のブロック図を図 3.2 に示す。

状態フィードバック制御を行うため、コントローラーは次式で表される。

$$\begin{aligned} u &= -Kx \\ K &= [k_1, k_2, k_3, k_4] \end{aligned} \tag{3.9}$$

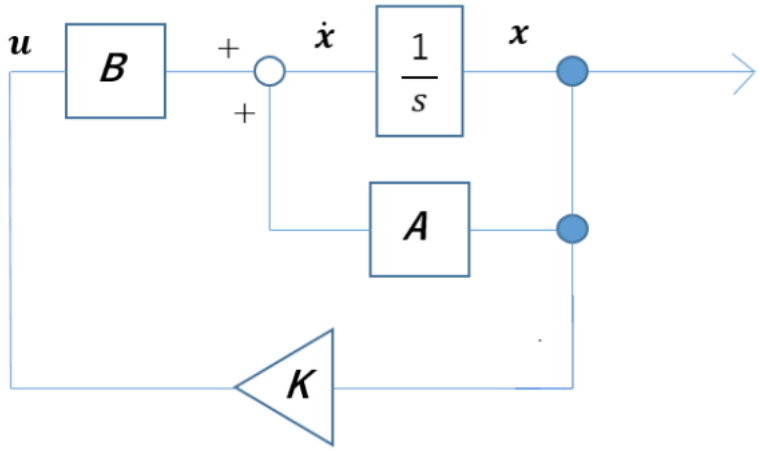

図 3.2: フィードバック制御のブロック図

これは、次のステップで台車にかける力 u を現在の状態 x とゲイン K で求めている。また、状態フィードバックを用いた状態方程式は次式のように書き換えることができる。

$$\begin{aligned} \dot{x} &= Ax + Bu && (3.10)\\ &= (A - BK)\,x && (3.11)\\ &= Cx && (3.12) \end{aligned}$$

このシステムに対して、本研究では、極配置法を用いて制御の方針を決める。

3.3.2. 極配置法

制御対象の応答特性に強く関係しているのが極である。制御対象の系が可制御である場合、状態フィードバック制御を施すことによって、制御対象のもつすべての極を任意の場所に配置できる。そこで、希望する応答から極を特定し、制御した結果の系（閉ループ系）がそこに極をもつように状態フィードバックゲイン K を決定する。

極配置法は、線形システム $\dot{x} = Ax$ の安定性が、システム行列 A の固有値に依存し、その固有値の実部が負であれば、システムが安定することを利用したものであり、適切な固有値 λ を決定することで、状態フィードバックゲイン K を求める方法である。極配置法の指針としては、固有値 λ を決めるときは、実部の絶対値を大きな値に取れば、目標値への収束を早くすることができるが、その分、必要な入力が大きくなることを考慮する必要がある。次の MATLAB/simulink シミュレーションは、極の配置を決めるために行うものである。

3.4. MATLAB/simulink シミュレーション

3.4.1. MATLAB

MATLAB[9] は MathWork 社が開発している、データ解析、モデリング、シミュレーション、可視化などの数値解析、アルゴリズムの開発に必要な統合開発環境を提供するアプリケーションソフトウェアである。さらに、制御系設計、信号処理、画像処理など各種分野に対して利便性の高い数値演算ライブラリが Toolbox と呼ばれる MATLAB を拡張するパッケージの形で提供されている。また、プログラミング言語として、効率的な数値解析プログラム開発を行う事も可能である。

MATLAB の特徴を以下に簡単に述べる。

・データアクセスの容易性データの基本構造は行列であり、変数の型宣言や配列の大きさを宣言する必要がない。また行列を直接演算する事ができ、行列の一要素ごと演算する必要がない。

・対話型操作環境とプログラミング機能統合開発環境により、コマンド実行、ファイル操作およびヘルプの参照などのオペレーションを対話形式で行える。また、MATLAB はインタプリタ型式の言語であるため、コンパイル、リンクといったプログラムの作成および変更から実行するまでの手間が少ない。

・豊富な数値計算関数とグラフィックス関数 MATLAB のみで 800 以上の関数、コマンドが提供されている。拡張パッケージである Toolbox により、さらに関数、コマンドが提供されている。また、豊富なグラフィックス関数を用いることにより、容易に計算結果を可視化することができる。

3.4.2. simulink

Simulink[10] はシステムをグラフィカルに表現するために、ダイナミックシステムの GUI（Graphical User Interface）ベースのシミュレーションツールとして開発された MATLAB の Add-on ソフトウェアの１つである。Simulink モデルはブロック線図を用いてシステムの動作をあらわす。各ブロックは加算、ゲインといった基本演算から、微積分、PID 制御といったように内部状態を有する複雑な演算まで多様な種類があり、また、Data Store とよばれるようなメモリ機能を持つブロックもある。これらのブロックを線でつないでシステムの動作を表現し、離散系・連続系、その混在系のダイナミックシミュレーションモデルの構築からシミュレーションの実行まで行うことが可能である。

3.4.3. simulink での線形制御器設計

今回設計した倒立振子システムを MATLAB/simulink 上でシミュレーションを行うためのブロック図を図 3.4 に示す。これは、フィードバック制御を用いた状態方程式を表している。ま

た、ROS でのロボット制御のことを考慮し、離散化を行っている。

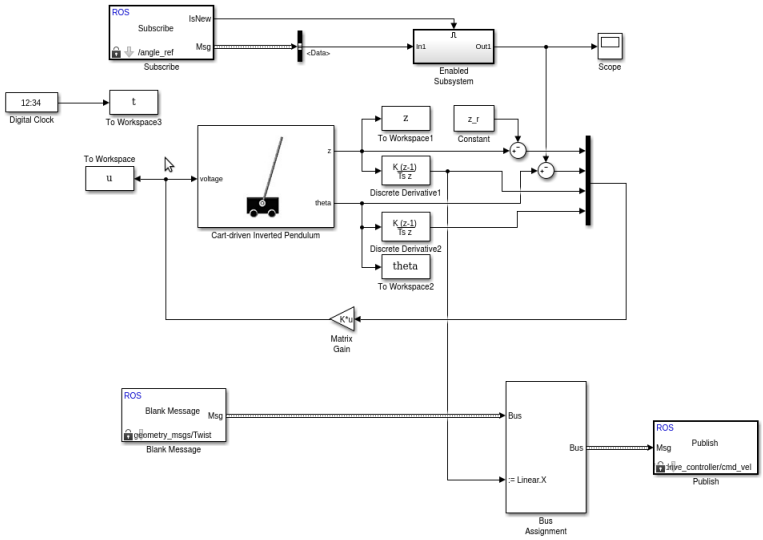

図 3.3: simulink のブロック図

離散化の計算は、微分が

$$y\left(t_n\right) = K\left(\frac{u\left(t_n\right) - u\left(t_{n-1}\right)}{T_s}\right) \tag{3.13}$$

積分が前進オイラー法を用いた

$$y\left(t_n\right) = y\left(t_n - 1\right) + K * T_s * u\left(t_{n-1}\right) \tag{3.14}$$

となっている。これらの式において、$u(t_n)$ と $y(t_n)$ はそれぞれ現在のタイムステップのブロックの入力と出力、K はゲイン、T_s はシミュレーションの離散ステップサイズを表している。

今回 MATLAB/simulink を用いたシミュレーションを行う目的は倒立振子をより安定に制御できるような極配置を決めることである。また、極配置法の指針により決めた固有値からゲインを求めるために、MATLAB の place のアルゴリズムを用いた [11]。これは、システム $\dot{x} = Ax + Bu$ と自己共役閉ループ極配置ベクトル p において、状態フィードバック $u = -Kx$ が位置 p に閉ループの極を配置するようにゲイン行列 K を計算するものである。つまり、今回は、極配置指針を考慮した任意の p を決めることにより、ゲイン K とその結果を得ることができ、振り子の傾きや入力の大きさの時系列的な変化を見比べ、どの極配置がより安定して制御できるのかの判定を行う。

3.4.4. simulink でのシミュレーション結果

シミュレーションとしては、倒立振子極の配置を表 3.2 に示す 3 つのものを試し、一番安定するものを調べる。倒立振子が垂直に立っている状態から、x 軸方向に 5[m] 進んだ場所を目標地点として、倒立振子が垂直で立った状態を目指すようにシミュレーションをおこなった結果を図 3.4 示す。

表 3.2: 極のパラメータ

色	パラメータ
赤	$\lambda = [-0.5 - 1.0 - 1.5 - 2.0]$
緑	$\lambda = [-1.0 - 1.5 - 2.0 - 2.5]$
青	$\lambda = [-1.5 - 2.0 - 2.5 - 3.0]$

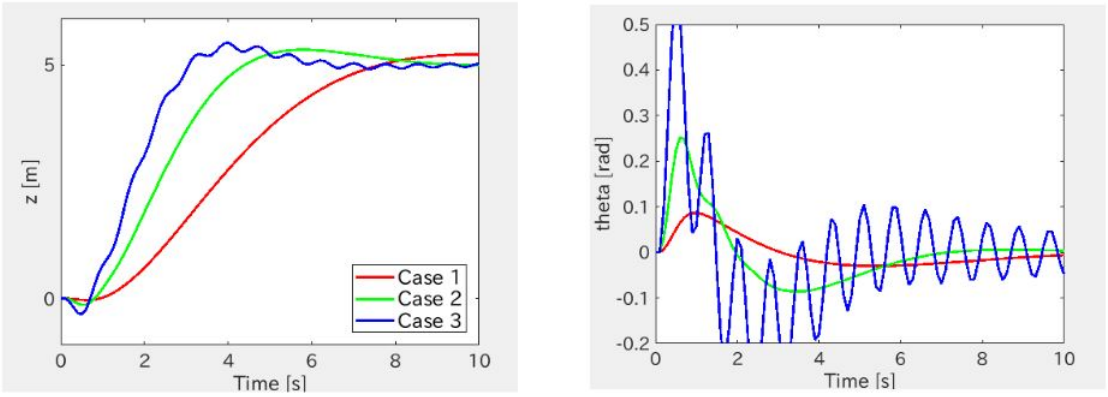

図 3.4: simulink シミュレーション結果

表 3.2 で示されているように、赤、緑、青の順に極の実数部分の値を大きくなるように極を配置し、シミュレーションを行った結果が図 3.4 である。図 3.4 のグラフでは、極の実数部分の値が大きい青線においては、目標地点に速く収束する代わりに、倒立振子の傾きの変動が激しく、逆に、極の実数部分の値が小さい赤線においては、目標地点に収束するのが遅くなる代わりに、倒立振子の傾きの変動は少なく、比較的安定した制御を行うことできていることが分かった。つまり、局配置法の指針でも述べたように、実部の絶対値を大きな値に取れば、目標への収束を早くすることができるが、その分、必要な入力が大きくなることがグラフからも分かる。本研究では、目標への収束時間と、安定性のバランスを考え、今後の実験において、図 3.4 のグラフ中の緑線の結果となる極配置を選択し、そこから算出される状態フィードバックゲインを用いた倒立振子の制御を行うようにした。

3.5. Gazeboシミュレーション

Gazebo は、物理エンジンを搭載し、接触や力のバランスなどの動力学的な変化を伴う実世界を再現し、そこにロボットやセンサなどのモデルを置くことで 3 次元でのシミュレーションを実行するオープンソースのロボットシミュレーターであり、アルゴリズムのテストやロボットの設計などを行うことができる。 また、ROS との連携も協力であるため、今回の ROS を用いたロボットのシミュレーションを行うために用いた。

3.5.1. シミュレーションと結果

今回は、MATLAB を用いて得た極配置によって設計された倒立振子の安定性を確かめるとともに、倒立振子の長さによる安定性の違いを調べるためのシミュレーションを行った。

ここで、制御している時の ROS ネットワーク内のシステムを説明する。

１．Gazebo 上のロボットにおいて、倒立振子と台車をつなぐ joint から GetJointParam 関数を用いて、倒立振子の傾きを取得する。

２．取得した傾きと、系の状態から、倒立振子を制御するために必要な力を計算する。（状態フィードバック制御）

３．台車にかかる制御に必要な力の値から、右の車輪、左の車輪の回転量を求める。

４．求めた車輪の回転量が Gazebo 上のロボットに反映され、倒立振子を制御するように移動する。

シミュレーション内容としては、倒立振子の長さを 0.25[m]、0.5[m]、1[m] にしたときに、倒立振子をあらかじめ傾きをつけた状態からシミュレーションを開始し、どのくらいの傾きまで安定制御が可能であると見なせるかの実験を行った。また、倒立振子における材料は同じものを使用すると仮定して、長さ 0.25[m] の時に、倒立振子自体の重さが 0.15[kg] になるように設定し、その他の長さに関しては、倒立振子の長さに比例して、重くなるように設定を行った。以下がそれぞれの長さの倒立振子を搭載した台車型倒立振子を Gazebo 上でシミュレーションした結果である。

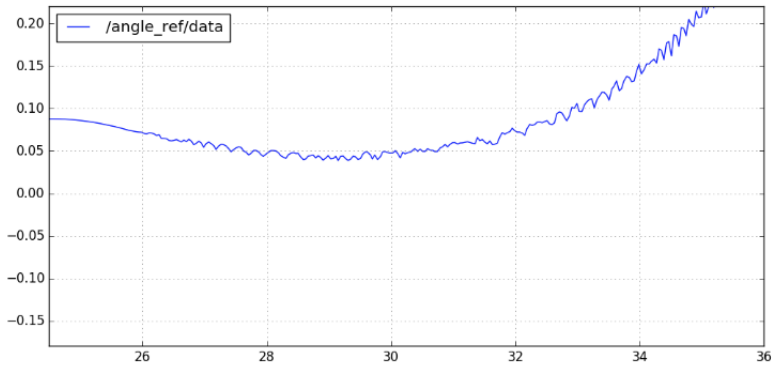

図 3.5: 長さ：0.25[m]、重さ：0.15[kg]

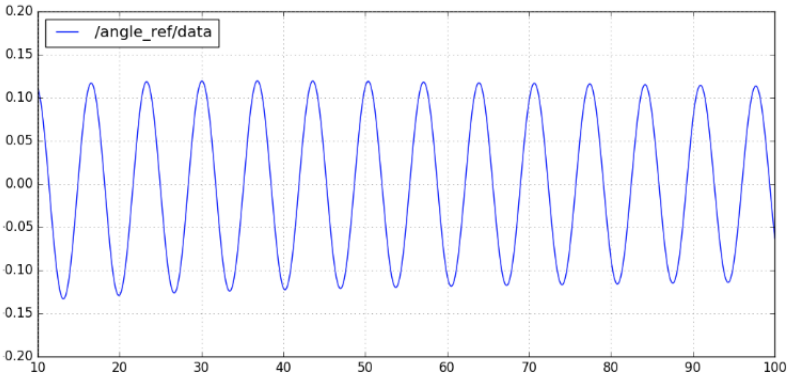

図 3.6: 長さ：0.5[m]、重さ：0.3[kg]

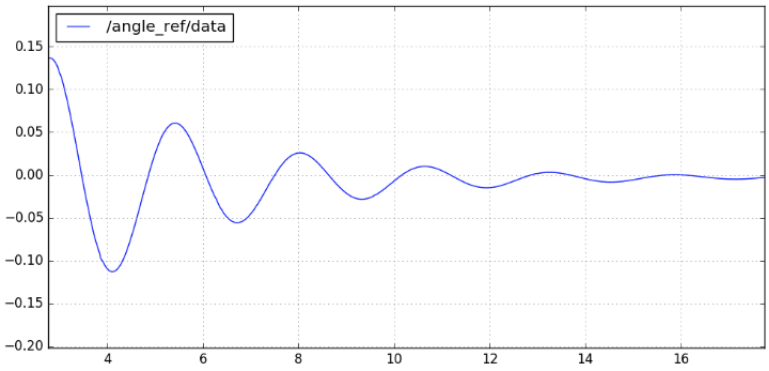

図 3.7: 長さ：1[m]、重さ：0.6[kg]

図 3.5、3.6、3.7 のグラフは、横軸がシミュレーション時間 [s]、縦軸が倒立振子の傾き [rad] を表している。左のグラフは 3 つのグラフを比較すると、倒立振子の長さが 0.25[m] の時と、

0.5[m] の時は、約 0.1[rad] の傾きからの制御を行うことができず、グラフが発散、または振動する結果となった。一方、倒立振子の長さが 1.0[m] の時は、グラフが収束しており、約 0.15[rad] の傾きに対して、安定的な制御を行うことができていることが確認できる。

この章では、倒立振子を垂直に立たせる制御を行う際に必要な制御方法と、その制御方法である状態フィードバックを用いる時に設定する必要のあるゲインを極配置方により求めた。また、制御を行うことが可能と思われるロボットについての実験を行った。本研究では、より安定制御を行う可能性のあるものを倒立振子の制御システムとして使用したいため、極配置が $\lambda = [-1.0 - 1.5 - 2.0 - 2.5]$ となる時の状態フィードバックゲインを用いた制御システムを用いて、移動ロボットとしては、倒立振子の部分の長さが 1.0[m] のものを今後の実験に使用する。

第4章　経路探索

4.1.　経路探索システム

4.1.1.　Turtlebot3

ここでは、本研究で、経路探索を行う際に使用するロボットである Turtlebot3 Burger について述べる。Turtlebot3 は、研究やレジャーで使用するための ROS システムがインストールされている。ロボット部品を目的に合わせて変更したり、プログラムの一部を変更したりすることで、ソフトウェアやハードウェアを自由に開発することが可能な小型移動ロボットである [12][13]。

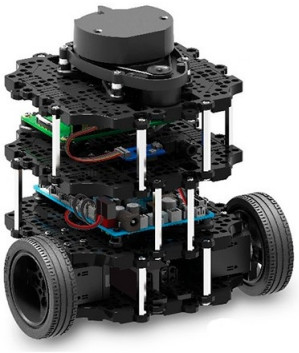

図 4.1: Turtlebot3 Burger

TurtleBot3 には、経路探索を行う際に必要な部品である、IMU や LiDAR などの部品が取り付けられており、その部品を用いて地図作成からロボットのナビゲーションまで行うことができるようなパッケージが予め用意されており、そのパッケージを用いることで経路探索に関す

る研究を比較的簡単に行うことが出来る。本研究においての経路探索は、TurtleBot3 のために用意されたパッケージを用いて行っていく。

4.2. グローバル経路計画

本研究では、経路探索においても特に、グローバル経路計画に焦点をあてて研究を行っていく。そのグローバル経路計画において代表的な２つのアルゴリズムであるダイクストラアルゴリズムと A*アルゴリズムを使用した経路計画の優位性を調べる。以下では、それぞれのアルゴリズムについての説明を行う。

4.2.1. ダイクストラ法

ダイクストラ法は、1956 年にオランダのコンピューター科学者エドガーダイクストラによって考案されたて [14]。その最初の出版物は、最短経路の問題を解決することを目的として、1959 年に出版された。最小経路問題は、交通工学、オペレーションズリサーチ、コンピュータサイエンス、人工知能などの分野で最も研究されている一般的な問題の 1 つである。

ダイクストラ法は、検索ルートとして頂点を選択すると、グラフの他のすべての頂点について、この頂点の最小コストを計算する。コストというのは、その経路を進むのに必要な時間や距離を表しており、よりコストが安くなるような経路を探してく。式グラフは、一般的に、ペア（V、A）で構成されている。ここで、V は頂点の集合であり、A はエッジと呼ばれる頂点のペアのセットのことである。エッジのセットが頂点の順序付けられたペアに関して定義される特別な場合の場合、グラフは有向グラフとなる。

グラフ内の特定の頂点（ノード）で、アルゴリズムは、エッジの重み/コストを使用して、その頂点と他のすべての頂点の間でコストが最も低いパス（つまり、最短パス）を見つける。このシステムを使用して、次の頂点への最短パスが決定されたらアルゴリズムを停止することにより、単一の頂点から次の頂点への最短パスコストを見つけることもできる。

アルゴリズムは、次のようになる。G（V、A）が有向グラフで、k がノードの場合、k の先行ノード、つまり、最初のノードからの最短経路の一部である直前のノードを保持するリスト A（k）を初期化する。次に、開始ノードから k までの最短距離を格納するリスト D（k）を開始し、すべての距離を無限としてマークする。方程式に適応すると、次のようになる。

$$D\left(k\right) = \infty \tag{4.1}$$

ノード s を初期ノードとして初期化し、その距離をゼロに設定する。

$$D\left(s\right) = 0 \tag{4.2}$$

近接するすべてのノードについて、距離を次のように更新する。

$$D(v) = D(s) + d \tag{4.3}$$

そして、エッジのリスト A におけるコストを次のように更新する。

$$A(v) = s \tag{4.4}$$

$D(v) > D(s) + d$ の場合のみ、エッジ d はノード s を次のノード v に接続する。つまり、最初のノードから以前にマークされた v までの距離が、新しい検出よりも大きい場合である。

次に、すべての中で最短距離 D（v）を持ち、まだ分析されていないノード v を選択する。 v が最後のノードである場合、最初のノードに到達するまで、その前のノードを再帰的に調べて、長さが D（v）になる最小パスを見つける。一方、v が最終ノードでない場合は、以下を定義する。

$$s = v \tag{4.5}$$

次に、探索したすべての近接するノードをまだ訪問していないノードとしてマークし、分析を再開する。

これらの操作を繰り返し、最も安価なコストであるパスを求める。

4.2.2. A*アルゴリズム

A*アルゴリズムは、完全性と理想的な効率を考慮した、コンピューターサイエンスで使用されるパス検索およびグラフ通過アルゴリズムである。これは、Peter Hart、Nils Nilsson、Bertram Raphael によって [15] で最初に公開され、ダイクストラ法の拡張として認識され、現在、元の提案されたアルゴリズムから派生したものも複数存在している。

A*アルゴリズムは、重み付けされたグラフィックスの観点から定式化された検索アルゴリズムである。これは、初期ノード（スタート地点）から開始して、最小のコスト（最短距離、最短時間など）で提供される目的ノード（ゴール地点）へのパスを見つけることを目的としていることを意味している。初期ノードから接続されたパスのツリーを維持しながら、完了条件が満たされるまで、一度につき 1 つのエッジでこれらのパスを拡張することによって行われる。

このアルゴリズムは、ダイクストラアルゴリズムとの主な違いとして挙げられるヒューリスティック関数を使用して検索を最適化するものである。つまり、初期ノードから順番に近いノードを探索していくダイクストラアルゴリズムとは違い、1 つのノードからターゲットまでのコストをある程度見積もることができる関数を定義することで、経路を計算する時間の効率化を

図ろうとしたものである。しかし、主な欠点としては、必ずしもダイクストラアルゴリズムと経路が同じにならず、その経路はダイクストラアルゴリズムで求めた経路とは違い、最適化された道ではない可能性があることである。

まず、A*アルゴリズムでは、メインループの各反復でどのパスを拡張するかを決定する必要がある。 これは、パスのコストと、パスを目標まで延長するために必要なコストの見積もりに基づいて行われる。 A*アルゴリズムは、次の関数 a（n）を使用して、コストが安いパスを探し、それを拡張していく。

$$a\left(n\right) = c\left(n\right) + h\left(n\right) \tag{4.6}$$

n がパス上の次のノードである場合、c（n）は n に到達するまでのコストであり、h（n）は n からゴールまでの最も安価なパスのコストを推定するヒューリスティック関数である。

A*アルゴリズムは、延長するように選択されたパスがゴールに到達するパスである場合、または延長する適格なパスがない場合に終了する。

A*アルゴリズムの実装は通常、優先キューを使用して、拡張する最小推定コストを有するノードの繰り返し選択を実行する。この優先キューは、オープンリスト、オープンセットと呼ばれる。 a（x）の値が最も低いノードは、アルゴリズムの各ステップでキューから削除される。その後、そのノードの近くの a と c の値が更新され、キューに追加される。近接しているすべてのノードが分析されたノードは、クローズドリストと呼ばれる別のリストに配置される。アルゴリズムは、目標が見つかるか、オープンリストが空になるまで続く。ヒューリスティック関数において、目的となる h がゼロである場合、目的の a 値は最短経路のコストとなる。

説明したアルゴリズムは、最短パスの長さのみを提供するものである。

マップ上で最短ルートを探す場合、h（x）はゴールからの直線距離を表すことができる。これは、物理的に 2 点間の最短距離を表している。ヒューリスティック関数 h がグラフのノード x とノード y を結ぶエッジ（x、y）に対して追加の条件

$$h\left(x\right) \leq l\left(x, y\right) + h\left(y\right) - h\left(x\right) \tag{4.7}$$

を満たしている場合、l はそのエッジの長さを示し、h は整合性があると言える。 このように、一貫したヒューリスティックにより、A*アルゴリズムは、ノードを 2 回以上処理することなく、理想的なパスを見つけることが保証される。 したがって、次の方程式のコストを削減してダイクストラアルゴリズムを実行することと同等であると言える。

$$l'\left(x, y\right) = l\left(x, y\right) + h\left(y\right) - h\left(x\right) \tag{4.8}$$

4.3. SLAM

SLAM は、自己位置推定と地図作成を同時に行う技術であり、ロボットは未知の環境を探索しながらセンサを用いて、自分の位置を推定しながら、地図の作成を行っていくを意味している。

ロボットアプリケーション開発フレームワークである ROS は、センサーの大部分をサポートし、さまざまな SLAM、経路計画、および画像処理アルゴリズムを効率的に実装することが可能であり、次のようないくつかの SLAM アルゴリズムを提供している。

・HectorSLAM
・Gmapping
・KartoSLAM
・CoreSLAM
・LagoSLAM

[16] によると、KartoSLAM、HectorSLAM、および Gmapping が最も正確なマップを作成する結果となっている。 [17] に示されている結果に基づくと、gmapping は、処理能力がそれほど優れていないロボットシステムでうまく機能する可能性がある。このアルゴリズムは、比較的少数の粒子を使用するものであり、非常に正確なマップを正常に作成するためにリサンプリングを実行するために必要な計算量を軽減する。ラズベリーパイのように処理能力とメモリ容量が制限された低コストのロボットで SLAM を実行する場合は非常に役立つと考えられる。

4.3.1. 自己位置推定

まずは、推定部分の説明を行う。

自己位置推定を行う際に一般的に用いられるセンサは、車輪の回転量を計測するエンコーダや、ロボットの姿勢変化による回転や方向の変化などの慣性運動量を計測する IMU などの慣性センサである。モーションオドメトリデータ $u_{1:\,t-1}$ と外部受容観測 $z_{1:\,t}$ [18][19] が与えられた場合に、マップ m とロボットの軌道 $x_{1:\,t}$ を推定する問題である。地図を推定するにはロボットの軌道が必要であり、その逆も同様である。これにより SLAM の問題を解くことが難しくなる [20]。 この問題を解決するために、後者を以下のような式になるように因数分解する。

$$p\left(x_{1:t}, m|z_{1:t}, u_{1:t-1}\right) = p\left(m|z_{1:t}, u_{1:t-1}\right) * p\left(x_{1:t}|z_{1:t}, u_{1:t-1}\right) \tag{4.9}$$

式 4.9 では、最初に $p\left(x_{1:t}|z_{1:t}, u_{1:t-1}\right)$ を評価してロボットの姿勢を推定し、次に $p\left(m|z_{1:t}, u_{1:t-1}\right)$ を評価してマッピングを行うことを示している。 これに従って、考えられる各姿勢の「パーティクル（粒子）」は、それぞれ異なるマップを持っている。

TurtleBot3 の経路探索は一般的に gmapping を用いて実装されており、その gmapping では FastSLAM2.0 の技術が使用されている。FastSLAM2.0 というのは、上記で説明したものと同

じ概念を採用し、提案分布推定とリサンプリングを実行するタイミングに関していくつかの変更を加えてグリッドマッピングに適用したものとなっている [17]。

その際に用いるアルゴリズムの手順を以下で説明する。

1）サンプリング

モーションモデル（オドメトリ）を使用してロボットのポーズの初期推定を最初に行うことにより、新しい粒子のセットを描画する。 次に、最新の観測結果との比較を行い、新しい粒子がこれまでに取得したマップにどの程度適合しているかを評価することで改善を進める（スキャンマッチング)。 スキャンマッチングを行うエージェントが成功を報告した場合、そのエージェントによって返された姿勢の周りに新しく提案された粒子がサンプリングされる。 その後、新しい提案分布のガウス近似 $N(\mu_t^{(i)}, \sum_t^{(i)})$ が得られ、そこから新しいポーズ（粒子）$xt(i)$ が推定される。 しかし、スキャンマッチングを行うエージェントが失敗を報告した場合は、最初に提案された分布が新しいポーズ（粒子）を推定するために使用される。

2）重要度の重み付け改善された提案分布から抽出された粒子が、次のように式 4.10 で表される重要度サンプリングのルールに基づいて各粒子に割り当てられた目標分布をどの程度適切に表すかを測定する。

$$\omega_t^{(i)} = \frac{p(x_{1:t}^{(i)}|z_{1:t}, u_{1:t-1})}{\pi(x_{1:t}^{(i)}|z_{1:t}, u_{1:t-1})} \tag{4.10}$$

ここで、ω は重み、p は目標分布、π は提案分布を表している。

3）適応的リサンプリングサンプリングステップから得られた粒子セットによって目標分布がどの程度適切に表されるかを表す N_eff の結果に応じて、リサンプリングステップの実行を行うか決める。N_eff は式 4.11 を使用して計算される。

$$N_eff = \frac{1}{\sum_{i=1}^{N}(\omega^{(i)})^2} \tag{4.11}$$

4）地図の推定最後に、それぞれの粒子が持つ地図は、これまでに得られたロボットのポーズと、これまでに得られた観測を考慮して、地図が更新される。

次に、計測部分の説明を行う。

地図を用いて自律走行を行うためには、ロボットが地図中の自身の位置と姿勢を推定する必要がある。屋外であれば GPS を利用して自己位置をある程度推定できるが、室内では、GPS による位置推定は困難である。現在、屋内ロボットにおいて最も多く使用されている自己位置推定方法は、移動量を車輪の回転量から推定するデッドレコニングである。

また、デッドレコニングを用いて求めた移動量を使用して、自己位置推定を行うためには、マップとロボットの位置を連携させる必要がある。

以下に、マップとロボットの位置を連携方法についての説明をする。

map 原点の座標中に odom 原点が存在し、SLAM によって odom 原点を基準にして map が生成される。SLAM を使って map を生成している時は、odom と map の位置は一致している。SLAM によって出来上がった map を参照しながら、odom 原点が map 上のどこに位置するか調整しながら自分の位置を認識するのが自己位置推定の役割になる。

odom 原点の座標中に base_link 原点が存在し、これらを繋ぎ合わせる方法として以下の様な方法がある

・モータエンコーダによって odom 原点から機体 (base_link) の移動量を算出する (エンコーダ値とホイール径から計算するノードを作る)

・IMU センサや GPS センサなどで odom 原点から機体の移動量を算出する

これらの方法を複合的に使ったり、状況に応じて使い分けたりして位置精度を向上させる。

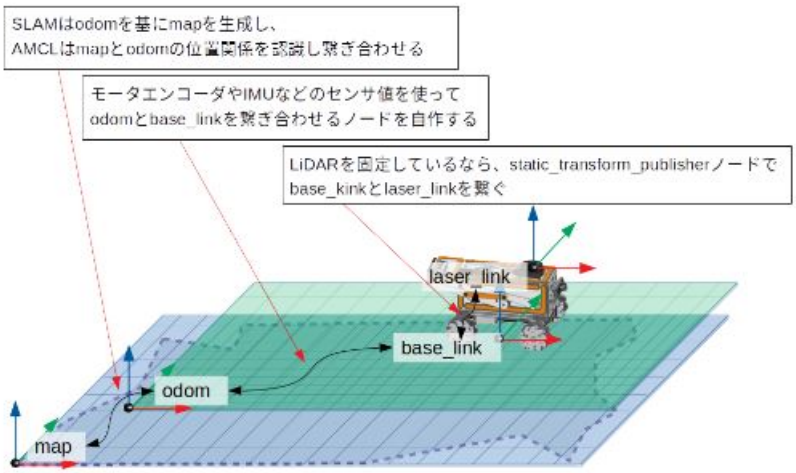

図 4.2: 自己位置推定方法

map 座標：マップの座標。ロボットの初期位置 (odom 原点) もマップ座標のどこかに存在する。

odom 座標：マップ上でロボットが最初に存在する場所。各車輪の回転情報からロボットの位置を計算する。(デッドレコニング)

base_link 座標：ロボットの足下からの各パーツが存在する場所。

4.4. シミュレーション

4.4.1. 実験環境

本研究における、経路探索のシミュレーションを行うために用いた環境を図 4.3 に示す。

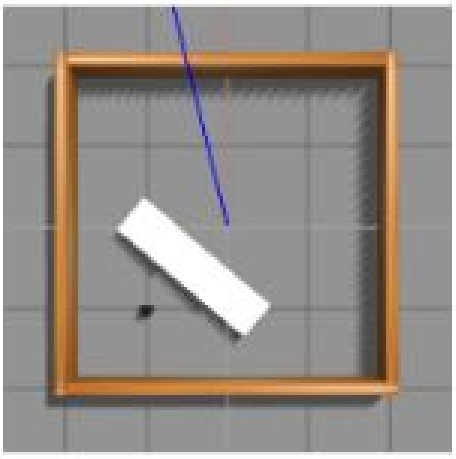

図 4.3: 実験環境

この環境は、4[m]x4[m] の正方形の壁に囲まれており、障害物である直方体 (1[m]x1[m]x2[m]) をおいたものである。この障害物は、ロボットが最終目標に到達するためのパスを作成しにくくすることを目的としている。特に、障害物の両端部分と壁との距離が近く、その部分は経路探索において課題になる部分と考えられる。

ロボットを特定の目的の地点に移動するには、事前に移動する場所の地図を入手する必要がある。 これは、未知の環境の SLAM（同時ローカリゼーションとマッピング）を実行する gmapping と呼ばれる ROS パッケージを介して行われた。それによって作成された地図は図 4.4 である。

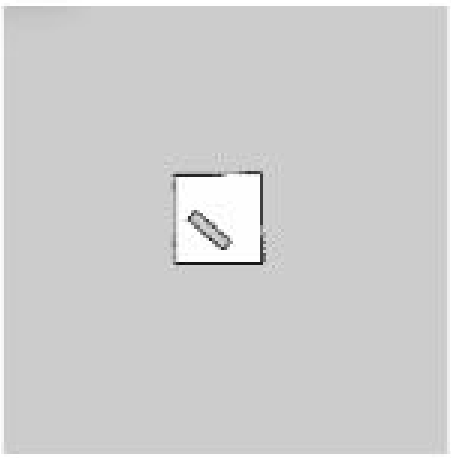

図 4.4: 作成されたコストマップ

図 4.4 の地図はコストマップと呼ばれるものである。空間の中で移動可能な領域や障害物があり移動不可能な領域、レーダーでは測ることができなかった未知の領域を数値で表現した地

図である。図 4.4 で言うと、白色の部分が移動可能な領域、黒色の部分が移動不可能な領域、灰色の部分が未知の領域となっている。

ロボットを目的地に到達させるために、ROS 変位システムは、ローカルとグローバルの 2 種類のパスプランナーを使用する。 今回の目的は、グローバルプランナーアルゴリズムの部分にあるため、ローカルプランナーはすべての実験を通じて同じアルゴリズムを用いた。 ローカルプランナーには、ROS のナビゲーションスタックの一部でもあるダイナミックウィンドウアプローチアルゴリズム [21] が選択された。一部のパラメータは、ロボットの軌道を計算し、狭い通路でロボットが動かなくなるのを防ぐために使用される。 以下の表 4.1 は、これらのパラメータの選択された値と、それらに関する簡単な説明を示している。

表 4.1: navigation 時のパラメータ

パラメータ	値	説明
inflation_radius	0.15	障害物とロボットの中心との最小距離 [m]
cost_scaling_factor	5	インフレ低下曲線のゲイン
controller_frequency	10	各コントローラの周波数 [Hz]
max_vel_x	0.18	最大速度 [m/s]
max_vel_theta	1.0	最大回転数 [rad/s]
sim_time	1.0	生成する軌道に要する時間 [s]
vx_samples	18	速度のサンプリング数
vtheta_samples	20	回転速度のサンプリング数

コストマップの計算に使用するスケーリング変数は以下の式に用いられる

$$exp(-1.0 * cost\_scaling\_factor * (distance\_from\_obstracle - inscribed\_radius)) * (254 - 1) \tag{4.12}$$

distance_from_obstacle は、ロボットの中心と障害物（壁）との距離であり、inscribed_radius は、ロボットに外接する円の半径のことであり、この式は、ロボットと障害物との衝突度合を 254 段階に分けるものである。cost_scaling_factor が小さいものほどロボットが障害物に対して敏感に反応するようになる。

4.4.2. 結果

シミュレーションは、最短経路探索アルゴリズムの動作を検証するために行った。 表 4.1 に示すように、すべてのテストで同じナビゲーションパラメーターを使用した。

図 4.3 の環境において、ダイクストラアルゴリズムを用いて経路計画を行った場合のシミュレーション結果を図 4.5、4.6 に示す。（図 4.5 は Gazebo の GUI 画面、図 4.6 は Rviz の GUI 画面）

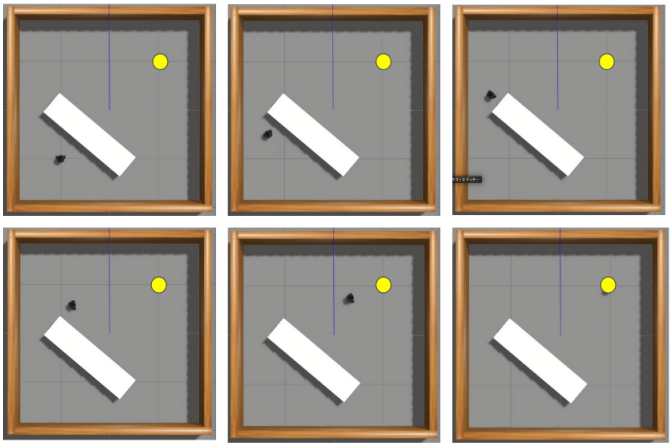

図 4.5: Dijkstra アルゴリズムを用いた経路探索 (Gazebo 画面)

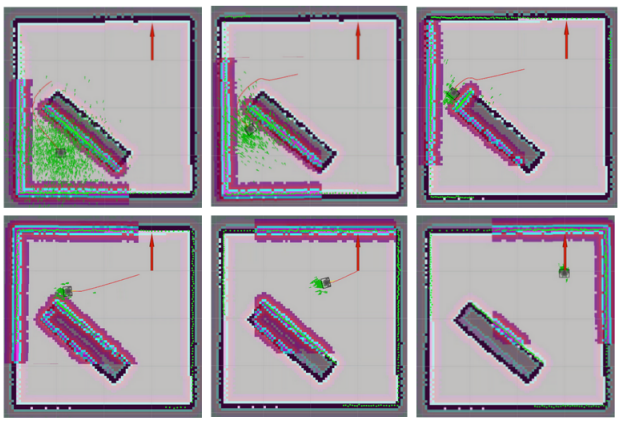

図 4.6: ダイクストラ法を用いた経路探索 (Rviz 画面)

また、A*アルゴリズムを用いて経路計画を行った場合のシミュレーション結果を図 4.7、4.8 に示す。(図 4.7 は Gazebo の GUI 画面、図 4.8 は Rviz の GUI 画面)

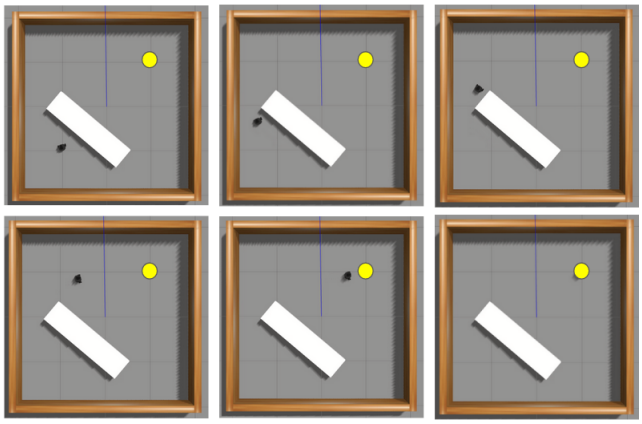

図 4.7: A*アルゴリズムを用いた経路探索 (Gazebo 画面)

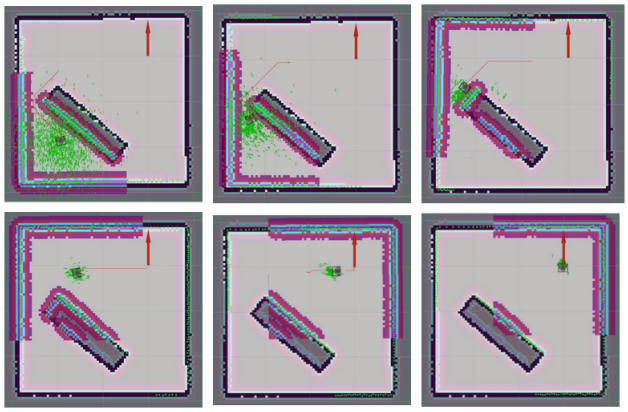

図 4.8: A*アルゴリズムを用いた経路探索 (Rviz 画面)

Rviz の画面に見える矢印が目標の座標と姿勢を表している。図 4.8 の左上を例に説明すると、図の右方向を x 軸の正、上方向を y 軸の正とすると、初期位置と初期方向は、図のロボットのように、それぞれ (-1,-1)、y 軸の正の向きになるようにした。そして、目標位置と目標方向は、図の赤矢印が示すように、(1,1)、y 軸の正の向きとなるようにした。

図 4.6 や図 4.8 の Rviz の画面では、さまざまな計算結果が視覚的に表されており、緑色の粒子状のものは、自己位置推定を行う際に用いる、自分の位置及び姿勢を推定するパーティクル（粒子）が見えている。結果から分かるように、シミュレーションを行う前（左上）から行った後（右下）までを比較すると、緑の点がロボットの周辺に収束していることが分かる。これは、後半になるほど計算に用いる情報が多くなるためと考えられる。マップ上の壁に重なって見える赤や青の線は、レーダーが壁とみなしている部分を表している。

また、計算から実際に移動するのにかかる時間の合計時間でダイクストラ法と A*アルゴリズムの有用さの比較を行った。以下の表 4.2 に同じ環境下で 2 つのアルゴリズムをシミュレーションした時に計測した、計算時間と移動時間の合計時間である。

表 4.2: 経路探索の結果

	ダイクストラ	A*
目的地に到達するまでの時間	33.04[s]	40.1[s]
目的地との位置的に許容できる誤差	0.05[m]	0.05[m]
目的地との方向的に許容できる誤差	0.1[rad]	0.1[rad]

結果は、ダイクストラ法を用いた方が時間が短縮される結果となった。ダイクストラ法の最短経路の正確性と A*アルゴリズムの経路生成時間の効率性を考慮すると、今回用意した閉鎖的な空間を用いた環境において、自律移動ロボットで経路探索を行うと、経路生成における時間の差よりも、移動における時間の差の方が結果に及ぼす影響が強いと考えられる。

第5章　倒立振子を搭載した自律移動ロボットによる経路探索

本章では、ロボットが自分自身の姿勢の安定を保ちつつ、ナビゲーションプログラムから求められた走行制御に基づいて二次元平面上を走行させるための制御アルゴリズムと設計したシステムについて述べる。

5.1.　使用したロボット

倒立振子を搭載した自律移動ロボットによる経路探索において、図5.1のように、turtlebot3-burgerの上に倒立振子を搭載したロボットを使用した。

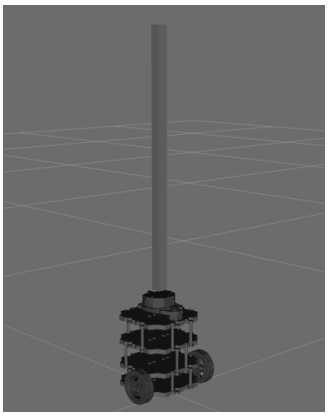

図 5.1: Turtlebot3 Burger

倒立振子の安定制御は第３章で求めた極配置を用いた状態フィードバック制御、経路計画方法は第４章で用いた、ダイクストラ法やA*アルゴリズムを用いたナビゲーションを使用する。

5.2.　システム

ロボットに自分自身の姿勢の安定を保ちつつ、指示に従って二次元平面上を走行させるためには

(a) ロボットの姿勢 (バランス) の制御
(b) ロボットの並進速度と位置の制御
(c) ロボットの中心まわりの回転速度と方位の制御

を同時に行う必要がある。従来より、この種の移動体の制御方法として (b) と (c) を分離して制御することが行われてきた [22]。倒立振子を搭載した移動型ロボットにおいては, これらの制御はいずれも (a) の倒立振子のバランスの制御と関連する。しかし、 (c) のロボットの方位の制御で大きな角速度を与えない限り、それがロボットの姿勢制御 (a) に与える影響は小さい。したがって、本研究では (a) と (b) を組み合わせた姿勢と速度制御と (c) のロボットの方位の制御を分離して行うことにした。

倒立振子を搭載した自律移動ロボットが旋回運動をする場合、実際には一次元の倒立振子モデルをそのまま制御に使用することはできない。しかし、旋回の時の本体の回転角速度を、倒立振子のバランスにほとんど影響を及ぼすことがないような速度に制限することにより、第 3 章で設計した倒立振子の安定制御をそのまま使用した。

二次元平面上を走行するロボットを、目標軌跡に追従させるためには、ロボットの並進速度と回転角速度を決定する必要がある。

まず、ロボットの目標速度の求め方について述べる。経路計画を行い、軌道を生成する navigation パッケージによって算出された速度と角速度のうち、速度を、倒立振子を制御した状態での最終目標速度 v_d となるように設定する。つまり、倒立振子を搭載した自律移動ロボットの目標となる状態は、振子の傾き $\theta = 0$、速度 $v = v_d$ である。

次に、回転角速度の求め方についてだが、本研究では、倒立振子の制御と回転角速度は切り離して考えるため、navigation パッケージによって算出された速度と角速度のうち、角速度をそのまま目標の角速度として扱う。

また、ナビゲーションを行うときに用いるパラメータを表 5.1 に示す。

表 5.1: ナビゲーション時のパラメータ

パラメータ	値	説明
inflation_radius	0.15	障害物とロボットの中心との最小距離 m
cost_scaling_factor	5	インフレ低下曲線のゲイン
controller_frequency	1	各コントローラの周波数 [Hz]
max_vel_x	0.005	最大速度 [m/s]
max_vel_theta	0.5	最大回転数 [rad/s]
sim_time	10	生成する軌道に要する時間 [s]
vx_samples	10	速度のサンプリング数
vtheta_samples	20	回転速度のサンプリング数

倒立振子の安定制御を考慮し、第 4 章の条件と比較し、最大速度と最大角速度を下げた。また、倒立振子が目的の状態になるまでの時間を考慮し、周波数も下げた。

第6章　結論

6.1.　まとめ

本研究では、ROS を用いて倒立振子を搭載した自律移動ロボットの経路計画に向けたダイクストラ法と A*アルゴリズムの比較検討を目的とし、そのシステムの開発を行った。

はじめに、倒立振子の安定制御を行うためのシステムを設計した。そのための台車型倒立振子を設計を行った。ラグランジュの運動方程式から数学モデルを算出し、さらにその数学モデルを制御理論で扱うために状態方程式に変形させた。制御方法としては、状態フィードバック法を用い、そのゲインの決め方としては、極配置法を用いた。極配置法では、任意の極を配置することで、安定制御を行うためのゲインを求めることができるため、MATLAB/simulink でシミュレーションを行い、その結果から、応答性と入力のバランスを考慮した極の配置方法を決定した。次に、その極から求めたゲインを用いた状態フィードバック制御を使用し、Gazebo シミュレーション上での倒立振子の安定制御を目指した。Gazebo は物理エンジンを搭載しており、より現実でのシミュレーションに近い実験を行うことができると考えられるため、実験を行う際は Gazebo シミュレーションを用いた。Gazebo 上での安定制御を目的とした実験では、倒立振子の長さを変えながら、シミュレーションを行った。安定性に優れていたものは、倒立振子の長さが 1.0[m] のものであったため、今後の実験において、これまでのシミュレーションで求めたゲインと振子の長さを用いて実験を行うことを目指した。

次に、経路計画において、代表的なグローバル経路計画であるダイクストラ法と A*アルゴリズムについての比較を行った。この実験をするにあたって、ROS の標準ロボットである、Turtlebot3-Burger を使用した。経路計画を行うにあたって、SLAM を用いて、グローバル経路計画のパスを生成するために必要となる地図の作成を行い、その地図を用いて、スタートからゴールまでのロボットの動きを生成するナビゲーションを行った。ダイクストラ法は、最短経路の正確さがあるため、移動時間が短くなり、A*アルゴリズムは、経路を計算する時間の効率化を図ろうとしたものであるため、必ずしもダイクストラ法と経路が同じにはならないことを考慮し、実験を行った。4[m]x4[m] の正方形内に直方体の障害物を設置した環境を作り、２つの経路計画アルゴリズムを用いて、スタートからゴールまでの時間を測り、比較を行った。この実験における時間は、経路生成から実際に移動するまでの時間を表しており、結果は、ダイクストラ法の方が短く、本実験の環境下での優位性が見られた。

最後に、いままでの倒立振子の安定制御方法と経路経路計画方法を合わせて、倒立振子を搭載した自律移動ロボットのシステムの設計を行った。ロボットは経路計画の時と同様に Turtlebot3-

Burger を使用し、その上に倒立振子を搭載するようにした。システムとしては、倒立振子の安定制御と並進速度の制御を一つの制御対象として、回転速度の制御と分離して制御を行うシステムの設計を行った。

6.2. 今後の展望

今後の取り組みと展望について述べる。本研究では、倒立振子を搭載した自律移動ロボットの経路計画に向けたダイクストラ法と A*アルゴリズムをシミュレーターを用いて比較検討を行うためのシステムの設計を行った。そのため、今後はシミュレーターでの実験を行う必要がある。さらには、現実世界での実験を行っていくことも必要となる。しかし、ROS は、ROS に対応したアクチュエータやレーダーをロボットのパーツとして使用することで、Gazebo シミュレーションのために組んだプログラムをそのまま活かすことができるため、本実験で開発したシステムをそのまま使用できる部分も多くあると考えられる。倒立振子の安定制御においては、本研究では、線形制御を用いたため、ファジィ制御などを用いた非線形制御を行うことで、より安定した制御や複雑な動きに対応できるような制御を目指すことができると考えられる。また、経路計画においては、本研究における比較対象として、グローバル計画の代表的な手法である、ダイクストラ法と A*アルゴリズムを選択したが、他にも様々な手法が研究されており、それらの手法との比較も必要になると考えられる。

謝辞

本研究を進めるにあたり、お忙しい中親身にご指導を頂きました、高知工科大学システム工学群 電子・光システム工学教室准教授 星野孝総先生に心から深く感謝致します。本研究を行うにあたり、高知工科大学システム工学群 Soft Intelligent System on Chip 研究室の皆様には、日頃から様々な意見を頂き、ありがとうございます。最後になりましたが、大学生活 6 年間を支えてくれた家族に深く感謝いたします。

参考文献

[1] 山藤河村: ” 同軸二輪車の姿勢制御”, *JRSC*, Vol. 7, No. 4, pp. 74–79, 1989.

[2] 松本治: ” 適応オブザーバを用いた車輪型倒立振子 の姿勢推定”, 第 9 回日本ロボット学会学術講演会予稿集, pp. 909–910, 1991.

[3] 油田信一: ” つくばチャレンジ：市街地における移動ロボットの自律走行の公開実験—11年の経緯と成果—”, 第 23 回ロボティクスシンポジア予稿集, Vol. 23, pp. 59–66, 2018.

[4] 出村公成, 出村賢聖: ”RoboCup@Home における ROS の利用”, 日本ロボット学会誌, Vol. 35, No. 4, pp. 295–298, 2017.

[5] Turtlebot3: ”Robotis”, 2019.

[6] ”RViz, Baxter”. `http://sdk.rethinkrobotics.com/wiki/Rviz`.

[7] ”Open Dynamics Engine ODE. Multibody Dynamics Simulation Software.”. `http://www.ode.org`.

[8] Koenig, Howard, N.: ”Design and use paradigms for gazebo, an open-source multi-robot simulator.”, *Intelligent Robots and Systems (IROS)*, pp. 2149–2154, 2004.

[9] ”The MathWorks: MATLAB”. `http://www.mathworks.com/products/matlab`.

[10] ”The MathWorks: Simulink”. `http://www.mathworks.com/products/Simulink`.

[11] Kautsky, N.K. Nichols, J., Dooren, P. Van: ”Robust Pole Assignment in Linear State Feedback”, *International Journal of Control*, Vol. 41, pp. 1129–1155, 1985.

[12] Ackerman, E.: ”Robotis and OSRF Announce TurtleBot 3: Smaller, Cheaper, and Modular”, *IEEE Spectrum*, 2019.

[13] J. C. Jesus, M. A. S. L. Cuadros, J. A. Bottega, Gamarra, D. F. T.: ”Deep Deterministic Policy Gradient for Navigation of Mobile Robots in Simulated Environments”, *In International Conference on Advanced Robotics (ICAR)*, 2019.

[14] Dijkstra, E. W.: "A note on two problems in connexion with graphs", *Numerische Mathematik*, Vol. 1, , 1959.

[15] P. E. Hart, N. J. Nilsson, Raphael, B.: ""A Formal Basis for the Heuristic Determination of Minimum Cost Paths"", *IEEE Transactions on Systems Science and Cy-bernetics*, Vol. 2, pp. 100–107, 1968.

[16] J. M. Santos, D.Portugal, Rocha, R. P.: "An evaluation of 2D SLAM techniques available in Robot Operating System", *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–6, 2013.

[17] G. Grisetti, C. Stachniss, Burgard, W.: "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters", *IEEE Trans.Robot.*, Vol. 23, No. 1, pp. 34–46, 2007.

[18] Durrant-Whyte, H., Bailey, T.: "Simultaneous localization and mapping: part I", *IEEE Robot. Autom. Mag.*, Vol. 13, No. 2, pp. 99–110, 2006.

[19] Bailey, T., Durrant-Whyte, H.: "Simultaneous localization and mapping (SLAM): part II", *IEEE Robot. Autom. Mag.*, Vol. 13, No. 3, pp. 108–117, 2006.

[20] J. M. Santos, D.Portugal, Rocha, R. P.: "A hybrid approach to RBPF based SLAM with grid mapping enhanced by line matching", *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1523–1528, 2009.

[21] D. Fox, W. Burgard, Thrun, S.: "The Dynamic Window Approach to Collision Avoidance", *IEEE Robotics and Automation Magazine*, Vol. 4, , 1997.

[22] Iida, S., Yuta, S.: "Vehicle command system and trajectory control for autonomous mobile robot", *Proc. of the 1991 IEEE/RSJInt. Workshop on Intelligent Robots and Systems*, pp. 212–217, 1991.