埼玉大学　工学部
機械工学科

令和5年度　卒業論文

○○○○○○○○○○○の研究

Study on XXXXXXXXXX

学科長	荒居善雄　教授　印
主指導教員	琴坂信哉　准教授　印
副指導教員	程島竜一　准教授

提出日	2023年2月XX日
研究室	設計工学
学籍番号	20TM028
氏　　名	長谷川 大晴

目次

図目次

表目次

第 1 章

序論

第 1 章では，本研究の背景と先行研究，そして研究の目的を述べる．

1.1 背景

1.1.1 多脚ロボットの対環境適応性能

不整地における多脚ロボット

近年，人間に代わって作業を行う移動ロボットの導入が進められている．Pudu 社が開発したロボットの BellaBot が，レストランで配膳の作業を行う姿は一般に見ることができるようになった．これらのロボットの多くはタイヤやクローラを用いての移動を行うが，その他の移動様式として，脚を使用して移動を行う脚ロボットが存在する．脚ロボットは他の移動様式を用いて移動するロボットに比べて，以下に示すような利点がある [1].

- 障害物をまたいで移動できる
- 脚接地点を離散的に選択できる

障害物をまたいで移動できることにより，脚ロボットはタイヤでは移動できないような不整地においても移動することが可能である．また，砂利で舗装された道のような，クローラではスリップしてしまうような環境においても移動することが可能である．実際に，林業を行う山間地において脚ロボットを導入し，作業を行う実証実験が行われている [2].

また，離散的に脚接地地点を選択できるため，タイヤやクローラによる移動と比較して環境に与える影響が小さい．この特徴を生かして，海底で作業を行うロボットの研究が行れた．これは，クローラによる移動では海底の砂が巻きあがってしまい，カメラやセンサを遮ってしまうためである．

以上より，脚ロボットは不整地における移動に適しており，対環境適応性能が高いと言える．

6 脚ロボットの静的安定性

前述したような特性を持つ多脚ロボットであるが，その性能は脚数によって変化する．ロボットの性能の指標として，歩行速度や消費エネルギーなどがあるが，不整地において用いることを考え，静的安定余裕 [3]（Stability Margin）を指標として分別することとする．静的安定余裕とは，ロボットが静的に安定するために必要な脚位置と重心位置の関係を表す指標である．支持脚を結んでできる多角形の重心が，支持脚の内部にある場合，ロボットは静的に安定する．そのため，静的安定余裕では多角形の辺から重心までの距離を評価に用いる．

2 脚ロボット

2 脚ロボットは，人間のように歩行を行うことができる．しかし，安定性を確保するためには，歩行速度を遅くする必要がある．そのため，2 脚ロボットは歩行速度が遅く，消費エネルギーが大きいという特徴を持つ．

4 脚ロボット

6 脚ロボット

脚数が 6 脚よりも多いロボット

1.1.2 固定歩容と自由歩容

多脚ロボットが歩行を行う際には，脚を適切な順番で動かす必要がある．

歩容にはさまざまな種類があるが，大きく分別するとカムやリンクを用いて，周期的に脚を動かす固定歩容と，非周期的に脚を動かす自由歩容がある．

1.1.3 グラフ探索による自由歩容パターン生成手法

本研究室においては，6 脚ロボットの自由歩容パターン生成手法として，グラフ探索による由歩容パターン生成手法を提案してきた．グラフ探索による自由歩容パターン生成手法は，脚位置や動作を離散化することで歩容をグラフに落としこみ，そのグラフの探索によって数動作先までの歩容パターンの組み合わせを網羅的に調べ，最適な歩容パターンを選択する手法である．この手法の特徴として，数動作先までを考慮して歩容パターンを生成するため，デッドロックに陥りにくいという点や，効率的な歩容パターンを生成することができるという点が挙げられる．グラフ探索による自由歩容パターン生成手法は 4 脚ロボットにおいて行われていたが [4]，6 脚ロボットにおいては行われていなかった．これは，脚の本数が増えることで脚の動かし方の組み合わせが増えるため，実時間内の計算が困難になるためである．

そこで本研究室では，グラフ探索による自由歩容パターン生成手法を 6 脚ロボットに適用す

るため，離散化された脚位置の組み合わせを利用してグラフの階層構造化を行った．また，自由歩容パターン生成による接地地点の計算と脚軌道の生成を分離した．これらによって，グラフ探索による自由歩容パターン生成手法を 6 脚ロボットに適用することが可能になった．

本研究室では，ロボットを動作させる地形やロボットの動作によって段階的に開発を行っており，これまでに 2 次元空間において，直進動作 [5]，目的姿勢での停止 [6]，旋回動作 [7] を行うための歩容パターン生成手法の実装に成功した．また 3 次元空間においても，離散化された脚位置の 3 次元空間への拡張 [8] を行うことで, 直進動作 [9] を行うための歩容パターン生成手法の実装に成功した．

1.2 本研究の目的

これまでの研究によって，3 次元の不整地において，重心高さを変更しつつ，自由歩容パターン生成を行うことが可能となった．しかし低頻度ではあるが，グラフ探索に成功したとしても脚軌道が生成できず，その歩容パターン通りに歩行することできなくなり，動作を停止してしまう問題が生じてしまった．

そこで本論文では，常に脚軌道生成に成功するような歩容パターン生成手法を提案し，脚軌道生成の失敗による動作停止を防ぐことを目的とする．

1.3 本論文の構成

本論文は，全 6 章から構成される．

第 2 章「歩容パターンの再評価手法の提案」では，常に脚軌道生成が可能になる手法として，歩容パターンの再評価手法を提案し，その機能を述べる．

第 3 章「歩容パターンの再評価手法の実装」では，提案したプログラムの実装方法を述べる．

第 4 章「再評価手法の有効性の確認のための歩行シミュレーション」では，提案手法を用いたシミュレーション実験の結果を述べる．

第 5 章「常に脚軌道生成が可能な自由歩容パターン生成手法を用いた実機による歩行実験」では，提案手法を用いた実機試験の結果を述べる．

第 6 章「結論」では本論文の結論と今後の課題を述べる．

第 2 章

歩容パターンの再評価手法の提案

第 2 章では，先行研究の手法およびその問題点を指摘し，常に脚軌道生成が可能な自由歩容パターン生成手法として，歩容パターンの再評価手法を述べる．

2.1　本研究室における自由歩容パターン生成の先行研究

2.1.1　グラフ理論について

本論文ではグラフ理論を用いた自由歩容パターン生成手法を論ずるため，まずはグラフ理論について述べる．グラフとは，頂点（ノード）とそれらを結ぶ辺（エッジ）からなる図形である．このグラフを用いて，さまざまな問題を取り扱う学問をグラフ理論という．

以降の説明を簡単にするため，この論文で用いるグラフ理論の用語について簡潔に述べる．エッジに向きがあるものを有向グラフ，逆に向きを持たないものを無向グラフという．また，閉路を持たず，かつ，すべての頂点間に経路が存在するグラフを木という．このような木構造をもつグラフのうち，図 2.1 のように，根となるノードを持ち，そのノードからすべてのノードに到達可能なものを根付き木という．

根付き木において，あるノードから遷移可能なノードをそのノードの子ノードと呼ぶ．逆に，あるノードに遷移可能なノードをそのノードの親ノードと呼ぶ．親ノードを持たないノードを根ノードと呼び，子ノードを持たないノードを葉ノードと呼ぶ．また，あるノードから根ノードまでのエッジの数をそのノードの深さと呼び，根ノード自身の深さは 0 となる．

図 2.1 においては，ノード A が根ノードであり，ノード B，C がその子ノードである．また，ノード B，ノード D，E，ノード C はノード F を子ノードとして持ち，ノード D，E，F は葉ノードである．ノード A の深さは 0 であり，ノード B，C の深さは 1，ノード D，E，F の深さは 2 となる．

グラフのあるノードから別のノードに到達するための経路をパスと呼び，これを求めること

をグラフ探索と呼ぶ．グラフ探索には，深さ優先探索，幅優先探索などのさまざまなアルゴリズムが存在する．深さ優先探索では，始点となるノードから，深さが深くなる方向を優先して探索を行う．これに対して，幅優先探索では，始点となるノードから，深さが浅いノードを優先して探索を行う手法である．

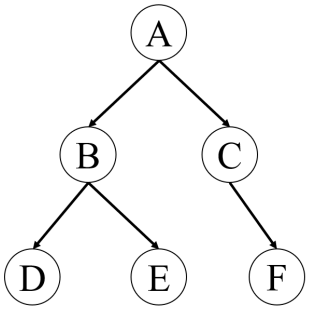

Fig. 2.1 Tree Graph

2.1.2 歩容パターングラフの定義

本研究においては，6 脚ロボットの歩容パターンをグラフを用いて表現する．グラフはロボットの状態をノードとし，ロボットの状態間の遷移，つまりロボットの動作をエッジとして作成する．グラフは有向の根付き木とし，現在のロボットの状態を根ノード，その姿勢から 1 動作で到達できる姿勢を子ノードとして根ノードに接続する．また，このようにして作られたグラフを歩容パターングラフと定義する．

グラフ探索がよく用いられる題材は路線図や回路図などであり，ノードの数が有限である．しかし，歩容パターングラフはロボットの状態や動作を対象とするため，無数の組み合わせが存在する．そのため，先行研究では状態や動作を離散化することで歩容パターン生成をグラフへ落とし込んでいる．以下に各要素の離散化について述べる．

脚位置の離散化

ロボットの脚位置は脚の可動範囲内であれば，無数の位置を取ることができる．そのため本手法では，基準となる脚位置を決め，その基準からの相対位置を用いて脚位置を離散化している．Prabir らが提案した手法では 2 次元平面での移動を前提としていたが [4]，これを三浦が 3 次元空間へ拡張した [8].

図 2.2 に支持脚の脚位置の離散化の様子を示した．図 2.2 のように脚位置の基準座標を 4 とし，脚位置 4 と同じ高さにあるかつ，進行方向に対して基準位置よりも前方にある脚位置を 6，後方にある脚位置を 2 とする．また，脚位置 6 よりも高い位置にある脚位置を 7，低い位

置にある脚位置を 5 とし，脚位置 2 よりも高い位置にある脚位置を 3，低い位置にある脚位置を 1 とする．このようにして，脚位置を 7 つに離散化している．遊脚している脚の脚位置は，支持脚の脚位置と 1 7 に対応させ，1’ 7’ とする．

これにより，脚位置 1 7 から脚位置 1’ 7’ への遷移によって，脚の遊脚運動を表現することができる．同様に，脚位置 1 7 から脚位置 1 7 への遷移によって，脚の接地運動を表現することができる．また，脚位置 1’ 7’ 内での遷移によって，脚の前後移動を表現することができる．このように脚位置を 7 つ離散化することによって，脚の水平方向の移動と垂直方向の移動をグラフで表現することができる．

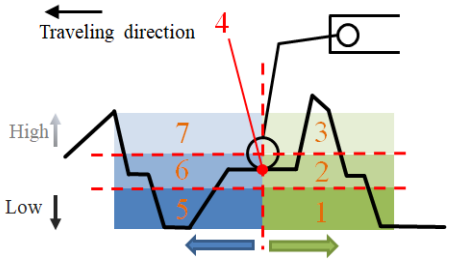

Fig. 2.2 Discretization of Leg Posistion

グラフの階層構造

エッジの離散化

エッジの離散化について述べる．前述したように歩容パターングラフにおいて，エッジはロボットの動作を表す．ロボットの動作は，脚の接地・遊脚運動と，重心の移動からなる．

脚軌道生成の分離

これまで離散化された脚位置やエッジについて論じてきたが，実際に脚を動作させるためには脚軌道について考える必要がある．これは，脚軌道生成をグラフ探索による歩容パターン生成から分離しているためである．歩容パターン生成に脚軌道生成を組み込んだ場合，

2.1.3 脚軌道生成の失敗

2.2 歩行シミュレーションによる脚軌道生成失敗時の脚先位置の特定

2.2.1 シミュレーション実験の目的

先行研究では脚軌道生成の失敗による動作の停止が報告された上，その原因は脚先が脚の可動範囲の外を通ることによるものであると考察されてきた．しかし，具体的に脚先がどのような位置になると脚軌道生成が失敗するのかは明らかにされていなかった．そのため予備実験として，先行研究と同じ条件で歩行シミュレーション実験を行い，ロボットの脚軌道を確認することで，脚軌道生成失敗の原因を考察した．

2.2.2 シミュレーションの条件

本研究室ではロボットの動作のシミュレーションを行うためのシミュレーターソフトウェアを自作し，シミュレーション実験で用いてきた．シミュレーターは C++ で記述されており，WindowsAPI を用いて GUI を実装し，ロボットを表示している．また，GUI の表示のプログラムをより簡単に記述するため，ゲームプログラミングに用いられるライブラリの DxLib を用いている．

シミュレーターは物理演算を行っておらず，トルク不足や摩擦，脚先の滑りによるずれを考慮していない．そのため，ロボットのアクチュエータは無限のトルクを持ち，脚先は滑りなく接地するものと仮定している．本来これらのパラメータを考慮すべきではあるが，本研究においては歩容パターン生成が可能であるかを確認することが目的であるため，これらのパラメータは考慮しないこととしている．

2.2.3 シミュレーションの結果

以下の図に脚軌道生成失敗時の脚先の座標を示す．

2.2.4 脚軌道生成の失敗の原因の考察

2.3 常に脚軌道生成が可能な自由歩容パターン生成手法の検討

常に脚軌道生成を可能にするためには，近似された脚可動範囲を適切に設定する必要がある．

2.4 歩容パターンの再評価手法

第 3 章

歩容パターンの再評価手法の実装

第 3 章では，〜を述べる．

3.1　グラフ探索による自由歩容パターン生成手法の実装

3.2　歩容パターンの再評価手法の実装

3.3　グラフ探索による自由歩容パターン生成手法の統合

先行研究において，グラフ探索による自由歩容パターン生成手法はロボットの動作によって別のものを使用しており，それぞれ別のプログラムで実装されている．不整地の踏破を行うためには，様々な動作を組み合わせて使用する必要がある．そのため本研究では，グラフ探索による自由歩容パターン生成手法を統合し，1 つのプログラムで実行できるようにした．

3.3.1　ロボットの動作

先行研究において，既に実装されているロボットの動作を表 3.1 に示す．表において，2 次元空間とはロボットが平面上で動作することを表し，3 次元空間とはロボットが立体的な地形で動作することを表す．

3.3.2　自由歩容パターン生成手法の切り替えアルゴリズム

Table. 3.1　実装済みのロボットの動作

動作＼ロボット	2 次元空間	3 次元空間
直進	○	○
その場旋回	○	×
旋回	○	×
旋回	○	×
特定姿勢での静止	○	×

第 4 章

実験

第 4 章では，〜を述べる．

4.1　○○の実験

4.1.1　○○の実験目的

4.1.2　○○の実験手順

4.1.3　○○の実験結果

4.1.4　○○の実験考察

4.2　××の実験

第 5 章

結論

5.1　結論

本論文では，〜〜を論じた．

第 1 章「序論」では，〜を述べた．第 2 章「理論と実施計画」では，〜を述べた．第 3 章「実験装置や開発機械」では，〜を述べた．第 4 章「実験」では，〜を述べた．第 5 章「結論」では本論文の結論と今後の課題を述べた．

5.2　今後の課題

付録 A

C++20 への移行

A.1　C++20 の新機能

C++ にはコンパイラの標準規格として，C++98，C++03，C++11 などが存在する．その中でも C++11 以降は約 3 年に一度のペースで新しい規格が策定されている．先行研究のプログラムでは，C++17 を使用していたが，本研究では C++20[10] を使用するように変更を行った．C++20 では，C++17 からの変更点として，以下のようなものがある．

- constexpr 関数の制限緩和
- 標準ライブラリの多くの関数が constexpr 関数に変更
- concept の導入
- std::number，std::format の導入

これらの機能を使用することで，プログラムの最適化を行うことができる上，可読性を向上されることができる．以下に各機能の詳細を記述する．

A.2　constexpr 変数・関数

constexpr 関数はコンパイル時に評価される関数であり，C 言語におけるマクロ関数のような処理を実現するために使用される．たとえば，以下のようなプログラムを考える．

Listing A.1　convert func as macro

```
1  #include <iostream>
2
3  // defined as macro
4  #define CONVERT_TO_RAD(deg) (deg * 3.1415926535f / 180.0f)
5
6  int main()
7  {
8    float deg = 90.0f;
9
```

```
10    // It will deploy deg * 3.1415926535f / 180.0f.
11    std::cout << CONVERT_TO_RAD(deg) << std::endl;
12  }
```

このプログラムでは，マクロ関数を使用して，度数法で表された角度をラジアンに変換している．プログラムを記述する際にはラジアンで角度を表現すると可読性が低くなるため，度数法で記述することによる利点は大きいが，実際の処理ではラジアンで角度を表現する必要があるので，このようなマクロが実際に使用されることは多いだろう．

しかし，マクロにはいくつかの問題点がある．まず 1 つとして，マクロは常にグローバルスコープで定義されることである．通常 C++ の開発においては，クラスや名前空間を使用して，変数や関数をスコープを限定して定義することが多い．スコープを限定することは，変数や関数の名前が衝突することを防ぐことができるため，大規模な開発や複数のライブラリを使用する場合には必須である．だが，マクロは名前空間内に定義したとしてもグローバルスコープに展開されるため，名前の衝突を防ぐことができないのである．

もう 1 つの問題点は，マクロは型の確認を行わないことである．C++ は python や java script などの動的型付け言語とは異なり静的型付け言語である．そのため，コンパイル時に型の不一致や意図しないキャストを警告として出力することができ，ランタイムエラーを防ぐことができる．しかし，マクロはコンパイル時に文字を張り付けるだけであるため，型の確認を行わない．そのためたとえば上記のマクロ関数において，引数を int 型や double 型，果てはその他のクラスなどに変更しても，float 型との掛け算演算子が定義されていればコンパイルは通ってしまう．先行研究のプログラムではマクロ関数を使用している箇所が多く存在したため，実際に浮動小数点型の float 型と double 型が混在している箇所が存在した．

これを constexpr 関数を使用することで，以下のように書き換えることができる．

Listing A.2　convert func as constexpr

```
1   #include <iostream>
2
3   // defined as constexpr function
4   constexpr float ConvertToRad(float deg)
5   {
6     return deg * 3.1415926535f / 180.0f;
7   }
8
9   int main()
10  {
11    // declared as constexpr variable
12    constexpr float deg = 90.0f;
13
14    // It will deploy 1.57079632675f.
15    std::cout << ConvertToRad(deg) << std::endl;
16  }
```

このように constexpr 関数を使用することで，マクロ関数のようにコンパイル時に評価される関数を定義することができる．また，constexpr 関数はコンパイル時に評価が行われるため，コンパイル時に型の確認を行うことができる．加えて，constexpr 関数はスコープを限定

することができるため，名前の衝突を防ぐことができる．

以上のようにマクロの持つ問題点を解決することができるが，constexpr 関数の本当の利点はコンパイル時に処理が実行されることである．A.2 の 14，15 行目にあるように，引数を含めてコンパイル時に評価が可能であれば，コンパイル時に関数が実行される．そのため，関数の呼び出しによるオーバーヘッドを削減することができる．

謝辞

本論文の研究と執筆にあたりその細部に至るまで終始懇切なる御指導と御鞭撻を賜りました，埼玉大学大学院理工学研究科 ○○○○教授に謹んで深謝の意を申し上げます.

本研究を共同遂行して頂いた，○○○○氏に御礼申し上げます.

本研究に懇切なる御助言を頂いた，○○○○氏に御礼申し上げます.

研究室において常に熱心な御討論を頂きました，OB・学生の方々に感謝の意を表します.

○○○○について有益なご助言を数多く賜りました○○○○氏（○○○○株式会社），に深謝申し上げます.

参考文献

[1] Freyr Hardarson：“Locomotion for difficult terrain”，1997.

[2] 国立研究開発法人 新エネルギー・産業技術総合開発機構：“NEDO 先導研究プログラム 2021 年度”，Vol.1, p.57, 2022.

[3] 広瀬，塚越，米田: “不整地における歩行機械の静的安定性評価基準”, J. of Robotic Systems, Vol.16, No.8, pp.1076-1082, 1998.

[4] Prabir K. Pal, K. Jayarajan: “Generation of Free Gait A Graph Search Approach”, IEEE Transactions on Robotics and Automation, Vol.7, No.3, 1991.

[5] 大木，程嶋，琴坂: “多脚ロボットの不整地踏破を目標とするグラフ探索を用いた歩行パターン生成”, ロボティクス・メカトロニクス講演会講演概要集, 2015.

[6] 中岡，程嶋，琴坂: “不整地における特定位置・脚着地点への遷移を目的とした多脚歩行ロボットの歩行動作計画”, 日本機械学会関東支部総会講演会講演論文集, 2016.

[7] 椎名，程嶋，琴坂: “グラフ探索を用いた多脚ロボットの旋回歩容パターン生成”, 日本機械学会関東支部総会講演会講演論文集, 2018.

[8] 三浦，程嶋，琴坂: “グラフ探索による多脚歩行ロボットの自由歩容パターン生成 第 4 報：出現頻度によるノード枝刈りを用いた探索時間の短縮”, 日本機械学会関東支部総会講演会講演論文集, 2019.

[9] 波東，程嶋，琴坂: “グラフ探索による多脚歩行ロボットの自由歩容パターン生成 第 5 報: 重心の上下移動のエッジを用いた重心高さの変更”, 日本機械学会関東支部総会講演会講演論文集, 2020.

[10] Thomas Koppe：“Changes between C++17 and C++20 DIS”. ISO/IEC JTC1 SC22 WG21 P2131R0. 2020. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p2131r0.html, (参照 2024/01/23).