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

第 1 章　序論 1
1.1　背景 1
1.2　本研究の目的 5
1.3　本論文の構成 5

第 2 章　歩容パターンの再評価手法の提案 7
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 7
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 14
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 15
2.4　歩容パターンの再評価手法 15

第 3 章　歩容パターンの再評価手法の実装 17
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 17
3.2　歩容パターンの再評価手法の実装 17
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 17

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 19
4.1　直進動作の自由歩容パターン生成シミュレーション 19
4.2　旋回動作の自由歩容パターン生成シミュレーション 19
4.3　動作統合時の自由歩容パターン生成シミュレーション 19

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実験 21
5.1　実験目的 21
5.2　実験に使用した 6 脚ロボット 21
5.3　歩行条件 21
5.4　実験に使用した地形 21
5.5　結果 21

5.6 考察 21

第 6 章 結論 23

6.1 結論 23

6.2 今後の課題 23

付録 A C++20 への移行 25

A.1 C++20 の新機能 25

A.2 constexpr 変数・関数 25

謝辞 29

参考文献 31

図目次

1.1 Demonstration Experiment with Spot 2
1.2 Crabster 2

2.1 Examples of simple graphs 8
2.2 Tree Graph 8
2.3 Discretization of Leg Posistion 10
2.4 Search in the Same Hierarchy 11
2.5 Search in the Difficult Hierarchy 12
2.6 Comparelegmodel 13
2.7 PhantomX Mark Ⅱ 15

表目次

2.1 Simulation Environment 15

3.1 実装済みのロボットの動作 18

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 不整地における多脚ロボットの活用

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 [1]．Pudu 社が開発したロボットの BellaBot[2] 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．これらのロボットは人間が移動を行う空間での使用を前提としており，多くはタイヤやクローラを用いて移動を行うが，その他の移動様式を用いるロボットとして，脚を使用して移動を行うロボット（以下脚ロボット）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 [3].

- 障害物をまたいで移動できるため，対地適応性が高い
- 脚接地点を離散的に選択できるため，環境に与える影響が小さい
- スリップすることなく全方向に移動できる

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凹凸が激しい地形や，不連続な地形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．この特徴を生かして，実際に Fig.1.1 のように Boston Dynamics Inc. によって開発された 4 足歩行ロボットの Spot[4] を用いて，林業を行う山間地で作業を行う実証実験が行われている [5]．山間地は斜面である上に，伐根作業によって木の根が残っているため凹凸が激しく，タイヤやクローラでは移動が困難である．しかし，脚ロボットである Spot は障害物をまたいで移動することができるため，このような環境での作業に適しており，導入による人手不

足の解消や，作業効率の向上が期待されている．

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．この特徴を生かしたロボットの実例として，韓国海洋科学技術院によって開発された 6 脚ロボットの Crabster[6] があげられる．Fig.1.2 に示した Crabster は，流れの早い海底での作業を想定して開発されており，6 本の脚による移動を行うことや，4 本の脚を海底に接地させ，残りの 2 本の脚を用いて作業を行うことが可能である．脚を用いた移動では海底の砂を大量に巻き上げることがないため [7]，カメラを用いた観察や，センサを用いた地形の計測に優れている．以上より，脚ロボットは対地適応性が高く，環境に与える影響が小さいため，不整地での移動に適しているといえる．

Fig. 1.1　Demonstration Experiment with Spot

Fig. 1.2　Crabster

不整地で脚ロボットを使用することを前提に，脚ロボットの歩行形態について考える．脚ロボットの歩行形態は大別して，動歩行と静歩行に分けられる．静歩行とは，支持脚多角形とも呼ばれる，地面に接地している脚をすべて結んでできる多角形上に重心をおき，常に静的な安定性を保ちながら歩行することである．それに対して動歩行とは静的には不安定であるが，動的に安定を保ちながら歩行することである．静歩行では常に支持脚多角形上に重心を置くため，動歩行と比較して歩行速度が遅くなる．しかし，動歩行は重心の位置や歩行の速度を踏まえた動的な制御が必要となるため，静歩行に比べて制御が複雑になる．加えて不整地の歩行では地形の状態を考慮する必要があるため，より制御の難易度は高くなる．そのため，静歩行を用いたほうが不整地歩行を実現しやすいことがわかる．

静歩行による不整地の歩行は，主に 4 脚以上の脚ロボットで研究されている．なぜならば，2 脚を有する脚ロボットは静的な安定性を保ちながら歩行する場合，重心が支持脚面積の中になるように制御する必要があるため，歩行速度が遅くなるからである．そのため，より早い静歩行には 4 脚以上の脚を有する脚ロボットが有利である．本論文ではそのような 4，6 脚以上の脚を有する脚ロボットを多脚ロボットと呼ぶことにする．

これまで，多脚ロボットの不整地での活用のための研究は多く行われてきた．しかし，実際に不整地で多脚ロボットが活用された事例は少ない．原因として，以下に示すような問題を多脚ロボットが抱えているためと考えられる [3].

- 脚の機構はアクチュエータの数が多いため，重量が大きくなってしまう
- 移動には歩行のための制御が必要であり，タイヤやクローラに比べて制御が複雑になる

多脚ロボットの積極的な活用を考えると，脚の機構の軽量化と，歩行のための制御の手法が必要である．そこで本研究では多脚ロボットの制御に着目し，不整地における多脚ロボットの活用のための制御手法を論じることとする．

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．脚ロボットの歩行中の脚を動かす順序や，脚を動かすタイミングのことを歩容と呼ぶ．歩容にはさまざまな種類があるが，大別すると周期的に同じパターンの脚の動作を繰り返す固定歩容と，非周期的に脚を動かす自由歩容に分けられる．

固定歩容は自然界においても多く見られる歩容であり，動物の歩行や，昆虫の歩行などがこれにあたる．代表的なものとしては，低速で 4 足歩行をする動物にみられるクロール歩容がある．これは，歩行中に常に 3 本の脚が地面に接地しており，1 本の脚が遊脚する歩容パターンである．3 本の脚が地面に接地しているため，常に静的な安定性を保つことができるが，1 本づつしか脚を動かせないため，一般に歩行速度は遅い．

6 足歩行する昆虫の中には，トライポッド歩容と呼ばれる歩容をとるものがある．この歩容パターンでは，まず片側の前脚と後ろ脚，反対側の中央の脚を地面に接地させ，残りの 3 本の脚を遊脚させる．次に，反対側の前脚と後ろ脚，片側の中央の脚を地面に接地させ，残りの 3 本の脚を遊脚させる動作を繰り返す．この歩容では同時に 3 本の脚を地面に接地させることができるため，クロール歩容よりも歩行速度が速くなる．

固定歩容は平地において比較的高速かつ安定した歩行を実現することができるが，不整地においては遊脚の可動範囲内の接地点が存在しない場合，歩容を維持することができない．多脚ロボットの対地適応性を生かすためには，地形に応じた自由歩容パターンを生成する必要があるといえる．

1.1.3 グラフ探索による自由歩容パターン生成手法

自由歩容パターンを生成する手法として，グラフ探索による自由歩容パターン生成手法がある [9]．これは，脚位置や動作を離散化することでロボットの歩容をグラフに落としこみ，そのグラフを探索すること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．

しかし、グラフ探索による自由歩容パターン生成手法は，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，グラフの規模が大きくなり，実時間内の計算が困難になるという問題がある．4 脚ロボットにおいて，静的安定性を保ちながら歩行する場合，1 度に遊脚することができる脚は 1 本であるため，実時間内の計算は容易である．だが，6 脚ロボットは静的安定性を保ちながら最大 3 本の脚を遊脚することができるため，グラフの規模が大きくなり，実時間内の計算が困難になる．この問題を解決するために Prabir らは歩容をウェーブ歩容に限定することで，グラフの規模を小さくすることに成功した [10]，また新らは歩容をトライポッド歩容に限定することで [11]，同様にグラフの規模を小さくしている．これらの手法は歩容を限定することでグラフの規模を小さくすることに成功しているが，ロボットの行う動作の種類が限定されてしまうという問題がある．

そこで本研究室ではロボットの動作を制限することなく，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 [12]，目的姿勢での停止 [13]，旋回動作 [14]

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，脚位置の離散化方法を 3 次元に適応させたことで [15], 直進動作 [16] 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．

これは，グラフ探索と脚軌道生成を分離したことによって生じた問題であり，先行研究では継続的にこの問題について言及されていたが，グラフの規模を大幅に増大させることなく，また歩容生成の成功率を低下させることなく，この問題を解決することはできなかった．

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

最初にグラフ探索よる自由歩容パターン生成手法において用いる用語を定義し，先行研究で行われてきた自由歩容パターン生成手法について述べる．また，先行研究で用いられてきた自由歩容パターン生成手法の問題点について述べる．

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説明をする．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．グラフ上のあるノードから別のノードにエッジを用いて移動することを遷移と呼ぶ．遷移の際，移動前のノードを始点，移動後のノードを終点と呼ぶ．グラフの種類は大別して有向グラフと無向グラフに分けられ，エッジに向きがあるものを有向グラフ（Fig.2.1(b)），逆に向きを持たないものを無向グラフ（Fig.2.1(a)）という．また，閉路を持たず，かつ，すべてのノード間にエッジ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，Fig.2.2 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．根付き木を図形として表現する場合は簡単のため，Fig.2.2 のように根を上部に配置することが多い．

根付き木には無向グラフと有向グラフの 2 種類が存在するが，後述する歩行パターングラ

フは有効グラフであるため，有向の根付き木について説明を行う．根付き木のエッジが，根が始点となるように伸びている場合，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．

例えば Fig.2.2 において，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．

(a) Undirected Graph (b) Directed Graph

Fig. 2.1 Examples of simple graphs

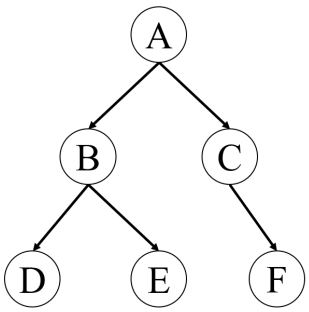


Fig. 2.2 Tree Graph

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．

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．

本手法では，まず歩容パターングラフを作成する．そして，根ノードから最も最適な動作を行う葉ノードまでのパスを，グラフ探索によって求め，そのパスに含まれる深さ 1 のノードを次の動作としてロボットに実行させる．これを繰り返すことで，ロボットの歩容パターンを生成しているのである．

グラフ探索による歩容パターン生成においては，網羅的にロボットの状態を調べ上げるため，実時間内の計算を行うにはグラフの規模を小さくすることが求められる．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し，そのすべてを網羅的に調べ上げることは困難である．そのため，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む必要がある．以下に各要素の離散化について述べる．

グラフの階層構造

前述のとおり，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 [9]，これを三浦が 3 次元空間へ拡張した [15].

Fig.2.3 に支持脚の脚位置の離散化の様子を示した．Fig.2.3 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 1〜7 に対応させ，脚位置 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，脚の上下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，遊脚の水平方向の移動を表現することができる．以上より，脚の上下運動と脚の水平方向の移動をグラフで表現することができることを示した．

このような脚位置の離散化を行うことで，脚位置の組み合わせを有限個に抑えることができ

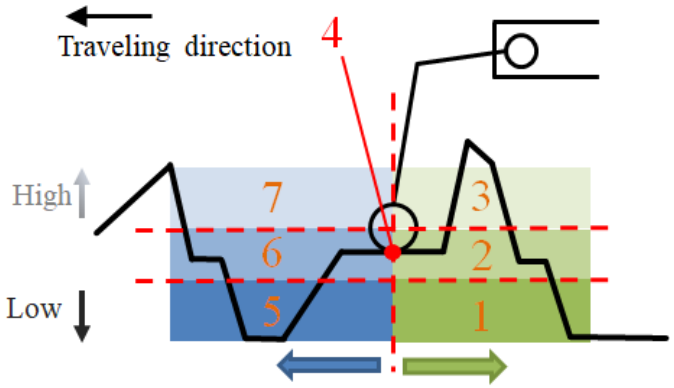


Fig. 2.3 Discretization of Leg Posistion

るが，脚位置の組み合わせは未だ，$7^6 = 117649 \approx 10^5$ 通り存在し，遊脚である時を含めるとさらに組み合わせが増えることがわかる．プログラムの実行環境によって左右されるが，プログラミング言語の C++ で作成した処理では約 1 秒間に 10^8 回程度の計算が可能であるとされており [17]，この組み合わせをすべて歩容パターングラフに追加した場合，実時間内の処理を行うにはグラフの規模が大きくなりすぎる．そこで，大木らは脚位置の組み合わせを階層構造化することで，遊脚時の脚位置 1'~7' を省略し，探索するノード数を減らすことに成功した [12].

階層構造とこれを利用した探索の方法を説明するために，遊脚の組み合わせと脚位置の組み合わせについて定義を行う．遊脚の組み合わせは，ロボットの各脚について，その脚が遊脚であるか支持脚であるかを表すノードの要素である．6 脚ロボットの右前脚を 1 番目の脚として，時計回りに 2 番目の脚から 6 番目の脚とする．この時，i 番目の脚が支持脚であることを $v_i = 1$，遊脚である時を $v_i = 0$ とすると，遊脚の組み合わせ V は式 (2.1) のように表すことができる．

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\} \tag{2.1}$$

遊脚の組み合わせは $2^6 = 64$ 通り存在するが，6 脚，5 脚，4 脚が遊脚である組み合わせや，隣り合う 3 脚が遊脚である組み合わせは実際には取りえない組み合わせであるため，探索するべき組み合わせは $2^6 - {}_6\mathrm{C}_6 - {}_6\mathrm{C}_5 - {}_6\mathrm{C}_4 - 6 = 36$ 通りとなる．

また，脚位置の組み合わせとは離散化した脚位置において，各脚がどの位置にあるかを表すノードの要素である．i 番目の脚が脚位置 j にあることを $k_i = j$ とすると，脚位置の組み合わせ K は式 (2.2) のように表すことができる．

$$K = \{k_1, k_2, k_3, k_4, k_5, k_6\} \tag{2.2}$$

式 (2.1)，式 (2.2) よりロボットの脚の状態は遊脚の組み合わせ V と脚位置の組み合わせ K を用いて表すことができるようになった．i 番目の脚の状態を $l_i = v_i \cdot k_i$ とすると，L は式 (2.3) のように表すことができる．

$$L_{ij} = \{v_1 \cdot k_1, v_2 \cdot k_2, v_3 \cdot k_3, v_4 \cdot k_4, v_5 \cdot k_5, v_6 \cdot k_6\} \tag{2.3}$$

式 (2.3) より，脚位置の組み合わせが $K = \{1, 1, 1, 1, 1, 1\}$ で遊脚の組み合わせが $V = \{0, 1, 0, 1, 0, 1\}$ である時の脚の状態を $L_{ij} = \{0, 1, 0, 1, 0, 1\}$ と表すことができる．同様に，脚位置の組み合わせが $K = \{3, 1, 3, 1, 3, 1\}$ で遊脚の組み合わせが $V = \{0, 1, 0, 1, 0, 1\}$ である時の脚状態は $L_{ij} = \{0, 1, 0, 1, 0, 1\}$ と表すことができる．このことから，ある脚位置の組み合わせから，別の脚位置の組み合わせへの遷移は，脚の状態 L が等しいときのみ可能であることがわかる．

以上の定義より，階層構造と探索方法について説明することができる．階層とは脚位置の組み合わせ K が等しく，かつ，遊脚の組み合わせ V が異なるノードの集合と定義され，遊脚の組み合わせが 36 通り存在することから，同じ階層内のノードは 36 個存在する．脚の上下運動を実現したい場合は，遊脚の組み合わせ V が異なるノードを探索する必要があるため，図 2.4 のように階層内のノード 36 個のみを探索すればよい．

また，脚の水平運動を実現したい場合は，脚位置の組み合わせ K が異なるノードを探索する必要があるため，図 2.5 のように脚の状態 L が等しくなるノードのみを探索すればよい．脚の状態 L が等しくなるノードは，最大で 3 脚が遊脚しているときの $7^3 = 343$ 個であるため，十分に実時間内の計算が可能である．

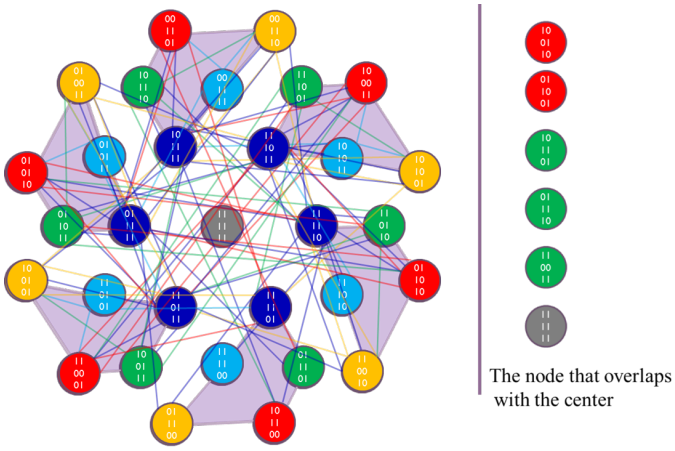


Fig. 2.4 Search in the Same Hierarchy

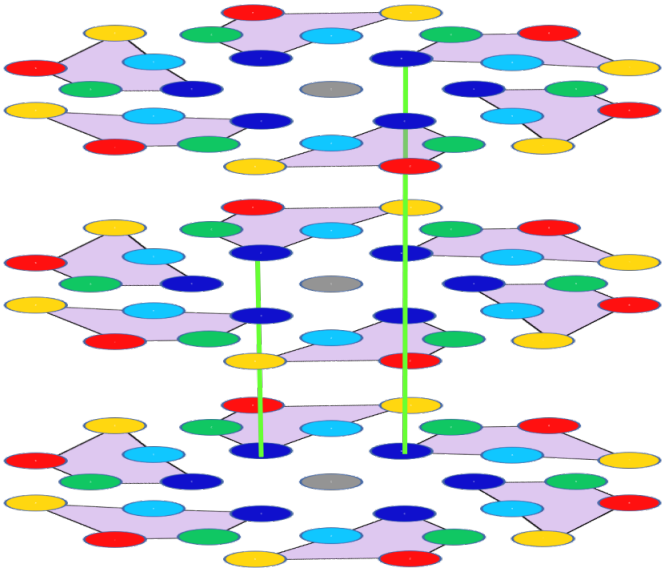


Fig. 2.5 Search in the Difficult Hierarchy

脚軌道生成の分離

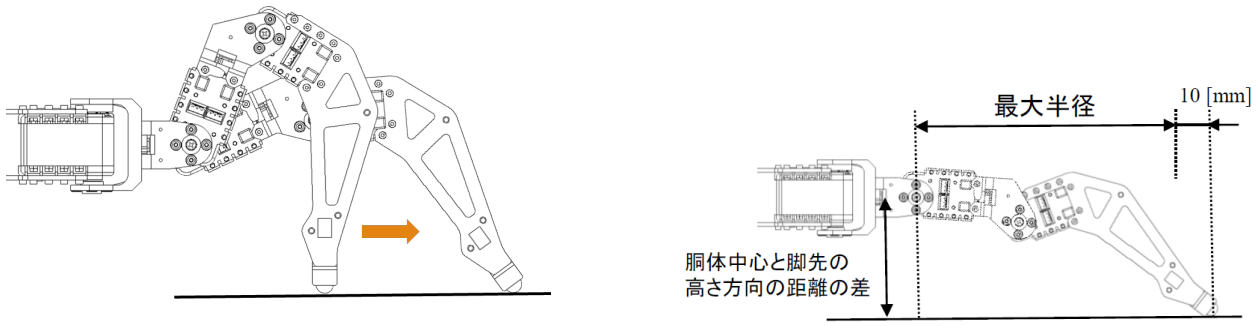
次に歩容パターングラフのエッジについて述べる．歩容パターングラフにおいて，エッジはロボットの動作を表す．ロボットの動作は脚の接地・遊脚運動と，重心の移動からなるため，これらの動作に対応するエッジを作成する．具体的には，脚の上下移動のエッジ，脚の水平移動のエッジ，重心の上下移動のエッジ，重心の水平移動のエッジ，そして，重心の回転のエッジを用いてロボットの動作を表現する．

これらのエッジには，移動前と移動後のノードを補完するための状態を持っておらず，単純に移動前と移動後のノードを結ぶのみである．これはつまり，歩容パターングラフを生成するプログラムと，脚軌道を生成するプログラムが分離されていることを意味する．脚軌道を考える場合，ロボットの関節の可動範囲を考慮する必要があり，逆運動学解を用いる脚の関節角度の計算が求められる．しかし，逆運動学の計算には計算負荷の大きい逆三角関数の計算が必要となり，各エッジについて網羅的に計算を行うことを考えると，実時間内の計算が困難になってしまう．そのため本手法では，歩容パターングラフを生成するプログラムを分離し，歩容パターングラフの生成時には脚の可動範囲は近似的な値を用いて計算することで，実時間内の計算を可能にしている．

近似的な脚の可動範囲の求め方について述べる．近似的な脚の可動範囲は脚の付け根を中心とする環状の扇形として表現する．簡単のため，以降は扇形の外径を最大半径，内径を最小半径と呼ぶことにする．まず，脚先が届くことができる範囲を求めるために，最大半径を以下の手順で求める．

(1) 図 2.6(a) のように水平方向に脚先を $1[mm]$ ずつ伸ばす.
(2) 脚先を伸ばすことができなくなった場合,
図 2.6(b) のように脚先と脚の付け根の高さ方向の距離の差をインデックスとして,
脚先と第 1 関節の水平方向の距離を最大半径として記録する.
(3) 脚先を $1[mm]$ 下げて同様の処理を繰り返す. 脚先を下げることができなくなった場合,
処理を終了する.

次に最小半径をロボットの脚長などのパラメータから決定する. 先行研究では実験機にあわせ, $120[mm]$ とした.



(a) Horizontal Movement of the Leg Tips (b) Determination of the Maximum Radius

Fig. 2.6 Comparelegmodel

2.1.3 脚軌道生成の失敗

先行研究ではグラフの階層構造化および脚軌道生成の分離によって, 実時間内の計算が可能になった. しかし, 実際に実機を用いて歩行実験を行ったところ, 低頻度ではあるが脚軌道生成に失敗することがあった. 失敗の原因として, 脚の可動範囲に近似的な値を用いていることがあげられる. 近似的な脚の可動範囲の境界近くは, 実際には脚の可動範囲外であるが, 近似的な値を用いているために脚の可動範囲内と判定されてしまっている領域があると考えられる. そのため, 脚軌道や脚の接地点が脚の可動範囲外になってしまい, 脚軌道生成に失敗することがあると考えられる.

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，波東らの研究 [16] の実機試験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特定することにした．

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を行ってきた．本論文でも同様に，シミュレーション実験に自作のシミュレーターソフトウェアを用いた．

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[18] 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によって得られた脚接地点に脚先を届かせることが可能であるか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．

シミュレーションの計算環境

シミュレーションの計算環境は Table.2.1 に示したとおりである．

モデルとするロボット

モデルとするロボットは，Trossen Robotics 社の PhantomX Mark Ⅱ [19] （以下 PhantomX）とする．PhantomX（Fig.2.7）は 6 脚ロボットであり，各脚に 3 つのアクチュエータを持つ．また，関節配置は脚の付け根からヨー・ピッチ・ピッチの順である．

Table. 2.1 Simulation Environment

CPU	11thGen Intel Core(TM) i5-11400
RAM	32.0GB
OS	Windows 11 Home
開発環境	Visual Studio 2022 Community
使用言語	C++20

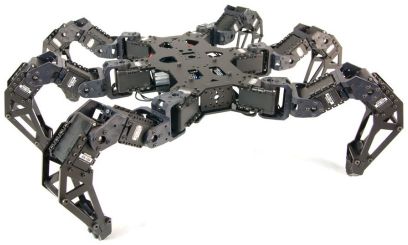


Fig. 2.7 PhantomX Mark Ⅱ

歩行する地形

歩行する地形を Fig.**??**に示す．地形は 5 種類あり，それぞれ平地，上り段差，下り段差，上り斜面，下り斜面である．実機試験の条件に合わせ，段差は上り，下りともに高さが 100$[mm]$ とし，斜面は上り，下りともに角度が 15$[deg]$ とした．

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 脚軌道生成に失敗する原因の考察

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，第 2 章で述べた歩容パターンの再評価手法の実装方法について述べる．

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

再評価手法の実装方法の説明の前に，グラフ探索による自由歩容パターン生成手法の実装方法について述べる．

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，さまざま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，すで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

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

第 4 章では，〜を述べる．

4.1　直進動作の自由歩容パターン生成シミュレーション

4.2　旋回動作の自由歩容パターン生成シミュレーション

4.3　動作統合時の自由歩容パターン生成シミュレーション

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実験

5.1　実験目的

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　実験に使用した 6 脚ロボット

5.3　歩行条件

5.4　実験に使用した地形

5.5　結果

5.6　考察

第 6 章

結論

6.1　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．

6.2　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[20] 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

```cpp
#include <iostream>

// defined as macro
#define CONVERT_TO_RAD(deg) (deg * 3.1415926535f / 180.0f)

int main()
{
  float deg = 90.0f;

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

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，マクロはプリコンパイル時に文字を置換するだけであるため，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] Sotnik S, Lyashenko V：“Prospects for Introduction of Robotics in Service”，International Journal of Academic Engineering Research. Vol.6，pp.4-9， 2022.

[2] Pudu Robotics Inc：“BellaBot”, https://www.pudurobotics.com/jp/products/bellabot (参照 2024/01/23).

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

[4] Boston Dynamics Inc：“Spot®”，https://bostondynamics.com/products/spot/ (参照 2024/01/23).

[5] 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.

[6] B. H. Jun, Hyungwon Shim: “A Dexterous Crabster Robot Explores the Seafloor”, The ACM Magazine for Students, Vol.20, pp.38-45, 2014.

[7] J. Y. Kim, B. H. Jun: “Mechanical Design of Six-Legged Walking Robot, Little Crabster”, Oceans - Yeosu, pp.1-8, 2012.

[8] 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.

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

[10] Prabir K. Pal, V. Mahadev and K. Jayarajan: “Gait generation for a six-legged walking machine through graph search”, Proceedings of the 1994 IEEE International Conference on Robotics and Automation, vol.2, pp.1332-1337，1994.

[11] 新, 田窪, 上野：“障害物環境下におけるトライポッド歩容の脚配置計画”, ロボティクス・メカトロニクス講演会講演概要集，2015.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 秋葉, 岩田, 北川：“プログラミングコンテストチャレンジブック”，毎日コミュニケーションズ，2010.

[18] DX ライブラリ置き場．https://dxlib.xsrv.jp/ (参照 2024/01/23).

[19] Trossen Robotics：“PhantomX AX Hexapod Mark II Kit”．`https://www.trossenrobotics.com/hex-mk2`，(参照 2024/01/23).

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