

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 XXXXXXXXXXXX

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

<b>第 1 章</b>	<b>序論</b>	1
1.1	背景 . . . . .	1
1.2	本研究の目的 . . . . .	5
1.3	本論文の構成 . . . . .	5
<b>第 2 章</b>	<b>歩容パターンの再評価手法の提案</b>	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 . . . . .	18
2.3	歩容パターンの再評価手法 . . . . .	23
<b>第 3 章</b>	<b>歩容パターンの再評価手法の実装</b>	27
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 . . . . .	27
3.2	歩容パターンの再評価手法の実装 . . . . .	39
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 . . . . .	41
<b>第 4 章</b>	<b>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</b>	43
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 . . . . .	43
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 . . . . .	43
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 . . . . .	43
<b>第 5 章</b>	<b>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</b>	45
5.1	実験目的 . . . . .	45
5.2	実験に使用した 6 脚ロボット . . . . .	45
5.3	歩行条件 . . . . .	45
5.4	実験に使用した地形 . . . . .	45
5.5	結果 . . . . .	45
5.6	考察 . . . . .	45

---

<b>第 6 章</b>	<b>結論</b>	47
6.1	結論 . . . . .	47
6.2	今後の課題 . . . . .	47
<b>付録 A</b>	<b>C++20 への移行</b>	49
A.1	C++20 の新機能 . . . . .	49
A.2	constexpr 変数・関数 . . . . .	49
<b>付録 B</b>	<b>脚の可動域を表示するプログラム</b>	53
B.1	概要 . . . . .	53
B.2	導入方法 . . . . .	53
B.3	プログラムの仕様 . . . . .	54
<b>付録 C</b>	<b>PhantomX Mark II のサーボモータ</b>	55
C.1	サーボモータの仕様 . . . . .	55
<b>謝辞</b>		61
<b>参考文献</b>		63

# 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	Image of Gait Pattern Graph . . . . .	9
2.4	Discretization of Leg Posistion . . . . .	10
2.5	Search in the Same Hierarchy . . . . .	12
2.6	Search in the Difficult Hierarchy . . . . .	12
2.7	Caluculation of the Maximum Radius . . . . .	14
2.8	Approximated Range of Motion . . . . .	14
2.9	PhantomX Mark II . . . . .	16
2.10	Leg Coordinate Axis . . . . .	16
2.11	Joint and Link Name . . . . .	16
2.12	Approximated Range of Motion in Simulation . . . . .	18
2.13	Approximated Range of Motion in Experimental Equipment . . . . .	18
2.14	Terrain . . . . .	21
2.15	Result of Simulation . . . . .	23
2.16	Result of Experimental Equipment . . . . .	23
2.17	Revaluation Methodology Procedures . . . . .	25
3.1	Graph Search Sequence . . . . .	29
3.2	Graph Search Class Diagram . . . . .	29
3.3	RobotStateNode Struct . . . . .	31
3.4	Vector3 Struct . . . . .	31
3.5	Quaternion Struct . . . . .	31

3.6	Leg State Bit . . . . .	31
3.7	RobotOperation Struct . . . . .	32
3.8	MapState Struct . . . . .	34
3.9	Map View . . . . .	34
3.10	NodeCreator Interface . . . . .	35
3.11	Discretization of the Vertical Shift of the Center of Mass . . . . .	35
3.12	Candidate Area of Center of Mass . . . . .	36
3.13	Determination of Center of Mass . . . . .	36
3.14	Gait Pattern Generator Revaluation Class Diagram . . . . .	40
3.15	Leg Range Before Revaluation . . . . .	41
3.16	Leg Range After Revaluation . . . . .	41
B.1	Git Bash . . . . .	54
B.2	Display Git Version . . . . .	54
C.1	Dscription AX-18A (page1–2) . . . . .	55
C.2	Dscription AX-18A (page3–6) . . . . .	56
C.3	Dscription AX-18A (page7–10) . . . . .	57
C.4	Dscription AX-18A (page11–13) . . . . .	58

# 表目次

2.1	Link Length of PhantomX . . . . .	16
2.2	Simulation Environment . . . . .	20
2.3	Failure Count of Simulation . . . . .	22
2.4	Failure Count of Experimental Equipment . . . . .	22
2.5	Walkable Terrain . . . . .	25
3.1	RobotOperationType Enum . . . . .	33
3.2	Leg Combination Table . . . . .	38
3.3	Implemented Robot Operation . . . . .	42



# 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、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。文献 [3] では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と述べられている。

- ・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

このような特徴をもつ脚ロボットであるが、実際に不整地で活用された事例は多くない。原因として、[3]では以下に示すような問題を脚ロボットが抱えているためと述べられている。

- ・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歩行の制御手法を論じることとする。

不整地で脚ロボットを歩行させることを前提に、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と定義される[8]。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、重心の位置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、実用的な歩行速度を得ることが難しいためである。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。

先述した問題点から、多脚ロボットの脚数は少ないほうが望ましい。実際に多くの研究において、静歩行が可能な最小の脚数を有する4脚ロボットや6脚ロボットが用いられている。

### 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に、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

最初にグラフ探索に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

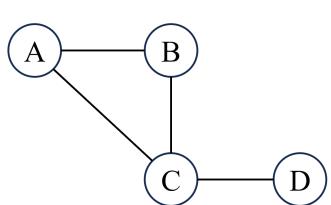
本論文ではグラフ理論を用いた自由歩容パターン生成手法を論ずるため、まずはグラフ理論について説明をする。グラフとは、頂点（ノード）とそれらを結ぶ辺（エッジ）からなる図形である。このグラフを用いて、さまざまな問題を取り扱う学問をグラフ理論という。

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。

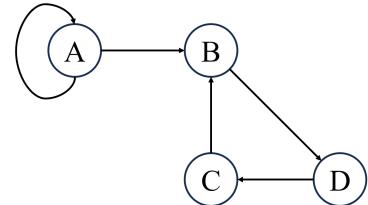
根付き木には無向グラフと有向グラフの2種類が存在するが、後述する歩行パターングラ

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。

たとえば 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

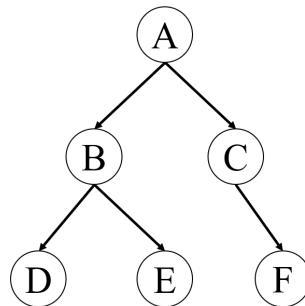


Fig. 2.2 Tree Graph

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、これを求めるの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。Fig.2.3 に歩容パターングラフのイメージを示した。

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に、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方法について述べる。

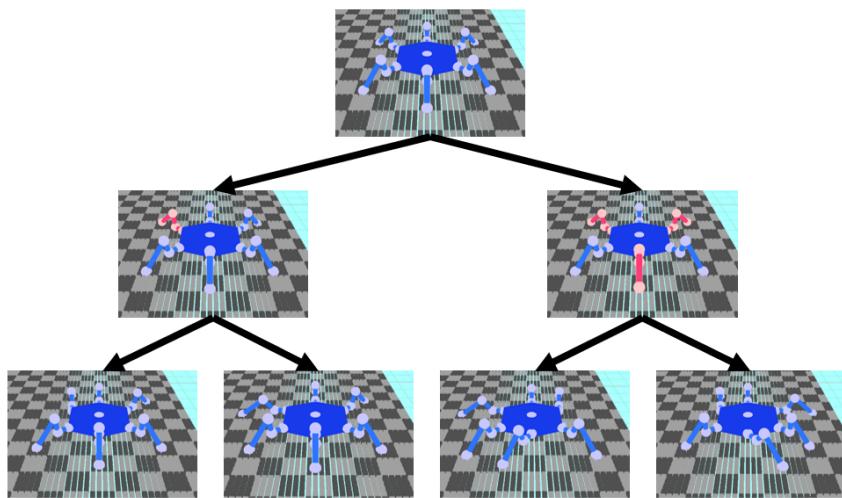


Fig. 2.3 Image of Gait Pattern Graph

### 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.2.4 に支持脚の脚位置の離散化の様子を示した。Fig.2.4 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ことを示した。

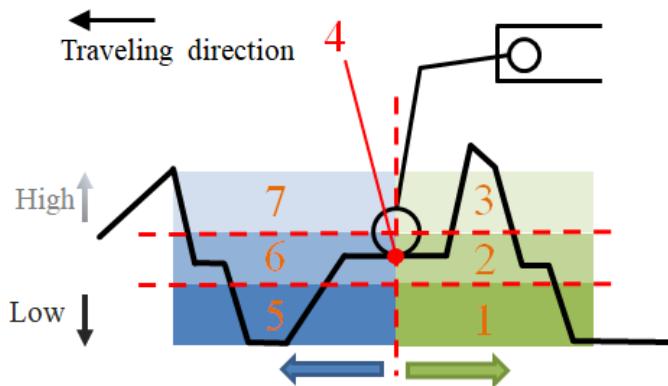


Fig. 2.4 Discretization of Leg Posistion

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てしまうことがわかる。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].

階層構造とこれを用い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。

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\} \quad (2.1)$$

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 $2^6 - {}_6C_6 - {}_6C_5 - {}_6C_4 - 6 = 36$ 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。

$$K = \{k_1, k_2, k_3, k_4, k_5, k_6\} \quad (2.2)$$

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。

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

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、階層構造と階層を用いたグラフの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2.5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 2.6 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。

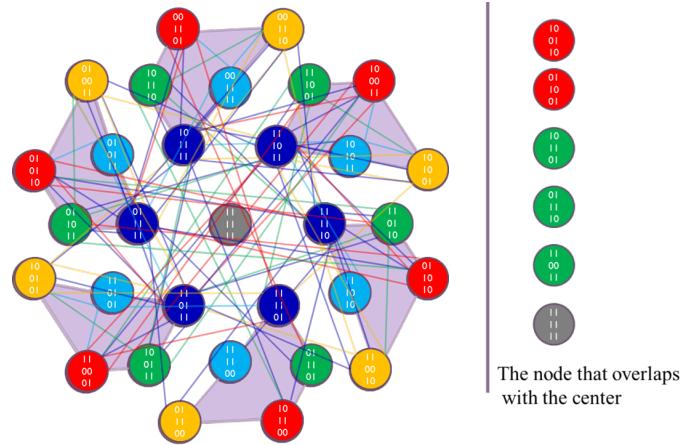


Fig. 2.5 Search in the Same Hierarchy

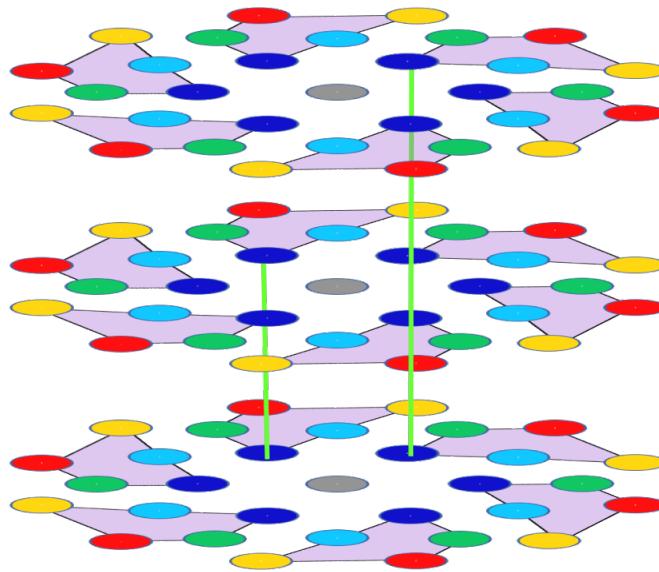


Fig. 2.6 Search in the Difficult Hierarchy

### 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。

近似された脚の可動範囲の求め方について述べる。近似された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、脚先を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) 脚の付け根と脚先が同じ高さになるように脚先を上げる。
- (2) 図 2.7(a) のように水平方向に脚先を  $1[mm]$  ずつ伸ばす。
- (3) 脚先を伸ばすことができなくなった場合、

図 2.7(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。

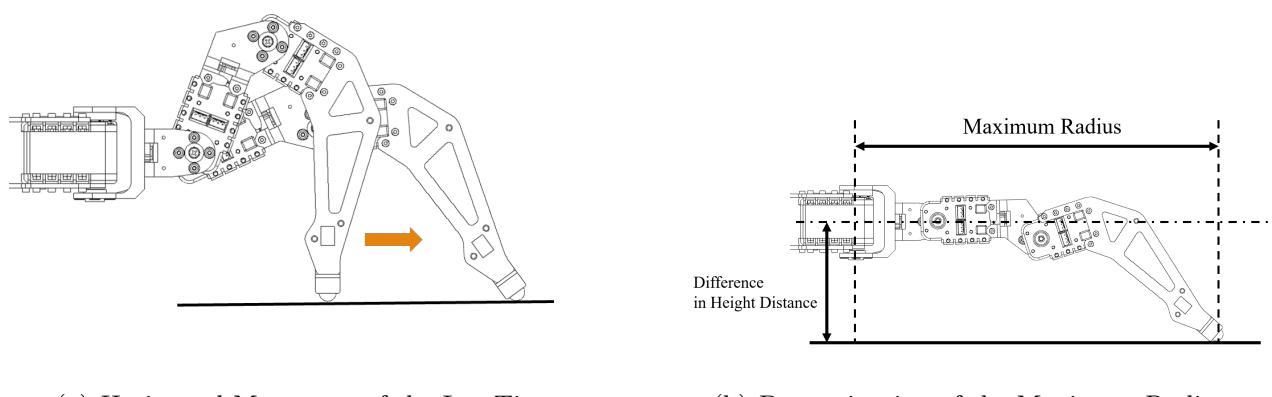
- (4) 脚先を  $1[mm]$  下げて (2) (3) の処理を繰り返す。脚先を下げることができなくなった場合、処理を終了する。

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。先行研究では実験で用いたロボットにあわせ、 $120[mm]$  とした。そして、扇形の中心角を隣の脚と干渉しないように決定する。こうして求められる近似された脚の可動範囲のイメージを Fig.2.8 に示す。Fig.2.8 では右中央の脚の近似された脚の可動範囲のみを示しており、赤い領域で示される図形が近似された脚の可動範囲である。

このように近似を行ったことで、以下に示す手順で脚先が脚の可動範囲内にあるか判定することができる。

- (1) 脚の付け根から脚先までの水平方向のベクトルを求め、ベクトルの傾きが扇形の中心角の範囲内にあるか判定する。
- (2) 脚の付け根から脚先までの水平方向の距離を求める。

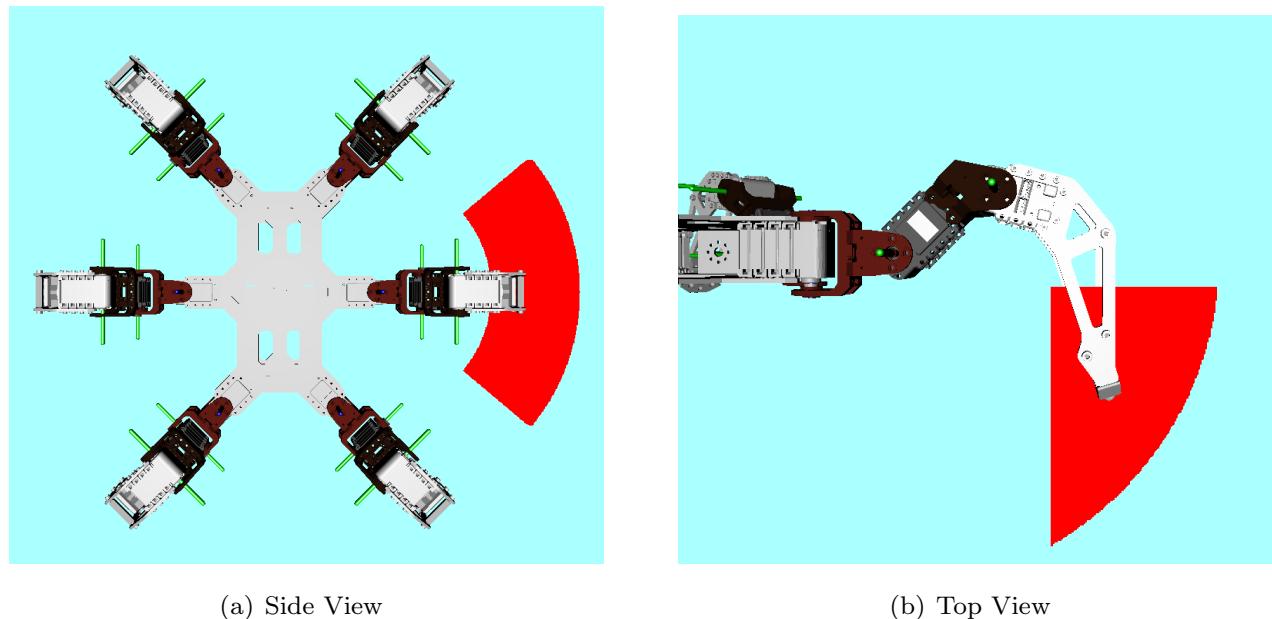
- (3) 最小半径よりも大きく、最大半径よりも小さいか判定する。
  - (4) 以上の条件を満たす場合、脚先は脚の可動範囲内にあると判定する。
- (2) (3) の手順は四則演算のみで計算が可能であり、(1) の手順もベクトルの内積を用いて計算が可能であるため同様に四則演算のみで計算が可能である。そのため、脚先が脚の可動範囲内にあるかの判定を高速に行うことができる。本手法では脚軌道生成だけでなく、脚の接地判定にもこの近似的な脚の可動範囲を用いている。



(a) Horizontal Movement of the Leg Tips

(b) Determination of the Maximum Radius

Fig. 2.7 Calculation of the Maximum Radius



(a) Side View

(b) Top View

Fig. 2.8 Approximated Range of Motion

### 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予想される。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。

失敗の原因についてより具体的に議論するために、脚の可動範囲の解析を行ったため、その方法と結果を示す。解析の対象とするロボットは、Trossen Robotics 社の PhantomX Mark II [19]（以下 PhantomX）とする。PhantomX (Fig.2.9) 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。

PhantomX の脚の可動範囲を求めるために、間接角度の逆運動解を求める式を導く。まず、Fig.2.10 に PhantomX の脚の座標系を示す。第 1 関節を原点とし、x 軸をロボットの前方方向にとり、z 軸をロボットに垂直で上向きにとる。y 軸は右手座標系となるように設定する。また簡単のため、解析には Fig.2.11 のように PhantomX のアクチュエータの回転軸を結んだ仮想的なリンクを用いる。リンク名は第 1 関節から Coxa Link, Femur Link, Tibia Link であり、関節名は Coxa Joint, Femur Joint, Tibia Joint である。リンク長は第 1 関節から第 3 関節までそれぞれ  $L_c$ ,  $L_t$ ,  $L_f$  とし、間接角度をそれぞれ  $\theta_c$ ,  $\theta_t$ ,  $\theta_f$  とする。間接の座標は  $P_c$ ,  $P_t$ ,  $P_f$  とし、脚先の座標を  $P_{end}$  とする。 $P_{end} = \{x_{end}, y_{end}, z_{end}\}$  の時のこのとき  $P_c$ ,  $P_f$ ,  $P_t$ ,  $\theta_c$ ,  $\theta_f$ ,  $\theta_t$  を求める式は式 (2.4) から式 (2.9) である。

$$P_c = \{0, 0, 0\} \quad (2.4)$$

$$\theta_c = \arctan \frac{y_{end}}{x_{end}} \quad (2.5)$$

$$P_f = \{L_c \cos \theta_c, L_c \sin \theta_c, 0\} \quad (2.6)$$

$$\theta_f = \arctan \frac{z_{end}}{\sqrt{x_{end}^2 + y_{end}^2} - L_c} + \arccos \frac{L_t^2 + L_f^2 - x_{end}^2 - y_{end}^2 - z_{end}^2}{2 \cdot L_t \cdot L_f} \quad (2.7)$$

$$P_t = \{(L_c + L_t \cos \theta_f) \cos \theta_c, L_t \cos \theta_f \sin \theta_c, L_t \sin \theta_f\} \quad (2.8)$$

$$\theta_t = \arctan \frac{z_{end} - z_t}{\sqrt{x_{end}^2 + y_{end}^2} - \sqrt{x_t^2 + y_t^2}} - \theta_f \quad (2.9)$$



Fig. 2.9 PhantomX Mark II

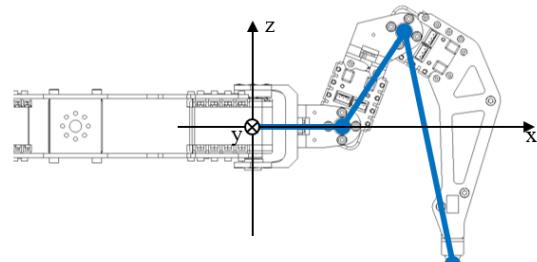
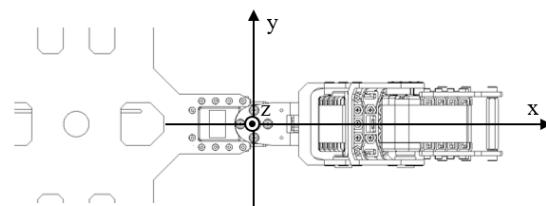


Fig. 2.10 Leg Coordinate Axis

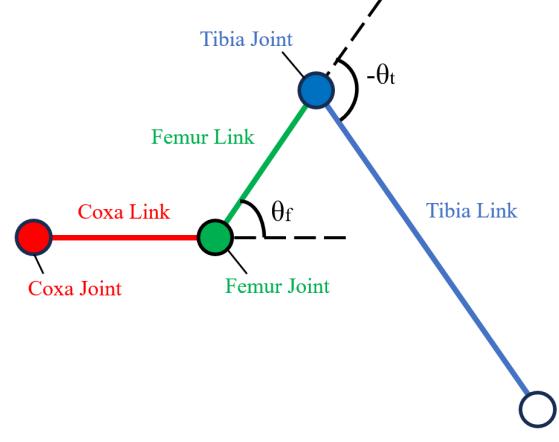


Fig. 2.11 Joint and Link Name

Table. 2.1 Link Length of PhantomX

Link Name	[mm]
Coxa Link	52
Femur Link	66
Tibia Link	130

以上の式を用いて、PhantomX の脚の可動範囲を求めたものが Fig.2.12 である。Fig.2.12 では、 $\theta_c = 0$ とした時の、x-z 平面における脚の可動範囲を示している。横軸を x 軸、縦軸を z 軸とし、単位は [mm] である。黒色の実線で示される領域が実際の脚の可動範囲であり、緑色の実線で示される領域が近似された脚の可動範囲である。赤色の点線で示される直線は遊脚の高さを表しており、この直線よりも高く上げることはない。青色の線と頂点で示される図形は脚の概形である。

Fig.2.12 より、 $100 < x < 150$ ,  $-50 < z < -20$  の範囲は、実際の脚の可動範囲と近似された脚の可動範囲が異なる領域になっているとわかる。この領域内の点を脚の接地点として選択してしまうと、脚の可動範囲外に脚先が出てしまうため、脚軌道を生成できず、脚が接地できないことによって転倒してしまう。また、この領域の点を脚の接地点として選択していくとも、脚軌道がこの領域を通過する場合、脚の可動範囲外に脚先が出てしまうため、矩形軌道の生成に失敗し、この領域を避けるような不完全な脚軌道が生成される。プログラムは矩形軌道を描くことを前提にしているため、このような不完全な脚軌道を生成すると、障害物に脚が引っかかってしまい歩行を継続できなくなる。以上から、この領域が脚軌道生成の失敗の原因であると予想できる。

Fig.2.12 は遊脚高さを 20[mm], 最小半径を 120[mm] とした時のものであるが、遊脚高さを 25[mm], 最小半径を 130[mm] とした時の脚の可動範囲を Fig.2.13 に示す。Fig.2.12 の条件は波東らの研究 [16] のシミュレーション実験時の条件であり、Fig.2.13 の条件は実機試験時の条件である。Fig.2.13 では、 $100 < x < 150$ ,  $-50 < z < -20$  の範囲の、実際の脚の可動範囲と近似された脚の可動範囲が異なる領域の大きさが小さくなっていることがわかる。実機試験時にあたり、条件を変更する必要があったことからも、近似値と実際の値の差が脚軌道生成失敗に影響していると予想できるだろう。

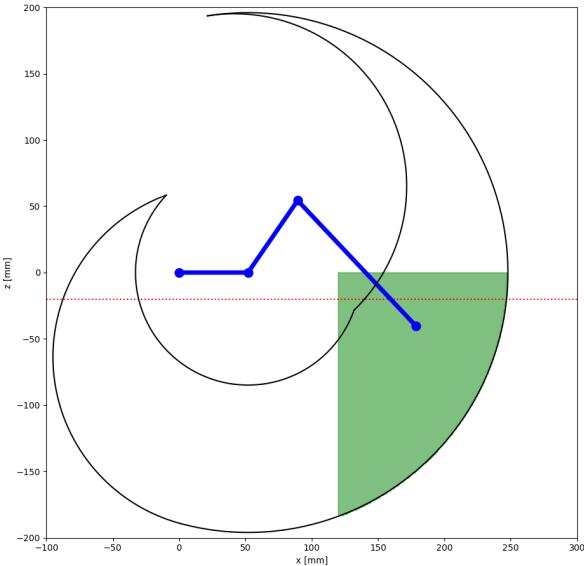


Fig. 2.12 Approximated Range of Motion in Simulation

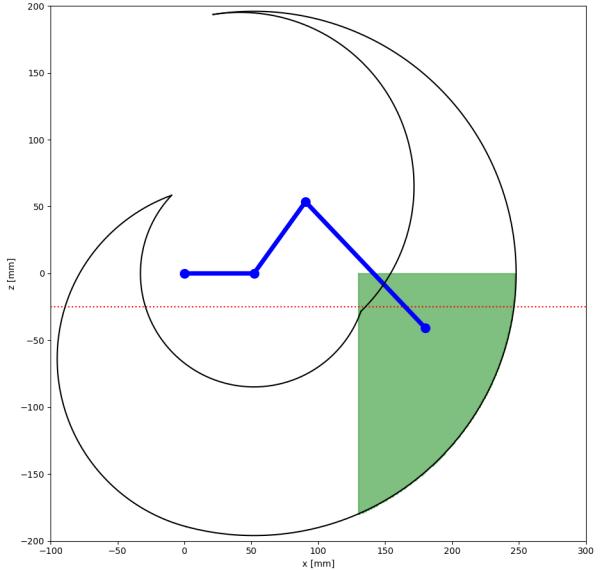


Fig. 2.13 Approximated Range of Motion in Experimental Equipment

## 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過することによるものであると考察されてきた。しかし、具体的にどのような理由で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う。実験の目的は脚軌道生成の失敗の回数、接地時の脚位置、そして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とする。

波東らは段差のある地形と斜面のある地形でシミュレーション、および実機による直進歩行動作の実験を行った。また、実機試験時にはシミュレーション実験と歩行時の条件を変更していた。本シミュレーションでは、波東らの研究のシミュレーションと実機試験それぞれの条件で歩行シミュレーションを行う。

## 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、Windows API 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.2.2 に示した。

### モデルとするロボット

モデルとするロボットは Fig.2.9 の PhantomX Mark II とする。

### 歩行する地形

歩行する地形を Fig.2.14 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。

### 歩行する時の条件

シミュレーション時の歩行条件は以下の通りである。

- ・ 脊体姿勢は常に水平である。
- ・ 最小半径を 120[mm] とする。
- ・ 重心から見た遊脚高さを -20[mm] とする。
- ・ 脊体を地形から最小 30[mm] 離す。
- ・ 常に静的安定余裕は 10[mm] 以上を保つ。

実機実験時の歩行条件は以下の通りである。

- ・胴体姿勢は常に水平である.
- ・最小半径を  $130[mm]$  とする.
- ・重心から見た遊脚高さを  $-25[mm]$  とする.
- ・胴体を地形から最小  $50[mm]$  離す.
- ・常に静的安定余裕は  $15[mm]$  以上を保つ.

以上の 2 つの条件で歩行シミュレーションを行う.

### シミュレーションの手順

これらの地形で水平方向に  $1200[mm]$  直進させ, Fig.2.14 のロボットの初期位置をランダムに変化させて計 5 回ずつ歩行させる.

このシミュレーションでは脚軌道生成に失敗した場合も, 仮に脚軌道生成に成功した場合と同様に歩行を継続させ, 脚軌道生成に失敗した場合の脚先の位置と回数を記録する. これを各地形, 各条件で行う.

Table. 2.2 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

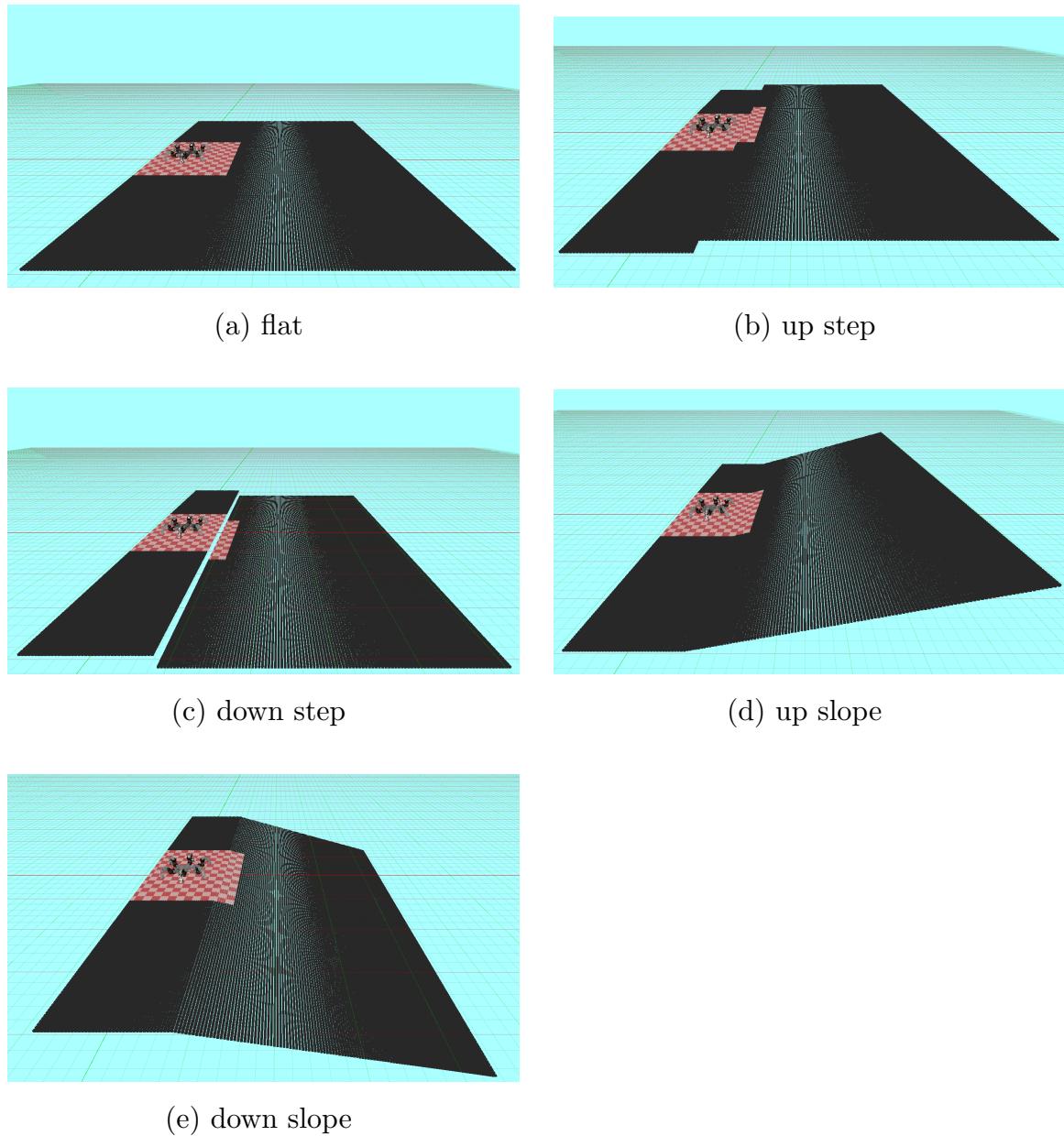


Fig. 2.14 Terrain

### 2.2.3 シミュレーションの結果

シミュレーション時の条件で歩行させたときの脚軌道生成失敗の回数を Table.2.3 に示す。また、実機実験時の条件で歩行させたときの脚軌道生成失敗の回数を Table.2.4 に示す。Table.2.3 では脚軌道生成の失敗率は多くの地形で 20[%] を超えており、動作を継続させることが難しいとわかる。とくに脚の接地点が脚の可動範囲外になってしまうことが多く、目標の地点まで脚先を届かせることができないため、転倒してしまう可能性が高い。Table.2.4 は脚軌道生成の失敗率は多くの地形で 3[%] 程度であり、Table.2.3 よりも著しく低いことがわかる。また、脚の接地点が脚の可動範囲外になってしまうことはなく、目標の地点まで脚先を届かせることができるため、転倒する可能性は低い。そのため、脚が地形に引っかかってしまわない限り、動作を継続させることが可能であるといえる。

Table. 2.3 Failure Count of Simulation

地形	グラフ探索 の回数	失敗の回数					失敗率 [%]	
		脚接地点が 可動範囲外	脚軌道が可動範囲外を通る			総失敗 回数		
			遊脚時	接地時	胴体平行 移動時			
平面	315	47	16	4	0	67	21.3	
上り斜面	460	63	11	22	1	97	21.1	
下り斜面	611	29	53	28	0	110	18.0	
上り段差	368	50	16	10	0	76	20.7	
下り段差	331	33	39	6	0	78	23.6	

Table. 2.4 Failure Count of Experimental Equipment

地形	グラフ探索 の回数	失敗の回数					失敗率 [%]	
		脚接地点が 可動範囲外	脚軌道が可動範囲外を通る			総失敗 回数		
			遊脚時	接地時	胴体平行 移動時			
平面	351	0	9	2	0	11	3.13	
上り斜面	645	0	10	1	0	11	1.71	
下り斜面	867	0	20	9	0	29	3.34	
上り段差	461	0	6	10	0	16	3.47	
下り段差	383	0	9	0	0	9	2.35	

シミュレーション時の条件で歩行させたとき、脚軌道生成に失敗した脚先の位置を Fig.2.15 に示す。また、実機実験時の条件で歩行させたとき、脚軌道生成に失敗した脚先の位置を Fig.2.16 に示す。どちらの図も Fig.2.12 と Fig.2.13 の  $50 < x < 250$ ,  $-200 < z < 0$  の範囲を拡大したものである。橙色の点は脚先が脚の可動範囲外になってしまった時の脚先の位置であり、青色の点は脚軌道が脚の可動範囲外を通ってしまった時の脚先の位置である。青色は始点、藍色は終点を表しており、明るい水色は中継点を表している。それぞれ、すべての地形での結果をまとめて図示している。この結果から、脚軌道生成に失敗した時の脚先の位置は、実際の脚の可動範囲と近似された脚の可動範囲が異なる領域を通っていることがわかる。

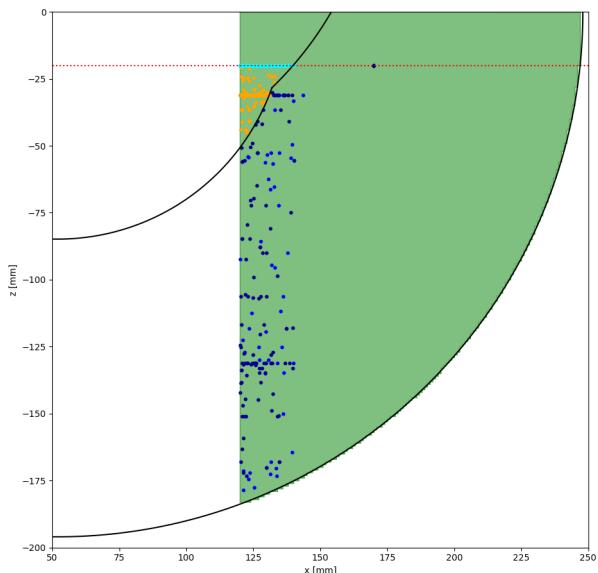


Fig. 2.15 Result of Simulation

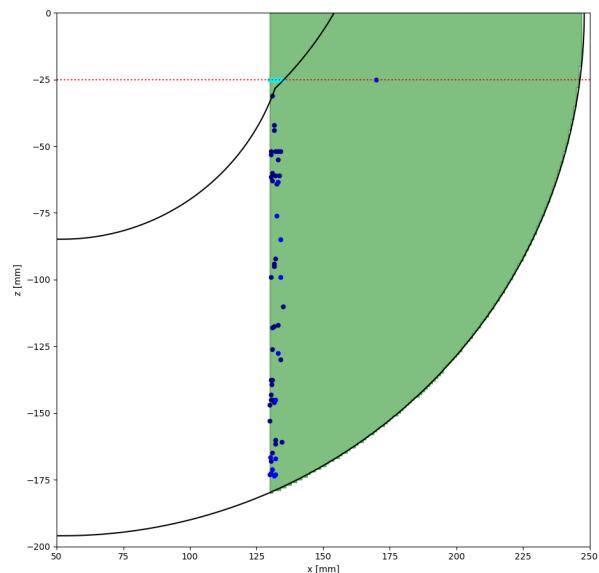


Fig. 2.16 Result of Experimental Equipment

## 2.2.4 脚軌道生成に失敗する原因の考察

脚軌道生成に失敗した時の脚先の位置はすべて、実際の脚の可動範囲と近似された脚の可動範囲が異なる領域を通っている。よって脚軌道生成に失敗する原因是、2.1.3 章で予想した通り、実際の脚の可動範囲と近似された脚の可動範囲が異なる領域によるものであるといえる。この領域は最小半径を  $140[mm]$  まで増やすことで、なくすことが可能であるため、脚軌道生成の失敗を防ぐためには最小半径の再設定が必要であると考えられる。

## 2.3 歩容パターンの再評価手法

脚軌道生成の失敗を防ぐためには近似された脚の可動範囲を小さくするか、より正確に近似することで誤差をなくせばよい。しかし、前者は脚の接地点として選択される領域を縮めてし

まうことになるため、グラフ探索による手法の利点である効率的な動作ができなくなってしまうことや、本来歩行可能であった地形を歩くことができなくなってしまうことなどの問題点がある。また、後者は細かく正確に近似するとキャッシュしておく値が増えるため、あらかじめ計算しておいた値にアクセスする際の呼び出しにかかる時間が長くなってしまう。これにより、計算時間が長くなってしまう問題点がある。

先行研究では、シミュレーション時と実機試験時の条件を変更していたとおり、前者の方法でこの問題に対応をしていた。しかし Table.2.5 の歩行条件と、それを適用したときの歩行可能な地形を示した表からもわかるように、上り段差と下り段差では歩行可能な地形が変わってしまい、シミュレーションで歩くことができた地形を歩くことができなくなってしまった。加えて、この上でも脚軌道生成に失敗してしまうことがあったため、根本的にこの問題を解決するための方法を考える必要がある。

そこで、歩容パターングラフ探索の再評価を行うことで、脚軌道生成の失敗を防ぐ手法を提案する。歩容パターングラフ探索の再評価とは、脚軌道生成に失敗した場合、グラフ探索の処理をやり直して、脚軌道生成に失敗しないような歩容パターンを生成することと定義する。グラフ探索の再評価の手順を Fig.2.17 に示す。再評価手法では基本的には従来手法通りにグラフ探索による歩容パターン生成を行い、脚軌道生成を行ってロボットを動作させる。脚軌道生成に失敗した場合のみ、グラフ探索による歩容パターン生成をやり直して、脚軌道生成に失敗しない新しい歩容パターンを生成する。歩容パターン生成をやり直す際、脚軌道生成に失敗しないよう、脚の近似された可動範囲を狭め、可動範囲外に接地することができないようにする。こうして歩容パターングラフを再生成し、グラフ探索を再度行うことで、脚軌道生成に失敗しないノードを取得することが可能となる。

再評価手法は前述した 2 つの解決法と比べ、それらのもつ問題点を克服している。以下にそれぞれの解決策と比較した再評価手法の利点を示す。

### 本来接地可能だった地点を選択できなくなってしまう問題

脚の可動域をある程度広く確保することが可能になり、可動範囲の境界に近い点を接地点として選択できるようになる。加えて、脚軌道生成に失敗する場合、狭められた可動域に切り替えて失敗を防ぐことができる。

### 値を呼び出す際にかかる時間が増えてしまう問題

再評価手法では歩容パターン生成をやり直す都合上、単純計算で探索にかかる時間が倍になってしまう。よってより正確に近似する方が実行時間的に優れると思える。しかし、歩容パターングラフの生成時には  $10^5 \sim 10^6$  程度の数のノードを生成するため、グラフ探索内の処理の時間の増加は、最終的な計算時間を大幅に増加させてしまう可能性がある。つまり再評価手法では計算時間の増加を約 2 倍程度で済ませることができる

といえる。

Table. 2.5 Walkable Terrain

地形	シミュレーション時	実機実験時
上り段差	130[mm]	100[mm]
下り段差	-130[mm]	-100[mm]
上り斜面	15[deg]	15[deg]
下り斜面	-15[deg]	-15[deg]

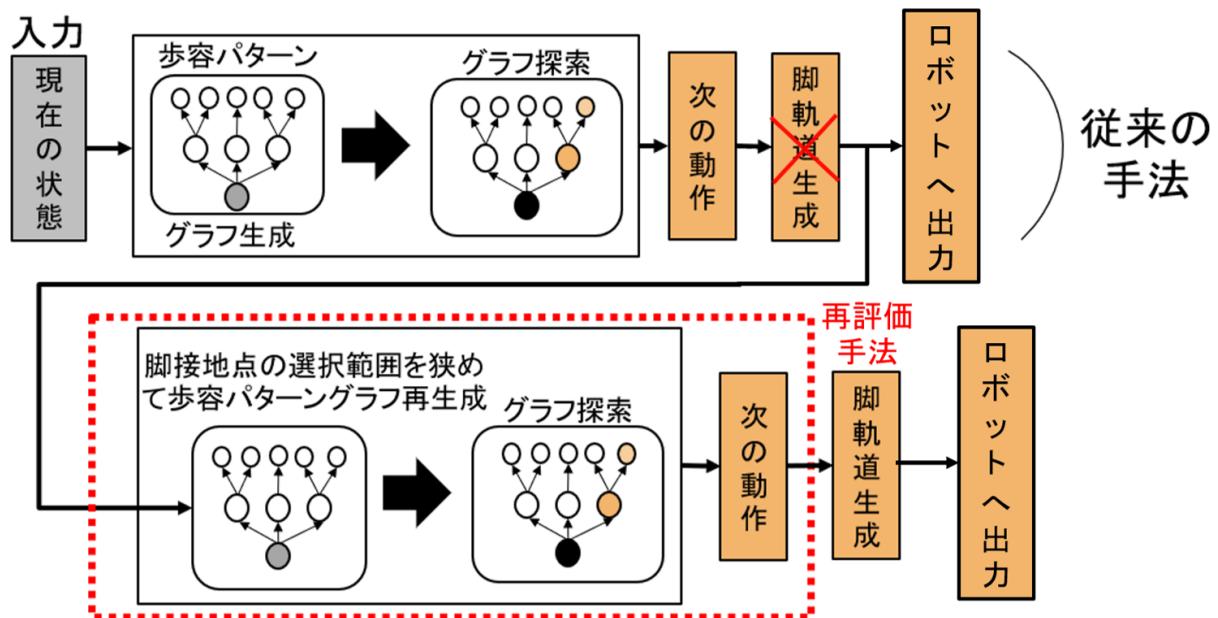


Fig. 2.17 Revaluation Methodology Procedures



## 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、3次元空間における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。波東らが用いた自由歩容パターン生成手法による直進動作と同様の手法によって歩容パターンを生成しているが、より早い処理の実現やプログラムの可読性・拡張性の向上のため、その実装方法を1部変更している。加えて、新たに3次元空間における旋回動作の実装と先行研究の手法の統合を行ったため、変更点を含め、改めて実装方法を説明する。

プログラムの実装はC++20を用いて行っており、命名規則はGoogle C++ Style Guide[20]にしたがっている。開発にはVisual Studio 2022を用いており、プログラムのビルドにはVisual Studio 2022のMSVCコンパイラを用いている。

また、この章におけるクラスや構造体の図はUML(Unified Modeling Language)を用いて表現されており、上からクラス・構造体の名前、メンバ変数、メンバ関数を表している。メンバ変数は、アクセス修飾子、変数名、型の順に記述されており、メンバ関数は、アクセス修飾子、関数名、引数、戻り値の順に記述されている。アクセス修飾子は記号を用いて表し、+はpublic、protectedは#、privateは-である。クラス間の関係は、継承関係は空白の三角形、包含関係は空白の菱形で表している。包含関係は、クラスのメンバ変数として他のクラスを持つことである。

### 3.1.1 プログラム全体の流れ

グラフ探索による自由歩容パターン生成手法では、歩容パターングラフの作成、歩容パターングラフの探索の2つの処理を行う。歩容パターングラフを作成する処理は GraphTreeCreator クラスで実装されており、歩容パターングラフの探索を行う処理は GraphSearcher クラスで実装されている。

これらの処理の流れを Fig.3.1 に示す。まず、GraphTreeCreator クラスに現在のロボットの状態を表すノードを渡す。GraphTreeCreator クラスは、渡されたノードを根ノードとする歩容パターングラフを作成する。次に、GraphSearcher クラスに作成された歩容パターングラフを渡す。このとき、値をコピーして渡すのではなく、ポインタを渡すことでメモリの使用量を削減するとともに、グラフ探索の処理時間を短縮している。GraphSearcher クラスは、渡された歩容パターングラフを探索し、最適な歩容パターンを見つけ、その歩容パターンを返す。以上の流れでグラフ探索による自由歩容パターン生成手法を実装している。

Fig.3.2 にグラフ探索による自由歩容パターン生成手法のクラス図を示す。GaitPatternGeneratorBasic クラスは、グラフ探索による自由歩容パターン生成手法を実装したクラスである。GaitPatternGeneratorBasic クラスは、IGaitPatternGenerator インターフェイスを継承しており、GetNextNodeByGraphSearch 関数を持っている。GetNextNodeByGraphSearch 関数は、現在のロボットの状態を表すノードを引数、地形の情報、目標位置・目標姿勢を引数に取り、グラフ探索によって最適な歩容パターンを返す。IGaitPatternGenerator インターフェイスを介することで、後述する再評価手法の実装や、グラフ探索による自由歩容パターン生成手法の切り替えを容易にしている。メンバ変数に GraphTreeCreator クラスのポインタと、IGraphSearcher インターフェイスを継承したクラスのポインタをもっており、GaitPatternGeneratorBasic クラスのコンストラクタでそれぞれ初期化される。GraphTreeCreator クラスは、歩容パターングラフの作成を行うクラスである。メンバ変数に NodeCreator クラスのポインタを持っており、行う動作によって異なる INodeCreator クラスを用いてグラフを作成する。IGraphSearcher インターフェイスは、グラフ探索を行う処理を表すインターフェイスである。行う動作によって異なる探索方法が必要なためインターフェイスを作成している。

以降は歩容パターングラフの作成、歩容パターングラフの探索を行う処理について説明する。

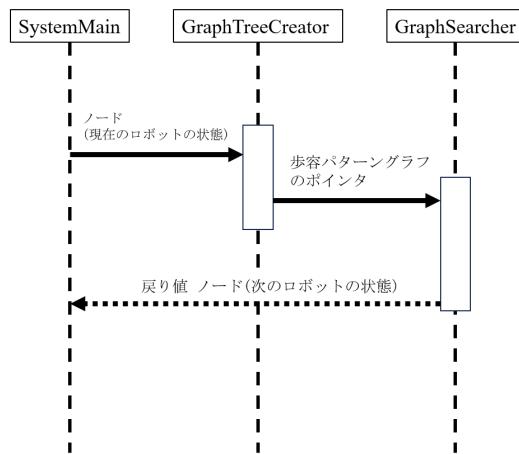


Fig. 3.1 Graph Search Sequence

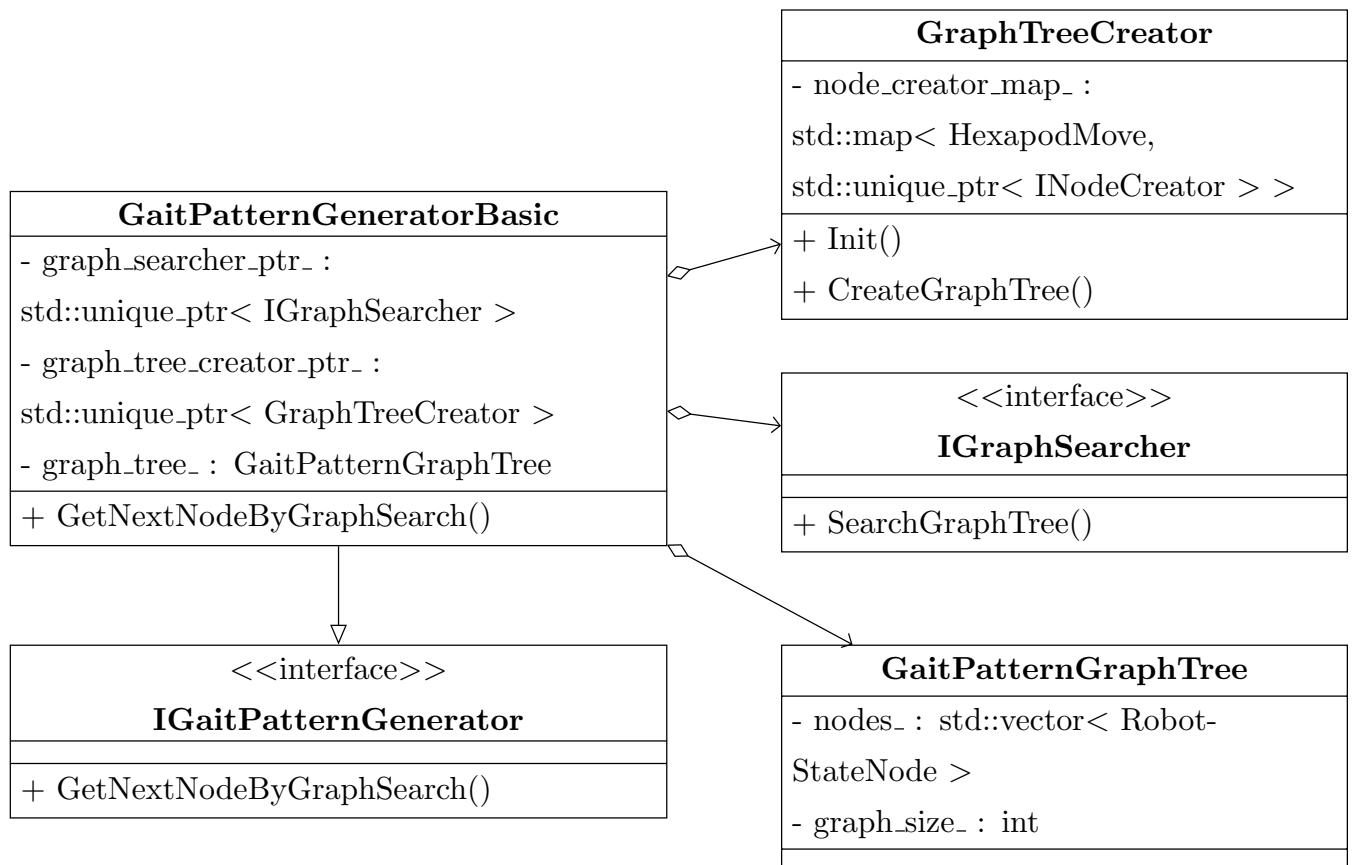


Fig. 3.2 Graph Search Class Diagram

### 3.1.2 ノードを表現する構造体

次に、歩容パターングラフを作成する処理を記述するために、ノードについて説明する。歩容パターングラフのノードは Fig.3.3 のような、ロボットの状態を表す構造体 RobotStateNode で表現した。歩容パターングラフでは RobotStateNode を配列とすることで作成するため、RobotStateNode 構造体はサイズを小さくし、メモリの使用量を少なくすることが求められる。

RobotStateNode のメンバ変数 leg\_state は、ロボットの脚の状態を表すビット列である。C++ の標準ライブラリの std::bitset を用いて実装されており、28 ビットの長さを持つ。ビット列の各ビットは、Fig.3.6 のように定義されている。下位 24bit は各脚の離散化された脚位置と遊脚状態を表し、上位 4bit は後述するロボットの重心位置を表す。このようにすることで複数のパラメータを 1 つの変数のみで表現することができ、メモリの使用量を削減することができる。

leg\_pos と leg\_reference\_pos は、ロボットの脚の位置を表す Vector3 構造体の配列である。Vector3 構造体は、3 次元ベクトルを表す構造体であり、Fig.3.4 のように定義されている。leg\_pos はロボットの脚の現在の位置を表し、leg\_reference\_pos は離散化された脚位置の脚位置 4 の位置を表す。座標系は脚の付け根を原点とするローカル座標系であり、単位は mm である。

center\_of\_mass\_global\_coord と posture は、グローバル座標におけるロボットの重心位置と姿勢を表す Vector3 構造体と Quaternion 構造体である。Quaternion 構造体は、クォータニオンを表す構造体であり、Fig.3.5 のように定義されている。

next\_move は、次に行う動作を表す HexapodMove 列挙型である。先行研究では int 型で表現していたが、可読性の向上のために列挙型を用いている。parent\_index は、親ノードのインデックスを表す int 型である。先行研究では親ノードへのポインタを用いていたが、自身の実行環境ではポインタのサイズは 8byte であるため、4byte の int 型を用いることでメモリの使用量を削減している。depth は、ノードの深さを表す int 型である。RobotStateNode は以上のように定義されており、188byte のサイズである。

このノードを用いて歩容パターングラフを作成する際、根ノードは parent\_index を -1、depth を 0 に設定する。その後、根ノードの next\_move を基に子ノードを作成し、parent\_index を根ノードのインデックス、depth を 1 に設定する。こうして作成した子ノードの next\_move を基に、さらに子ノードを作成することで歩容パターングラフを作成できる。歩容パターングラフは RobotStateNode 構造体の配列で表現できるが、処理を簡単にするため GaitPatternGraphTree クラスでラッパーしている。

RobotStateNode 構造体と Vector3 構造体、Quaternion 構造体はそれぞれメンバ変数を操

作するためのメンバ関数を持っているが、グラフ探索の処理とは直接的な関係がないため、ここでは説明を省略する。

<b>RobotStateNode</b>
+ leg_state : std::bitset<28>
+ leg_pos : std::array<Vector3, 6>
+ leg_reference_pos : std::array<Vector3, 6>
+ center_of_mass_global_coord : Vector3
+ posture : Quaternion
+ next_move : HexapodMove
+ parent_index : int
+ depth : int

Fig. 3.3 RobotStateNode Struct

<b>Vector3</b>
+ x : float
+ y : float
+ z : float

Fig. 3.4 Vector3 Struct

<b>Quaternion</b>
+ w : float
+ v : Vector3

Fig. 3.5 Quaternion Struct

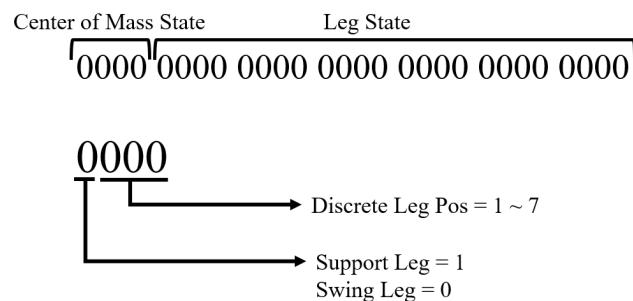


Fig. 3.6 Leg State Bit

### 3.1.3 目標位置・目標姿勢を表現する構造体

ロボットの目標位置・目標姿勢を表現する RobotOperation 構造体を Fig.3.7 に示す。メンバ変数の operation\_type\_ はロボットの動作を表す列挙型、RobotOperationType 型の変数である。この変数の値によってロボットの行うべき動作が決定される。これはつまり、グラフ探索において高く評価されるノードが変更されるということである。operation\_type\_以外の変数は具体的にどのような動作を行うかを表し、operation\_type\_の値によって使用する変数が決定される。

operation\_type\_ の値と使用する変数の対応を Table.3.1 に示す。直進動作を行う場合、指定することができるのは kStraightMoveVector と kStraightMovePosition の 2 つである。kStraightMoveVector は、ロボットの進行方向を単位ベクトルの straight\_move\_vector\_ で指定する。kStraightMovePosition は、ロボットの目標位置を straight\_move\_position\_ で指定する。

その場旋回動作を行う場合、指定することができるのは kSpotTurnLastPosture と kSpotTurnRotAxis の 2 つである。kSpotTurnLastPosture は、ロボットの旋回前の姿勢を spot\_turn\_last\_posture\_ で指定する。kSpotTurnRotAxis は、ロボットの旋回軸を spot\_turn\_rot\_axis\_ で指定し、その軸周りに右ねじの方向に旋回する。

RobotOperation
+ operation_type_ : RobotOperationType
+ straight_move_vector_ : Vector3
+ straight_move_position_ : Vector3
+ spot_turn_last_posture_ : Quaternion
+ spot_turn_rot_axis_ : Vector3

Fig. 3.7 RobotOperation Struct

Table. 3.1 RobotOperationType Enum

要素 動作	RobotOperationType
直進	kStraightMoveVector
	kStraightMovePosition
その場旋回	kSpotTurnLastPosture
	kSpotTurnRotAxis

### 3.1.4 地形を表現する構造体

地形を表現する MapState 構造体を Fig.3.8 に示す。グラフ探索においては連続的な地形を離散化する必要があるため、地形を離散化するための点群を表す配列 map\_point\_ をメンバ変数を持っている。map\_point\_ は Vector3 構造体の配列であり、脚の接地可能点を表す。map\_point\_ は地形を上から見た時、格子状に配置されており、隣の脚接地可能点との間隔は一定で 20[mm] である。この間隔はロボットの脚先の面積を考慮して決定した。

MapState 構造体を用いることで地形を離散化することができるが、依然として脚接地可能点の数が多い。そのため、ロボットを中心に地形を分割し、各分割領域における脚接地可能点を表す DividedMapState 構造体を作成した。Fig.3.9 において、赤い点で表される脚接地可能点が DividedMapState 構造体で表現されている。DividedMapState 構造体は、グローバル座標におけるロボットの重心位置を表す global\_robot\_com\_ を中心に地形を分割した際、各分割領域における脚接地可能点を表す divided\_map\_point\_ をメンバ変数を持っている。また、各分割領域の最上点の高さを表す divided\_map\_top\_z\_ をメンバ変数を持っている。脚の接地判定を行う際は、DividedMapState 構造体の中から脚先に近い領域群を選択し、その領域群における脚接地可能点を用いて接地判定を行うことで、計算する脚接地可能点の数を減らしている。

### 3.1.5 子ノードを作成するプログラム

歩容パターングラフを作成するためには、子ノードを作成するプログラムが必要であり、そのプログラムを INodeCreator インターフェイスを実装した各クラスで実装している。INodeCreator インターフェイスは Fig.3.10 のように定義されている。Create 関数は、引数に現在のノード、現在のノードのインデックス、作成した子ノードを格納する配列へのポイン

<b>MapState</b>
+ map_point_ : std::vector < Vector3 >

<b>DividedMapState</b>
+ global_robot_com_ : Vector3
+ divided_map_point_ :
std::vector < std::vector < Vector3 > >
+ divided_map_top_z_ : std::vector < float >

Fig. 3.8 MapState Struct

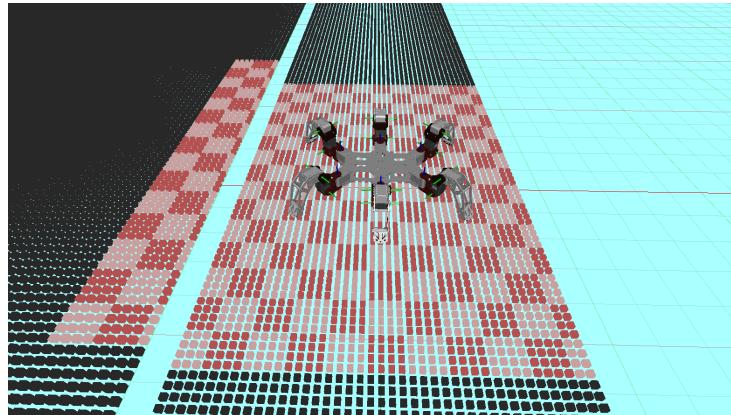


Fig. 3.9 Map View

タを持っている。戻り値で結果を返す場合、RobotStateNode 構造体のコピー処理が発生するため、引数で渡した配列のポインタを通じて値を返している。各クラスでは、Create 関数をオーバーライドすることで、子ノードを作成する処理を実装している。

歩容パターングラフの規模を小さくするためには1つの親ノードがもつ子ノードの数は少なくする必要がある。しかし、数を少なくしすぎると、グラフ探索によって必要な歩容パターンが消えてしまう可能性がある。各クラスでは1つの親ノードから10個程度の子ノードを作成するようにしており、深さ5の歩容パターングラフのノード数はが $10^5 \sim 10^6$ 程度となるようにしている。ロボットの動作に応じて異なるクラスを用いたため、以下に各クラスの説明を記述する。

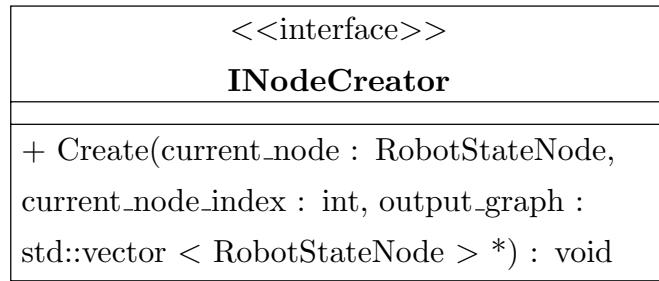


Fig. 3.10 NodeCreator Interface

### 重心の上下移動

重心の上下移動を行うための処理は NodeCreatorComUpDown クラスで実装している。NodeCreatorComUpDown クラスは処理を行うノードから、重心を上下移動させることで作成できる子ノードを戻り値として返す。

重心の移動後の座標は無数に存在するため、Fig.3.11 に離散化の様子を示した。まず、現在の脚位置から下げることが可能な最低の重心高さと、現在の脚位置から上げることが可能な最大の重心高さを計算する。この時、近似された脚の可動範囲から脚先が届くかを判定しており、また DevidedMapState の divided\_map\_top\_z\_ を用いて、地形との干渉を考慮している。次に、最低の重心高さから最大の重心高さまで等間隔で 5 分割し、各重心高さにおける子ノードを作成する。重心の高さを変更しない場合を考慮して、そのままの重心高さを持つ子ノードも作成する。離散化時の分割数はグラフの規模を考慮して決定した。

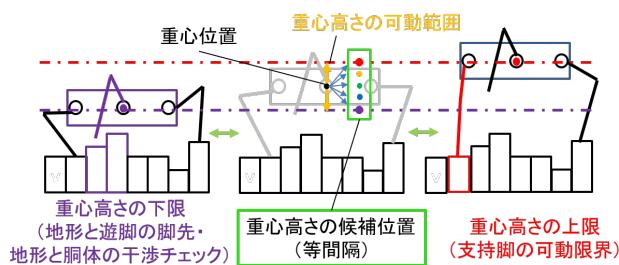


Fig. 3.11 Discretization of the Vertical Shift of the Center of Mass

### 重心の平行移動

重心の平行移動を行うための処理は NodeCreatorComMove クラスで実装している。NodeCreatorComMove クラスは処理を行うノードから、重心を平行移動させることで作成できる子ノードを戻り値として返す。

上下移動の場合と同様に、重心の移動後の座標は無数に存在するため以下の手順で離散化を行う。まず重心を平行移動させる候補領域を Fig.3.12 のように決定する。脚先を投影してできる六角形の対角線をすべて結び、その交点から 7 通りの重心の候補領域を決定する。そして、それぞれ右前方にあるものから順に 1~6 と番号を振る。また、中央にあるもののうち、ロボットの前方を上としたときに逆三角形となるものを 7、ロボットの後方を上としたときに逆三角形となるものを 8 とする。

次にこれらの候補領域から実際に重心を平行移動させる地点を決定する。Fig.3.13 に示すように、候補領域を格子状に分割する。そして、各格子点において重心を平行移動させた時、もっとも進行方向への移動量が大きくなる点を選択し子ノードの重心座標とする。この時に静的安定余裕を確認し、規定された値を下回る場合はその点を選択しない。また、このとき RobotStateNode 構造体の leg\_state\_ の上位 4bit に、候補領域の番号を離散化された重心位置として格納する。加えて、RobotStateNode 構造体の leg\_reference\_pos\_ を現在の leg\_pos\_ の値に更新する。

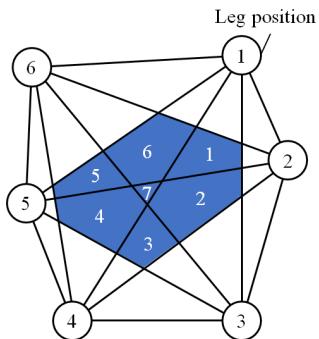


Fig. 3.12 Candidate Area of Center of Mass

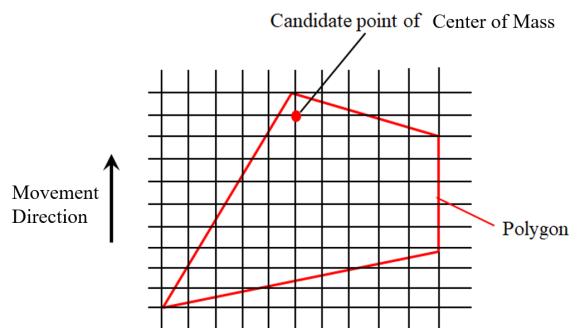


Fig. 3.13 Determination of Center of Mass

### 胴体の回転運動

胴体の回転運動を行うための処理は NodeCreatorBodyRot クラスで実装している。NodeCreatorBodyRot クラスは処理を行うノードから、胴体を回転させることで作成できる子ノードを戻り値として返す。

重心を中心として、重力の方向を軸とした回転運動を行う。回転量は  $-20 \sim 20[\text{deg}]$  の範囲を  $2[\text{deg}]$  刻みで離散化しており、1 つのノードにつき 21 個の子ノードを作成する。回転運動を行う際には、近似された脚の可動範囲から脚先が届くかを判定しており、届かない場合、そのノードは作成しない。

### 脚の上下運動・水平運動

第2章で述べたように、脚の水平運動は脚の状態が等しく、別の階層にあるノード間での遷移で行うことができ、脚の上下運動は同じ階層にあるノード間での遷移で行うことができる。脚の状態が等しく、別の階層にある子ノードを生成する処理は NodeCreatorLegHierarchy クラスで実装しており、同じ階層にある子ノードを生成する処理は NodeCreatorLegUpDown クラスで実装しているため、まずは、NodeCreatorLegHierarchy クラスの説明を行う。

NodeCreatorLegHierarchy クラスでは、処理を行うノードの脚の状態を表す leg\_state\_から、遊脚している脚を取得し、その脚の離散化された脚位置を 1 ~ 7 に変更した子ノードを作成する。つまり 1 脚遊脚している場合は 7 つの子ノードを作成し、3 脚遊脚している場合は  $7^3 = 343$  の子ノードを作成する。このクラスでは実際の脚位置を変更するのではなく、あくまで脚の状態を表す leg\_state\_を変更するのみである。

次に NodeCreatorLegUpDown クラスを説明する。NodeCreatorLegUpDown クラスでは、処理を行うノードの脚の状態を表す leg\_state\_から、遊脚の組み合わせを変更した子ノードを作成する。遊脚の組み合わせのうち、接地しえない脚がある組み合わせは作成しないようにするため、最初に現在遊脚中の各脚について、その脚が離散化された脚位置で指定された脚位置に接地することができるかを判定する。また、重心の位置から静的安定性を保つことができない場合も作成しない。処理の流れを以下に示す。

- (1) 離散化された重心位置から同時に遊脚することができない隣り合う 2 脚を遊脚するノードを削除する

RobotStateNode の leg\_state\_から重心位置を取得し、その重心位置から同時に遊脚することができない隣り合う 2 つの脚を、同時に遊脚するノードを作成する候補から削除する。Table.3.2 にある重心位置において、同時に遊脚することができない脚の組み合わせを示す。この表における重心位置や脚の番号は Fig.3.12 のものと同じである。

- (2) 離散化された脚位置で指定された位置に接地可能な点があるか確認する

各脚ごとに RobotStateNode の leg\_state\_から離散化された脚位置を取得し、その脚位置で指定された脚位置に接地可能な点があるかを確認する。接地可能な点がない場合、その脚を接地するノードを作成する候補から削除する。接地可能な点が複数ある場合はもっとも移動方向への移動量が大きくなる点を選択する。

- (3) 子ノードを作成する

(1) (2) の処理の中で候補から削除されなかった、階層内のノードを作成し、これらを子ノードとして返す。

Table. 3.2 Leg Combination Table

重心位置	遊脚可能な組	遊脚不可能な組
1	(3,4) (4,5) (5,6)	(6,1) (1,2) (2,3)
2	(4,5) (5,6) (6,1)	(1,2) (2,3) (3,4)
3	(5,6) (6,1) (1,2)	(2,3) (3,4) (4,5)
4	(6,1) (1,2) (2,3)	(3,4) (4,5) (5,6)
5	(1,2) (2,3) (3,4)	(4,5) (5,6) (6,1)
6	(2,3) (3,4) (4,5)	(5,6) (6,1) (1,2)
7	(2,3) (4,5) (6,1)	(1,2) (3,4) (5,6)
8	(1,2) (3,4) (5,6)	(2,3) (4,5) (6,1)

### 直進動作時におけるノード生成の順番

直進動作を行う時の、未探索のノードから遷移可能なノードをグラフに追加する際のルールについて説明する。RobotStateNode の next\_move\_には、次の動作を表す HexapodMove 型の変数が格納されている。この変数の値によってどの NodeCreator クラスを使用するかが決定される。それぞれの NodeCreator クラスは子ノードを作成する際に、親ノードの next\_move\_の値から子ノードの next\_move\_の値を以下のルールに基づいて決定する。

- ・ 親ノード：脚の水平運動 → 子ノード：脚の上下運動
- ・ 親ノード：脚の上下運動 → 子ノード：重心の上下移動
- ・ 親ノード：重心の上下移動 → 子ノード：重心の平行移動
- ・ 親ノード：重心の平行移動 → 子ノード：脚の水平運動

このようなルールを定めることによって明らかに無意味な動作（たとえば、重心を上げる→重心を下げる動作を繰り返すものなど）を生成することをあらかじめ防ぐことができる。なお、直進動作以外の動作を行う場合は別のルールを用いてグラフを作成している。

#### 3.1.6 グラフ探索時のノードの評価方法

グラフ探索時では歩容パターングラフのもっとも深いノードをすべて比較し、最高評価となったノードへのパスから深さ 1 のノード次のノードとして返す。RobotOperation で与えられた動作に対して、適切な動作を行う RobotStateNode を評価するために、グラフ探索時には複数のノードの評価項目がある。IGraphSearcher を継承した具象クラスではノードを評価

する関数を複数持っており、それぞれの関数で評価項目を計算している。この章では3次元空間の直進動作におけるノードの評価方法を説明する。

3次元空間の直進動作のグラフ探索はGraphSearcherStraightMoveクラスで実装している。処理の流れを以下に示す。

#### (1) 高さの評価

進行方向の地形の最大高さを確認し、その地点を歩行するために必要な最低の重心高さを求める。求めた重心高さと現在の重心高さの差を評価値として使用し、値が小さいほど評価を高くする、現在の最高評価ノードと比較して評価が高ければ最高評価ノードを更新する。

#### (2) 移動量の評価

(1) が最大評価ノードと同じであれば、直進動作の移動量を評価する。まず、根ノードから評価を行うノードまでの移動量を計算する。目標方向と移動量の内積を計算し、その値が大きいほど評価を高くする。RobotOperationで目標位置が指定されている場合は、根ノードから目標位置までの移動ベクトルを正規化したもの目標方向として使用する。現在の最高評価ノードと比較して評価が高ければ最高評価ノードを更新する。

#### (3) 脚の回転角などの平均値の評価

(2) も最大評価ノードと同じであれば、脚の回転角度の平均値を評価する。根ノードと評価を行うノードの脚の回転角度の平均値を計算し、その値が大きいほど評価を高くする。現在の最高評価ノードと比較して評価が高ければ最高評価ノードを更新する。

## 3.2 歩容パターンの再評価手法の実装

歩容パターンの再評価手法はIGaitPatternGeneratorインターフェイスを継承したクラスを作成することで実装できる。IGaitPatternGeneratorインターフェイスを用いたことで、デコレーターパターンを用いて容易に再評価手法を実装している。デコレーターパターンとは、既存のクラスに新たな機能を追加するためのデザインパターンである。デコレータ（デコレータパターンを用いて作られたクラス）はあるインターフェイスを継承するかつ、自身と同じインターフェイスを継承したクラスをメンバ変数として持つ。そしてメンバ関数を呼び出す際に、メンバ変数のメンバ関数と自身の持つ処理を実行することで、既存のクラスをそのままに新たな機能を追加することができる。

再評価手法はGaitPatternGeneratorRevaluationクラスで実装されている。GaitPatternGeneratorRevaluationクラスはIGaitPatternGeneratorインターフェイスを継承しており、メンバ変数にIGaitPatternGeneratorインターフェイスのポインタを2つ持っている。これらのポインタはコンストラクタで初期化され、1つ目のポインタは再評価手法を適用する前の

歩容パターン生成手法を、2つ目のポインタは再評価手法を適用した後の歩容パターン生成手法を指している。

メンバ関数である GetNextNodeByGraphSearch 関数が呼ばれた場合、まず、1つ目のポインタの GetNextNodeByGraphSearch 関数を呼び出し結果を取得する。取得したノードから脚軌道生成を行い、成功した場合はそのノードを返す。失敗した場合は、2つ目のポインタの GetNextNodeByGraphSearch 関数を呼び出し結果を取得する。こうすることで、第 2 章で提案した、脚軌道生成に失敗した場合のみグラフ探索をやり直す処理を実装できる。

再評価時には脚軌道生成の失敗を防ぐために、近似された脚の可動範囲を狭める。近似された脚の可動範囲は第 2 章で述べたように、最小半径を 140[mm] に設定すればよいため、狭めた前と後の脚の可動範囲は以下のようないい条件に設定する。また、それぞれを図示したものを Fig.3.15 と Fig.3.16 に示す。

#### (1) 再評価前

- 最小半径を 130[mm] に設定
- 最大半径は第 2 章で述べた計算方法で設定
- 遊脚高さは -20[mm] に設定

#### (2) 再評価後

- 最小半径を 140[mm] に設定
- 最大半径は第 2 章で述べた計算方法で設定
- 遊脚高さは -20[mm] に設定

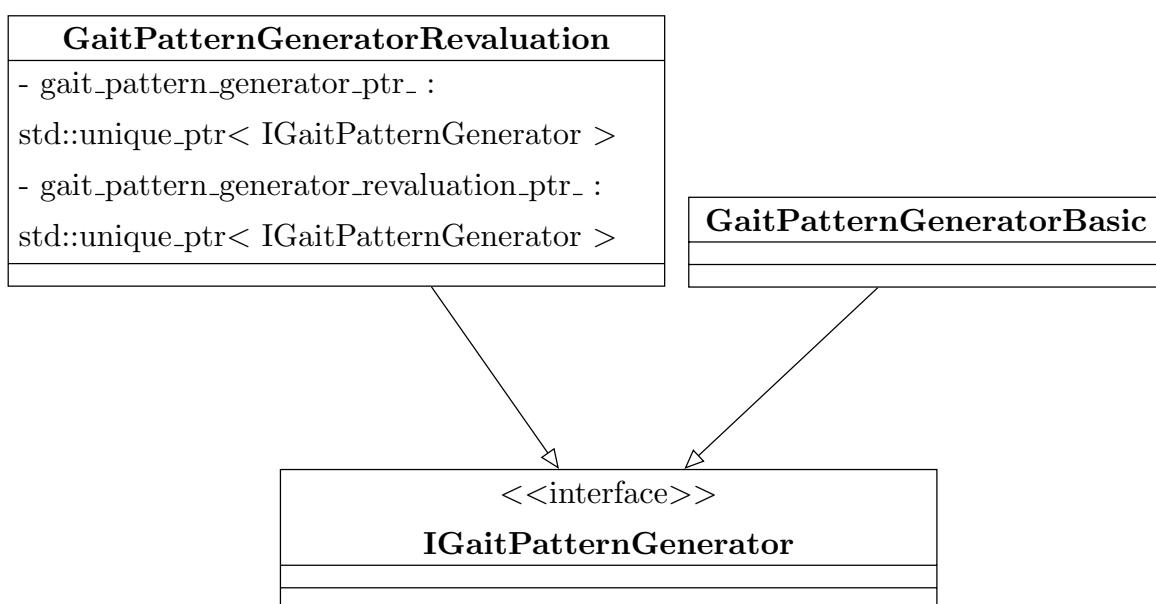


Fig. 3.14 Gait Pattern Generator Revaluation Class Diagram

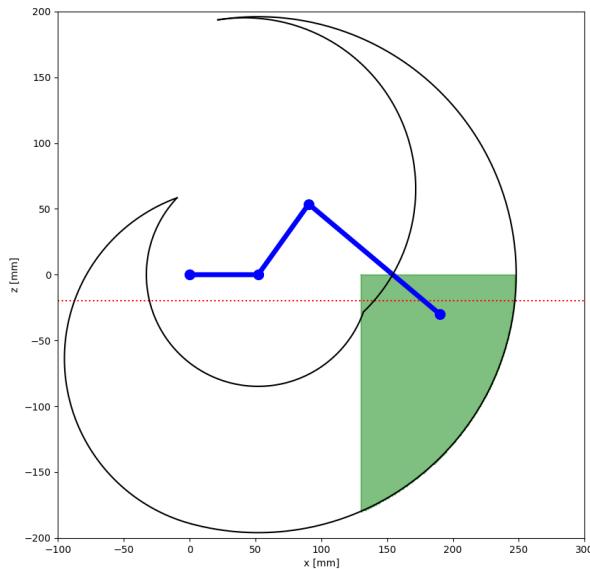


Fig. 3.15 Leg Range Before Revaluation

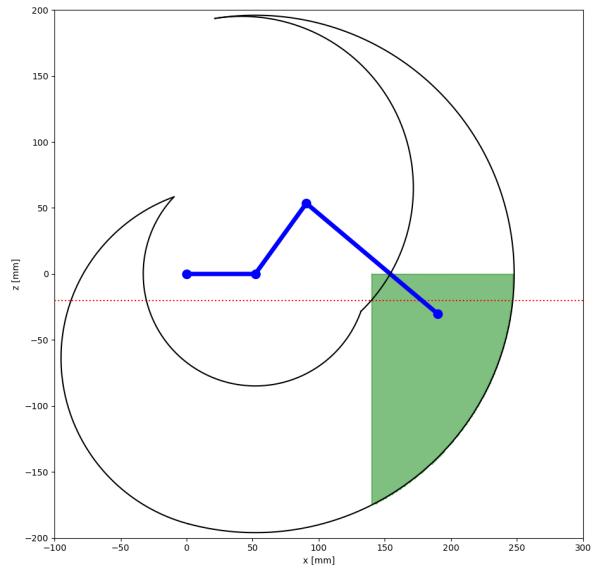


Fig. 3.16 Leg Range After Revaluation

### 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。しかし、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。より柔軟な動作をグラフ探索による自由歩容パターン生成手法で実現するため、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。

この作業は本研究の趣旨とは異なるが、直進以外のさまざまな動作でも脚軌道生成の失敗を防ぐことができるかを確認することができれば、再評価手法がより汎用的なものであることを示すことができると考え、本作業を実施した。

#### 3.3.1 3次元空間における旋回動作の実装

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 Table.3.3 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。また、○がついている動作は実装済みであることを表し、×がついている動作は未実装であることを表す。この表より、3次元空間における動作は直進以外実装されていないことがわかる。より柔軟な動作を実現するという目的のもとでは、統合されたプログラムは3次元空間に対応していることが望ましい。よって3次元空間における旋回動作を実装した。

旋回動作の研究は椎名ら [14] によって行われており、旋回の半径によって異なる歩容パターングラフの作成と、グラフ探索が行われていた。今回は簡単のため、旋回の半径が 0[mm] である場合（超信地旋回的に旋回を行う場合）のみを対象とし、以降、旋回と記述するときはそのような旋回を指すものとする。

Table. 3.3 Implemented Robot Operation

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

### 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

## 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、実装した再評価手法を用いたシミュレーションの結果を述べる。

- 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

第 5 章では、実機を用いた歩行実験を行い、

### 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 などがある。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[21] 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、可読性を向上させることができる。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も多いだろう。

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javascript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。



## 付録 B

# 脚の可動域を表示するプログラム

### B.1 概要

近似的な脚の可動域が適切であるかを評価するためには, PhantomX の可動域を正確に把握する必要がある. そのためには可動域の可視化を行うことが必要だろう. そこで, PhantomX の脚の可動域を可視化するプログラムを作成した. プログラムは簡単のため Python を用いて作成しており, GitHub を通じてだれでも利用できるようにしている. 以下にプログラムの導入方法を説明し, プログラムの仕様について述べる.

### B.2 導入方法

プログラムは GitHub を通じて公開しているため, まずは GitHub からプログラムをダウンロードする方法を説明する. そして Python のプログラムをコンパイルする方法を説明する.

#### B.2.1 Git の使用方法

Git とは, ソースコードなどの変更履歴を記録・追跡するための分散型バージョン管理システムである. 分かりやすい例を挙げるならば, Google ドキュメントの履歴機能や, Microsoft Word の変更履歴機能のようなシステムをさまざまなプログラムのソースコードに対して適用したものといえるだろう. また, GitHub とは, Git を用いてソースコードを管理するためのサービスである. GitHub を用いることで, ソースコードの変更履歴を保存することができるだけでなく, 自分のソースコードを公開したり, 他の人が作成したソースコードをダウンロードしたりすることもできる.

Git を使うため, まずは Git をインストールする必要がある. 次のリンクから Git のインストーラをダウンロードし, 実行する. インストーラではオプションの設定を変更する必要はないため, すべてのページで変更をせずに「Next」を選択してインストールを行う.

- Git for Windows <https://git-scm.com/downloads> (アクセス日 2023/12/30)

Git は多くのアプリケーションとは違い、コマンドを用いて操作を行う。そのため、インストールが完了しても、Git のアプリケーションが起動することはない。Git のインストールが成功したかどうかは、コマンドを用いて確認する。インストールが完了した場合、Git のコマンドが使用できるようになっているはずである。一度パソコンを再起動したのち、Fig.B.1 のようにスタートメニューに「Git Bash」と入力して Git Bash を起動する。Git Bash が起動したら、ListingB.1 のコマンドを入力する。Fig.B.2 のように Git のバージョンが表示されれば、インストールは成功している。

Listing B.1 Git のバージョン確認コマンド

```
1 git --version
```

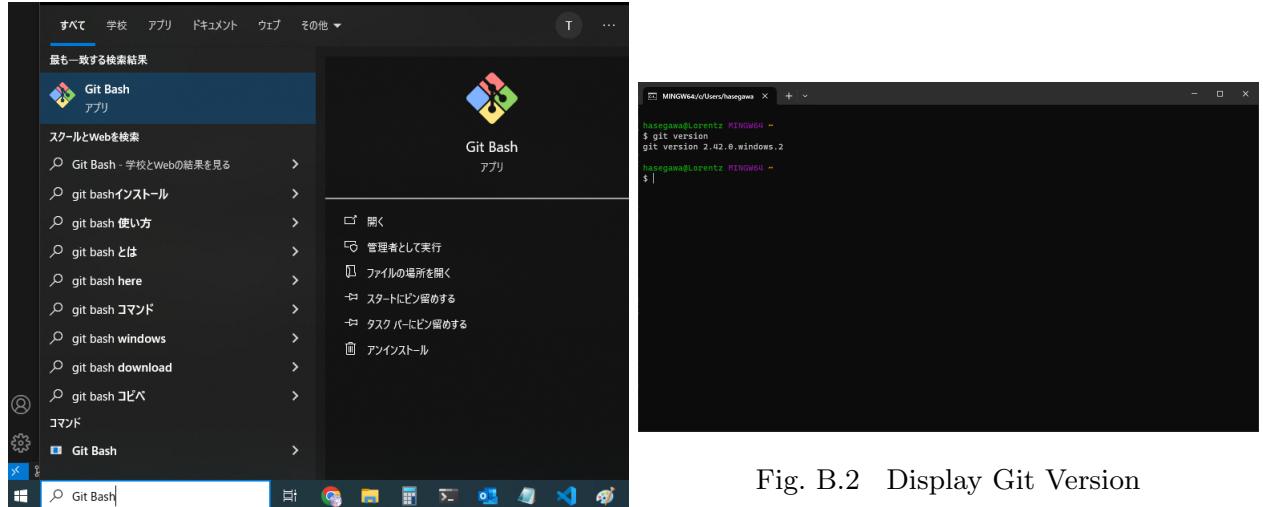


Fig. B.2 Display Git Version

Fig. B.1 Git Bash

## B.2.2 python のコンパイル方法

本項目では、Windows 10 における python のコンパイル方法を説明する。Mac OS や Linux における python のコンパイル方法については、ここでは説明しない。（しかし、「python Mac 環境構築」、「python Linux 環境構築」などで検索すると最適な方法が見つかるだろう。）

## B.3 プログラムの仕様

## 付録 C

# PhantomX Mark II のサーボモータ

### C.1 サーボモータの仕様

PhantomX Mark II は計 18 個のサーボモータを搭載しており、Dynamixel AX-12A, あるいは Dynamixel AX-18A を搭載したモデルがある。今回使用したのは Dynamixel AX-18A を搭載したモデルであるため、以下に Dynamixel AX-18A の仕様をまとめた Web ページ [22] を掲載する。

表示	
Home > 製品取扱説明書 > Dynamixel > AXシリーズ > AX-18F / AX-18A	
<b>AX-18F / AX-18A</b>	
部品の写真	
	
※AX-18Aは、AX-18Fと同じ性能で外形を改善したモデルです。現在はAX-18Aのみ販売しております。	
主な仕様概要	
<ul style="list-style-type: none"> <li>● 重量 : 54.5g (AX-18F), 55.9g(AX-18A)</li> <li>● サイズ : 32mm × 50mm × 40mm</li> <li>● 最小制御角 : 0.33°</li> <li>● ポア比 : 254 : 1</li> <li>● Stall Torque : 1.8N.m (at 12V, 2.2A)</li> <li>● No load speed : 97rad/s(at 12V)</li> <li>● 動作モード           <ul style="list-style-type: none"> <li>● 回転モード F (0 ~ 360°)</li> <li>● 斜傾モード F (無限回軸)</li> </ul> </li> <li>● 動作温度 : -5~120° (推薦電圧11.1V)</li> <li>● Command Signal : Digital Packet</li> <li>● Protocol Type : Half-duplex Asynchronous Serial Communication (9bit, 1stop, No Parity)</li> <li>● Link (Physical) : TTL Level Multi Drop (daisy chain type Connector)</li> <li>● ID : 254 ID (0~250)</li> <li>● 通信速度 : 7840bps→1 Mbps</li> <li>● Feedback : Position, Temperature, Load, Input Voltage, etc.</li> <li>● Material : Engineering Plastic</li> </ul>	

Control Table																																																																																
EEPROM and RAM																																																																																
<p>Control Tableはダイナミクセルの内部に存在する値で、ダイナミクセルの現在の状態と駆動に関するDataで構成されています。ユーザーは、Instruction Packetを通じてControl Tableのデータを変更することにより、ダイナミクセルを操作することができます。</p>																																																																																
<p><b>Address</b></p> <p>Addressは、データの位置です。ダイナミクセルにデータの書き込みや読み取りをするためには、PacketにそのデータがあるAddressを指定しなければなりません。</p>																																																																																
<p><b>アクセス</b></p> <p>ダイナミクセルデータには、読み取り専用 (R) と読み書きが可能なもの (RW) の2つがあります。読み取り専用 (R) は、主にセンシング用に使用されるデータであり、読み書きが可能なもの (RW) は、操作のためのデータです。</p>																																																																																
<p><b>初期値</b></p> <p>Control Tableの右に表示される初期値は、EEPROM領域のデータの場合、工場出荷値であり、RAM Areaデータの場合、電源が印加された時の初期値です。</p>																																																																																
<p><b>上位バイト/下位バイト</b></p> <p>Control Tableには、名称は同じでありますが、(L) と (H) が後ろについてAddressが区別されているものがあります。これは、(R)で必要なデータをbitsずつ各Address (low, high) に分けて表現したものです。この2つのAddressは、1つのInstruction Packetに同時にwriteされなければなりません。</p>																																																																																
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Area</th> <th>アドレス (10進数)</th> <th>名前</th> <th>意味</th> <th>アクセス</th> <th>初期値 (16進数)</th> </tr> </thead> <tbody> <tr> <td rowspan="14">E P R O M</td> <td>0 (0x00)</td> <td>Model Number(L)</td> <td>モデル番号の下位バイト</td> <td>R</td> <td>18(0X12)</td> </tr> <tr> <td>1 (0x01)</td> <td>Model Number(H)</td> <td>モデル番号の上位バイト</td> <td>R</td> <td>0 (0X00)</td> </tr> <tr> <td>2 (0x02)</td> <td>Version of Firmware</td> <td>ファームウェアバージョンの情報</td> <td>R</td> <td></td> </tr> <tr> <td>3 (0x03)</td> <td>ID</td> <td>ダイナミクセルのID</td> <td>RW</td> <td>1 (0X01)</td> </tr> <tr> <td>4 (0x04)</td> <td>Baud Rate</td> <td>ダイナミクセルの通信速度</td> <td>RW</td> <td>1 (0X01)</td> </tr> <tr> <td>5 (0x05)</td> <td>Return Delay Time</td> <td>応答遅延時間</td> <td>RW</td> <td>250 (0XFA)</td> </tr> <tr> <td>6 (0x06)</td> <td>CW Angle Limit(L)</td> <td>時計回りの最終角度値の下位バイト</td> <td>RW</td> <td>0 (0X00)</td> </tr> <tr> <td>7 (0x07)</td> <td>CW Angle Limit(H)</td> <td>時計回りの最終角度値の上位バイト</td> <td>RW</td> <td>0 (0X00)</td> </tr> <tr> <td>8 (0x08)</td> <td>CCW Angle Limit(L)</td> <td>反時計回りの最終角度値の下位バイト</td> <td>RW</td> <td>255 (0XFF)</td> </tr> <tr> <td>9 (0x09)</td> <td>CCW Angle Limit(H)</td> <td>反時計回りの最終角度値の上位バイト</td> <td>RW</td> <td>3 (0X03)</td> </tr> <tr> <td>10 (0x0B)</td> <td>the Highest Limit Temperature</td> <td>内部限界温度</td> <td>RW</td> <td>75 (0X4B)</td> </tr> <tr> <td>12 (0x0C)</td> <td>the Lowest Limit Voltage</td> <td>最小限界電圧</td> <td>RW</td> <td>60 (0X3C)</td> </tr> <tr> <td>13 (0x0D)</td> <td>the Highest Limit Voltage</td> <td>最大限界電圧</td> <td>RW</td> <td>140 (0X8C)</td> </tr> <tr> <td>14 (0x0E)</td> <td>Max Torque(L)</td> <td>トルク限界値の下位バイト</td> <td>RW</td> <td>215 (0XD7)</td> </tr> </tbody> </table>				Area	アドレス (10進数)	名前	意味	アクセス	初期値 (16進数)	E P R O M	0 (0x00)	Model Number(L)	モデル番号の下位バイト	R	18(0X12)	1 (0x01)	Model Number(H)	モデル番号の上位バイト	R	0 (0X00)	2 (0x02)	Version of Firmware	ファームウェアバージョンの情報	R		3 (0x03)	ID	ダイナミクセルのID	RW	1 (0X01)	4 (0x04)	Baud Rate	ダイナミクセルの通信速度	RW	1 (0X01)	5 (0x05)	Return Delay Time	応答遅延時間	RW	250 (0XFA)	6 (0x06)	CW Angle Limit(L)	時計回りの最終角度値の下位バイト	RW	0 (0X00)	7 (0x07)	CW Angle Limit(H)	時計回りの最終角度値の上位バイト	RW	0 (0X00)	8 (0x08)	CCW Angle Limit(L)	反時計回りの最終角度値の下位バイト	RW	255 (0XFF)	9 (0x09)	CCW Angle Limit(H)	反時計回りの最終角度値の上位バイト	RW	3 (0X03)	10 (0x0B)	the Highest Limit Temperature	内部限界温度	RW	75 (0X4B)	12 (0x0C)	the Lowest Limit Voltage	最小限界電圧	RW	60 (0X3C)	13 (0x0D)	the Highest Limit Voltage	最大限界電圧	RW	140 (0X8C)	14 (0x0E)	Max Torque(L)	トルク限界値の下位バイト	RW	215 (0XD7)
Area	アドレス (10進数)	名前	意味	アクセス	初期値 (16進数)																																																																											
E P R O M	0 (0x00)	Model Number(L)	モデル番号の下位バイト	R	18(0X12)																																																																											
	1 (0x01)	Model Number(H)	モデル番号の上位バイト	R	0 (0X00)																																																																											
	2 (0x02)	Version of Firmware	ファームウェアバージョンの情報	R																																																																												
	3 (0x03)	ID	ダイナミクセルのID	RW	1 (0X01)																																																																											
	4 (0x04)	Baud Rate	ダイナミクセルの通信速度	RW	1 (0X01)																																																																											
	5 (0x05)	Return Delay Time	応答遅延時間	RW	250 (0XFA)																																																																											
	6 (0x06)	CW Angle Limit(L)	時計回りの最終角度値の下位バイト	RW	0 (0X00)																																																																											
	7 (0x07)	CW Angle Limit(H)	時計回りの最終角度値の上位バイト	RW	0 (0X00)																																																																											
	8 (0x08)	CCW Angle Limit(L)	反時計回りの最終角度値の下位バイト	RW	255 (0XFF)																																																																											
	9 (0x09)	CCW Angle Limit(H)	反時計回りの最終角度値の上位バイト	RW	3 (0X03)																																																																											
	10 (0x0B)	the Highest Limit Temperature	内部限界温度	RW	75 (0X4B)																																																																											
	12 (0x0C)	the Lowest Limit Voltage	最小限界電圧	RW	60 (0X3C)																																																																											
	13 (0x0D)	the Highest Limit Voltage	最大限界電圧	RW	140 (0X8C)																																																																											
	14 (0x0E)	Max Torque(L)	トルク限界値の下位バイト	RW	215 (0XD7)																																																																											

Page 1

Page 2

Fig. C.1 Description AX-18A (page1-2)

	15 (0x0F)	Max Torque(H)	トルク限界値の上位バイト	RW	3 (0x03)
	16 (0x10)	Status Return Level	応答レベル	RW	2 (0x02)
	17 (0x11)	Alarm LED	アラーム用LEDの機能	RW	360 (0x24)
	18 (0x12)	Alarm Shutdown	アラーム用シャットダウン (Shut down)	RW	360 (0x24)
R A M	24 (0x18)	Torque Enable	トルクのOn / Off	RW	0 (0x00)
	25 (0x19)	LED	LED On/Off	RW	0 (0x00)
	26 (0x1A)	CW Compliance Margin	CW Compliance margin	RW	1 (0x01)
	27 (0x1B)	CCW Compliance Margin	CCW Compliance margin	RW	1 (0x01)
	28 (0x1C)	CW Compliance Slope	CW Compliance slope	RW	32 (0x20)
	29 (0x1D)	CCW Compliance Slope	CCW Compliance slope	RW	32 (0x20)
	30 (0x1E)	Goal Position(L)	目標位置値の下位バイト	RW	-
	31 (0x1F)	Goal Position(H)	目標位置値の上位バイト	RW	-
	32 (0x20)	Moving Speed(L)	目標速度値の下位バイト	RW	-
	33 (0x21)	Moving Speed(H)	目標速度値の上位バイト	RW	-
	34 (0x22)	Toque Limit(L)	トルク限界値の下位バイト	RW	A0D14
	35 (0x23)	Toque Limit(H)	トルク限界値の上位バイト	RW	A0D15
	36 (0x24)	Present Position(L)	現在位置値の下位バイト	R	-
	37 (0x25)	Present Position(H)	現在位置値の上位バイト	R	-
	38 (0x26)	Present Speed(L)	現在速度値の下位バイト	R	-
	39 (0x27)	Present Speed(H)	現在速度値の上位バイト	R	-
	40 (0x28)	Present Load(L)	現在荷重値の下位バイト	R	-
	41 (0x29)	Present Load(H)	現在荷重値の上位バイト	R	-
	42 (0x2A)	Present Voltage	現在電圧	R	-
	43 (0x2B)	Present Temperature	現在温度	R	-
44 (0x2C)	Registered	Instructionの登録状況	R	0 (0x00)	
45 (0x2D)	Moving	移動の有無	R	0 (0x00)	
47 (0x2F)	Lock	EPPONのロック	RW	0 (0x00)	
48 (0x30)	Punch(L)	Punch値の下位バイト	RW	32 (0x20)	
49 (0x31)	Punch(H)	Punch値の上位バイト	RW	0 (0x00)	

## Address Function Help

## EEPROM Area

Model Number  
ダイナミクセルのモデル番号です。

Firmware Version  
ダイナミクセルファームウェアのバージョンです。

## Page 3

## The Highest Limit Temperature

動作温度の上限値です。  
使用範囲は10~99 (0x10~0x63) であり、単位は攝氏温度です。

例えば、値が80であると、80°Cです。

内部の温度がこの値を超える場合は、Status PacketのERRORのOver Heating Error Bit (Bit2) が「1」に設定されて返送され、Alarm LED / Shutdownのフラグ (flag) のうち、過熱 (Overheating) が設定されていると、機能が発揮されます。

注意：温度の上限ダブルット値よりも高く設定しないでください。  
温度アラームのショットダウンが発生した場合、20分以上休憩して、ダイナミクセルの温度を十分に下げた後、使用してください。温度が高い状態での使用時に製品が発熱する恐れがあります。

## The Lowest (Highest) Limit Voltage

電圧動作範囲の上限と下限値です。  
上限と下限それぞれ50~250 (0x32~0x95) まで使用可能であり、単位は0.1Vです。

例えば、値が80であると、80%です。

現在の電圧の値がこの範囲を超える場合、Status PacketのERRORのVoltage Range Error Bit(Bit0) が「1」に設定されて返送され、Alarm LED / Shutdown

のフラグ (flag) のうち、入力電圧エラー (Input Voltage Error) が設定されていると、機能が発揮されます。

## Max Torque

モータの最大出力の基準値です。

0~1023 (0x3FF) まで使用可能であり、単位は、約0.1%です。

例えば、値が512であると、約50%であり、最大出力対比50%だけを使用するという意味です。

電源が入るとTorque Limit (Address 34, 35) は、この値を初期値として使用します。

## Status Return Level

状態パケット (Status Packet) の変換方式を決定します。

値	動作方式
0	すべての命令に対して変換しない (ただし、PING命令は除外)
1	READ命令に対してのみ変換する。
2	すべての命令に対して変換する。

参考：命令パケット (Instruction packet) のIDがブロードキャスト (Broadcast) のIDの場合には、この値に関係なく状態パケット (Status Packet) が変換されません。

## Alarm LED

## Alarm Shutdown

ダイナミクセルは、動作中に発生する危険な状況を感知して、自らを保護することができます。

設定することができる危険な状況は、以下の表のとおりです。

BIT	名前	内容
BIT 7	0	-

## Page 5

ID	ID		
ダイナミクセルを識別するための固有の番号です。	0~253 (0xFD) まで使用可能で、254 (0xFE) はブロードキャスト (Broadcast) のIDとして特別に使用されます。		
Instruction packetを送信するとき、ブロードキャストのIDを使用すると、すべてのダイナミクセルに命令を下すことができます。	接続されたダイナミクセルのIDが重複しないように注意してください。		
Baud Rate	コントローラと通信するための通信速度です。		
0~254 (0xFE) まで使用可能で、計算式は次のとおりです。 Speed(BPS) = 2000000/(Data+1)	接続されたダイナミクセルのIDが重複しないように注意してください。		
Data	設定BPS	目印BPS	設定
1	1000000.0	1000000.0	0.000 %
3	500000.0	500000.0	0.000 %
4	400000.0	400000.0	0.000 %
7	250000.0	250000.0	0.000 %
9	200000.0	200000.0	0.000 %
16	117647.1	117647.0	-2.124 %
34	5742.9	5742.9	0.794 %
103	19230.8	19230.0	-0.160 %
207	9615.4	9600.0	-0.160 %

参考：BaudはBaudrate誤差が3%以内であると、通信に支障がありません。

Return Delay Time  
コントローラからInstruction Packetを受信した後、Status Packetを送信するまでの時間です。  
0~254 (0xFE) まで使用可能で、単位はusecです。  
例えば、値が100の場合、20 usecだけ時間が経過した後にStatus Packetを応答します。

## CW/CCW Angle Limit

動作が許可される角度を設定することができます。

値の範囲は、最低値は、Goal Position (Address 30, 31) と同じです。

- CW Angle Limit : Goal Position (Address 30, 31) の最小値
- CCW Angle Limit : Goal Position (Address 30, 31) の最大値

CW/CCW値によって、次の2つの動作モードを設定することができます。

動作方式	CW / CCW
車輪モード	車輪のもの
関節モード	関節とのもの

車輪モードは、モータが無段階駆動をするため、車輪型駆動ロボットに使用できます。

関節モードは、特定の角度での制御が可能であり、多関節ロボットに使用できます。

## Page 4

Bit 6	Instruction Error	定義されていないinstructionが送信された場合、またはreq_write命令なしでAction命令が渡された場合
Bit 5	Overload Error	モータの最大出力で制御することができない範囲が継続的に適用される場合
Bit 4	Checksum Error	送信されたInstruction PacketのChecksumが合わない場合
Bit 3	Range Error	該当Addressの値の範囲外の値をInstruction Packetに送信する場合
Bit 2	Overheating Error	内部の温度が設定されて動作温度範囲を越えている場合
Bit 1	Angle Limit Error	適用したGoal Positionで設定したCW / CCW Angle Limit範囲を超えている場合
Bit 0	Input Voltage Error	印加された電圧が設定された動作電圧範囲を超えていている場合

各Bitの機能は、「OR」の論理が適用されるので、重複設定が可能です。つまり、0x05 (2進数: 00000101) に設定されている場合、Input Voltage ErrorとOverheating Error、両方をすべて感知することができます。

危険な状況が発生すると、Alarm LEDの場合LEDを点滅させ、Alarm Shutdownの場合はTorque Limit (Address 34, 35) の値を0にすることでモータ出力が0%になるようになります。

RAM Area

値	意味
0	モータの電源を遮断してTorqueが発生しないようにします。
1	モータに電源を印加してTorqueを発生させます。

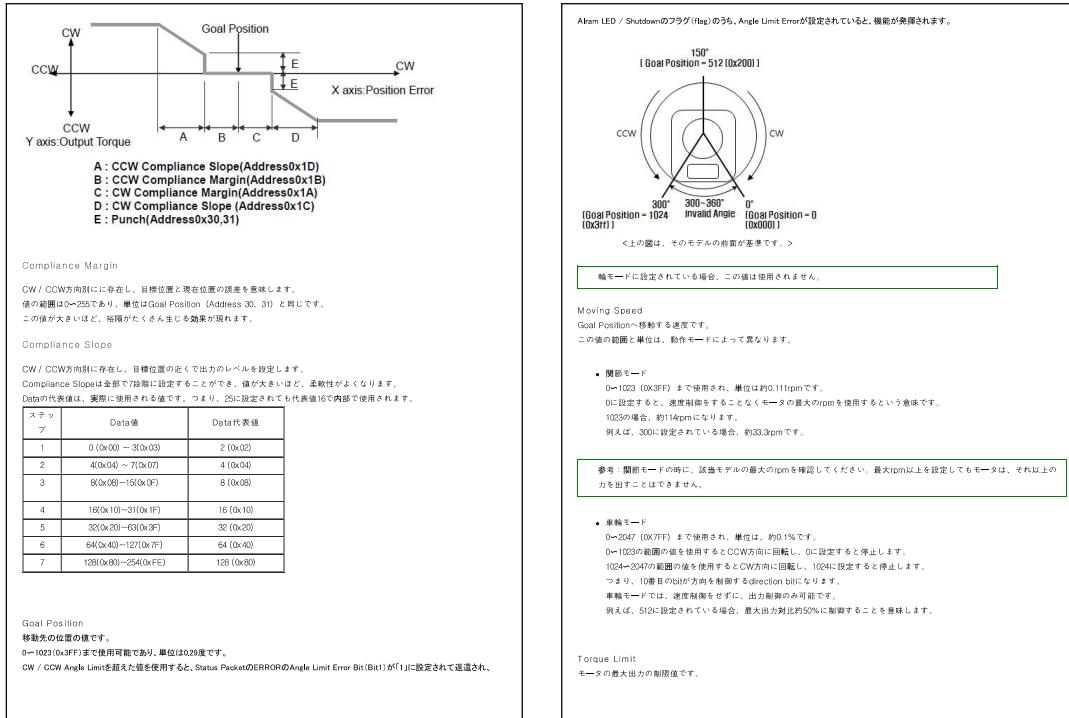
LED

値	Meaning
0	LEDがOFFにします。
1	LEDがONにします。

Compliance  
Complianceは、モータの制御の柔軟性を設定することです。  
下の図は、位置とモーターの出力の関係を示しています。

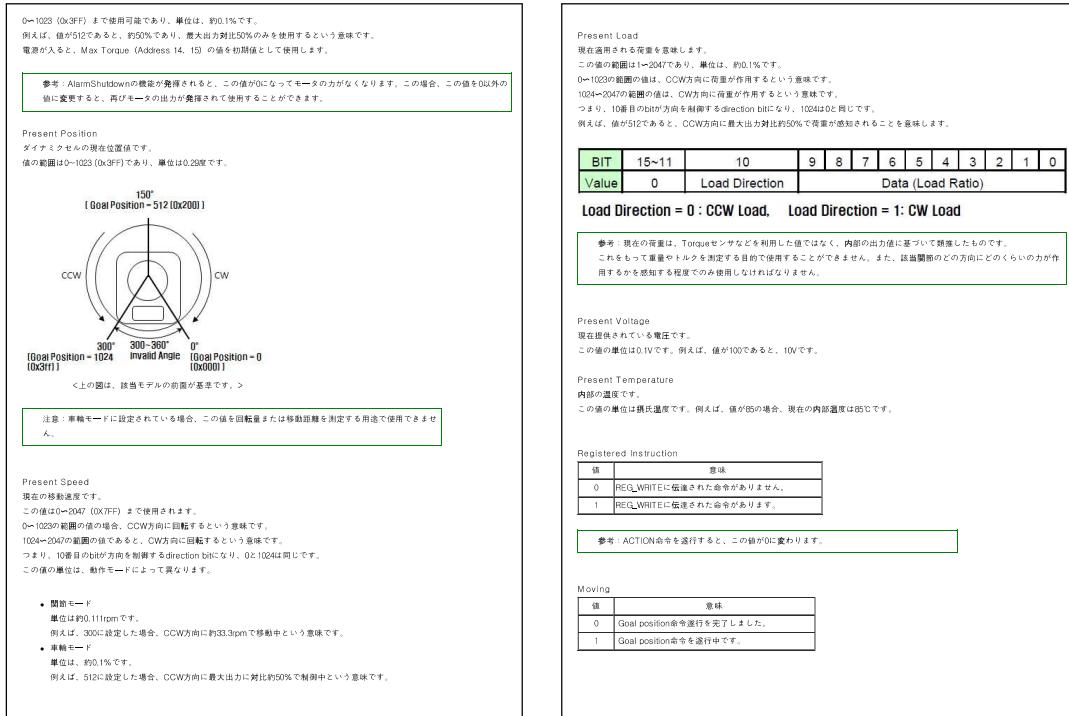
## Page 6

Fig. C.2 Description AX-18A (page3-6)



Page 7

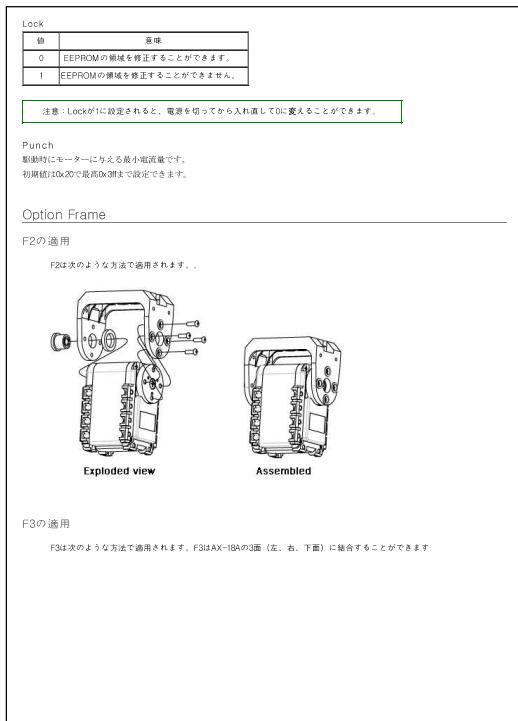
Page 8



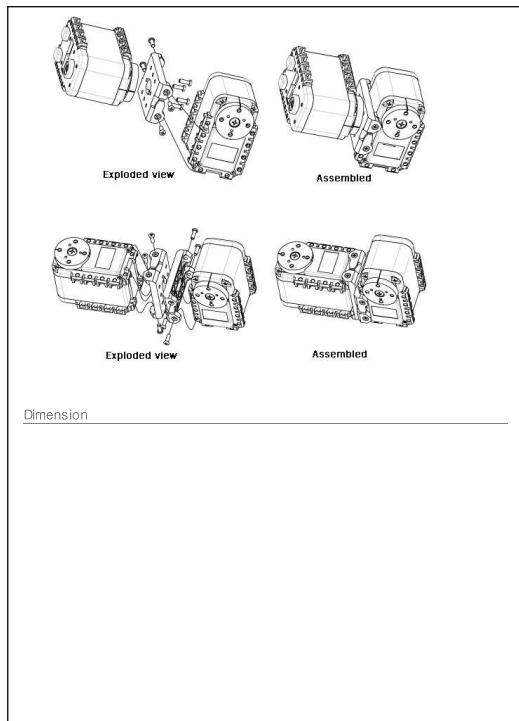
Page 9

Page 10

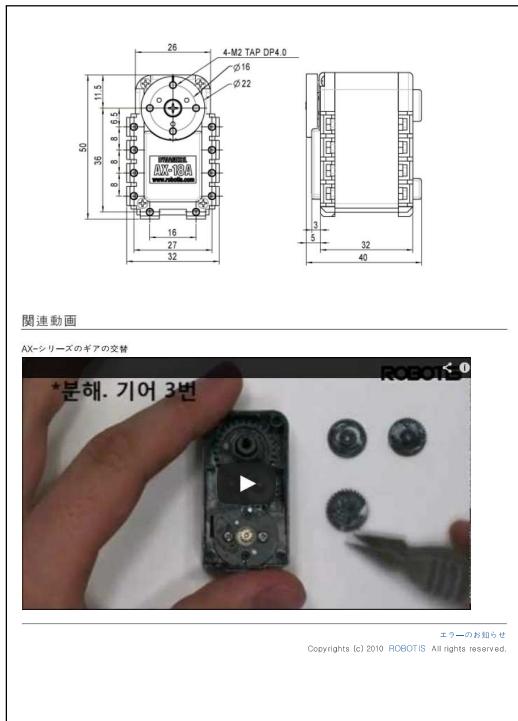
Fig. C.3 Description AX-18A (page 7-10)



Page 11



Page 12



Page 13

Fig. C.4 Description AX-18A (page11–13)





## 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 ライブライリ置き場. <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] Google : “Google C++ Style Guide”. <https://google.github.io/styleguide/cppguide.html>, (参照 2024/01/23).
- [21] 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).
- [22] ROBOTIS : “AX-18A User Manual”. <https://e-shop.robotis.co.jp/product.php?id=235>, (参照 2023/9/15).