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

歩容パターンの再評価手法の実装

第 1 章では，第??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．

1.1　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．

1.1.1 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.1.1 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.1.2 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．GaitPattern-GeneratorBasic 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，GaitPattern-GeneratorBasic 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．

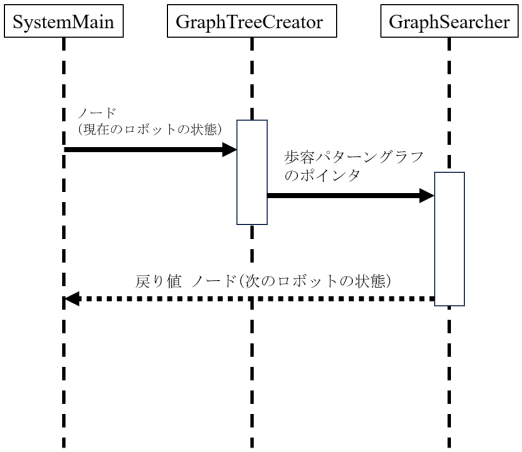

Fig. 1.1 Graph Search Sequence

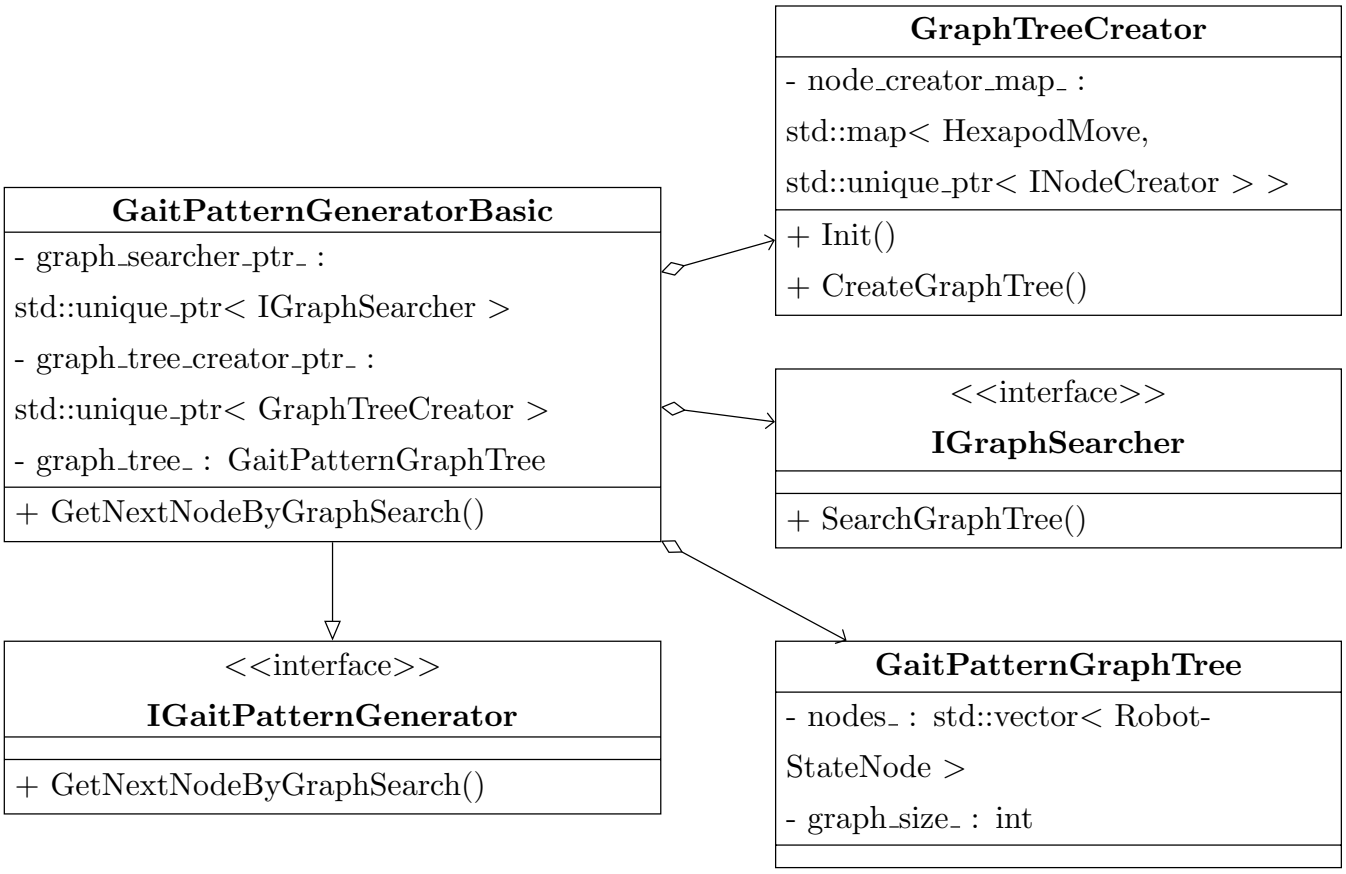

Fig. 1.2 Graph Search Class Diagram

1.1.2 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.1.3 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.1.6 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.1.4 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.1.5 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．

RobotStateNode
+ 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. 1.3 RobotStateNode Struct

Vector3
+ x : float
+ y : float
+ z : float

Fig. 1.4 Vector3 Struct

Quaternion
+ w : float
+ v : Vector3

Fig. 1.5 Quaternion Struct

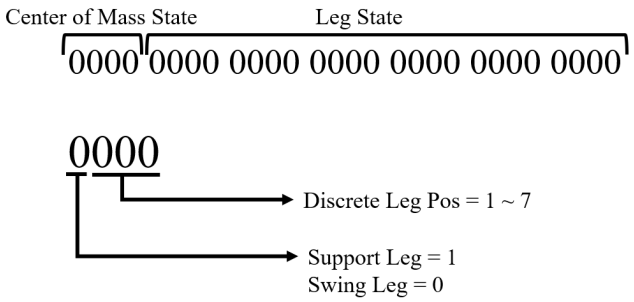

Fig. 1.6 Leg State Bit

1.1.3 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.1.7 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.1.1 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. 1.7 RobotOperation Struct

Table. 1.1 RobotOperationType Enum

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

1.1.4 地形を表現する構造体

地形を表現する MapState 構造体を Fig.1.8 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_をメンバ変数に持っている．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は多い．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.1.9 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の，各分割領域における脚接地可能点を表す divided_map_point_をメンバ変数に持っている．また，各分割領域の最上点の高さを表す divided_map_top_z_をメンバ変数に持っている．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．

1.1.5 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.1.10 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

MapState
+ map_point_ : std::vector < Vector3 >

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

Fig. 1.8 MapState Struct

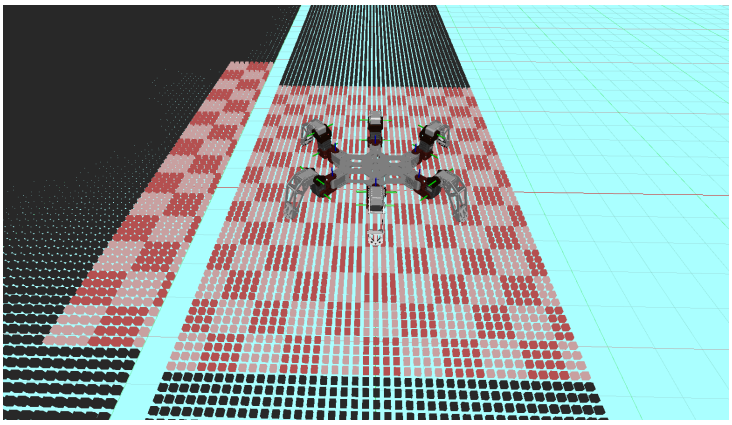

Fig. 1.9 Map View

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．

<<interface>> **INodeCreator**
+ Create(current_node : RobotStateNode, current_node_index : int, output_graph : std::vector < RobotStateNode > *) : void

Fig. 1.10 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.1.11 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．

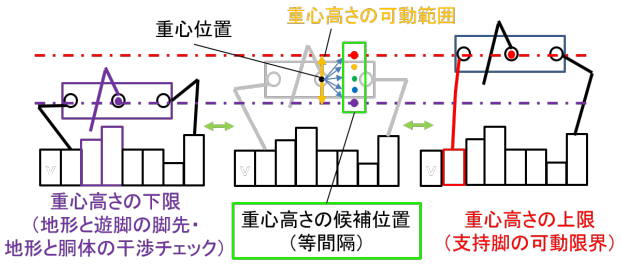

Fig. 1.11 Discretization of the Vertical Shift of the Center of Mass

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.1.12 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.1.13 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．

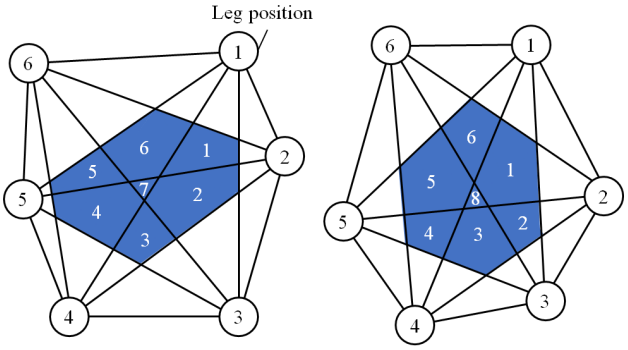

Fig. 1.12 Candidate Area of Center of Mass

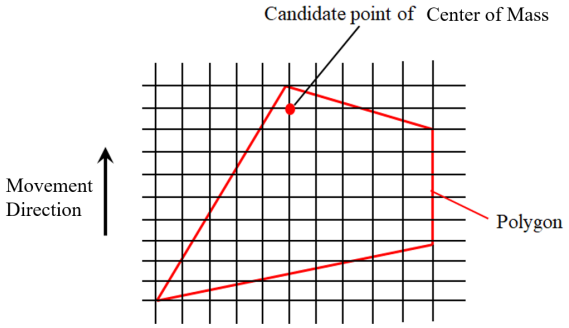

Fig. 1.13 Determination of Center of Mass

胴体の回転移動

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 $-20 \sim 20$[deg] の範囲を 2[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 **??** 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.1.2 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.1.12 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. 1.2 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．

1.1.6 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を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．

1.2 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 **??** 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 **??** 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条件に設定する．また，それぞれを図示したものを Fig.1.15 と Fig.1.16 に示す．

(1) 再評価前
- 最小半径を 130$[mm]$ に設定
- 最大半径は第 **??** 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 **??** 章で述べた計算方法で設定
- 遊脚高さは $-20[mm]$ に設定

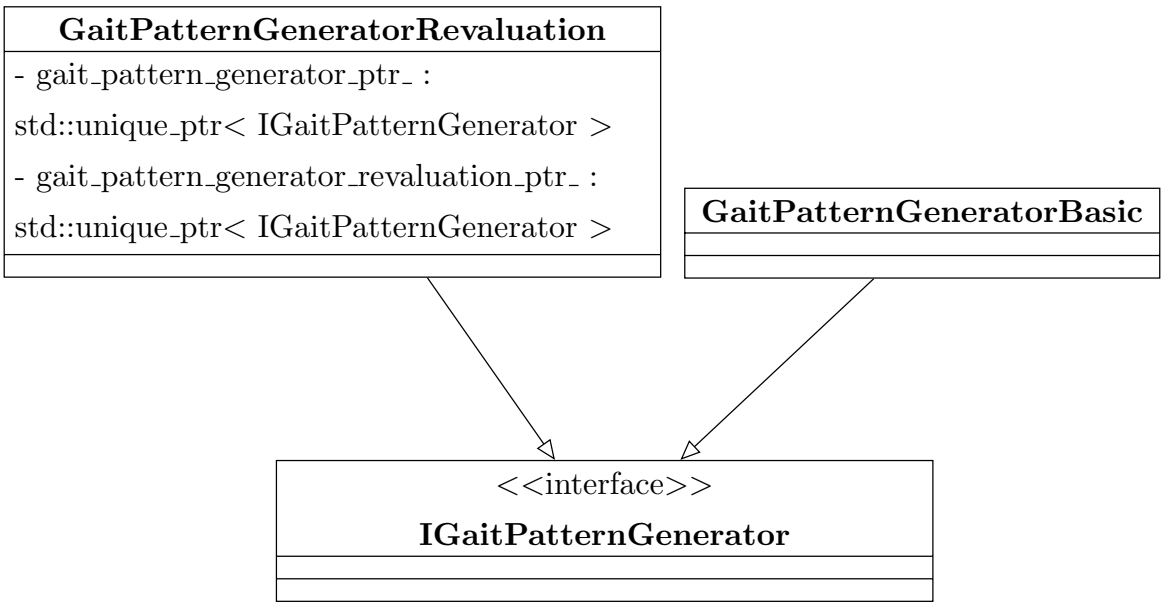

Fig. 1.14 Gait Pattern Generator Revaluation Class Diagram

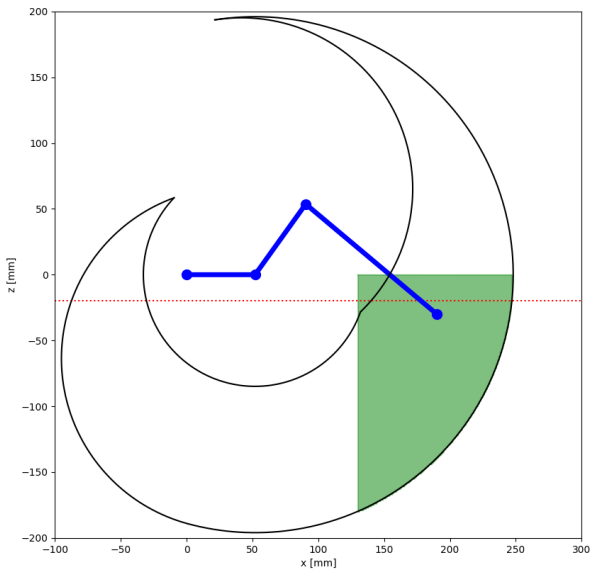

Fig. 1.15 Leg Range Before Revaluation

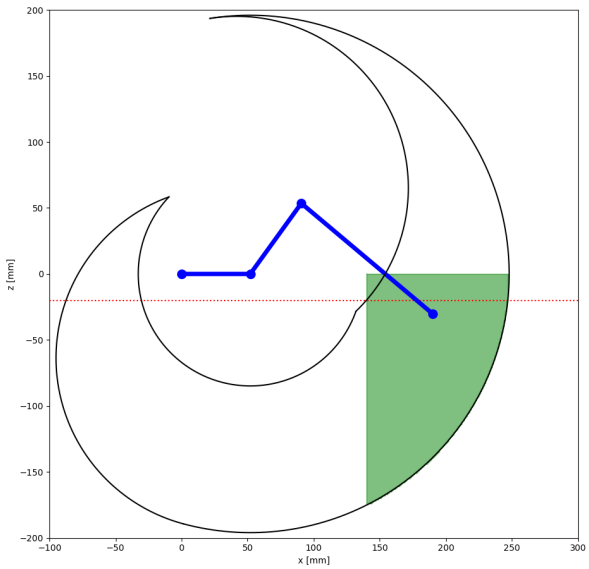

Fig. 1.16 Leg Range After Revaluation

1.3 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．そのため，本作業を実施した．

1.3.1 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.1.3 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．

Table. 1.3 Implemented Robot Operation

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	○	×
特定姿勢での静止	○	×

1.3.2 自由歩容パターン生成手法の切り替えアルゴリズム

第 2 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 2 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．

2.1　直進動作の自由歩容パターン生成シミュレーション

2.2　旋回動作の自由歩容パターン生成シミュレーション

2.3　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 3 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 3 章では，実機を用いた歩行実験を行い，

3.1　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．

3.2　実験に使用した 6 脚ロボット

3.3　歩行条件

3.4　実験に使用した地形

3.5　結果

3.6　考察

第 4 章

結論

4.1　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．

4.2　今後の課題

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] 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).