修 士 学 位 論 文

四脚移動ロボットの
静的全方位歩行に関する研究

平成 12 年度

茨城大学大学院 理工学研究科

システム工学 専攻
富山 隆志

Abstract

Researches on legged robots have been performed for many years, but the utilization of the legged robots cannot be yet in prospect. One reason would be the difficulty and the complexity of the leg motion planning to guarantee the robot stability. Traditionally, this kind of leg motion was realized by applying a periodic movement pattern seen in animals' walking. This technique, however, can not directly apply to arbitrary direction walking. On the other hand, there have also some works that do not persist in animals' walking pattern but treated the robot walking as a search problem. The walking of the robot was realized by the search using the rule set or the search using the nonlinear optimization technique, all directional walking, however, has not been realized ether.

In this study, we limit discussion on the static walking of a quadruped robot, and present two all-direction-walking gates to realize all direction static walking of the robot by computing a center of turning from the walk velocity and angular velocity given by an operator. One is based on the static walk pattern and another is not depend on the walk pattern but uses a search technique.

The walk gate planning technique based on walking pattern chooses the crawl gate or turning gate from center of turning and makes gate transition continuously when the center of turning is changed.

Another planning technique of the walk gate called "free gate" does not depended on the walking pattern, but uses a search technique with evaluation function including stability and mobility for 16 sets of swing leg sequences.

Through the walking experiments by a computer simulation and by real robot, the validity and inherent problems of the proposed techniques were verified. The walking-pattern-based walking gate planning technique makes the walking speed and all directional walking practically sufficient, but the free gate planning technique only makes a walking toward fixed direction converge to a fixed pattern of swing leg sequence.

要旨

脚移動ロボットの研究は古くからなされてきている．しかし未だに実用化の目処が立っていない．その主たる原因の一つとしては，転倒に対する安定性を保つような脚運動の計画が複雑で困難であることが挙げられる．従来の研究ではこの脚運動に動物に見られるような周期的な運動パターンを当てはめて定常的な歩行を実現していたが，このような手法では任意の方向に進行方向が変化するような歩行を実現することが困難であった．また動物に見られるような運動パターンに固執せず歩行を探索問題として捉え，ルールによる探索や非線型最適化による探索により歩行を実現した研究もなされているが，これらも任意の方向に歩行できるものではなかった．

この問題を解決するため，本研究では操縦型4脚移動ロボットの静歩行に議論を限定し，操縦入力により与えられた歩行速度と旋回角速度から旋回中心を算出して，この旋回中心回りに歩行することで全方位歩行を実現する．本論文では，この全方位歩行を実現するために必要な歩容計画手法として，静的な歩行パターンを基にした歩容計画手法と，歩行パターンに依らない探索による歩容計画手法の２つの手法を提案している．

歩行パターンを基にした歩容計画手法は，歩行速度と旋回角速度から求めた旋回中心の位置からクロール歩容と循環歩容のどちらかの歩行パターンを選択し，旋回中心が変わるときは各歩行パターン間の遷移を連続的に行うという歩容計画手法である．歩行速度や進行方向の変化も旋回中心位置が変わる歩容遷移として捉えることで停止状態から最高速度まであらゆる歩行パターンを実現している．

一方，歩行パターンに依らない歩容計画手法は一般には自由歩容と呼ばれる歩容計画手法であり，本論文で提案する手法は安定性と機動性を含んだ評価基準を用いて2歩1組の16通りの遊脚順序に対して評価を行い，評価関数の値が最も大きくなるような遊脚順序と着地点を選択するものである．特に1歩目の遊脚の着地点は次の遊脚運動の安定性を左右するため，あらかじめ2歩目の遊脚運動を保証するような着地領域を求めておき，この領域と脚の可動範囲との積集合となる領域内で着地点探索を行い常に次の1歩につながるような遊脚運動を実行することで安定性を保った歩行を実現している．

計算機シミュレーション及び実機による歩行実験では，本手法の有効性と内在する問題点を検証した．歩行パターンを基にした歩容計画による全方位歩行では，実用上十分な全方位歩行性能と歩行速度が得られることを確認した．しかし，実機を用いた歩行実験の結果，理論上は転倒のモーメントが0でも実機のモデル化誤差により転倒のモーメントが働いてしまうことも判明した．歩行パターンに依らない自由歩容計画を用いた全方位歩行では，進行方向を一定にとった歩行であれば一定のパターンに収束することを確認した．しかし任意の姿勢からでは歩行を続けられるまでに至らなかった．

本論文の最後で，提案する手法におけるそれぞれの問題点について考察し，不整地歩行における全方位歩行に向けて提案する手法の改良すべき点などについて

述べている.

目 次

第 1 章　序論 **1**

1.1　脚移動ロボット 1
1.2　従来の歩行に関する研究 1
1.3　本研究の目的 2
1.4　本論文の構成 2

第 2 章　四脚移動ロボットの全方位歩行 **5**

2.1　はじめに 5
2.2　脚移動ロボットの安定性評価基準 5
2.3　歩行パターン 7
2.3.1　動物の歩行パターン 7
2.3.2　歩行パターンによるロボットの歩行 9
2.3.3　歩行パターンを用いないロボットの歩行 10
2.4　四脚移動ロボットのモデル化 11
2.5　全方位歩行 13
2.6　２つの歩容計画アプローチ 14
2.7　まとめ 15

第 3 章　クロール歩容と循環歩容に基づく歩容計画 **17**

3.1　はじめに 17
3.2　基準歩容の計画 17
3.2.1　旋回中心の位置による歩容の選択 20
3.2.2　遊脚の着地位置と支持脚の終了位置 24
3.3　連続的な歩容遷移 27
3.3.1　歩容遷移のパターン 28
3.3.2　クロール歩容からクロール歩容への遷移 28
3.3.3　クロール歩容から循環歩容への遷移 29
3.3.4　循環歩容からクロール歩容への遷移 31
3.3.5　循環歩容から循環歩容への遷移 31
3.3.6　歩容遷移時間の比較 32
3.4　まとめ 33

第 **4** 章　遊脚順序探索と着地点探索に基づく自由歩容の計画　**35**
4.1　はじめに 35
4.2　遊脚運動に関する考察 35
4.2.1　遊脚運動の必要性 35
4.2.2　遊脚順序 35
4.2.3　遊脚の着地点 36
4.3　評価関数に基づく自由歩容計画 36
4.4　基本探索 37
4.4.1　遊脚候補の選択 37
4.4.2　着地点探索 38
4.5　特別ルールの適用 44
4.5.1　デッドロックに結び付く脚を含んだ遊脚順序の優先 44
4.5.2　歩行速度調整 44
4.5.3　揺動運動の付加 44
4.6　まとめ 45

第 **5** 章　歩行制御系の構築　**47**
5.1　はじめに 47
5.2　歩行制御系の概要 47
5.3　クロール歩容と循環歩容による歩容計画系 47
5.4　自由歩容計画系 48
5.5　脚先端軌道計画系 48
5.6　脚先端軌道追従制御系 49
5.7　実時間制御用歩行制御系の構築 49
5.8　まとめ 50

第 **6** 章　歩行実験　**55**
6.1　はじめに 55
6.2　実験機材 55
6.3　クロール歩容と循環歩容に基づく全方位歩行の実機実験 58
6.3.1　実験項目 58
6.3.2　結果と考察 58
6.4　自由歩容による全方位歩行のシミュレーション実験 65
6.4.1　実験項目 65
6.4.2　結果と考察 65
6.5　まとめ 71

第 **7** 章　結論　**73**
7.1　本論文のまとめ 73
7.2　今後の課題 73

付 録 **A** 歩容遷移中と遷移後の遊脚順序 **81**
A.1 クロール歩容からクロール歩容への遷移 81
A.2 クロール歩容から循環歩容への遷移 84
A.3 循環歩容からクロール歩容への遷移 87
A.4 循環歩容から循環歩容への遷移 87

付 録 **B** **RTLinux** による実時間制御 **89**
B.1 RTLinux とは 89
B.2 RTLinux カーネルの入手とインストール 89
B.3 module を作る 90
B.4 簡単なリアルタイム処理プログラム 91
B.5 まとめ 96
B.6 簡単なリアルタイム処理プログラムのサンプル 97
B.6.1 control.h 97
B.6.2 timer_module.c 97
B.6.3 usr_app.c 99
B.6.4 Makefile 101

第1章　序論

1.1　脚移動ロボット

人は道具を得る以前は自らの力で、自らの脚で物を運ばなければならなかった。やがて人は家畜を得て、ウシやウマ、ロバといった動物に荷物を運ばせるようになった。そしてより多くの物資を運搬するためにソリという道具がが発明され、より効率的に物資を運搬するためにソリの下に敷かれた丸い棒が車輪という機構の発明へとつながった。

このように車輪という地上における移動機構の発明は動物の運動形態を真似ることによって得られたものではなかったが、空を飛ぶための翼はもっぱら鳥の翼を真似ることによって空中を移動する機構を得たものであった。では、地上を移動する機構として人が脚機構を真似てこなかったのは脚機構が地上を移動する上で不利であったためかというとそうではなく、

- 脚機構が複雑になってしまうこと
- 自由度が高いため脚の操作が難しいこと
- 十分な出力を得られる動力源が無かったこと

などの技術的な問題点が脚機構を用いる上での障害となっていたからである。しかし、近年では複雑な脚機構を制御するためにコンピュータを使用することができるようになったため、また、小型で高出力のアクチュエータも少しづつ開発されているため、車輪移動に引けをとらない潜在能力を秘めた脚移動が盛んに研究されるようになっており、脚移動機構は不整地移動用のプラットフォームや能動的な足場を形成する作業用プラットフォームとして期待が高まっている。

この脚移動の研究はまだ基礎的な段階であるが、昨今では6脚、4脚のみでなくヒューマノイドの分野で2脚移動も盛んに研究されており、山岳地帯や地震等の被災地における移動手段、地雷撤去処理からペットロボットに至るまで、脚移動ロボットの応用分野も幅広く研究されている。

1.2　従来の歩行に関する研究

脚移動の研究は、身近な動物の定常的な歩行（走行）パターンを観察することから始まり ([1][2][3])、クロール、トロット、ペース、ギャロップといった歩行（走

行）パターンを 4 脚移動ロボットで実現するという試みがなされた。特に、4 脚の歩行パターンの中でも静力学的に安定で扱いやすいクロール歩容をロボットの歩行に応用した事例は多い。例えば、クロール歩容を前方への直進だけでなく斜めに並進できるように拡張した基準側行歩容 [4] や任意の旋回中心回りの定常的な歩行に拡張しつつ旋回角速度の最大化をはかった基準旋回歩容 [5] などが提案されている。また、定常的な歩行パターン同士をつなぐ研究では、任意の 4 脚支持姿勢から任意の 4 脚支持姿勢に移行する踏み替え歩容 [8]、左右揺動を付加したクロール歩容と循環歩容を組み合わせて前進、後退、左右旋回、その場旋回を実現した研究 [12] などが報告されている。しかし、連続的な全方位歩行を実現する研究は、動歩行では間欠的トロット歩容 [11] が、静歩行では文献 [12] があるだけでほとんど検討されてこなかった。

一方、脚移動の研究には動物にみられるような歩行パターンによらないアプローチもある。この歩行パターンに頼らない歩き方を自由歩容いう。自由歩容計画の手法には、グラフ探索を用いたもの [19],[20] やモンテカルロ法を用いたもの [21] など歩容生成を非線型最適化問題として取り扱う手法、ルールセットを元に遊脚・支持脚を切り替えるもの [22]、外界センサからの入力を引金にした反射により歩行を実現したもの [23] など、様々な手法が提案されている。しかし、これらの研究の多くは並進のみを取り扱っており、旋回を含めた全方位歩行は実現できていない。

1.3　本研究の目的

脚移動ロボットを移動用、もしくは作業用機械のプラットフォームとして考えた場合、ロボットの推進動作はできる限りなめらかであることが望ましく、また、緊急時に胴体を一時停止しても安定性を失わないため、静歩行の利用価値が大変高い。そこで、本研究では静歩行に議論を限定し、四脚移動ロボットの静的全方位歩行を実現することを目的とする。

本研究の目的のうち、本論文では遊脚運動の順序やそのタイミングを計画する歩容計画に注目し、静力学に安定な歩行パターンの組み合わせに基づく歩容計画手法と評価関数をもちいた自由歩容計画手法という 2 つのことなる歩容計画手法を提案する。

1.4　本論文の構成

本論文は、以下のような構成になっている。

まず、第 2 章では脚移動の基本概念のうち最も重要な安定性評価基準と歩行パターンについて述べた後、本研究で取り扱う 4 脚移動ロボットのモデル化を行い全方位歩行への導入とする。

次に、第 3 章ではクロール歩容と循環歩容に基づく歩容計画について述べる。こ

の歩容計画手法は、大きく分けると基準歩容計画と各歩容間の連続的な遷移手順に分けることができる。

続いて第 4 章では遊脚順序と着地点探索に基づく自由歩容計画について述べる。特にこの歩容計画手法の中心となっている 2 歩目の遊脚運動を保証するための 1 歩目の着地点探索に重点をおいて説明する。

さらに第 5 章では第 3 章、第 4 章で述べた歩容計画を用いて歩行するための歩行制御系の構築について述べる。

第 6 章では本研究で提案する歩容計画手法について実機と計算機シミュレーションによる歩行実験を用いて検討し、最後の第 7 章では本論文のまとめと今後の課題について述べる。

第2章　四脚移動ロボットの全方位歩行

2.1　はじめに

全方位歩行への導入として、先ず脚移動に関する基礎的な概念のうち安定性評価基準と歩行パターンについて説明する。続いて本研究で取り扱う四脚移動ロボットのモデル化を行い、全方位歩行を実現する 2 つのアプローチについて述べる。

2.2　脚移動ロボットの安定性評価基準

脚移動は、車輪やクローラによる移動と異なり、接地面が次々に変化するため常に胴体が転倒しないよう安定性の低下に注意する必要がある。この脚移動ロボットの安定性を数値で表す安定性評価基準には様々なものが提案されているが大きく分けると、静力学的な安定性評価基準、動力学的な安定性評価基準、力学的エネルギによる安定性評価基準の 3 つに分類することができる。これらの主な評価基準には以下のようなものがある。

- 静力学的な安定性評価基準
  - 安定余裕 (Stability Margin) [24]
  - 縦安定余裕 (Longitudinal Stability Margin) [24]
  - 傾斜度安定余裕 (Gradinet Stability Margin) [25]
  - Tipover Stability Margin [26]
- 動力学的な安定性評価基準
  - ZMP ([24][11]) による安定余裕
  - 転倒安定余裕 (Tumble Stability Margin)[27]
- 力学的エネルギによる安定性評価基準
  - Energy Stability Margin [28]
  - Dynamic Energy Stability Margin [29]

– 正規化エネルギー安定余裕 (Normalized Energy Stability Margin) [30]

これらの評価基準は歩行環境や安定化制御の手法に応じて使い分ける。最も広く使用されているのは安定余裕であり、これは水平面上を静歩行する場合にのみ有効な安定性評価基準である。したがって、動力学的な安定化制御による動歩行を行う場合は、ZMP による安定余裕を用いて動力学的な安定性を評価する必要がある。

安定余裕の定義

支持脚の着地点が構成する多角形を水平面上に投影したものを支持脚多角形 (support polygon) という。脚移動ロボットの重心を支持脚多角形と同じ水平面上に投影したとき、重心の投影点と多角形との辺までの距離のうち最小となるものを安定余裕 **(Stability Margin)** という。安定余裕の単位は [m] である。重心の投影点が支持脚多角形の内部に入っていれば脚移動ロボットは静力学的に安定であり転倒のモーメントが働かない。

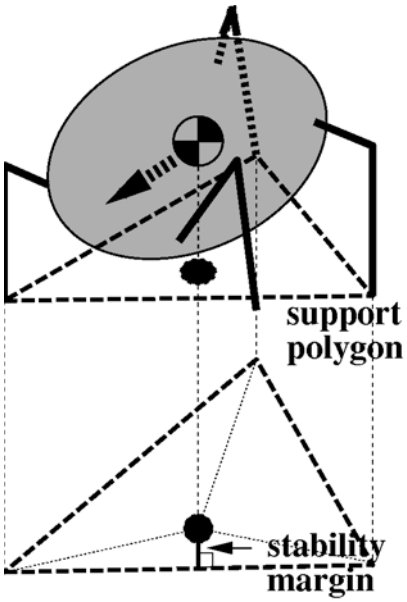


図 2.1: 安定余裕

本研究では水平面上を静歩行することを仮定するため、安定性評価基準の中で最も簡単で広く使われている安定余裕を脚移動ロボットの安定性評価基準として使用する。

2.3 歩行パターン

4 脚移動ロボットの歩行パターンには、動物の歩行パターンを応用したものと、歩行パターンによらないロボット特有の動きをするものがある。本節では、まず動物の歩行パターンとそのパターン生成の仕組みについて簡単に説明し、次に動物の歩行パターンをロボットに応用したものについて述べ、続いて歩行パターンによらないロボットの歩行について述べる。

2.3.1 動物の歩行パターン

脚移動ロボットの研究を行う上で、動物が歩行する様子を観察することは安定した歩行パターンを見いだす上で非常に有意義である。

四足動物の代表的な歩行 (走行) パターンを図 2.2 - 図 2.7 に示す。

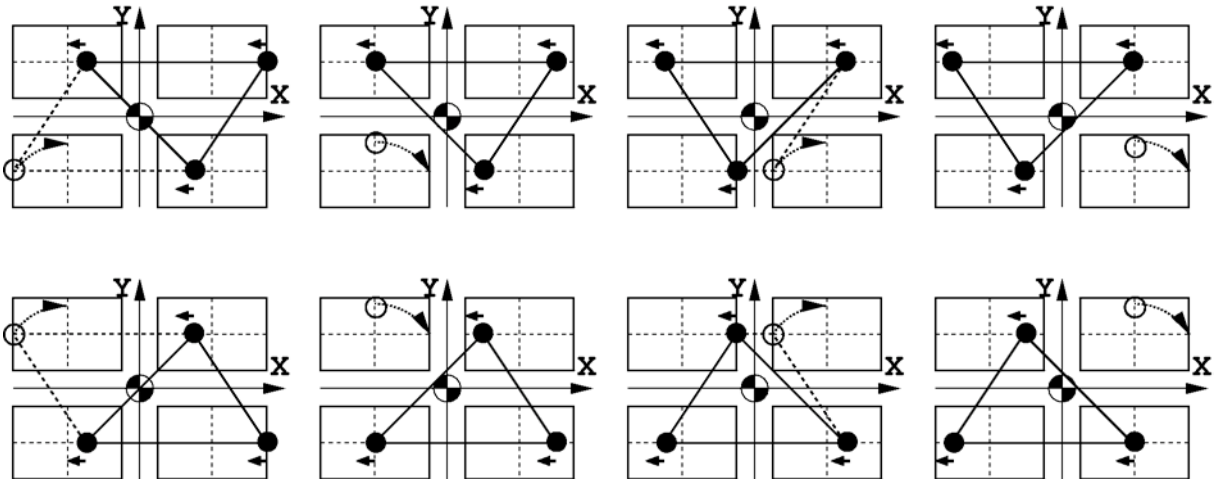


図 2.2: クロール ($\beta = 0.75$)

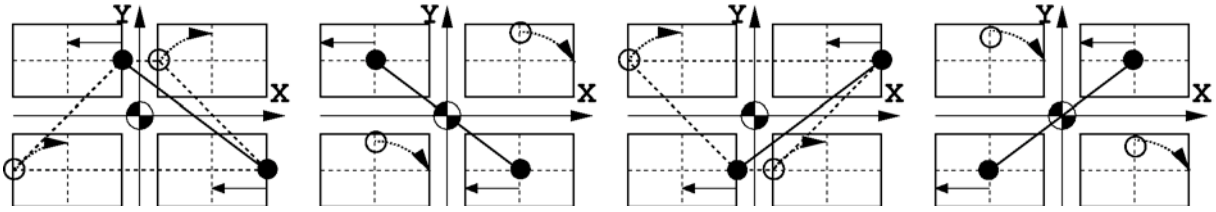


図 2.3: トロット ($\beta = 0.5$)

これらの歩行パターンは、各脚同士の位相差とデューティ比 (**duty factor :** 運動 **1** 周期中の接地率) によって分類することができる。例えば、全ての脚のデュー

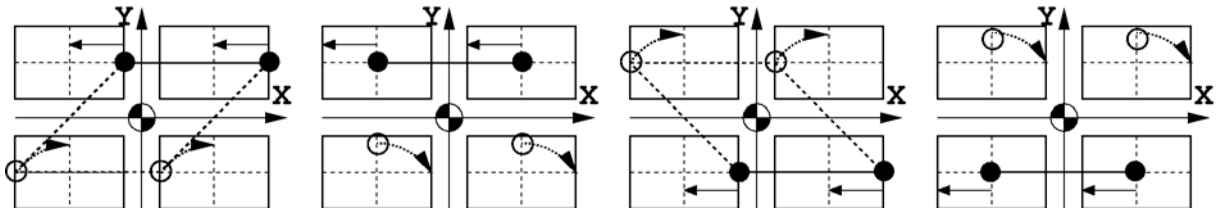


図 2.4: ペース ($\beta = 0.5$)

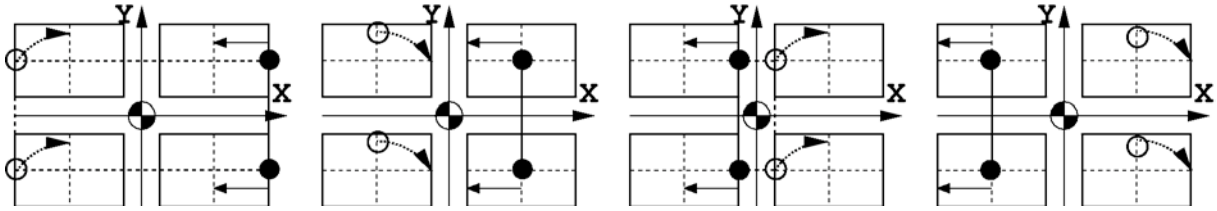


図 2.5: バウンド ($\beta = 0.5$)

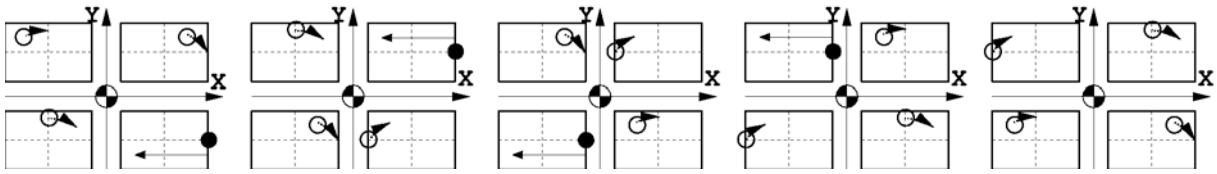


図 2.6: トランスバース ギャロップ（ネコ）

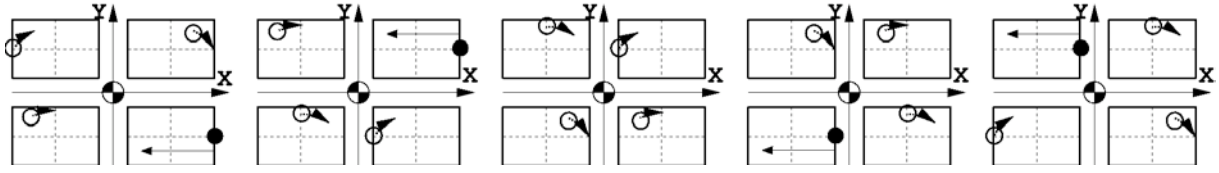


図 2.7: トランスバース ギャロップ（ウマ）

ティ比が等しい歩行パターンは正規歩容 (regular gate) といい、左右対称にある脚の位相差が 0.5 となっている歩行パターンは対称歩容 (symmetric gate) という。

最もゆっくりとした歩行パターンであるクロール **(crawl)** (図 2.2) は爬虫類、両生類などの低速歩行する四足動物でみることができる。デューティ比が 0.75 以上で常に 3 本以上の脚が接地しており静力学的に安定である。また、全ての脚のデューティ比が等しく、左右の脚の位相差が半周期ずれており 0.5 であるから正規対称歩容 **(regular symmetric gate)** に分類することができる。ロボットに用いる際には動力学的な安定化制御が必要なトロット (図 2.3)、ペース (図 2.4)、バウンド (図 2.5) も左右対称な歩行パターンであるが、歩行ではなく走行に分類される キャンター、ギャロップ 等の高速移動パターンになると左右の脚の対称性は崩れてくる。

この動物の周期的な運動パターンは、脊髄にあるとされている CPG(central pattern generator) という神経振動子によってリズムが生成され [16]、単位移動距離辺りの消費エネルギによってパターンが切り替わるものと考えられている [17]。

2.3.2 歩行パターンによるロボットの歩行

動物にみられる歩行パターンを 4 脚移動ロボットに応用する場合、静力学的に安定なクロール歩容が使用されることが多い。これは、クロール歩容をロボットの歩行パターンとして解析すると、クロール歩容が 4 脚正規対称歩容のなかで縦安定余裕を最大にする歩容 [24] であり、またデューティ比を 0.75 にとると静歩行の中では歩行速度を最も大きくとることができるからである。

デューティ比と歩行速度の関係は、遊脚の上下動の時間を無視すると式 (2.1) で表すことができる。

$$v = \frac{1-\beta}{\beta} v_{max} \tag{2.1}$$

ここで、v は歩行速度、v_{max} は脚振り速さの最大値、β はデューティ比を表している。この式から、歩行速度はデューティ比と反比例するため、歩行速度を上げるにはデューティ比を下げれば良いことがわかる。しかし、静的安定には少なくとも 3 本の支持脚が必要であるから、静歩行におけるデューティ比の最小値は 3/4 = 0.75 となり、静歩行において歩行速度を最大にするデューティ比は 0.75 となることがわかる。

このクロール歩容を 4 脚移動ロボットの歩行に応用した研究は大変多く、例としては、

基準側行歩容 [4]
: 前方に直進するだけのクロール歩容を全ての方向への並進に拡張し、並進方向と臨界側行角によりクロール歩容を使い分ける。

基準旋回歩容 [5]
: 任意の点回りの定常的な静歩行を実現。旋回中心位置に対して旋回角速度を

最大化する歩容を非線型最適化手法により得られた状態地図から選択し、クロール歩容と循環歩容を使い分ける。

間歇クロール歩容 [6]
4 脚支持相において胴体重心を NE 安定余裕 [30] が最大となる方向に移動するようクロール歩容を拡張。

などがある。クロール歩容以外でも、トロットやペース、バウンドといった歩行パターンが動力学的な安定化制御を行った上で実現されている [11][14][15]。

任意の方向への並進や旋回を含めた全方位歩行については、動歩行では文献 [11] において ZMP を規範としてフィードフォアワード的な動的安定化をはかりつつ間欠的なトロット歩容を用いて操縦者の入力に応じてリアルタイムで任意の方向に歩行する研究が報告されている。一方、静歩行における全方位歩行では文献 [12] において胴体に左右揺動を付加したクロール歩容と循環歩容を用いて前進後退、左右の旋回、その場旋回といった歩行パターン間を組み合わせて全方位歩行を実現した研究が報告されているが、斜め方向への並進や任意の旋回中心回りの旋回ができず、また、右旋回から左旋回や後退から左右旋回といった歩行パターン間を遷移する際に中間に別の歩行パターンを挟む必要などがあり、必ずしも全方位歩行を実現したとは言えなかった。

そこで本研究では、全方位歩行の 1 つ目の歩容計画アプローチとして任意の旋回中心回りのクロール歩容と循環歩容を組み合わせて 4 脚移動ロボットの連続的な全方位静歩行を実現する。

2.3.3　歩行パターンを用いないロボットの歩行

ロボットの歩行には動物の歩行に見られるようなパターンを用いない、ロボット特有の歩行方法が存在する。この歩行パターンに依存しない歩行を自由歩容 (**free gate**) という。この自由歩容はリズミカルな歩行が不可能な不整地などを歩行する場合の歩行パターンと考えられている。

自由歩容の計画手法も実に様々な手法が提案されている。例としては、

グラフ探索による自由歩容生成 [19][20]
ロボットの状態をノード、状態から状態に遷移する踏み替えをコストを付加したリンクにとり、評価関数には移動距離や安定性を組み込み、A^* アルゴリズムによるグラフ探索を用いて評価関数を最小化することで自由歩容を生成した。不整地における歩行も実現している。

モンテカルロ法による自由歩容生成 [21]
評価関数を「各脚の目標位置までの距離」と「胴体の目標位置までの距離」の総和、制約条件を「安定性」「可動範囲」「関節角度の許容範囲」とし、モンテカルロ法を用いて評価関数を最小化した。遊脚復帰順序は考慮せず、提案された手法により自動的に順序が決定される。

可動限界到達時の脚踏み変えルールに基づく歩行 [22]
: 歩行のための動作を「重心推進動作」と「胴体の移動を伴わない踏み替え」に分離し、4 脚支持で胴体を推進させて支持脚が可動限界に達したら止まって踏み替えを行うというルールで歩行した。一定速度で歩行する場合は間欠的なクロール歩容に収束するようなルールとなっている。

反射の組み合わせによる歩行 [23]
: 外界センサ入力に基づく複数の反射の組み合わせにより静歩行を実現した。視覚目標追従に基づき胴体を揺動・推進させ、足裏の力センサから求めた ZMP による安定余裕が一定値を下回ると踏み替えを行い安定余裕を一定値以上に保持するという動作により歩行を実現した。

などを挙げることができる。これらの研究事例は必ずしも自由歩容生成を目的としているものばかりではないが、歩行パターンを適用して歩行している訳ではないという点では共通している。また、実現できたのは並進のみで、旋回を含んだ全方位歩行が実現できていないという点も共通している。

そこで、本研究では２つ目の全方位歩行の歩容計画アプローチとして安定性と機動性を考慮した評価関数を最大化する自由歩容生成を用いて静的全方位歩行を実現する。

2.4 四脚移動ロボットのモデル化

本研究における全方位歩行の歩容計画に入る前に、本研究では図 2.8 に示すような四脚移動ロボットを取り扱うものとし、この四脚移動ロボットのモデル化を行う。まず、座標系はロボットの胴体重心を原点としロボットの前方を x 軸、上方を z 軸にとった胴体重心座標系を使用する。また、関節角度は図 2.8 の状態がすべて角度が 0 となっている状態とし、半時計回りを正の回転方向とする。脚番号の振り方は、胴体座標系の xy 平面の象限と一致するものとし、関節番号は脚の付根に近い方から第 1、第 2、第 3 関節と呼ぶものとする。

この胴体重心座標系から 図 2.8 のロボットを図 2.9 のようなリンクモデルに見立てると、第 i 脚の各関節角度と脚の姿勢とその 1 階微分を次式のように表すことができる。

$$p_i = b_i + f_i(\theta_i) \tag{2.2}$$
$$\dot{p}_i = J_i(\theta_i) \cdot \dot{\theta}_i \tag{2.3}$$

ここで、p_i は第 i 脚の先端位置、$\dot{p}_i$ は第 i 脚の先端速度、b_i は第 i 脚の付根位置、θ_i は第 i 脚の関節角度、$\dot{\theta}_i$ は第 i 脚の関節角速度を意味する。f_i、J_i はそれぞれ関節角度と脚先端位置の関係、関節角加速度と脚先端速度の関係を表してお

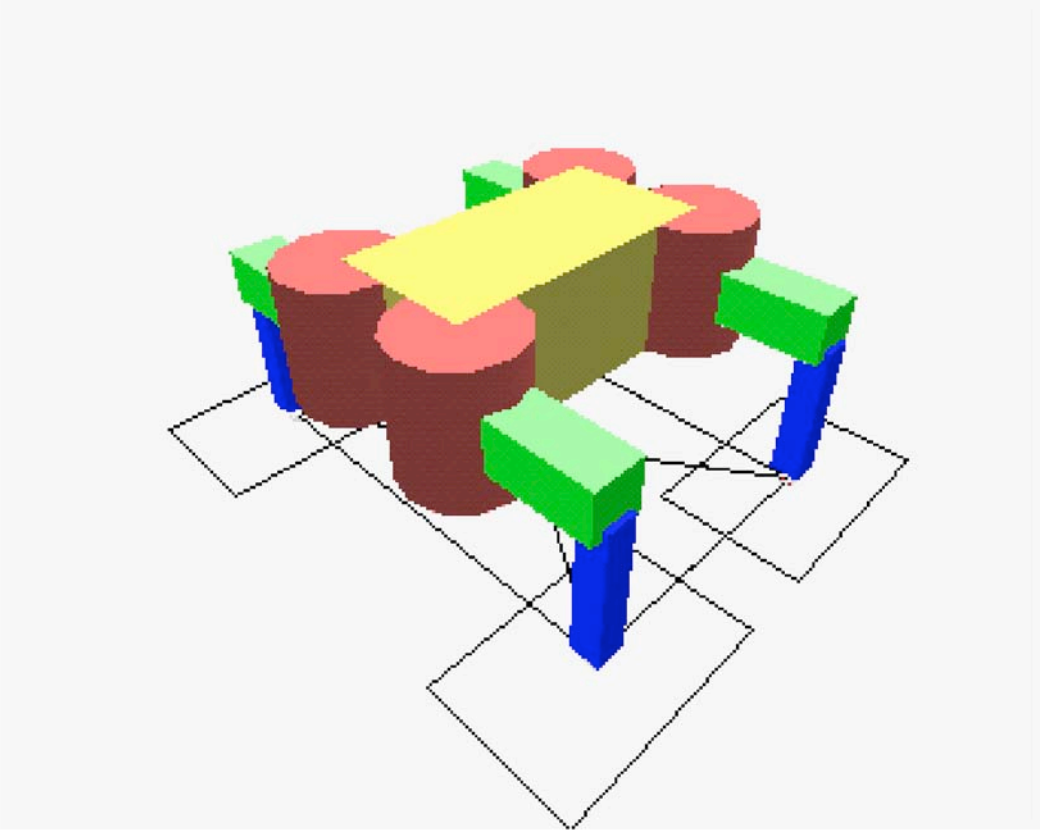


図 2.8: 四脚移動ロボットのモデル

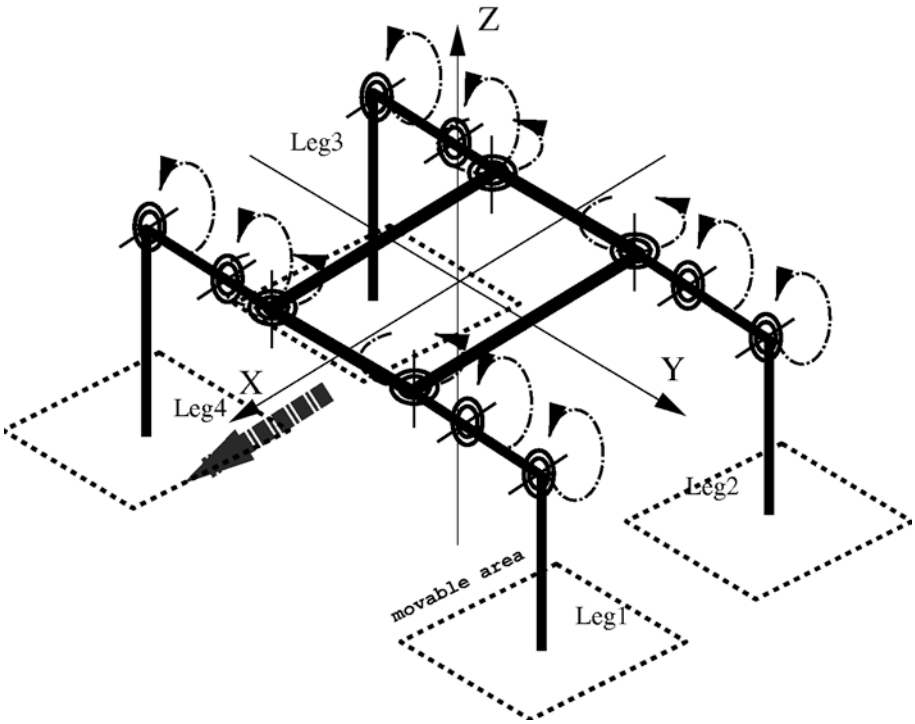


図 2.9: 四脚移動ロボットのリンクモデル

り、i=1 または i=2 のときは次式のような形をしている。

$$f_i(\theta_i) = \begin{bmatrix} -S_1(l_1+C_2l_2+S_{23}l_3) \\ C_1(l_1+C_2l_2+S_{23}l_3) \\ S_2l_2+C_{23}l_3 \end{bmatrix} \tag{2.4}$$

$$J_i(\theta_i) = \begin{bmatrix} -C_1(l_1+K_{23}) & S_1L_{23} & -S_1C_{23}l_3 \\ -S_1(l_1+K_{23}) & -C_1L_{23} & C_1C_{23}l_3 \\ 0 & K_{23} & S_{23}l_3 \end{bmatrix} \tag{2.5}$$

ここで l_1, l_2, l_3 はそれぞれ第 1 リンク、第 2 リンク、第 3 リンクの長さであり、それ以外の記号はそれぞれ、

$$\begin{aligned} S_j &= \sin\theta_j \\ C_j &= \cos\theta_j \\ S_{jk} &= \sin(\theta_j+\theta_k) \\ C_{jk} &= \cos(\theta_j+\theta_k) \\ K_{23} &= C_2l_2+S_{23}l_3 \\ L_{23} &= S_2l_2-C_{23}l_3 \end{aligned}$$

の通りである。i=3 または i=4 のときは第 1 関節の初期角度を i=1 または i=2 の場合に π だけ加えた形になる。

2.5 全方位歩行

全方位歩行の歩容計画について述べる前に、本研究における全方位歩行について、いくつかの定義をしておく。

まず、ロボットは水平面上において静歩行するものとする。次に、ロボットは人間の操縦入力により任意の歩行速度と旋回角速度を与えられるものとする。また、この操縦入力は歩行中に何度でも行うことができ、それに応じてロボットは歩行方向を変化させるものとする。

この全方位歩行を実現するためには任意の点回りの旋回ができれば良い。例えば、旋回中心を無限遠方にとれば並進を作ることができ、旋回中心を胴体重心の真下とればその場旋回、重心から離れたところにとれば緩やかな旋回を作ることができるからである。4 脚移動ロボットはこのような全方位歩行を実現するのに十分な自由度を備えている。そこで、本研究では操縦入力から与えられた歩行速度と旋回角速度から旋回中心を算出して任意の点回りに旋回することにより、全方位歩行を行うものとする。

歩行速度と旋回角速度から以下のようにして旋回中心の位置を求めることができる。まず、歩行速度を v_{in}, 胴体の鉛直軸回りの旋回角速度を θ_{in} とすると重心の

旋回半径 r_g は、次式のようになる。

$$r_g = \frac{|v_{in}|}{|\dot{\theta}_{in}|} \tag{2.6}$$

ただし、旋回角速度が0となる場合は旋回角速度の大きさを0でない十分小さい値で代用するものとする。

胴体重心から見て旋回中心 Q の位置は歩行速度の向きに対して常に直角であるから、旋回角速度が正の時に旋回中心が進行方向に対して左手に来るように旋回中心位置を与えると、次式のようになる。

$$\begin{aligned} Q &= Rot(k, \frac{\pi}{2}) \frac{v_{in}}{|v_{in}|} r_g \\ &= Rot(k, \frac{\pi}{2}) \frac{v_{in}}{|\dot{\theta}_{in}|} \end{aligned} \tag{2.7}$$

ここで $Rot(k, \frac{\pi}{2})$ はz軸回りに $\pi/2$ だけ回転させる回転変換行列である。旋回角速度が負になった場合は旋回中心が進行方向に対して右手に来るように旋回中心位置を与える。それには先ほどの式(2.7)の回転変換を逆 $\pi/2$ に取ればよい。このようにして求めた旋回中心の位置は、式(2.7)の右辺に -1 をかけたものとなるので、式(2.7)を次式のように拡張すれば旋回角速度の正負によらず旋回中心位置を与えることができる。

$$Q = Rot(k, \frac{\pi}{2}) \frac{v_{in}}{\dot{\theta}_{in}} \tag{2.8}$$

以上のように操縦入力で与えられた歩行速度と旋回角速度から旋回中心を求め、歩行パターンを用いた歩容計画と歩行パターンによらない歩容計画という2つのアプローチから、得られた旋回中心回りに静歩行するような全方位歩行を実現する。

2.6 2つの歩容計画アプローチ

本研究では全方位歩行を

1. 歩行パターンによる歩容計画
2. 歩行パターンによらない歩容計画

という2つの歩容計画アプローチにより実現する。

歩行パターンによる歩容計画は、静力学的に安定な歩行パターンであるクロール歩容と循環を旋回中心の位置によって使い分け、また歩行パターンの間を連続的に遷移させることによって全方位歩行を実現する歩容計画方法である。それに対して歩行パターンによらない歩容計画は、2歩1組の実行可能な全ての遊脚順序

に対し、安定性と機動性を考慮した評価関数を最大化するような遊脚順序とその着地点の探索を行う歩容計画手法である。

第 3 章では前者の実現方法として、連続的な全方位歩行を行うことを考慮して基準となる歩行パターンの脚配置等の計画を行い、進行方向に応じたクロール歩容や循環歩容等の歩容間をできる限り短時間で連続的に遷移させる手法について述べ、第 4 章では後者の実現方法として、安定性と機動性を含む評価関数や探索ルール、着地点決定の方法などについて述べる。

2.7 まとめ

この節では、4 脚移動ロボットの基礎的概念として主に安定性評価基準と歩行パターンについて述べた。また、本研究で使用する 4 脚ロボットのモデル化を行い、歩行速度と旋回角速度から旋回中心を算出する方法について述べ、全方位歩行の歩容計画について述べる準備を行った。

第3章　クロール歩容と循環歩容に基づく歩容計画

3.1　はじめに

四足ロボットの静歩行において、最も基本となるのはクロール歩容 (図 2.2) である。任意の旋回中心回りのクロール歩容は緩やかな旋回だけでなく、旋回中心を無限遠方にとることで並進にも利用すことができる。しかし、旋回中心が胴体重心に近くなるとクロール歩容より循環歩容 (図 3.3) の方が安定性や旋回速度の点で有利となることも知られている [5]。したがって、効率の良い定常的な静歩行を実現するには、任意の旋回中心回りのクロール歩容と循環歩容を用いればよい。さらに、これらの歩容間を連続的に遷移できると、全方位への連続的な静歩行が可能となる。

本章では、歩容遷移を視野に入れて任意の旋回中心回りのクロール歩容および循環歩容を計画し、これらの歩容間を連続的に遷移させることで静歩行による連続的な全方位歩行を実現する方法について述べる。

本研究ではクロール歩容と循環歩容に基づく全方位歩行を以下の手順で実現する。

1. 旋回中心位置から使用する歩容と脚軌道を決定する
   (a) 旋回中心の位置から歩容を選択する
   (b) 遊脚の着地位置と支持脚の終了位置を求める
   (c) 脚先端の軌道計画を行う
2. 旋回中心の変更に応じて歩容を連続的に遷移させる
   (a) 新しい旋回中心の位置から次の歩容と脚軌道を求める
   (b) 現在の歩容から新しい軌道へ順次脚を沿わせていく

本研究ではクロール歩容と循環歩容における接地率 β を 0.75 に固定し、遊脚時間 T_{sw} を 1.0[s] に固定するものとする。

3.2　基準歩容の計画

本節では、歩行パターンを用いた全方位歩行の基準歩容となる任意の旋回中心回りのクロール歩容と循環歩容の設計方法について述べる。

基準歩容は、できる限り少ない歩数で歩容遷移を完了させるために、遷移前の歩容と遷移後の歩容とで共通の脚配置が存在するように設計する。各歩容で共通となる脚配置には、図 3.1 のような脚基準位置 c_i を用いて脚軌道を計画する。クロール歩容と循環歩容は旋回中心の位置により選択するが、クロール歩容と循環歩容の設計の差は旋回中心の位置と遊脚順序だけであり、他の設計は全く同一である。

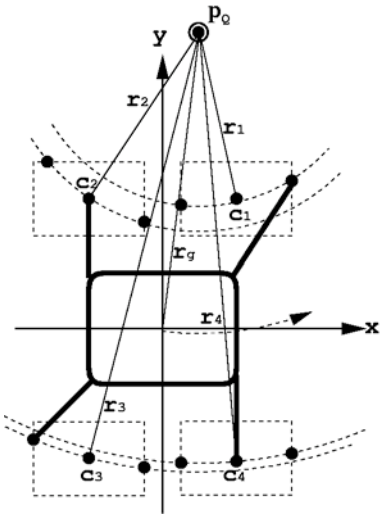


図 3.1: 基準位置を通過する軌道計画

ここで、普通の一般的なクロール歩容と全方位歩行のために脚配置を修正したクロール歩容とでそれぞれ出しうる歩行速度の限界で歩行した場合の動きの違いについて説明する。

図 2.2 の一般的なクロール歩容では支持脚のストロークが可動範囲を目一杯使っており、また、遊脚化を始める脚の位置や着地する位置なども可動範囲の限界に来ている。しかし、図 3.2 の全方位歩行のためのクロール歩容では支持脚のストロークが可動範囲の 3/4 となり、また、遊脚化を始める位置や着地する位置が前脚と後脚では異なる位置になっている。これは全方位歩行のために歩容遷移の前後で脚基準位置が共通な脚配置になるよう支持脚の軌道を修正したためである。

基準歩容の設計は、

1. 旋回中心の位置による歩容の選択
2. 遊脚の着地位置と支持脚の終了位置の算出
3. 脚先端軌道の計画

の順に行う。

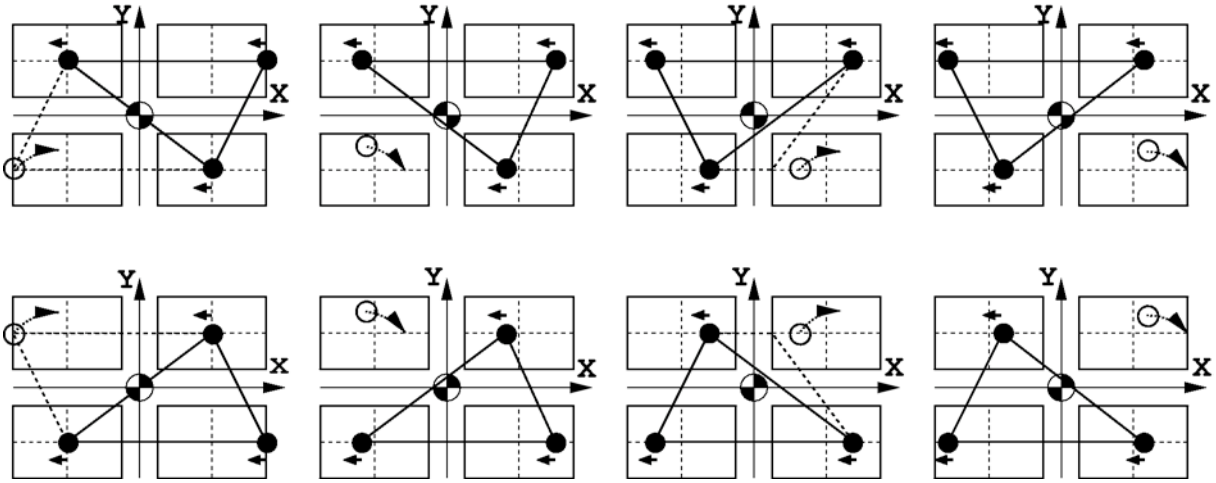


図 3.2: 全方位歩行のためのクロール歩容 ($\beta = 0.75$)

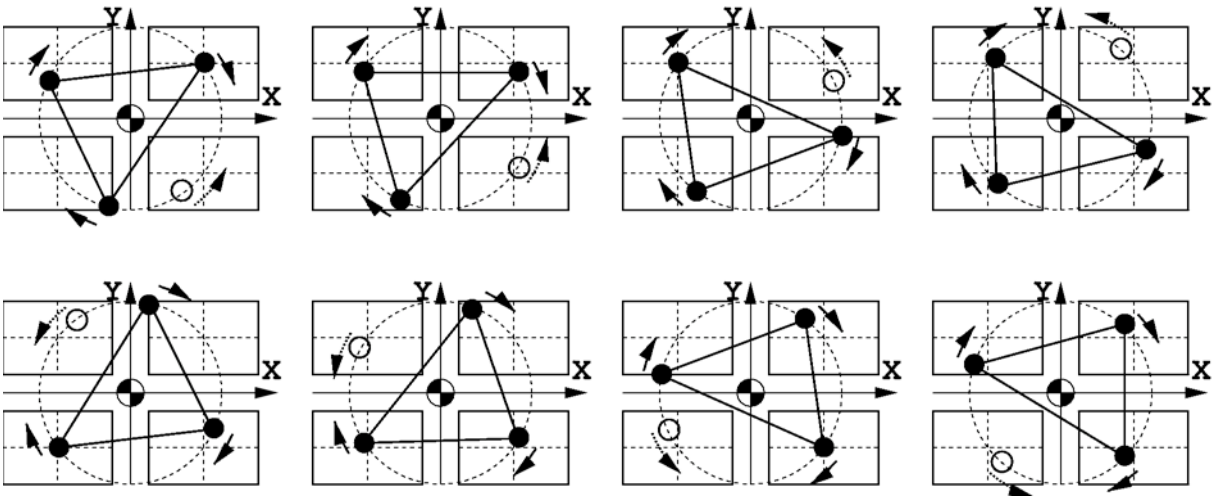


図 3.3: 全方位歩行のための循環歩容 ($\beta = 0.75$)

3.2.1　旋回中心の位置による歩容の選択

前章でも述べたように、本研究では人間による操縦からロボットに対して歩行速度と旋回角速度が指令されるものとする。歩行速度と旋回角速度が指令されると、まず、この指令を元に式 (2.6) と式 (2.8) により旋回半径と旋回中心の位置を求め、旋回中心の位置から歩容を選択する。

本研究では、旋回中心の位置と回転方向を図 3.4 に当てはめ、相当する歩容を図 3.5 から選択するものとする。例えば、旋回中心回りを反時計方向に旋回する場合、旋回中心が原点付近にあれば、図 3.4(a) の (5) に当たるので、図 3.5 から (5) の循環歩容を選択することになる。

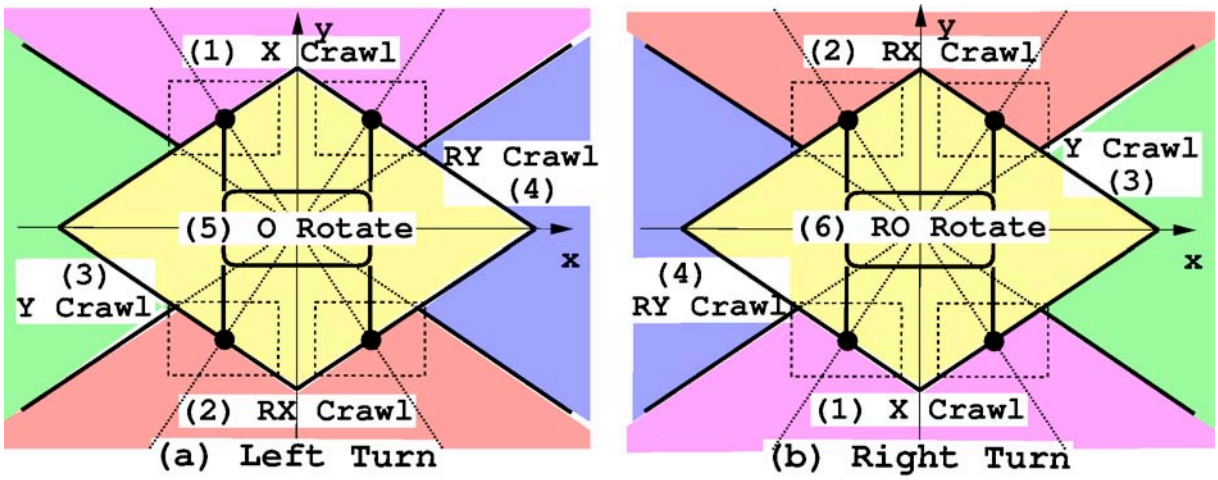


図 3.4: 旋回中心の位置による歩容の選択

このような旋回中心の位置による歩容決定の仕方では、歩行速度、旋回角速度も 0 で停止している場合は、旋回中心が胴体重心の真下に来るため循環歩容という扱いになる。したがって、歩行を開始する場合は循環歩容からの歩容遷移として扱い、歩行を停止する場合は循環歩容への歩容遷移として扱えば、停止した状態から歩行速度の上限までの歩行速度を連続的に取り扱うことが可能となる。

なお、図 3.4 は循環歩容を用いた場合の旋回中心位置と安定余裕の関係と、並進する場合にクロール歩容が切り替わる臨界側行角 [31] を旋回中心によって歩容を選択するように拡張して得たものである。ここでこの図 3.4 の導出について述べる。

循環歩容を用いる旋回中心位置の導出

循環歩容にはクロール歩容のような支持脚三角形が対角に切り替わる瞬間が無いため、旋回中心が胴体重心に近ければ安定余裕を常に正に保ったまま歩行が可能である。したがってできる限り循環歩容を用いたいが、循環歩容を用いる際に

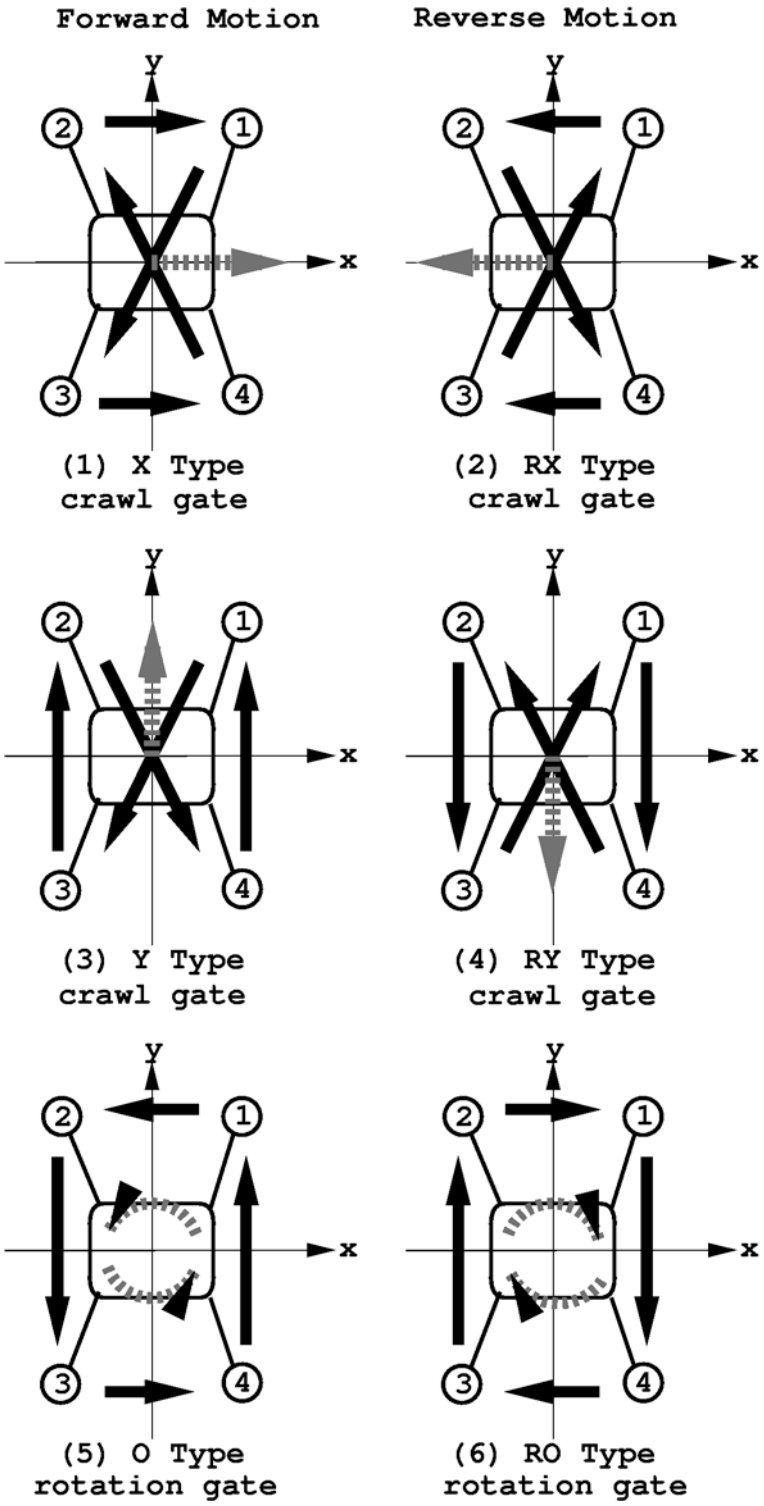

Forward Motion
Reverse Motion
y
x
1
2
3
4
(1) X Type
crawl gate
(2) RX Type
crawl gate
(3) Y Type
crawl gate
(4) RY Type
crawl gate
(5) O Type
rotation gate
(6) RO Type
rotation gate

図 3.5: 静歩行の種類

安定余裕が保てなくなる限界の旋回中心位置が不明である。そこで、第 2 章でモデル化した 4 脚移動ロボットのシミュレータを用いて、旋回中心を胴体座標系の第 3 象限にとり旋回角速度を固定して安定余裕が負になり始める旋回中心の位置を実験により求めた結果、図 3.6 のような安定余裕の等高線を得た。

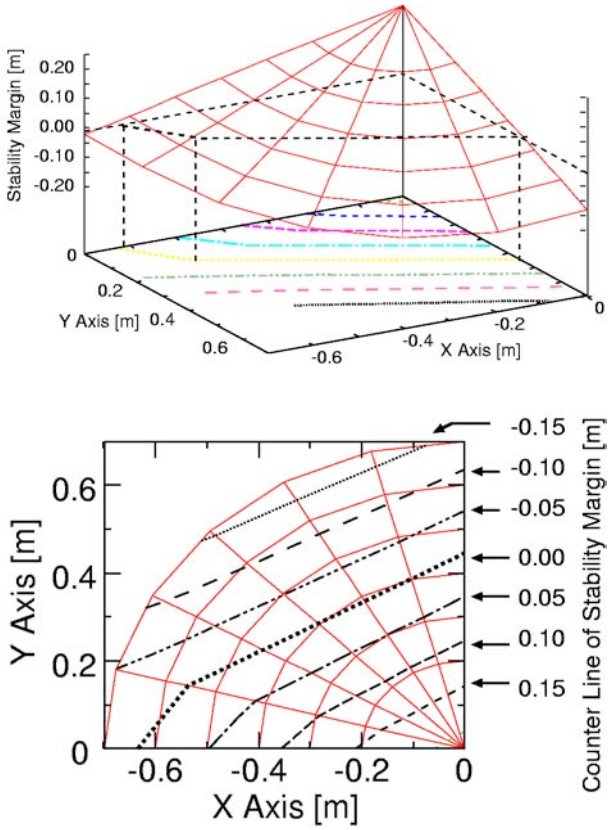


図 3.6: 循環歩容における旋回中心位置と安定余裕の等高線

この図の網目状の線の交点は旋回中心を取った位置であり、その点の高さはその旋回中心まわりに循環歩容で歩行した場合の安定余裕の値を表している。この図から、以下のようなことがわかる。

- 安定余裕が 0 となるラインはほぼ直線である。
- 旋回中心が胴体重心から遠ざかると、安定余裕はその距離に比例して減少する。
- 安定余裕は旋回中心が原点から脚基準位置に向かう直線に近いほど減少しやすく、胴体座標系の軸に近いほど減少しにくいという傾向がある。

これらの結果を元に、循環歩容を使用する限界となる境界線を次式のように考えた。

$$y = -\frac{p_{cix}}{p_{ciy}}x + \frac{p_{cix}^2 + p_{ciy}^2}{p_{ciy}} \tag{3.1}$$

この直線は、脚基準位置 p_{ci} を通り、傾きが原点から脚基準位置に向かう直線と直行する傾きとなっている。実験に用いたロボットの大きさをこの式に代入すると

$$y = \frac{2}{3}x + \frac{1.3}{3} \tag{3.2}$$

となり、脚基準位置を通りつつ x 軸上で安定余裕が 0 になる点と y 軸上で安定余裕が 0 になる点を結んだ直線とほぼ一致する。この直線より胴体重心よりであれば安定余裕が 0 にはならないので、この式 3.1 で表される直線を循環歩容を用いる領域の境界線として使用することにする。

クロール歩容を用いる旋回中心位置

クロール歩容は旋回中心が循環歩容になる領域より外側にある場合に使用するものとする。クロール歩容は進行方向の角度に応じて X 型、RX 型、Y 型、RY 型の 4 つに分類できるが、これらの使い分けは臨界側行角 [31] を旋回中心に応用して使い分ければ良い。

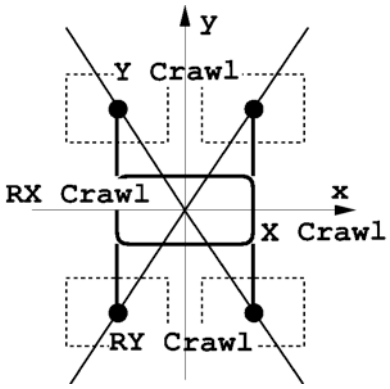


図 3.7: 臨界側行角によるクロール歩容の使い分け

ロボットの進行方向を元に臨界側行角を用いてクロール歩容の使い分けを行うと、図 3.7 のようになる。これを旋回中心に応用すると、旋回角速度が正の場合、旋回中心は進行方向からみて左手 $\frac{\pi}{2}$ だけ回転した方向にあるから、旋回中心を用いてクロール歩容を使い分けるには臨界側行角の分類を $\frac{\pi}{2}$ だけ回転させた分類を使用すれば良い。そうすると、旋回中心を用いたクロール歩容の使い分けは、図 3.4 のようになる。図 3.4 中の胴体の上にある菱形の領域は前節で提案した循環歩容を使用する旋回中心の領域である。

旋回中心の位置から使用する歩容を選択したら、続いて選択された歩容に応じて各脚に時間位相 t_i を与える。本研究では $t_i = 0$ を遊脚開始時刻,$t_i = T_{sw}$ を遊脚終了時刻とする。

3.2.2　遊脚の着地位置と支持脚の終了位置

次は、旋回中心 Q の位置と歩容、脚可動範囲から実行可能な最大旋回角速度を求め、遊脚の着地位置と支持脚の終了位置を算出する。本論文では脚可動範囲を長方形として扱うが、本手法は任意の脚可動範囲形状に対して適用可能である。

最大旋回角速度の算出

旋回中 Q の位置と脚可動範囲から実行可能な最大旋回角速度を求める。ある歩容から別の歩容に遷移する際に共通の脚位置がとれるよう、支持脚の軌道は必ず脚基準位置 p_{ci} を通過するよう計画するという条件から、各脚の旋回半径 r_i が次式のように求まる。

$$r_i = ||p_{ci} - p_Q|| \tag{3.3}$$

さらに、それぞれの脚基準位置 p_{ci} を旋回中心 Q からみた偏角 ϕ_{ci} で表すと、

$$\phi_{ci} = \tan^{-1} \frac{p_{ciy} - p_{Qy}}{p_{cix} - p_{Qx}} \tag{3.4}$$

と表すことができる。また、支持脚が描く円弧の軌道とその脚の可動範囲との交点のうち、進行方向にあるものを p_{fi}, 進行方向と反対側にあるものを p_{ri} とすると、p_{fi},p_{ri} の偏角はそれぞれ

$$\phi_{fi} = \tan^{-1} \frac{p_{fiy} - p_{Qy}}{p_{fix} - p_{Qx}} \tag{3.5}$$

$$\phi_{ri} = \tan^{-1} \frac{p_{riy} - p_{Qy}}{p_{rix} - p_{Qx}} \tag{3.6}$$

のように求めることができる。

これらの $\phi_{ci}, \phi_{fi}, \phi_{ri}$ に対して軌道分割係数 K_{qfi}, K_{qri} を用いて、θ_{fi}, θ_{ri} を

$$\theta_{fi} = \frac{|\phi_{fi} - \phi_{ci}|}{K_{qfi}} \tag{3.7}$$

$$\theta_{ri} = \frac{|\phi_{ri} - \phi_{ci}|}{K_{qri}} \tag{3.8}$$

のようにして求め、全ての脚の θ_{fi}, θ_{ri} のうち、もっとも小さいものを 1 周期辺りの最大旋回角度 θ_{max} とし、これを 1 周期の時間 T で割ったものを最大旋回角速度 $\dot{\theta}_{max}$ とする。すなわち、

$$\theta_{max} = min\{\theta_{fi}, \theta_{ri} : i = 1,2,3,4\} \tag{3.9}$$

$$\dot{\theta}_{max} = \frac{\theta_{max}}{T} \tag{3.10}$$

のように、実行可能な最大旋回角速度を求め、これにより旋回角速度の大きさを制限する。ここで軌道分割係数 K_{qfi}, K_{qri} は, 図 3.8 のように支持脚が可動範囲内に描く円弧軌道を歩容に応じて分割する係数である。

軌道分割係数 K_{qfi}, K_{qri} は歩容に応じてそれぞれ表 3.1, 表 3.2, 表 3.3, 表 3.4, 表 3.5, のように与える。

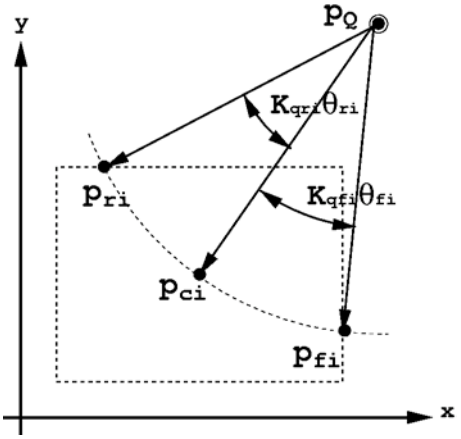


図 3.8: 脚軌道分割係数

表 3.1: X 型クロールの脚軌道分割係数

i	K_{qfi}	K_{qri}
1	0.5	$\beta-0.5$
2	$\beta-0.5$	0.5
3	$\beta-0.5$	0.5
4	0.5	$\beta-0.5$

表 3.2: RX 型クロールの脚軌道分割係数

i	K_{qfi}	K_{qri}
1	$\beta-0.5$	0.5
2	0.5	$\beta-0.5$
3	0.5	$\beta-0.5$
4	$\beta-0.5$	0.5

表 3.3: Y 型クロールの脚軌道分割係数

i	K_{qfi}	K_{qri}
1	$\beta-0.5$	0.5
2	$\beta-0.5$	0.5
3	0.5	$\beta-0.5$
4	0.5	$\beta-0.5$

表 3.4: RY 型クロールの脚軌道分割係数

i	K_{qfi}	K_{qri}
1	0.5	$\beta-0.5$
2	0.5	$\beta-0.5$
3	$\beta-0.5$	0.5
4	$\beta-0.5$	0.5

表 3.5: 循環歩容の脚軌道分割係数

i	K_{qfi}	K_{qri}
1	0.5	$\beta-0.5$
2	0.5	$\beta-0.5$
3	0.5	$\beta-0.5$
4	0.5	$\beta-0.5$

遊脚の着地位置

遊脚開始位置 p_{si} は $t_i = 0$ における脚先端位置である。遊脚終了位置、すなわち遊脚の着地位置の偏角 ϕ_{ei} は脚基準位置の偏角 ϕ_{ci}, 脚軌道分割係数 K_{qfi}, 1 周期辺りの旋回角度 θ を用いて

$$\phi_{ei} = \phi_{ci} + K_{qfi}\theta \tag{3.11}$$

と表すことができる。上式より、遊脚の着地位置 p_{ei} は

$$p_{ei} = \begin{bmatrix} p_{Qx} + r_i cos\phi_{ei} \\ p_{Qy} + r_i sin\phi_{ei} \\ -p_{ciz} \end{bmatrix} \tag{3.12}$$

のように求めることができる。

支持脚の終了位置

支持脚の開始位置は遊脚の着地点 p_{se} であり、支持脚の終了位置 p_{hi} は $t_i = T$ における脚先端位置である。支持脚終了位置の偏角 ϕ_{hi} は遊脚終了位置の偏角 ϕ_{ei} と旋回角速度 $\dot{\theta}$ をもちいて

$$\phi_{hi} = \phi_{ei} - \dot{\theta}(T - T_{sw}) \tag{3.13}$$

と表すことができるので、支持脚終了位置 p_{hi} は、

$$p_{hi} = \begin{bmatrix} p_{Qx} + r_i \cos\phi_{hi} \\ p_{Qy} + r_i \sin\phi_{hi} \\ -p_{ciz} \end{bmatrix} \tag{3.14}$$

$$\tag{3.15}$$

のように求めることができる。

3.3 連続的な歩容遷移

前節までの計算で任意の旋回中心回りの定常的な歩行に必要なパラメータの算出が終了する。この節では歩行中に旋回中心を変更し、歩容を遷移させる場合について述べる。

歩容を遷移させる場合、少ない歩数で遷移を完了するには遷移の前後でできる限り多くの脚が共通する脚配置をとることが望ましい。そのために前節では対角三角切り替え点 (DTE 点)[13] において支持脚対角線をなす 2 脚が脚基準位置にくるよう脚軌道を計画し、この 2 脚が歩容遷移前後の共通の脚配置をとるように歩容を計画した。こうすることにより、DTE 点は遷移する前後の歩容に挟まれるため、歩容遷移は DTE 点の 1 歩手前から行うものとする。

新しい旋回中心が与えられた場合、新しい旋回中心を用いて歩容の選択と脚軌道計画を行い、DTE 点の 1 歩手前から新しい歩容の脚軌道に脚を順次に沿わせて行く。この脚を新しい軌道に沿わせて行く順序は歩容遷移のパターンに依存する。本節ではこの連続的な歩容遷移の手順について詳しく述べる。

3.3.1 歩容遷移のパターン

歩容遷移のパターンには、図 3.5 に示した X,RX,Y,RY,O,RO Type の 6 種類の各歩容から各歩容への遷移があり、単純計算で $6^2 = 36$ 通りの遷移パターンがある。さらに、クロール歩容からの遷移パターンには後述するように、それぞれクロール歩容からクロール歩容が 2 ケース、クロール歩容から循環歩容が 2 ケースを持っているため、歩容遷移のパターンは $4\times6\times2+2\times6=60$ 通りが存在することになる。ここから実際には起こり得ないパターンを差し引くと、起こり得る歩容遷移のパターンは $60-4\times2=52$ 通りであることがわかる。

しかし、実際には 52 通りの遷移パターンを用意する必要はなく、大きく分けて次の 4 つのパターンを用意すれば良い。

- クロール歩容から異なるクロール歩容
- クロール歩容から循環歩容
- 循環歩容からクロール歩容
- 循環歩容から循環歩容

この節ではこれらの 4 つの遷移パターンについて、歩容遷移を連続かつ安定に、できる限り少ない歩数で行う手順を述べる。

3.3.2 クロール歩容からクロール歩容への遷移

クロール歩容からクロール歩容への遷移手順は、歩容遷移開始時に遊脚化する脚により次の 2 つのケースに分けることができる。

case 1 : 遷移 1 歩目の遊脚が遷移後の進行方向に対して前脚となる場合

case 2 : 遷移 1 歩目の遊脚が遷移後の進行方向に対して後脚となる場合

これら 2 つのケースの例をそれぞれ図 3.9, 図 3.10 に示す。ここで各脚には以下のような記号を付した。

U step1 で遊脚化する脚
C 脚 U の対角にある脚
A 脚 U,C 以外で遷移前の進行方向に対して前脚となる脚
B 脚 U,C 以外で遷移前の進行方向に対して後脚となる脚

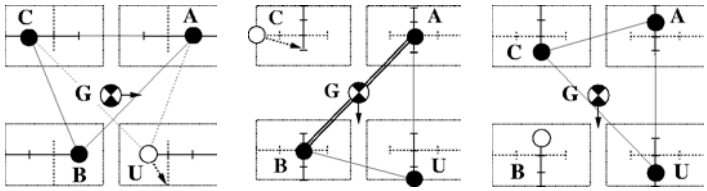


図 3.9: クロール歩容からクロール歩容への遷移 (ケース 1)

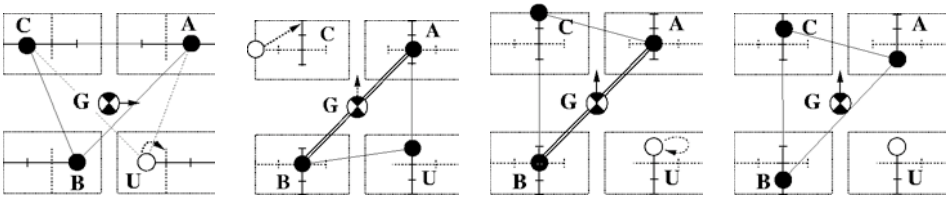


図 3.10: クロール歩容からクロール歩容への遷移 (ケース 2)

図 3.9、図 3.10 の step1 は DTE 点の 1 歩手前の状態である。step1 では 1 歩目の脚 U を遊脚運動させて次の歩容の軌道に沿わせながら遷移を開始する。step2 は DTE 点であり、脚 A,B が基準位置に来るため、この時点で 4 脚中 3 脚が次の歩容の脚軌道に沿うことになる。このとき、重心 G が脚 A,B によってつくられる切り替え辺上に存在するため、脚 C が遊脚化可能となり、step3 で脚 C を次の歩容の脚軌道に沿わせることで歩容の遷移が完了する。

ここで、ケース 1 のように脚 U が遷移後の進行方向に対して前脚となる場合は、図 3.9 のように step2 の時点で重心の進行方向に支持脚三角形が形成されるため、胴体が停止せずに歩容の遷移を行うことができる。

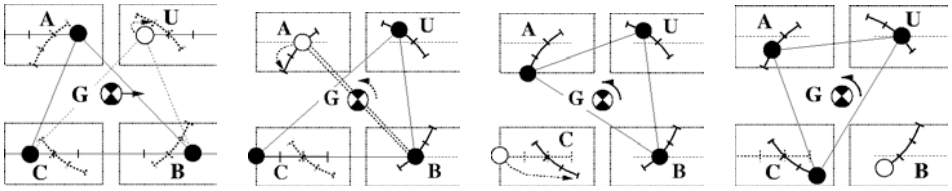
しかし、ケース 2 のように脚 U が遷移後の進行方向に対して後脚となる場合は、図 3.10 のように step2 の時点で進行方向に支持脚三角形が形成されないため、step2 で胴体の移動を遊脚時間 1 回分だけ停止させ、step3 で進行方向に支持脚三角形が形成されるのを待つ必要がある。

このクロール歩容からクロール歩容への遷移の具体的な遊脚順序を付録の表 A.1, 表 A.2 に示す。

3.3.3 クロール歩容から循環歩容への遷移

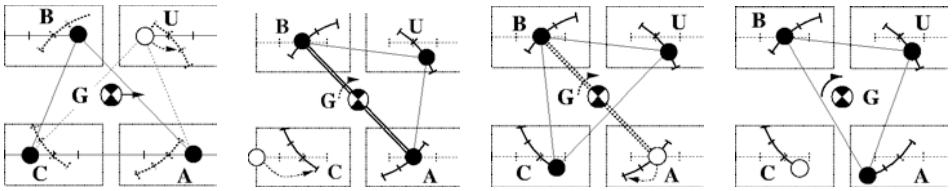
クロール歩容から循環歩容への遷移手順は、遷移 2 歩目の脚配置から 2 つのケースに分けることができる。これらの 2 つのケースを図 3.11, 図 3.12 に示す。各脚には以下のような記号を付した。

U　step1 で遊脚化する脚
C　脚 U の対角にある脚
A　遷移後の遊脚順序が脚 U の次になる脚
B　脚 A の対角となる脚



step 1　step 2　step 3　step 4

図 3.11: クロール歩容から循環歩容への遷移 (ケース 1)



step 1　step 2　step 3　step 4

図 3.12: クロール歩容から循環歩容への遷移 (ケース 2)

step1 は DTE 点の 1 歩手前であり、クロール歩容からクロール歩容への遷移と同様、1 歩目の脚 U を次の軌道に沿わせながら歩容遷移を開始する。このとき、step 1 においてのみだが、脚 U の着地位置を新しい歩容におけるその脚の着地位置から $2\dot{\theta}T_{sw}$ 分だけ脚が移動した後の位置にとっておく。これは、歩容遷移中にはいる胴体の一時停止解除後に、脚 U がちょうど循環歩容の脚位置に来るように合わせるためである。

step1 から step2 に移る際に、脚 A が遊脚化可能であるかどうかによって遷移パターンは 2 つに別れる。

case 1

図 3.11 のように step2 で脚 A が遊脚化可能な場合、脚 A を遊脚化し、次の歩容の脚軌道に沿わせる。脚 B はこの時点で脚基準位置にあり、すでに次の歩容の軌道に沿っていると見なす。ここで、step2 では脚 C の旋回分の脚可動範囲が保証されないため、胴体の移動を 1step、遊脚時間 1 回分だけ停止させる必要がある。その後、step3 で脚 C を次の歩容の軌道に沿わせ、歩容の遷移が完了する。

case 2

図 3.12 のように step2 で脚 A が遊脚化できない場合は、脚 C を遊脚化し次の歩容の脚軌道の終端位置に沿わせる。ケース 2 の step2 では脚 A、B が切り替え辺を形成するため、重心が辺 AB 上に乗り、脚 C が遊脚化可能となる。step3 では脚 A が

遊脚化可能となり、これを次の歩容の軌道に沿わせると歩容の遷移が完了する。脚 B はケース 1 と同様に、step2 の時点で脚基準位置にくるため、すでに次の歩容の軌道に沿っているものとみなし、遊脚運動を省略する。ここで、step3 では脚 A の遊脚化を保証するために、脚 C を支持脚軌道の終点に移動しているため、脚 C についての旋回分の可動範囲が保証されない。そのため、胴体を step2,step3 と停止させる必要がある。

このクロール歩容から循環歩容への遷移の具体的な遊脚順序を付録の表 A.3, 表 A.4 に示す。

3.3.4 循環歩容からクロール歩容への遷移

循環歩容からクロール歩容への遷移は、循環歩容のどのような脚配置からでも実行可能である。その遷移遷移手順を図 3.13 に示す。各脚には以下のような記号を付した。

U step1 で遊脚化する脚
C 脚 U の対角にある脚
A 脚 U,C 以外で遷移後の進行方向に対して前脚となる脚
B 脚 U,C 以外で遷移後の進行方向に対して後脚となる脚

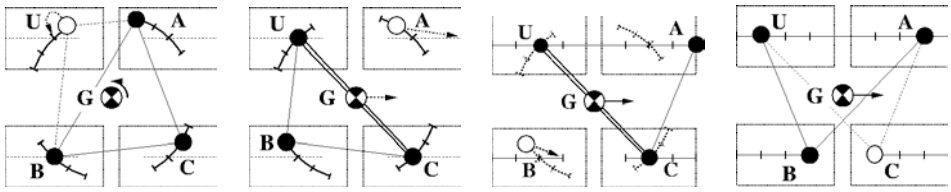


図 3.13: 循環歩容からクロール歩容への遷移

step1 では脚 U を遊脚化し、脚基準位置に移動させて遷移を開始する。脚 U が基準位置にくると脚 C も基準位置にくるため、step2 開始時点では辺 UC 上に重心 G が乗る。すると脚 B,A が遊脚化可能となるため、脚 A を遊脚化し、次の歩容の軌道に沿わせる。脚 A を先に遊脚化したのは、遷移後の進行方向に支持脚三角形を形成するためである。step3 では残った脚 B を遊脚化し、次の歩容の軌道に沿わせることで歩容の遷移を完了する。

この循環歩容からクロール歩容への遷移の具体的な遊脚順序を付録の表 A.5 に示す。

3.3.5 循環歩容から循環歩容への遷移

循環歩容から循環歩容への遷移も、循環からクロールへの遷移と同様、どのような脚配置からでも実行可能である。循環歩容を用いて歩行する場合、旋回中心

が脚基準位置を円周とした円内にある限りは不安定になることはないため、場合分けは必要ない。

その遷移手順を図 3.14 に示す。各脚には以下のような記号を付した。

U　step1 で遊脚化する脚
C　脚 U の対角にある脚
A　遷移後の遊脚順序が脚 U の次になる脚
B　脚 A の対角となる脚

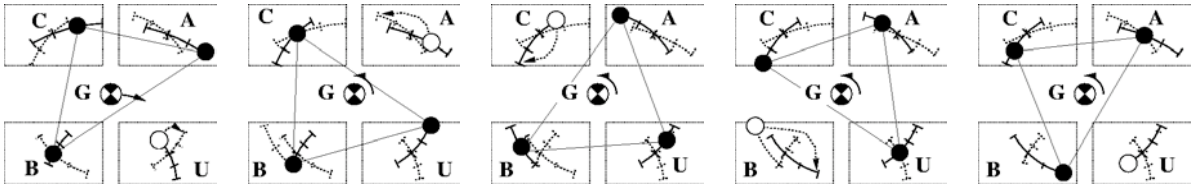


図 3.14: 循環歩容から循環歩容への遷移

循環歩容から循環歩容への遷移では、遷移後の遊脚順序にしたがって脚軌道を遷移後の軌道に順次沿わせて行けば、歩容遷移が 4step で完了する。

この循環歩容から循環歩容への遷移の具体的な遊脚順序を付録の表 A.6 に示す。

以上のような各歩容遷移手順に、さらに「歩容遷移中、遊脚運動前の支持脚位置と遊脚運動後の着地位置が同じ位置になる場合は遊脚運動を省略する」 というルールを付加して歩容遷移時間の短縮を図る。

このようにして得られる基準歩容と歩容遷移をもとに支持脚、遊脚の脚先端軌道を計画して脚先端ベースの軌道追従制御を行うことで歩行パターンに基づく全方位歩行が可能となる。

3.3.6　歩容遷移時間の比較

これまでに述べたそれぞれの歩容遷移パターンが完了するまでの時間を表 3.6 に示す。

もっとも遷移が早く完了するのはクロール歩容からクロール歩容への遷移の case 1 である。遷移が完了するまでにもっとも時間がかかる循環歩容から循環歩容への遷移の場合でも、遷移開始と同時に胴体は新しい旋回中心回りに運動を始めるため、進行方向自体は連続的にすばやく切り替わっている。また、歩容遷移中に 1step ないし 2step 胴体の移動を停止する場合でも、十分に連続的な歩容遷移が可能である。

表 3.6: 歩容遷移時間の比較

→	X		RX		Y		RY		O		RO	
case	1	2	1	2	1	2	1	2	1	2	1	2
X	1/2T	—	—	3/4T	1/2T	3/4T	1/2T	3/4T	3/4T	3/4T	3/4T	3/4T
RX	—	3/4T	1/2T	—	1/2T	3/4T	1/2T	3/4T	3/4T	3/4T	3/4T	3/4T
Y	1/2T	3/4T	1/2T	3/4T	1/2T	—	—	3/4T	3/4T	3/4T	3/4T	3/4T
RY	1/2T	3/4T	1/2T	3/4T	—	3/4T	1/2T	—	3/4T	3/4T	3/4T	3/4T
O	3/4T		3/4T		3/4T		3/4T		T		T	
RO	3/4T		3/4T		3/4T		3/4T		T		T	

X,RX,Y,RY,O,RO は図 3.5 の歩行パターンに準拠する

ただし、遊脚運動の前後で遊脚運動する脚の位置が変わらない場合は、歩容遷移時間は表 3.6 より短くなる。例えば、クロール歩容を用いた並進中に進行方向が 180 度逆向きに変化する場合は必ずクロール歩容からクロール歩容への遷移の case 2 になるため歩容遷移完了までの時間は遊脚時間 3 歩分の $3/4T$ だが、この場合は step2 の脚 C の遊脚運動が省略されて歩容遷移時間は遊脚運動 2 歩分の $1/2T$ となる。また基準姿勢で停止した状態からの歩行開始時や、歩行状態から基準姿勢で停止する場合にも歩容遷移中に無駄な遊脚運動が含まれるためこの遊脚運動を省略するルールが適用されるなど、特定の場合では歩容遷移時間が表 3.6 より短くなる。

3.4 まとめ

本節では、全方位歩行用に支持脚軌道を調整したクロール歩容と循環歩容を組み合わせ, これらの歩容間を連続的に遷移させる手順について述べた。

第4章　遊脚順序探索と着地点探索に基づく自由歩容の計画

4.1　はじめに

全方位歩行の歩容計画の 1 つ目のアプローチとして、前章ではクロール歩容と循環歩容を用いた歩容計画方法について述べた。この章ではもう 1 つのアプローチとして、歩行パターンを使用しない自由歩容の計画方法について述べる。

4.2　遊脚運動に関する考察

自由歩容の計画方法を論ずる前に、遊脚運動の必要性、遊脚順序とその着地点について考察する。

4.2.1　遊脚運動の必要性

胴体を目的の方向に移動させる場合、できる限り安定性を大きく保つことが望ましい。一般に 3 脚支持よりは 4 脚支持のほうが安定性を大きくとることが可能であるから、できる限り 4 脚支持で胴体を推進するのが望ましい。しかし、胴体を推進させると重心が進行方向にある支持脚多角形の辺に近づくため次第に安定性が低下してゆくという問題がある。そこで支持脚多角形を更新することで安定性を回復させることが必要となる。

また、4 脚支持のまま胴体を推進させるといずれかの脚が可動限界に達してしまうという問題もある。そこで可動限界に達しそうな脚を持ち上げて進行方向に着地させて機動性を回復させることが必要となる。このように、遊脚運動は安定性と機動性の両方を回復させるために必要な運動なのである。

この考察から、本研究では自由歩容における遊脚運動を開始するタイミングを安定性か機動性のどちから一方でも既定値を下回った時点とすることにする。

4.2.2　遊脚順序

クロール歩容や循環歩容の様に定型的な歩行パターンを使用する場合は、その歩行パターンにしたがった規則的な運動をしている限り、安定性が崩れる前に周

期的に支持脚多角形が更新され、支持脚も可動限界に達する前に遊脚化のタイミングが訪れるため安定性も機動性も十分に保証されている。しかし、歩行中に進行方向が大きく変化したり、不整地などで地面に着地できない領域が存在する場合などは規則的なパターンを適用することは難しく、その場面に応じて有効な遊脚順序を選択する必要がある。

このとき、遊脚運動を1歩分しか考えないとその1歩目の遊脚運動が実行できてもその次の2歩目の動作につながらない状態が発生しやすい。この考察から、本研究における自由歩容の遊脚順序は、1歩分だけでなくその先につながる2歩目まで考慮して計画するものとする。

4.2.3 遊脚の着地点

遊脚運動が安定性と機動性の回復を目的としているため、遊脚の着地点は着地後の安定性と機動性をある一定値以上に回復できる位置である必要がある。また、遊脚順序を2歩目まで考慮するため、1歩目の遊脚の着地点は2歩目の遊脚運動を保証できるような位置であることが望ましい。

この考察から、本研究における自由歩容計画では遊脚順序とその着地点探索を組にして、2歩目の遊脚運動を保証できるような1歩目の着地点探索を行うものとする。

4.3 評価関数に基づく自由歩容計画

本研究では、実行可能な全ての2歩1組の遊脚順序に対し2歩目の遊脚運動を保証するような1歩目の着地点探索を行い、安定性と機動性を含む評価関数が最大となるような遊脚順序とその着地点を選択することで自由歩容を計画する。

自由歩容計画は4脚支持の状態で行い、計画により得られた遊脚順序とその着地点は1歩のみを実行する。1歩目の遊脚が着地した後、全ての遊脚順序に対して再び評価を行い、前の計画で得られた評価は考慮しないものとする。

ここで遊脚運動が「実行可能」であるとは、

- その脚が遊脚運動をしている間、安定性を保てること
- 全ての支持脚が遊脚時間1回分の間に可動範囲外に出ないこと
- 2歩目の遊脚運動を保証できる1歩目の着地点が存在すること

といった条件を満たしていることを意味する。

また、安定性と機動性を含む評価関数は次式のように定義する。

$$f_{sm} = \frac{l_{sm}}{l_{max}} \tag{4.1}$$

$$f_{km} = \frac{t_{km}}{t_{max}} \tag{4.2}$$

$$f = k_s f_{sm} + (1-k_s) f_{km} \tag{4.3}$$

ここで l_{sm} は安定余裕、l_{max} はその支持脚の配置で得られる安定余裕の最大値であり、t_{km} は運動余裕、t_{max} は各脚の最大運動余裕の最小値である。

安定余裕は一般に「胴体重心投影点から支持脚多角形の辺までの距離の最小値」で定義される。運動余裕は、本研究では「現在の支持脚位置から可動範囲に到達するまでの時間」で定義するものとする。遊脚に関しては運動余裕を定義しない。

安定余裕と運動余裕とでは単位や大きさが異なるため、それぞれその姿勢で取りうる安定余裕の最大値、運動余裕の最大値で正規化する。これらの値に 係数 k を用いて重み付をしたものを評価関数とする。

以上のような評価関数を用いて遊脚順序と着地点を評価し、この評価値が最も大きくなるような遊脚順序とその着地点を実行することにより歩行を実現する。

4.4 基本探索

基本探索は大きく分けると、遊脚候補の選択と着地点探索に分けることができる。この節では、自由歩容計画の中心となる基本探索について述べ、特に基本探索の中心となる着地点探索について重点的に述べる。全ての探索に失敗した場合は、歩行速度を抑える、胴体を横に動かすといった一時的な歩行速度の調整を行うが、これらについては次節で述べる。

4.4.1 遊脚候補の選択

遊脚順序と着地点の探索は 2 歩を 1 組とした 16 通りの組合わせのうち、実行可能なもののみを評価する。ここで遊脚運動が「実行可能」であるとは、

- その脚が遊脚運動をしている間、安定性を保てること
- 全ての支持脚が遊脚時間 1 回分の間に可動範囲外に出ないこと
- 2 歩目の遊脚運動を保証できる 1 歩目の着地点が存在すること

といった条件を満たしていることである。また、間に何の動作も挟まずに同じ脚が連続する遊脚順序は無駄な動作であるから評価の対象から除外する。

4.4.2 着地点探索

1 歩目の遊脚の着地点は、2 歩目の遊脚運動が安定を保ったまま実行できるか否かに大きな影響を与える。それゆえ、連続して歩き続けるためには次の 2 歩目が確実に実行できるような位置に 1 歩目の遊脚を着地させなければならない。しがたって、1 歩目の遊脚の着地点は「**2** 歩目の遊脚運動を保証するために **1** 歩目の遊脚が着地すべき領域」と「**1** 歩目の脚の可動範囲」の積集合となる領域 から探索を行って求めることになる。

「2 歩目の遊脚運動を保証するために 1 歩目の遊脚が着地すべき領域」とは、1 歩目の遊脚がその領域に着地していれば 1 歩目の着地直後に 2 歩目が地面を離れてから着地するまでの間に安定余裕が設定した最低値を下回らないような領域である。

このような領域は、次のように求めることができる。先ず 1 歩目、2 歩目の遊脚順序を 脚 i1, 脚 i2 の順番と仮定し、逆に遊脚化しない脚を 脚 sp1、脚 sp2 とする。この支持脚 sp1, sp2 の遊脚運動 2 回分後だけ時間が経った後の位置をそれぞれ p'_{sp1} ,p'_{sp2} とし、これらの点から胴体重心（すなわち座標系の原点）にそれぞれ直線を引くと図 4.1 のように p'_{sp1} ,p'_{sp2} からみて胴体重心の向こう側に 2 直線に挟まれた領域ができる。この領域が安定余裕の最低値を 0 とした場合の「2 歩目の遊脚運動を保証するために 1 歩目の遊脚が着地すべき領域」である。1 歩目の遊脚 i1 がこの領域内に着地していれば、2 歩目の遊脚 i2 が次に遊脚化してから着地するまでの間、胴体重心が脚 i1 の着地点と p'_{sp1} ,p'_{sp2} が作る三角形の外側には出ないため安定性を保証することができる。

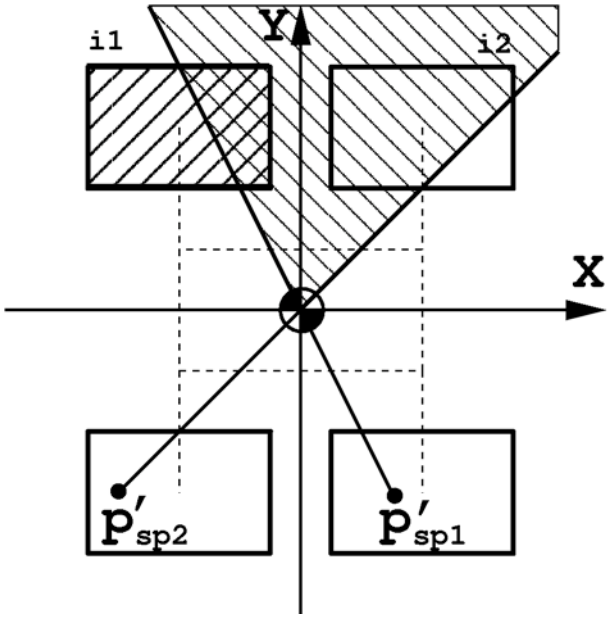


図 4.1: 2 歩目の遊脚運動を保証する領域

これを安定余裕の最低値が一定値以上になるよう拡張するには、図4.1のように p'_{sp1} ,p'_{sp2} から胴体重心に引いた直線を、図4.2のように胴体重心を中心とし最低安定余裕を半径とする円に引いた接線に変えれば良いのだが、ひとつの点からこの円に対して2本の接線が引けるため、接線同士の交点が4つできてしまい、4つの交点の中から p'_{sp1} ,p'_{sp2} からみて重心より向こう側にある点を選ぶ必要がある。こうして選ばれた点を p_{sl} とすると、ベクトル $p'_{sp1} - p_{sl}$, $p'_{sp2} - p_{sl}$ に挟まれた領域が「2歩目が遊脚運動中の最低安定余裕を保証するために1歩目が着地すべき領域」となる。

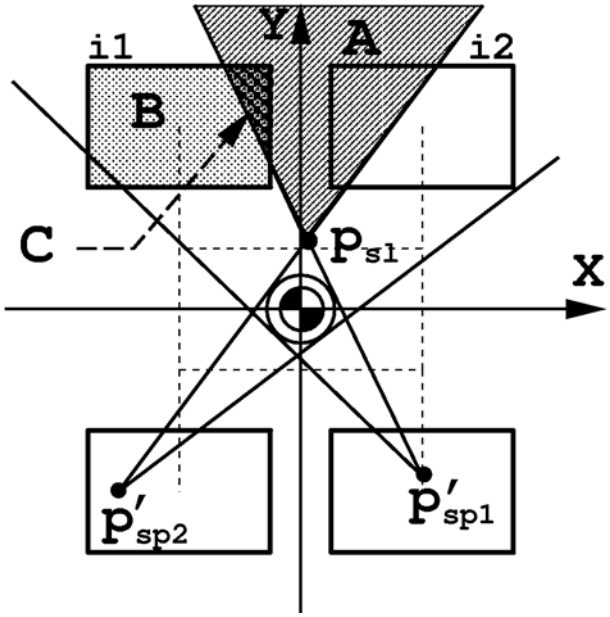


図 4.2: 2歩目の遊脚運動を保証する領域 (その 2)

このような領域と脚可動範囲との積集合となる領域が着地点探索を行う領域なのだがここでもう1つ、探索区間を旋回中心を中心とした円軌道上に制限し、着地点探索の区間を1次元に落として単純化する。この円軌道は遊脚が着地した後は支持脚軌道となる。こうして得られた円弧軌道上の着地点探索区間で評価関数の値を最大化するような点を求め、これをその遊脚順序における1歩目の遊脚の着地点とする。

この流れを箇条書きにまとめると、次のようになる。

1. 遊脚化しない脚 sp1, sp2 の遊脚時間2回分後の位置 p'_{sp1},p'_{sp2} を求める
2. p'_{sp1} を通り最低安定余裕を半径とした円に接する2本の線と、p'_{sp2} を通り最低安定余裕を半径とした円に接する2本の線の合わせて4本の接線を求める。
3. p'_{sp1},p'_{sp2} を除いた接線同士の交点を4つ求める

4. 4 つの交点のうち、ベクトル $-kp'_{sp1}, -kp'_{sp2}$ に挟まれた領域にある 1 点を選び、これを p_{sl} とする

5. ベクトル $p'_{sp1}-p_{sl}, p'_{sp2}-p_{sl}$ に挟まれた領域と脚可動範囲との積集合となる領域を求める

6. この領域内を通過する支持脚軌道を求め、着地点探索区間とする

7. 着地点探索区間内で評価関数を最大にする点を求める

以下では、これらの計算について詳しく述べる。

遊脚化しない脚の遊脚時間 2 回分後の位置

まずはじめに遊脚化しない脚 sp1, sp2 の遊脚時間 2 回分後の位置 p'_{sp1},p'_{sp2} を求める。脚 sp1, sp2 の現在の位置をそれぞれ p_{sp1},p_{sp2} とし、原点を旋回中心 Q にとり x 軸,y 軸を胴体重心座標系と平行にとった旋回中心座標系を考える。旋回中心座標系からみた p_{sp1},p_{sp2} の偏角 ϕ_{sp1}, ϕ_{sp2} は、

$$\phi_{sp1} = atan2(p_{sp1y}-p_{Qy}, p_{sp1x}-p_{Qx}) \tag{4.4}$$
$$\phi_{sp2} = atan2(p_{sp2y}-p_{Qy}, p_{sp2x}-p_{Qx}) \tag{4.5}$$

と表すことができる。ここで atan2(A,B) とは、計算機で $tan^{-1}\frac{A}{B}$ を計算するときに使用すべき関数で、A,B の正負により $-\pi$ から π の値をとる。旋回角速度を $\dot{\theta}$ として遊脚時間 2 回分後すなわち 2Tsw 秒後の偏角を求めると

$$\phi'_{sp1} = \phi_{sp1}-\dot{\theta}\cdot 2T_{sw} \tag{4.6}$$
$$\phi'_{sp2} = \phi_{sp2}-\dot{\theta}\cdot 2T_{sw} \tag{4.7}$$

となる。旋回中心からの距離、すなわち旋回半径を r_{sp1}, r_{sp2} とすれば遊脚時間 2 回分後の支持脚の位置を胴体重心座標系で次式のように得ることができる。

$$p'_{sp1} = \begin{bmatrix} p_{Qx}+r_{sp1}\cos\phi'_{sp1} \\ p_{Qy}+r_{sp1}\sin\phi'_{sp1} \\ p_{Qz} \end{bmatrix} \tag{4.8}$$
$$p'_{sp2} = \begin{bmatrix} p_{Qx}+r_{sp2}\cos\phi'_{sp2} \\ p_{Qy}+r_{sp2}\sin\phi'_{sp2} \\ p_{Qz} \end{bmatrix} \tag{4.9}$$

4 本の接線

次に p'_{sp1},p'_{sp2} からそれぞれ胴体重心（すなわち座標系の原点）を中心とし、最低安定余裕 L_{lsm} を半径とした円に引いた 4 本の接線を求める。p'_{sp1} から引いた接

線と最低安定余裕の円との 2 つの接点をそれぞれ p_{t1},p_{t2}、p'_{sp2} から引いた接線と最低安定余裕の円との 2 つの接点をそれぞれ p_{t3},p_{t4} とおくと、これらの接点は次のように求めることができる。

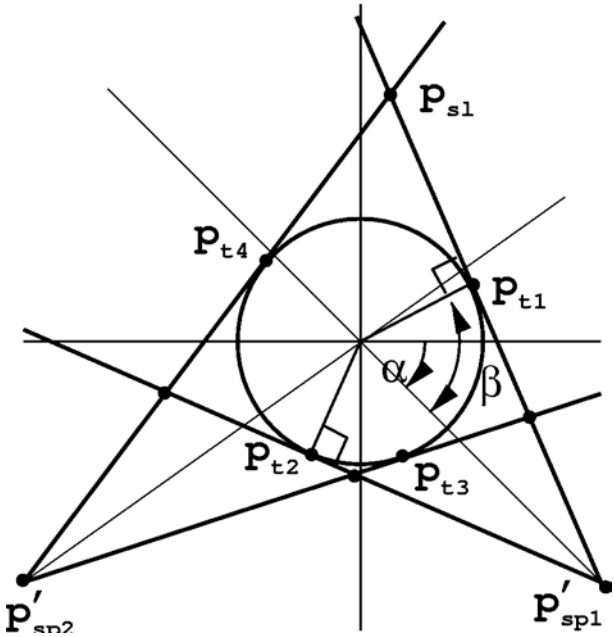


図 4.3: 最低安定余裕を半径とした円とその接線

まず原点と p'_{sp1},p_{t1} が作る三角形について考えると図 4.3 より

$$\alpha_1 = atan2(p'_{sp1y}, p'_{sp1x}) \tag{4.10}$$

$$\beta_1 = cos^{-1}\frac{L_{lsm}}{||p'_{sp1}||} \tag{4.11}$$

であるから接点 p_{t1} は

$$p_{t1} = \begin{bmatrix} L_{lsm}\cos(\alpha_1+\beta_1) \\ L_{lsm}\sin(\alpha_1+\beta_1) \\ p_{sp1z} \end{bmatrix} \tag{4.12}$$

のように求めることができる。次に原点と p'_{sp1}, p_{t2} が作る三角形について考えると, $\triangle op'_{sp1}p_{t1} \equiv \triangle op'_{sp1}p_{t2}$ であり、$\angle p'_{sp1}op_{t1} = \angle p'_{sp1}op_{t2}$ であるから p_{t2} は

$$p_{t2} = \begin{bmatrix} L_{lsm}\cos(\alpha_1-\beta_1) \\ L_{lsm}\sin(\alpha_1-\beta_1) \\ p_{sp1z} \end{bmatrix} \tag{4.13}$$

のように求めることができる。同様に、p_{t3}, p_{t4} も、

$$\alpha_2 = atan2(p'_{sp2y}, p'_{sp2x}) \tag{4.14}$$

$$\beta_2 = cos^{-1}\frac{L_{lsm}}{||p'_{sp2}||} \tag{4.15}$$

より、

$$p_{t3} = \begin{bmatrix} L_{lsm}\cos(\alpha_2+\beta_2) \\ L_{lsm}\sin(\alpha_2+\beta_2) \\ p_{sp2z} \end{bmatrix} \tag{4.16}$$

$$p_{t4} = \begin{bmatrix} L_{lsm}\cos(\alpha_2-\beta_2) \\ L_{lsm}\sin(\alpha_2-\beta_2) \\ p_{sp2z} \end{bmatrix} \tag{4.17}$$

のように求めることができる。

したがって、4つの接線はこれらの点を用いて

$$p'_{sp1}+m_1(p_{t1}-p'_{sp1}) \tag{4.18}$$
$$p'_{sp1}+m_2(p_{t2}-p'_{sp1}) \tag{4.19}$$
$$p'_{sp2}+m_3(p_{t3}-p'_{sp2}) \tag{4.20}$$
$$p'_{sp2}+m_4(p_{t4}-p'_{sp2}) \tag{4.21}$$

のように表すことができる。ここで m_1, m_2, m_3, m_4 は任意のスカラー量である。

接線同士の交点

p'_{sp1} ,p'_{sp2} を除いた接線同士の交点は4つ得られるのでこれらの交点をそれぞれ $p_{cr1}, p_{cr2}, p_{cr3}, p_{cr4}$ とおくと、これらの交点は以下の組合わせの接線同士の連立方程式

$$\begin{cases} p_{cr1} = p'_{sp1}+m_1(p_{t1}-p'_{sp1}) \\ p_{cr1} = p'_{sp2}+m_3(p_{t3}-p'_{sp2}) \end{cases} \tag{4.22}$$

$$\begin{cases} p_{cr2} = p'_{sp1}+m_1(p_{t1}-p'_{sp1}) \\ p_{cr2} = p'_{sp2}+m_4(p_{t4}-p'_{sp2}) \end{cases} \tag{4.23}$$

$$\begin{cases} p_{cr3} = p'_{sp1}+m_2(p_{t2}-p'_{sp1}) \\ p_{cr3} = p'_{sp2}+m_3(p_{t3}-p'_{sp2}) \end{cases} \tag{4.24}$$

$$\begin{cases} p_{cr4} = p'_{sp1}+m_2(p_{t2}-p'_{sp1}) \\ p_{cr4} = p'_{sp2}+m_4(p_{t4}-p'_{sp2}) \end{cases} \tag{4.25}$$

をそれぞれ解くことにより得ることができる。

こうして得られた $p_{cr1}, p_{cr2}, p_{cr3}, p_{cr4}$ の4点からベクトル $-p'_{sp1}$ とベクトル $-p'_{sp2}$ に挟まれた領域にある点を選び、この点を p_{sl} とする。p_{sl} は常に1つしか得られない。

積集合となる領域

先ほど得られた p_{sl} に向かって p'_{sp1} ,p'_{sp2} からそれぞれ直線を引くと、2 直線に挟まれ、かつ、p'_{sp1} ,p'_{sp2} からみて p_{sl} より向こう側にある領域が「2 歩目の遊脚運動中に最低安定余裕を保証する 1 歩目の着地すべき領域」となる。この領域と脚可動範囲の領域の積集合となる領域が着地点探索を行う領域である。もし積集合となる探索領域が存在しなければ、着地すべき領域が存在しないことになるため、その遊脚順序による着地点探索は失敗とする。

着地点探索区間の決定

ここまでで得られた探索領域に、さらに探索を旋回中心を中心とした円弧軌道上で行うという制限を加え、探索区間を 1 次元化して単純化してこの探索領域を通過する円弧軌道上で着地点探索を行う。このとき、探索は円軌道の半径を脚基準位置を通る円軌道の半径にとる。これは、遊脚が着地した後の支持脚軌道が可動領域の中心付近を通過すると、支持脚軌道の左右に余裕ができて進行方向や旋回中心が変化したときに対処しやすいからである。

もし始めに設定した旋回半径の円弧軌道が探索領域を通過しない場合、または探索区間内で有効な着地点が見つからなかった場合は、まず、支持脚軌道が胴体重心から見て外側に広がるよう少しづつ半径を更新して新しい探索区間を得る。円弧軌道が探索領域を通過する限界まで半径を変更しても探索に成功しない場合は、逆に円弧軌道が胴体重心に向かって近づくよう円弧軌道の半径を変更する。それでも探索に成功しない場合は、有効な着地点が存在しないものと判断し、この遊脚順序における着地点探索は失敗とする。

評価関数を最大にする点の探索

探索区間内で評価関数を最大化するために本研究では 2 分探索を使用する。この探索によって得られた点を、i1, i2 を遊脚順序とした場合の 1 歩目の着地点とし、また、このときの評価関数の値を遊脚順序を i1, i2 とした場合の評価関数の値とする。

以上のような着地点探索を全ての実行可能な i1, i2 の遊脚順序に対して行い、得られた各遊脚順序の評価関数の値を比較して評価関数の値が最大となるような遊脚順序を選択して、これを実行する。そしてこの遊脚運動を実行する際には、着地点探索によって得られたその遊脚順序で評価関数を最大にするような着地点に 1 歩目の遊脚を着地させる。1 歩の遊脚運動を終えるとまた全ての実行可能な遊脚順序に対して探索を行い、これを繰り返すことで歩行する。

4.5 特別ルールの適用

デッドロックに陥る危険性がある場合や基本探索による通常の探索では実行可能な全ての遊脚順序で探索が失敗してしまう場合、状況に応じて特別ルールを追加して探索をやり直す。

4.5.1 デッドロックに結び付く脚を含んだ遊脚順序の優先

運動余裕が遊脚時間1回分を下回っていて次に遊脚化しなければ脚可動範囲外に出てしまう危険性があるにも関わらず、その脚を遊脚化すると安定余裕の最低値を守ることができない脚が存在する場合、この脚は機動性を損なう原因となるのに遊脚化できないため、デッドロックの引金になる危険性がある。

このような脚が存在する場合、この問題となっている脚を2歩目の遊脚に含む遊脚順序の評価関数に対してバイアス値を与え、デッドロックに結び付く危険性のある脚を優先的に遊脚化するという方法をとる。1歩目の遊脚の着地点は2歩目の遊脚運動を保証するような点であるため、現在直ちに遊脚化できなくとも次には遊脚化できるようになる。

4.5.2 歩行速度調整

歩行速度が大きいと支持脚が可動範囲外に出てしまう危険性がある。1歩目の遊脚となる候補の脚以外で運動余裕が遊脚時間1回分を下回っている脚が存在する場合、一時的に歩行速度と旋回角速度を落として脚が可動範囲外に出ることを防ぐ。

4.5.3 揺動運動の付加

全ての遊脚順序に対して探索が失敗であった場合、いったん歩行速度と旋回角速度を0にして4脚支持のまま胴体に揺動運動を付加する。

揺動の方向と振り幅は以下のように決定する。

1. まず運動余裕が遊脚時間1回分(Tsw秒)を下回っていて、かつ、その脚を遊脚化すると安定余裕の最低値を下回ってしまう脚を求める。
2. 次に、胴体重心から先ほどの脚を遊脚化したときにできる支持脚三角形の重心に向かうベクトルを求める。
3. これを先ほど求めた脚から最も近い可動限界の辺に射影したベクトルを求めて、これを胴体が平行移動する方向とする。
4. 胴体の移動量は最低安定余裕を保てる程度とする。

以上のような特別ルールを付加して再度基本探索を行い、評価関数が最大となるような遊脚順序を実行する。

4.6 まとめ

本章では全方位歩行のもう 1 つのアプローチである自由歩容生成について述べた。本章では遊脚／支持脚の脚先端軌道計画については述べなかったが、歩容計画は遊脚化のタイミングとその着地点を計画するものであるから、歩行パターンを用いた場合でも自由歩容を用いた場合でも同じものを使用して脚先端の軌道計画を行えば良いということを付け加えておく。

第5章　歩行制御系の構築

5.1　はじめに

本章では 前章までで述べた歩容計画系を用いて全方位歩行を実現するための歩行制御系を構築する。本論文では２つの歩容計画手法を提案しているが、歩行制御系全体から見ると歩容計画系の設計が異なるだけで、他の部分は全く同じ設計で良い。

5.2　歩行制御系の概要

歩行制御系は大きく分けると、

- 歩容計画系
- 脚先端軌道計画系
- 軌道追従制御系

に分けることができる。さらに、実際にロボットを動かすには

- 操縦入力インターフェース
- 関節角度出力インターフェース

といった、計算機に対する入出力のインターフェースも必要であるが、入出力のインターフェース設計については本論文では取り扱わず、本節では歩容計画系、脚先端軌道計画系、軌道追従制御系について述べる。これらの３つは制御階層を形成し、処理は歩容計画、脚先端軌道計画、軌道追従制御の順に実行される。

5.3　クロール歩容と循環歩容による歩容計画系

歩容計画系は、操縦入力から指令された歩行速度と旋回角速度を必要に応じて調整し、旋回中心の位置を求め、最終的には遊脚運動を開始するタイミング、遊脚の着地点、その他脚運動を行うのに必要な諸々のパラメータを算出する。

クロール歩容と循環歩容による歩容計画系では、遊脚順序やその着地点等は旋回中心の位置から選ばれた基準歩容に従う。図 5.1 にクロール歩容と循環歩容による歩容計画系の大まかな流れを示す。

操縦入力があった場合、旋回中心の位置や旋回半径を更新して次の歩容で使用するパラメータ等を算出する。その後、現在の歩容と次の歩容から歩容遷移パターンを選択して歩容遷移を開始する。ただし、歩容遷移中や操縦入力が無い場合は旋回中心の更新を行わない。

5.4　自由歩容計画系

遊脚順序と着地点探索による自由歩容計画系では、遊脚順序も着地点も探索によって求める。この自由歩容計画系の大まかな流れを図 5.2 に示す。

2 歩 1 組の全ての遊脚順序に対して歩行速度の調整と基本着地点探索を行う。旋回中心位置のや旋回半径等の更新は、基本着地点探索の中で行う。全ての遊脚順序に対して探索が失敗した場合、特別ルールを付加して再び探索を行う。探索に成功している場合は、成功した遊脚順序のなかから評価関数の値が最大となるような遊脚順序を選び、これを実行する。

5.5　脚先端軌道計画系

次に、歩容計画に沿って歩行するために必要な遊脚、支持脚の先端軌道を計画する。遊脚軌道は歩容計画によって得られた開始位置 p_{si} と終了位置 p_{ei} とを結んだ曲線となれば良い。本研究では遊脚軌道に cos 関数を用いて次式のように指令位置 p_{di} と指令速度 $\dot{p}_{di}$ を与えるものとする。

$$p_{di} = \begin{bmatrix} \frac{p_{six}-p_{eix}}{2}\cos(\frac{\pi}{T_{sw}}t_i) \\ \frac{p_{siy}-p_{eiy}}{2}\cos(\frac{\pi}{T_{sw}}t_i) \\ \frac{h_{sw}}{2}(1-\cos(\frac{2\pi}{T_{sw}}t_i) \end{bmatrix} + \frac{1}{2}(p_{si}+p_{ei}) \tag{5.1}$$

$$\dot{p}_{di} = \begin{bmatrix} -\frac{1}{2}(p_{six}-p_{eix})\frac{\pi}{T_{sw}}\sin(\frac{\pi}{T_{sw}}t_i) \\ -\frac{1}{2}(p_{siy}-p_{eiy})\frac{\pi}{T_{sw}}\sin(\frac{\pi}{T_{sw}}t_i) \\ -h_{sw}\frac{\pi}{T_{sw}}\sin(\frac{2\pi}{T_{sw}}t_i) \end{bmatrix} \tag{5.2}$$

支持脚先端の指令位置と指令速度はそれぞれ指令位置の偏角 ϕ_{di} を用いて次式のように与える。

$$\phi_{di} = \phi_{ei} - \dot{\theta}(t_i - T_{sw}) \tag{5.3}$$

$$p_{di} = \begin{bmatrix} p_{qx} + r_i \cos\phi_{di} \\ p_{qy} + r_i \sin\phi_{di} \\ -p_{ciz} \end{bmatrix} \tag{5.4}$$

$$\begin{aligned} \dot{p}_{di} &= \begin{bmatrix} -r_i \sin(\phi_{di})\dot{\phi}_{di} \\ r_i \cos(\phi_{di})\dot{\phi}_{di} \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} r_i \sin(\phi_{di})\dot{\theta} \\ -r_i \cos(\phi_{di})\dot{\theta} \\ 0 \end{bmatrix} \end{aligned} \tag{5.5}$$

5.6 脚先端軌道追従制御系

脚先端軌道計画により得られた軌道を分解速度法をもちいて軌道追従することで歩行に必要な脚先の位置と脚の姿勢を生成する。分解速度法のブロック線図を図 5.3 に示す。

これを式で表すと、次式のようになる。

$$\dot{p}(t+dt) = \dot{p}_d(t+dt) + K_p(p_d(t+dt) - p(t)) \tag{5.6}$$

$$\dot{\theta}(t+dt) = J^{-1}(\theta(t))\dot{p}(t+dt) \tag{5.7}$$

$$\theta(t+dt) = \theta(t) + \dot{\theta}(t+dt)dt \tag{5.8}$$

$$p(t+dt) = f(\theta(t+dt)) \tag{5.9}$$

式 (5.6) は位置のフィードバックによる誤差の修正を意味する。式 (5.7) は速度レベルの逆運動学である。逆運動学そのものを解くのは難しく、解も複数存在する可能性があるが、速度レベルの逆運動学ならばヤコビ行列が正則である限り簡単に解ける。式 (5.8) はオイラー積分である。本来、分解速度法は式 (5.7) によって得られた角速度を発生するようなトルクをロボットに入力して、実際にロボットが運動した結果として関節角度を得るのであるが、本歩行制御プログラムでは実機に対して関節角度を指令するのみのオープンループ制御で、関節角度のフィードバックが無いため、このオイラー積分で関節角度を得ることにした。このようにして得られた関節角度を式 (5.9) の順運動学式に代入すると、脚先端位置を得ることができ、これをまた式 (5.6) にフィードバックする。このようにして分解速度制御をもちいて手先（脚先）ベースの軌道追従を行う。

5.7 実時間制御用歩行制御系の構築

本章で構築した歩行制御系に入出力のインターフェースを付加し、実時間制御が行えるようにした実時間制御用歩行制御系のの処理の流れを図 5.4 に示す。

5.8　まとめ

本章では歩行制御系を歩容計画系、脚先端軌道計画系、軌道追従制御系の３つに分けてそれぞれの設計について述べた。本論文では２つの歩容計画の手法を提案しているが、どちらの歩容計画手法を用いても、歩行計画系が変わるだけで他の部分は全く同じものを使って歩行制御を行うことが可能である。

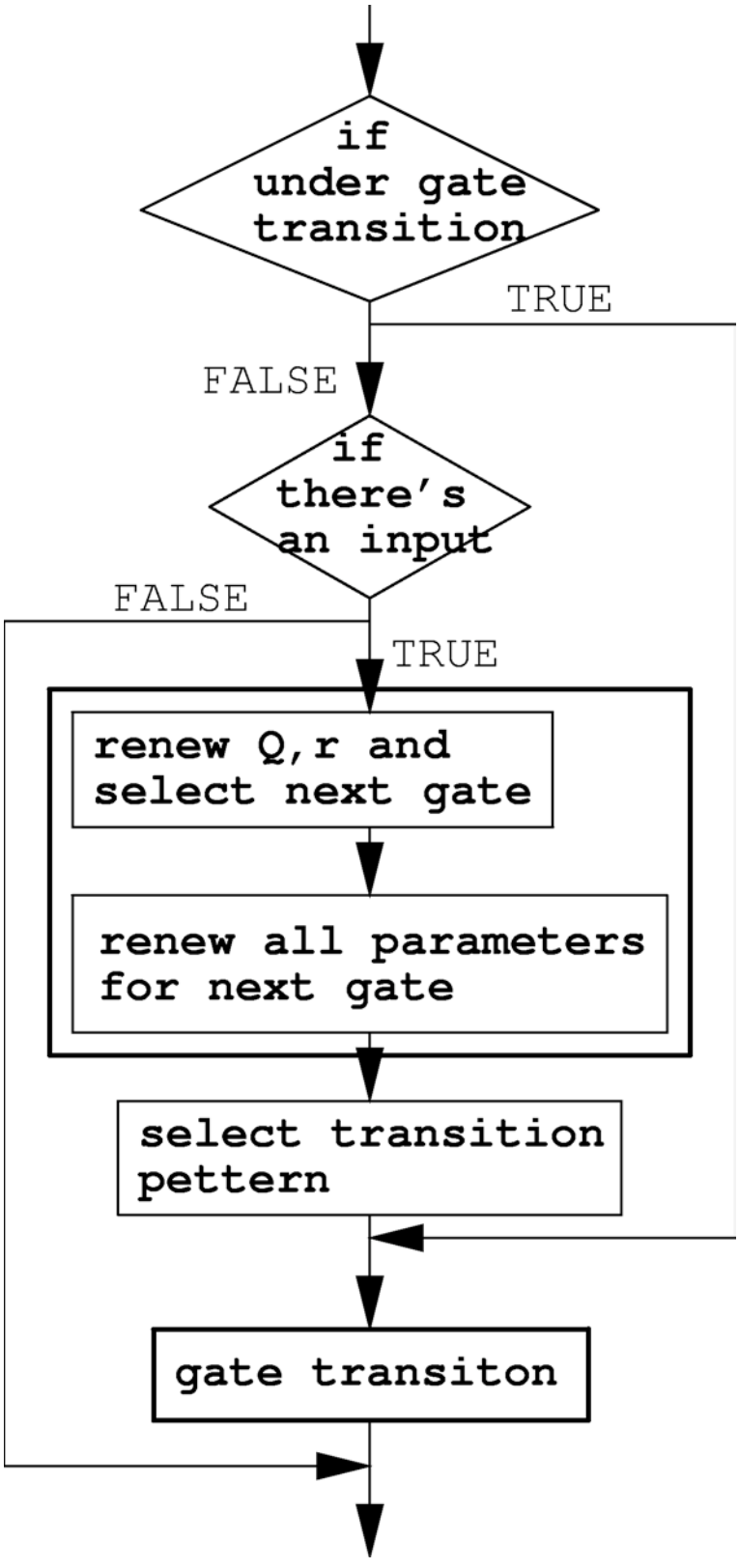


図 5.1: 歩行パターンによる歩容計画の流れ

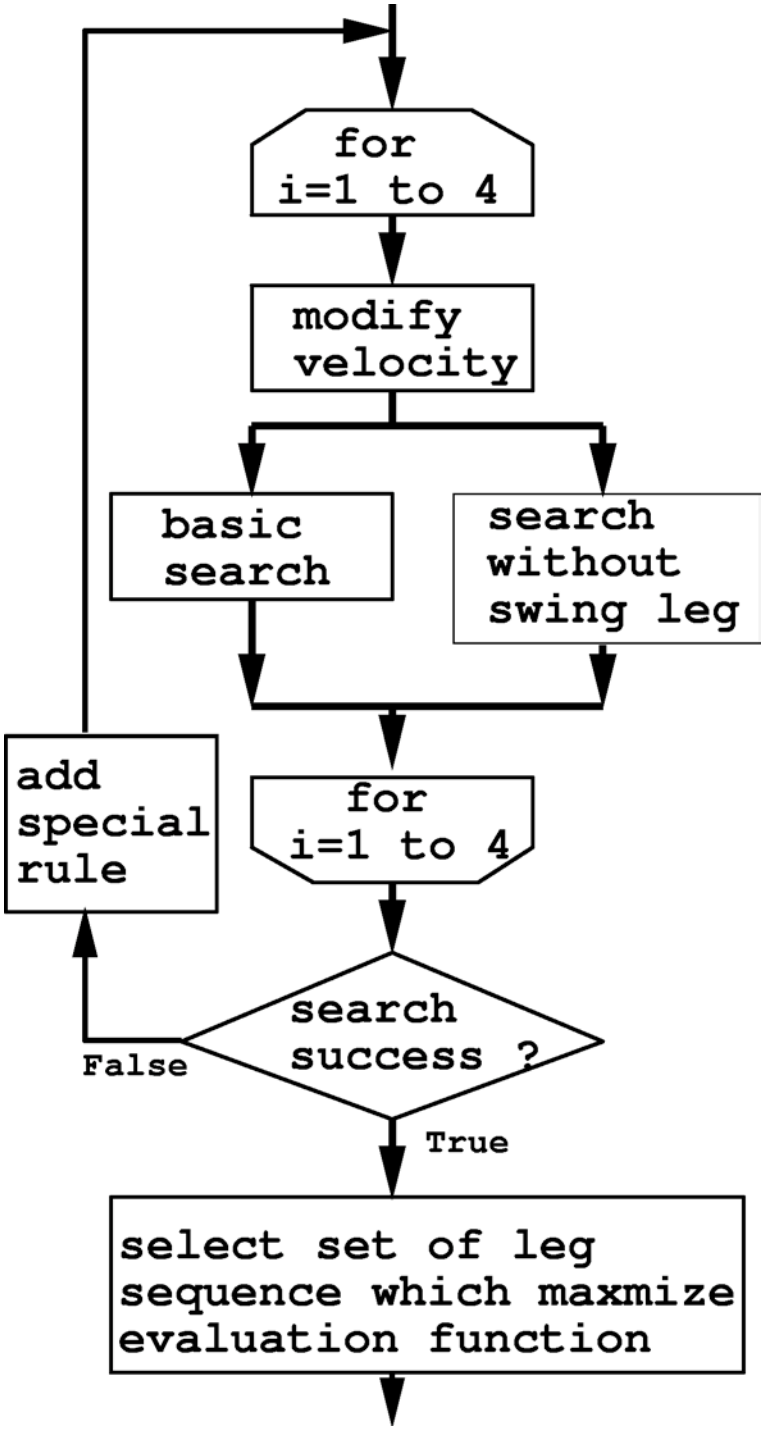


図 5.2: 自由歩容計画系の流れ

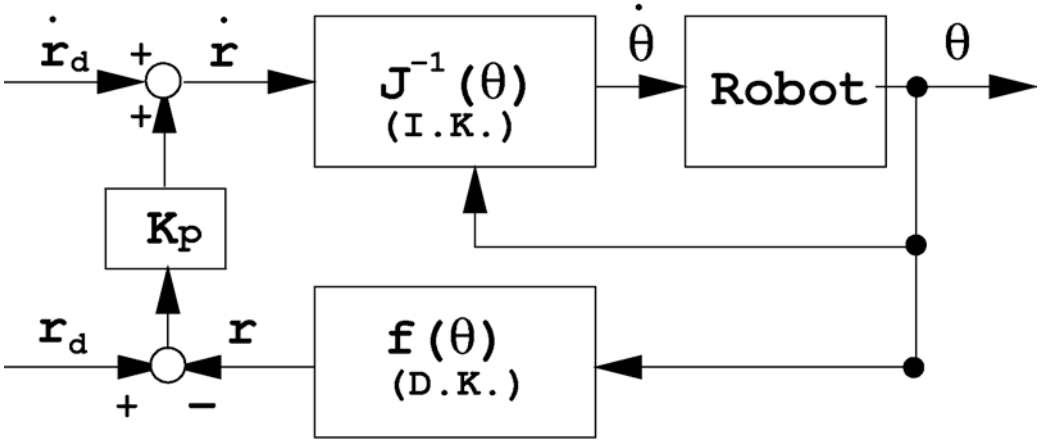


図 5.3: 分解速度制御法のブロック線図

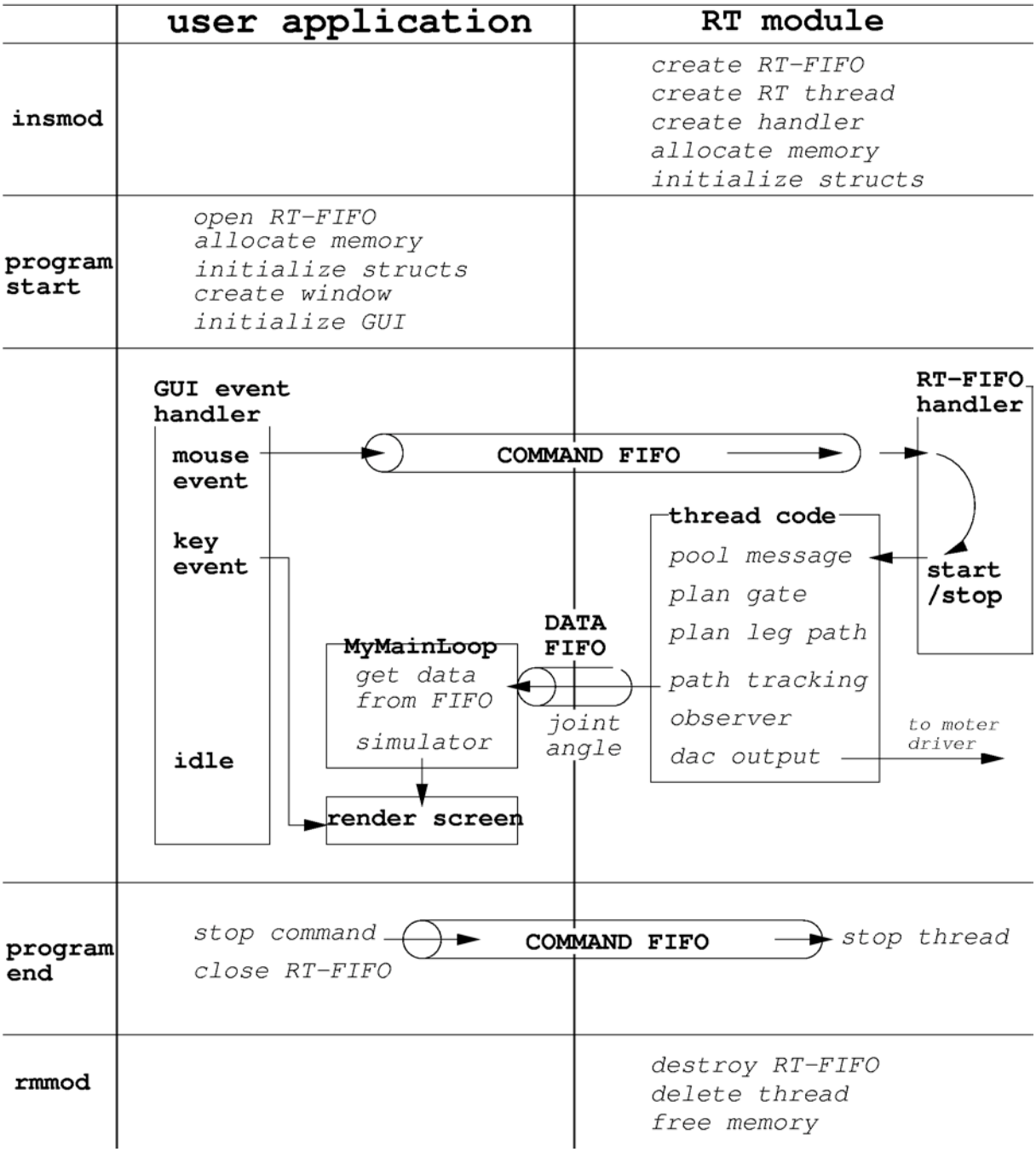


図 5.4: 実時間制御の流れ

第6章　歩行実験

6.1　はじめに

本章では実機および計算機シミュレーションによる歩行実験を用いて本論文で提案した２つの歩容計画手法について検討する。

6.2　実験機材

実機モデルには普及型四脚歩行ロボット TITAN-VIII[32] を使用する。TITAN-VIII は文部省科学研究補助金重点領域研究「知能ロボット」の一環として汎用、ローコストな標準的な脚型ロボットの研究用プラットフォームとして開発されたものである。

表 6.1: TITAN-VIII 仕様概要

寸法	0.40 × 0.60 × 0.25 [m]
脚数	4
自由度	12 (3 自由度 × 4 脚)
本体重量	約 20 kg
過搬重量	約 7 kg
アクチュエータ	DC サーボモータ (50W) × 12
モータドライバ	Titech Robot Driver Ver.2 × 12
センサ	各関節にポテンショメータ × 12
電源	DC 24V

脚機構は、リンクとワイヤを組み合わせた平面 2 自由度機構と、それ全体を回転させる機構によって構成されており、合計 3 自由度を有する。この機構を駆動するワイヤは 螺旋状に V 字溝を切ったプーリにより巻き取られ、さらにプーリは DC モータによって回される。

モータドライバには岡崎産業製 Titech Robot Driver Ver.2 (PC-0121-2) を使用する。このモータドライバの 位置制御モードを使用することにより、各関節に取りつけたポテンショメータからのフィードバックを用いて指令電圧に応じた関節角度を得ることができる。

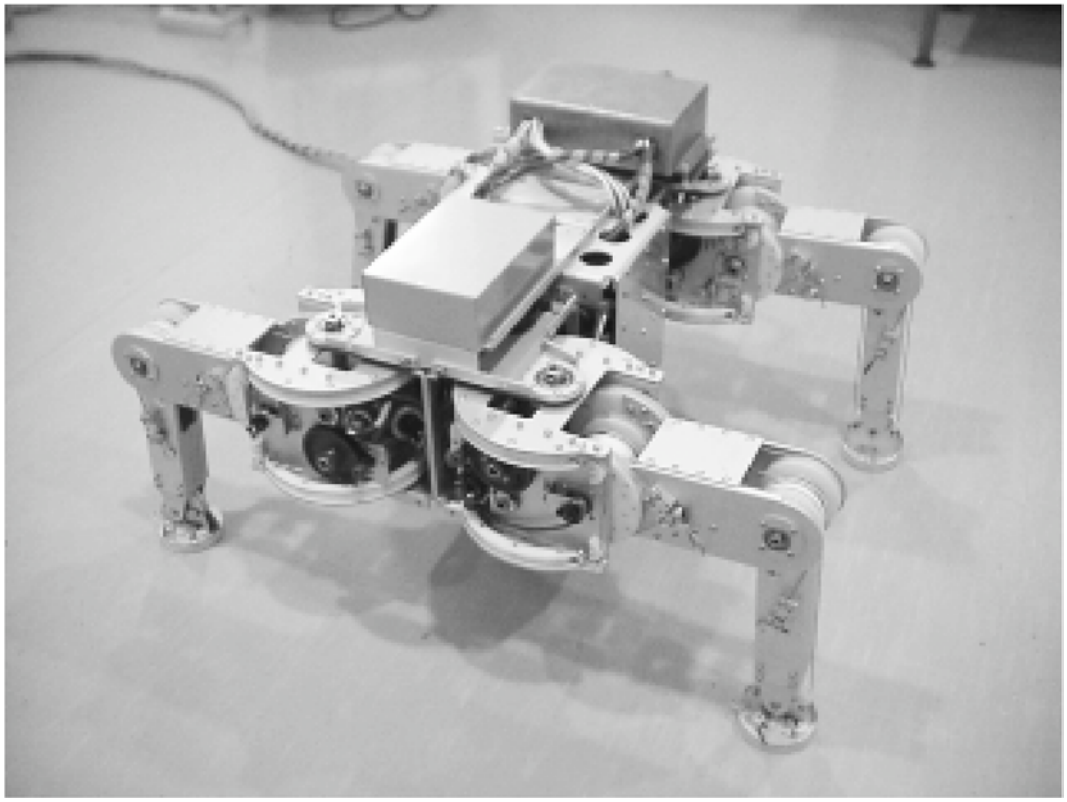


図 6.1: TITAN-VIII 実機モデル

関節角度を指令する指令電圧は PC の ISA バスに接続した DA コンバータから出力する。

歩容および脚先端軌道の計画は OS に RTLinux を搭載した PC/AT 互換機 (Intel MMX Pentiun 200MHz, RAM 64MB) から行い、脚先端の軌道追従には分解速度法を用いる。また、操縦には OpenGL を用いて作成した GUI 的な操縦パネルを使用し、マウスで進行方向ボタンや旋回ボタンをクリックしてロボットを操縦する。

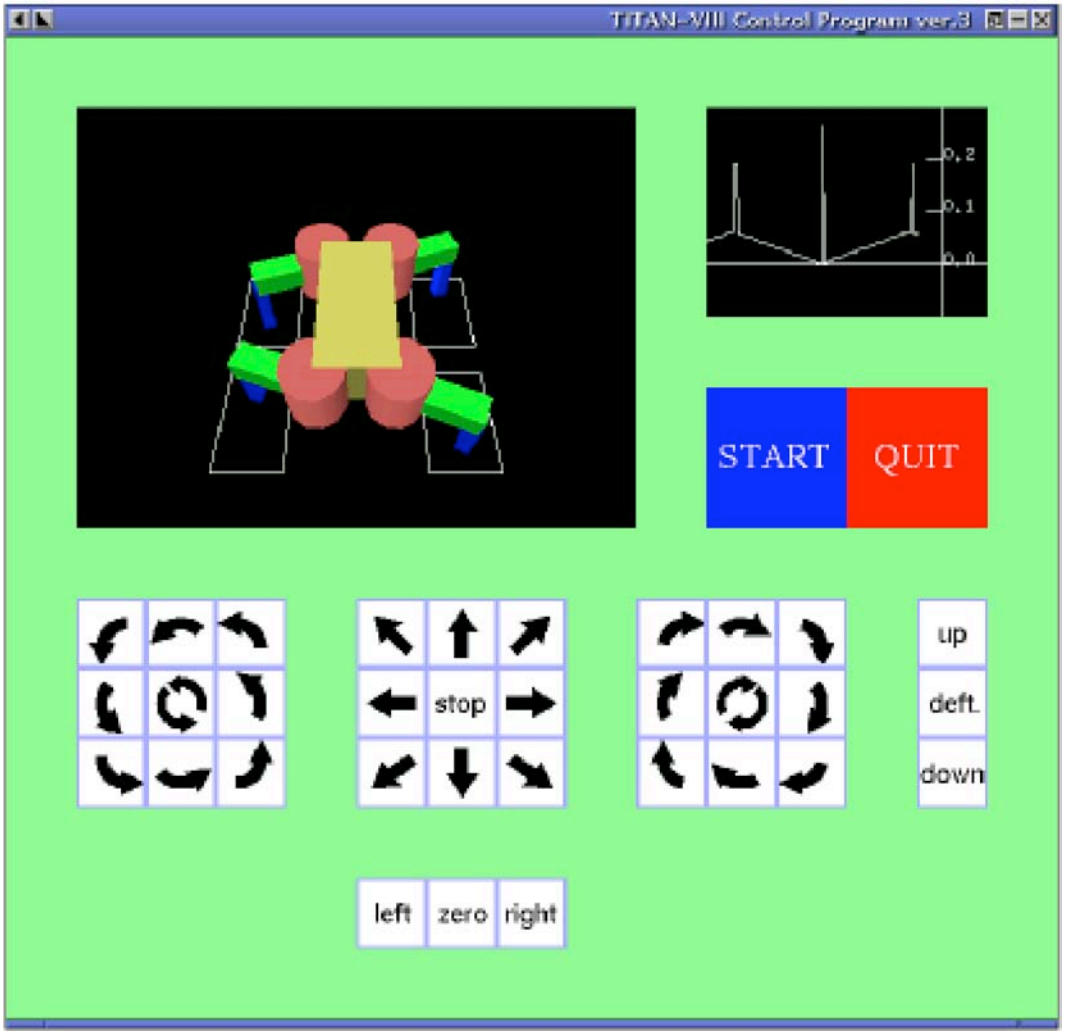


図 6.2: OpenGL による GUI 操縦パネル画面

6.3　クロール歩容と循環歩容に基づく全方位歩行の実機実験

6.3.1　実験項目

クロール歩容と循環歩容に基づく歩容計画による全方位歩行の歩行性能を検証するために以下のような歩行実験を行った。

- 基準歩行 (前後左右への並進、左右への旋回、左右のその場旋回、斜め方向への並進、ロボットの前方を中心とした旋回)
- 歩容遷移のパターン比較
  - クロール → クロール
  - クロール → 循環
  - 循環 → クロール
  - 循環 → 循環
- 総合（8 の字歩行）

6.3.2　結果と考察

基準歩容による歩行

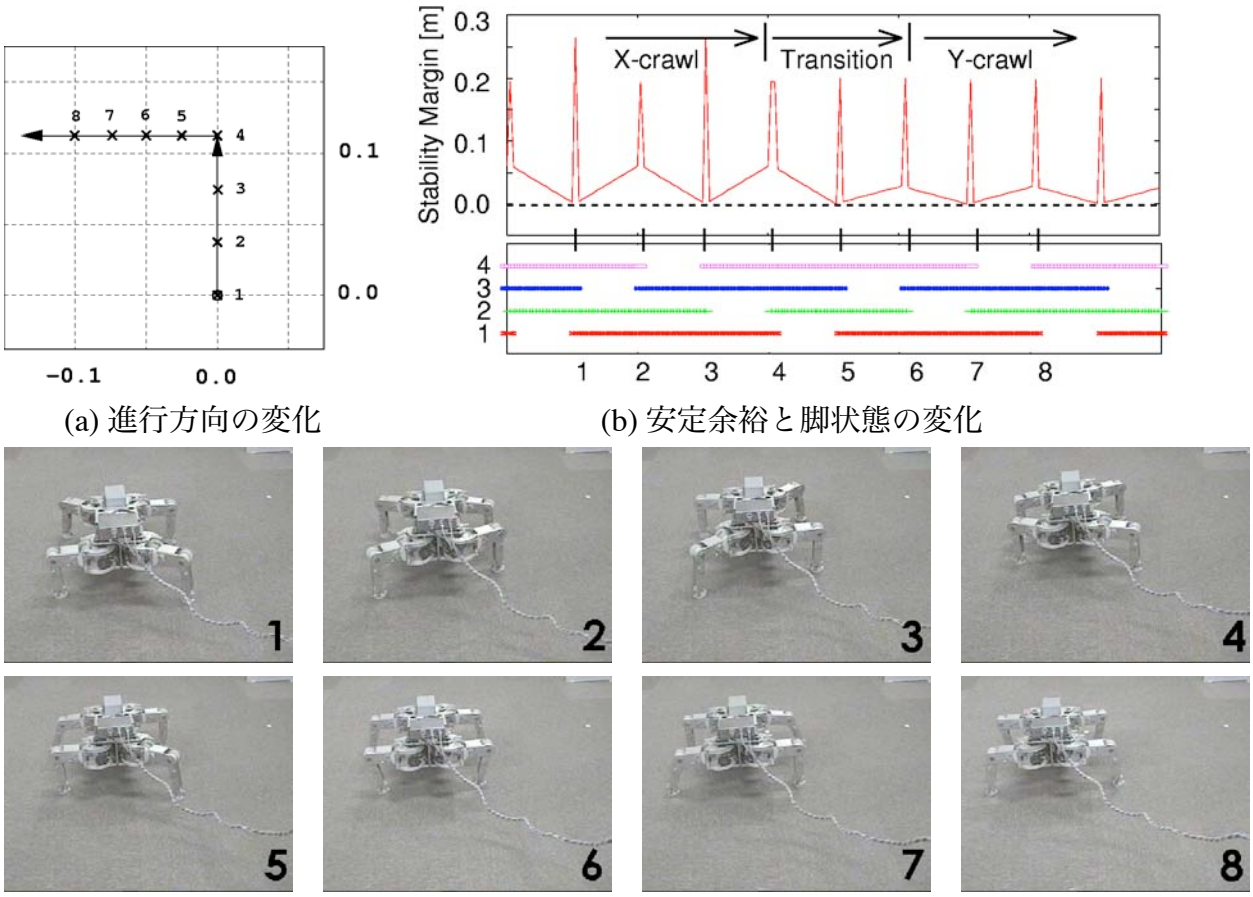
基準歩容による歩行の実験では、通常よくつかわれる進行方向に対して定常的に安定した歩行が可能であることを確認した。

しかし、旋回中心を胴体前方にとった旋回では転倒はしないものの歩行中多少の胴体の傾きがみられた。この原因は、前脚と後脚の基準位置の距離が短く、脚可動範囲も広くとらなかったために、常に重心が支持脚三角形の辺の付近にさられることになり、歩行中に安定余裕が大きく回復できなかったのではないかと考えている。

歩行遷移パターンの比較

歩容遷移パターンの比較は、それぞれの歩容遷移パターンが実際にどのように振る舞うかを確認し、比較するために実験を行った。比較した各歩容遷移パターンの (a) 進行方向の変化、(b) シミュレーションによる安定余裕と遊脚状態の変化、および (c) 実機実験の連続写真を図 6.3 から図 6.8 に示す。

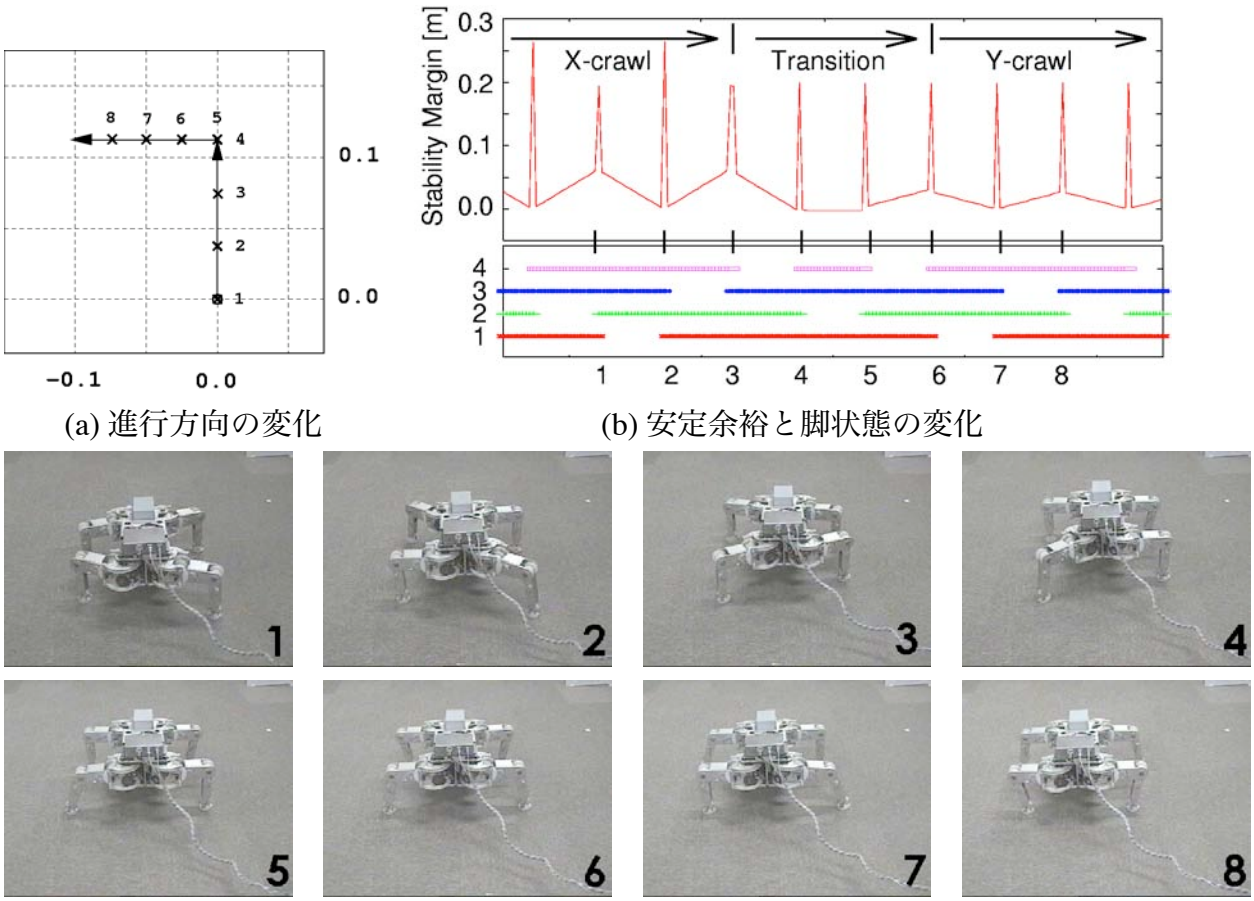
全体的には理論通りの安定した歩行が実現できた。しかし、クロール → クロールのケース 2 とクロール → 循環のケース 2 の両パターンとも、歩容遷移 2 歩目で胴体が傾いてしまう傾向が見られた。この原因は、両パターンとも歩容遷移の 2 歩



(a) 進行方向の変化 (b) 安定余裕と脚状態の変化

(c) 実機による歩容遷移の連続写真

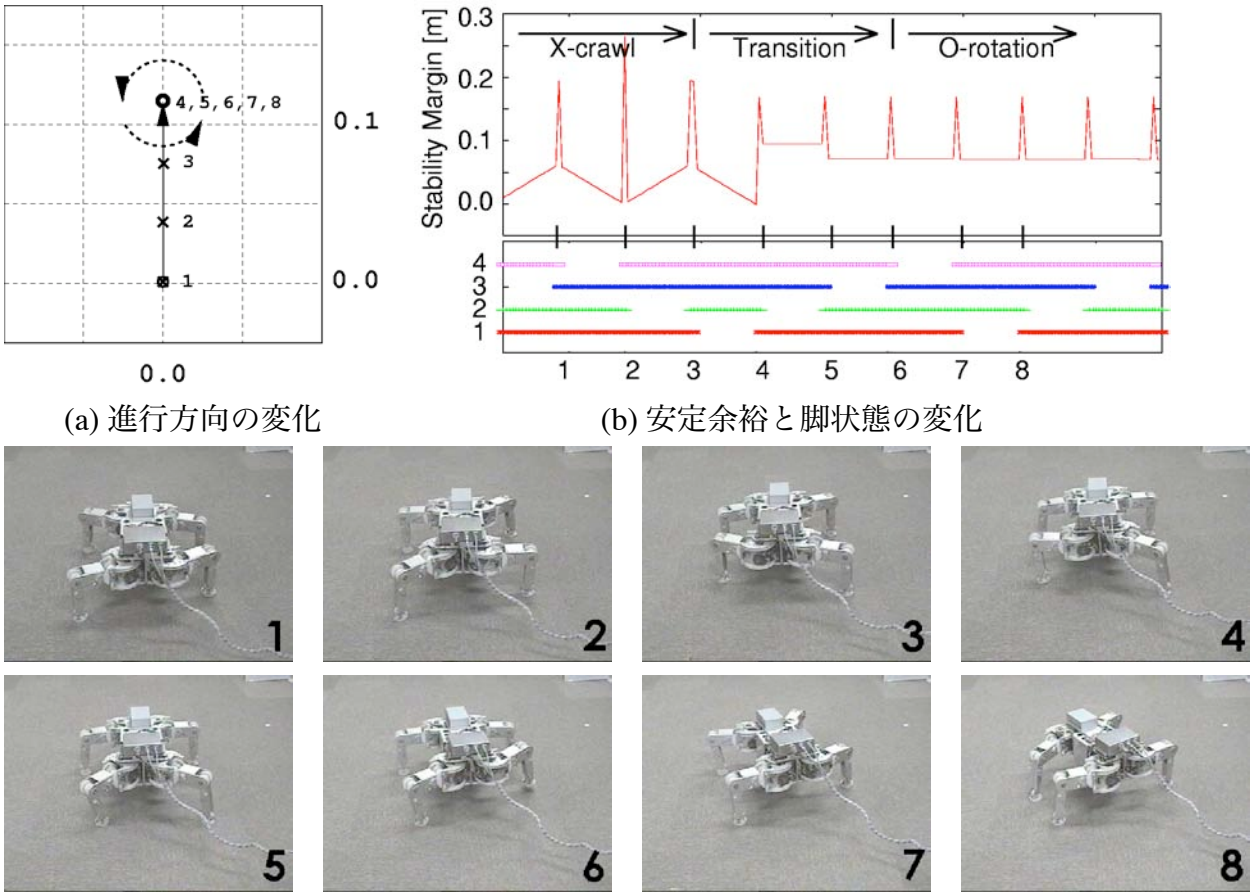
図 6.3: クロール歩容からクロール歩容への遷移 case 1



(a) 進行方向の変化　　(b) 安定余裕と脚状態の変化

(c) 実機による歩容遷移の連続写真

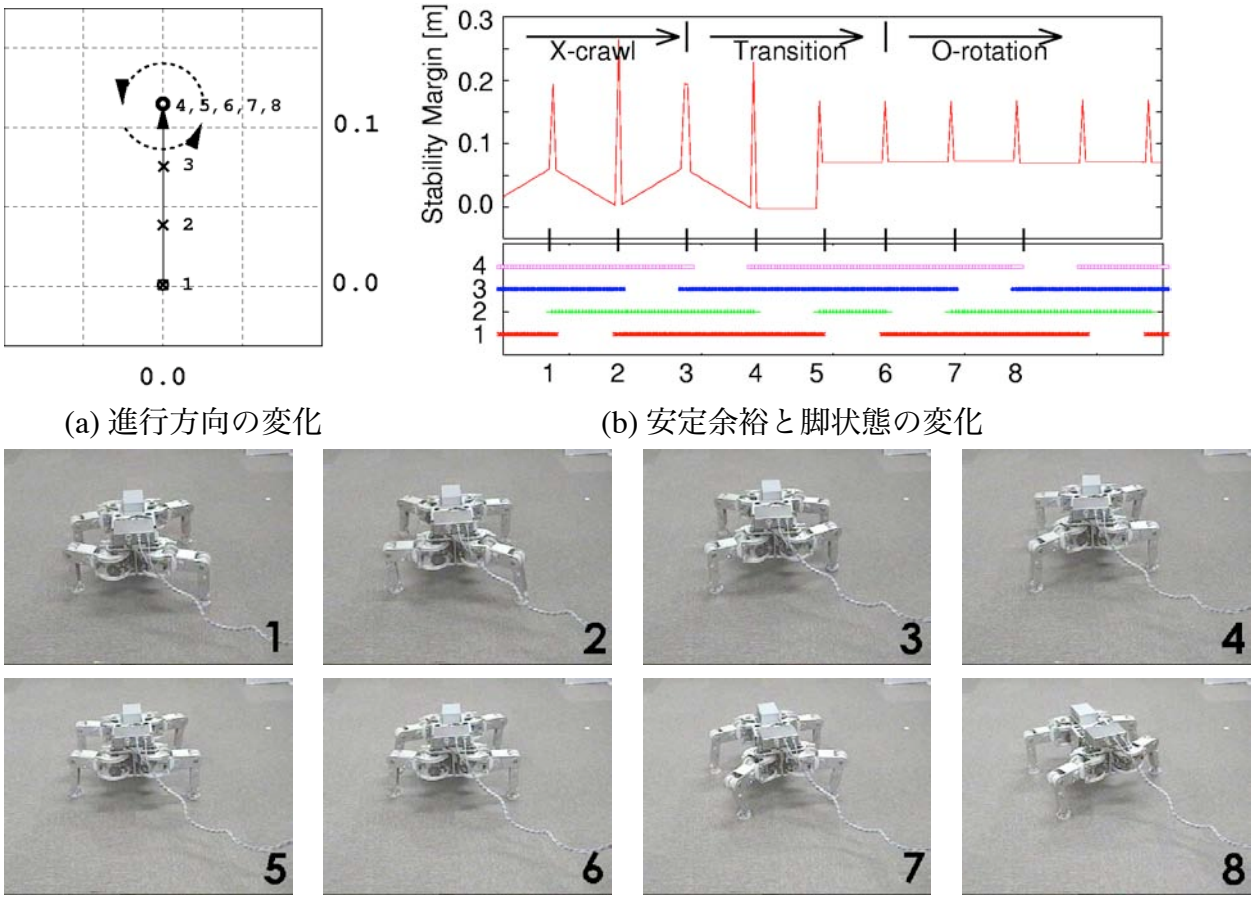
図 6.4: クロール歩容からクロール歩容への遷移 case 2



(a) 進行方向の変化 (b) 安定余裕と脚状態の変化

(c) 実機による歩容遷移の連続写真

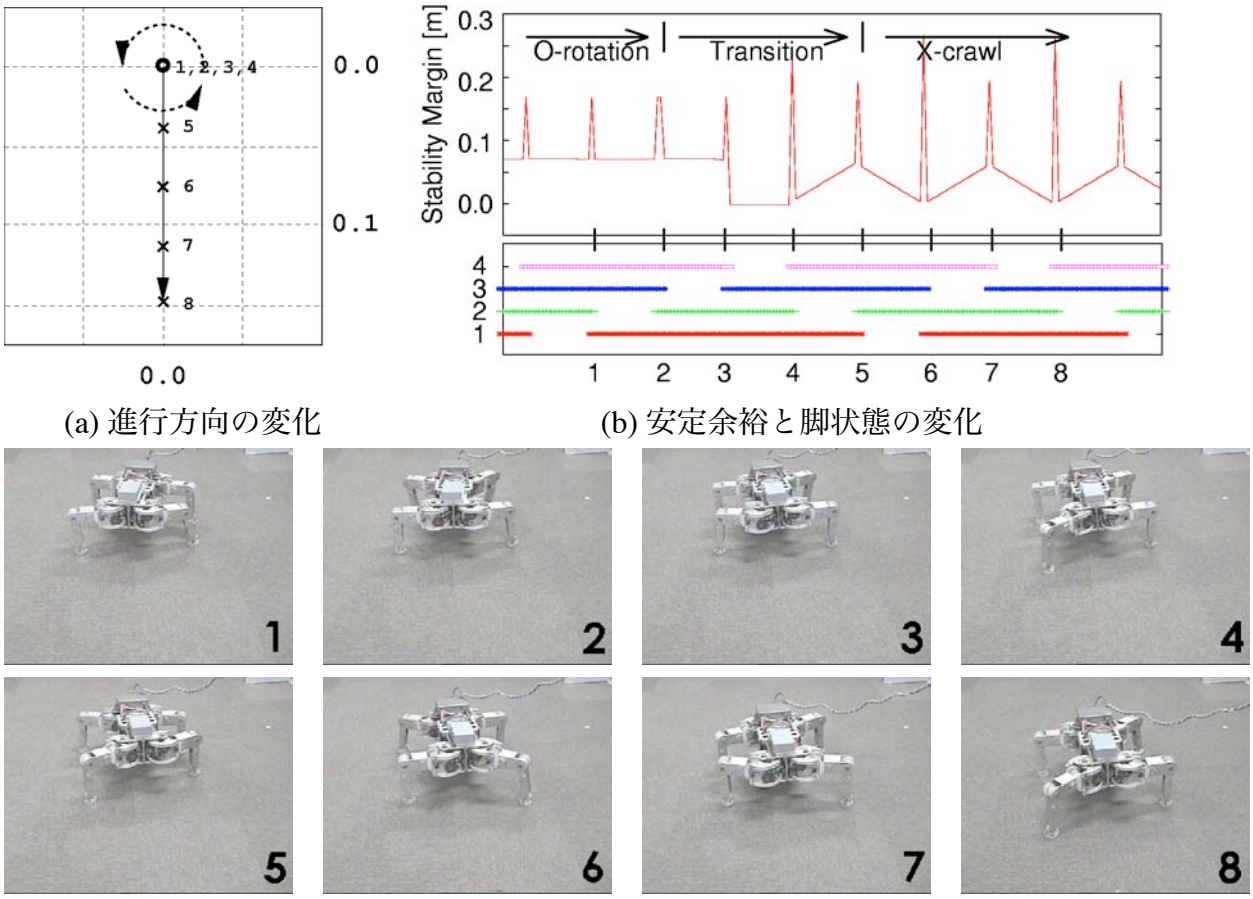
図 6.5: クロール歩容から循環歩容への遷移 case 1



(a) 進行方向の変化　　(b) 安定余裕と脚状態の変化

(c) 実機による歩容遷移の連続写真

図 6.6: クロール歩容から循環歩容への遷移 case 2

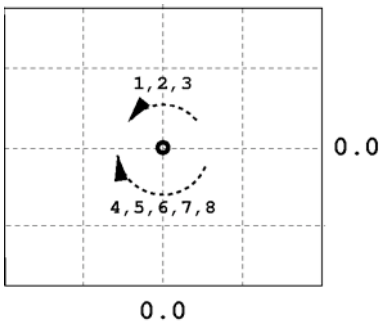


(a) 進行方向の変化

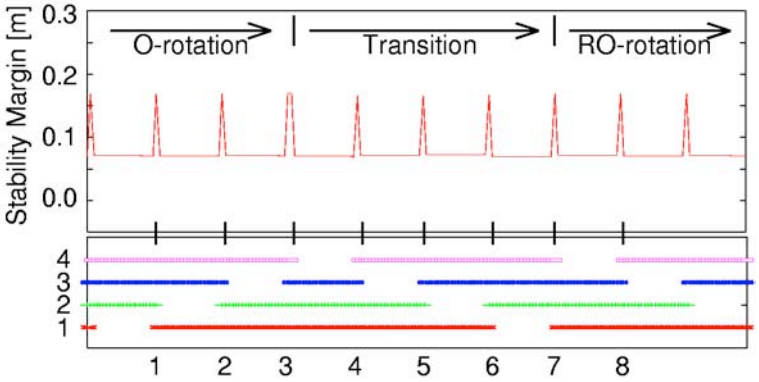
(b) 安定余裕と脚状態の変化

(c) 実機による歩容遷移の連続写真

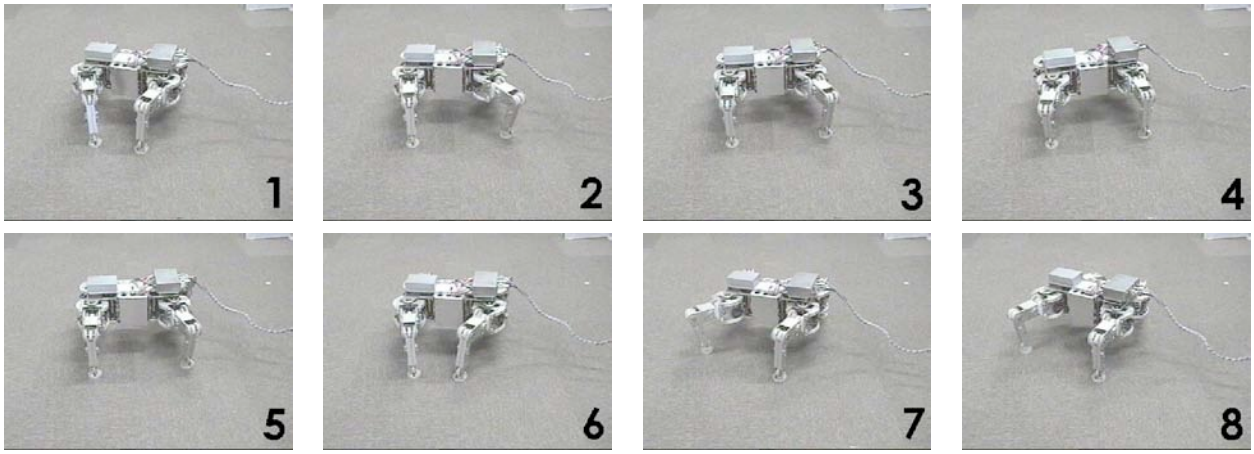
図 6.7: 循環歩容からクロール歩容への遷移



(a) 進行方向の変化



(b) 安定余裕と脚状態の変化



(c) 実機による歩容遷移の連続写真

図 6.8: 循環歩容から循環歩容への遷移

目で重心が支持脚対角線上で停止するため、理論上のモデルと実機モデルでの重心位置の誤差が胴体を傾けていることにあるのではないかと考えている。

総合

総合的な歩行の実験として8の字歩行を行った。これまでの実験は各歩行パターンそれぞれを確認するものであったのに対し、今度の実験は四足移動ロボットが実用上必要とするであろう軌道を歩行する実験である。その例として並進のみによる8の字軌道と常に進行方向に頭を向けた8の字軌道を歩行し、操縦者の指令に応じて8の字軌道を歩行できることを確認した。

6.4 自由歩容による全方位歩行のシミュレーション実験

6.4.1 実験項目

遊脚順序と着地点探索に基づく歩容計画による全方位歩行の歩行性能を検証するために、以下のような実験を行った。

- 定常的な並進および旋回の基本歩行実験
- 安定重み係数と歩行速度の比較実験
- 最低安定余裕と歩行速度の比較実験

6.4.2 結果と考察

定常的な並進および旋回の基本歩行実験

本手法を用いた自由歩容による基本歩行性能を確認するために、安定重み係数を $K = 0.5$、最低安定余裕を $L_{lsm} = 0.0$[m] に固定して定常的な並進および旋回を行った。並進歩行の各歩行速度における状態を表6.2に、旋回歩行の各旋回中心位置および各旋回角速度における歩行状態を表6.3に示す。

並進歩行においては、低・中速度の歩行は十分可能だがある程度歩行速度が上がってくると歩行ができなくなってくることがわかった。この歩行ができない状態というのは、踏み替え回数や胴体の揺動運動ばかり行ってほとんど胴体が進行方向に移動しない状態である。また、左右や斜め方向への並進にみられる変則クロールというのは一時停止と踏み替えを含むクロール歩容に似た遊脚順序のパターンである。

なお、実験条件として脚可動範囲を $0.30 \times 0.2[m]$ の長方形にとったため、遊脚時間 Tsw を 1[s] に固定すると、歩行パターンを用いて作った一般的な $\beta = 0.75$ のクロール歩容の最高速度は $v = 1.0[m/s]$ である。また、歩行パターンに基づく全方位

表 6.2: 並進歩行性能

歩行速度 [m/s]	前進後退	左右	斜め
0.03	クロールに収束	変則クロールに収束	変則クロールに収束
0.05	クロールに収束	変則クロールに収束	変則クロールに収束
0.07	クロールに収束	変則クロールに収束	ほとんど移動できない
0.09	ほとんど移動できない	ほとんど移動できない	ほとんど移動できない

表 6.3: 旋回歩行性能

旋回角速度	旋回中心位置	歩行状態
0.21	(0.0, 0.0)	4 脚支持相をもつ循環歩容に収束
0.26	(0.0, 0.0)	循環歩容に収束
0.42	(0.0, 0.0)	間欠的に回転する循環歩容に収束
0.2	(0.0, 0.15)	循環歩容の遊脚順序に収束
0.2	(0.15, 0.0)	変則的な循環歩容に似た遊脚順序に収束
0.04	(0.0, 1.0)	踏み替えを含んだ変則クロールに収束
0.04	(0.0, 1.5)	周期的に停止して踏み替える変則クロールに収束

歩行では脚可動範囲の 3/4 しか利用していないため最高速度も 3/4 の $v = 0.75[m/s]$ である。

旋回歩行においては、胴体重心の近傍を中心とした旋回ではほぼ循環歩容に近いパターンに収束し、旋回中心が胴体重心からある程度離れると並進に近いパターンに収束することがわかった。

しかし、これらの歩行性能は必ずしも任意の 4 脚支持姿勢から歩行を開始して得られるという訳ではなく、歩行開始時の姿勢によっては数歩移動するとその場で踏み替えを繰り返す無限ループに入ってしまうこともある。このような初期姿勢の最も顕著なものは各関節角度を 0 にとった基準姿勢で、この姿勢の時はどの脚も遊脚化できないという特殊な状態になっている。基準姿勢からの歩行は可能な場合とそうでない場合があるが、本実験では同じ初期姿勢をとらせるために、敢えて基準姿勢をはずしたところから歩行を開始することにした。

安定重み係数による歩行状態の変化

安定重み係数 K を変化させたときの歩行状態の変化を調べるために、直進と胴体重心回りの旋回についてそれぞれ歩行速度、旋回角速度を固定して歩行した。直進歩行において安定重み係数を変化させた場合の歩行状態を表 6.4 に、胴体重心回りの旋回において安定重み係数を変化させた場合の歩行状態を表 6.5 に示す。

表 6.4: 安定重み係数による直進歩行状態の変化

安定重み係数	歩行速度	歩行状態
0.00	0.05	4 脚支持相を持つクロールに収束
0.25	0.05	4 脚支持相を持つクロールに収束
0.50	0.05	4 脚支持相を持つクロールに収束
0.75	0.05	踏み替えのみで移動しないパターンに収束
1.00	0.05	踏み替えのみで移動しないパターンに収束

表 6.5: 安定重み係数による旋回歩行状態の変化

安定重み係数	旋回角速度	歩行状態
0.00	0.21	4 脚支持相をもつ循環歩容に収束
0.25	0.21	4 脚支持相をもつ循環歩容に収束
0.50	0.21	4 脚支持相をもつ循環歩容に収束
0.75	0.21	停止して踏み替えを挟む循環歩容に収束
1.00	0.21	停止して踏み替えを挟む循環歩容に収束

直進、旋回のどちらの場合も安定重み係数が 0.5 以下ではクロール歩容や循環歩容といったパターンに収束しているが、0.75 以上では踏み替えが増えて移動効率の悪いパターンに収束している。この実験結果から、安定重み係数が 0.5 以下では評価関数の値が機動性の影響が大きく、着地点探索では運動余裕が最大となるような着地点を選択していることがわかる。逆に、安定重み係数が 0.5 より大きい場合は評価関数の値が安定性の影響が大きく、着地点探索では安定余裕が最大となるような着地点を選択していることがわかる。

最低安定余裕による歩行状態の変化

最低安定余裕 L_{lsm} を変化させたときの歩行状態の変化を調べるために、前方への直進と胴体重心回りの旋回についてそれぞれ歩行速度や安定重み係数を変化させて歩行した。直進した場合の歩行状態を表 6.6 に、胴体重心回りの旋回をした場合の歩行状態を表 6.7 に示す。

直進歩行では、最低安定余裕が大きくなるほど、歩行速度が大きくなるほど歩行が困難になっていることがわかった。また、最低安定余裕の値を 0.04[m] にとると歩行速度や安定重み係数によらず歩行ができなった。ここで「歩行不可」と表示されている歩行状態は、踏み替えの繰り返や胴体の揺動運動の繰り返しといった胴体を進行方向にほとんど推進しない状態にを指している。

胴体重心回りの旋回では、直進に比べて最低安定余裕の値を大きくとった歩行が可能であることがわかった。最低安定余裕の値を大きくとりすぎると胴体が揺

表 6.6: 最低安定余裕と直進歩行状態の変化

最低安定余裕	歩行速度	安定重み係数	歩行状態
0.01	0.03	0.00	歩行不可
0.01	0.03	0.25	4 脚支持相を持つクロール歩容に収束
0.01	0.03	0.50	4 脚支持相を持つクロール歩容に収束
0.01	0.03	0.75	4 脚支持相を持つクロール歩容に収束
0.01	0.03	1.00	停止と踏み替えが多くなる
0.01	0.05	0.00	歩行不可
0.01	0.05	0.25	4 脚支持相を持つクロール歩容に収束
0.01	0.05	0.50	歩行不可
0.01	0.05	0.75	歩行不可
0.01	0.05	1.00	歩行不可
0.02	0.03	0.00	変則クロールに収束
0.02	0.03	0.25	4 脚支持相を持つクロール歩容に収束
0.02	0.03	0.50	4 脚支持相を持つクロール歩容に収束
0.02	0.03	0.75	4 脚支持相を持つクロール歩容に収束
0.02	0.03	1.00	歩行不可
0.02	0.05		歩行不可
0.03	0.03	0.00	変則クロールに収束
0.03	0.03	0.25	4 脚支持相を持つクロール歩容に収束
0.03	0.03	0.50	4 脚支持相を持つクロール歩容に収束
0.03	0.03	0.75	歩行不可
0.03	0.03	1.00	歩行不可
0.03	0.05		歩行不可

表 6.7: 最低安定余裕と旋回歩行状態の変化

最低安定余裕	旋回角速度	安定重み係数	歩行状態
0.01	0.21	0.00	4 脚支持相を持つ循環歩容に収束
0.01	0.21	0.25	4 脚支持相を持つ循環歩容に収束
0.01	0.21	0.50	4 脚支持相を持つ循環歩容に収束
0.01	0.21	0.75	変則的な循環歩容に収束
0.01	0.21	1.00	変則的な循環歩容に収束
0.01	0.26	0.00	循環歩容に収束
0.01	0.26	0.25	循環歩容に収束
0.01	0.26	0.50	循環歩容に収束
0.01	0.26	0.75	変則的な循環歩容に収束
0.01	0.26	1.00	変則的な循環歩容に収束
0.06	0.21	0.00	4 脚支持相を持つ循環歩容に収束
0.06	0.21	0.25	4 脚支持相を持つ循環歩容に収束
0.06	0.21	0.50	4 脚支持相を持つ循環歩容に収束
0.06	0.21	0.75	変則的な循環歩容に収束
0.06	0.21	1.00	変則的な循環歩容に収束
0.06	0.26	0.00	循環歩容に収束
0.06	0.26	0.25	循環歩容に収束
0.06	0.26	0.50	循環歩容に収束
0.06	0.26	0.75	変則的な循環歩容に収束
0.06	0.26	1.00	変則的な循環歩容に収束
0.07	0.21		歩行不可
0.07	0.26	0.00	循環歩容に収束
0.07	0.26	0.25	循環歩容に収束
0.07	0.26	0.50	循環歩容に収束
0.07	0.26	0.75	歩行不可
0.07	0.26	1.00	歩行不可

動運動を繰り返すのみで歩行ができなくなってしまうが、歩行状態の収束の仕方は旋回角速度にほとんど関係がないこともわかった。

考察

実験により、本手法による自由歩容計画を用いた全方位歩行では

1. 前進後退、左右および斜め方向へ並進、旋回といった定常的な歩行が可能
2. 安定重み係数は着地点探索に影響を与える
3. 最低安定余裕を大きくとると並進歩行速度の上限が下がる

といった結果を得た。

1. に関しては、全方位歩行に必要なあらゆる定常的な歩行を実現することができたわけだが、先に述べたように任意の姿勢から歩行を開始して収束できるとまでは行かなかった。これは歩行中の任意の姿勢から別の方向に進行方向を変えた場合に歩行が収束しない可能性があることを意味するため、全方位歩行を実現する上で大きな障害となる危険性がある。

任意の姿勢から歩行を開始した場合、効率的な歩行を続けられなくなってしまう確率は、胴体重心近傍回りの旋回歩行より並進歩行の方が遥かに高い。これは胴体重心近傍まわりの旋回ならば 1 歩の遊脚運動で十分に安定性が回復できるのに並進に近くなると 1 歩の遊脚運動では必ずしも安定性が回復できるわけではないからである。したがって、並進運動において安定性を十分に回復させるためには 2 歩の遊脚運動が必要となるのだが、1 歩目の着地点探索を行った上でどのようにして 2 歩目の着地点を決めるか等が解決しなかったため、本手法では探索の深さを 2 歩としながらも 1 歩しか遊脚運動を実行しなかった。このような理由から、遊脚運動は安定性を十分回復させるために 2 歩 1 組で実行すべきであると言える。

2. に関しては、安定重み係数が評価関数を支配してしまい、その結果、係数が 0.5 以下では運動余裕が最大となるような着地点を選び、係数が 0.5 より大きければ安定余裕が最大となるような着地点を選ぶというような着地点探索となってしまった。特に 1 歩目の着地点はすでに 2 歩目の遊脚運動の安定性を保証するような領域内で探索を行うため、安定重み係数を 0 にして運動余裕のみで評価を行っても安定性に問題はない。しかし、表 6.6 からわかるように、最低安定余裕をある程度持たせた場合のみ、評価関数中の安定余裕が良い方に働く場合もあるので安定性が意味をなさないわけではない。

3. に関しては最低安定余裕を大きくとると並進歩行速度の上限が下がるが、最低安定余裕の大きさと旋回角速度の大きさにはほとんど関係が見られなかった。これは、並進では胴体が移動すると同時に安定余裕が小さくなるが、旋回中心が胴体重心の近傍にあると旋回運動をしても安定余裕がほとんど変化しないため、旋回角速度と安定性には関係がないというのが原因であると考えている。

6.5 まとめ

本章では、歩行パターンに基づく全方位歩行と自由歩容による全方位歩行の実機実験および計算機シミュレーション実験により、各歩容計画手法の有効性と内在する問題点を検証した。

歩行パターンに基づく歩容計画による全方位歩行では、前進後退から 8 の字歩行まであらゆる歩行を連続的に実現し、本手法が十分有効であることを検証した。しかし、実機実験ではモデル化誤差により重心の位置が正確に算出できないため、理論上安定余裕が 0 でも胴体が傾いてしまうことも判明した。

また、探索を用いた自由歩容計画にほる全方位歩行では進行方向を一定にとった歩行では一定のパターンに収束することを確認した。しかし、任意の姿勢からでは歩行を続けられるまでに至らなかったため、歩行中の方向転換を含めた連続的な歩行は実現できなかった。

第7章　結論

7.1　本論文のまとめ

本論文では、4脚移動ロボットの歩容生成について議論し、静的安定な全方位歩行を実現するための歩容計画手法としてクロール歩容と循環歩容に基づく歩容計画手法と、遊脚順序と着地点探索に基づく自由歩容計画手法の二つの歩容計画手法を提案した。

一つめのクロール歩容と循環歩容に基づく歩容計画手法は、歩行速度と旋回角速度から算出される旋回中心の位置から使用する歩容を選択し、旋回中心が変わるときは各歩容間の遷移を連続的に行う歩容計画手法である。実機を用いた歩行実験ではこの歩容計画手法により、十分な移動性能を持つ連続的な全方位歩行を実現した。

二つ目の遊脚順序と着地点探索に基づく自由歩容計画手法は、安定性と機動性を評価基準を用いて16通りの遊脚順序に対して評価を行い、評価関数が最も大きくなるような遊脚順序と着地点を選択するという歩容計画手法である。計算機シミュレーションを用いた歩行実験では、この歩容計画手法により進行方向を一定にとった歩行であれば一定のパターンに収束して行くことを確認した。

本研究で提案した全方位歩行の歩容計画手法は、前者の手法は主に凹凸の少ない比較的なだらかな地面で、後者の手法は主に凹凸の激しい不整地で応用されることを考えた手法である。前者の手法では良好な歩行性能を得ることができたが、後者の手法では水平な整地においても連続的な全方位歩行を実現するに十分な歩行性能を得るまでには至らなかった。脚移動ロボットの本来の利点を活かすためには、不整地での応用が期待できる後者の自由歩容による歩行性能の向上が必要である。

7.2　今後の課題

静歩行パターンに基づく歩容計画ではデューティ比を0.75として安定余裕は0までを認めたため、理論上は安定でも実機では胴体が傾いてしまうとうことが判明している。安定性を向上するためには、デュテーィ比を上げるなど、より安定性の高い歩行パターンを使用する方法、重心位置のモデル化誤差を解消するためにロボットの脚先に力センサを配置して脚先にかかっている床反力から重心の位置を正確に算出する方法などが考えられる。また、今後はさらに歩行速度を向上

させるために動的な全方位歩行との連続性を検討するといった課題がある。

一方、遊脚順序と着地点探索に基づく自由歩容計画では、本研究では十分な全方位歩行性能を得られなかったため、まず任意の姿勢から歩行を続けられるようにすることが当面の課題である。しかし、本自由歩容計画手法で採用した 2 歩 1 組で計画して 1 歩のみを実行する方法は、1 歩のみでは並進時の安定性を十分に回復できないという問題点があるため、2 歩目の着地点をどのように決定するか考慮した上で 2 歩 1 組で計画したものを 2 歩まで実行するという改善策が考えられる。また、歩行を評価関数を用いた非線型最適化問題として捉えるのであれば、遊脚運動を開始するタイミングも評価によって得られるべきであろう。今後、本自由歩容計画手法を凹凸のある不整地に拡張するには着地点の探索領域に着地禁止領域を設定することで対処することができるが、傾斜地に応用する場合には安定性評価基準を傾斜地向けのものに変更する必要がある。いずれにせよ自由歩容はこれから先不整地歩行を実現する際の有力な候補であるから、解決すべき課題は数多く残っている。

謝辞

本研究の遂行に当りご指導下さいました茨城大学工学部システム工学科 馬書根助教授に深く感謝の意を表します。また、本論文をまとめるに際して懇切なご指導と熱心なご討論をいただきました堤泰行教授、青島伸一助教授に深く感謝いたします。また、研究においてご意見、ご協力下さいました同学科石黒美佐子教授、浜松芳夫教授、城道介助教授、米澤保雄助教授、坪井一洋講師、新納浩幸講師、近藤久助手、岩崎唯史助手、梅津信幸助手に感謝の意を表します。

最後に、ゼミ等における熱心な討論、ご協力いただいた同研究室の渡辺充氏、林志強氏、李黎氏、李権志氏、新谷博明氏、冨田修平氏、大豆生田吉広氏に感謝いたします。

関連図書

[1] E. Muybridge : Animals in Motion, Dover Pub., 1957.

[2] P.P.Gambaryan : How Mammals Run, Jhon Wiley & Sons, 1974.

[3] R. McN. Alexander: “The Gait of Bipedal and Quadrupedal Animals”, Int. J .Robotics Research, vol.3, no.2, pp.49-59, 1984.

[4] 広瀬, 野瀬, 梅谷: “4 足歩行機械の知的歩容決定に関する研究 その 2 基準側行歩容とその決定”, 第 23 回自動制御連合講演会予稿集, pp.579-580, 1980.

[5] 広瀬, 菊池, 梅谷 : “4 足歩行機械の基準旋回歩容”, 日本ロボット学会誌, 第 2 巻, 6 号, pp.545-555, 1984.

[6] 塚越, 広瀬 : “間歇クロール歩容の提案とその生成原理”, 日本ロボット学会誌, vol.17, no.2, pp.301-309, 1999.

[7] 広瀬, 福田, 菊池: “4 足歩行機械の制御システム”, 日本ロボット学会誌, vol.3, no.4, pp.304-323, 1985.

[8] 広瀬, 横井: “4 足歩行機械の踏み変えに関する研究”, 日本ロボット学会誌, vol.4, no.4, pp.364-372, 1986.

[9] 広瀬, 横井: “4 足歩行機械の知的歩容決定に関する研究 その 9 完全全方向歩容の実現”, 第 25 回計測自動制御学会学術講演会予稿集, pp.323-324, 1986.

[10] 伊藤 (聡), 湯浅, 伊藤 (宏): “エネルギー消費量による四足歩行パターン遷移の発信器 — 力学モデル”, 計測自動制御学会論文集, vol.32, no.11, pp.1535-1543, 1996.

[11] 米田, 飯山, 広瀬: “4 足歩行機械の間欠トロット歩容 — 全方向歩行の動的制御 —”, 日本ロボット学会誌, vol.14, no.6, pp.881-886, 1996.

[12] V. Hugel and P. Blazevic : “Towards efficient implentation of quadruped gaits with duty factor of 0.75”, IEEE International Conference on Robotics & Automation, pp.2360-2365, 1999.

[13] 広瀬, 国枝: “4 足歩行機械の一般化基準脚軌道”, 計測自動制御学会論文集, vol.25, no.4, pp.455-461, 1989.

[14] 佐野, 古荘, 三浦 : “離散時間モデルを用いた歩行周期制御に基づく 4 足歩行ロボットのペース歩行”, 日本ロボット学会誌, vol.6, no.5, pp.367-378, 1991.

[15] 本多, 佐野, 川名, 坂口, 古荘 : “関節型四足歩行ロボットによるバウンス歩容の実現”, 第 11 回日本ロボット学会学術講演会予稿集, pp.637-638, 1993.

[16] M.L. Shik and G.N. Orlovsky: “**Neurophysiolosy of Locomotor Automatism**”, Physiol.Review, vol.56, p.p.465-501, 1976.

[17] D.F. Hoyt and C.R. Taylor : “Gate and the energetics of locomotion in horses”, Nature, vol.292-16, pp.239-240, 1981.

[18] 伊藤宏司, 伊藤正美: 生体とロボットにおける運動制御, 計測自動制御学会編, コロナ社,1991.

[19] Prabir K. and K. Jayarajan : “Generation of Free Gate — A Graph Search Approach”, IEEE Trans. Robotics and Automation, vol.7, no.3, 1991.

[20] D.J. Pack and H. Kang : “An Omnidirectional Gate Control using a Graph Search Method for a Quadruped Walking Robot”, IEEE International Conference on Robotics and Automation, vol.1, pp.988-993, 1995.

[21] 中村, 関, 森, 安達 : “脚型ロボットの不整地歩行のためのモンテカルロ法を用いた自由歩容生成”, ROBOMEC’99 講演会, CD-ROM, 1999.

[22] 安達, 小谷内, 新井, 本間, 篠原, 西村 : “可動限界到達時の脚踏み変えルールに基づく 4 脚ロボットの操縦型歩行制御”, 日本ロボット学会誌, vol.16, no.3, pp.329-336, 1998.

[23] 宮下, 細田, 竹内, 浅田 : “反射の組合わせによる歩行の実現”, 日本ロボット学会誌, vol.18, no.3, pp.381-386, 2000.

[24] 日本ロボット学会編 : ロボット工学ハンドブック, コロナ社, 1990.

[25] 広瀬, 岩崎, 梅谷 : “歩行機械の静的安定性評価基準”, 第 21 回自動制御連合後援会予稿集, pp.253-254, 1978.

[26] E.G. Paradopoulos and D.A. Ray : “A New Measure of Tipover Stability Margin for Mobile Manipulators”, IEEE International Conference on Robotics and Automation, pp.3111-3116, 1996.

[27] 米田, 広瀬 : “歩行機械の転倒安定性”, 日本ロボット学会誌, vol.14, no.4, pp.517-522, 1996.

[28] D.A. Messuri and C.A. Klein : “Automatic Body Regulation for Maintaing Stability of a Legged Vehicle During Rough-Terrain Locomotion”, IEEE Journal of Robotics and Automation, vol.RA-1, no.3, September, 1985.

[29] A. Ghasempoor and N. Sepehri : “A New Measure of Machine Stability for Moving Base Manipulators”, IEEE International Conference on Robotics and Automation, pp.2249-2254, 1995.

[30] “不整地における歩行機械の静的安定性評価基準”, 日本ロボット学会誌, vol.16, no.8, pp.1076-1082, 1998.

[31] 広瀬, 福田, 菊池 : “4 足歩行機械の制御システム”, 日本ロボット学会誌, vol.3, no.4, 1985.

[32] 広瀬, 有川 : “普及型歩行ロボット TITAN-VIII の開発”, 第 1 回重点領域研究「知能ロボット」シンポジウム予稿集, pp.12-15, 1996.

[33] 舟木, 羅 : LINUX リアルタイム計測／制御開発ガイドブック, 秀和システム,1999

[34] A.RUBINI 著 : LINUX デバイスドライバ, オライリー・ジャパン,1998

[35] M.Woo, J.Neider, T.Davis : OpenGL プログラミングガイド, アジソン・ウェスレイ・パブリッシャーズ・ジャパン,1997

[36] M.Kilgard : OpenGL Programming for the X Window System (日本語版), アジソン・ウェスレイ・パブリッシャーズ・ジャパン, 1997

付 録A　歩容遷移中と遷移後の遊脚順序

A.1　クロール歩容からクロール歩容への遷移

表 A.1: 遷移中と遷移後の遊脚順序 (クロール → クロール 1)

前	後	U,A,B,C	case	step1	step2	step3	step4	遷移後の遊脚順序
X	X	0,3,1,2	case1	U(0)	C(2)	—	—	3,1,0,2
		1,0,2,3	—	—	—	—	—	—
		2,3,1,0	—	—	—	—	—	—
		3,0,2,1	case1	U(3)	C(1)	—	—	0,2,3,1
	RX	0,3,1,2	case2	U(0)	C(2)	U(0)	—	1,3,2,0
		1,0,2,3	—	—	—	—	—	—
		2,3,1,0	—	—	—	—	—	—
		3,0,2,1	case2	U(3)	C(1)	U(3)	—	2,0,1,3
	Y	0,3,1,2	case1	U(0)	C(2)	—	—	1,3,0,2
		1,0,2,3	—	—	—	—	—	—
		2,3,1,0	—	—	—	—	—	—
		3,0,2,1	case2	U(3)	C(1)	—	—	0,2,1,3
	RY	0,3,1,2	case2	U(0)	C(2)	U(0)	—	3,1,2,0
		1,0,2,3	—	—	—	—	—	—
		2,3,1,0	—	—	—	—	—	—
		3,0,2,1	case1	U(3)	C(1)	—	—	2,0,3,1
RX	X	0,1,3,2	—	—	—	—	—	—
		1,2,0,3	case2	U(1)	C(3)	U(1)	—	0,2,3,1
		2,1,3,0	case2	U(2)	C(0)	U(2)	—	3,1,0,2
		3,2,0,1	—	—	—	—	—	—
	RX	0,1,3,2	—	—	—	—	—	—
		1,2,0,3	case1	U(1)	C(3)	—	—	0,2,1,3
		2,1,3,0	case1	U(2)	C(0)	—	—	1,3,2,0
		3,2,0,1	—	—	—	—	—	—
	Y	0,1,3,2	—	—	—	—	—	—
		1,2,0,3	case1	U(1)	C(3)	—	—	0,2,1,3
		2,1,3,0	case2	U(2)	C(0)	U(2)	—	1,3,0,2
		3,2,0,1	—	—	—	—	—	—
	RY	0,1,3,2	—	—	—	—	—	—
		1,2,0,3	case2	U(1)	C(3)	U(1)	—	2,0,3,1
		2,1,3,0	case1	U(2)	C(0)	—	—	3,1,2,0
		3,2,0,1	—	—	—	—	—	—

表 A.2: 遷移中と遷移後の遊脚順序 (クロール → クロール 2)

前	後	U,A,B,C	case	step1	step2	step3	step4	遷移後の遊脚順序
Y	X	0,1,3,2	case1	U(0)	C(2)	—	—	3,1,0,2
		1,0,2,3	case2	U(1)	C(3)	U(1)	—	0,2,3,1
		2,1,3,0	—	—	—	—	—	—
		3,0,2,1	—	—	—	—	—	—
	RX	0,1,3,2	case2	U(0)	C(2)	U(0)	—	1,3,2,0
		1,0,2,3	case1	U(1)	C(3)	—	—	2,0,1,3
		2,1,3,0	—	—	—	—	—	—
		3,0,2,1	—	—	—	—	—	—
	Y	0,1,3,2	case1	U(0)	C(2)	—	—	1,3,0,2
		1,0,2,3	case1	U(1)	C(3)	—	—	0,2,1,3
		2,1,3,0	—	—	—	—	—	—
		3,0,2,1	—	—	—	—	—	—
	RY	0,1,3,2	case2	U(0)	C(2)	U(0)	—	3,1,2,0
		1,0,2,3	case2	U(1)	C(3)	U(1)	—	2,0,3,1
		2,1,3,0	—	—	—	—	—	—
		3,0,2,1	—	—	—	—	—	—
RY	X	0,3,1,2	—	—	—	—	—	—
		1,2,0,3	—	—	—	—	—	—
		2,3,1,0	case2	U(2)	C(0)	U(2)	—	3,1,0,2
		3,2,0,1	case1	U(3)	C(1)	—	—	0,2,3,1
	RX	0,3,1,2	—	—	—	—	—	—
		1,2,0,3	—	—	—	—	—	—
		2,3,1,0	case1	U(2)	C(0)	—	—	1,3,2,0
		3,2,0,1	case2	U(3)	C(1)	U(3)	—	2,0,1,3
	Y	0,3,1,2	—	—	—	—	—	—
		1,2,0,3	—	—	—	—	—	—
		2,3,1,0	case2	U(2)	C(0)	U(2)	—	1,3,0,2
		3,2,0,1	case2	U(3)	C(1)	U(3)	—	0,2,1,3
	RY	0,3,1,2	—	—	—	—	—	—
		1,2,0,3	—	—	—	—	—	—
		2,3,1,0	case1	U(2)	C(0)	—	—	3,1,2,0
		3,2,0,1	case1	U(3)	C(1)	—-	—	2,0,3,1

A.2 クロール歩容から循環歩容への遷移

表 A.3: 遷移中と遷移後の遊脚順序 (クロール → 循環 1)

前	後	U,A,B,C	case	step1	step2	step3	step4	遷移後の遊脚順序
X	O	0,1,3,2	case1	U(0)	A(1)	C(2)	—	3,0,1,2
			case2	U(0)	C(2)	A(1)	—	2,3,0,1
		1,2,0,3	—	—	—	—	—	—
			—	—	—	—	—	—
		2,3,1,0	—	—	—	—	—	—
			—	—	—	—	—	—
		3,0,2,1	case1	U(3)	A(0)	C(1)	—	2,3,0,1
			case2	U(3)	C(1)	A(0)	—	1,2,3,0
	RO	0,3,1,2	case1	U(0)	A(3)	C(2)	—	1,0,3,2
			case2	U(0)	C(2)	A(3)	—	2,1,0,3
		1,0,2,3	—	—	—	—	—	—
			—	—	—	—	—	—
		2,1,3,0	—	—	—	—	—	—
			—	—	—	—	—	—
		3,2,0,1	case1	U(3)	A(2)	C(1)	—	0,3,2,1
			case2	U(3)	C(1)	A(2)	—	3,0,1,2
RX	O	0,1,3,2	—	—	—	—	—	—
			—	—	—	—	—	—
		1,2,0,3	case1	U(1)	A(2)	C(3)	—	0,1,2,3
			case2	U(1)	C(3)	A(2)	—	3,0,1,2
		2,3,1,0	case1	U(2)	A(3)	C(0)	—	1,2,3,0
			case2	U(2)	C(0)	A(3)	—	0,1,2,3
		3,0,2,1	—	—	—	—	—	—
			—	—	—	—	—	—
	RO	0,3,1,2	—	—	—	—	—	—
			—	—	—	—	—	—
		1,0,2,3	case1	U(1)	A(0)	C(3)	—	2,1,0,3
			case2	U(1)	C(3)	A(0)	—	3,2,1,0
		2,1,3,0	case1	U(2)	A(1)	C(0)	—	3,2,1,0
			case2	U(2)	C(0)	A(1)	—	1,0,3,2
		3,2,0,1	—	—	—	—	—	—
			—	—	—	—	—	—

表 A.4: 遷移中と遷移後の遊脚順序 (クロール → 循環 2)

前	後	U,A,B,C	case	step1	step2	step3	step4	遷移後の遊脚順序
Y	O	0,1,3,2	case1	U(0)	A(1)	C(2)	—	3,0,1,2
			case2	U(0)	C(2)	A(1)	—	2,3,0,1
		1,2,0,3	case1	U(1)	A(2)	C(3)	—	0,1,2,3
			case2	U(1)	C(3)	A(2)	—	3,0,1,2
		2,3,1,0	—	—	—	—	—	—
			—	—	—	—	—	—
		3,0,2,1	—	—	—	—	—	—
			—	—	—	—	—	—
	RO	0,3,1,2	case1	U(0)	A(3)	C(2)	—	1,0,3,2
			case2	U(0)	C(2)	A(3)	—	2,1,0,3
		1,0,2,3	case1	U(1)	A(0)	C(3)	—	2,1,0,3
			case2	U(1)	C(3)	A(0)	—	3,2,1,0
		2,1,3,0	—	—	—	—	—	—
			—	—	—	—	—	—
		3,2,0,1	—	—	—	—	—	—
			—	—	—	—	—	—
RY	O	0,1,3,2	—	—	—	—	—	—
			—	—	—	—	—	—
		1,2,0,3	—	—	—	—	—	—
			—	—	—	—	—	—
		2,3,1,0	case1	U(2)	A(3)	C(0)	—	1,2,3,0
			case2	U(2)	C(0)	A(3)	—	0,1,2,3
		3,0,2,1	case1	U(3)	A(0)	C(1)	—	2,3,0,1
			case2	U(3)	C(1)	A(0)	—	1,2,3,0
	RO	0,3,1,2	—	—	—	—	—	—
			—	—	—	—	—	—
		1,0,2,3	—	—	—	—	—	—
			—	—	—	—	—	—
		2,1,3,0	case1	U(2)	A(1)	C(0)	—	3,2,1,0
			case2	U(2)	C(0)	A(1)	—	0,3,2,1
		3,2,0,1	case1	U(3)	A(2)	C(1)	—	0,3,2,1
			case2	U(3)	C(1)	A(2)	—	1,0,3,2

A.3 循環歩容からクロール歩容への遷移

表 A.5: 遷移中と遷移後の遊脚順序 (循環 → クロール)

前	後	U,A,B,C	case	step1	step2	step3	step4	遷移後の遊脚順序
O	X	0,3,1,2	—	U(0)	A(3)	B(1)	—	0,2,3,1
(RO)		1,0,2,3	—	U(1)	A(0)	B(2)	—	3,1,0,2
		2,3,1,0	—	U(2)	A(3)	B(1)	—	0,2,3,1
		3,0,2,1	—	U(3)	A(0)	B(2)	—	3,1,0,2
	RX	0,1,3,2	—	U(0)	A(1)	B(3)	—	2,0,1,3
		1,2,0,3	—	U(1)	A(2)	B(0)	—	1,3,2,0
		2,1,3,0	—	U(2)	A(1)	B(3)	—	2,0,1,3
		3,2,0,1	—	U(3)	A(2)	B(0)	—	1,3,2,0
	Y	0,1,3,2	—	U(0)	A(1)	B(3)	—	0,2,1,3
		1,0,2,3	—	U(1)	A(3)	B(2)	—	1,3,0,2
		2,1,3,0	—	U(2)	A(1)	B(3)	—	0,2,1,3
		3,0,2,1	—	U(3)	A(3)	B(2)	—	1,3,0,2
	RY	0,3,1,2	—	U(0)	A(3)	B(1)	—	2,0,3,1
		1,2,0,3	—	U(1)	A(2)	B(0)	—	3,1,2,0
		2,3,1,0	—	U(2)	A(3)	B(1)	—	2,0,3,1
		3,2,0,1	—	U(3)	A(2)	B(0)	—	3,1,2,0

A.4 循環歩容から循環歩容への遷移

表 A.6: 遷移中と遷移後の遊脚順序 (循環 → 循環)

前	後	U,A,B,C	case	step1	step2	step3	step4	遷移後の遊脚順序
O	RO	0,3,1,2	—	U(0)	A(3)	C(2)	B(1)	0,3,2,1
		1,0,2,3	—	U(1)	A(0)	C(3)	B(2)	1,0,3,2
		2,3,1,0	—	U(2)	A(1)	C(0)	B(3)	2,1,0,3
		3,0,2,1	—	U(3)	A(2)	C(1)	B(0)	3,2,1,0
RO	O	0,1,3,2	—	U(0)	A(1)	C(2)	B(3)	0,1,2,3
		1,2,0,3	—	U(1)	A(2)	C(3)	B(0)	1,2,3,0
		2,1,3,0	—	U(2)	A(3)	C(0)	B(1)	2,3,0,1
		3,2,0,1	—	U(3)	A(0)	C(1)	B(2)	3,0,1,2

付 録B　RTLinuxによる実時間制御

B.1　RTLinux とは

Real-Time Linux は、Linux OS と共存可能な リアルタイム OS である。しかし、厳密な意味ではリアルタイム OS ではなく、RTLinux が提供するのは スケジューラとプロセス間通信のみである。RTLinux は、仮想マシンを Linux に提供し、Linux を優先順位の低い 1 つのリアルタイムプロセスとして実行することで、リアルタイム処理と従来の Linux OS の処理の共存を実現している。このリアルタイム処理というのは、ある幅で処理開始タイミングと処理終了までの時間制限を保証した処理のことである。RTLinux はリアルタイム OS であり、リアルタイム OS は、リアルタイム処理を行うマルチタスク OS である。しかし、CPU の処理能力を越える処理は、とうぜん制約された時間内では処理できない。

ここで、MS-DOS 上で割り込み処理プログラムを走らせたのと、どう違うと言うのだろうかという疑問もわいてくる。たしかにタイマーカウンタのハードウエアを用意して DOS の割り込みを使えば、DOS でもリアルタイム処理は可能である。しかし、MS-DOS は一度に 1 つのプログラムしか走らせることができない上、開発環境が UNIX の資産を受け継いでいる Linux に比べると遥かに貧弱である。RTLinux ならば、リアルタイム処理を実行しながら、複数のプログラムを実行できる上、開発環境も整っているので開発も楽にできるという強みがある。また、タイマーカウンタ等のハードウェアを用意しなくても、（CPU にそこまでの処理能力があるかどうかは別として）OS 自体が 10^{-9} 程度の時間制約まで守れる設計になっている。MS-DOS や MS-Windows といった OS の設計では 1ms 以下のサンプリングは保証されない[1]。

したがって、安価で開発環境の整っている RTLinux の方が、リアルタイム制御を行うのに（開発行程も含めて）向いているといえるだろう。

B.2　RTLinux カーネルの入手とインストール

RTLinux カーネルは、非商用で開発されているいわゆる Free Soft である。カーネルはインターネットで公開されており、www.rtlinux.org からだれでも自由に入手できる。RTLinux のカーネルをダウンロードするときは、現在使用中の Linux カーネルのバージョンに合わせてダウンロードする。例えば、カーネルが 2.2.x な

[1]Intel 社の 80386 命令アーキテクチャ AI-32 の仮想 86 モードが使用できないため。MS-DOS ベースの OS は AI-32 のリアルモードしか使用できないが、RTLinux は AI-32 の能力をフル活用することができる。

らば、rtl-v2.x を選ぶ。インストールする際には Linux カーネルに RTLinux のパッチを当てることになるので、パッチ済カーネルをダウンロードしてきた方が簡単でよい（ただし、ダウンロードには時間がかかる）。

ダウンロードしてきたファイル (例えば rtlinux-2.0-prepatched.tgz) は、展開する場所 /usr/src に展開する。それからインストールが完了するまでの手順は、

1. README, INSTLL, GettingStarted.txt, FAQ などの文書に目を通す。
2. INSTALL に書いてある手順にしたがって、カーネルをコンパイルする。
   (a) コンフィギュレーション
   (b) コンパイル
   (c) インストール、再起動
3. man path の追加

rtl-v2.x には RTLinux で使用する関数の man page がついている。再起動したら、実際に RTLinux がインストールされたことを確認するために、rtl/examples 以下にあるサンプルプログラムを実行してみよう。

B.3 module を作る

RTLinxu では、リアルタイム処理のプログラムをモジュールの形で作成するので、まずはじめにモジュールについて説明する。モジュールはカーネルを拡張する小さな部品である。Linux module のソースコードは、main のかわりに、一組の init_module と cleanup_module という関数をもっている。

まず、モジュールプログラムの第一歩として、何もしない hello_module.o を作ってみよう。

hello_module.c

```
/*  何もしない module の作成    hello_module.c  */
#define MODULE
#include <linux/module.h>

int  init_module(void)    {  printk("Hello world !\n");  return 0; }
void cleanup_module(void) {  printk("Goodbye world \n"); }
```

コンパイル、実行、確認には、

```
[root]# gcc -c hello_module.c
[root]# insmod hello_module.o
[root]# lsmod
Module                  Size  Used by
hello_module             180   0  (unused)
[root]# rmmod hello_module
```

と入力すればよい。

X 上のターミナルから実行したのでは何も表示されない。printk の出力は、コンソールから実行して確認するか、dmesg により吐き出される文字列から確認できる。insmod,lsmod,rmmod といったコマンドは、スーパーユーザ (root) しか実行できない。

insmod により モジュールがカーネルに組み込まれると、最初に 関数 init_module が呼び出される。デバイスドライバ等を作成する場合は、ここでデバイスの登録や I/O ポートの設定、メモリの確保などを行う。init_module は、普通は戻値として 0 を返し、モジュールの初期化処理に失敗した場合は 負の値を返すことになっている。逆に、rmmod によりモジュールがカーネルから削除されると、関数 cleanup_module が呼び出される。

RTLinux では、このモジュールの中でリアルタイム・スレッド[2] の生成、スケジューリング、実行、停止、削除等を行う。したがって、RTLinux における リアルタイム処理 のプログラムは、大まかに言えば以下のような作りになる。

- init_module において thread を生成し、thread の本体となる関数を指定する。
- thread のスケジューリングと実行／停止を行う。
- cleanup_module において thread を削除する。

このリアルタイム処理プログラムの大まかな構造をもとに、次はサンプルプログラムについて説明する。

B.4 簡単なリアルタイム処理プログラム

こんどは実際に簡単なリアルタイム処理プログラムについて説明する。プログラムの例として、付録 A に示すプログラムを使用する。このサンプルプログラムはリアルタイム・スレッドを 1 つだけ生成し、ユーザプログラムからこのスレッドを起動／停止し、スレッドでリアルタイム処理されたデータをユーザプログラムに返すという、リアルタイム処理と FIFO を用いたプログラム間通信の雛形となるプログラムである。さらに、このプログラムはスレッド内での浮動小数点演算の実行も実現している。

このプログラムの大まかな流れを示すと、以下のようになる。

1. timer_module.o をカーネルに組み込む
   (a) RT-FIFO の生成
   (b) リアルタイム・スレッドの生成とスケジューリング

[2] ここで タスク **(task)** やプロセス **(process)**、スレッド **(thread)** という概念が出てくるが、すべて実行中のプログラムという意味で理解してほしい。厳密に言えば、プロセスは並列処理システムを CPU 側からでなく、プログラム側から考えた概念であり、同じものを Intel の CPU アーキテクチャである IA-32 の用語ではタスクと呼ぶ。これがカーネル内ではスレッドという処理単位になるのだが、CPU が 1 つしかない場合は、視点が違うだけで同じものを指しているだけである。

(c) FIFO ハンドラの登録

2. アプリケーションを実行

(a) RT-FIFO を開く

(b) リアルタイム・スレッドを起動する指示を FIFO を通して timer_module に送る

(c) timer_module の FIFO ハンドラが指示を受け取り、スレッドに渡す

(d) スレッドが周期的に実行されると、データの値が計算される

(e) 変化したデータの値を FIFO を通して usr_app に送る

(f) ループが回っている間、timer_module からデータを受け取り表示する

(g) スレッドを停止する

3. timer_module をカーネルから削除する

(a) FIFO を削除する

(b) スレッドを削除する

この流れに沿ってソースコードを追ってみることにする。

このプログラムのエントリーポイントとなるのは、timer_module.c の 関数 init_module である。コマンド insmod により timer_module.o がカーネルに組み込まれると、

```
rtf_destroy(1);
rtf_destroy(2);
rtf_destroy(3);

rtf_create(1, 200);
rtf_create(2, 200);
rtf_create(3, 1000);
```

の部分で /dev/rtf1, /dev/rtf2, /dev/rtf3 に RT-FIFO の生成を行う。続いて、

```
pthread_attr_init (&attr);
sched_param.sched_priority = 4;
pthread_attr_setschedparam (&attr, &sched_param);
ret = pthread_create (&theThread,  &attr, thread_code, (void *)junk );
```

で、リアルタイム・スレッドの生成とスケジューリングを行う。この処理により、優先順位が 4 のスレッドが生成され、このスレッドは 関数 thread_code を実行するものとなる。

```
pthread_setfp_np (theThread, 1);

rtf_create_handler(COMMAND_FIFO, &my_handler);
```

その後、先ほど生成したスレッドで浮動小数点演算を行うことを許可し、FIFO ハンドラーに 関数 my_handler を登録して、関数 init_module が終了する。

つぎに、ユーザプログラムが起動すると usr_app.c の 関数 main にが始まり、

```
if ((ctl = open("/dev/rtf1", O_WRONLY)) < 0) {
  fprintf(stderr, "Error opening /dev/rtf1\n");
  exit(1);
}

if ((fd0 = open("/dev/rtf3", O_RDONLY)) < 0) {
  fprintf(stderr, "Error opening /dev/rtf3\n");
  exit(1);
}
```

の部分で/dev/rtf1 と /dev/rtf3 の RT-FIFO が開かれ通信可能な状態になる。/dev/rtf1 はスレッド制御メッセージ用に書き込み専用、/dev/rtf3 はデータ用に読み込み専用としてオープンする。

ユーザプログラムからスレッドに制御メッセージを送ると、設定した周期でスレッドが走り始める。

```
// start the thread
Msg.command = START_THREAD;
Msg.period = 0.5;
if (write(ctl, &Msg, sizeof(STMsg)) < 0) {
  fprintf(stderr, "Can't send a command to RT-thread\n");
  exit(1);
}
```

ここでは周期を 0.5[s] に指定した。ここでまた処理を timer_module の方に戻って追い掛ける。(ただし、ここでユーザアプリケーションが一時停止するわけではない)

timer_module 側では FIFO からメッセージを受け取ると、関数 my_handler が呼び出される。

```
while( (err = rtf_get(COMMAND_FIFO, &Msg, sizeof(STMsg)))==sizeof(STMsg) ){
  rtf_put (COMMAND_FIFO + 1, &Msg, sizeof(STMsg));
  pthread_wakeup_np (theThread);
}
```

関数 my_handler は FIFO から STMsg の大きさだけメッセージを読み出し、これをスレッドに引き渡す。この rtf_put の第 1 引数がメッセージの引き渡し先である。引き渡し先を複数用意すれば、複数のスレッドを制御することが可能である。

関数 my_handler からスレッドがメッセージを受け取ると、処理がスレッドの本体である関数 thread_code に移る。

スレッドの本体

```
while (1) {
  int ret;
  int err;

  ret = pthread_wait_np();
  err = rtf_get (COMMAND_FIFO + 1, &Msg, sizeof(STMsg));

  if( err == sizeof(STMsg) ){

    switch (Msg.command) {
    case START_THREAD :
      pthread_make_periodic_np(pthread_self(),
                               gethrtime(),
                               real2in(Msg.period) );
      break;

    case STOP_THREAD :
      pthread_suspend_np(pthread_self());
      break;

    default:
      rtl_printf("RTL task: bad command\n");
      return 0;
    }
  }

  // Write something here to execute periodicaly.

  Data.up++;
  Data.down -= 0.5;
  rtf_put(DATA_FIFO, (char *)(&Data), sizeof(STData) );
} // end of while
```

thread_code の無限ループは、普段は関数 pthead_wait_np により次の実行周期を待機する状態になっている。関数 my_handler から pthread_wakeup_np によりスレッドが起動されると、スレッドは my_handler から渡されたメッセージを読みに行く。このメッセージに基づき、 関数 pthread_make_periodic_np でスレッドの実行周期を決定して周期処理を開始する。ここで、周期を指定している第 3 引数の値 real2in(Msg.period) は、マクロ定義による関数で、指定された秒数をコンピュータの内部時間に置き換える関数である。その実体は、

```
#define in2real(i)      ( (double)(i) / NSECS_PER_SEC )
```

のように定義している。

スレッドが動き始めると、データの値を計算してデータ構造体の中身を DATA_FIFO に送り、関数 pthread_wait_np により CPU の使用権を放棄して次の周期まで待つ。そして次の周期が来るとまた COMMAND_FIFO にメッセージを読みに行き、メッ

セージがなければデータの値を更新してデータ構造体の中身を DATA_FIFO に送り … という処理を続ける。

スレッドが動き続けている間、DATA_FIFO にはデータ構造体の中身が送られ続け、ユーザプログラム側では

```
// main loop
for(i=0; i<10; i++){

  FD_ZERO(&rfds);
  FD_SET(fd0, &rfds);
  tv.tv_sec = 1;
  tv.tv_usec = 0;

  retval = select(FD_SETSIZE, &rfds, NULL, NULL, &tv);
  if (retval > 0) {
    if (FD_ISSET(fd0, &rfds)) {
      read(fd0, &Data, sizeof(STData) );
      printf("count : %d  %3.1f \n", Data.up, Data.down );
    }
  }

}
```

という部分で DATA_FIFO からデータの読み出しを行うことができる。

もしここで、スレッドの周期が速いためにユーザプログラムが DATA_FIFO の中身を受け取りきらないまま、スレッドからどんどん書込があれば、FIFO を生成したときに指定したバッファの大きさまでは保存され、バッファの大きさを越えると古いメッセージから破棄されることになる。

逆に、スレッドの周期が遅く、ユーザプログラムが DATA_FIFO の読み出しをまっていてもなかなかデータが送られてこない場合は、

```
  tv.tv_sec = 1;
  tv.tv_usec = 0;
```

で指定したタイムアウト期間だけ待ち、それでもデータが来なければ、その回は読み出しを行わない。このようにして、サンプルプログラムは、ユーザプログラムのメインループが回っている間はスレッドからのデータを読み込んでは表示するという作業を繰り返す。

メイン関数のループが終了すると今度はスレッドに停止命令を出す。

```
// stop the thread
Msg.command = STOP_THREAD;
if (write(ctl, &Msg, sizeof(STMsg)) < 0) {
  fprintf(stderr, "Can't send a command to RT-thread\n");
  exit(1);
}
```

するとスレッドは DATA_FIFO から関数 my_mandler を介してメッセージを受け

取り、

```
      case STOP_THREAD :
        pthread_suspend_np(pthread_self());
        break;
```

このようにしてスレッドを停止する。スレッドはこの状態で停止していても、また起動するメッセージを受け取れば、いつでも周期処理に入ることができる。

ユーザアプリケーションが終了すれば、もう timer_module は不要なのでコマンド rmmod によりカーネルから削除する。そうすると、関数 cleanup_module が呼び出され、

```
void cleanup_module(void)
{
  rtf_destroy(1);
  rtf_destroy(2);
  rtf_destroy(3);

  pthread_delete_np (theThread);
}
```

使用した RT-FIFO とスレッドを削除して、サンプルプログラムの全行程が終了する。

B.5 まとめ

- RTLinux は、Linux OS と共存可能なリアルタイム OS である。
- RTLinux では、リアルタイム処理をスレッドという処理単位で実行する。
- スレッドはモジュール内で生成、起動、停止、削除される。
- ユーザプログラムとスレッドとの通信には、FIFO を使用する。
- モジュールプログラムは書式が決まっているので、リアルタイム処理プログラムを書くには、雛形の必要なところだけを変更すれば良い。

B.6 簡単なリアルタイム処理プログラムのサンプル

B.6.1 control.h

```
/*
  Simple RTL example program
  control.h
 */

#define COMMAND_FIFO    1
#define DATA_FIFO       3

#define START_THREAD    1
#define STOP_THREAD     2

// RT thread control message
typedef struct {
  int     command;
  double  period;
} STMsg;

// data from RT thread
typedef struct {
  int    up;
  double down;
} STData;
```

B.6.2 timer_module.c

```
/*
  Simple RTL example program
  timer_module.c
 */

#include <linux/errno.h>
#include <rtl.h>
#include <time.h>
#include <rtl_sched.h>
#include <rtl_fifo.h>
#include "control.h"

#define in2real(i)      ( (double)(i) / NSECS_PER_SEC )
#define real2in(r)      ( (long long)(NSECS_PER_SEC * (r) ) )

pthread_t theThread;

void *thread_code(void *t)
{
```

```
  STMsg   Msg;
  STData  Data = {0, 5.0};

  while (1) {
    int ret;
    int err;

    ret = pthread_wait_np();
    err = rtf_get (COMMAND_FIFO + 1, &Msg, sizeof(STMsg));

    if( err == sizeof(STMsg) ){

      switch (Msg.command) {
      case START_THREAD :
        pthread_make_periodic_np(pthread_self(),
                                 gethrtime(),
                                 real2in(Msg.period) );
        break;

      case STOP_THREAD :
        pthread_suspend_np(pthread_self());
        break;

      default:
        rtl_printf("RTL task: bad command\n");
        return 0;
      }
    }

    // Write something here to execute periodicaly.

    Data.up++;
    Data.down -= 0.5;
    rtf_put(DATA_FIFO, (char *)(&Data), sizeof(STData) );
  } // end of while

  return 0;
}

int my_handler(unsigned int fifo)
{
  STMsg Msg;
  int   err;

  while( (err = rtf_get(COMMAND_FIFO, &Msg, sizeof(STMsg)))==sizeof(STMsg) ){
    rtf_put (COMMAND_FIFO + 1, &Msg, sizeof(STMsg));
    pthread_wakeup_np (theThread);
  }

  if (err != 0) {
```

```
    return -EINVAL;
  }
  return 0;
}


int init_module(void)
{
  pthread_attr_t attr;
  struct sched_param sched_param;
  int ret;
  int junk = 0;

  rtf_destroy(1);
  rtf_destroy(2);
  rtf_destroy(3);

  rtf_create(1, 200);
  rtf_create(2, 200);
  rtf_create(3, 1000);

  pthread_attr_init (&attr);
  sched_param.sched_priority = 4;
  pthread_attr_setschedparam (&attr, &sched_param);
  ret = pthread_create (&theThread,  &attr, thread_code, (void *)junk );

  pthread_setfp_np (theThread, 1);

  rtf_create_handler(COMMAND_FIFO, &my_handler);

  return 0;
}


void cleanup_module(void)
{
  rtf_destroy(1);
  rtf_destroy(2);
  rtf_destroy(3);

  pthread_delete_np (theThread);
}
```

B.6.3 usr_app.c

```
/*
  Simple RTL example program
  usr_app.c
 */
```

```
#include <stdio.h>
#include <sys/time.h>
#include <fcntl.h>
#include <unistd.h>
#include <rtl_fifo.h>
#include <rtl_time.h>
#include "control.h"

int main(int argc, char **argv)
{
  int             i;
  fd_set          rfds;
  struct timeval tv;
  int             retval;
  int             ctl, fd0;
  STMsg           Msg;
  STData          Data;

  ////////////////////////////////////////////////////////
  // Open RT-FIFO
  if ((ctl = open("/dev/rtf1", O_WRONLY)) < 0) {
    fprintf(stderr, "Error opening /dev/rtf1\n");
    exit(1);
  }

  if ((fd0 = open("/dev/rtf3", O_RDONLY)) < 0) {
    fprintf(stderr, "Error opening /dev/rtf3\n");
    exit(1);
  }

  // start the thread
  Msg.command = START_THREAD;
  Msg.period = 0.5;
  if (write(ctl, &Msg, sizeof(STMsg)) < 0) {
    fprintf(stderr, "Can't send a command to RT-thread\n");
    exit(1);
  }

  ////////////////////////////////////////////////////////
  // main loop
  for(i=0; i<10; i++){

    FD_ZERO(&rfds);
    FD_SET(fd0, &rfds);
    tv.tv_sec = 1;
    tv.tv_usec = 0;

    retval = select(FD_SETSIZE, &rfds, NULL, NULL, &tv);
    if (retval > 0) {
```

```
      if (FD_ISSET(fd0, &rfds)) {
        read(fd0, &Data, sizeof(STData) );
        printf("count : %d  %3.1f \n", Data.up, Data.down );
      }
    }

  }

  ////////////////////////////////////////////////////////
  // stop the thread
  Msg.command = STOP_THREAD;
  if (write(ctl, &Msg, sizeof(STMsg)) < 0) {
    fprintf(stderr, "Can't send a command to RT-thread\n");
    exit(1);
  }

  return 0;
}
```

B.6.4 Makefile

```
all: timer_module.o usr_app
# simple.o

include rtl.mk

usr_app: usr_app.c
        $(CC) ${INCLUDE} -O2 -Wall usr_app.c -o usr_app

timer_module.o: timer_module.c
        $(CC) ${INCLUDE} ${CFLAGS} -c timer_module.c -o timer_module_tmp.o
        ld -r -static timer_module_tmp.o -o timer_module.o -L/usr/lib -lm
        rm -f timer_module_tmp.o

#test, remove any modules, load new ones and run app
test: all
        (cd ${RTL_DIR}; ./rmrtl)
        @echo "Now insert the fifo and scheduler"
        @echo "Type <return> to continue"
        @read junk
        (cd ${RTL_DIR}; ./insrtl)
        @echo "Now start the real-time tasks  module"
        @echo "Type <return> to continue"
        @read junk
        @insmod timer_module.o
        @echo "Now start the application"
        @echo "Type <return> to continue"
        @read junk
```

```
        @./usr_app
        rmmod timer_module
        (cd ${RTL_DIR}; ./rmrtl)

clean:
        rm -f usr_app *.o *~

insrtl :
        (cd ${RTL_DIR}; ./insrtl)

rmrtl :
        (cd ${RTL_DIR}; ./rmrtl)
```