修士論文

2001年3月

自律移動ロボットの協調行動に関する研究

知識科学研究科　高橋洋一

修 士 論 文

自律移動ロボットの協調行動に関する研究

北陸先端科学技術大学院大学
知識科学研究科知識システム基礎学専攻

高橋洋一

2001 年 3 月

修 士 論 文

自律移動ロボットの協調行動に関する研究

指導教官　櫻井彰人 教授

北陸先端科学技術大学院大学
知識科学研究科知識システム基礎学専攻

950052 高橋洋一

審査委員：　櫻井彰人 教授 (主査)
林幸雄 助教授
橋本敬 助教授

2001 年 2 月

目 次

図 目 次

第1章

はじめに

1.1 研究の背景と目的

ロボット技術は, 制御用のコンピュータに関する研究を行う情報工学から, 要素技術であるアクチュエータやセンサの開発を行う機械工学, 電気電子工学などの分野まで多岐にわたる, いわば現代の科学技術の集大成といえるものである.

こうしたロボットに代表されるような大規模なシステムは従来, 長期にわたって安定に動作し, 常に最適な特性, 効率を維持することが最も望ましいとされてきた (伊藤 2000).

このような要求を満たすために設計者は, 外部の環境変動による影響をできるだけ抑え, かつ最適な効率が維持されるようなシステムを実現することに努力を傾けてきた.

そのためには,1948 年にノーバート・ウィナーが発表したサイバネティックスの理論が発表されて以後, 急速に研究が進み, 発展した, フィードバック制御機構を中心とした人工システムの設計理論が有効であった.

ところが,1980 年代後半から 90 年代にかけて, インターネットや知能ロボットなどの新しい人工システムが登場してきた, 例えば, 知能ロボットは, 周りの人間や環境に対して自らの応答を適応させることにその価値がある. たとえ効率を犠牲にしても, 変動する環境に対して適切に対処することが求められる. すなわち, これからのシステムは安定と効率ではなく, 変化と適応が重要になることがはっきりしてきた.

そのような流れの中, 近年, 複数の自律した移動ロボットを協調的に行動させて効率よく運用しようという要求が各方面から出されてきている. そのため, 複数の移動ロボットについての研究が盛んに行われてきている.

本研究では, 特に追随行動を扱う. 追随行動の従来研究としては, 群秩序生成可能な追従戦略を用いた協調行動を行わせる研究がある (新井 民夫 1994). これは人工ポテンシャル法を動的環境下において局所情報のみを用いて実時間で計画できるように単純化した分散管理型局所計画器 LAP(Local Avoidance Planner) である仮想インピーダンス法に追従戦略を組み合わせて, 追従戦略を動作計画に組み込む

ことにシミュレーション上で成功している.

ところで, 従来の古典的プラニングの処理は, 観測, モデリング, プラニング, 実行という流れで進行する一連の手続きであり, 生成されたプランが実環境で確実に実行されることが前提となっている. しかし, 移動ロボットを取り巻く環境は, 移動ロボットが関与しないところで状態が刻々と変化する動的環境である. このような環境では古典的プラニングの成功は保証されない. このような課題を解決する手法として多重目標, 多重センサ, 頑健性, 拡張性の点で優れていると言われる服属アーキテクチャ(subsumption architecture)(R.A.Brooks 1986) を本研究では適用する.

つまり, 本研究では, 服属アーキテクチャを適用した移動ロボットの追随行動に関する問題を取り扱う.

前述した通り, 移動ロボットを取り巻く環境には不確実性が生じており, 局所的にしか得られない環境に関する情報をもとに追随行動を行わなければならない. そこで, 服属アーキテクチャに基づいたシステムを実装したロボットに追随行動を行わせることによって, 服属アーキテクチャの優位性を実証する. その際の比較対象として, 静的環境のセンサ値から出力へマッピングを行い, 環境の認識を行うモジュールを, フィードフォワード型ニューラルネットワークで実装する. これと服属アーキテクチャに基づいて実装者の経験により設計されたモジュールとを比較して考察を行う.

さらに, 実験で観察された現象をもとに, 追随行動に関する課題と問題点について考察を行う. これが本研究の大きな目的である.

1.2 本論文の要旨

以下に本論文の構成を述べる. 第 2 章では, 本研究において最も重要なアーキテクチャである, 服属アーキテクチャについて説明する. 第 3 章では, 本研究において行う実験の準備として, タスクと実験環境, そして使用する移動ロボットの能力についてまとめ, 実装の参考とする. また, 合わせて既存の研究例の 1 つを紹介する. 第 4 章では, 移動ロボットに実装したシステムの構成について詳しく説明する. 第 5 章では, 実機を用いて行った実験について, 方法とその結果をまとめる. 第 6 章では, 第 5 章で行った実験に関する考察を行う.

第2章

服属アーキテクチャ

知能ロボットに対する典型的なアーキテクチャは, 以下のような処理形態を持つ (吉田 1993). まず, センサ系が環境を認識する (perception). 続いて, 認識結果から内部に外界の物体, 事象のモデルを何らかの形で作る. (modeling). 次にモデルに応じた行動計画を発生し (planning), さらに計画を実行すべきタスク群に分解し, 実行する (task execution). そして分解した動作群を順次駆動系 (actuators) へ伝達する. 従来の知能ロボットのほとんどがこのような処理形態を持っている.

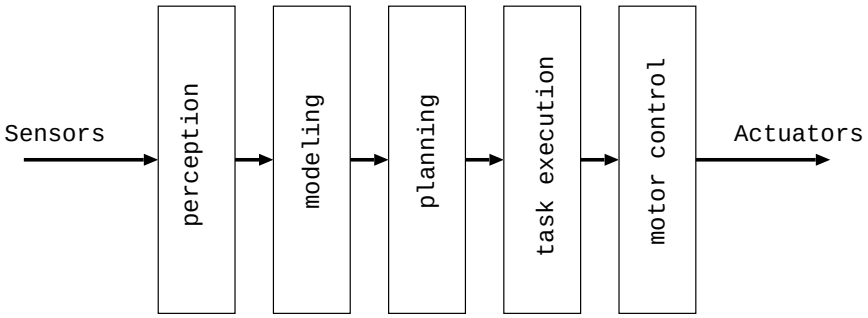

図 2.1: 機能モジュール構成による移動ロボット制御システム

制御システムを構成する一般的な方法は, 問題を制御システム内部処理機能単位に扱う分割によって行われる. つまり, 問題を図 2.1 に示すように分割する.

このプラニングの最大の問題点は, 生成されたプランが現実世界で確実に実行されるという前提を置いていることである (山田誠二 1997). 実世界では, そのような前提が成り立つことの方が稀である. 環境は絶えず変化し, 変化しないまでもプラニングされた行為はしばしば失敗する.

これに代わるものとして,Brooks は非同期にタスクを遂行する行動 (behavior) により, 分割された図 2.2 のようなアーキテクチャを提案した (R.A.Brooks 1986). このアーキテクチャでは, 上位レベルが下位レベルの行動を抑制するなどの制御をするために, 服属アーキテクチャ(subsumption architecture) と呼ばれる. 各レベルのタスクはセンサからの情報を直接受取り並列に処理が進むため, 服属アーキテク

チャは従来のプラニングよりも, 多重目標, 多重センサ, 頑健性, 拡張性の点で優れているといわれる.

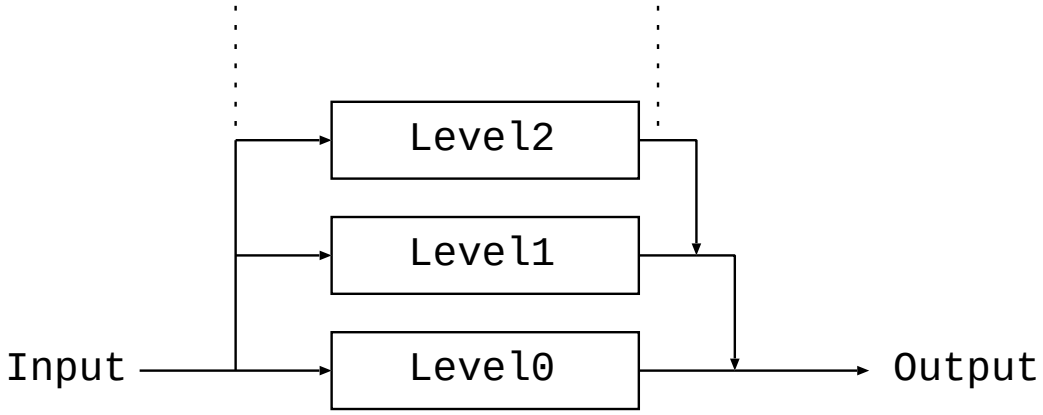

図 2.2: 服属アーキテクチャ

各レベルは, 図 2.3 のようなモジュールを接続して構成されている. モジュールの入力はセンサか他のモジュールの出力に接続され, 出力はアクチュエータか他のモジュールの入力に接続される, さらに, 次のような接続も可能である.

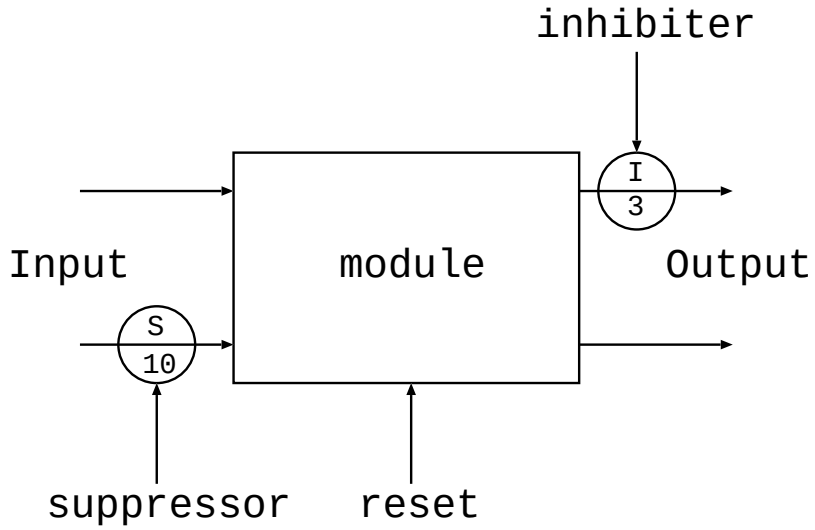

図 2.3: モジュールの概略

□ 抑制 (inhibiter):出力に接続され, トリガーにより, その出力は一定時間削除される.

□ 置換 (suppressor):入力に接続され, トリガーにより, その入力は一定時間上位レベルの出力で置き換えられる.

図 2.3 中の丸の中の I と S が抑制と置換を表わし, その下の数字が上記の一定時間である.

システムを構築していく際には, まず, レベル 0 の能力を有する, 完全なロボットの制御システムを構築する, これは, 図のようなモジュール同士を接続して構築する. このとき, レベル 0 の制御システムのデバッグを行い, その後はシステム変更を行わない. 次に別の制御層を構築し, これをレベル 1 の制御システムと呼ぶ. レベル 1 の制御システムは, レベル 0 のシステムのデータに対して, 前述したような抑制と置換を行うことができる. この層は, 時折データ路に干渉してくる上位層には全く気づかずに動作し続ける. 同様の過程を繰り返して, 図に示すような高次の能力を達成する. このような方式を用いると, 開発の初期段階で動作可能なシステムを入手したことになる. つまり, 付加的な層は後で加えればよく, 以前に動作していたシステムを変更する必要がない.

しかしながら,Brooks の服属アーキテクチャは反応型エージェントの構築には向いているが言語処理や対話などを含む高度に知的な作業を行う記号処理のモデルとしては必ずしも適切ではないとの指摘がある (中島 1999).

その主張は, 服属アーキテクチャは理想的には各層は独立で, 必要なときには上位の層が下位の層の出力を抑圧する構造であるが, これをこのまま実現することは不可能であるというものである。現実的には下位の層の内部処理を抑圧する必要がある. 例えば,2 個のモータを駆動して動くロボットの場合, 下層の歩行モジュールのモータ駆動回路にバイアスをかけることにより, 左右の回転差を生じ方向を変えることが可能である. このバイアスは出力ではなく, サーボの入力にかかるため, 図のような構造がとられる. つまり, 層という単位は細かな抑制を行なうには大きすぎ, かといって抑制に適切な単位の層は細か過ぎて扱いにくい. これはモジュール性を著しくそこなっている. 内部にまで変更が及ぶので層の構造を後から大きく変えることはできないというものである.

第3章

実験に関する諸準備

本章では, 実験に先立って移動ロボットのタスクと実験環境を明らかにする. さらに, 実験に用いる移動ロボット Khepera の特性について実験に即して説明する. このことを通して, 本研究における実験をする上で,Khepera のどのような点が問題となるかを明らかにする.

3.1 タスクと実験環境

まず始めに, ロボットのタスクを設定する. 本研究における移動ロボットのタスクは,「壁を避けて徘徊しつつ, フィールド上の他の移動ロボットに対して追随行動を行うこと」である. タスクは, 追随行動を開始してからある程度の時間, 他の移動ロボットから離れずに追随できたときに達成されるものとする.

次に本研究における環境について説明する. 本研究における環境は,「テーブル上で周囲を発砲スチロール製の壁に囲まれたフィールド」である (図 3.1). また, 実験は通常の蛍光灯の光の下で行うものとする.

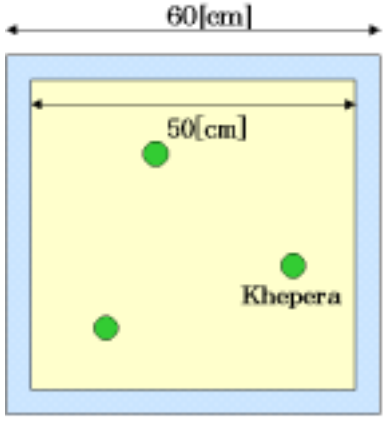

図 3.1: 実験環境 (上方から見た図)

移動ロボットの詳細については, 次節において説明する.

3.2　移動ロボットの概要と特性

3.2.1　小型移動ロボット Khepera

Khepera(K-Team S.A. 製) はスイス・ローザンヌ連邦工科大学, マイクロコンピュータ・インタフェース研究所 (EPFL/lami) で開発された研究開発用小型移動ロボットである.

直径が約 5.5cm で高さが 3cm 余り, インクリメンタルエンコーダ付きでそれぞれ独立して駆動する DC サーボモータを 2 個, 距離を測定できる赤外線近接センサと光の強度を測定する光センサが一体となったセンサを前方 6 箇所と後方 2 箇所の計 8 箇所装備している.

また, マイクロプロセッサとメモリを内蔵し (Motorola 68331:MHz, RAM:256Kbyte), 時分割による並列処理プログラムが実行できる. 各センサの値は AD 変換器を介して, 車輪の回転は左右それぞれの車軸に結合されたインクリメンタルエンコーダによって, プログラム上で利用できる. RS-232C インターフェースをもち, ホストコンピュータからのプログラムのダウンロードや, センサ情報などのアップロードに使用できる. NiCd バッテリも内蔵し, 電源供給線に頼らず自律的に行動できる.

図 3.2, 図 3.3 にそれぞれ, Khepera ロボットの構造, 概観とセンサ類の位置を示す[1].

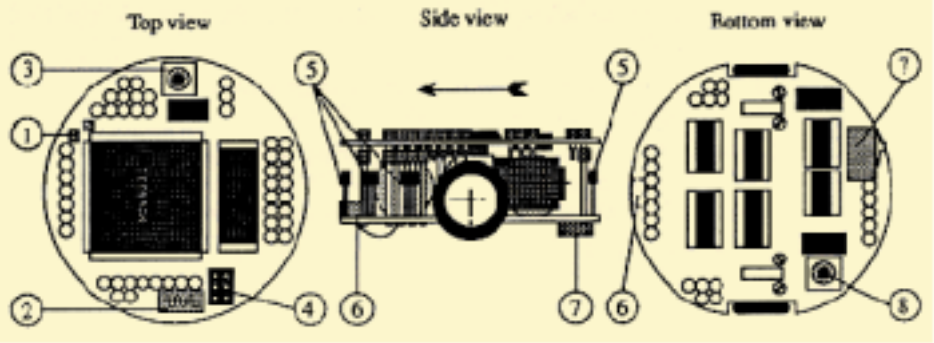

図 3.2: Khepera ロボットの構造

図 3.2 中の番号は以下に対応している.

1. LED
2. シリアル線コネクタ
3. リセットボタン
4. 動作モード選択用ジャンパ
5. 赤外線近接センサ
6. バッテリ充電コネクタ

[1] (K-Team 1998)

7. ON-OFF バッテリスイッチ
8. 第 2 リセットボタン

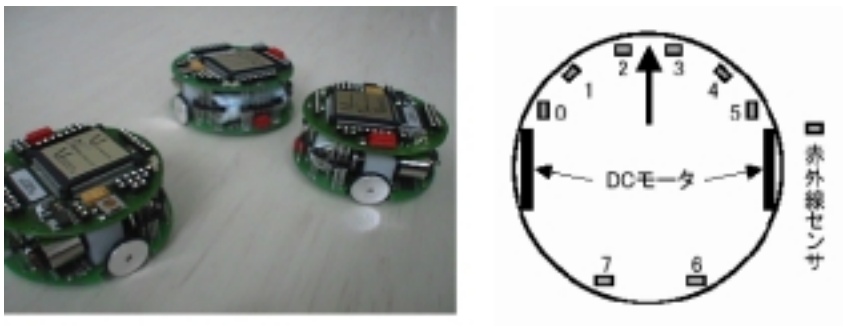

図 3.3: 概観 (左図) とセンサの位置 (右図)

次に,Khepera の制御方法について説明する. Khepera の制御方法には, ホストコンピュータでプログラムを実行する方法と, Khepera に搭載された CPU でプログラムを実行する方法の 2 つの方法がある. 前者の方法では, 決められたプロトコルを使用してシリアルケーブルでコンピュータと通信する.

後者の方法はダウンロードモードと呼ばれ, クロスコンパイラを用いてワークステーションなどで Khepera 実行用プログラムをコンパイルし Khepera の RAM に Download することによって,Khepera 内蔵の CPU, バッテリを使って行動することができる. この方法では, 完全にロボットが独立して外部からの制御なしに行動することができる. しかし, このためセンサなどのロボット情報のモニタリングもできなくなってしまうためアルゴリズム開発には向いていない.

3.2.2 近接センサの特性

Khepera の光センサは,KheperaAPI の命令によって 0-1023 の値を返す. 近接センサの値が大きいほど障害物との距離が近いことを示している.

図 3.4 3.5 3.6 は, 本研究で使用する障害物を並べて得られる十分な幅を持った平面に対して,Khepera を正面から近づけていったときの,Khepera から平面までの距離と近接センサの関係を示している. 「KHE-73X」というのは機体の製造番号で固有のものである. グラフの凡例におけるセンサ番号は, 図 3.3 右図のセンサ位置にふってある番号に対応している. グラフの各値は, それぞれ 50 回の計測の平均値を用いてプロットしたものである. これらは以下のグラフに対しても同様である. センサ 2,3 は平面に対して正面を向いている. また, センサ 1,5 の斜め方向に平面があることになる. 図から明らかなように, 近接センサは数センチメートル先の物体を認識するのがせいぜいである. また, 表には現れていないが, しばしば対象物がなくてもセンサが反応する場合があることもわかった.

左右のセンサで多少で多少のばらつきが生じているが, これはセンサの取り付け位置が初期の状態からずれていることに起因して生じる. 例えば,KHE-733 のセン

サ 2 の位置はセンサ 3 の位置より前方に 5[mm] ほどずれているためにこのような特性の違いが生じている.

ただ, 近接センサの値は複数を組み合わせて利用するので, 影響は軽微と推測される. これからの実験では機体を区別せずに扱うものとする

図に表記されていないセンサについては, この状況下では何も反応を示さなかった.

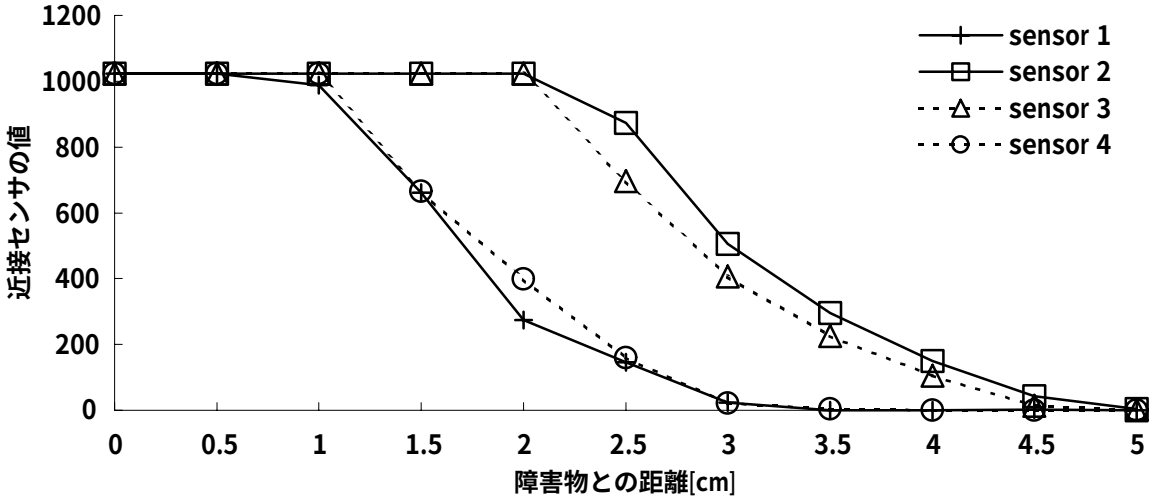

図 3.4: 障害物との距離に対する近接センサの値 KHE-731

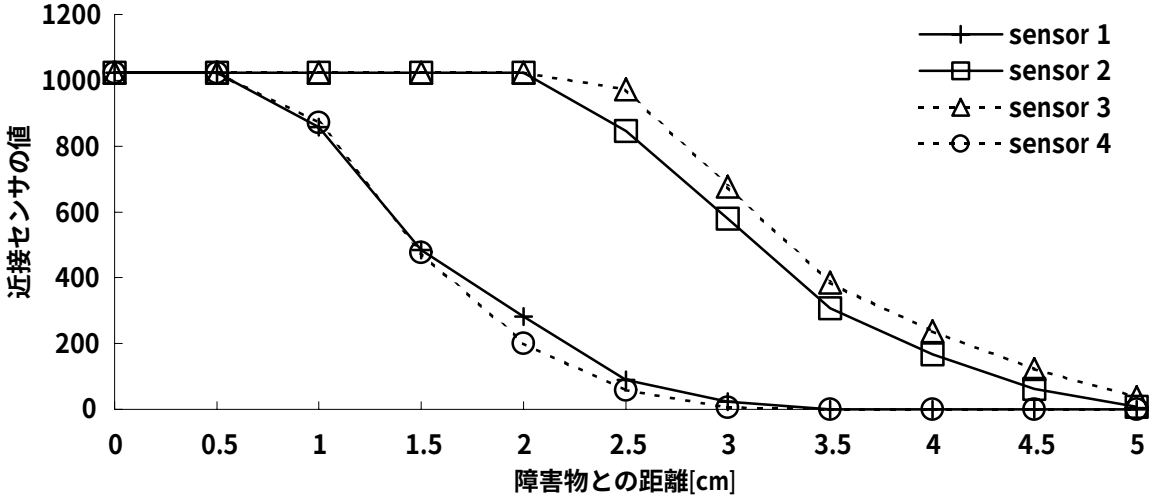

図 3.5: 障害物との距離に対する近接センサの値 KHE-732

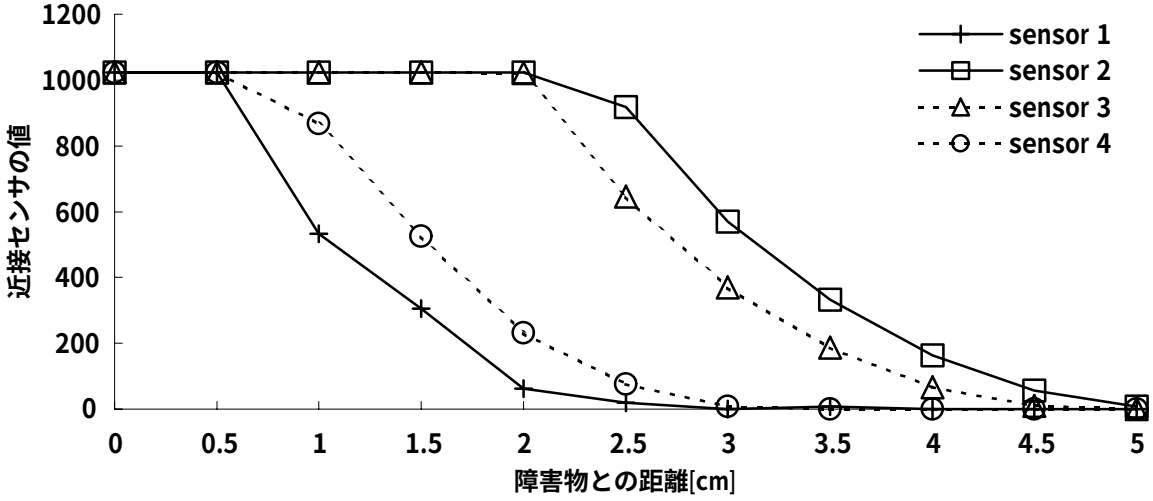

図 3.6: 障害物との距離に対する近接センサの値 KHE-733

3.2.3 移動対象物に対する近接センサの特性

本研究において追随行動を行う対象の Khepera は移動対象物として扱われる. そこで, 移動対象物に対する近接センサ特性を測定した.

これを見ると, 壁に対する近接センサの特性と比較して, 対 Khepera の近接センサ特性は大きく劣るものである. この理由は, 実機の場合は, 図を見るとわかるように, 側面から見た場合の疎な部品配置から考えて, 純粋な円筒形の対象物とは見なせないことに起因すると考えられる.

このことから, 側面に近接センサや突出した DC モーター後部を避ける形で覆うカバーをつける (図 3.8) ことによってセンサ感度の向上が見込まれるのではないだろうかと考えた. 以下はその結果である. これを見ると, カバーをつける前後で対移動対称物における近接センサの特性は明らかに変化してしまっている. 当初意図した結果とは違い, 逆に感度は低下してしまったことになる. 原因としては黒テープの使用により光が吸収される効果の方が大きかったことがが考えられる. だが, その他の実際的な問題として,Khepera 本体が異常に熱くなる場合がある, バッテリー充電コネクタ部も隠れることによって実験時に扱いにくくなるなどの不具合が見られたため, この処置は適当でないと考え, 実験時のカバーの使用は中止した.

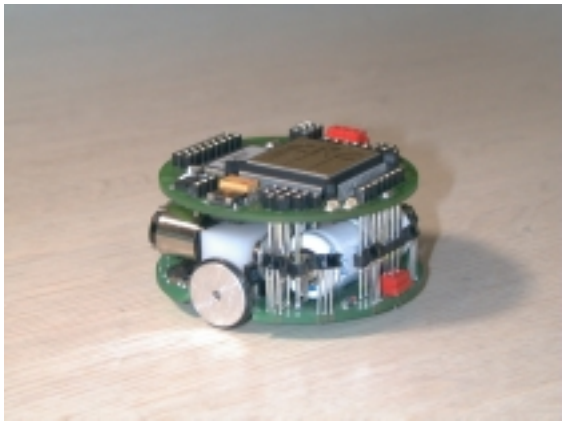

図 3.7: 通常状態

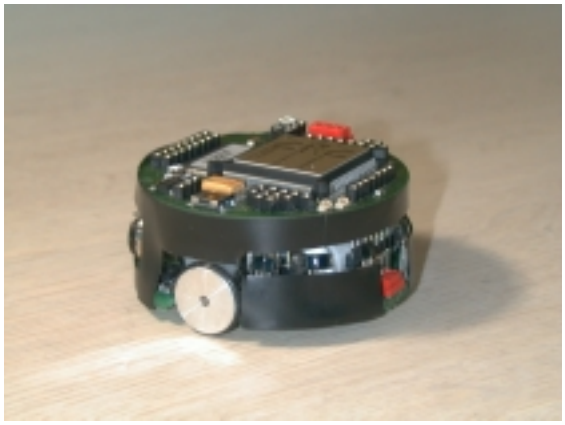

図 3.8: 機体周囲をカバーで覆った状態

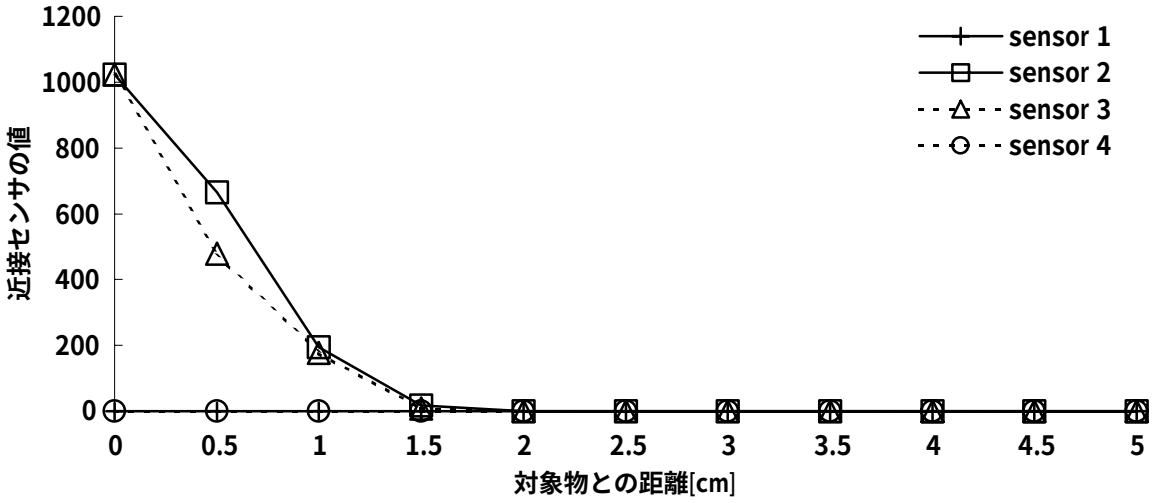

図 3.9: 移動対象物に対する近接センサの特性

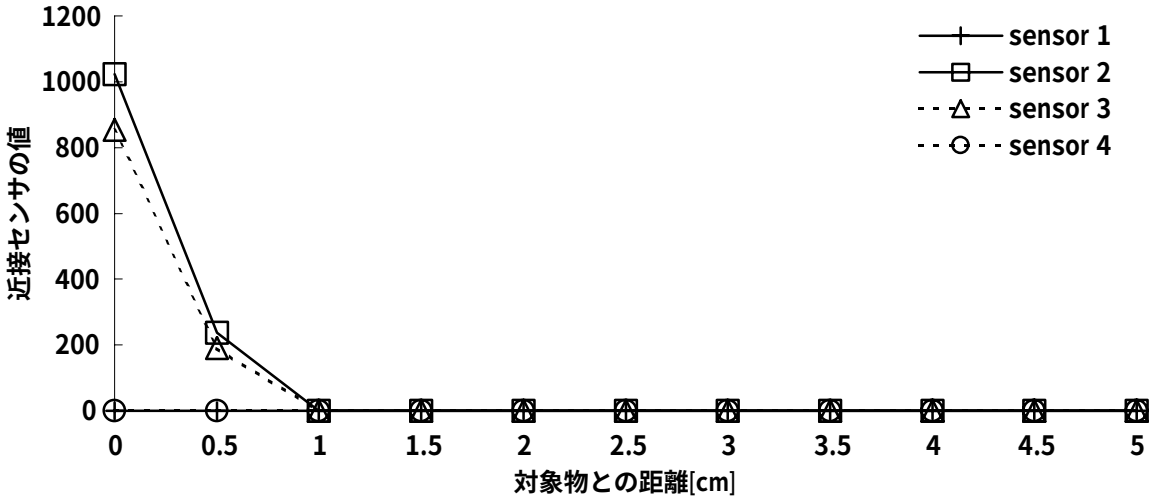

図 3.10: 移動対象物に対する近接センサの特性 (機体周囲を覆った状態)

3.2.4 Khepera シミュレータ Webots の概要

また, 実装するプログラムは,WEBOTS(Cyberboyics Ltd. 製) という Khepera 用のソフトウェア上で,C 言語と Khepera を実際に操作するための専用 API(Khepera API) を組み合わせて行なう[2].

本研究では Linux 版の WEBOTS(Ver.2.0) を Debian GNU/Linux 2.2R がインストールされた PC(Intel PentiumII 333MHz Memory 160MB) にインストールして使用した.

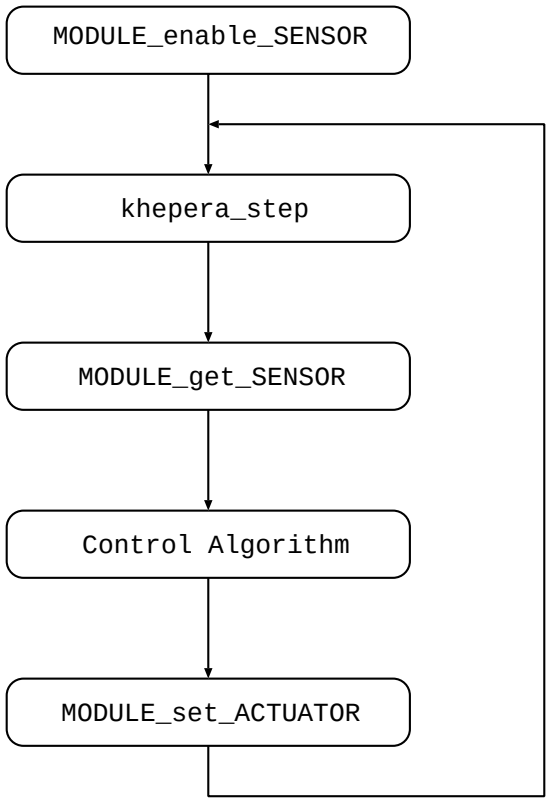

図 3.11: WEBOTS のプログラムの基本構造

このソフトウェアは, コンピュータ上に任意の環境を構築し, その環境内で実機を用いることなく様々なシミュレーションを行なうことが可能である. また, 実機を用いる際にも繁雑な操作の必要も無く, コンピュータと実機をつなぐだけでプログラムに沿った行動を実機に行なわせることができる. 複数台の Khepera を用いる際でも, プログラムはそれぞれに設定でき, 独立したプロセスとして動作させることができる. ただ,WEBOTS 内の環境は実世界に対して理想化されているので,WEBOTS のシミュレーションにおいてタスクを達成するということが, 実世界でのタスク達成を保証しない (丁子 2000). そのことを端的に示すために, シミュ

[2](Cyberbotics 1999b, 1999a)

レータ上における理想化されたセンサの特性について調べた結果を示す (図 3.13).

よって本研究においては,WEBOTS のシミュレーション機能はプログラムの基本的な部分の確認にのみ使用した.

図 3.11 は,WEBOTS を用いて作成する一般的なプログラムの基本構造を示したものである. プログラムは, 実行されると Khepera を初期化し, 各種センサを使用可能な状態にする (Khepera enable SENSOR). どの種類, どの場所のセンサを使用可能にするかは任意に設定できる. また, 図中の **SENSOR** は,**proximity**,**light**,**speed** と置き換えることができ, それぞれ近接センサ, 光センサ, インクリメンタルエンコーダを示す. 例えば,**MODULE enable proximity** は, 近接センサを使用可能にすることを指す. その後, センサからの情報入手 **MODULE get SENSOR**, 制御アルゴリズムによる処理 **(Control Algorithm)**, モータへの命令 **MODULE set ACTUATOR** と続く. **ACTUATOR** は,**speed** と置き換えて **MODULE set speed** とすることでモータに速度を指示する. そして,**khepera step** という命令によって処理の 1 単位時間が終了する. この一連の流れを繰り返すことによって khepera は制御される. 実際には,**SENSOR** や **ACTUATOR** に対して置き換えられるものが **Khepera API** において他にもいくつか与えられているが, 本研究では使用しないので省略する. また, 本研究において設計するのは, 図の **Control Algoritm** にあたる部分である.

WEBOTS では事前にホストコンピュータ側でクロスコンパイル環境を構築しておくことにより, シミュレータからこれを利用して **Control Algoritm** を実機にダウンロードして (図 3.12) 自律的に行動させることが可能である. この環境は WEBOTS を通常インストールした状態では構築されていないので, 本研究を行うにあたり,Linux 側に Khepera に搭載されている Motorola 68331 MPU の GCC クロスコンパイル環境を構築し,WEBOTS 側から利用した.

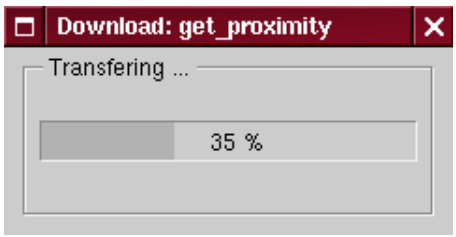

図 3.12: Controller プログラムを実機にダウンロード中

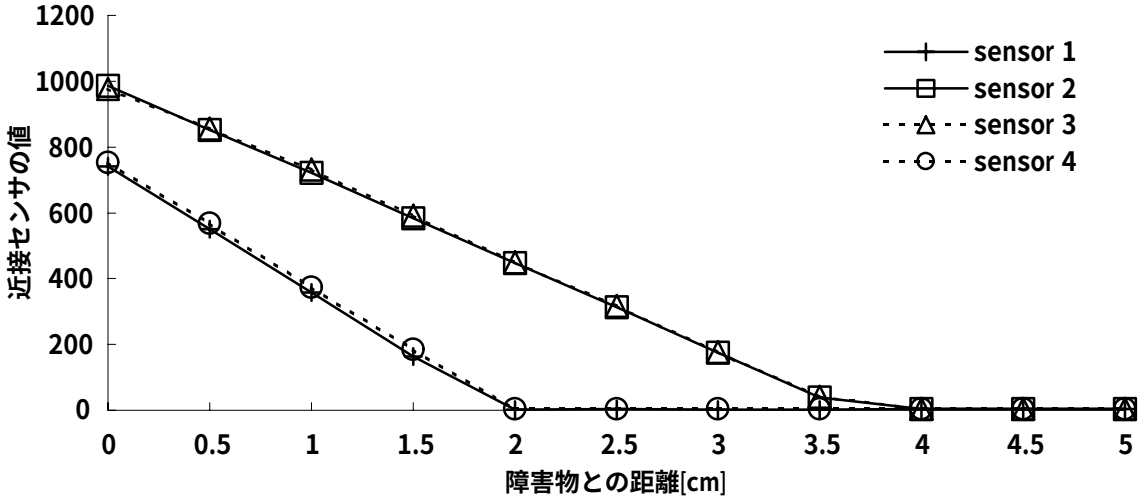

図 3.13: 障害物との距離に対する近接センサの値 (シミュレータ)

シミュレータの近接センサの値を前述の測定結果のグラフの概形から以下のように 1 次の線形関数で近似して補正をかける関数を作成した. これによりシミュレータにおける機体の挙動が実機に近似できることを期待したが, 考えた通りの挙動にはならず使用しなかった. 参考のため示す.

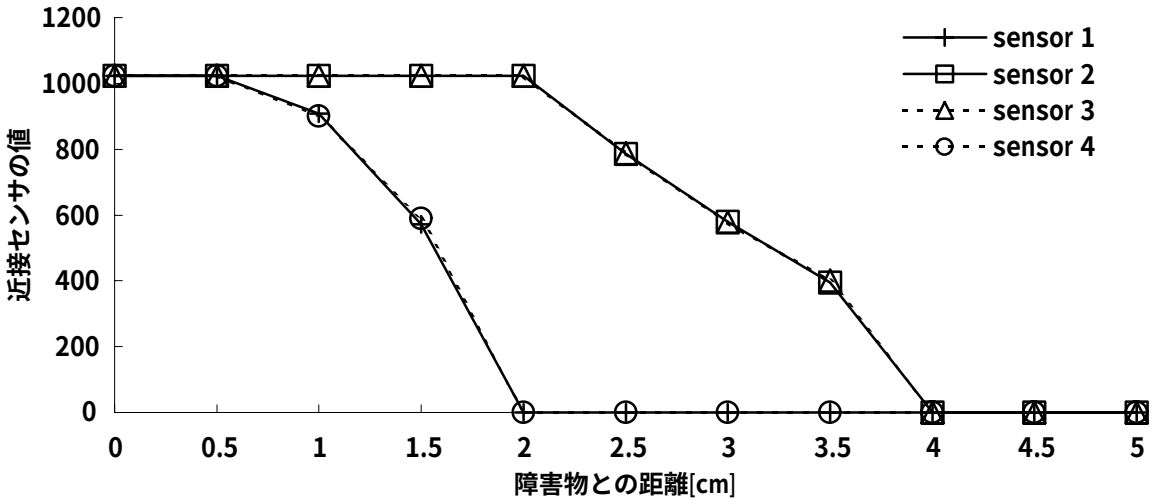

図 3.14: 障害物との距離に対する近接センサの値 (補正後)

3.3 Khepera(自律移動ロボット) を使った研究の紹介

3.3.1 Floreano の研究の紹介

本研究の実験で用いている研究開発用ロボット Khepera の開発者の一人である Floreano の研究を 1 つ紹介する.(Floreano,Nolfi and Mondada 1998) この研究では Khepera とホストコンピュータを有線のシリアル接続で繋いで制御し, 捕食者-被捕食者間競争の実験を行っている (黒山 1998), (横井, 石田 1998),

ここでは被食者は捕食者の 2 倍のスピードで移動する (逃げる) ことができ, 捕食者はこれを 2 次元視覚センサで検知しようとする. また, 両者は障害物回避のための近接センサを装備している. これらの外部センサ情報を, ロボットの行動をコントロールする再帰的パーセプトロンに入力し, この強度パラメータを競争によって共進化させる.

彼のこの研究の手法はニューラルネットワークと GA を組み合わせたもので図 3.15 に示すように, 染色体として Khepera ロボットのセンサとモーターを直接結ぶニューラルネットワークの染色体をいくつも用意し, 各染色体ごとに Khepera を一定時間動作させ, それぞれの染色体で起こった行動のパフォーマンスをホストコンピュータで計算している. そのパフォーマンスを適応度として GA の手法に基づき, 選択, 交叉, 突然変異を行い, この操作を繰り返すことによって, 染色体の最適化つまりロボットの学習を行っている.

100 世代の進化実験の結果,10 数世代ごとに捕食者と被食者の優勢関係が交替し, それに伴う戦略の進化が確認された.

第 20 世代頃, 捕食者は外壁を障害物として認識しながら, 壁伝い行動を獲得した. それに対して, 補食者は,2 次元視覚センサから捕食者を確認し, これに近付くという戦略で優位に立った.

第 70 世代頃には捕食者は, 一定の場所で旋回しつづけ, 捕食者の接近を IR センサで感知するとすばやく後退するという戦略で高い適応度を得た. さらに, 第 90 世代頃には最高速で外周を伝う被食者に対し, 捕食者が待ち伏せをするという戦略まで現れた.

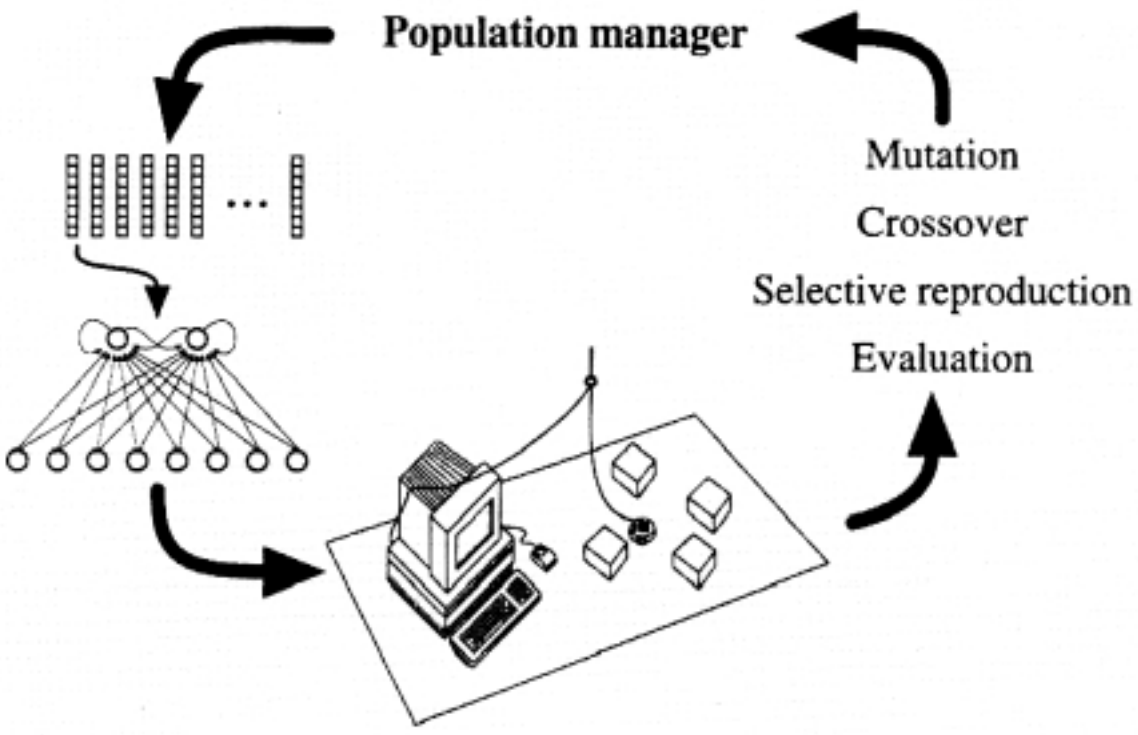

図 3.15: floreano による Khepera の実験

第4章

実装

本章では, 服属アーキテクチャを適用して構成されるロボットの制御システムの構造を説明する. 最初に, 設計を進めていく上での基本的な方針を明らかにする.

本研究における実験環境では, 移動ロボットは事前に実環境について正確な情報をもたないことばかりでなく, 自身のセンサの性能も充分ではないことが前章で示された.

ところで, センサからの環境情報が不確かな場合には, 環境の位置に関する数値情報を捨てた定性表現で環境を表現することが有効であることが示されている (魏世杰, 大澤幸生, 八木康史, 谷内田正彦 1998). そこで, 本研究においても, この文献に沿って環境の定性表現を用いる. それをもとにし, さらに服属アーキテクチャを適用して構成される移動ロボットの制御システムについてここで説明する.

4.1 設計の基本方針

本研究における実験環境では, 移動ロボットは対象物の存在や形・大きさを知らない. さらに, 移動ロボットではセンサ有効範囲に対象物が存在しない場合にも, 存在すると判断しかねないことが前章で明らかになった.

本研究では設計方針として, 設計者が追随行動における基本的な状況をトップダウンに想定し, その状況に対して適切な行動を取れるようにモジュールを構成していく. 具体的には, 前章で示されたセンサの能力を考慮しつつ, 追随行動を以下の 2 段階のタスクに分割した.

Step1 移動ロボットに装備されているセンサをセンサ 0 から 5 までの 6 個とセンサ 6 から 7 までの 2 個に分ける. これはセンサを前方と後方にまとめていることに相当する. この 2 組のセンサの反応の組み合わせを見て, 今, 自機の前後の状態を調べ, 次の行動を判断する.

Step2 Step1 において, 前方がセンサの反応がなく後方に反応があれば, 現在, 前方

が空いていて, 後方から他のロボットに追随されているものと判断する. 無論そうではなく, 後ろが壁である場合もあり得るがこの状態でロック状態から抜け出せないことはないので, ここではそのような場合は考えないことにする. ここで, 後方からのロボットが追随しやすいように速度を落しつつ徘徊を続けるそうでなければ速度を上げつつ他の対象を求めて徘徊を続ける. 追随行動は原理的にはこのように非常に簡単なタスクの分割で実現できる.

しかし, このままでは壁に向かった際に離れられなくなるので, 壁を判断するモジュールがこれと並列に絶えずセンサの値を監視している.

壁の見分け方は (実装者の) 経験から前方のセンサのうちセンサ 1, センサ 4 に反応があり, センサ 2, センサ 3 がある値以上のとき壁に向かっている可能性が高いことがわかっている. このとき壁に向かっているものと判断し反転して壁を避ける. もしそうでなければ,Step2 と同様に速度を上げつつ他の対象を求めて徘徊を続ける.

4.2 レベル 0 徘徊モジュール

最下層の目的はロボットの基本的な動きを制御することである. 目標となる対象物を発見できず, また壁にも向かっていないとき, 移動ロボットは一定速度で移動する. このときの移動ロボットの動きはこれはいわゆるブライテンベルクのビークル (V. ブライテンベルク 1987) におけるブライテンベルク係数で与えられるセンサとアクチュエータの接続の強度によって規定される.

Step1 で述べたように前方にセンサの反応がなく後方に反応があれば現在, 前方が空いていて, 後方から他のロボットに追随されているものと判断する.

4.3 レベル 1 衝突回避モジュール

このシステムの最上位モジュールであるこのモジュールは衝突回避すべき状況かどうかの状況の判断を行いその結果に応じて適切に衝突回避を行う.

4.3.1 ニューラルネットワークによる学習を用いたモジュール

前述のシステムでは, 回避を行うロボットのレベル 1 モジュールにおいて実際に衝突回避を行うかどうか判断する部分は実装者の経験によって実装されていた.

ここで比較の対象として, この部分に 3 層フィードフォワード型のニューラルネットワークを適用したモジュールを実装した. そして静的な環境から得られた教師信号をこのネットワークに与え, 誤差逆伝搬学習法を適用し, 壁と Khepera を識別させた. ここでは, 壁の場合には 0, ケペラの場合には 1 を出力するような教師信号である. ネットワークの入力は 8, 中間層は 6, 出力は 1 であり, シグモイド関数には

bipolar 型のものを使用した. 出力値が 0.5 以上の時,Khepera と認識する. 充分学習を行わせるため,10 万回学習を繰りかえした.

実際にはネットワークに対する学習はホストコンピュータで行い, 学習によって得られた重み係数ベクトルを, 実機にこのシステムをダウンロードするときにネットワークの重みとして取り込むようにした.

4.3.2 フィードフォワード型ニューラルネットワーク

このモジュールで採用した誤差逆伝搬学習法のアルゴリズムを示す (熊沢 1998).

誤差逆伝搬法は逐次更新学習法の 1 つであって, 訓練データが与えられる都度, 結線重みを修正する. 具体的な修正方法を図 4.1 のネットワークを例に示す.

実現したい入出力関係の訓練データが複数の入出力対として与えられる. 訓練データが L 個あり, その l 番目は, 入力 $(a_1^{(l)}, a_2^{(l)}, \cdots, a_N^{(l)})$ に対して, 出力 $(t_1^{(l)}, t_2^{(l)}, \cdots, t_M^{(l)})$ を要求するものとする.

以下の **Step 1** では, 訓練データを選択し,**Step 2**,**Step 3** では訓練データに対する誤差評価尺度を小さくするようにパラメータを修正する. 以上を全訓練データの誤差評価尺度が十分小さくなるまで繰り返し行う.

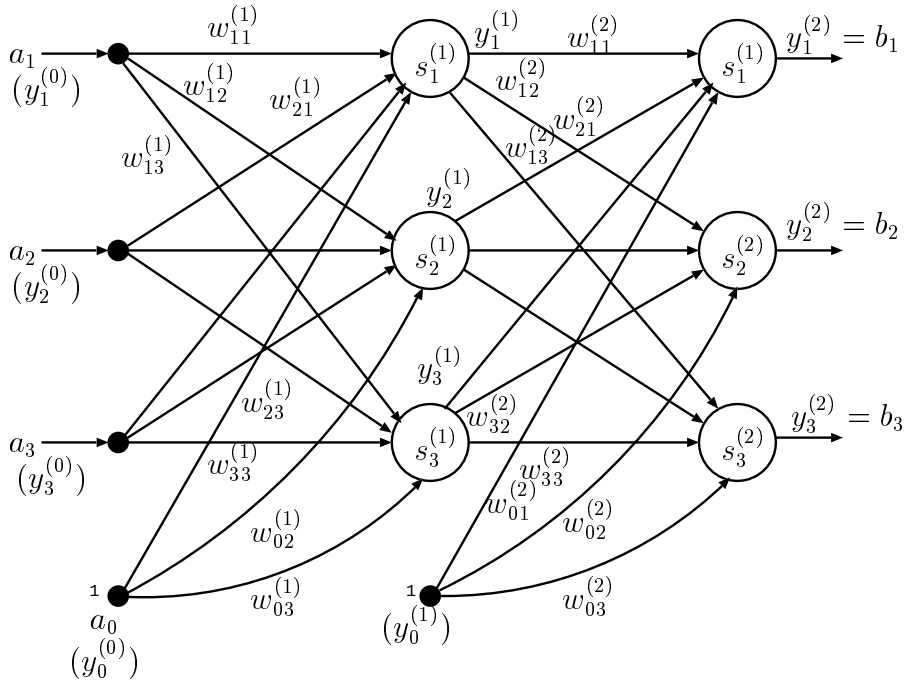

図 4.1: 誤差逆伝搬法で学習するフィードフォワード型ニューラルネットの例

Step1 訓練データの選択

最初の手続きとして, ここで l 番目の訓練データを選択し, このデータに対して, 以下の 2 つのステップを実行する. これらの 2 つのステップの実行の間, 訓練データは固定するので, 訓練データの識別に用いた変数右型の (l) は必要ない. 以後, これを省略し, 入力 $(a_1^{(l)}, a_2^{(l)}, \cdots, a_N^{(l)})$ を $(a_1, a_2, \cdots, a_N)$ と書く. その代わり, 変数右肩の添字を層を識別するために用いる. 変数右肩の添字 (1) は, 第 1 層の変数であること, また添字 (2) は第 2 層の変数であることを示す.

Step2 出力の計算

図 4.1 には, 入力層, 中間層, 出力層の 3 層からなるフィードフォーワード型ニューラルネットが示してある.

入力層は黒ドットで示されるノード (信号の中継点) からなる.

他の層は白丸で表される演算素子を含む. この演算素子はニューロンの工学的モデルであり, 誤差逆伝搬法では, 各種のニューロンのモデルの中でも, 決定的アナログモデルが用いられる.

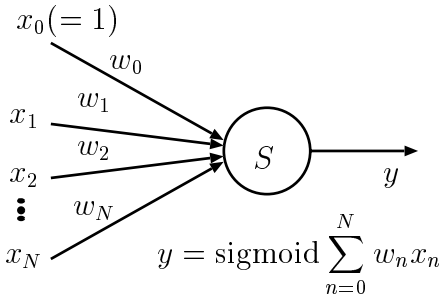

図 4.2: 出力計算モードにおける素子の動作

素子の動作を図 4.2 に示す.

ここで, 素子への入力を $x_0, x_1, x_2 \ldots x_N$, 出力を y, 結線重みを $w_0, w_1, w_2 \ldots w_N$ とし, しきい値は結線重みの 1 つ (w_0) として表されているものとする. またシグモイド関数のゲインを α で表す. 各層に常に 1 の値を出力するノードがあって, その出力が w_0 の結合重みで各素子へ入力する. 記法の一貫性を保つため, このノードの出力値を入力層で a_0, 第 1 層で $y_0^{(1)}$ と記述する. 図 4.2 の素子の動作を次のように定式化する.

$$s = \sum_{n=0}^{N} w_n x_n$$

$$y = \text{sigmoid}(s)$$

このように動作する素子を組み合わせて, 図のネットワークを構成し, 入力層の i 番目のノードに入力 a_i を加える, 入力ノードは $i = 1, 2, \cdots, N$ 個あるものとし, これらの全てに同時に入力を加える.

なお, ネットワーク中の結線の重み全てに, 初期値として乱数で発生したでたらめな数を割り当てておく. ここで注意すべきことは, 初期値を全て 0 とすると, 対象性のために勾配値が 0 となり, 後に示す学習手続きが進行しないことである. 初期値を乱数等で与えるのは, 対象性を壊すためである.

入力ノードに入力を与えた後,j 番目の中間素子は, 入力 a_j に結線の重み $w_{ij}^{(1)}$ をかけて $i = 1, 2, \cdots, N$ について総和を求め, まず,

$$s_j^{(1)} = \sum_{i=0}^{N} w_{ij^{(1)}} a_i$$

$$y_j^{(1)} = \mathrm{sigmoid}(s_j^{(1)})$$

を得る. これを重み付け総和 (Weighted Sum) と呼ぶ.

次に $s_j^{(1)}$ をシグモイド関数に代入し, 素子の出力

$$y_j^{(1)} = \mathrm{sigmoid}(s_j^{(1)})$$

を得る. 中間素子が $j = 1, 2, \cdots, K$ の K 個あるとき, これらの全てについて上述の手続きにより出力を求める.

こうして, 中間層素子の出力を求めた後,j 番目の出力層素子は, 中間層素子の出力 $y_i^{(1)}, i = 1, 2, \cdots, K$ をその入力として受けとり, これに結線の「重み」$w_{ij}^{(2)}$ をかけて総和を求め, まず, 重み付け総和

$$s_j^{(2)} = \sum_{i=0}^{K} w_{ij^{(2)}} y_i^{(1)}$$

を得る. 続いて,$s_j^{(2)}$ をシグモイド関数に代入し, 素子の出力

$$y_j^{(2)} = \mathrm{sigmoid}(s_j^{(2)})$$

を得る. 出力層素子が $j = 1, 2, \cdots, M$ の M 個あるとき, これらの全てについて上述の手続きにより出力を求める.

$y_j^{(2)}$ は, 出力層素子の出力であるが, これがネットワークの出力 b_j になる. b_j は, アルゴリズムの手続きを記述する際には,$y_j^{(2)}$ と記述した方が便利である. 同様に, ネットワークへの入力 a_i も,$y_j^{(0)}$ と書く方が, 記法を統一できて良い. 以下状況に応じて両記法を使い分ける.

以上に述べた手続きでネットワークの出力 $b_j, j = 1, 2, \cdots, , M$ を計算した後, これらを学習目標の出力と比較する. ここで, 入力 $a_i, j = 1, 2, \cdots, N$ に対して, 本来望む目標出力を $t_j, j = 1, 2, \cdots, M$ と記すことにする. ここで,t は目標を意味する英語 *target* の t である. 最初は, 重みをランダムに与えたので, 当然なが

ら,$b_j, j = 1, 2, \cdots, M$ は $t_j, j = 1, 2, \cdots, M$ と一致しない. そこで, 勾配法の原理により, 誤差評価尺度

$$E = \sum_{i=1}^{M} |b_i - t_i|^2$$

を小さくするように, ネットワーク内部の結線の重みを修正する. なお, しきい値は, 先に述べたように, 結線の 1 つとして表されている.

以上の手続きの中で求めた各素子の出力値 $(y_j(l))$ は, 次の **Step 3** で利用するので, 各素子の中に記憶しておく.

Step3 結合重みの修正

結線の重みの修正手続きを次に示す.

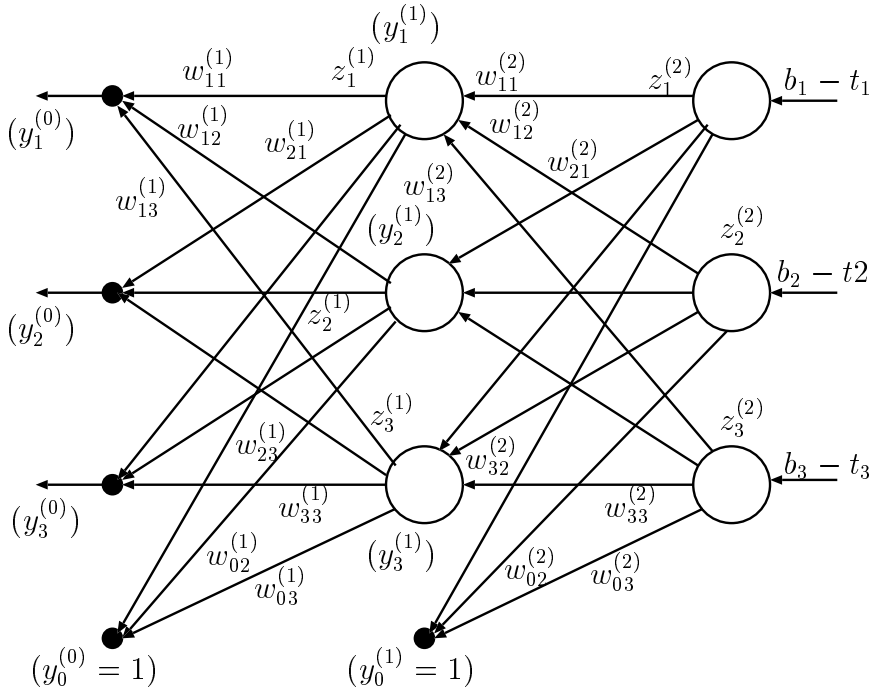

図 4.3: 誤差逆伝搬モードの信号の流れ

まず, 結線の重み修正時 (誤差逆伝搬モードと呼ぶ) のネットワークの信号の流れを図 4.3 のように定める. ノード数, 素子数, 素子間の結線構造, 及び結線の重みは, 出力の計算時 (出力計算モードと呼ぶ) に用いた図 4.1 と変わらないが, 素子の機能と信号の流れる方向が異なることに注意が必要である.

誤差逆伝搬モードにおける素子機能を, 図 4.4 に示す. 素子内部にはメモリーがあり,Step2 で求めた自分の出力 y(以後簡単のため $y_j(l)$ の添字を省略してこのように表示する) を保持しておく. 図中 () 内の値は,**Step 2** で求めた値が保存されてい

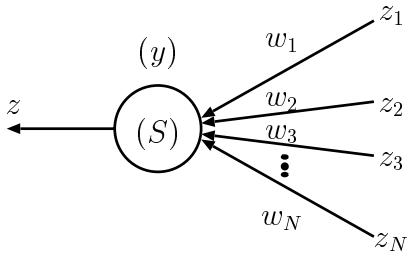

図 4.4: 誤差逆伝搬モードの素子機能

ることを意味する. 素子の右側から与えられる入力 $z_1, z_2, \cdots, z_N$ に, 素子の右側の結線の重み $w_0, w_1, w_2, \cdots, w_N$ を掛け, 重み付け総和

$$u = \sum_{n=1}^{N} w_n z_n$$

を求めた後, メモリーに保持されている $y = \mathrm{sigmoid}(s)$ を用いて $\mathrm{sigmoid}'(s)$ を計算し, これに u を掛け,

$$z = \mathrm{sigmoid}'(s)u$$

を出力する. これは左側の層の素子への入力となる.

以上のように動作する素子を図のように接続する. ここで図 4.1 と図 4.3 は全く同様の結線構造と結線の重みを持つことに再度注意が必要である. 結線を流れる信号の方向だけが逆となっている. 図 4.3 の右から,**Step 2** で求めたネットワークの出力 b_i と目標出力 t_i との誤差 $b_i - t_i$ を加える. この入力に基づき各層で図 4.4 に示した素子機能が働き, 素子の出力値 $z_i^{(l)}$ が右側の層から次々と定まる. こうして, 誤差逆伝搬モードで各層の素子出力値 $z_i(l)$ を求めた後, これと出力計算モード (**Step 2**) で求めた各層の素子出力 $z_j^{(l)}$ を用いて, 次式で結線の重みを修正する.

$$w_{ij}^{(l+1)} = w_{ij}^{(l+1)} - \varepsilon y_i^{(l)} z_j^{(l+1)}$$

$w_{ij}^{i}(l+1)$ は, 第 l 層の第 i 素子と第 l+1 層の第 j 素子を結ぶ結線の重みであるが, その修正量が自分の接続している左側の層の素子の出力計算モードにおける出力値 $y_i^{(l)}$ と, やはり自分が接続している右側の層の素子の誤差逆伝搬モードにおける素子出力値 $z_j(l+1)$ の積によって定まることに注意が必要である.

第5章

実機による実験

5.1 複数自律移動ロボットによる実験

第 4 章で述べたシステムを Khepera に実装し, 実験を行った. 実験の手順について説明する.

3 台の Khepera に 4 章で実装したシステムをダウンロードし, フィールド上にランダムに配置する. 実験は 1 ラウンドを 5 分で区切り, 各モジュールについて 20 セット, 計 40 セット行なった.

5 分という長さは khepera が追随行動を行なうのに十分な長さであると考えられる.

実験の様子は MPEG カメラ (日立製 MP-EG10) で撮影し,10 セット分の実験終了ごとに,MPEG1 形式の動画ファイルをノート PC の PC カードスロットとネットワークを経由して PC(富士通 FMV6400TX2) に転送し, 目視で解析を行なった.

どのような行動を追随行動と見なして判定したかを以下に示す.

- 互いに接近した Khepera のどちらか先頭になりある程度の時間追随行動を行う.
- 追随行動を行っている状態で, 先導者が壁に跳ね返っても元の位置関係が維持された場合は引き続き 1 回分とし, 新たな追随行動回数の分にカウントしない.
- 3 台の機体が列になった場合は, 追随行動を 2 回分とカウントする.

以下図 5.1,5.2 にその結果を示す.

5.1.1 観察された追随行動

図 5.3 では, 既に 2 台連結した Khepera の先導者が 3 台目の機体 (=khepera) を近接センサの検知範囲に捉えて, その機体を新たな先導者として追随し, 結果として 3 台が連なって追随行動を行う様子を示している.

また, 図 5.4 は,Khepera が先導者に対して追随行動を行っている状態で先導者が壁に当たった後, 先導者がそれまで追随を行っていた後方の機体に入れ替わり, 今まで先導者だった機体が状況に適応して新たに決まった先導者に対し, 引き続き追随行動を行っていく様子を示している.

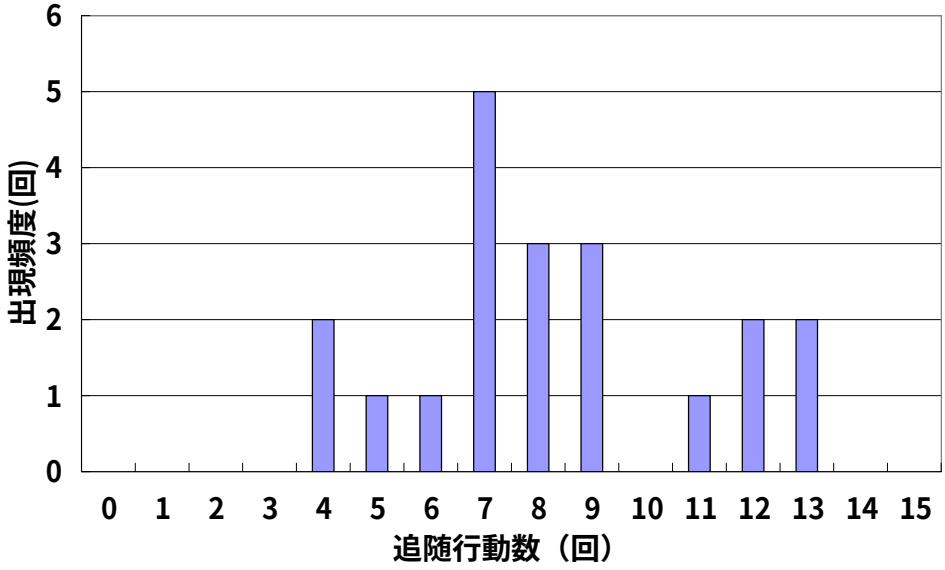

図 5.1: 追随行動数分布 (服属アーキテクチャのみで構成したモジュール)

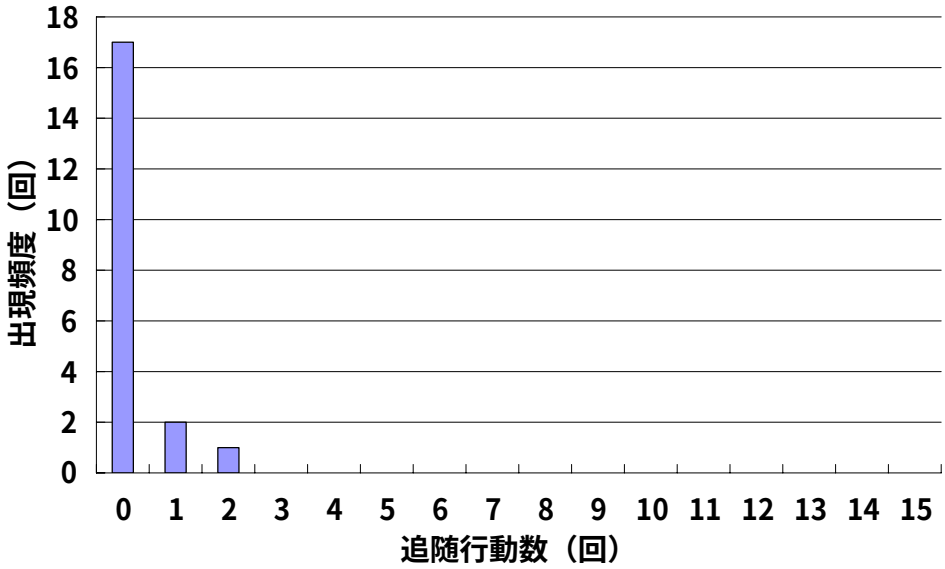

図 5.2: 追随行動数分布 (識別機構をニューラルネットワークに置き換えたモジュール)

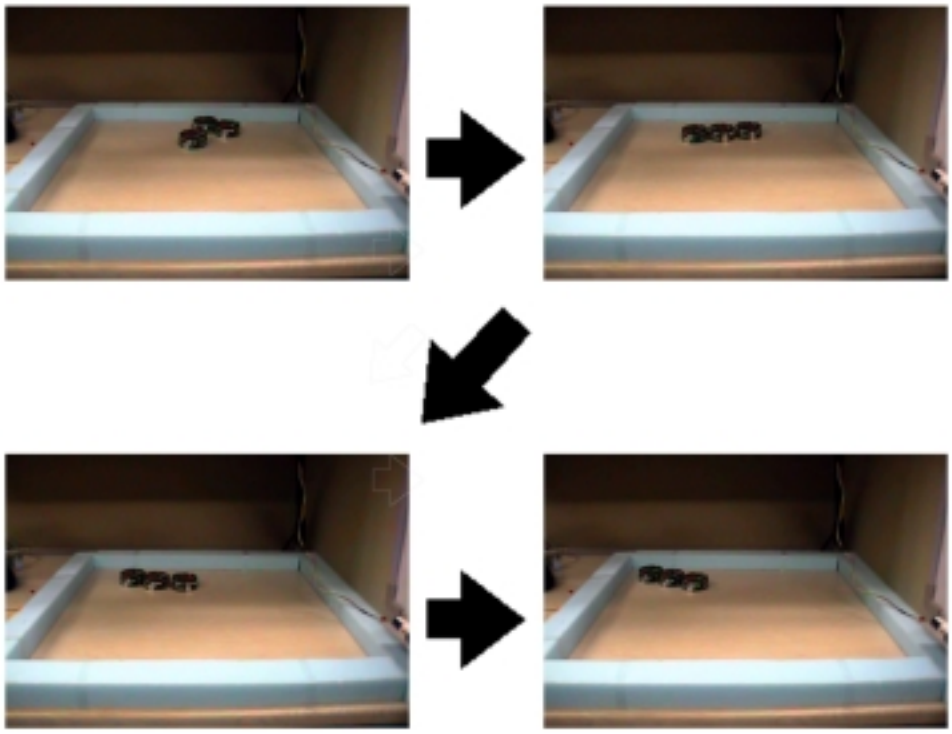

図 5.3: 3 台の機体による追随行動

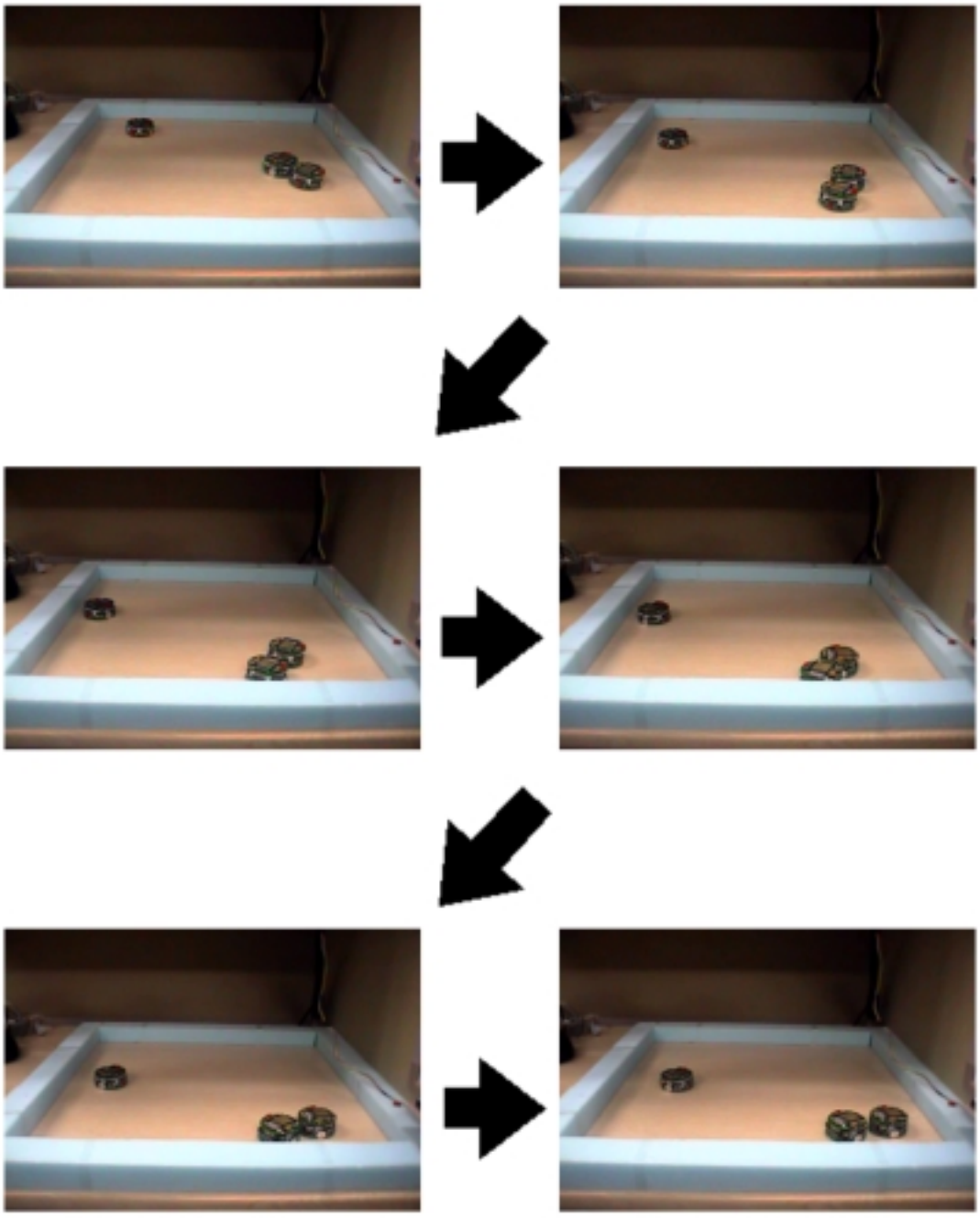

図 5.4: 先頭の入れ替わり

第6章

考察

6.1 考察

結果からは, 衝突回避のための識別部分には全体としてサブサンプション構造を持つモジュールを組み込んだ移動ロボットの方が, ニューラルネットワークによる識別機構を持つモジュールよりも相対的に成績が良くなることが確認された.

このことにより服属アーキテクチャの有用性が, 改めて実証されたといえる.

だが, 第 3 章で述べたように服属アーキテクチャの適用にも問題がないとは言えない. 今回観察された追随行動のバリエーションはどれも現実世界では普通に起こり得るようなありふれたものだが, 今回の研究では相当の試行回数を重ねなければ観察できなかった.

また, 実験結果を解析する際に一定時間で追随行動が何回起っているかを調べる他に, 一定時間で追随行動が総計でどれくらいの時間起こっているかを調べられれば, もう少し違った観点からの考察ができたものと考えられる. 実際の実験では動画データを解析し, そのような数値を得ることは困難であったが, 機体の軌跡を抽出するなどして, 何らかの方法でこの数値を得ることは不可能ではないと考えられる.

一方, ニューラルネットワークによる識別機構を組み込んだ機体の成績が悪かったのにはいくつか原因があると考えられるが, その中でも一番の理由と考えられるのは, 今回適用したニューラルネットが静的な値から環境に対してマッピングを行っている点だろう.

これについても, ニューラルネットの中間層, 学習係数などを変えて実験を行い最適なネットワーク構造が得られれば, 今よりもパフォーマンスが上がりロバストなシステムになる可能性は依然として残っている.

ところで, (中村, 石黒, 内川,Rolf 2000) も指摘されていることだが, 静的なセンサ情報から表象 (出力) へのマッピングでは, 貧弱な能力しか実現できないのに対し, 識別過程に進化的計算手法などのロボット自身の動きをも反映したアプローチでは, システムと環境との相互作用を利用することによって, 動的な環境の下でもロバストな識別能力を獲得できると考えられる

今後は環境との相互のインタラクションをどのように学習に取り入れていくかが課題となる.

謝辞

本研究を行うにあたり, 御指導していただきました櫻井彰人教授，および多くの助言をして下さった荒木修助手に感謝致します．また協力していただいた櫻井研究室の皆様に感謝致します。

参考文献

V. ブライテンベルク (1987). 模型は心を持ちうるか 人工知能・認知科学・脳生理学の焦点. 哲学書房. 加地大介 訳.

丁子英樹 (2000). “服属アーキテクチャを適用した移動ロボットの衝突回避行動に関する研究.” 修士論文, 北陸先端科学技術大学院大学知識科学研究科.

Cyberbotics (1999a). *WEBOTS 2.0 Reference Manual*.

Cyberbotics (1999b). *WEBOTS 2.0 User Guide*.

Floreano, D.,Nolfi, S.,and Mondada, F. (1998). “Co-Evolutionary Robotics:From Theory to Practice.” In R., P. (Ed.), *From Animals to Animats IV*. MIT Press.

伊藤宏司編 (2000). 知の創発. NTT 出版株式会社.

K-Team (1998). *Khepera USER MANUAL* (Fifth edition).

熊沢逸夫 (1998). 電子情報工学シリーズ 学習とニューラルネットワーク. 森北出版株式会社.

黒山和宏 (1998). “自律移動ロボットの行動学習に関する研究.” 修士論文, 神戸大学工学部機械工学科.

中村弘, 石黒章夫, 内川嘉樹,RolfPfeifer (2000). “環境との相互作用を用いた自律移動ロボットの識別能力の実現.” 日本ロボット学会誌, **Vol.18** (7), 963–971.

中島秀之 (1999). “認知ロボットのアーキテクチャ.” 日本ロボット学会誌, **Vol.17** (1), 16–19.

新井 民夫太田 順 (1994). “群秩序生成可能な追従戦略を用いた複数移動ロボット系の動作計画.” 日本ロボット学会誌, **Vol.12** (4), 603–608.

R.A.Brooks (1986). “A Robust Layered Control System For A Mobile Robot.” *IEEE Journal Robotics and Automation*, **RA-2**, 14–23.

魏世杰, 大澤幸生, 八木康史, 谷内田正彦 (1998). “自律走行車の身体に基づく環境表現による定性的移動計画.” 人工知能学会誌, **13** (5), 803–810.

山田誠二 (1997). 適応エージェント. 共立出版株式会社.

横井浩史, 石田崇 (1998). “集団行動の創発とファジィ理論.” 日本ファジィ学会誌, **10** (4), 637–646.

吉田典晃 (1993). “サブサンプションアーキテクチャを用いたロボットの制御.” 日本ロボット学会誌, **Vol.11** (8), 1118–1123.