自律型水中ロボットにおける
自己組織的行動獲得システムの開発

指導教官　石井和男

西田　周平

目次

図目次

表目次

第1章

序論

1.1 はじめに

1.1.1 ロボットの知能化

ロボット元年といわれている 1980 年から現在まで，各分野において様々なロボットが研究，開発されている．多くのロボットは工場，核廃棄物処理場，災害現場，宇宙，深海などの極限環境で活躍している，あるいは，さらなる活躍が期待されている．一方で，人の生活をサポートする警備や福祉，医療ロボットの開発も行われており，人との共生をテーマとしてサービス分野への進出が期待されている．人間の目に見えないところで働いていたロボットが人間の目に見える環境へと活躍の場を広げてきている．

最近ではペットロボット，二足歩行ロボットキットなどが一般向けに販売されるようなってきた．ロボットによるサッカーの大会“RoboCup”や二足歩行ロボットの大会“Robo-One”などの影響は大きい．今後もロボットの活躍の場は広がり，ロボットを利用した新しいアプリケーションが出てくるであろう．

他方では，ロボットを使って，人間の知能が発達する過程と原理の解明を目指す研究が行われている[1]．ロボットの脳にあたるコンピュータの発展はいまだに衰えを知らず高速化は続いている．ロボット元年から見れば信じられないほど処理能力は向上している．しかし，演算処理の能力が上がったからといって，人間のように学習や訓練によって今まで解決できなかった問題が解けるようになったりはしない．

1983 年のロボット学会発足当時に知能ロボットはシーケンシャルなものではなくセンサベースなものであり，さらに必要な要素として，作業計画，環境の理解，知識の獲得と利用，人とのインターフェース，行動が挙げられている[2]．第 1 回の ISRR（International Symposium on Robotics Research, 1983）での当時 MIT の人工知能研究所の所長 Winston の言葉 “Robotics is the intelligent connection of perception to action”を受けて，有本は，ロボット知能は，知覚そのものと，過去に得た膨大なデータの構造化された記憶と，現時点で感覚として得た信号から，マッチングを取り，そして運動生成に結びつけるネットワークとから成り立つ」と述べている[3]．知識の関係の取得，

知識のデータベース化及び知覚と運動をどのようにつなぐかなどは非常に大きな課題であるとともに興味深い分野である．

ロボットの自律化は，特に他からの支援なしに作業を完遂できることが望まれる分野では大きく期待されている．海洋開発はそのような分野のひとつである．水中ロボットの知能化，自律化の必要性について述べていきたい．

1.1.2 水中ロボットの開発動向

水中ロボットの必要性

地表上での海洋の面積は 3 億 6000 万 km^2 で，陸地の面積 1 億 5000 万 km^2 と比較すると 2.4 倍である．平均深度は 4750m であることを考えると膨大な空間である．日本近海には，貴重な鉱物資源やエネルギー資源があるとされている．各国の経済発展に伴う銅などの枯渇が深刻化してきている[4].

海底下の地殻内にアーキアと呼ばれる微生物が存在することが明らかになってきている．その量は地球上の生物存在量の半分を占めるといわれており，生命の起源や新たな微生物資源の発見も期待されている[5].

このように，海洋開発や科学的調査などは，人類に大きな利益をもたらすが，非常にコストがかかる．このような背景をうけて，ロボットに代表されるような無人機械による調査や開発が期待されている．

水中ロボットの種類

我が国の海洋研究開発機構（JAMSTEC）は世界で一番深く潜れる有人潜水艇「しんかい 6500」を有する．2006 年 7 月 15 日に公開された，小松左京原作の SF 小説のリメイク版映画「日本沈没」には「わだつみ」という名でしんかい 6500 とマントルまで掘削できる地球最深部探査船ちきゅうが登場している[6]．しんかい 6500 はその名の通り，水深 6500[m]まで潜行できる有人潜水艇では世界一の潜行深度を持つ．ウッズホール海洋研究所（WHOI: Woods Hole Oceanographic Institution）の「アルビン」は 1986 年にタイタニック号の潜水調査を行ったことで一躍有名となった．

無人の水中機械は大きく二つに分類される．これは，アンビリカブル・ケーブル（遠隔操作用の索）によってロボットがオペレータに接続されているか否かの違いである．ケーブルで母船と繋がれていて遠隔操縦される有索潜水機は ROV（Remotely Operated Vehicle）と呼ばれ，海洋研究開発機構の ROV「かいこう」は 1995 年に世界最深部であるマリアナ海溝（10,911[m]）への潜行に成功した．また，H2 ロケットの打ち上げ失敗原因調査のため海洋に沈んだエンジンの捜索においても活躍している．ケーブルによって電源の供給やロボットとの通信などが行えるという利点があるが，長いケーブルを持つために運動が制限されることや，支援母船システムの規模が大きくなる等の問題点がある．しかしながら，2003 年 5 月 29 日にかいこうのランチャーとビークルをつなぐ 2 次ケーブルの破断事故によりビークルは行方不明となっており，ROV の取扱いの難しさを語っている[7].

一方，ロボット自身にエネルギーを搭載し，コンピュータからの命令で自動的に海中を運動する無索無人潜水機は自律型海中ロボット（AUV：Autonomous Underwater Vehicle）と呼ばる．人間からのサポートをほとんど必要とせず，ケーブル等からの束縛から離れて自由に潜航できるので，AUV は，地球海洋科学調査のための次世代ツールとして期待されており，その実用化が望まれている．しかし，AUV の実現には，運動制御，センサ情報の取得，行動決定，衝突回避，自己位置推定など様々な問題がある．海中は暗黒で電波も減衰が大きくほとんど届かない．また，陸上とは違いその環境を用意に想像し難い．このような環境で行動する AUV には高い自律性と信頼性が要求される[8]-[10].

1992 年に Twin-Burger（Fig1.3）は自律型水中ロボットの知的行動の研究のための多目的なソフトウェア開発用テストベットとして東京大学生産技術研究所で開発された AUV である[11]. Twin-Burger は非航行型のロボットであり，定点に留まって行う作業や，複数のロボットあるいはダイバーとの協調作業をも含めた広い範囲の行動が可能なロボットとして開発された．自身の状態を把握するためのセンサ，外環境を認識するためのセンサを搭載している．これらを用いて，自身の状態及び外環境を把握し，自らに搭載しているコンピュータで判断を行う．そして，推進器として搭載している 5 基のスラスタの水力を制御して運動する．3 次元空間において，前後（Surge），左右（Sway），上下（Heave），上下軸周りの回転（Yaw）の制御を行う．Twin-Burger で得られた知見を基に，水中ロボットの実用化のためのプロトタイプとし

て Tri-Dog1（Fig1.4）が 2001 年に開発された[12]．鹿児島湾奥部に存在する「たぎり」と呼ばれる海底噴気帯において全自動で数 100[m^2]規模の海底面の写真撮影に成功した．デットレコニングによる測位誤差をあらかじめ設置したランドマーク及び海底噴気をもとにパーティクルフィルタを用いて自己位置を修正する手法を提案し，ロボット単独でドリフトのない測位を実現，誤差の少ないモザイク画像を生成することに成功している[13]．r2D4（Fig1.5）は 2003 年 7 月に完成，2005 年には，伊豆・小笠原海域の明神礁海底火山のカルデラに潜航し，マンガンイオン計で熱水の湧出を確認，インターフェロメトリーソナーで中央火口の斜面を詳細に観察した．自律型の水中ロボットを用いた研究は，海洋観測の分野で成果を挙げてきている．海洋観測の際の航法や，自己位置の測定法なども大きな課題である．

水中ロボットの運動制御

水中ロボットは，水という密度の高い（空気の約 750 倍）媒質の中で行動し，水中で 3 次元的に動き回るため，その運動について以下のような特徴を持つ．

- 海中ロボットの運動は前後 Surge），左右（Sway），上下（Heave），ロール（Roll），ピッチ（Pitch），ヨー（Yaw）の 6 自由度である．
- 運動モード間の相互干渉が大きい．
- 水中ロボットは，魚雷のような特殊な例をのぞいて，一般に，数ノットまでの低速の運動がほとんどである．
- 水からうける揚力や抗力等の非線形流体力の影響はきわめて大きく，さらに加速度に比例する力を代表する量として，付加質量を考慮しなければならない．
- 水中ロボットは浮力が利用できるために，設計の自由度は大きく，定点を保持して仕事をする定点型のロボットや，長い距離を泳ぎつづける航行型のロボットが存在する．

水中ロボットの運動方程式は，ロボットが左右対称で中性浮力，重心と浮心が一致すると仮定すると次式で表現される[8],[9].

$$
\begin{aligned}
m(\dot{u}+qw-rv) &= F_{A_x}+F_{H_x}+F_{T_x} \\
m(\dot{v}+ru-pw) &= F_{A_y}+F_{H_y}+F_{T_y} \\
m(\dot{w}+pv-qu) &= F_{A_z}+F_{H_z}+F_{T_z} \\
I_{xx}\dot{p}-I_{xz}\dot{r}+\left(I_{zz}-I_{yy}\right)qr-I_{xz}pq &= M_{A_x}+M_{H_x}+M_{T_x} \\
I_{yy}\dot{q}+\left(I_{yy}-I_{zz}\right)rp+I_{zx}\left(p^2-r^2\right) &= M_{A_y}+M_H+M_{T_y} \\
I_{zz}\dot{r}-I_{zx}\dot{p}+\left(I_{yy}-I_{xx}\right)pq-I_{xz}qr &= M_{A_z}+M_{H_z}+M_{T_z}
\end{aligned}
\tag{1.1}
$$

m, I は質量及び慣性モーメント，u, v, w は並進速度，p, q, r は角速度，F, M は力及びモーメントを表す．添え字の A は付加質量による項，H は流体力，T はロボットによる推力を意味する．

水中環境において行動する水中ロボットの運動解析[14]-[16]]は，非線形な流体力や付加質量，潮流に代表される外乱等のため不確定な要因が多く難しい課題となっている．1980 年代以降，ニューラルネットワークやファジー等，Computational Intelligence[17], [18]と呼ばれる新しい情報処理技術が注目され，水中ロボットへも様々な応用が試みられている．

ニューラルネットワークを水中ロボットの運動制御へ応用した例は，藤井らによる運動モデルを経由して間接的にコントローラを設計する手法[19]-[22]や，須藤らによる海中ロボットの定高度航行への応用[23], [24]，Yuh らが提案するロボットの実験データから直接的にコントローラを調整する手法[25], [26]やファジーとの組合せ[27], Guo らによるオンライン調整法[28]，PID 制御や非線形制御とニューラルネットワークを組み合わせた手法[29], [30]がある．大別すると，フォワードモデルを介して間接的にコントローラを生成する手法，制御誤差から直接的にコントローラを生成する手法，既存の制御理論とニューラルネットワークの組合せに分類することができる．船舶に適用した例としては，R.S. Burns による教示型のコントローラ[31]や小川原らによる学習型フィードフォワードコントローラ[32]等がある．R.S. Burns は，シミュレーションにおいて速度を変えた場合の制御性能を最適制御と比較しており，良好な結果が得られることを示した．小川原らは，PID コントローラと学習型フィードフォワードコントローラを組合せ，風などの外乱を補償するシステムを構築している．また，山本らはスラスタに変わる手法として，船のラダーを左右方向に振動させて推力を得る手法を考案しており，推力の制御にニューラルネットワークを用いている[33]．他にも，超音波通信等で問題になるマルチパスを考慮して，音源の方向を特定する手法

[34]や，海底面と海面での温度，海底面でのマルチパスの影響を含んだ到達時間等から１年間の深度変化による音速への影響を出力するネットワークの構築[35]，故障診断[36]，[37]等にも応用されており，ニューラルネットワークは幅広い信号処理の道具として今後の発展が期待されている．

Fig 1.1 SHINKAI6500
(http://www.jamstec.go.jp/jamstec-j/gallery/yujin/6500.html)

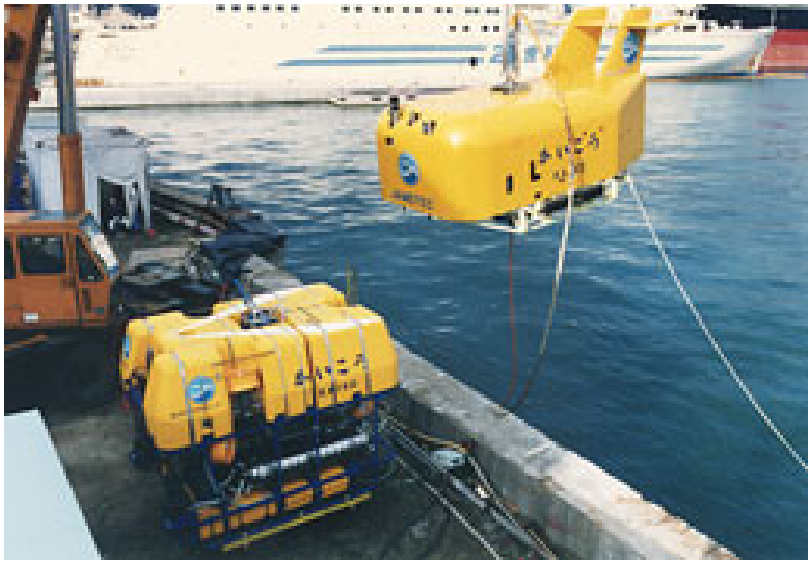


Fig 1.2 KAIKO
(http://www.jamstec.go.jp/jamstec-j/gallery/mujin/kaiko.html)

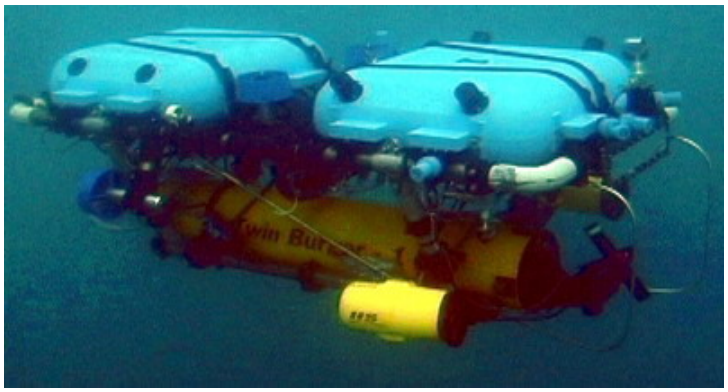


Fig 1.3 Twin-Burger 1

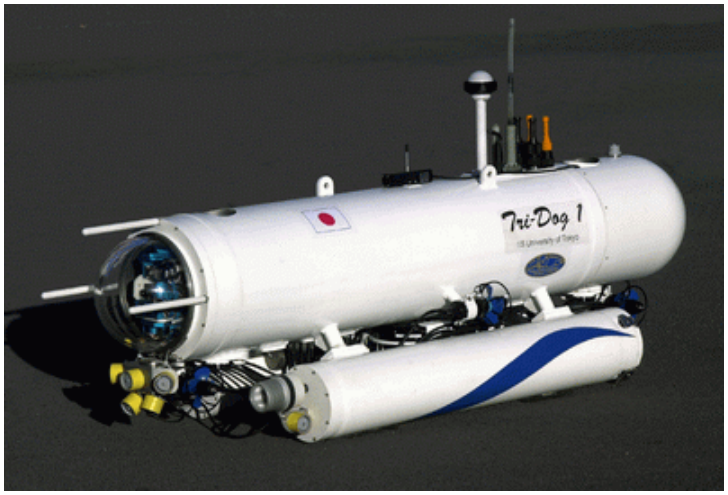


Fig 1.4 Tri-Dog 1
(http://underwater.iis.u-tokyo.ac.jp/robot/tri/tridog.html)

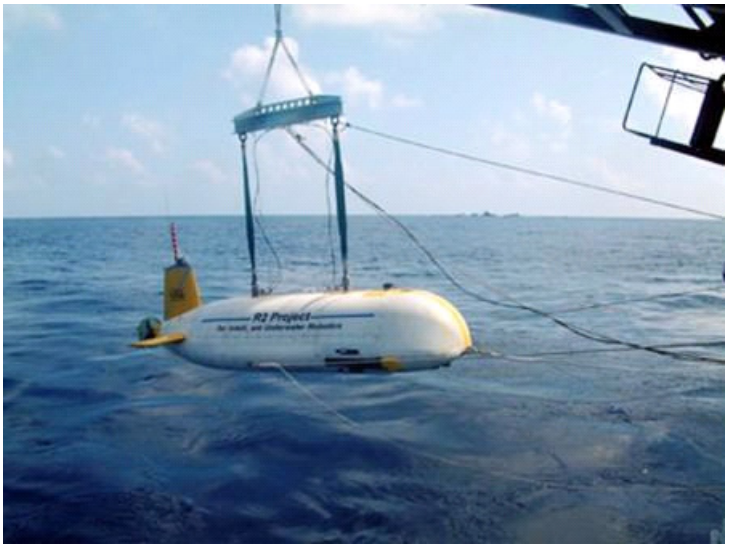


Fig 1.5 r2D4
(http://underwater.iis.u-tokyo.ac.jp/top/myoujin2005/Myoujin05.html)

1.2 研究目的

水中ロボットには，高い自律性の要求のため解決すべき過大が多く存在する．例えば，複雑な動特性，環境認識の必要性，行動決定手法の確立，自己位置同定，各種情報を取得するためのセンサ開発，アクチュエータの開発などが挙げられる．本論文では，運動制御，環境認識と環境認識について扱う．

運動制御

水中ロボットの運動制御システムは以下に起因する水中ロボットの動特性の変化の影響を受ける．

- ミッションによる搭載機器の変更
- マニピュレータの位置及び姿勢
- スラスタの特性
- 潮流や波などの外乱

動特性の変化を想定し，様々な状況に適応できる制御器が必要となる．

環境認識と行動決定

水中でロボットが周囲の環境から得られる情報としては以下が考えられる．

- 画像

  カメラによって取得．水中へ十分な光が届かないことからライトやストロボなどによって光源を用いて撮影したり，観測対象物に十分に近接して撮影する必要がある．

- 音響

  クジラやイルカが超音波によってコミュニケーションを取っていることが知られているように，水中では音響による情報の取得が有効である．超音波測距センサによる距離情報，サイドスキャンソナーによる超音波画像などが挙げられる．

- CTDO
  海水や湖水の成分調査のために用いられている．塩分濃度，深度，水温，電気伝導度，溶存酸素量などを計測する．
- 磁場
  地磁気を計測し，環境固有の特徴が得られる．他にも，KDDI 研究所の AE2 は海底ケーブルから出る磁場を計測しケーブルトラッキングを行う等が報告されている．

水中ロボットとの無線での通信は超音波で行うのが一般的であるが，伝送容量が小さく，環境についての様々な情報をロボットに与えるのは困難である．したがって，自律型水中ロボットは搭載されたセンサ情報から環境を認識し行動を決定しなければならない．

自己組織的行動獲得

このような自律型水中ロボットの問題点を考慮すると，ロボットが状態や環境と行動の関係を学習によって自己組織的に得ることが望ましい．そこで，本研究の目的は，自律型水中ロボットにおける自己組織的行動獲得システムの開発である．

Fig1.6 に提案システムの概念図を示す．人間がある行動を決定する過程（ここでは移動すること）を例に考える．人間は頭の中に移動するフィールドの特徴を記した地図を基に目的地への経路計画を立てる．一方，人間は，自身の状態の変化による運動特性に関する知識及び自身の状態と対となる運動制御に関する知識を持っていると考えられる．したがって，荷物を持って歩く，雨が降っているので傘を差して歩く，乗り物で移動するなど様々な状態においても適切に行動を選択できる．つまり，環境に関する情報と運動に関する情報から適切な行動を選択する．また，目的地や経路計画に併せて，徒歩で移動することを選択した場合においても，疲れないように楽に歩く，急ぐので走るなどの行動を選択できる．

以上のような概念を水中ロボットの行動獲得へ適用する．Fig1.7 に示すように，ロボットは作業フィールドの環境特徴地図（Fig1.7 左）をセンサ情報を基に作成する．自状態と環境特徴地図を基に，目標となる経路や行動を決定する．一方，ロボットは

移動しながら自状態の変化を動特性に関する知識を獲得し，対応した制御器を作成する．自状態と動特性に関する知識とを照らし合せて適切な制御器を選択する．目標行動に適した制御器を選択し目標値に対して最適な出力を算出する．この出力値をロボットの制御力として採用する．

提案する自己組織的行動獲得システムでは，基本となる環境に関する知識及び運動に関する知識を必要とする．よって，本研究では，基本システムとなる環境認識と行動決定システム及び運動制御システムを脳型情報処理技術を用いて構築する．環境認識と行動決定システムでは，ロボットが取得したセンサ情報から自動的に環境を認識し行動を決定しなければならない．しかし，環境がどのような特徴を持っているのかはあらかじめ知ることが出来ないので環境情報を基にクラスタリングする必要がある．このような問題には，自己組織化マップに代表される教師なし学習が有効である．

自己組織化マップ（SOM: Self-Organizing Map）[38], [39]は Kohonen の提案する脳の記憶のメカニズムを模倣したアルゴリズムで，強力なデータマイニングツールとして知られている．教示データの幾何学的関係を保ったまま，低次元化し，学習の過程で類似したデータがクラスタリングされ，特徴を抽出することが出来る．このような特徴から，環境情報などのデータベース化に適していると考えられる．

SOM を拡張した例として，山川らの提案している自己組織化関係ネットワーク（SORN: Self-Organizing Relationship Network）[40]がある．SOM を基に対象システムの入出力関係を近似できるように改良することによって開発された．SOR ネットワークの特長としては，対象システムの入出力データと，それに対して設計者が与えた評価基準に基づいて学習を行えることが挙げられる．望ましい入出力（教師信号）を得ることが難しいが，得られた入出力データを評価できる場合に有効である．学習データに与える評価は設計者の主観に基づく評価でも，定量的な評価でも良く，対象システムから得られた望ましい学習データだけでなく，望ましくない学習データからも学習（斥力学習）を行う．SORN を用いることで，状態と行動の適切な関係を得られる．

運動制御システムにおいては非線形かつ複雑な動特性を持った水中ロボットの制御器を構築しなければならない．このような問題には，階層型ニューラルネットワークに代表される教師あり学習が有効である．また，ミッションの違いによる装備品，搭載機器の変化によるロボット自身の動特性の変化や，潮流の変化によるロボット周囲

の環境の変化に起因する運動特性の変化は，制御システムに大きな影響を与える．ロボットが自身や環境の変化に対して適応的に行動することができれば望ましい．現在まで，ニューラルネットワークを始めとする生物の情報処理アルゴリズムの仕組みを参考にした学習能力に注目し，水中ロボットの運動制御や行動決定に関して研究を行い，非航行型の自律型水中ロボット Twin-Burger を用いた実験を通じて有効性の研究が行われており，水中ロボットの動特性をリカレント型ニューラルネットワークで表現し，オンラインで制御器と動特性の調整を行っている [41]-[44]．提案されているニューラルネットワークによる適応制御手法では，水中ロボットの動特性及び制御器をオンラインで獲得することができるが，一対の動特性と制御器しか持っていないために適応する (動特性と制御器を調整する)過程で，過去における学習によって得た情報の影響は低下していく．つまり，過去において適応していた動特性及び環境に戻った場合においても再学習する必要があり，常時，ロボットにおいて計測される時系列情報を追加学習しなければならない．より高速に，かつ柔軟に動特性などの変化に適応するには，過去を含めた様々な適応状態を保持しながら，新しい動特性や環境へ適応する行動獲得システムが必要となる．

人は一度獲得した技能は，新たに技能を獲得したからといって脳における知識獲得（技能）に関して失うことはない．獲得した複数の技能に対する複数の順モデルと逆モデルに関するモジュール構造の存在の可能性が指摘されている．そのような構造の一つとして川人らは多重順逆対モデル（MOSAIC: MOdular Switching And Identification for Control）[45]を提案しており，複数の動特性と制御器の対によって制御システムを構成している．同定器の予測誤差を用いて各制御器の出力に重み付けをし，制御出力を決定することでロバストな制御の実現を行っており[46]，[47]，MOSAIC と強化学習を組み合わせて，2 リンクの振り子の振り上げ運動の獲得に成功している[48].

自己組織化マップを用いて，ロボットの状態及び周囲の環境の変化とロボット動特性の関係を表現したダイナミクスマップを得ることができれば，ロボットを中心とした局所環境の状態遷移の観測や，得られたダイナミクスマップを基にして同時に複数の動特性に対応した制御器を獲得することが可能である．しかしながら，一般的な SOM は入力としてベクトル空間しか取り扱うことができない．さらに学習過程が教師無し学習であるため，ロボットの時系列情報から入出力関係を学習してロボットの動特性や制御器を表現するには，競合層における (入力, 出力) の組としてベクトル

表現する等の工夫が必要となる．入出力の写像関係を得るには，教師有り学習アルゴリズムを持ったニューラルネットワークが適している．ロボットの行動決定システムには，ロボットがおかれた環境を自動的に認識し把握する“教師無し”アルゴリズム，及び動特性や制御則獲得のための“教師有り”アルゴリズムの両者の考え方が必要であり，両者の長所を取り入れた情報処理技術が望まれる．徳永らによって提案されたモジュラーネットワーク自己組織化マップ (mnSOM: modular network Self-Organizing Map) [49] は，関数をモジュールの要素として扱えるため，自己組織化マップの教師無しアルゴリズムを継承しつつ，モジュールの要素として教師有り学習アルゴリズムを有する情報処理システムを導入することにより，上記の両者のアルゴリズムを同時に実現することが可能である．このような特徴を活かして，動的システムのマッピング，非線形主成分解析，自己組織化制御器などの適応例が報告されている [50]-[53].

本論文では，自律型水中ロボットにおける自己組織的行動獲得システムの開発について述べる．本論文の構成は，第 2 章にて，提案システムの適用対象としている自律型水中ロボット”Twin-Burger”について述べる．第 3 章では，ニューラルネットワークの基礎としてその概要及び本研究で情報処理システムとして用いた SOM および mnSOM のアルゴリズムを紹介し，代表的な適用例を示す．第 4 章では，提案する水中ロボットにおける自己組織的行動獲得システムの基礎システムの開発として，SOM を用いた環境認識及び行動決定システム， mnSOM を用いた適応制御システムについて提案し，水中ロボットへの適用を想定したシミュレーション結果について述べる．第 5 章では自律型水中ロボット Twin-Burger を用いた実験について，第 6 章では得られた結果に対する考察を行い．第 7 章は結論とする．

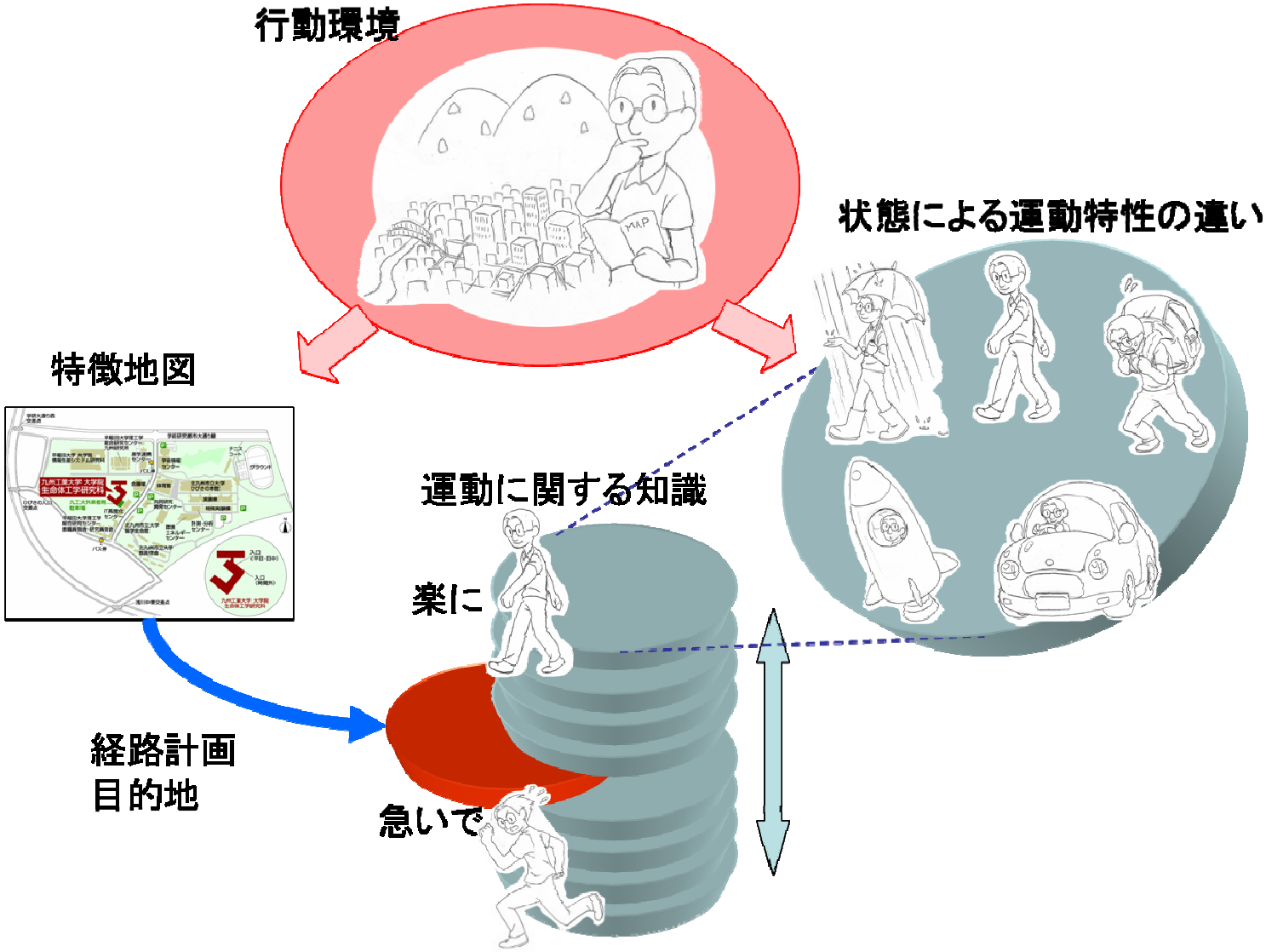


Fig 1.6 Concept of Self-Organizing Decision-Making System

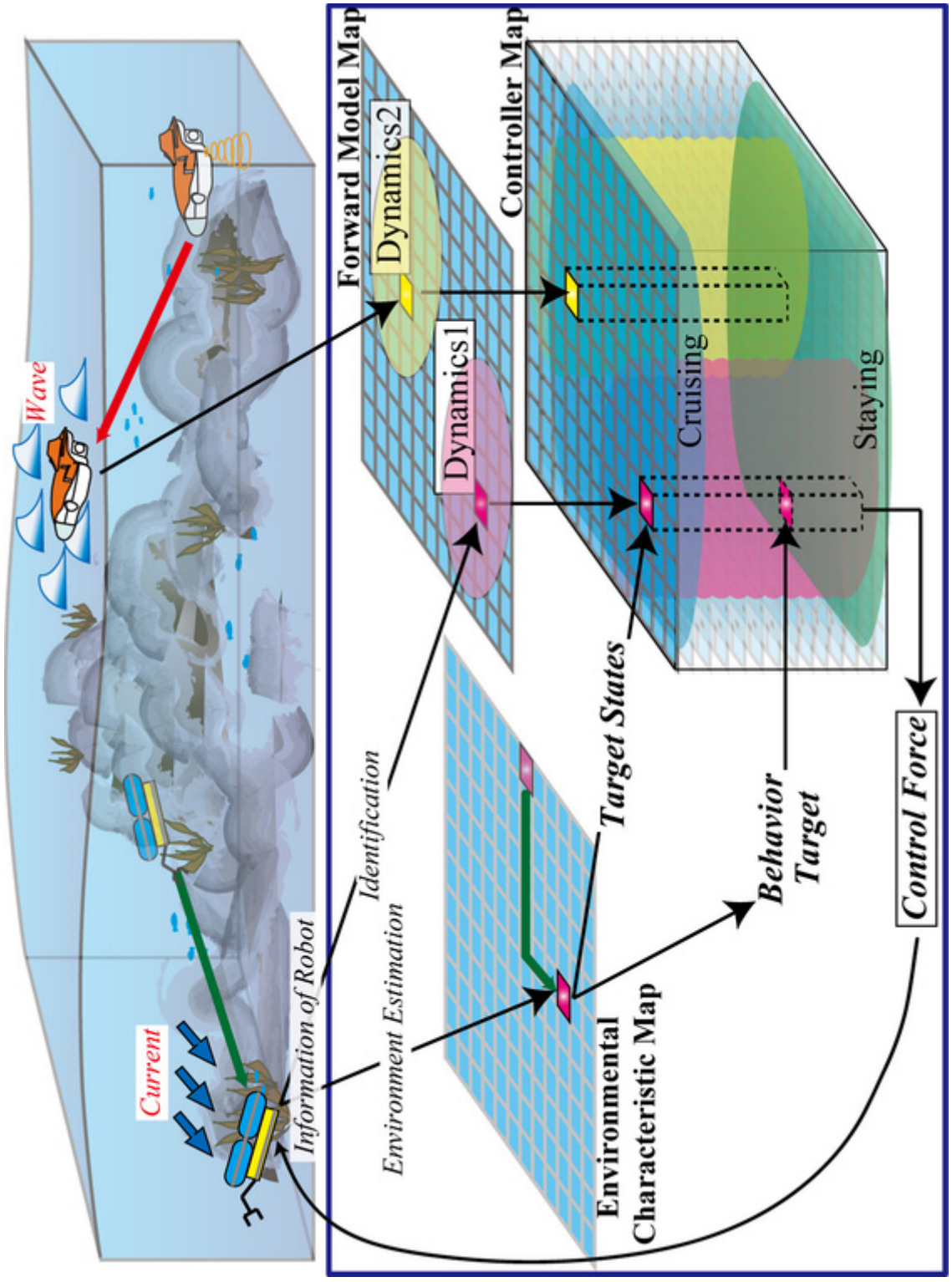


Fig 1.7 Self-Organizing Decision-Making System for AUVs

第2章

自律型水中ロボット

“Twin-Burger”

本研究で提案するシステムは，自律型水中ロボット Twin-Burger を適用の対象としている．Twin-Burger は，1992 年に東京大学生産技術研究所において開発された非航行型のロボットで，定点に留まって行う作業や複数のロボットあるいはダイバーとの協調作業をも含めた広い範囲の行動を目的とした知的行動研究開発のためのテストベットである．旧ロボットシステムでは複数のトランスピュータを用いて，ハードウェア的に並列処理を行っていた．しかし，システムの老朽化，トランスピュータの製造中止及び近年における汎用 CPU の処理能力向上などの点から，システムを再構築を行った．

本章では，新たな Twin-Burger のハードウェアシステム及びソフトウェアシステムについて紹介する．

2.1 ハードウェア

Twin-Burger は，全長 1.3 [m]，乾重量約 110 [kg]で，バッテリを格納するアルミニウム製シリンダ，2 つの FRP 製耐圧容器及びその他の機器をフレームに取り付けたオープンフレーム構造のロボットである．フレームの上部に FRP 容器を，下部にシリンダを取り付けることにより，重心位置，浮心位置の距離を十分にとり，ピッチとロールに関する静的安定を確保してある．Table2.1 に Twin-Burger の仕様を示す．以降，Twin-Burger の搭載機器の詳細について述べる．

2.1.1 センサ

Twin-Burger に搭載されているセンサ類は，Twin-Burger 自身の内部的な情報を得るためのものと外部の情報を得るためのものとの 2 つに分類される．

内部情報収集用センサ

運動計測用のセンサとして，3 軸まわりの角度及び角速度を検知する姿勢センサ，前後進及び左右方向の対水速度を計測するセンサ及び深度センサを搭載している．

- 姿勢センサ

  3 軸まわりの角度及び角速度計測用のセンサとしては，TCM2 をを採用する．液面の傾きを計測することでピッチ及びロール角を計測している．また，方位を計測するために 3 軸の磁束密度計を備え，磁気速度ベクトルの水平面投射から方位を求めている．

- 対水速度計

  対水速度を計測するセンサとしては，プロペラ式流速計を採用し，光反射式で正逆方向の流速を感知するものとした．流速の情報は，センサヘッド（プロペラ部分）から出力される 2 相のパルスをカウントすることにより回転方向のステータスと流速に対応したパルスに変換し増幅する回路を介してコンピュータに取り込むことができる．流速計の出力信号は極性信号とパルス信号である．

- 深度計

  深度を計測するセンサとしては，絶対圧タイプの圧力センサを用いている．実験の条件に応じて 0 ~ 50[m]と 0 ~ 10[m]の 2 種類のレンジを使用する．いずれも増幅器内蔵型で大きいレンジは歪みゲージ式，小さいレンジは半導体感圧式のものを採用し，感圧部のネジ径を変換するニップルを用いることにより，交換して使用することが可能である．

外部情報収集用センサ

- 距離センサ

  外部環境認識のためのセンサとして，8 チャンネルの超音波距離センサを搭載する．このセンサは 8 方向の水中障害物を検知し，出力データとして障害物までの距離を 8 [bit]バイナリで出力するものである．検知可能距離は 0.3 ~ 12.75 [m]である．

2.1.2 アクチュエータ

推進器として 5 基のスラスタを搭載している．スラスタは定格出力 48.6 [W]の DC モータの回転を，ギヤによって 1/4.33 に減速し，さらに水密のためにマグネットカプ

リングを介してプロペラに伝える構造のものである．そのため，指令値の急激な変化による脱調を起こすという問題が存在する．モータを両軸にすることにより，駆動軸の反対側に磁気式の 2 相のエンコーダを取付け，回転数のモニタを可能とした．モータは，MOSFET による電流駆動によってドライブされ，0～10 [V]の電圧入力によって正逆方向に関して制御が可能である．スラスタの取り付け位置は，Fig.2.1 のようになっている．左右に取り付けられた前後方向のスラスタ２基を用いて前後進及び差動による回頭を行い，中心部に取り付けられたスラスタにより横方向の，上下方向のスラスタにより上下方向の並進を行う．

2.1.3 電源

Twin-Burger の電源は，シリンダ内に搭載する 25.2[V]及び 28.8[V]の２個の Ni-Cd 電池から供給される．25.2[V]系はコンピュータ関係，28.8[V]系はアクチュエータ関係への電源供給を受け持つ．センサなどの搭載機器に必要な電源系統は，5，12，及び±12[V]である．これらの電圧は主電源 25.2[V]より DC/DC コンバータを介して生成し各機器へ供給している．スラスタの駆動用電源のみ 28.8[V]の電池より直接供給される．バッテリによる活動時間は約 2 時間となっている．

2.1.4 コンピュータシステム

Twin-Burger の 2 つの耐圧容器内にひとつずつコンピュータを搭載している．コンピュータシステム 1 では，主に知的行動に関する処理，及び超音波センサからの距離データの取得などの外環境情報の取得を行う．コンピュータシステム 2 では，各種内部状態センサ情報の取得，アクチュエータの制御等のロボットの運動に関する処理を行う．

それぞれのコンピュータシステムの構成は以下のとおりである．

コンピュータシステム 1（Hull#1）

- CPU ボード：Advantech（PCN6351）
- CPU：AMD K6-2 266MHz
- 画像処理ボード IP5000

コンピュータシステム 2（Hull#2）

- CPU ボード：Advantech（PCA6753）
- CPU：NS GXM processor 200MHz
- D/A ボード
- A/D ボード
- カウンタボード

Hull#1 には，PCI，ISA バスのどちらにも対応している Advantech（PCN6351）を使用し，バックプレーンボードに搭載している．また，ビジョンシステムから得られる画像情報の取得及び処理のために画像処理簿ボード IP5000 を搭載しいている．Hull#2 には，省電力 CPU を搭載した Advantech（PCA6753）を用い，同様にパックプレーンボードに PCA6753，D/A，A/D，カウンタボードを搭載している．

Fig.2.1 にハードウェアとインターフェースの接続を示す．

Hull#1 では，CCD の映像を IP5000 よりコンピュータへ，超音波センサへの信号の送信とデータの取得は，パラレルポートを介して行っている．

Hull#2 では，深度計は深度に応じた電圧を出力するので，それを A/D ボードを介して取り込む．スラスタは，D/A ボードより 0～10[V]の指令電圧をモータドライバへ与え，モータドライバより電流制御で駆動している．流速計は，センサ用回路を介して，速度に応じたパルスと，その方向を出力する，それをカウンタボードで取り込む．姿勢センサ TCM2 は，Roll，Pitch，Yaw を ASCII コードで出力する．それをシリアルポートシリアルポートより取り込んでいる．CCD カメラ用サーボモータはシリアルポートから制御信号をサーボモータドライバに送り，サーボモータドライバからの PWM 信号により制御している．

Table 2.1 Specifications of Twin-Burger

Dimensions	1.54[m] x 0.86[m] x 0.54[m]
Weight	120[kg]
Battery	25.2[V] 4[Ah] 28.8[V] 4[Ah]
Activity Time	2 [h]
Sensor	TCM2 (Roll, Pitch, Yaw) Depth Flow Ultrasonic Range Finder x 8
Actuator	40[W] Thrusters x 5
Computer	AMD K6-2 266[MHz] NS GXM processor 200[MHz]

2.2 ソフトウェア

2.2.1 開発環境

自律型のロボットは，判断機構を初めとした外部環境把握，運動制御等の複数の処理を並列的に処理しなければならない．また，海中という極限環境であるということを考えると，AUV には高い自律性が要求される．また，判断機構等に関しては要求に対する補償が必要である．つまり，自律型ロボットには並列処理，及び，実時間処理が可能なシステムが必要ということになる．本研究ではロボットシステムのオペレーティングシステムとして，並列処理が可能な Linux にリアルタイム性を付加した RTLinux を採用した．

RTLinux は Michael Baraonov，Victor Yodaiken によって開発された，Linux をリアルタイム OS に拡張したものである RTLinux には，スケジューリングや割り込みを行うための API が用意されており，容易に実時間処理を実現することが出来る．以下に特徴をまとめると，

- フリーウェアであり，ソースコードが公開されている
  - Linux の資源活用できる
  - ネットワークへのアクセスが容易
  - マルチタスク・マルチユーザ
- API・ドライバ等が提供されている
  - スケジューリング，割り込み
  - メモリアクセス，シリアルポート使用のためのドライバ

などが挙げられる．

ロボットを制御するためには，直接 PORT へ IO アクセス出来ることは不可欠である．また，ネットワークを介してロボットへアクセスし，状態の監視，コマンドの送信などを行うことも不可欠なことである．RTLinux に用意されている API，ドライバ

を用いることでソフトウェアの開発も容易に行える．割り込みに対する遅れは最大で15[μsec]となっており，ロボットを制御するには十分な性能であると考えられる．

以上のことより，本研究の対象とする AUV のソフトウェア開発環境として RTLinux は十分な機能・性能を持っており，採用することとした．今回は，Linux2.2.18，RTLinux3.0 を用いている．Finite State Machine Labs（FSMLabs）（http://fsmlabs.com/）が RTLinux を Linux カーネルへのパッチとして提供しており，最新版は ver.3.2 で，Linux2.4 系にも対応している．

2.2.2 開発したソフトウェア

リアルタイム性が必要な処理はカーネルモジュールとして開発し，リアルタイム性を要求しない処理は Linux プロセスとしてソフトウェアを開発した．Fig2.2 に開発したソフトウェアの構成を示す．カーネルモジュールと Linux プロセスとのデータの通信は mbuff と呼ばれる共有メモリを介して行う．Fig2.2 において mbuff を中心に，左側がカーネルモジュール，左側が Linux プロセスで作成したソフトウェアである．カーネルモジュールは，アクチュエータの制御及び各センサ情報取得に対して個別に作成し，それぞれ 10[Hz]で動作する．超音波測距センサのデータ取得 1[Hz]で動作する．Linux プロセスでは，ロボットの状態表示やデータ保存のためのユーザインターフェース，Hull#1，#2 間の通信及び外部情報を基に目標状態の決定を行っている．作成した Linux プロセスは全て 1[Hz]で動作する．

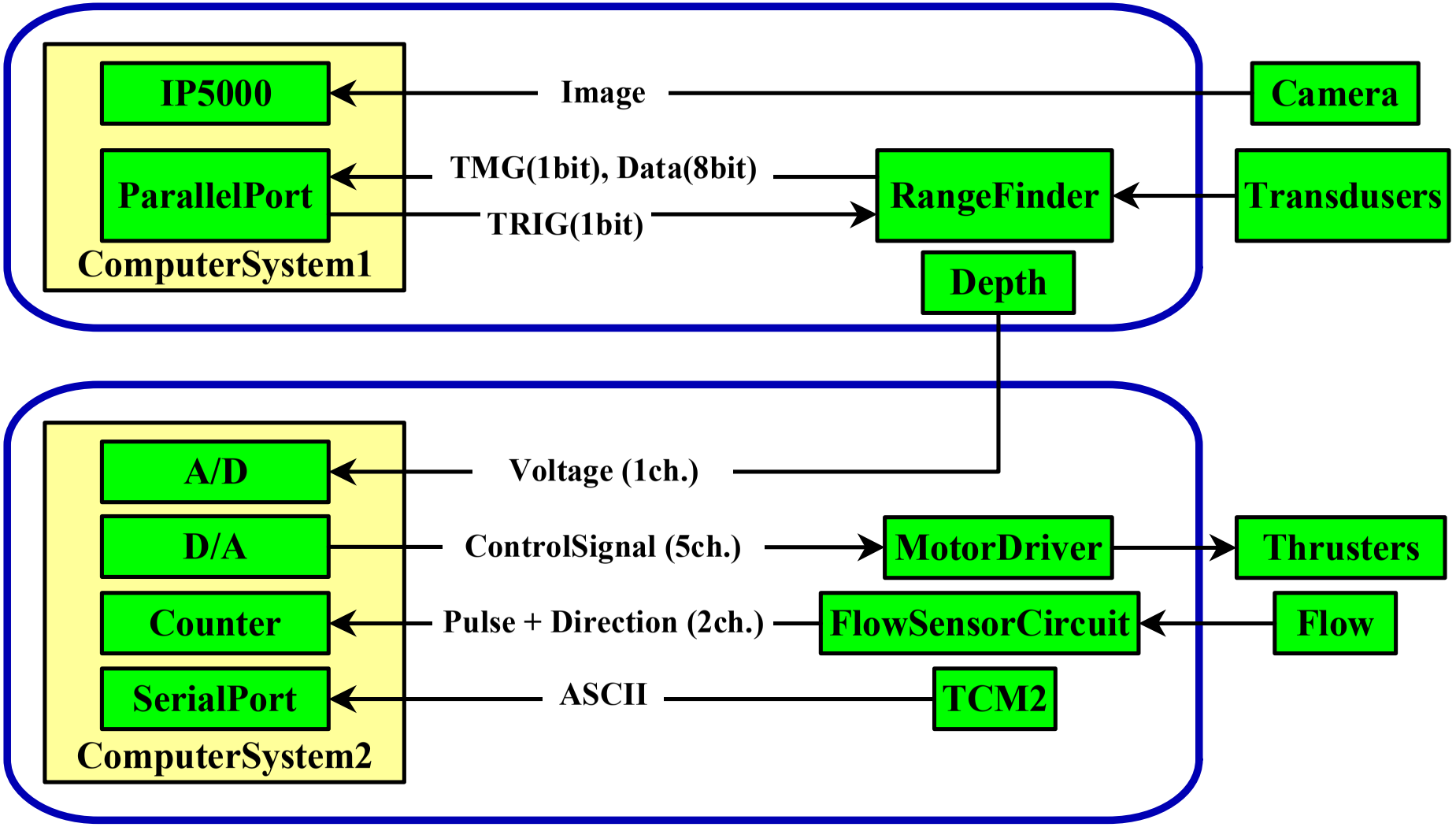
以下に，作成したプログラムリストを挙げ，詳細を述べる．

Hull#1

- Kernel Modules
  - log.c
    - データの管理
    - 共有メモリへのアクセス
    - 実行周期 0.1 [sec]
  - range.c
    - 超音波センサのデータの取得
    - パラレルポートを使用した割り込み処理
    - 実行周期 1.0 [sec]
- User Programs
  - somnavi.c
    - 障害物回避ナビゲーションプログラム
    - 超音波センサのデータと，SOM によって学習した結果より，ロボットの目標角度を求める
  - houghnavi.c
    - ケーブルトラッキングによるナビゲーションプログラム
    - カメラの画像よりケーブルを認識し，ロボットの目標角度を求める
  - client.c
    - Hull#2 コンピュータとの Socket 通信プログラム
    - 超音波センサのデータ，ナビゲーションプログラムより求めた目標角度を Hull#2 へ送る

Hulll#2

- Kernel Modules
  - log.c
    - データの管理
    - 共有メモリへのアクセス
    - 実行周期 0.1 [sec]
  - actuate.c
    - 制御力より制御電圧を求める
    - D/A ボードへのアクセスし，アクチュエータへ制御電圧の出力
    - 実行周期 0.1 [sec]
  - sensor.c
    - 姿勢，対水速度，深度の計測
      - 姿勢センサ
        - シリアルポート通信よるデータの取得
        - 実行周期 0.2 [sec]
      - 流速計
        - カウンタによるパルスのカウント
        - 実行周期 0.1 [sec]
      - 深度計
        - A/D 変換による深度の計測
        - 実行周期 0.1 [sec]
  - control.c
    - センサデータと目標値より制御力を求める
    - 実行周期 0.1 [sec]
  - ui.c
    - ユーザプログラムからのコマンドの受信
    - 実行周期 0.1 [sec]
- User Programs
  - gui.c
    - コマンドの送信
    - ロボットの状態をディスプレイに表示
    - ログファイルの作成
  - server.c
    - Hull#1 コンピュータとの Socket 通信



F ig 2.1 Hardware Structure of “Twin-Burger”

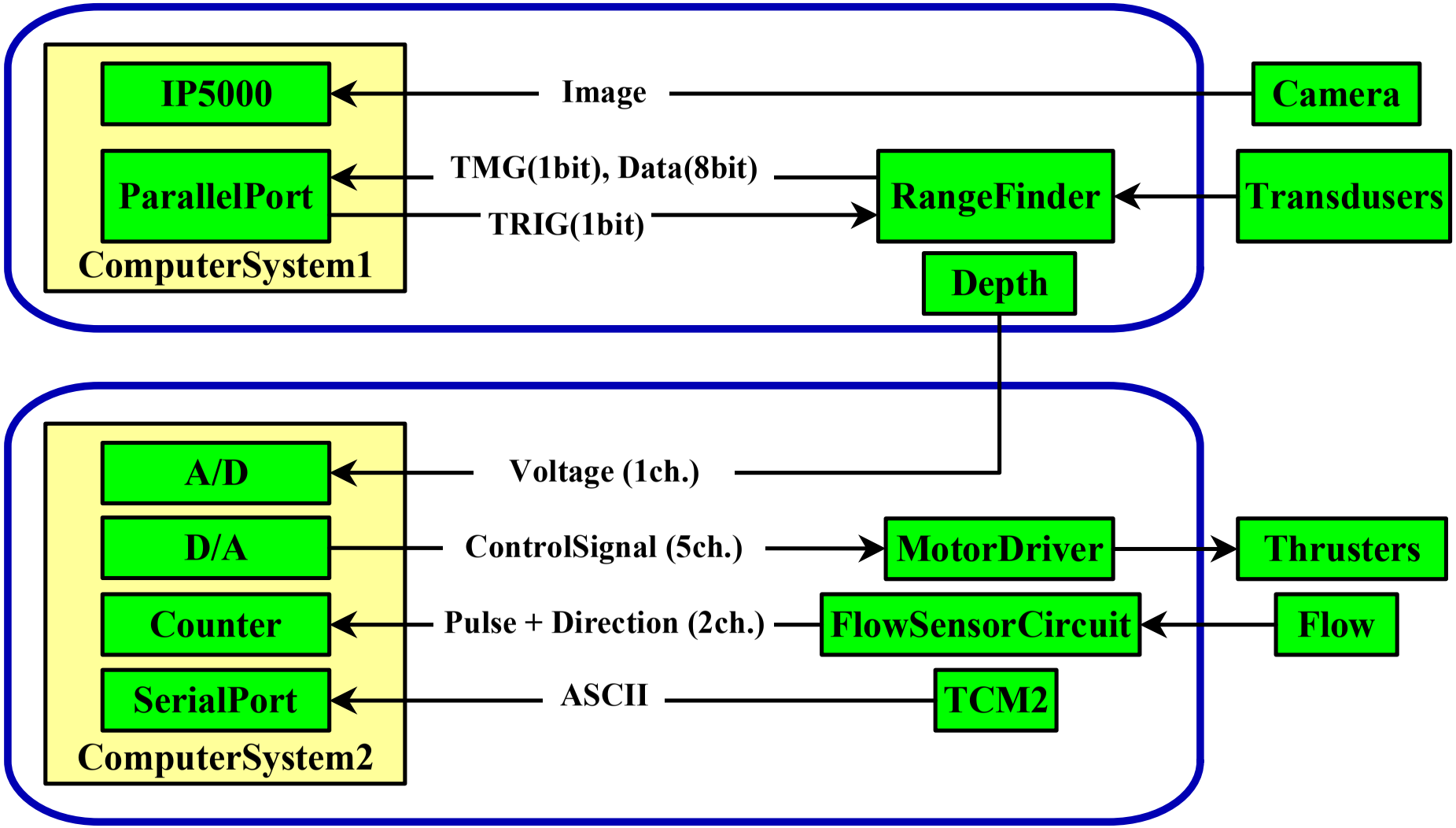


Fig 2.2 Software Structure of “Twin-Burger

第3章
ニューラルネットワークの基礎

3.1 ニューラルネットワーク

ロボットの自律化を考えたとき，得られたセンサ情報から環境や状態を把握することは重要である．膨大な時系列データに含まれている環境や状態を表す特徴的な情報を取得できれば，ロボットの行動決定に活用できる．このような問題には，教師無し学習である自己組織化マップが有効である．また，ロボットの動特性や制御器を表現するためには，制御入力（力やトルクなど）に対するロボットの状態（速度や加速度など）の変化の関係を学習する必要がある．このように入出力関係を学習するためには，教師あり学習が可能なニューラルネットワークが有効である．

環境や状態の変化に起因する動特性の変化に適応できる制御システムの実現を考えると，教師なし学習と教師あり学習の両方の特徴を備えた情報処理技術が必要となる．このような特徴を持った手法として，モジュラーネットワーク型自己組織化マップが提案されている．

この章ではニューラルネットワークの基礎として，ニューラルネットワークの概要，自己組織化マップ，モジュラーネットワーク型自己組織化マップをアルゴリズム及び適用例について述べる．

3.1.1 ニューロンの数式モデル

ニューロン（脳の神経細胞）の活動を数式で模擬したものである．ニューラルネットワークとは，ニューロンのモデルを基本素子（ユニット）としたネットワークのことである．ユニットは方向性を持ったシナプスで結合されており，それぞれに結合加重を持っている．結合加重はユニット間の信号の伝播を制御しており，結合加重がせいの場合は興奮性結合，不の場合は抑制性結合を意味する．

基本素子としては，Fig3.1 に示す入力加重和型のユニットがよく用いられる．式(2.1)に示すように，他のユニットからの入力 y_j と結合加重 w_{ij} の加重和とユニット i の持つバイアス値 b_i によってユニット i の活性値 x_i が決定され，活性値が式(2.2)に示すように出力関数 f によって変換され出力 y_i が決定される．

$$x_i = \sum y_j w_{ij} + b_i \tag{3.1}$$

$$y_i = f(x_i) \tag{3.2}$$

ユニットの活性値を出力に変換する関数には，様々なものがある．加重和が閾値よりも大きければ 1，それ以外では 0 を出力するステップ関数．加重和が±∞に近づくと出力が飽和する単調増加で微分可能なシグモイド関数．線形なシステムを表現するために線形関数などが用いられる．本論文で述べるニューラルネットワークの学習ではシグモイド関数を採用している．

3.1.2 階層型ニューラルネットワーク

階層型のニューラルネットワークは，入出力間の静的な写像関係を得るために用いられる．Fig3.2 に示しているのは典型的な 3 層ネットワークである．左から入力層，隠れ層，出力層からなっており，この順に信号が伝播する．

ここでは，階層型ニューラルネットワークの学習アルゴリズムである誤差逆伝播法について述べる．Fig3.2 の階層型ネットワークに，K 個の入力のデータが与えられるとする．入力データをネットワークに与えたときに得られる出力データと出力目標値が出来るだけ一致するように，すなわち，式(3.3)の自乗誤差和を誤差評価関数とし，これが最小になるように結合加重を修正していくことで学習が成り立っている．

$$E = \sum_k E^k = \frac{1}{2} \sum_k \sum_i (y_i^k - d_i^k)^2 \tag{3.3}$$

ここで，E^k，y_i^k，d_i^k はそれぞれ，k 番目の入力データに対する評価値，出力層のユニット i の出力値，出力ユニット i の目標値である．

誤差逆伝播法では，式(3.4)のように最急降下法を用いて結合加重の修正量Δw_{ij} を以下のように定義する．

$$\Delta w_{ij} = -\eta \frac{\partial E^k}{\partial w_{ij}} \tag{3.4}$$

ηは学習係数で[0, 1]の範囲で選択される．式(3.4)を用いて入力データが与えられるたびに結合加重を修正する方法を逐次学習と呼ばれている．また，式(3.4)の E^k の偏微分の代わりに E の偏微分を用いてすべての入力データに対して結合加重を修正する方法はバッチ学習と呼ばれており，入力データの定時の順序に意味がある場合などはこの方法でなければうまく学習できない場合がある．式(3.4)の偏微分は次のように表現できる．

$$\frac{\partial E^k}{\partial w_{ij}} = \frac{\partial E^k}{\partial x_i^k}\frac{\partial x_i^k}{\partial w_{ij}} \tag{3.5}$$

x_i^k は k 番目の入力データに対する入力ユニット i の出力値である．ここで，式(3.5)右辺の第 2 項は，式(3.1)より，一つ前のユニットの出力と等しい．

$$\frac{\partial x_i^k}{\partial w_{ij}} = y_j^k \tag{3.6}$$

次に，式(3.5)右辺第一項をユニット i の誤差信号δ_i^k とおいて，x_i^k に対する出力 y_i^k を用いて以下のように変形する．

$$\delta_i^k = \frac{\partial E^k}{\partial x_i^k} = \frac{\partial E^k}{\partial y_i^k}\frac{\partial y_i^k}{\partial x_i^k} \tag{3.7}$$

ここで，式(3.7)の右辺第二項は，式(3.2)より，

$$\frac{\partial y_i^k}{\partial x_i^k} = f'(x_i^k) \tag{3.8}$$

ここで，出力関数としてシグモイド関数を採用していたとすると，その微分は以下の式(3.9)のようになる．

$$
\begin{aligned}
f(x) &= \frac{1}{1+\exp(-x)} \\
f'(x) &= \left\{\frac{\exp(-x)}{1+\exp(-x)}\right\}\left\{\frac{1}{1+\exp(-x)}\right\} \\
&= \frac{1+\exp(-x)-1}{1+\exp(-x)}\frac{1}{1+\exp(-x)} \\
&= \{1-f(x)\}f(x)
\end{aligned}
\tag{3.9}
$$

式(3.7)の第一項は，y_i^k が出力層の出力値の場合は，

$$
\frac{\partial E^k}{\partial y_i^k} = \frac{1}{2}\frac{\partial \sum_j (y_j^k - d_j^k)^2}{\partial y_i^k} = y_j^k - d_j^k \tag{3.10}
$$

したがって出力層ユニットの誤差信号δ_i^kは，

$$
\delta_i^k = (y_i^k - d_i^k) f'(x_i^k) \tag{3.11}
$$

次に，式(3.7)の第一項は，y_i^k が隠れ層の出力値の場合は，

$$
\frac{\partial E^k}{\partial y_i^k} = \sum_j \frac{\partial E^k}{\partial x_j^k}\frac{\partial x_j^k}{\partial y_i^k} \tag{3.12}
$$

ここで，右辺第一項はユニット j の誤差信号δ_j^kと，右辺第二項は式(3.1)より w_{ji} となるので，隠れ層ユニットの誤差信号は式(3.12)のようになる．

$$
\delta_i^k = f'(x_i^k)\sum_j \delta_j^k w_{ji} \tag{3.13}
$$

すなわち，誤差信号δ_i^k が結合加重 w_{ji} を解して逆伝播させることで，再帰的に隠れユニットの誤差信号が求まる．したがって，何層ネットワークの場合でもこのアルゴリズムは適用できる．

最終的に，結合加重の修正量は，式(3.13)のように表現できる．

$$
\Delta w_{ij} = -\eta \delta_i^k y_j^k \tag{3.14}
$$

なお，バイアス値は，式(3.1)より結合加重 1 で結合されたユニットと考えることが出来るので，結合加重と同様の方法で更新可能である．

実際の学習には，慣性項を用いた以下の式を用いる．

$$\Delta w_{ij}(t) = -\eta \delta_i^k y_j^k + \alpha \Delta w_{ij}(t - \Delta t) \tag{3.15}$$

αは慣性項係数で，[0, 1]の範囲で選択される．慣性項を用いることで，結合加重の修正が振動的なときや，修正量Δw の変化が少ないときに学習速度を大きくする効果がある．

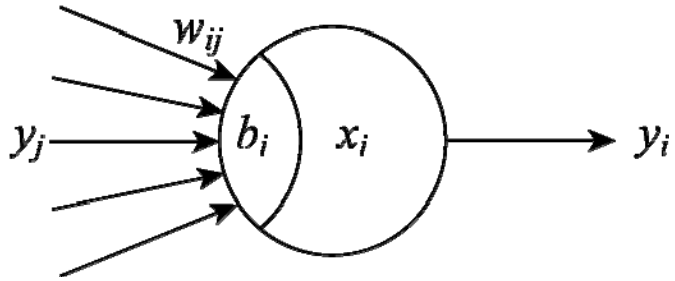


Fig 3.1 Basic Neuron Model

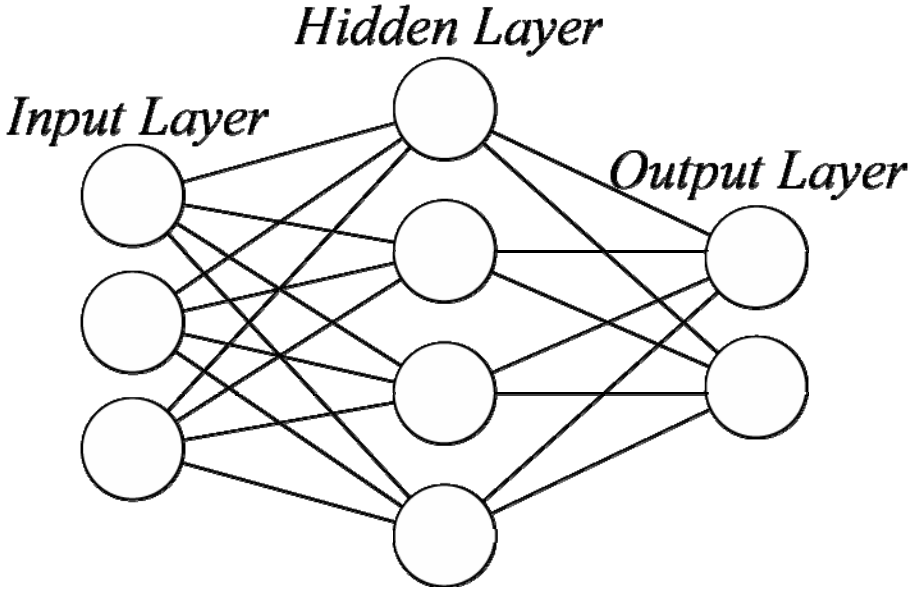


Fig 3.2 Three Layers Neural Network

3.2 自己組織化マップ

自己組織化マップ（SOM: Self-Organizing Map）は Kohonen らによって提案された脳の学習アルゴリズムを模擬した教師無し学習ニューラルネットワークである．多次元ベクトルの入力データ群を自己組織化し，二次元平面で表現することができる．Fig3.3 に SOM の構造の概念図を示す．Fig3.3 の左側，多次元空間上に多次元の要素をもった入力ベクトルが広がっているとき，任意の多次元ベクトル群を 1 または 2 次元へ低次元化する写像関係を得る，データの傾向を低次元化された競合層(Fig3.3 右側)で幾何学的に評価する等が SOM の大きな特徴である．入力空間での入力ベクトル間の位置関係は低次元化された出力空間でも保存されており，学習の過程において，類似した特徴を持つデータが幾何学的に近い位置に配置されクラスタリングされる位相保持，入力データ間の中間の値を得ることが出来る内挿補間という特徴を持つ．

3.2.1 SOM のアルゴリズム

SOM のアルゴリズムは評価過程，競合過程，協調過程，適応過程の 4 つのプロセスからなっていると考えると理解しやすい．各プロセスの内容を含め以下に SOM アルゴリズムを述べる．

評価過程

入力空間の M 個のデータがランダムに選択され，また，データ次元が N で与えられたとすると，入力パターンベクトル $\boldsymbol{x}_i$ は以下のようにあらわされる，

$$\boldsymbol{x}_i = (x_{i1}, x_{i2}, ...x_{iN}), \quad i = 1,2,...,M \tag{3.15}$$

出力層内の各ユニットの重みベクトルは基本的には入力空間と同じ次元となる．出力層に K 個のユニットがあり，k 番目のユニットの重みベクトルを $\boldsymbol{w}^k$ とあらわすこととする．

$$\boldsymbol{w}^k = \left(w_1^k, w_2^k, ..., w_N^k\right), \quad k = 1,2,...,K \tag{3.16}$$

評価関数 E は $\boldsymbol{x}$ と $\boldsymbol{w}$ のユークリッド距離を用いて以下のように表される．

$$E_i^k = \left\| \boldsymbol{x}_i - \boldsymbol{w}^k \right\| \tag{3.17}$$

E は $\boldsymbol{x}$ と $\boldsymbol{w}$ の類似度をユークリッド距離として表現している．他に $\boldsymbol{x}$ と $\boldsymbol{w}$ 内積を用いる方法もある．

競合過程

入力ベクトル $\boldsymbol{x}_i$ に対して，評価関数 E_i^k を最小にする $\boldsymbol{w}^k$ のインデックスを k_i^* とすると，競合過程は以下のように表される．

$$k_i^* = \arg\min_k \left(E_i^k\right), \quad k = 1,2,...,K \tag{3.18}$$

ここで，k^* の示す競合層のユニットを勝者ユニット(BMU: Best Matching Unit)と呼ぶ．

協調過程

BMU が得た学習権利を近傍ユニットに分配するために近傍関数 $h(.)$ の定義をする．関数 $d(k, k^*)$ は BMU と各ユニットとの距離を表すとすると，近傍関数は学習が収束するために，以下のような 2 つの条件を満たす必要がある．

① 近傍関数 $h(\ )$ は $d(\ ) = 0$ となる点について対称で，$d(\ ) = 0$ のとき最大値 1 をとる．

② 近傍関数 $h(\ )$ は，単調減少であり，$d(\ ) \rightarrow \infty$ において 0 へ収束する．

これらの条件を満たす関数として，例えばガウス関数を用いて，近傍関数 $h(\ )$ を以下のように定義する．

$$h_i^k(t) = \exp\left\{-\frac{d(k,k_i^*)^2}{2\sigma(t)^2}\right\} \tag{3.19}$$

$\sigma(t)$は近傍領域と呼ばれ，適応過程において各ユニットが影響を受ける BMU の学習の影響範囲を示している．競合層において BMU の学習の影響は BMU を中心に近傍ユニットへ広がり， BMU からの距離が離れるとその影響は小さくなる．

また，SOM のアルゴリズムの特徴のひとつは，時間とともに近傍領域を減少させることにある．これを指数関数的に減少させたとすると，

$$\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\tau_1}\right) + \sigma_{\min}, \quad t = 0, 1, 2, \ldots \tag{3.20}$$

σ_0 は初期値，$\sigma_{\min}$ は最終値，τ_1は時定数である．したがって，学習回数 t が増加すると$\sigma(t)$は指数関数的に減少し，それに伴って近傍領域が減少する．

適応過程

この過程において，競合層のユニットの重みベクトルを入力ベクトルに近づける．式(3.17)において定義した評価関数 E は，式(3.21)に示すように $\boldsymbol{x}$ と $\boldsymbol{w}$ の自乗誤差と定義しなおすことが可能である．

$$E = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{w}) \cdot (\boldsymbol{x} - \boldsymbol{w})^T \tag{3.21}$$

この誤差関数を減少させる$\Delta \boldsymbol{w}$ を最急降下法によって求める．
両辺を $\boldsymbol{w}$ で偏微分する．

$$\frac{\partial E}{\partial \boldsymbol{w}} = -(\boldsymbol{x} - \boldsymbol{w}) \tag{3.22}$$

ここで，学習係数をηとすると，

$$\Delta \boldsymbol{w} = -\eta \frac{\partial E}{\partial \boldsymbol{w}} = \eta(\boldsymbol{x} - \boldsymbol{w}) \tag{3.23}$$

$\Delta \boldsymbol{w}$ はある時間 t における，$\boldsymbol{w}$ の更新量を示しているとすると，Δt 後の $\boldsymbol{w}$ の値は以下のように表現できる．

$$\begin{aligned} \boldsymbol{w}(t+\Delta t) &= \boldsymbol{w}(t) + \Delta \boldsymbol{w} \\ &= \boldsymbol{w}(t) + \eta(\boldsymbol{x} - \boldsymbol{w}(t)) \\ &= (1-\eta)\boldsymbol{w}(t) + \eta \boldsymbol{x} \end{aligned} \tag{3.24}$$

式(2.24)は $\boldsymbol{x}$ と $\boldsymbol{w}$ の内分点に $\boldsymbol{w}$ の更新値があることを示しており，繰り返し更新を行うことで $\boldsymbol{w}$ が $\boldsymbol{x}$ に近づくことを示している．この処理をすべての入力に対して行うことで学習が成立する．

また，ここで学習係数は学習回数の増加とともに減少するようにする．例えば，指数関数的に減少させたとすれば,

$$\eta(n) = \eta_0 \exp\left(-\frac{t}{\tau_2}\right), \quad t = 0,1,2,... \tag{3.25}$$

これに協調過程の近傍関数を適用すると，最終的な学習即は以下のようになる．

$$\boldsymbol{w}^k(t+1) = \boldsymbol{w}^k(t) + \eta(t) h_i^k(t)\left(\boldsymbol{x}_i - \boldsymbol{w}^k(t)\right) \tag{3.26}$$

最後にアルゴリズムをまとめると,

初期化：　マップの初期値が学習結果に影響しないように小さな乱数で行う

評価過程：　各入力ベクトルに対して類似度を示す評価関数の値を算出

競合過程：　評価関数の値が最大(最小)となる勝者ユニットを決定

協調過程：　各ユニットへの学習分配率を近傍関数により決定

適応過程：　学習則に従ってユニットの値を修正

2)～5)の過程を学習が収束するまで繰り返すことで学習が成立する．

入力ベクトルを選択するたびに 2)~5)を繰り返す手法は逐次型の学習アルゴリズムである．一方，予めすべての入力ベクトルに対して評価，競合過程を適用した後，適応過程によって各ユニットの値を修正する手法はバッチ型学習 SOM（BL-SOM: Batch Learning SOM）と呼ばれている． BL-SOM では近傍関数の値をすべての入力ベクトルに対して正規化し以下のような学習分配率$\Psi_i^k(t)$を定義する．

$$\Psi_i^k = h_i^k(t) \Big/ \sum_i h_i^k(t) \tag{3.27}$$

この学習分配率を用いて学習即を以下のように定義される．

$$\boldsymbol{w}^k(t+1) = \boldsymbol{w}^k(t) + \eta\left(\sum_i \Psi_i^k \boldsymbol{x}_i - \boldsymbol{w}^k(t)\right) \tag{3.28}$$

一般的に BL-SOM ではηを 1 とするので,

$$\boldsymbol{w}^k(t+\Delta t) = \sum_i \Psi_i^k \boldsymbol{x}_i \tag{3.28}$$

これによって，逐次型学習 SOM において BMU 及びその近傍付近が大きな学習量を得るアルゴリズムが改善され，すべてのユニットが一様な学習量を得るようになり，学習の速度と安定性を向上させることができる．

3.2.2 SOM の適用例

SOM の具体的適用例を挙げて SOM の挙動や学習後得られる情報について述べたい．ここでは，SOM の適用例としてよく用いられる動物の分類試験について述べる．Table3.1 に示すように，17 種類の動物を 21 の特徴によってパラメータを付けてあり，値は[0, 1]である．学習のパラメータとしては，競合層は，ユニットサイズ 10x10 で六角格子状の配置にした．評価過程における評価関数はユークリッド距離とし，近傍関数は，初期近傍 5，初期学習係数 0.5，学習回数 50 回で，以下のように設定した．Fig3.3 に競合層の状態遷移を学習回数 1，2，3，4，5，10，100，200，300，400 回のマップを示している．図中の正方形が競合層上のベクトルユニットを示している．各動物を表現した入力ベクトルを最もよく表現しているユニット(BMU)にはその動物の種類が書かれている．各正方形の色は特徴要素”small”，”middle”，”large”のパラメータに対してそれぞれ 0.0，0.5，1.0 の重みをかけた[0,1]のパラメータ変化を，青→緑→赤という色で示している．1 回目から 5 回目の学習の初期段階では，右上に哺乳類，左下に鳥類と分類されている．10 回目を見ると，哺乳類では，右上を見ると大型草食，中央上に大型肉食，右下に中型肉食という風になっている．また，鳥類では，左上に猛禽類，右下に装飾の鳥類といった風に哺乳類，及び鳥類の中での分類が細分化されてきている．100，200，300 回と進むにつれさらに細分化が進み，最終的

に 400 回には Fig3.4-(j)のようなマップが得られた．学習回数 400 回付近からそれ以降では競合層における各動物の配置に変化はなかったので，ここで学習終了とした．Fig2.5 には最終的に得られたマップを示している．図中の各点が競合層のベクトルユニットを示しており，Fig3.4 と同様に，各入力に対する BMU にはその動物の種類が記載されている．六角形の色は各ベクトルユニット間のユークリッド距離を明暗で示しており，距離が近ければ白，遠いと黒で表現されている．

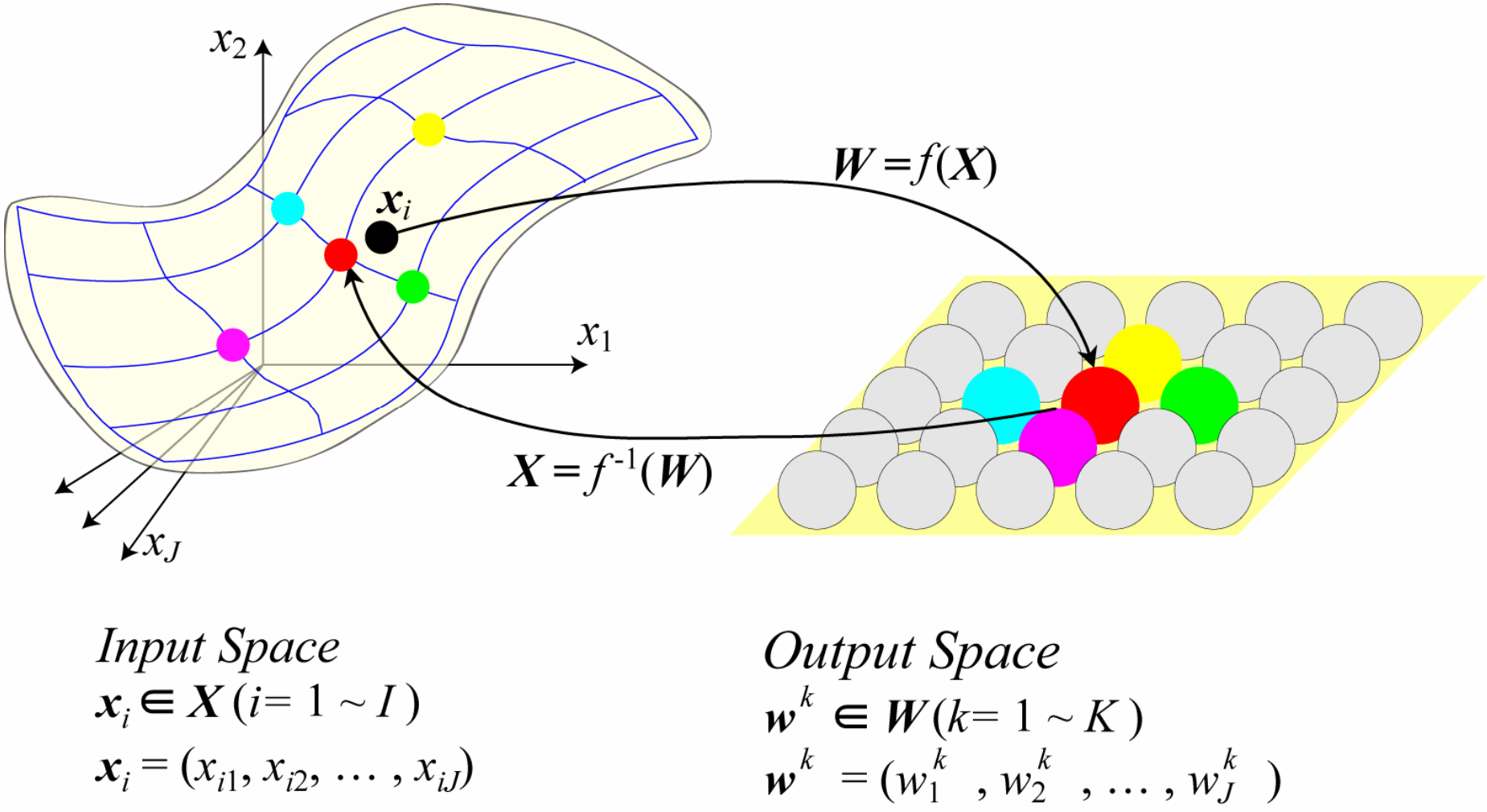
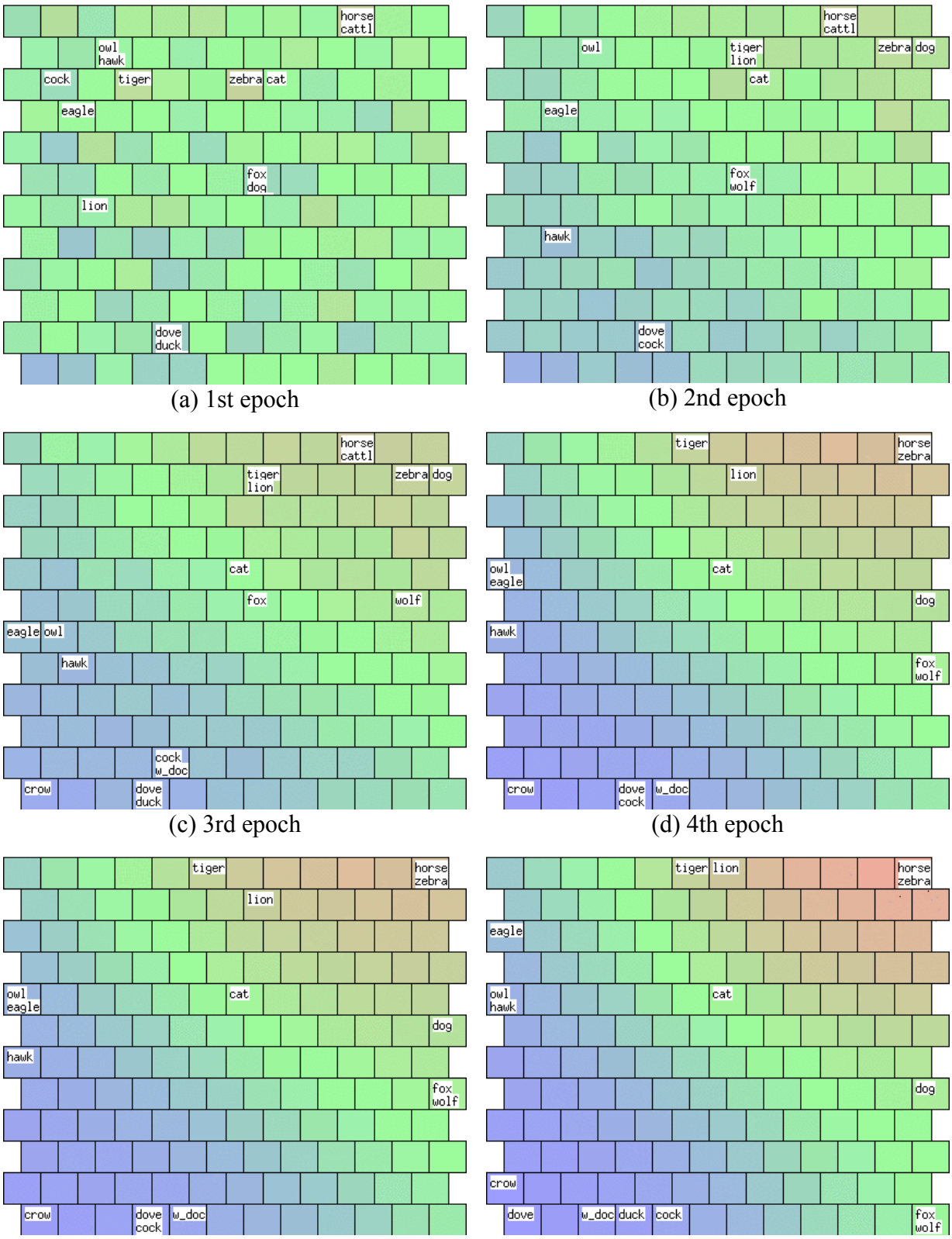


Fig 3.3 The Architecture of Self-Organizing Map

Table 3.1 The Characteristic Parameter of the Animals

	small	middle	large	nocturnality	2 lges	4 legs	fur	hoof	hair	feather	striped
dove	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
cock	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
duck	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
w_duck	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.3
owl	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
hawk	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
eagle	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
crow	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
fox	0.0	1.0	0.0	0.5	0.0	1.0	1.0	0.0	0.0	0.0	0.0
dog	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0
wolf	0.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0
cat	1.0	0.0	0.0	0.5	0.0	1.0	1.0	0.0	0.0	0.0	0.0
tiger	0.0	0.0	1.0	0.5	0.0	1.0	1.0	0.0	0.0	0.0	1.0
lion	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0
horse	0.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0
zebra	0.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0
cattle	0.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0

	hunting	run	fly	swim	livestock	herbivore	carnivore	Canidae	Panthera	pet
dove	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
cock	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0
duck	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0
w_duck	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0
owl	0.0	0.0	1.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0
hawk	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
eagle	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
crow	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
fox	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0
dog	0.0	1.0	0.0	0.0	0.0	0.5	0.5	1.0	0.0	1.0
wolf	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0
cat	1.0	0.0	0.0	0.0	0.0	0.5	0.5	0.0	1.0	1.0
tiger	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0
lion	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0
horse	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0
zebra	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
cattle	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0

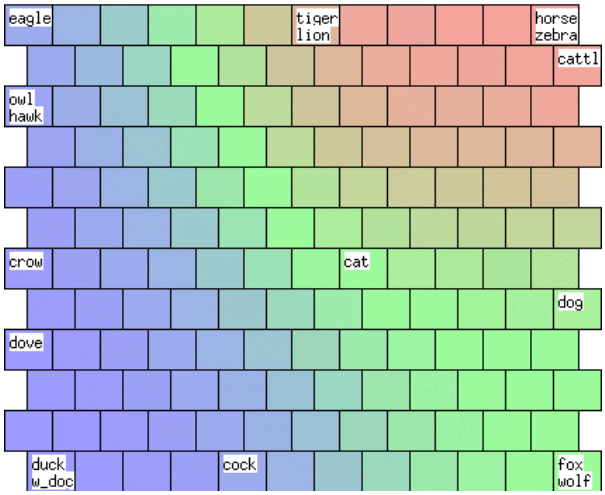


(a) 1st epoch (b) 2nd epoch

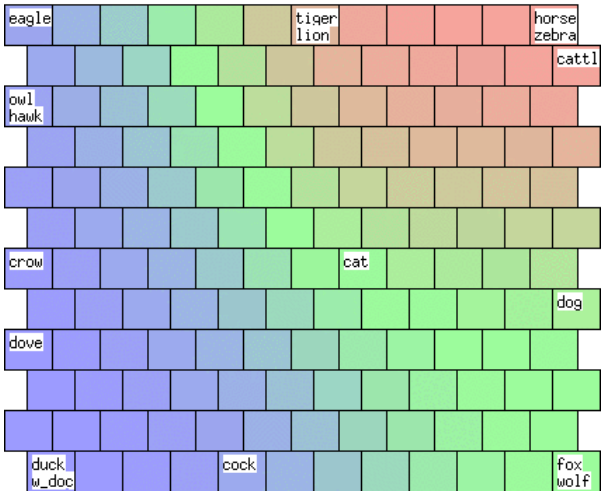
(c) 3rd epoch (d) 4th epoch

(e) 5th epoch (f) 10th epoch

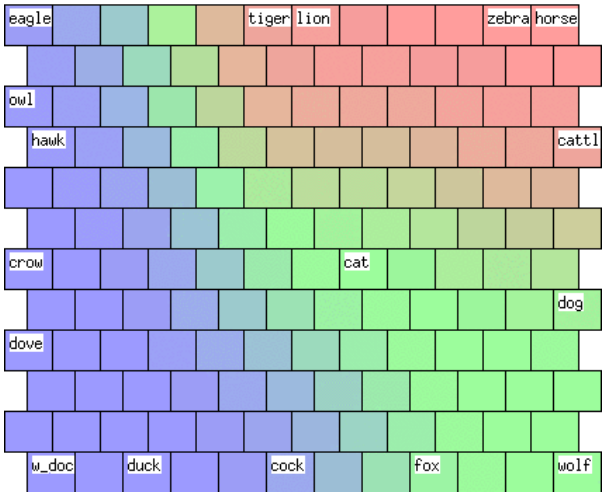
Fig 3.4 Transition of competitive layer during learning



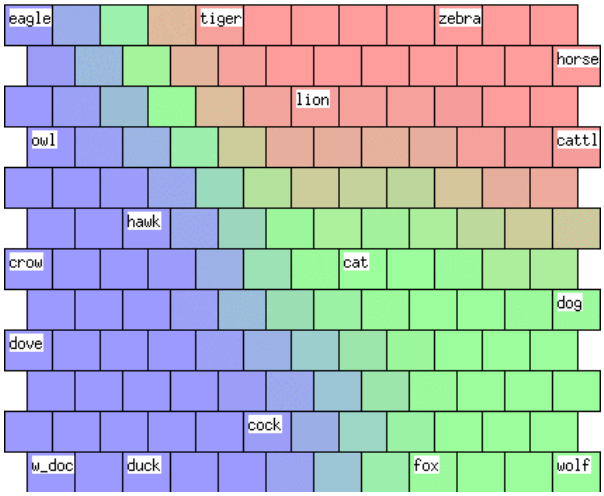
(g) 100th epoch



(h) 200th epoch



(i) 300th epoch



(j) 400th epoch

Fig 3.4 Transition of competitive layer during learning

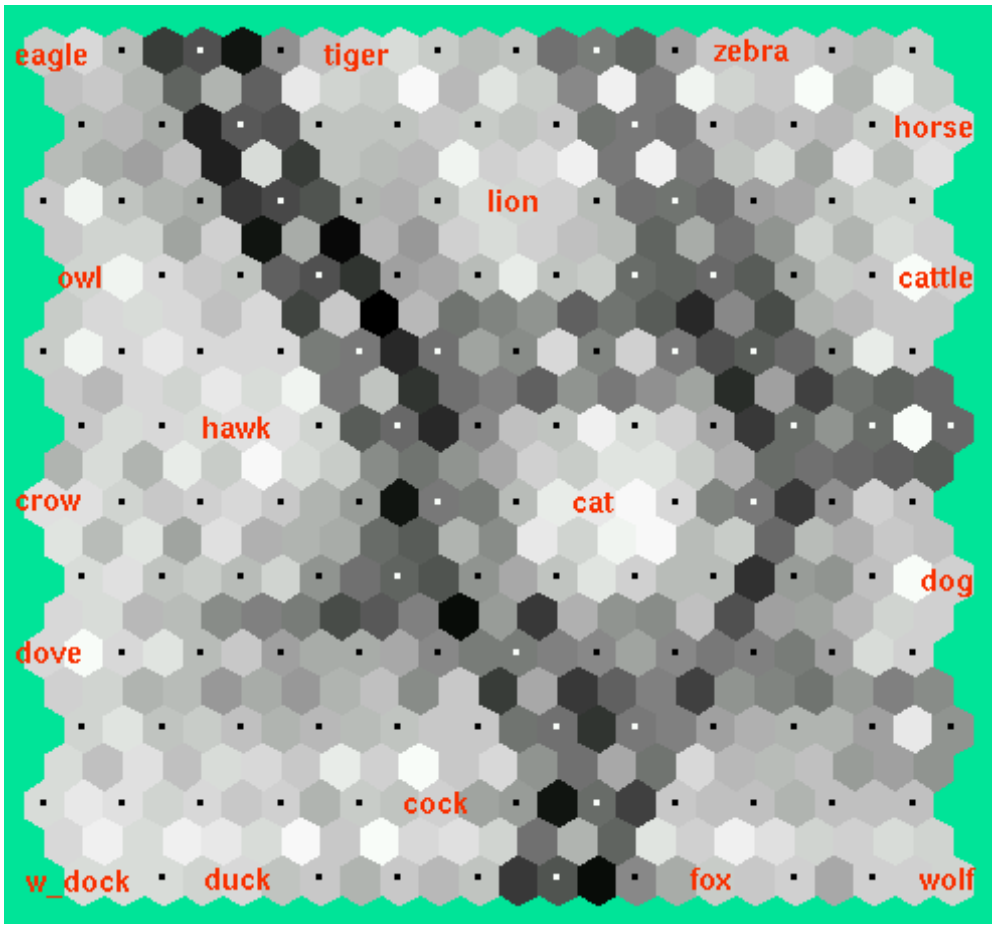


Fig 3.5 Feature Map of animals

3.3 モジュラーネットワーク型自己組織化マップ

3.3.1 mnSOM の概要

mnSOM の概念図を Fig3.6 に示す．mnSOM は，入力空間において複数のデータクラスが有する入出力の写像関係を競合層のモジュールとして表現することが可能な情報処理システムである．競合層には通常の SOM と同様に，例えば六角格子状や正方格子状にモジュールを配置する．学習の過程で入力空間のデータクラスが表現している関係を教師有り学習によって獲得しながら，並行してクラス間の評価関数から算出された類似度を基に教師無し学習によって類似した特徴を有するクラスは近接し，特徴が異なるクラスは遠距離に配置された競合層マップを得ることができる．

Fig2.6 の左図に示される $F_1(\boldsymbol{x})$と $F_2(\boldsymbol{x})$は，入力空間におけるある入出力関係を表現している．学習後，Fig.3.6 の右図に示されるような競合層が得られたとする．競合層において，$F_1(\boldsymbol{x})$と $F_2(\boldsymbol{x})$の中間に位置するモジュールは，入力空間においても $F_1(\boldsymbol{x})$と $F_2(\boldsymbol{x})$の中間に位置する関数を表現しており，入力空間での幾何学的関係が競合層にも保存されたマップが得られる．mnSOM の適用例として関数のパラメータ推定，非線形主成分解析，動的システムのマッピング，自己組織化適応制御器などの有効性が報告されている[50]-[53].

3.3.2 mnSOM のアルゴリズム

mnSOM の学習アルゴリズムは，モジュールとして何を採用したかによって異なるが，ここでは mnSOM のモジュールに階層型ニューラルネットワークを用いた場合についての学習のアルゴリズムについて述べる．次章で述べるフォワードモデルマップ及びコントローラマップの学習には以下のアルゴリズムを用いている．

I 個のデータクラス$\{D_1, D_2, \ldots, D_I\}$があり，それぞれ J 個の入出力対$(\boldsymbol{x}_{ij}, \boldsymbol{y}_{ij})$を持っており関数 f_i を構成しているとすると，式(3.29)，(3.30)の関係が成り立つ．

$$\boldsymbol{D}_i = \{(\boldsymbol{x}_{i1}, \boldsymbol{y}_{i1}),\ \ (\boldsymbol{x}_{i2}, \boldsymbol{y}_{i2}),\ \ \ldots,\ \ (\boldsymbol{x}_{iJ}, \boldsymbol{y}_{iJ})\} \tag{3.29}$$

$$\boldsymbol{y}_{ij} = f_i(\boldsymbol{x}_{ij}) \tag{3.30}$$

一方，mnSOM のモジュールが K 個あり，k 番目のモジュールの持つニューラルネットワークの結合加重ベクトルを $\boldsymbol{w}^k$ とすると，それぞれ，式(3.31)に示しているような関数 g^k を満たす.

$$\hat{\boldsymbol{y}}_{ij} = g^k(\boldsymbol{x}_{ij}) = g(\boldsymbol{x}_{ij}, \boldsymbol{w}^k) \tag{3.31}$$

このとき，学習は以下のように行われる.

i) 初期化

各モジュールの $\boldsymbol{w}^k$ をランダムに初期化する.

ii) 評価過程

式(3.32)に従って， i 番目のクラス D_i の入力データ $\boldsymbol{x}_{ij}$ に対する k 番目のモジュールの評価値 $E_i(k)$を求める． E_i^k は入力クラス D_i の写像関係 $f_i(\ )$と $g^k(\ )$の関数間の距離であり，入力クラス D_i と k 番目のモジュールの類似度を意味する．各モジュールにおいて，全てのクラス D_i に対する類似度 E_i^k を計算する.

$$\begin{aligned} E_i^k &= \frac{1}{J}\sum_{j=1}^{J}\left\| f_i(x_{ij}) - g^k(x_{ij}) \right\|^2 \\ &= \frac{1}{J}\sum_{j=1}^{J}\left\| \boldsymbol{y}_i - \hat{\boldsymbol{y}}_{ij} \right\|^2 \end{aligned} \tag{3.32}$$

iii) 競合過程

式(3.33)に従って，入力クラス $\mathrm{D_i}$ に対する評価値（類似度）E_i^k が最小となる競合層モジュールを勝者モジュールとし，入力クラス D_i の勝者モジュールのインデックス（モジュール番号）を k_i*とする.

$$k_i^* = \underset{k}{\operatorname{argmin}}\ E_i^k \tag{3.33}$$

iv) 協調過程

勝者モジュールからの距離および学習回数によって，学習分配率Ψ_i^k を決定する．Ψ_i^k は k 番目のモジュールが D_i の入出力関係を学習する割合を示す．関数 $h(\ )$は近傍関数であり，競合層における勝者モジュールと各モジュールとの配置から計算される幾何学的な距離 $d(k, k_i^*)$及び学習回数 t の増加に伴って単調減少する関数を選択する．

$$\Psi_i^k = \exp\left\{-\frac{d(k,k_i^*)^2}{2\sigma(t)^2}\right\} \Big/ \sum_{i'=1}^{I} \exp\left\{-\frac{d(k,k_{i'}^*)^2}{2\sigma(t)^2}\right\} \tag{3.34}$$

v) 適応過程

各モジュールにおいて，誤差逆伝播法を用いて結合加重を更新する．式(2.21)に示すように学習率ηと学習分配率Ψ_i^k を用いて $\boldsymbol{w}^k$ を更新する．ここで，$\eta\Psi \rightarrow \eta$と置き換えると更新式は一般的な誤差逆伝播法と同様となる．

$$\Delta \boldsymbol{w}^k = -\eta \sum_i \left(\Psi_i^k \frac{\partial E_i^k}{\partial \boldsymbol{w}^k} \right) \tag{3.35}$$

*ii)～v)*を学習が収束するまで繰り返すことで，mnSOM の学習は実現される．

3.3.3 mnSOM の適用例

3 次までの Legendre 正規化関数を用いて 6 種の三次元多項式を生成し，mnSOM によって分類実験を行った．

Legendre 正規化関数は，(3.36)のように定義されており，1 次，2 次，3 次の順に F_1，F_2，F_3 と表している．

$$\begin{aligned} F_1 &= x \\ F_2 &= \left(3x^2 - 1\right)/2 \\ F_3 &= \left(5x^3 - 3x\right)/2 \end{aligned} \tag{3.36}$$

これらの Legendre 正規化関数に式(3.37)に示すように係数をかけて 6 種の 3 次多項式を作成した．作成した 6 種の多項式を Fig2.7 に示している．

$$\begin{pmatrix} P1 \\ P2 \\ P3 \\ P4 \\ P5 \\ P6 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.5 & 0.5 \\ -0.5 & -0.5 & -0.5 \\ 1.0 & 0.0 & 0.0 \\ -1.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 1.0 \\ 1.0 & 0.0 & -1.0 \end{pmatrix} \begin{pmatrix} F_3 \\ F_2 \\ F_1 \end{pmatrix} = \begin{pmatrix} \left(5x^3 + 3x^2 - 3x - 1\right)/4 \\ -\left(5x^3 + 3x^2 - 3x - 1\right)/4 \\ \left(5x^3 - 3x\right)/2 \\ -\left(5x^3 - 3x\right)/2 \\ x \\ -x \end{pmatrix} \tag{3.37}$$

学習には 6 種の 3 次関数集合から，SOM には Legendre 正規化基底関数に対する係数を入力とし，101 個のデータを 6 種の関数からサンプリングし，入力信号として mnSOM へ与えた．学習パラメータは SOM を用いた場合と mnSOM を用いた場合ともにマップサイズ 10x10，初期近傍$\sigma_0 = 10$，最小近傍$\sigma_{min} = 2$，時定数$\tau_0 = 50$ を用いている．また，mnSOM は入力層:1，中間層:5，出力層:1 の 3 層の階層型ニューラルネットワークをモジュールとした．

Fig3.8 には，mnSOM，SOM で学習した得られた競合層における結果を，Fig3.9 には，Legendre 展開パラメータ空間に学習結果をプロットしたものである．ここで，b_i はそれぞれ Legendre 正規化基底関数 F_i に対する係数を示している($i = 1 \sim 3$)．Fig3.8, 3.9 には，(a)は多項式関数の集合を mnSOM でマッピングした場合，(b)は SOM で係数をマッピングした場合に得られた結果を競合層及びパラメータ空間上に示している．Fig3.4 から競合層上の各関数配置された位置は一致していることが確認できる．また，パラメータ空間上において，mnSOM の各モジュールが表現している Legendre 展開パラメータと SOM によって得られたパラメータマップはほぼ一致していることが確認できる．このような結果が得られることは，参考文献[51]によって報告されており，同様の結果が得られることが確認できた．

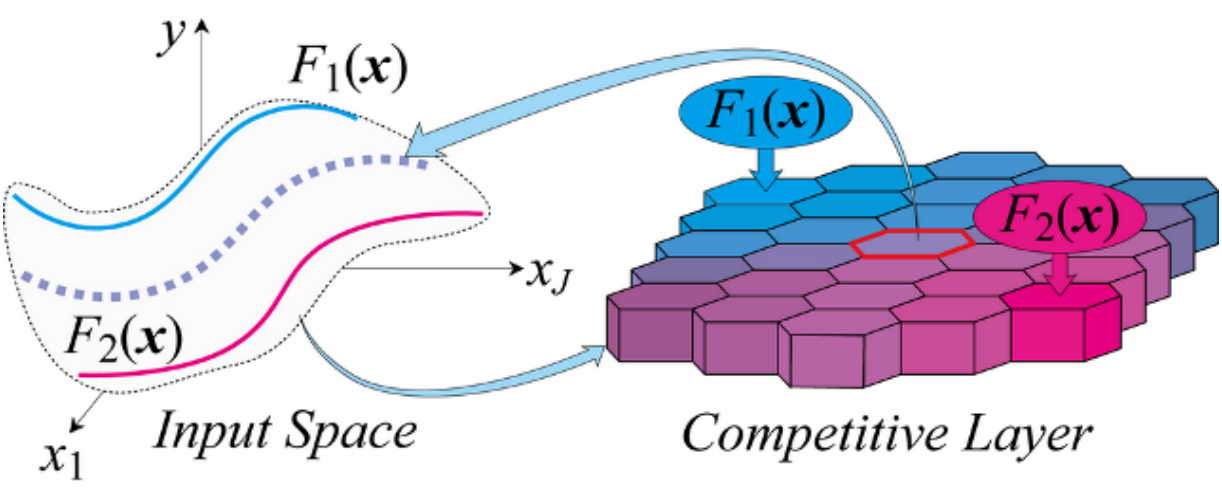


Fig 3.6 Architecture of mnSOM

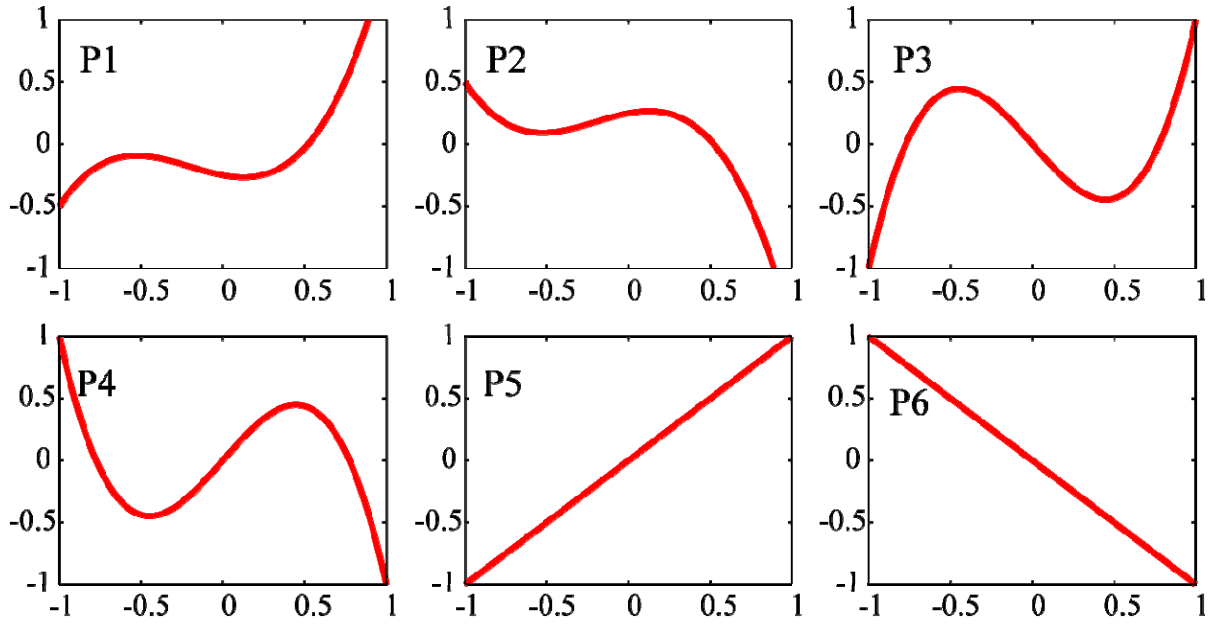
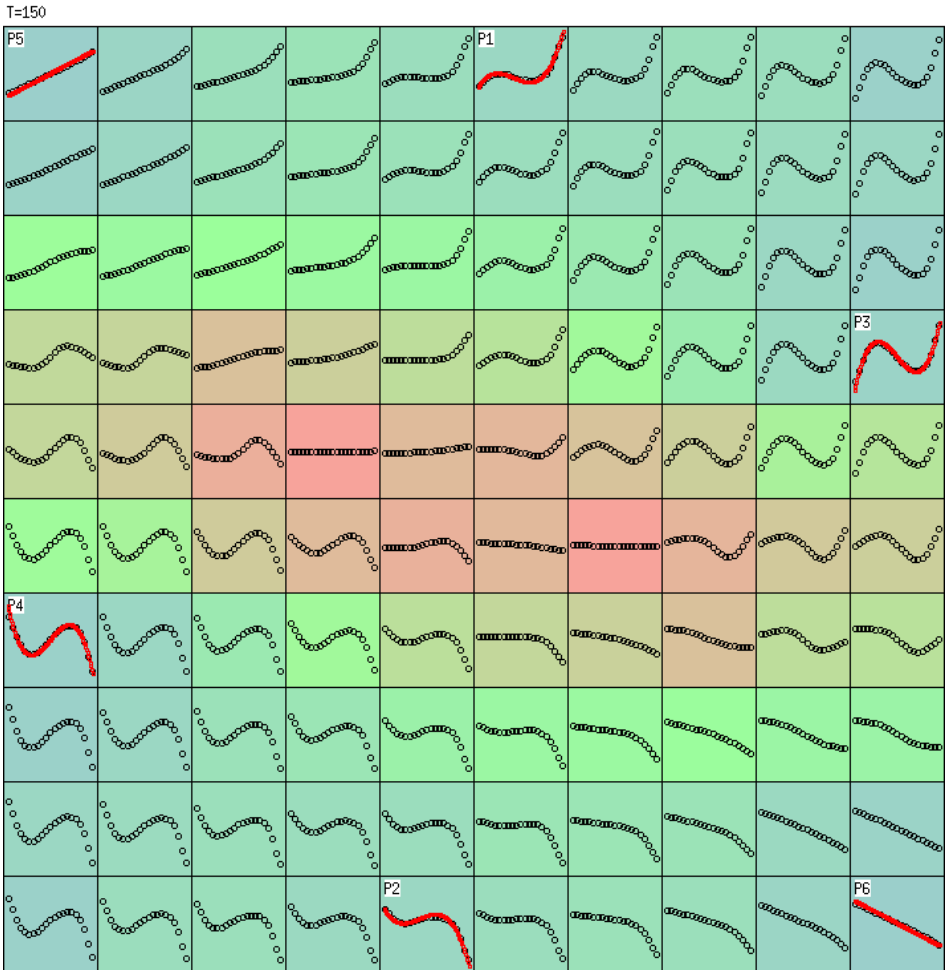
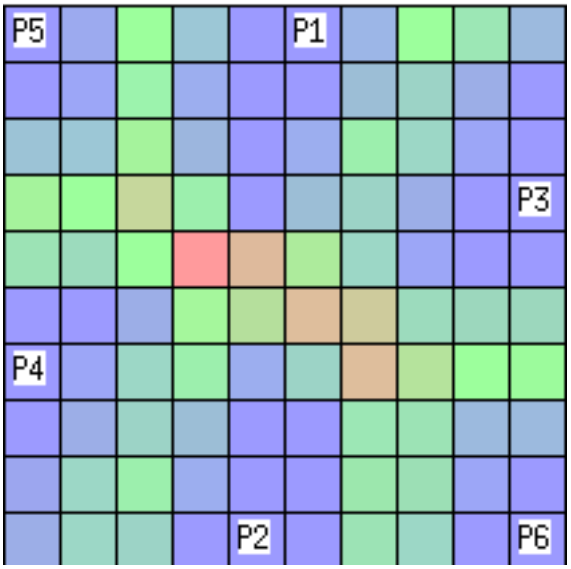


Fig 3.7 Cubic Function Family



(a) A Map created by mnSOM



(b) A Map created by SOM

Fig 3.8 Cubic Functions Map

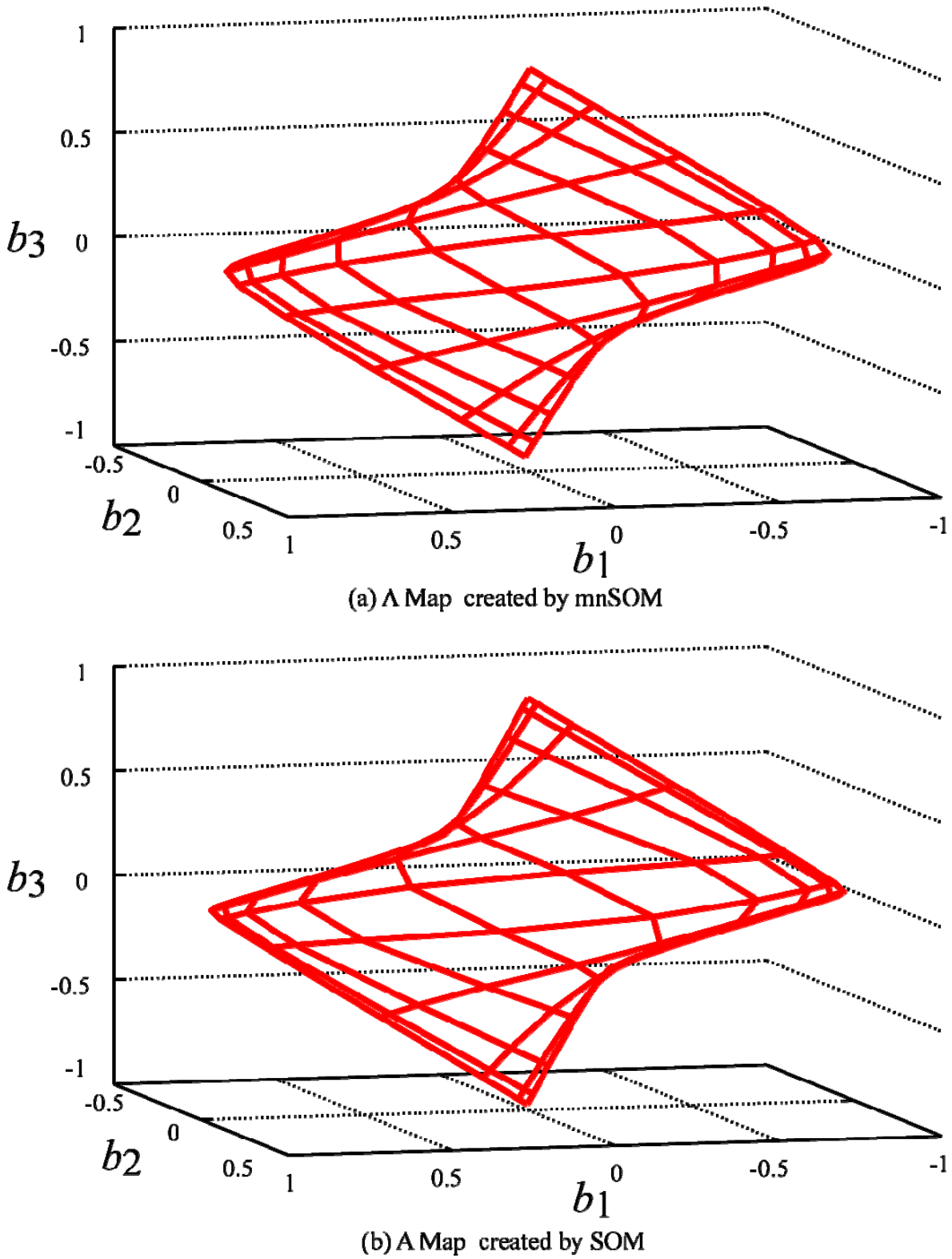


Fig 3.9 The Feature Map of Cubic Functions in the Coefficient Space of Legendre Expansion

第4章

自己組織的行動獲得システム

4.1 SOM を用いた環境認識と行動決定

ロボットやその頭脳に当たるコンピュータは，決まった作業，処理の繰り返しにおいては人間よりもはるかに優れた能力がある．しかし，状況に応じて変化する要素が含まれるときや，曖昧な情報を含むときには，その処理の能力は人間に遠く及ばない．ロボットに対して柔軟な処理を実装しようとしたときには，生物の情報処理メカニズムにヒントを得た考え方，ソフトコンピューティングなどの考え方が有効であると考えられる．

自律型水中ロボットの特長を活かして大いに活用するためには，外部からのサポート技術の研究開発も重要であるが，実機自体の自律性の向上が第 1 に望まれる．自律型水中ロボット実現のための課題には運動制御，環境認識，自己位置同定などがある．ロボットは得られたセンサ情報を基にして，自身の状態の把握，動特性の同定を行うのはもちろんのこと，環境の変化なども認識し，行動を決定することが望ましい．

水中ロボットが環境を認識するためには，複数のセンサ情報から特徴を見つけ出し，状態や環境を認識するために必要な情報を抽出する必要がある．本節では自己組織化マップの低次元化，特徴抽出などの能力に着目し，SOM を用いた水中ロボットの環境認識と行動決定システムとして，移動ロボットの重要な基本行動である障害物回避手法を提案する．また，移動軌跡を評価し追加学習を行う手法について述べる[54]-[62].

4.1.1 障害物回避への適用

入力データの準備

環境情報とそのとき取るべき望ましい行動は一種の入出力関係にある．距離情報と回避方向の組をひとつのベクトルとして，SOM へ入力することでその関係を得る．

ある方向にある物体を障害物として認識するかどうかを，最大値 r_{max} と最小値 r_{min}，及び i 番目の距離センサのデータ r_i^{data} から式(4.1)を用いて線形に正規化して表現した r_i で定義する．

$$r_i = \begin{cases} 0 & if \quad r_i^{data} < r_{\min} \\ (r_i^{data} - r_{\min})/(r_{\max} - r_{\min}) & if \quad r_{\min} \le r_i^{data} \le r_{\max} \\ 1 & if \quad r_i^{data} > r_{\max} \end{cases} \tag{4.1}$$

実機には Fig4.1 の左図に示すように距離センサが取り付けられており，6 方向の距離情報が得られる．距離情報は連続量であるが，初期マップの作成のための入力データとして，ロボットと障害物との位置関係を単純化し障害物の有無の 2 値とし，6 方向の障害物の有無の組合せの 64 通りの状態を考える．

Fig4.2 には，64 通りの状態に対して人が回避方向を八方向で与えた場合について，Fig4.3 には，以下の式(4.2)によって回避方向を決定した場合について示している．図中の八角形の辺は，太線でその方向に障害物があることを，細線では障害物がないことを表現している．赤矢印は各状態での障害物回避方向である

$$(t_x, t_y) = -\sum_i (1 - r_i)\boldsymbol{e}_i \Big/ \left| \sum_i (1 - r_i)\boldsymbol{e}_i \right| \tag{4.2}$$

$\boldsymbol{e}_i$ は r_i 方向の単位ベクトルである．式(4.2)は障害物からの斥力の合力の方向に回避するということを示している．例えば，38(Fig.4.3 左下)の状態の持つデータは(1 1 1 0 0 0 cos(3π/4) sin(3π/4))となる．以上の 6 つの距離情報と 2 つの目標方向の合計 8 要素を持ったベクトルを，障害物回避を実現するための教示データとし，Fig4.2 に示す 64 種の状態を与えることとした．

一般的な SOM 教師無し学習である．しかし，このように，SOM の入力ベクトルにおいて入出力関係（障害物の方向とそのときの回避すべき方向）を表現することで，入出力写像関係が得られる．

行動決定アルゴリズム

学習結果

実機の距離データ $\boldsymbol{r}^{data}$ とマップ内の k 番目のユニットの距離データ部分 $\boldsymbol{r}^{k}$ の比較を行い，勝者ユニットのインデックスを k^*とする．勝者ユニットは現在のロボットの状態を一番良く表現していることになる．

$$k^* = \underset{k}{\operatorname{argmin}}\left(\left\|\boldsymbol{r}^{k} - \boldsymbol{r}^{data}\right\|\right) \tag{4.3}$$

したがって，勝者ユニットの持つ回避方向ベクトル(t_x, t_y)を取り出し，式(3.4)に示すように算出したθ^{k^*}へ回避すればロボットは障害物を回避できる．

$$\theta^{k^*} = \operatorname{atan}\left(t_y^{k^*} \Big/ t_x^{k^*}\right) \tag{4.4}$$

64 通りの状態と行動の組において，人が回避方向を与えたものを 40x40，数式で回避方向を与えたものを 20x20 のユニットを持つ SOM によってそれぞれ学習した結果を Fig4.4 に示す．図内の数値は各状態の勝者ユニットを，それ以外のユニットは黒点で示している．Fig4.5 はロボットを中心にどちらへ進むべきかを色で表現しており，たとえば，黄緑色の領域では前進，赤色の領域では右後方向へ移動すれば障害物を回避できることを示している．周囲の環境によってロボットの状態がクラスタリングされていることが分かる．

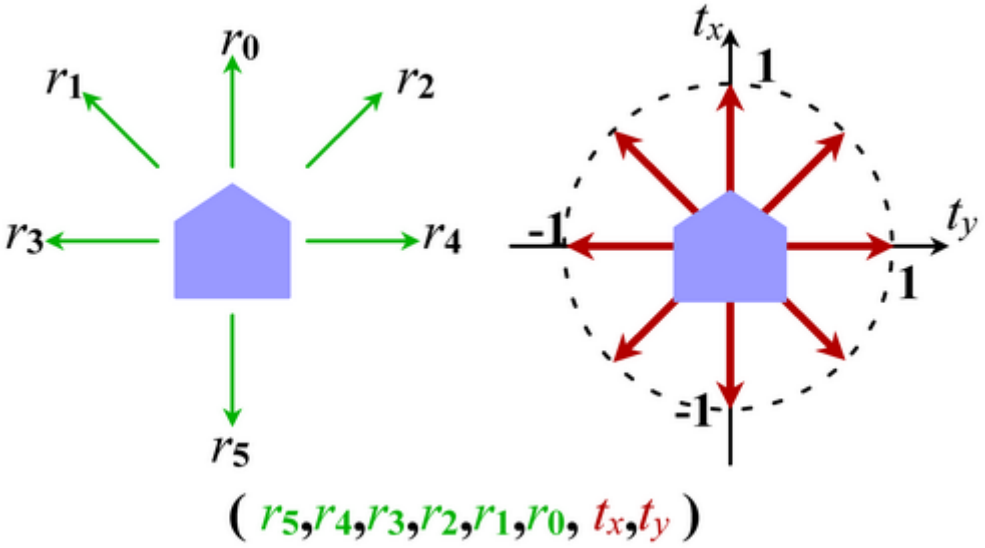


Fig 4.1 Teaching Data for Collision Avoidance

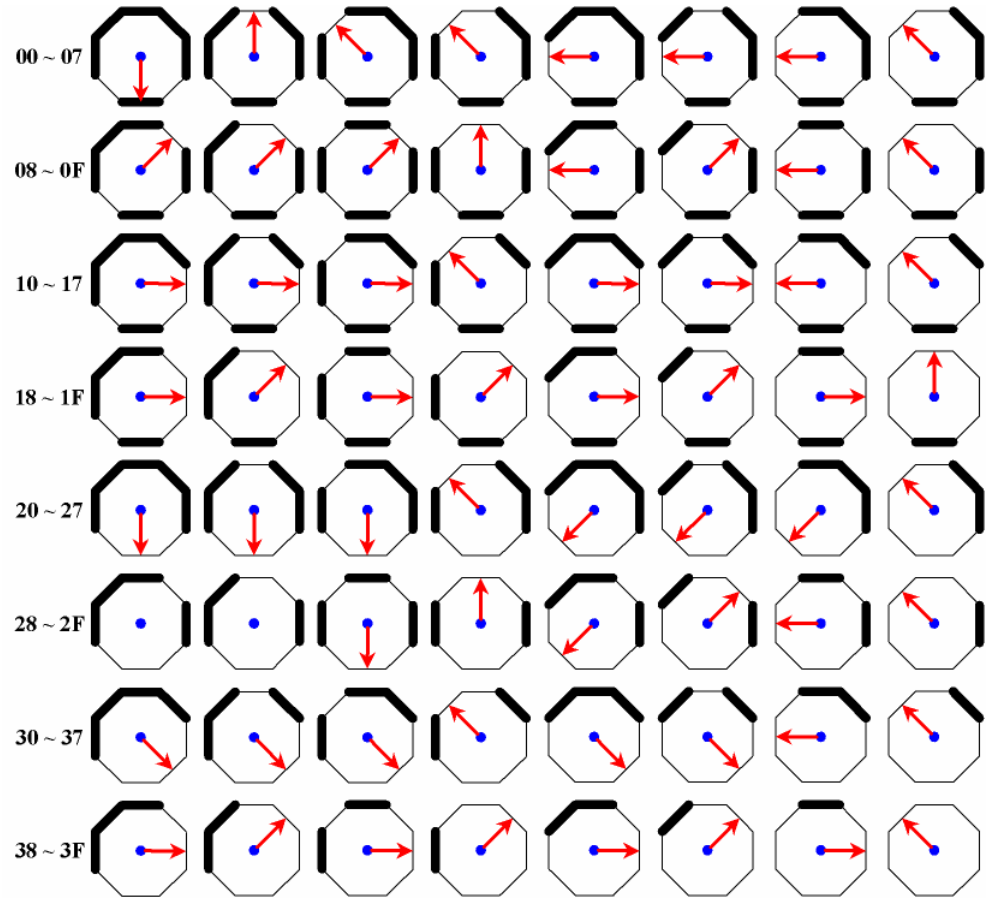


Fig 4.2 Typical Situation between a Robot and obstacles manually

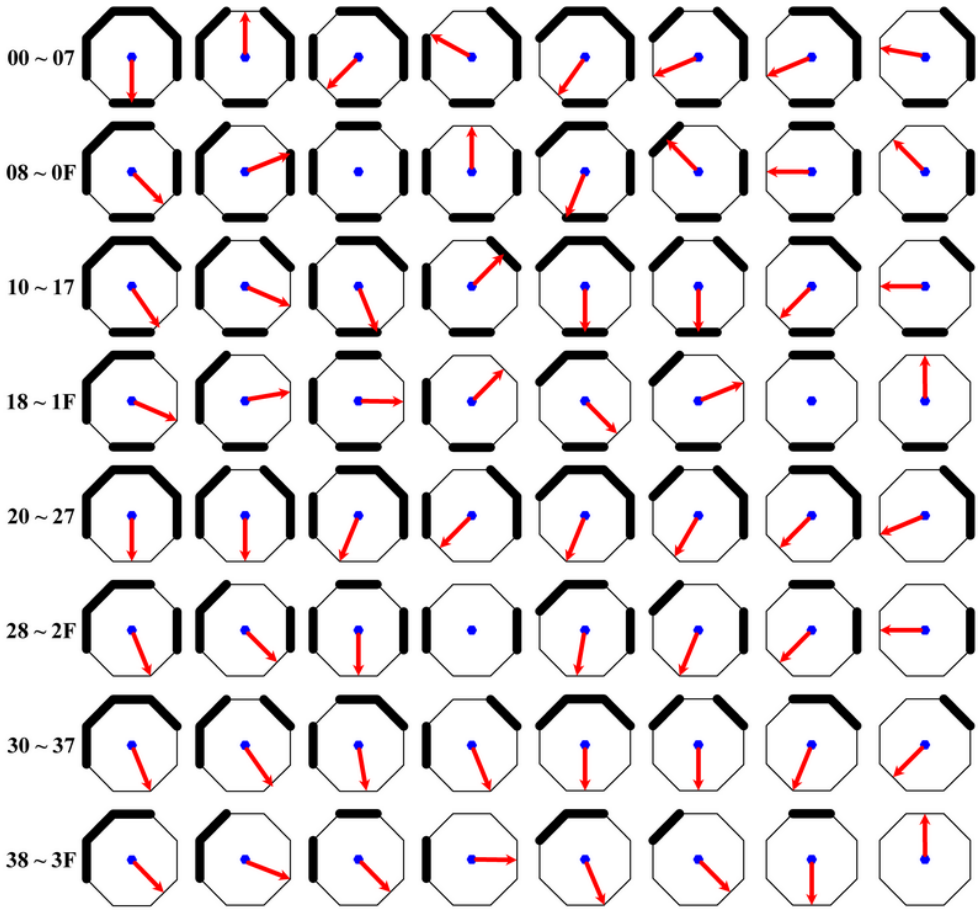
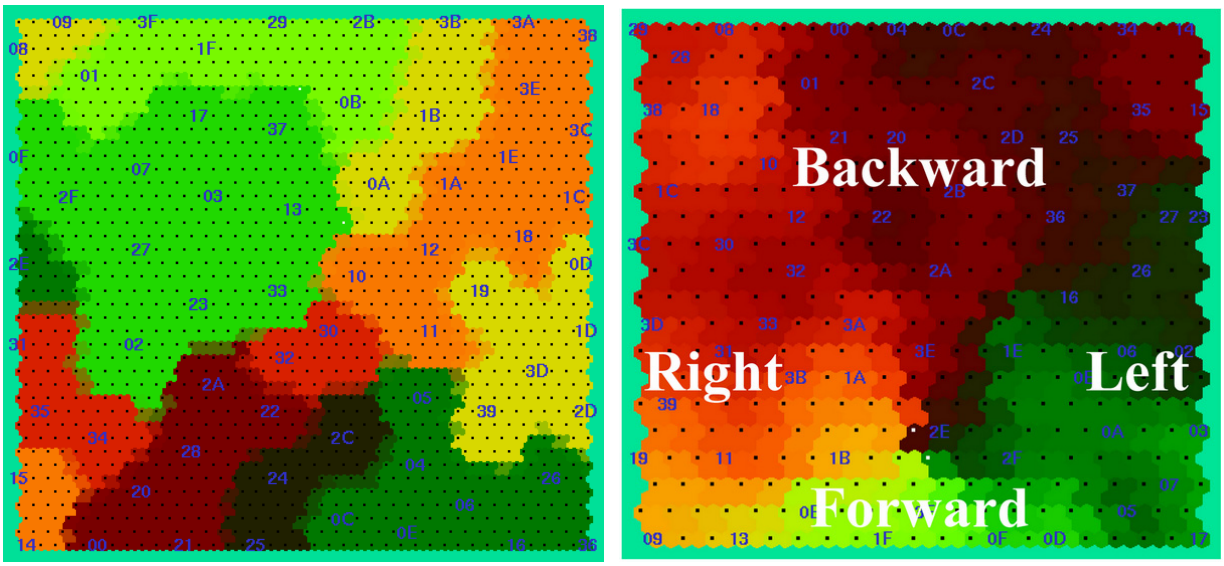


Fig 4.3 Typical Situation between a Robot and obstacles by Equation (4.2)



(a) Manually (b) Equation (4.2)

Fig 4.4 A Map of Surrounding Situations of a Robot and Desired Behavior

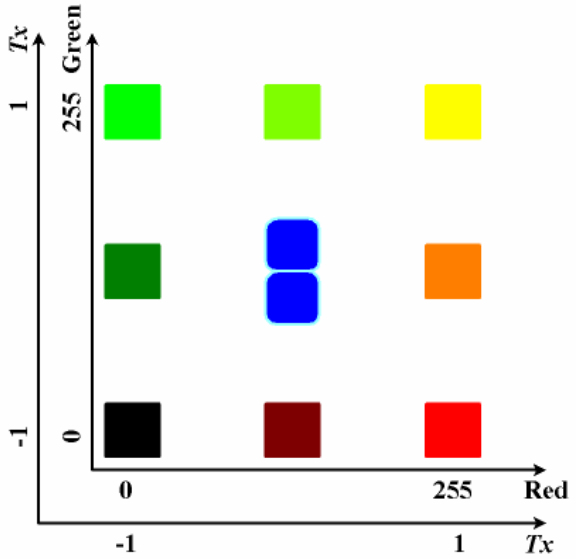


Fig 4.5 The Color corresponding Target Directions

4.1.2 環境情報の追加

行動軌跡の評価値による追加学習

Fig4.4 で得られた結果（以降，初期マップと呼ぶ）には，ロボットの周囲の障害物の情報と，そのときの望ましい回避方向という静的な情報しか含まれていない．多様かつ動特性を考慮した行動の獲得には実機から得られた情報を用いる必要がある．また，同時に環境への適応性を考慮して，初期マップを基にしたロボットの行動を評価して，追加学習をすることを考える．

設定した目的地に滑らかに到着する行動を得られるかどうか，環境適応性の評価試験としてシミュレーションした．

評価関数としては以下のように定義した．

$$E(t) = \tanh\left(Qk_1^{(T-t)} + k_2\left\{1 - \frac{1}{r_{\min}(t)}\right\}\right) \tag{4.5}$$

ここで，k_1，k_2 は定数，T はシミュレーションステップの最大値，ここでは，$r_{\min}(t)$ はそのとき計測された正規化距離データの最小値とする．Q は報酬で，設定した時間内にゴールに到達すれば 1，到達しなければ-1 を与える．この評価関数は，第一項はロボットが早くゴールに到達するほど，また，第二項が障害物から離れて行動しているほど評価値が大きくなることを表現している．

評価値を用いて更新量を以下のように定義する．

$$\begin{cases} \Delta \boldsymbol{w}_x(t) = \eta\left(\boldsymbol{x} - \boldsymbol{w}_x(t)\right) \\ \Delta \boldsymbol{w}_y(t) = \eta\left(\boldsymbol{y} - \boldsymbol{w}_y(t)\right)\cdot E(t) \end{cases} \tag{4.6}$$
$$\boldsymbol{w} = (\boldsymbol{w}_x, \boldsymbol{w}_y), \boldsymbol{x} = (r_5, r_4, r_3, r_2, r_1, r_0), \boldsymbol{y} = (t_x, t_y)$$

ここで，$\boldsymbol{x}$ は計測された超音波センサからのデータ，$\boldsymbol{y}$ はそのときに取った行動を表現している．$\boldsymbol{w}_x$，$\boldsymbol{w}_y$ は各ユニットの持つ重みベクトルのそれぞれ $\boldsymbol{x}$，$\boldsymbol{y}$ に対応する要素を示している．$\boldsymbol{x}$ は実際の状態を表現しているので，$\boldsymbol{x}$ に対しては一般的な SOM と同じように学習する．$\boldsymbol{y}$ に関して，すなわち，行動に関しては Fig4.5 に示すように，行動に対する評価値が正のとき，すなわち，良い行動をとったときは，赤い矢印

で表現しているように行動 $\boldsymbol{y}$ に評価値の大きさに応じてより強く近づく．評価値が負のときすなわち悪い行動をとったときは，青い矢印が示すように行動 $\boldsymbol{y}$ から離れるように学習する．

シミュレーション

修正に使用する教示データは，初期マップとして，人間が直感によって与えた回避方向をもとに作成したもの，及び 4.1.1 項で示した手法によって作成したもの使用した．シミュレーション空間内の初期位置・姿勢(x, y, ϕ)を変えて高度した際に得られた観測データからシミュレーションデータを作成し，サンプルデータとして 111 個のデータを用いて初期マップへの追加学習を行った．Fig4.7 に，評価値の推移の例を示す．一連の行動後に，評価を行いよい行動(Good Case)と判断された状態には近づくように，悪い行動(Bad Case)と判断された行動からは離れるように学習が行われる．初期マップが獲得した幾何学的関係を失わないように，学習回数 100 回，学習率 0.0005，初期近傍 5 で調整した．修正後のマップを Fig4.8 に示す．Fig4.8 では修正された部分の確認が難しいので，初期マップと調整後のマップとの差分を Fig4.9 に示す．Fig4.9 では初期マップから変化のあった部分は暗く表示されている．

調整後のマップを用いたシミュレーション結果を Fig4.10 に示す．修正後のマップを用いた場合の方が，初期マップを用いた場合に比べ，ロボットが滑らかに移動しゴールに到達している．

Fig4.11 に初期マップの，Fig4.12 に調整後のマップの各ユニットが表現している目標方向を示している．t_x を縦軸，t_y を横軸にプロットしている．Table4.1 マップが表現している行動の多様性を表現するために，ユニットの要素 t_x，t_y の平均，分散を示している．初期マップよりも調整後のマップのほうが平均の絶対値が小さいこと，また，分散が大きいことから，調整後のマップのほうが表現している行動に偏りが少なく，より多くの行動を表現しているといえる．

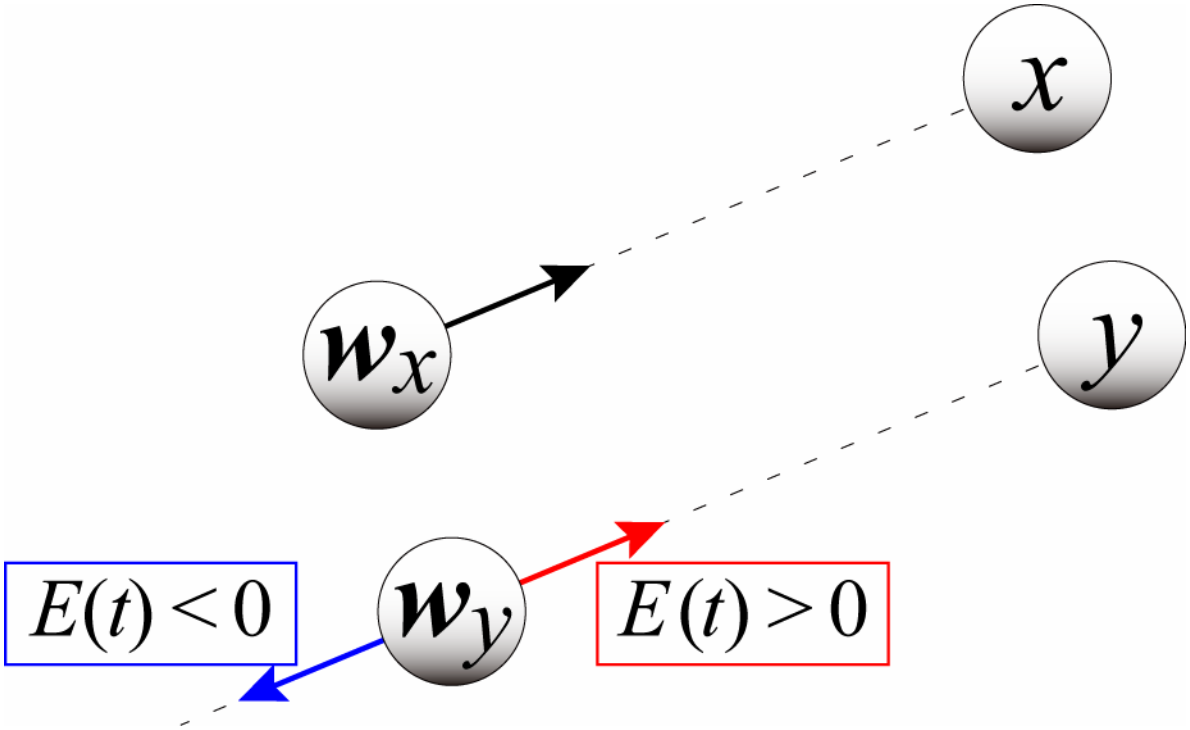
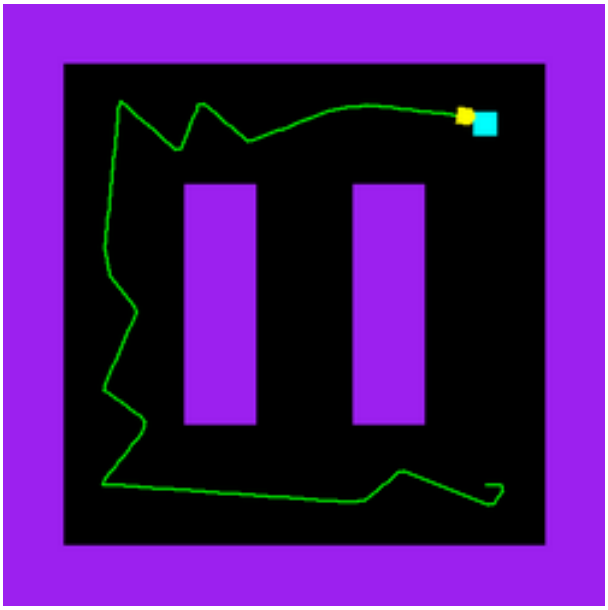
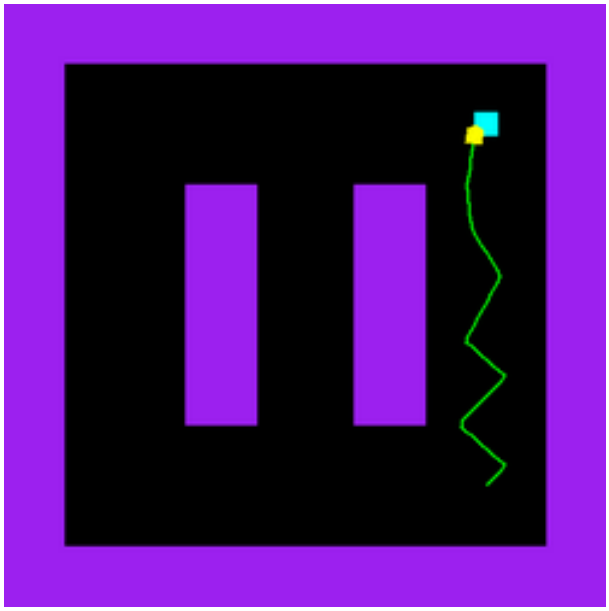


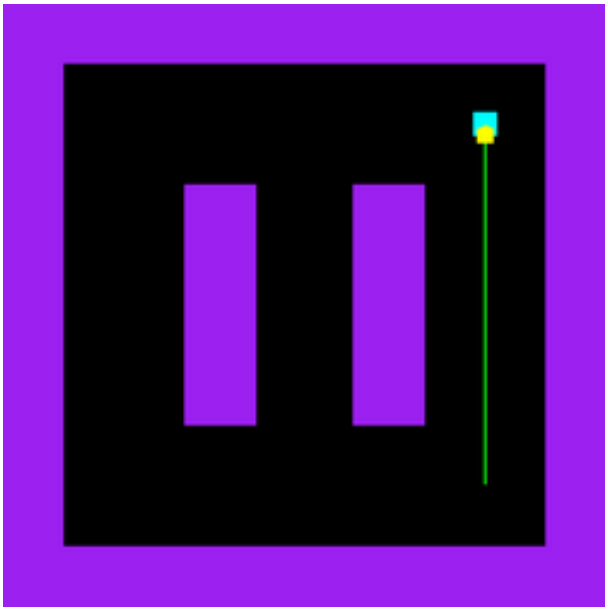
Fig 4.6 The Concept of Adjustment



(a) (150, -150, 0)

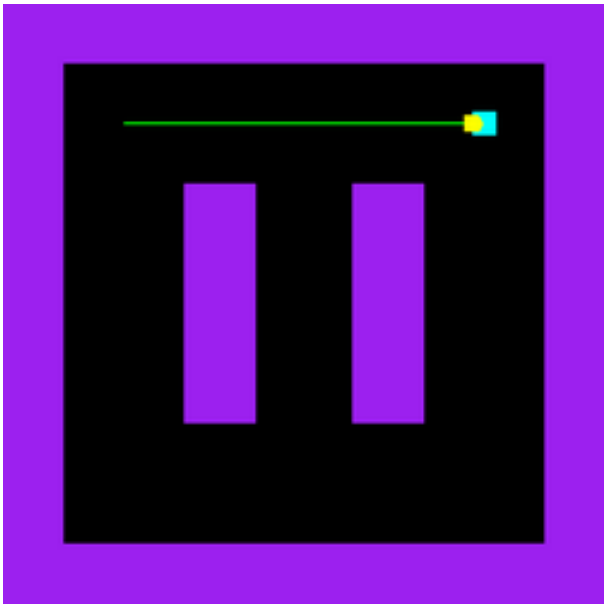


(b) (150, -150, 45)

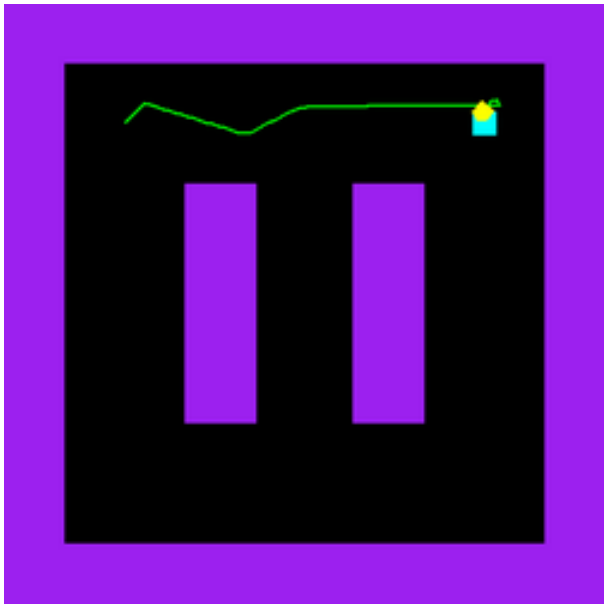


(c) (150, -150, 90)

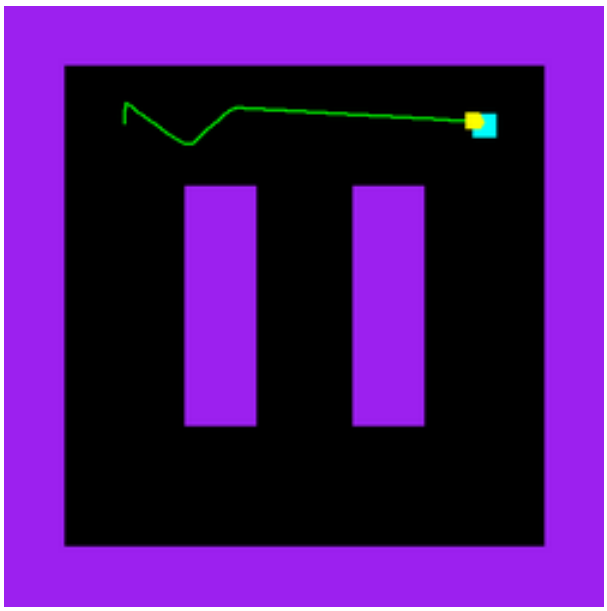
Fig 4.7 Trajectories of a Robot with Initial Map



(d) (-150, 150, 0)

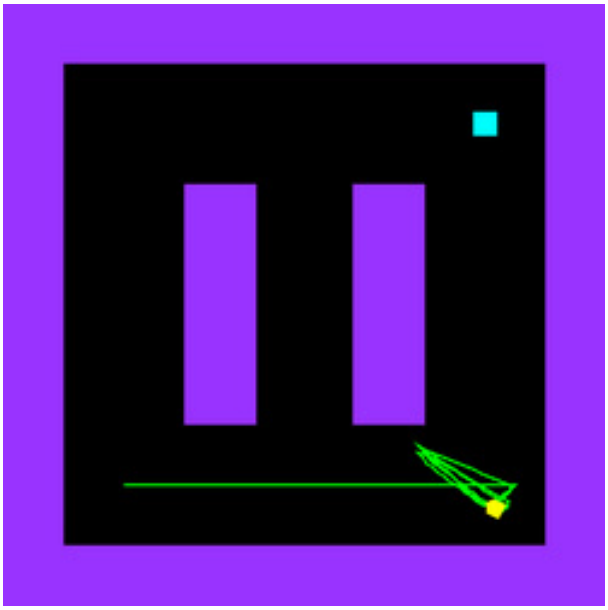


(e) (-150, 150, 45)

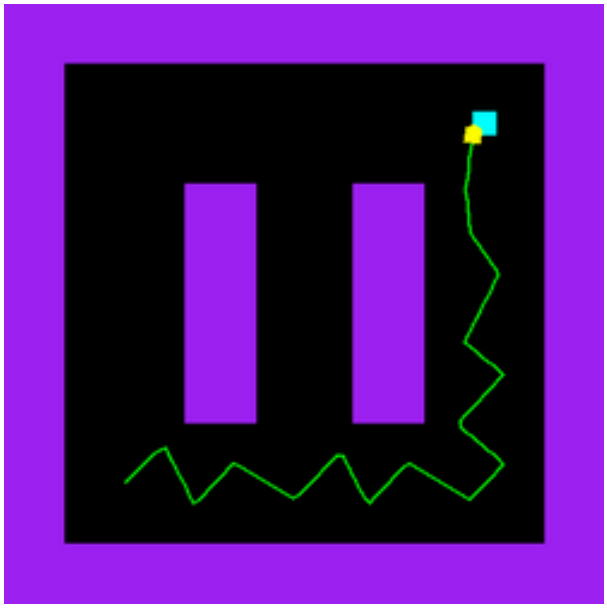


(f) (-150, 150, 90)

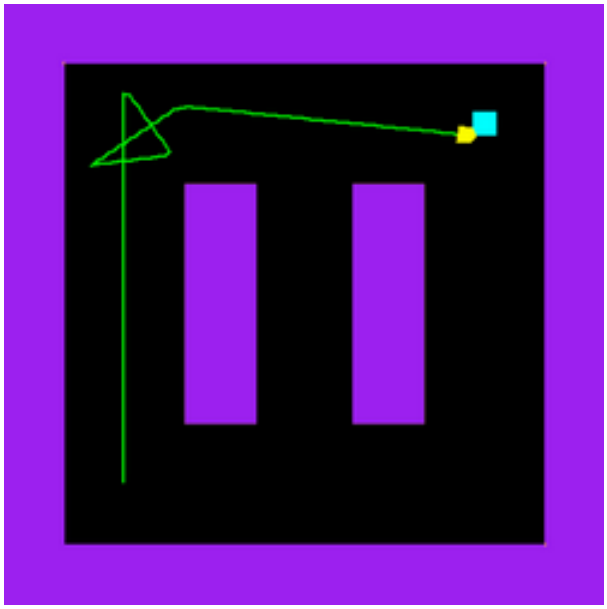
Fig 4.7 Trajectories of a Robot with Initial Map



(g) (-150, -150, 0)



(h) (-150, -150, 45)



(i) (-150, -150, 90)

Fig 4.7 Trajectories of a Robot with Initial Map

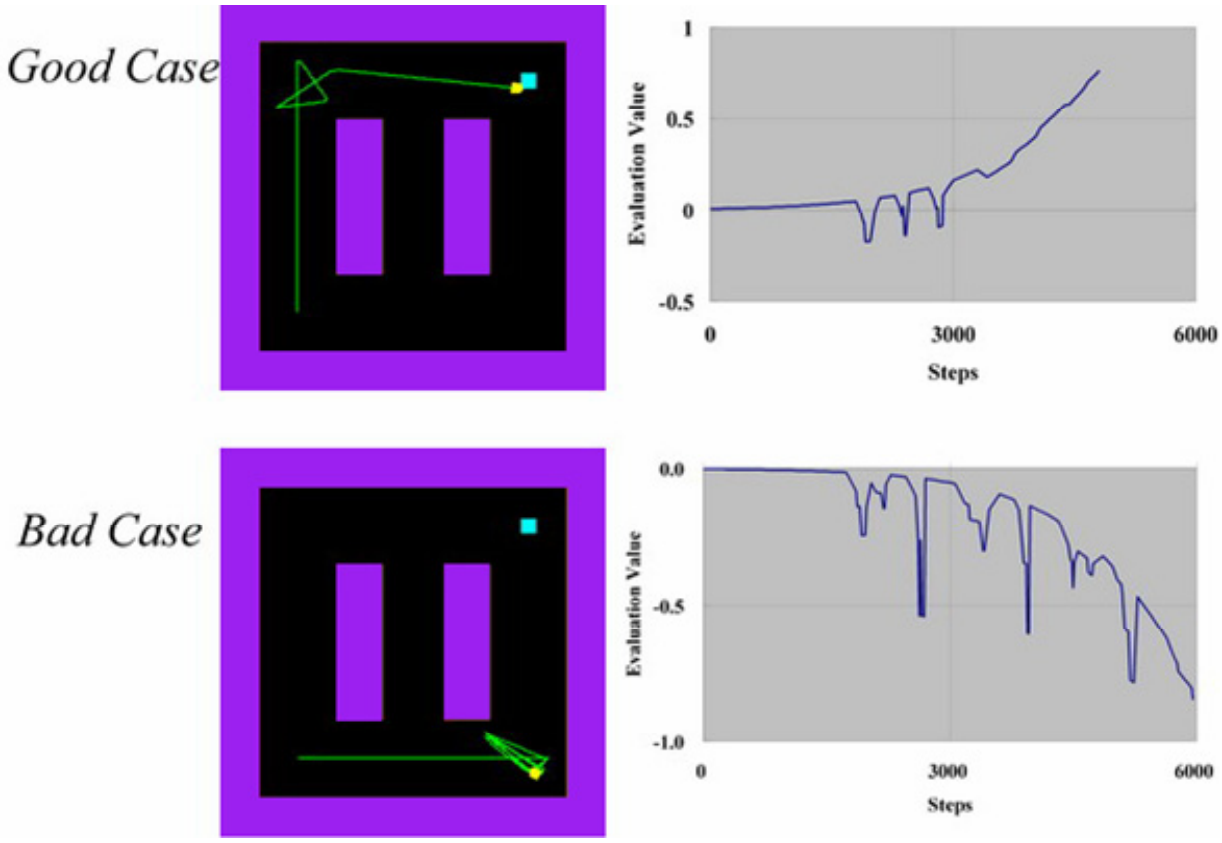
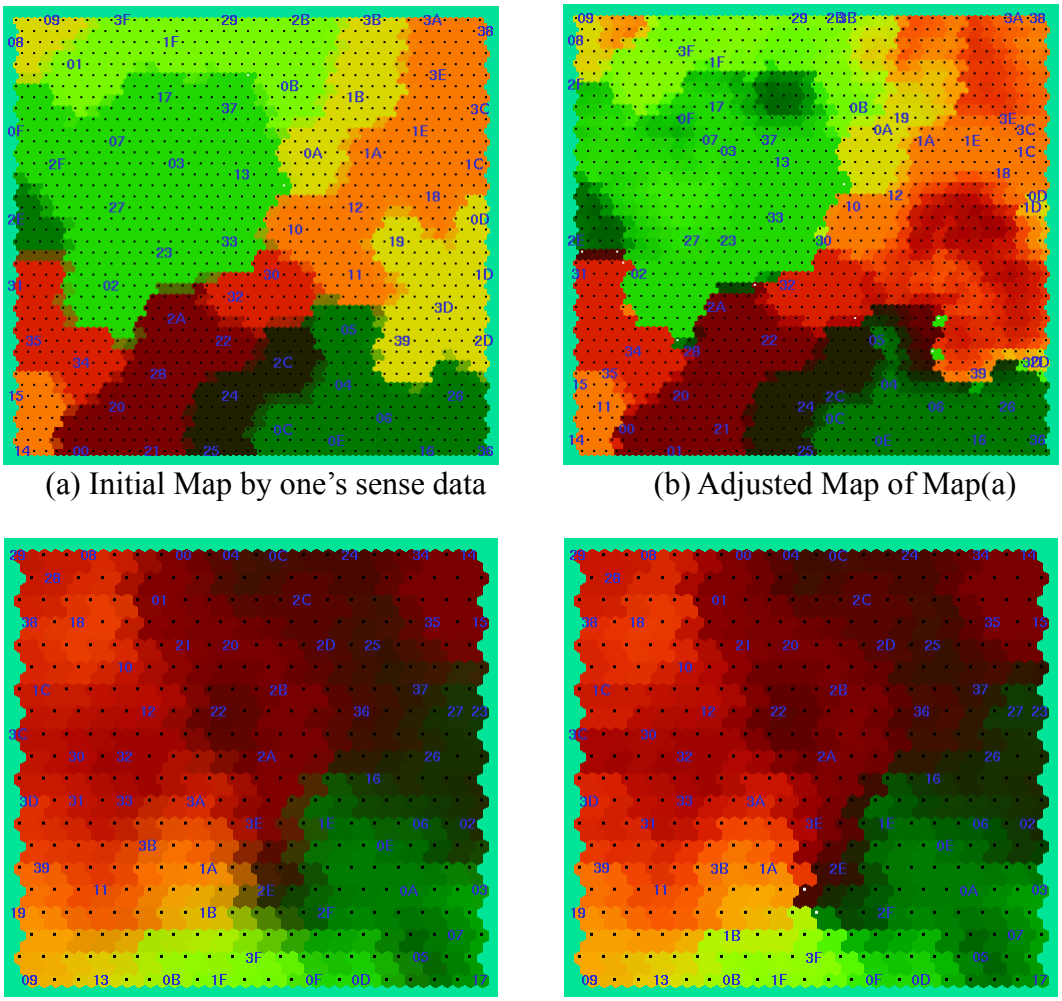


Fig 4.8 Transition of Evaluation Value on good and bad cases



(a) Initial Map by one's sense data (b) Adjusted Map of Map(a)

(c) Initial Map by equation (3.2) (d) Adjusted Map of Map (c)

Fig 4.9 An Adjusted Map using Trajectory and Evaluation Value

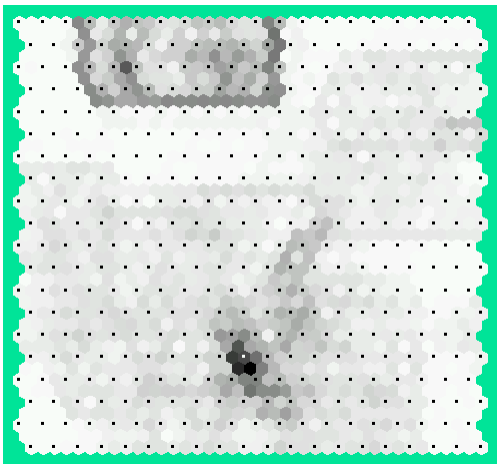
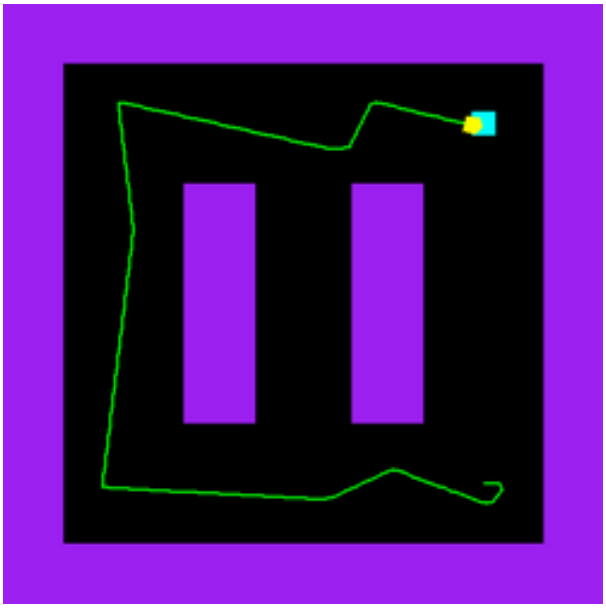
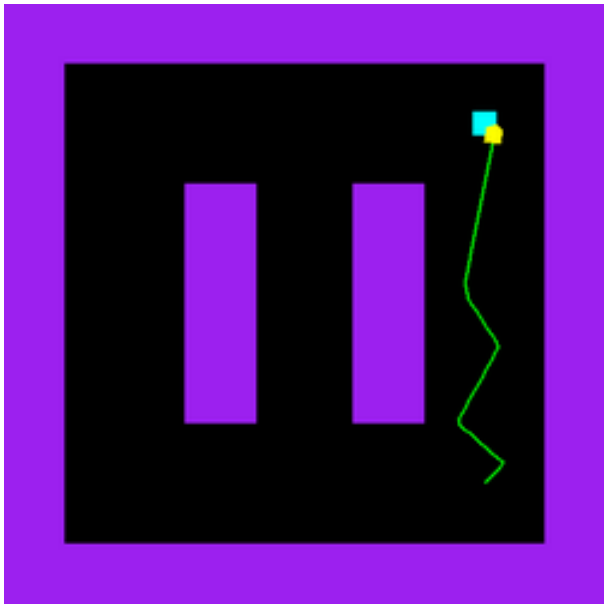


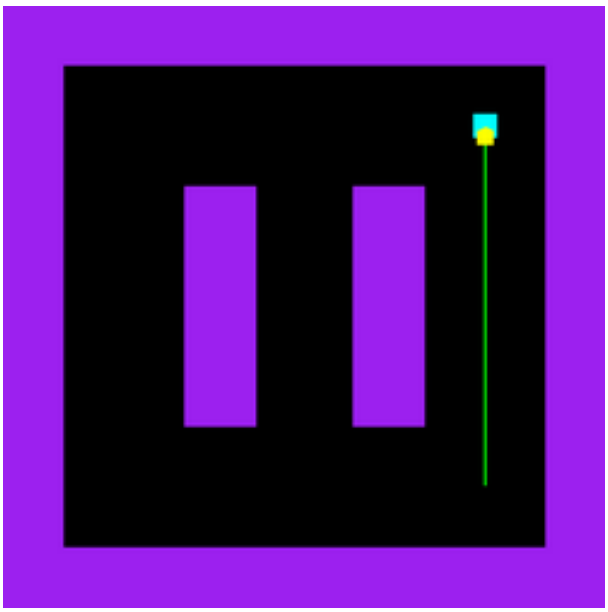
Fig 4.10 Difference Map between Initial Map (Fig3.8-(c)) and Adjusted one (Fig3.8-(d))



(a) (150, -150, 0)

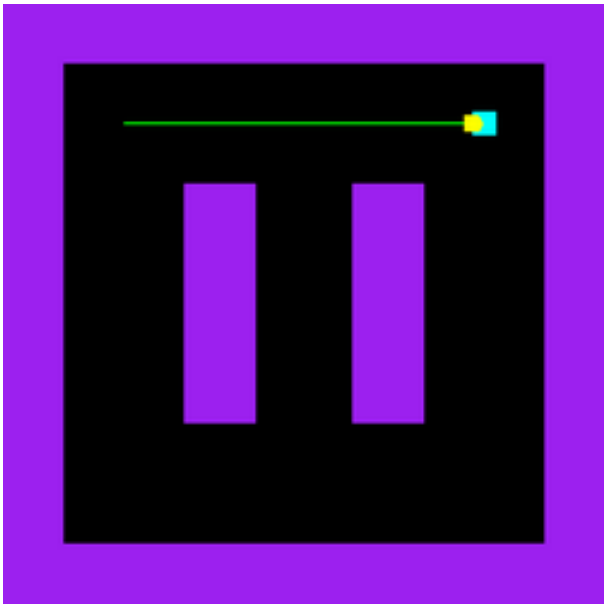


(b) (150, -150, 45)

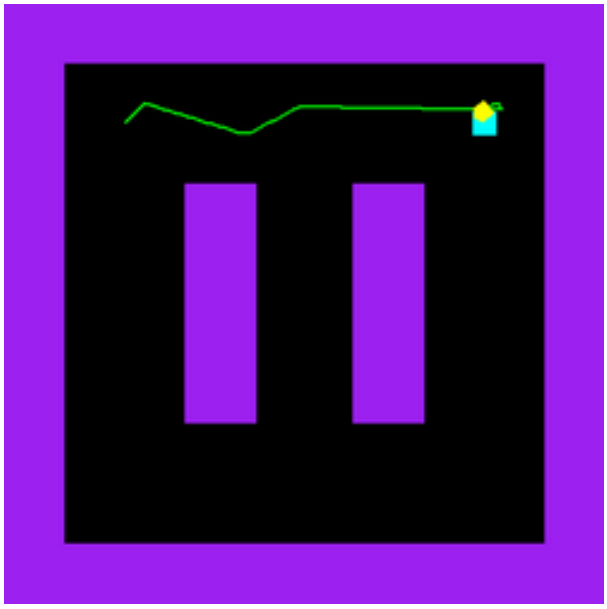


(c) (150, -150, 90)

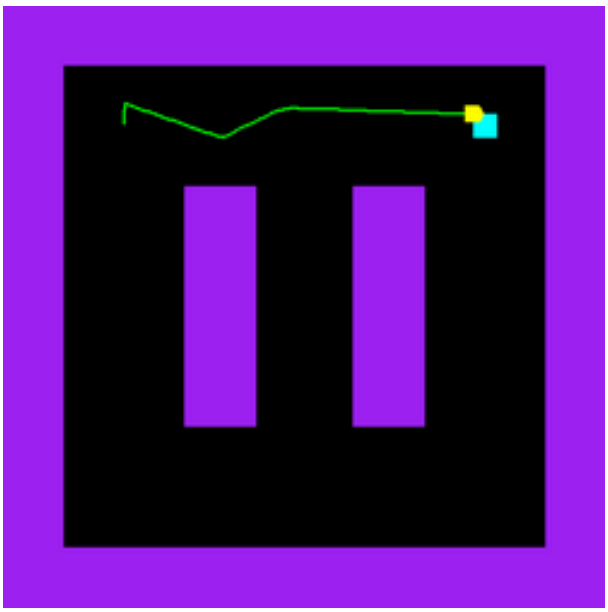
Fig 4.11 Trajectories of a Robot with Adjusted Map



(d) (-150, 150, 0)

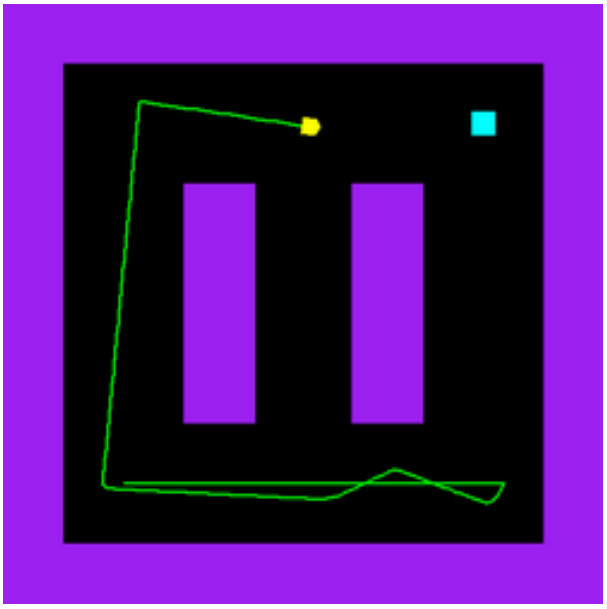


(e) (-150, 150, 45)

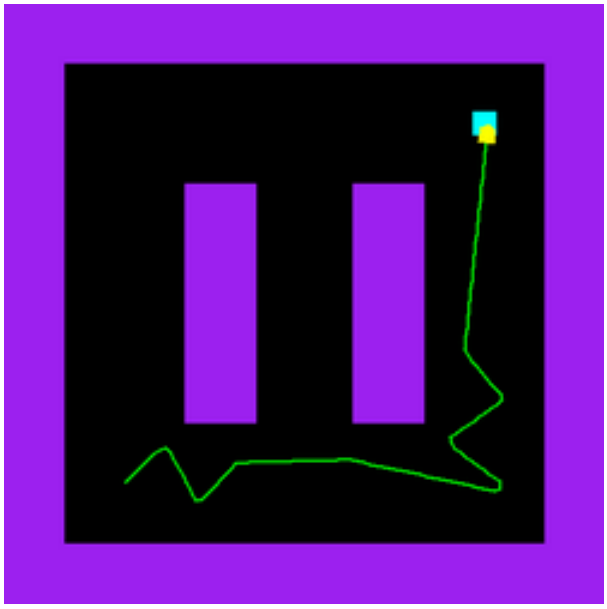


(f) (-150, 150, 90)

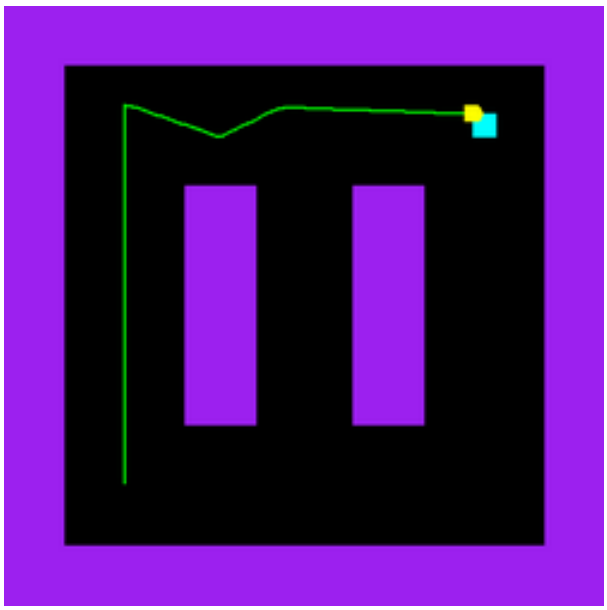
Fig3.10 Trajectories of a Robot with Initial Map



(g) (-150, 150, 0)

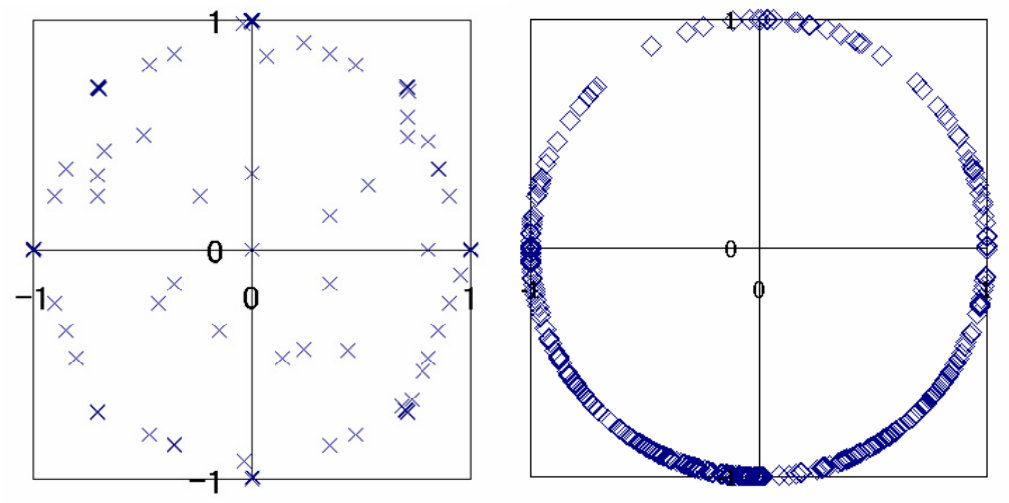


(h) (-150, 150, 0)



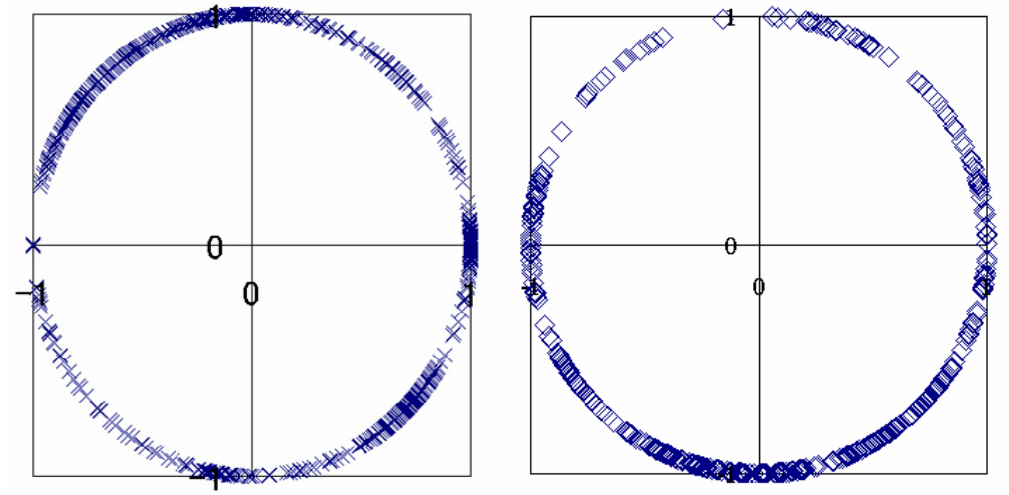
(i) (-150, 150, 0)

Fig3.10 Trajectories of a Robot with Adjusted Map



(a) Target Direction are given manually (b) Equation (4.2)

Fig 4.12 Target Direction of Initial Map



(a) Target Direction are given manually (b) Equation (4.2)

Fig 4.13 Target Direction of Adjusted Map

Table 4.1 Comparison of Distribution between Initial and Adjusted Map

	Initial Map		Adjusted Map	
	t_x	t_y	t_x	t_y
Average	-0.495	-0.044	-0.443	0.014
Variance	0.326	0.426	0.369	0.434

4.2 mnSOM を用いた運動の同定と制御

本研究では，環境やロボットの動特性の変化に応じて自己組織的にロボットの行動を獲得するシステムの開発を目標としている．本論文ではその第一段階として，動特性の変化に適応する制御システムの開発を行う[63]-[73]．良好な制御性能を得るには，動特性の変化に素早く追従し，制御則を適応的に調整していく必要がある．そこで，“教師無し”及び “教師有り”アルゴリズムの長所を取り入れたモジュラーネットワーク自己組織化マップ(mnSOM: modular network Self-Organizing Map)を用いて実現する．ロボットの制御において扱う情報は，時系列データであるのでモジュールに MLP のひとつであるリカレント型のニューラルネットワークを用い，mnSOM の基本モジュールとする．学習アルゴリズムは，先に述べた MLP 型 mnSOM のアルゴリズムと同じである．リカレントニューラルネットワーク型 mnSOM を用いて，状態量と操作量に関する様々な時系列データを学習させることにより，ロボットの動特性を取得しながら，異なった動特性を有するモジュール群を競合層に配置していく．本研究で提案する制御システムは，Fig4.11 に示すように，ロボットの動特性を表現するフォワードモデルマップ及びこれに対応したコントローラマップの 2 層のリカレントニューラルネットワークから構成されている．

リカレントニューラルネットワーク型 mnSOM を用いた適応型制御システムは，フォワードモデルマップの作成，コントローラマップの作成，及びロボット制御への適用の 3 つのフェーズ構成されている．各フェーズに関して以下に述べる．

4.2.1 適応制御システムの提案

フォワードモデルマップ

動作環境の変化や動特性の変動，アクチュエータの特性変化や故障を想定，或は，実機により計測し，状態量と操作量の組からなるデータクラスを準備し学習する．Fig4.13-(a)において，ある動特性を表現する時系列データ $S_i(\boldsymbol{x})$は矢印で示された赤色の競合層モジュールが勝者モジュールとして対応し，別の動特性を表現する時系列データ $S_j(\boldsymbol{x})$も同様に矢印で示された競合層モジュールが勝者モジュールとして対応している状況を示している．モジュールの色は学習した動特性の影響を意味しており，学習の過程で，勝者モジュールの中間には $S_i(\boldsymbol{x})$と $S_j(\boldsymbol{x})$から得られた動特性によって補完された動特性を有するモジュールが得られる．各モジュールの学習係数及び学習回数は同じ値とし，各データクラスに対する勝者モジュールが選択され，学習が収束したと判断された段階で学習を終了し，次の学習段階に移行する．得られた学習結果は，以降，フォワードモデルマップと呼ぶ.

水中ロボットの速度及び制御入力と加速度の関係を得るために，リカレントニューラルネットワーク（Fig4.14 においてフォワードモデルとして示したネットワーク）を基本モジュールとした mnSOM を用いてフォワードモデルモマップを作成する．以降，状態変数 S は位置，ΔS は速度及び$\Delta 2S$ は加速度，u は制御入力を示すものとする．フォワードモデルモマップ作成のアルゴリズムを以下に述べる.

M 個の時系列データクラス $\boldsymbol{D}_i = (\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots, \boldsymbol{D}_M)$があり，それぞれ N 単位時間の入出力対$(\boldsymbol{x}_{ij}, \boldsymbol{y}_{ij}) = \{(\boldsymbol{x}_{i1}, \boldsymbol{y}_{i1}), (\boldsymbol{x}_{i2}, \boldsymbol{y}_{i2}),\ldots, (\boldsymbol{x}_{iN}, \boldsymbol{y}_{iN})\}$を持っており，関数$f_i(.)$を構成しているとすると，式(4.6)の関係が成り立つ.

$$\begin{aligned} &\boldsymbol{D}_i = \left(\boldsymbol{x}_{ij}, \boldsymbol{y}_{ij}\right) = \left\{\left(\Delta S_{ij}, u_{ij}\right), \quad \Delta^2 S_{ij}\right\} \\ &\Delta^2 S_{ij} = f_i(\Delta S_{ij}, u_{ij}) \end{aligned} \quad \left(i = 1 \sim M, j = 1 \sim N\right) \tag{4.6}$$

一方，mnSOM のモジュールが K 個あり，k 番目のフォワードモデルの持つニューラルネットワークノ結合加重は $\boldsymbol{w}^k$ であるとすると，入力データ$(\Delta S_{ij}, u_{ij})$と出力データ $\Delta^2 \hat{S}_{ij}^k$ は，式(4.7)に示す関数 $F^k(.)$を構成している.

$$\Delta^2 \hat{S}_{ij}^k = F(\Delta S_{ij}, u_{ij}; \boldsymbol{w}^k) = F^k(\Delta S_{ij}, u_{ij}) \tag{4.7}$$

このとき，学習は以下のように行われる．

i) *初期化*: mnSOM の結合加重 $\boldsymbol{w}^k$ を乱数で初期化する

ii) *評価過程*: 式(4.8)を用いて，$\Delta^2 \hat{S}_{ij}(k)$ と$\Delta 2S_{ij}$ の平均二乗誤差 $E_{Fwd_j}{}^k$ のクラス $\boldsymbol{D}_i$ に対して計算する．式(4.8)は$f_i(.)$と $F_k(.)$の距離を求めていることになる．

$$E_{Fwd_i}(k) = \frac{1}{2N} \sum_{j=1}^{N} \left\{ \Delta \hat{S}_{ij}(k) - \Delta S_{ij} \right\}^2 \tag{4.8}$$

iii) *競合過程*: 式(4.9)に従って，誤差が最小となる勝者モジュールのインデックスを k_i^*を決定する．

$$k_i^* = \arg\min_k \left(E_{Fwd_i}{}^k \right) \tag{4.9}$$

iv) *協調過程*: 式(4.10)に従って勝者モジュールからの距離および学習回数によって，学習分配率Ψ_i^k を決定する．Ψ_i^k は k 番目のモジュールが i 番目のデータクラスの入出力関係を学習する量を示す．$h(.)$は近傍関数で，距離 L 及び学習回数 T が増加するにつれて単調減少する関数を選択する．

$$\Psi_i^k = h\left\{ L\left(k, k_i^*\right), T \right\} \Big/ \sum_{i'=1}^{M} h\left\{ L\left(k, k_{i'}^*\right), T \right\} \tag{4.10}$$

v) *適応過程*: 誤差逆伝播法によって学習する．式(4.11)に示すように学習率ηと学習分配率Ψ_i^k を用いて $\boldsymbol{w}^k$ を更新する．

$$\Delta \boldsymbol{w}^k = -\eta \sum_{i=1}^{M} \left\{ \Psi_i^k \frac{\partial E_i^k}{\partial \boldsymbol{w}^k} \right\} \tag{4.11}$$

ii)～*v)*の過程を学習が収束するまで繰り返す．

リカレントニューラルネットワーク型 mnSOM を用いた場合でもの基本的に mnSOM アルゴリズムには変更はないが，入力データとして与える水中ロボットの時系列データは，順序に意味があるのでバッチ型の学習法を用いる．

コントローラマップ

第 2 段階として，作成したフォワードモデルマップを用い，各モデルに対応したコントローラを作成する．Fig4.13-(b)の上部に示されたコントローラマップの作成においては，競合層における幾何学的な配置は考慮せず，対応するフォワードモデルに対してコントローラを適応させるものとする．コントローラマップにおける各コントローラに対して制御目標値が与えられる．対応するフォワードモデルから得られるロボットの状態量と目標値との偏差量をもとに操作量が算出され，フォワードモデルマップに送信される．フォワードモデルマップでは，操作量から計算される次の時間ステップの状態量と目標値との誤差をコントローラマップに逆伝播することにより，誤差逆伝播法を用いて制御則の調整を行う．制御目標値，学習係数及び学習回数はここではすべてのコントローラにおいて同じ値とし，学習が収束するまで行う．

フォワードモデルマップのときと同様に，リカレントニューラルネットワーク型 mnSOM を用いて，目標値と現在の状態の誤差から制御入力を決定するコントローラマップを作成する．Fig4.13-(b)に示すように，初期コントローラマップと学習済みのフォワードモデルマップを結合し，Fig4.14 に示したコントローラとフォワードモデルのネットワークを一つのネットワークとした基本モジュールを持つ mnSOM として構成し，フォワードモデルの各ウェイトを固定して，コントローラのウェイトを調整する．以下にアルゴリズムを述べる．

コントローラマップ作成における入力データクラス $\boldsymbol{D}_i$ は目標状態 $\boldsymbol{r} = (r, \Delta r)$を用いて以下の式(4.12)のように表現する．

$$\begin{aligned} \boldsymbol{D}_i &= \left(\boldsymbol{x}_{ij}, \boldsymbol{y}_{ij}\right) = \left\{\left(r_{ij} - S_{ij}, \Delta r_{ij} - \Delta S_{ij}, u_{ij-1}\right), u_{ij}\right\} \\ u_{ij} &= g_i\left(r_{ij} - S_{ij}, \Delta r_{ij} - \Delta S_{ij}, u_{ij-1}\right) \end{aligned} \tag{4.12}$$

ここで，r は目標位置，Δr は目標速度である．

一方， k 番目のコントローラの持つネットワークの結合加重は $\boldsymbol{v}^k$ と表すとすると，入力$(r_{ij} - S_{ij}, \Delta r_{ij} - \Delta S_{ij}, u_{ij-1})$と出力 u_{ij} は，式(4.13)に示す関数 $G^k(.)$を構成している．

$$\begin{aligned} \hat{u}_{ij}^k &= G(r_{ij} - S_{ij}, \Delta r_{ij} - \Delta S_{ij}, u_{ij-1}; \boldsymbol{v}^k) \\ &= G^k(r_{ij} - S_{ij}, \Delta r_{ij} - \Delta S_{ij}, u_{ij-1}) \end{aligned} \tag{4.13}$$

先に述べたように，mnSOM の学習のプロセスに変更はないが， コントローラマップ作成において異なるのは，評価過程における評価関数と，それに伴って変化する適応過程における結合加重の更新式である．更新式そのものは式(4.11)と同じである．式(4.14)にコントローラマップに対する評価関数を示す．式(4.14)は，目標状態に現在の状態が近ければ，また，制御入力の絶対値が少ないほど，値が小さくなり，良い評価であることを示している．

$$E_{Ctl_i}^{k} = \frac{1}{2N}\sum_{j=1}^{N}\left\{p_1\left(r_{ij}-S_{ij}\right)^2 + p_2\left(\Delta r_{ij}-\Delta S_{ij}\right)^2 + p_3 {u_{ij}}^2\right\} \tag{4.14}$$

ここで，p_1，p_2，p_3 は，それぞれ位置，速度及び制御入力に対する評価値への重み係数である．

ロボットの制御及びオンライン調整

第 1 段階及び第 2 段階の学習は，準備されたデータのオフライン学習及び調整であった．第 3 段階では制御対象であるロボットの制御と並行して，ロボットの動作環境や制御特性の変化に対してオンライン学習による適応を行う．ロボットのミッション遂行中において，ある一定期間サンプリングされた時系列データ（Fig4.13-(c)における $S_x(\boldsymbol{x})$）をフォワードモデルマップへ入力する．フォワードモデルマップは動特性推定器として用いられ，最も動特性を表現する勝者モジュールが選択される．ロボットの制御には，勝者となったフォワードモデルに対応するコントローラから得られる操作量がロボットへの制御入力として採用される．また，これと並行して，フォワードモデルマップの追加学習による動特性変化への適応，及びフォワードモデルマップの更新に伴うコントローラマップの調整が行われる．フォワードモデルマップを用いた動特性の推定と制御入力の決定は以下のように行う．

N'時間観測したデータを用いて，式(4.15)を用いて FMMs との誤差を計算する．

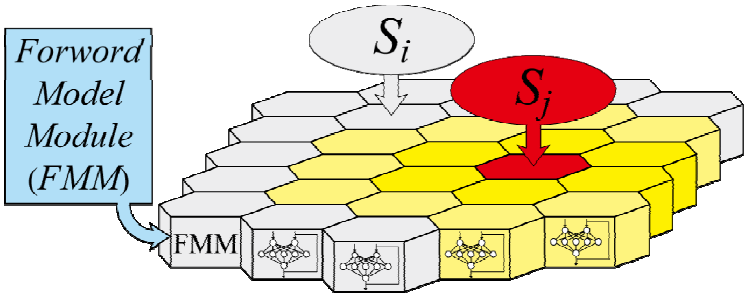
$$E_{Exp}{}^{k} = \frac{1}{2N'}\sum_{j=1}^{N'}\left\{\Delta\hat{S}_{Exp_j} - \Delta S_{Exp_j}\right\}^2 \tag{4.15}$$

次に，式(11)に従って，観測された状態を最も良く表現しているフォワードモデルのインデックス k^*_{Exp} を求める．

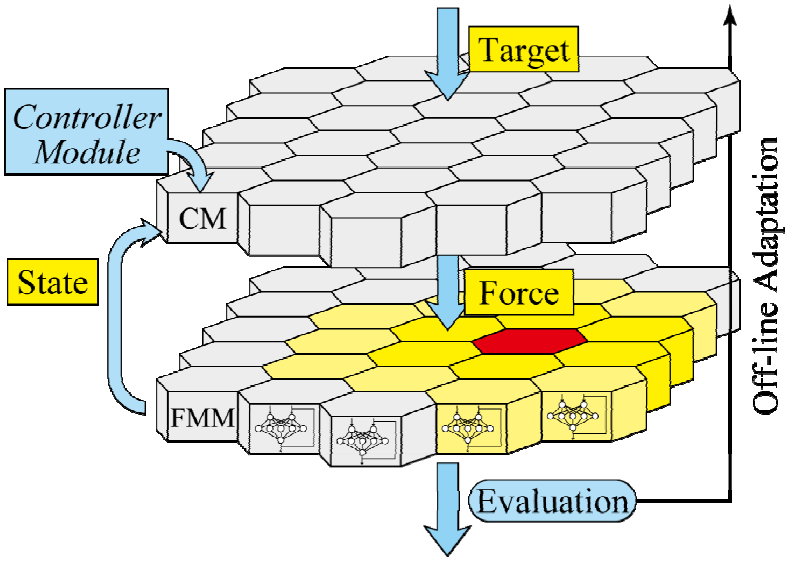
$$k_{Exp}{}^*_j = \underset{k}{\operatorname{argmin}}\left(E_{Exp}{}^k\right) \tag{4.16}$$

インデックス $k_{Exp}{}^*$ に対応するコントローラの出力をロボットへの制御入力として採用する．

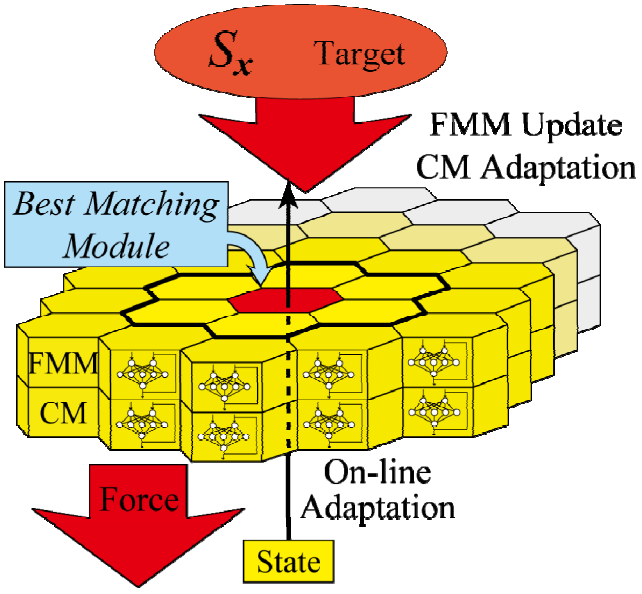
$$u_{Exp} = G^{kExp^*_j}(r_j - S_{Exp\,j}, \Delta r_j - \Delta S_{Exp\,j}, u_{Exp\,j-1}) \tag{4.17}$$



(a) Building a Forward Model Map



(b) Adaptation of a Controller Map using the Forward Model Map



(c) Implementation of the Control Map to Robot Control and On-line Adaptation using the Forward Model Map and the Controller Map

Fig 4.14 Learning Processes of an Adaptive Controller System using RNN-mnSOM

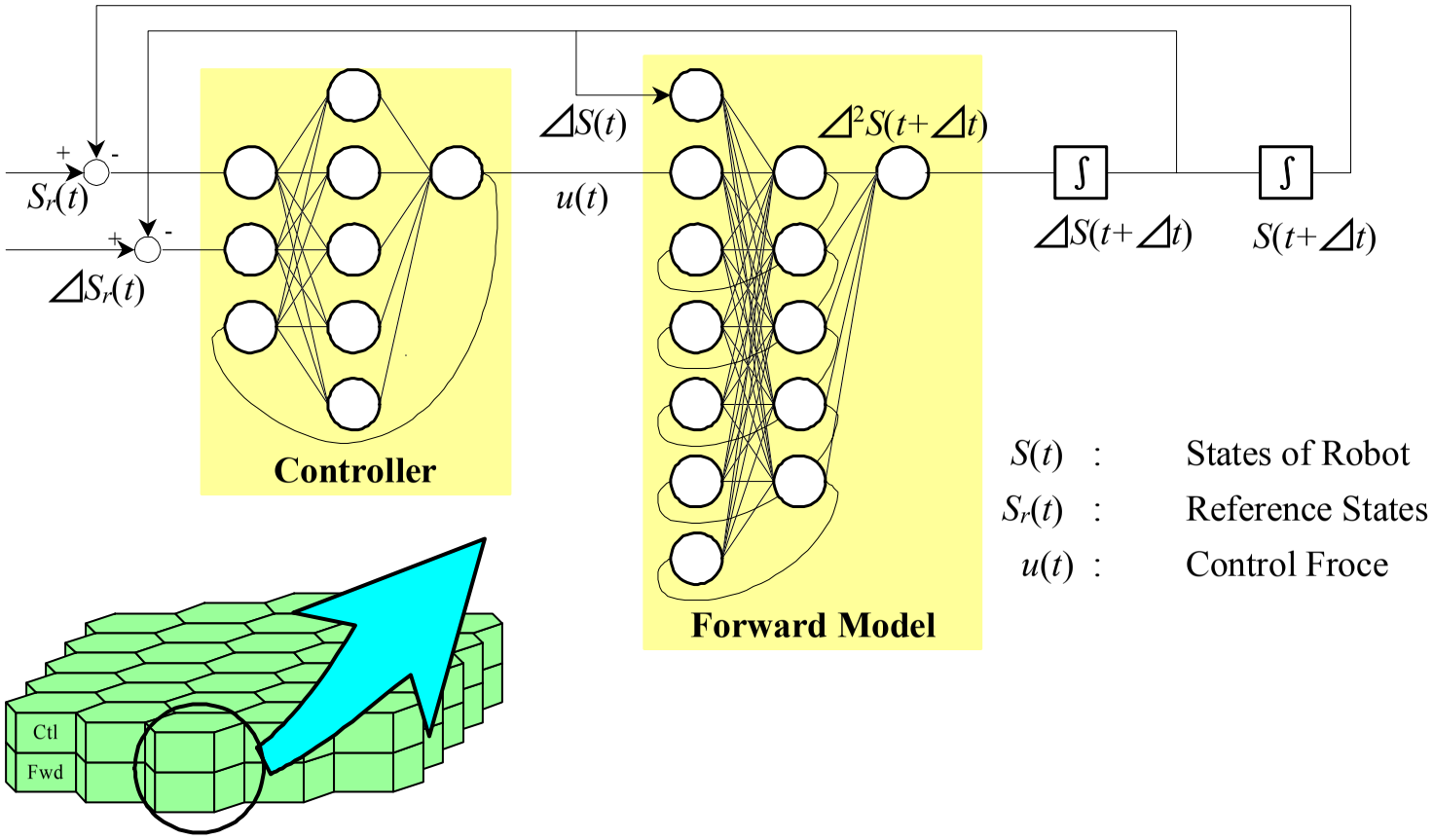


Fig 4.15 The Network Structure of a Forward Model Module and a Controller Module

4.2.2 シミュレーション

フォワードモデルマップの作成

提案手法の第 1 段階として，フォワードモデルマップ作成シミュレーションを行った．入力時系列群の関係とそれらを補間した動特性が反映されたマップが得られるかを検証するために，まずは，ひとつの運動モードに関してパラメータを変えてシミュレーションを行った．

式(4.18)に単純化した水中ロボットの運動方程式を示す．

$$F = M\ddot{x} + C\dot{x}|\dot{x}| \tag{4.18}$$

F は力，$\dot{x}$ は速度，$\ddot{x}$ 加速度を示しており，M，C はそれぞれ質量及び付加質量，非線形流体力に関するパラメータである．式(4.18)を用いてパラメータ M，C を Table4.2 示すように変化させ，速度の絶対値が 0.2[m/s]を超えたら制御入力 5[N]の方向を反転させるリミットサイクル運動の時系列データを 50 秒間，10[Hz]でサンプリングしたものを用意し，6x6 格子状モジュール構造の mnSOM に入力した．

Fig4.13 は，Table4.2 示している 9 個のパラメータから生成した時系列を mnSOM へ入力し，100,000 回学習を行った結果の得られたマップである．マップの各正方形はフォワードモデルを表しており，それぞれある動特性を表現したリカレントニューラルネットワークを持っている．入力時系列に対する勝者モジュールにはその時系列の速度(青細線)，加速度(赤太線)を，時間を横軸にとってプロットしている．入力したデータクラス D_i (i = 0～8)は，それぞれマップ上の(1, 1)，(1, 1)，(2, 6)，(4, 1)，(4, 2)，(4, 6)，(6, 1)，(6, 3)，(6, 6)に配置されている．

得られたフォワードモデルマップを検証するため，各モジュールに対してリミットサイクル試験を行った．各モジュールから得られた時系列データを用いて，最小二乗法によって運動方程式のパラメータ，を算出して，各モジュールが表現している動特性を数値的に評価する．運動方程式の係数 M，C の推定を行った結果を Fig3.14，時系列の速度を横軸，加速度を縦軸にとった状態推移の様子を Fig3.15 に示す．

Fig4.14 における赤い四角は入力データ作成時に用いた係数，青い四角はフォワードモデルマップから推定された各フォワードモデルモジュールのパラメータを表している．パラメータ *M-C* 空間において，フォワードモデルマップは格子状に広がっており，入力データのパラメータに完全には一致していないものの，中間に存在するモジュールは入力データによって補完された中間的な振る舞いを表現していることがわかる．

Fig4.15 のグラフの位置は，Fig4.13 の各モジュールの位置に対応している．リミットサイクルの平行四辺形は左側から右側に向かって徐々に面積が狭くなっている．制御入力が一定であることから，*M*（質量及び付加質量）が増加すると加速度の最大値が減少することから，左側から右側に向かって *M* が増加していることが分かる．また，上段から下段に向かうにつれて傾きが急峻になっている．*C* が上昇すると加速度の減衰が大きくなることから，右から左に向かって *C* の値が上昇していることが分かる．

コントローラマップの作成

得られたフォワードモデルマップを用いてコントローラマップを作成する．コントローラモジュールの結合加重は，乱数を用いて初期化し，設定した目標軌道にフォワードモジュールの出力が追従するようにコントローラの結合加重を調整する．両者を一つの MLP としてみなし，フォワードモデルモジュールから出力される状態量をコントローラモジュールの評価値として用いる．評価値を誤差逆伝播法によってコントローラネットワークへ逆伝播することにより，教師無し学習による結合加重の更新が可能となる．

コントローラマップ生成のため，0～25[sec]において 0.5[m]，25～50[sec]で-0.5[m]，目標速度を 0.0[m/s]とし，10[Hz]でサンプリングした時系列をすべてのコントローラに対しての目標軌道として入力した．コントローラマップの調整を 10,000 回行った結果を Fig4.18 に示す．コントローラマップの配置は，Fig3.15 のフォワードモデルマップの配置に幾何学的に対応している．Fig4.18 の各時系列データは，目標軌道に対する制御結果を示しており，横軸は時間[sec]，水色線は目標値[m]，青点線はロボットの位置[m]，赤細線は操作量[10N]を表している．各コントローラモジュールは全て目標値

を追従しており，フォワードモデルモジュールが有する動特性に対応した制御器を生成できている．

4.2.3 適応性能試験

得られたフォワードモデルマップとコントローラマップを用いた制御システムの適応性能を検証するためのシミュレーションを行った．比較対象システムとして，参考文献[14]-[17]において提案されたニューラルネットワークを用いたオンラインコントローラ適応システムを用い，学習していない時系列データに対する適応能力の評価を行った．初期状態を(M, C) = (80, 25）のときとし，ネットワークを調整してフォワードモデル及びコントローラのネットワークを構築した．初期状態は，Fig4.15 のフォワードモデルマップ上では最左上のモジュールが最もよく表現している．

まず，提案システムは入力された時系列の動特性を推定し適切なコントローラのインデックスを得る．ここで入力した時系列は，フォワードモデルマップを作成する際に教示データとして与えていないパラメータ(M, C) = (95, 40)及び(120, 120)とした．(M, C) = (95, 40)はフォワードモデルマップに対する入力時系列のパラメータの範囲内のものであり，学習によって内挿されていることが期待される．一方，(M, C) = (120, 120)は学習領域外のパラメータである．それぞれのパラメータで作成したリミットサイクル時系列データに対する評価関数の推移を Fig4.19 に示す．図中の実線が提案する適応制御システム，点線が比較対象システムを表している．上図がフォワードモデルの推定誤差を，下図がコントローラの評価値の推移を表している．提案システムは比較対象システムと比較するとの誤差が小さいことが分かる．提案システムにおいては，与えられた時系列を最もよく表現しているフォワードモデルモジュール及びそれに対応するコントローラを調整するため，モデル誤差及び制御誤差とも比較対象システムに比べて良好な適応性能を示している．

また，このとき適応した結果作成されたフォワードモデルネットワークが表現している動特性をリミットサイクル試験によって推定した結果を Table4.3 に示す．(M, C) = (95, 40)の場合は，比較対象システムと提案システムにおいて，M の推定結果はどちらも良好であるが，C の推定結果は比較システムのほうが良好な結果であった．ま

た，$(M, C) = (120, 120)$の場合は，提案システムの方が良好な推定結果であった．

Table 4.2 Coefficient *M* and *C* for Limit Cycle Motion

	(*M*, *C*)	
*D*0 : (80, 25)	*D*3 : (90, 25)	*D*6 : (100, 25)
*D*1 : (80, 50)	*D*4 : (90, 50)	*D*7 : (100, 50)
*D*2 : (80, 100)	*D*5 : (90, 100)	*D*8 : (100, 100)

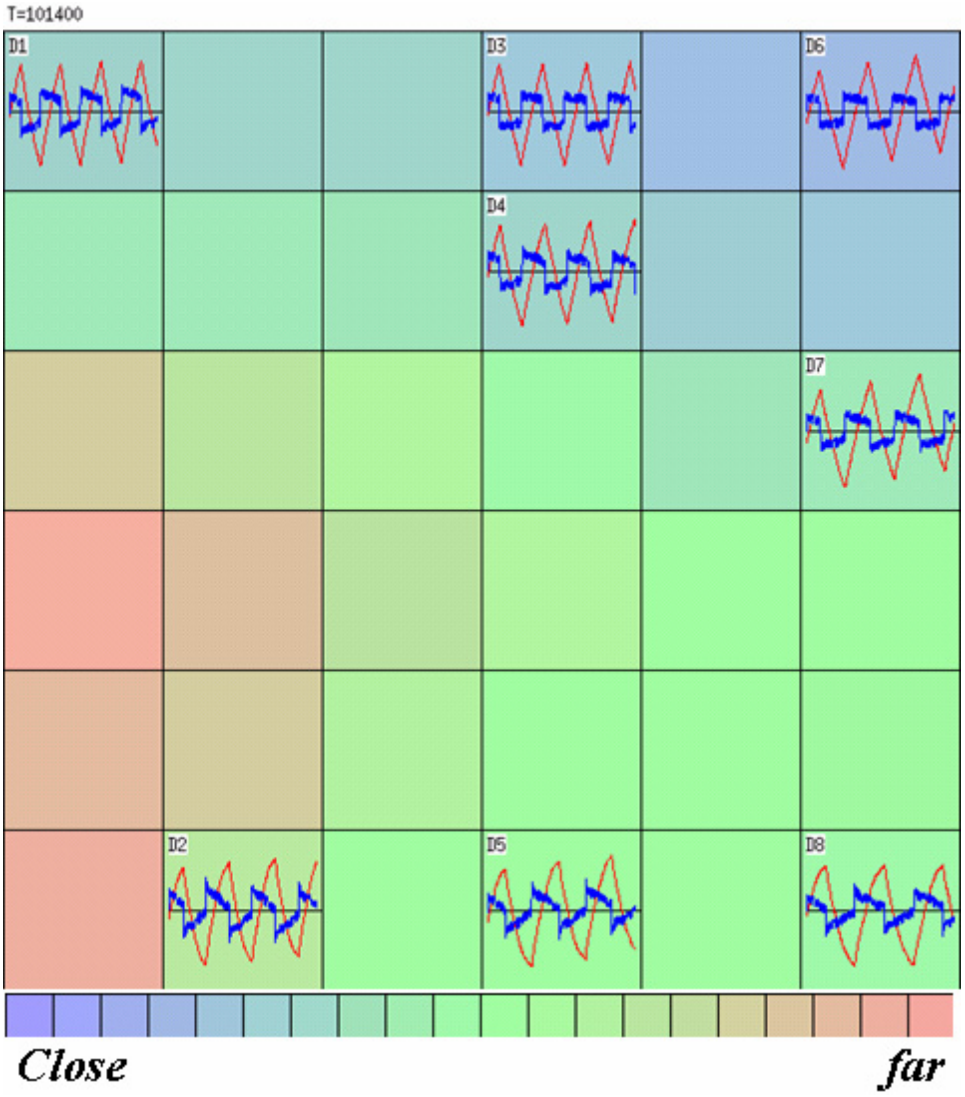


Fig 4.16 A Forward Model Map Obtained from the Time Series of Limit Cycle Simulation Data

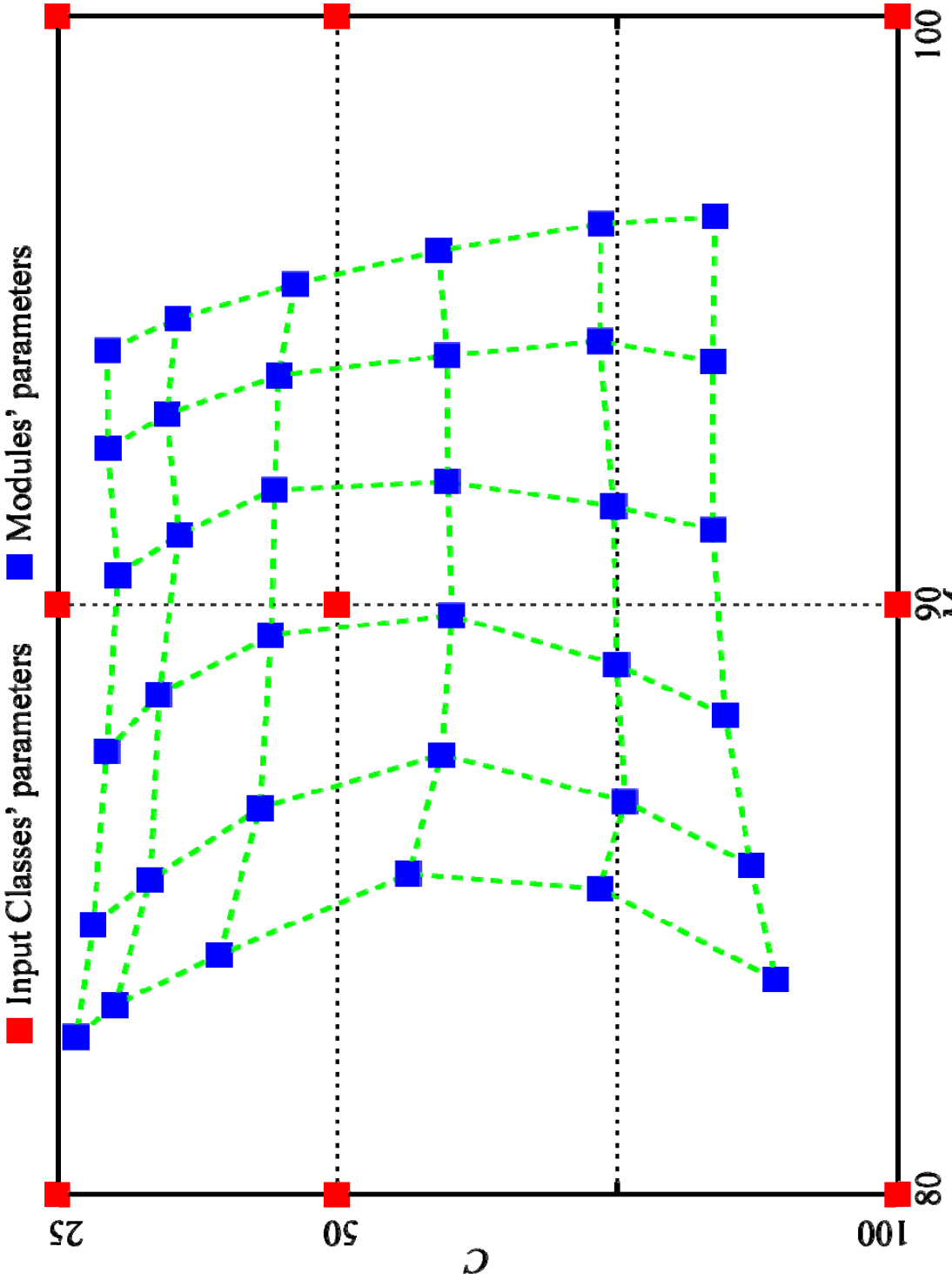


Fig 4.17 Forward Model Map Evaluation in *M-C* Space by the Least Square Method

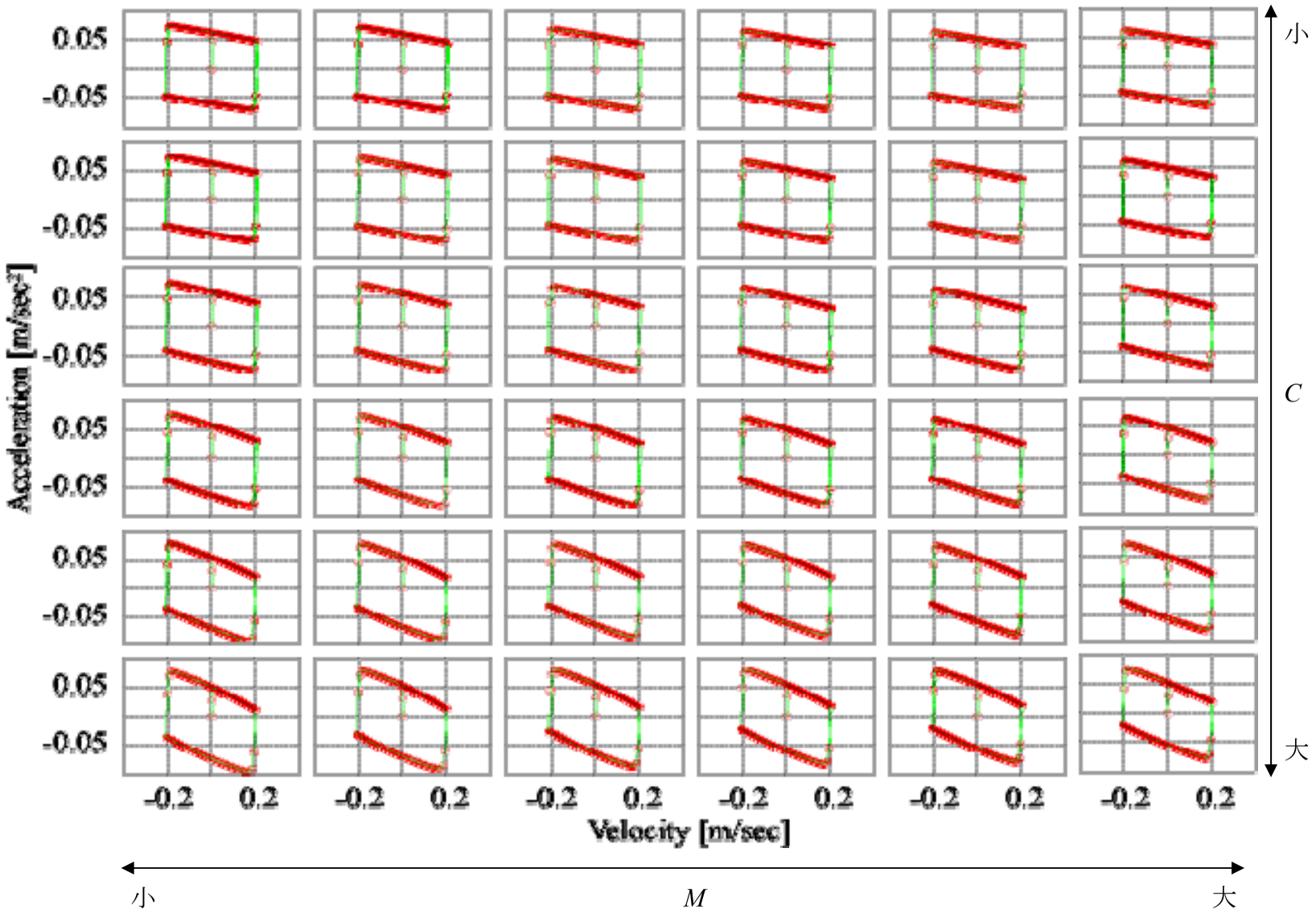


Fig 4.18 Acceleration-Velocity Relationship Obtained from Limit Cycle Simulation with FMM

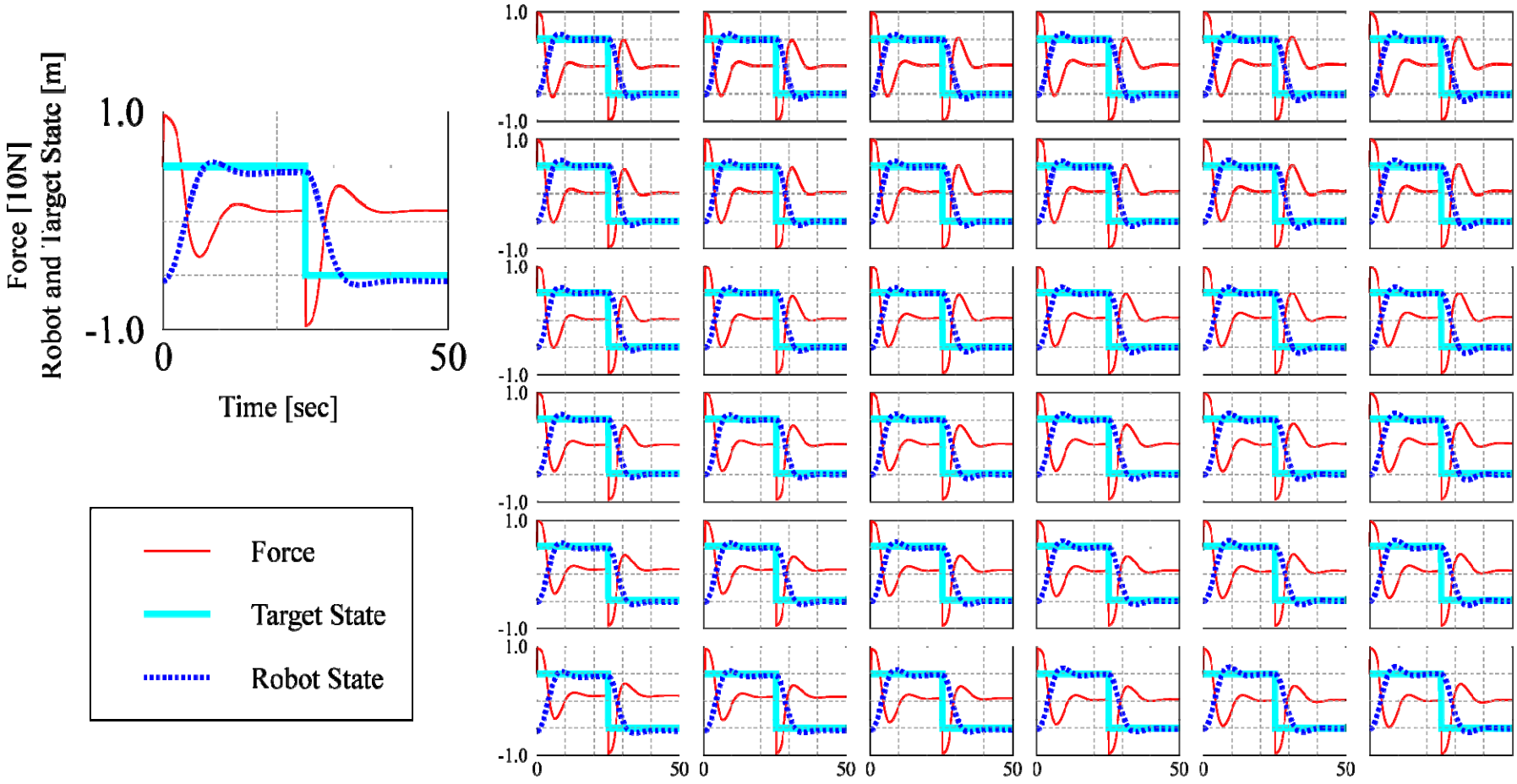


Fig 4.19 Controller Map are adjusted using corresponding Forward Model Modules

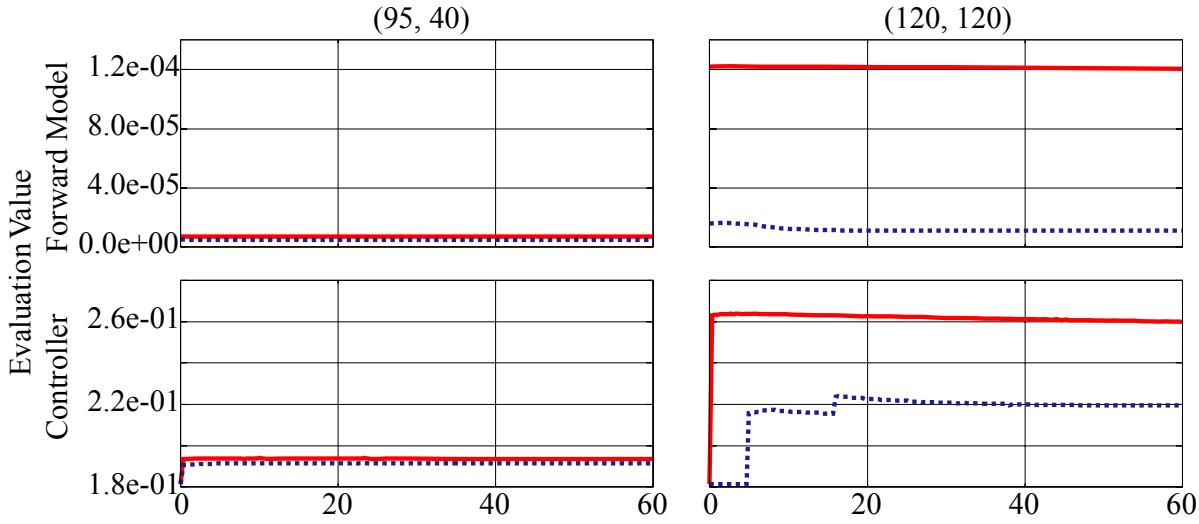


Fig 4.20 Transition of Evaluation Values

Table 4.3 Comparison of Estimated Parameters between Reference and Proposed System

(*M*, *C*)	Reference	Proposed
(95, 40)	(95.7, 29.3)	(94.5, 38.1)
(120, 120)	(149.8, 93.9)	(119.6, 128.3)

第5章

実験

障害物回避実験

自己組織化マップを用いた障害回避手法を自律型水中ロボット Twin-Burger へ適用した．実験は，直径 6[m]，深さ 1.1[m]の円形の水槽で外壁に衝突せずに回避をしながら航行を続ける実験を行った．推力 10[N]で前進を行い，一秒に一回 SOM の学習により得た環境認識マップから，現在の超音波センサの距離データのマッチングをとって，前進する方角を決める．式(5.1)に示したように，出力された目標角度が 0.5[rad]よりも小さいときはそのまま前進し，目標角度が 0.5[rad]以上のときは，前進を停止し回頭を行う．

$$\begin{cases} \text{forward} \quad for \quad \left|\hat{\theta}\right| \le 0.5[rad] \\ \text{turn} \quad for \quad \left|\hat{\theta}\right| > 0.5[rad] \end{cases} \tag{5.1}$$

ロボットが物体を障害物までの距離データを Fig4.1 示したように，1.0 ~ 1.8[m]の範囲で線形に正規化した．これによって，ロボットは 1.8[m]以上の位置にある物体は障害物として認識せず，1.8[m]以下の物体を障害物として認識する．距離の設定範囲は任意に変更可能である．

Fig5.2 に実験結果を示す．上図は実機が計測した距離データ，下図は実機の状態値の変化をプロットしている．上図の Fore，R-30，L-30，Right，Left の順に前方，右前，左前，右，左方向の超音波センサの距離データを表している．下図の YAW は実機の方位角[rad]，YAW_T は目標方位[rad]，YAW_F は制御入力である回転モーメント[Nm]を示している．なお，実機の方位角は時計回りを正としている．実機はは 55～66[sec]まで，方位角 2.8[rad]の方角に前進している．65[sec]において，計測された距離データが前，左前，左の順に 1.3，1.1，1.5[m]となっている．そのとき，ロボットの方位角は 2.8[rad]で目標角度約 4.1[rad]の回頭命令が出ている．65～74[sec]の間にかけてロボットは右に回頭し，最終的に 74[sec]において実機の方位角-2.0[rad]なり，障害物がないと認識され，実機は再び前進を行っている．この実験により，障害物センサから得られる環境情報を元に障害物回避が実現できることを確認できた．

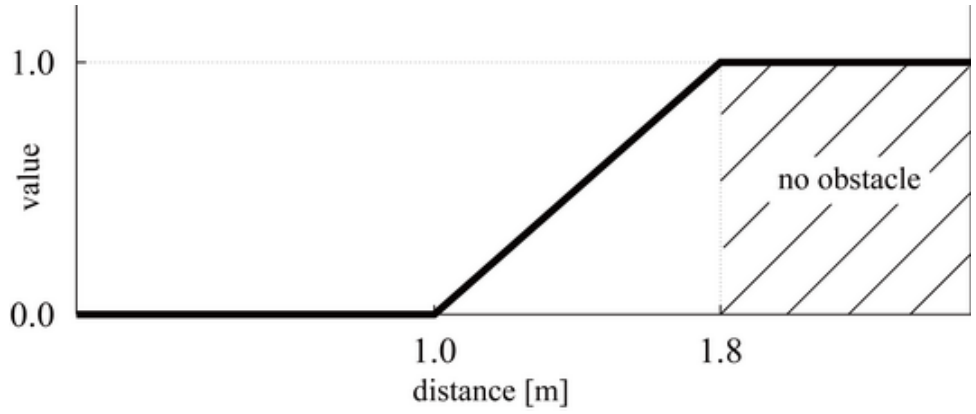


Fig 5.1 Normalized Distance

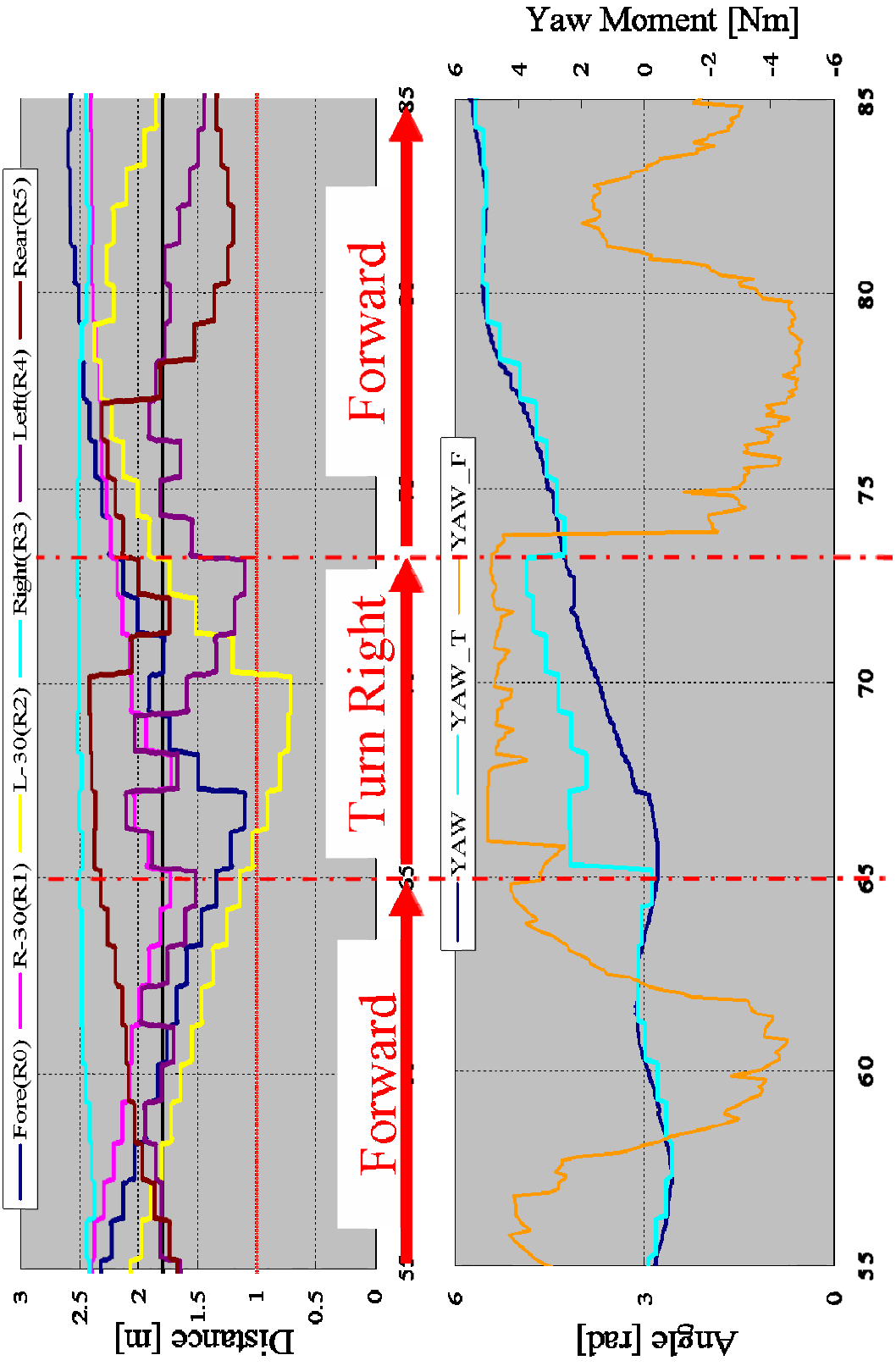


Fig 5.2 Transitions of Distance data, Yaw Angle and Yaw Moment

第6章

考察

6.1 SOM の学習における学習パラメータの影響

6.1.1 乱数の影響

SOM の学習では，はじめに競合層を乱数で初期化する．これによる学習結果への影響を調査した．4.1.1 で述べた障害物回避マップの作成を 10 回行った．競合層のユニット数を 20x20 とし 0 から 1 の乱数で初期化し，学習率を 0.1，学習回数を 5,000 回行った結果のマップを Fig6.1 に示す．全く同じ結果が得られるわけではないが，全てのマップにおいて，各ユニットの持つ回避方向の情報の位置関係はほぼ等しい配置が得られている．

得られたマップの違いを定量的に評価するために，学習終了時において，式(5.1)に示すような評価関数を定義した．

$$E = \frac{1}{2KM} \sum_k \sum_i \left\{ \Psi_i^k \cdot \left(\boldsymbol{x}_i - \boldsymbol{w}^k \right) \right\}^2 \tag{6.1}$$

M は入力データ，K は競合層のユニットの総数，$\boldsymbol{x}_i$ は i 番目の入力データ，$\boldsymbol{w}^k$ は k 番目のユニットの重みベクトル，Ψ_i^k は式(2.27)で定義した学習分配率である．式(5.1)は入力に対してマップの広がりを考慮した評価関数となっており，SOM のアルゴリズムはこの評価関数を概ね最小化するように動作する．評価値の平均値，分散，最大値，最小値を求めた結果を Table5.1 に示す．入力ベクトルは 0 から 1 で正規化されていることを考慮すると，評価値の平均値は 8.864 $\times 10^{-4}$ で，距離情報の単位[m]から十分小さいといえる．また，分散が 8.332×10^{-10} であることなどから，SOM の学習によって得られた障害物回避マップは，初期値によらず安定した結果が得られといえる．

6.1.2 競合層の大きさの影響

学習回数とマップサイズによる影響の違いを比較した．Fig6.2 には学習回数による比較を示している．5,000 回のときの値を 1 として 1,000 回，10,000 回と比較した．評

価値の平均値に関してはどの場合もほとんど差はなかったが，分散は学習回数が進むにつれて減少していることが分かる．Fig6.3 には，競合層のユニット数に関する比較を示している．20x20 のときの値を 1 として 10x10，40x40 と比較した．評価値の平均値，分散ともにユニット数の増加に従って減少することが分かる．しかしながら，学習回数およびユニット数を増加させることは，学習時間の増加と等価である．競合層のユニットはここで，ロボットの状態と行動の組を表現している．入力した 64 種の状態を表現する，追加学習をするのに十分なユニット数を確保するために，4.1 節では，学習回数 5000，ユニット数 20x20 を選択している．

6.2　3 次元空間での障害物回避

4.1.1 項で提案した障害物回避手法は水平面上の障害物の距離情報と回避方向のみを表現している．実際の水中ロボットの運動と同様に，3 次元の運動への適用を考察する．例えば，矢状面上（左右軸に垂直な面）での障害物回避マップを作成することを考える．矢状面にも水平面と同様の方向の距離情報が得られるとすると，Fig4.1 に示した教示ベクトルの要素を$\left(r_{5,},r_4,r_3,r_2,r_1,r_0,t_x,t_z\right)$のように変更し，入力データの作成及び学習を行うことで適用可能である．ここで，r_i (i = 0 ~ 5)は昇順に前方，上下前方，上下，後方の距離データを，(t_x, t_z)はそのときの矢状面での取るべき回避方向の角度を単位円上の座標で表現しているものとする．3.1.1 項で示した学習プロセスを踏むことで，矢状面での回避マップを得ることが出来る．

6.3 PID 制御と提案制御手法の比較

PID 制御器との制御性能比較試験を行った．PID 制御とは，入力値の制御を出力値と目標値との偏差，その積分，及び微分の三要素によって行う手法である．入力値は以下のように定義される．

$$x(t) = K_P e(t) + K_I \sum_{k=0}^{t} e(k) + K_D \{e(t) - e(t-1)\} \tag{6.2}$$

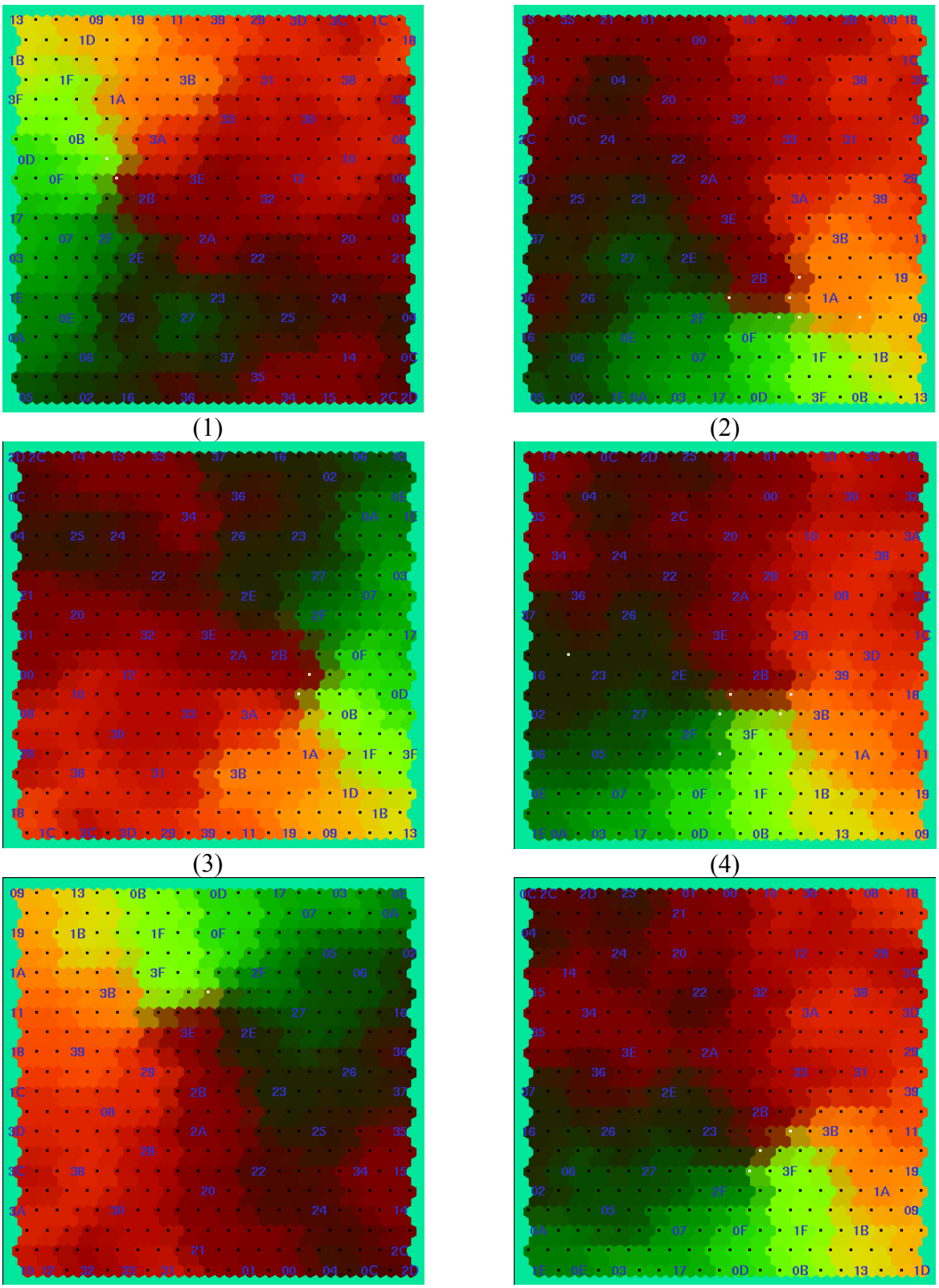
ここで，$x(t)$，$e(t)$，はそれぞれ時間 t における入力量と偏差を示しており，K_P，K_I，K_D はそれぞれ，比例，積分及び微分の要素にかかる重み（ゲイン）を表現している．

PID 制御器のゲインは，$(M, C) = (80, 25)$の状態に対して Ziegler and Nichols によって提案された限界感度法を適用し求まった値を参考にし，$K_P = 54$，$K_D = 88$，$K_I = 8.3$ とした．提案手法及び PID 制御器の初期状態における制御試験結果を Fig5.4-(a)に示す．赤実線は目標値，緑破線及び赤点線はそれぞれ提案制御手法の制御対象の状態量及び制御入力を，青点線及び水色一点差線はそれぞれ PID 制御器の制御対象の状態量及び制御入力を示している．ここで，制御器の出力値は±10[N]に制限している．全ての状態量は正規化を行っており，実際の制御入力に対して 10 分の 1 のスケールでプロットしている．Fig5.4 から，提案制御手法のほうが PID 制御器よりもオーバシュートも制御入力も小さい良好な結果を得ていることが分かる．制御力に関しても，提案手法のほうが小さな値で制御できていることが分かる．

初期状態$(M, C) = (80, 25)$に対して，外乱として 2.0[N]の外力を与えた場合の制御結果を Fig5.4-(b)に示す．PID 制御器では正方向のオーバシュートが増大しているが，提案手法では外乱による影響は少ない．最終的には，どちらの制御器も-2.0[N]の推力を保って目標値に留まっている．定常的な外乱のある環境における適応性は良好であることが確認できた．

また，3.2.3 項で行った適応性能試験と同様に矩形目標入力に対する追従試験を行った．初期状態$(M, C) = (80, 25)$からパラメータを(95, 40)，(120, 120)へ変化させたときの

適応性能を提案手法と PID 制御器で比較した結果を Fig5.5 に示す．提案手法では適応後に得られた制御器を用いた結果を示している．PID 制御のゲインは上記の試験と同じく $K_P = 54$，$K_D = 88$，$K_I = 8.3$ とした．緑実線及び青点線はそれぞれ提案手法と PID 制御器を用いた場合の状態量，赤破線は目標状態量を示している．初期状態である Fig5.4-(a)と比較すると，PID 制御器においてはどちらの状態のときもオーバシュートは減少しているが，目標値へ収束する時間が遅くなっている．特に，$(M, C) = (120, 120)$のとき(Fig5.5-(b))では，今回設定した 25[sec]内に目標値へ収束することが出来なかった．提案手法では，どちらの場合においても，調整後には良好な制御器を得ることが出来ている．

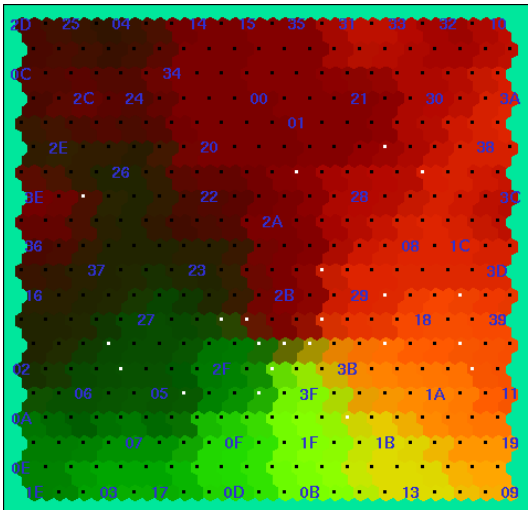


(1) (2)

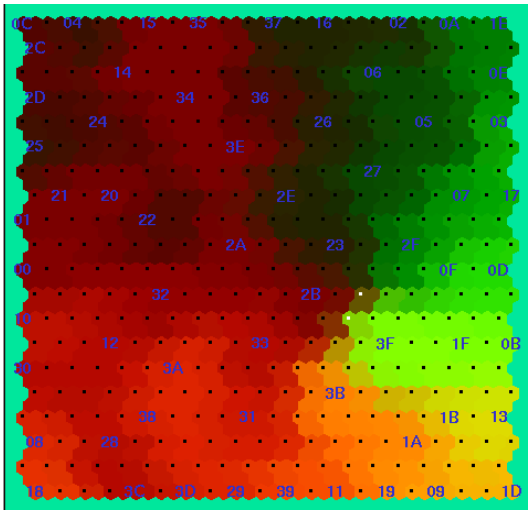
(3) (4)

(5) (6)

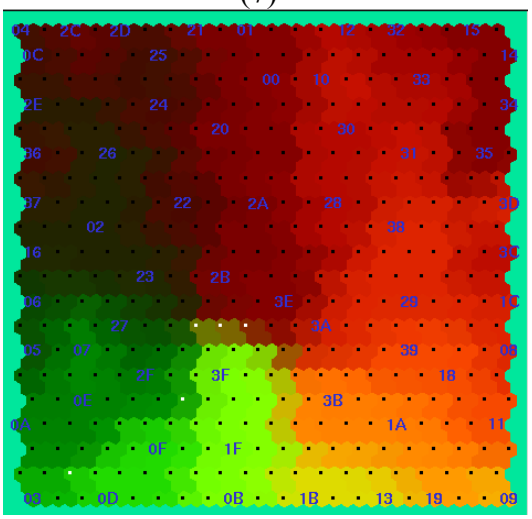
Fig 6.1 Comparison of Difference depend on Initial States



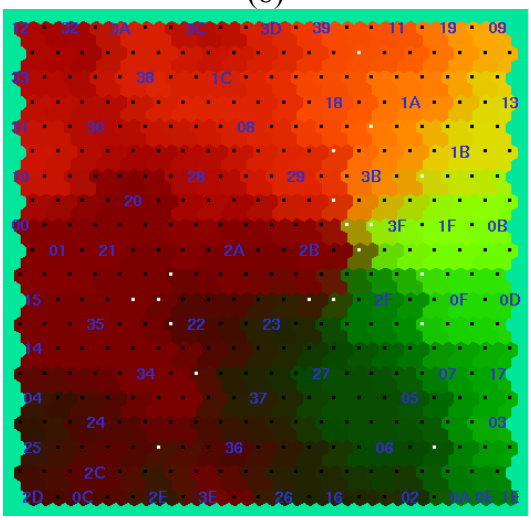
(7)



(8)



(9)



(10)

Fig 6.1 Comparison of Difference depend on Initial States

Table5.1 Evaluation Value of Learning
(map size: 20x20, iteration: 5000, learning rate: 0.1)

Average	8.864×10^{-4}
Variance	8.332×10^{-10}
Max	9.330×10^{-4}
Min	8.410×10^{-4}

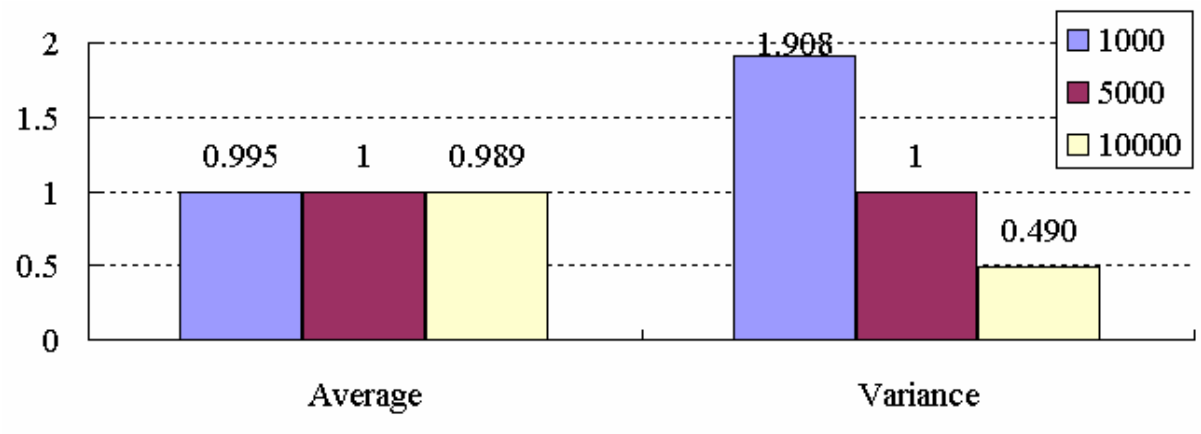


Fig 6.2 Comparison of Evaluation Value by the difference of iteration

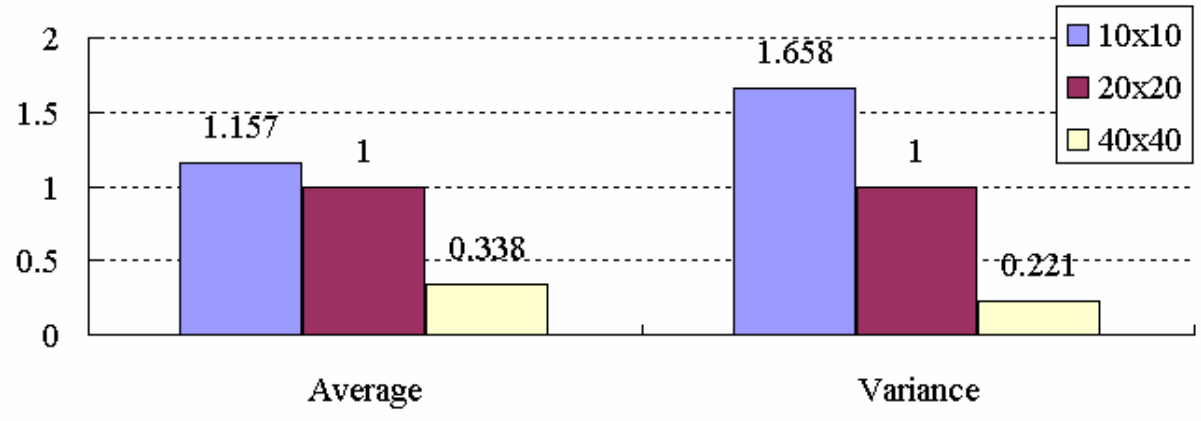
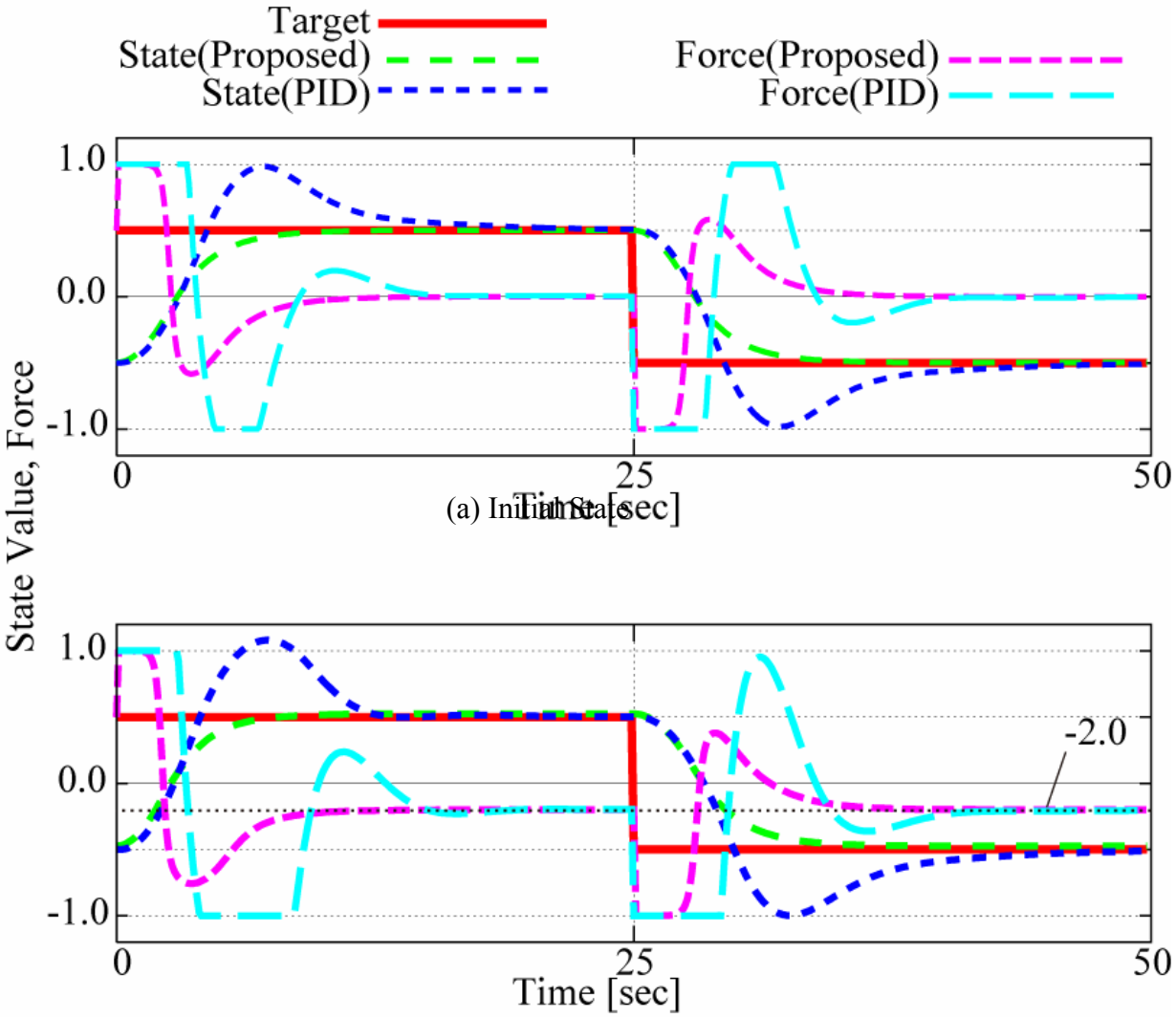


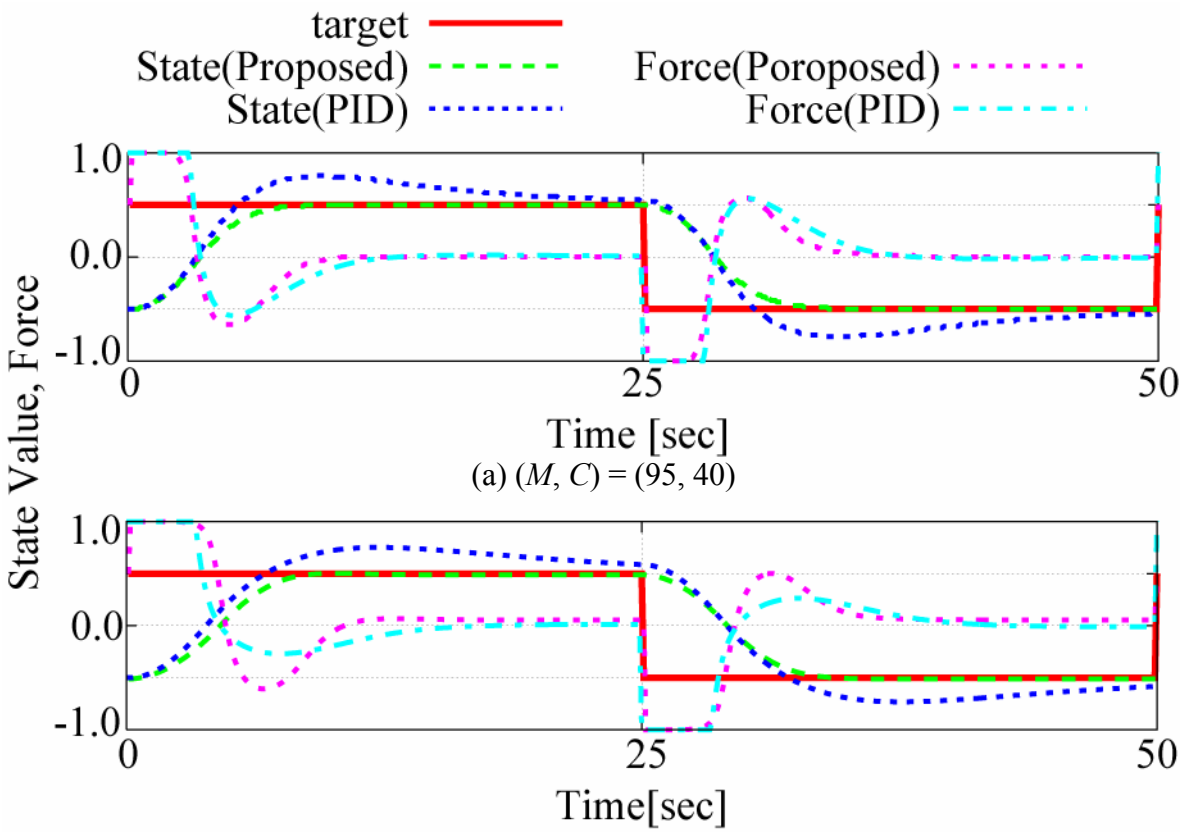
Fig 6.3 Comparison of Evaluation Value by the difference of unit number



(a) Initial State

(b) With Disturbance

Fig 6.4 Comparison between Proposed Controller and PID controller for disturbance



(a) (*M*, *C*) = (95, 40)

(b) (*M*, *C*) = (120, 120)

Fig 6.5 Comparison between Proposed Controller and PID Controller for Changing Property

第7章

結論

自律型水中ロボットの実現は，海洋開発等の分野において期待されている．解決しなければならない問題も多くある．複雑で非線形性を持った動特性を考慮した運動制御，搭載されたセンサの限られた情報による状態及び環境の把握し障害物回避や自己位置同定を行うこと，アクチュエータやセンサの開発など様々である．

本論文では，第 4 章において，水中ロボットの自己組織的行動獲得システムの提案を行い，脳型情報処理手法を用いてその基本システムとなる SOM を用いた環境認識と行動獲得システム及び mnSOM を用いた水中ロボットの運動の同定と制御システムの構築を行った．

4.1.1 項において述べた SOM を用いた環境認識と行動獲得システムでは，水中ロボットの障害物回避に適用しシミュレーション及び実験によって有効性を示した．障害物の有無とそのときの望ましい回避方向の組を入力ベクトルとして SOM に与えることで障害物回避に関する単純なルールを表現した障害物回避マップを生成できた．第 5 章において，水中ロボットの障害物回避システムとして適用し有効性を実験を通して確認した．

4.1.2 項では，行動軌跡を評価することによる初期マップへ追加学習法を提案し，シミュレーションによって有効性の検証を行った．目的に応じた評価関数を設定することで，目的とするミッションやフィールドおける設計者の意図した行動を自己組織的に得ることが出来た．

4.2 節においては，mnSOM を用いた水中ロボットの運動の同定と制御システムでは，動特性の異なる水中ロボットの運動時系列をリカレントニューラルネットワーク型の mnSOM を用いて学習することで，同時に複数の動特性を表現したリカレントニューラルネットワークを得るとともに，入力時系列間の関係，すなわち動特性の関係を表したフォワードモデルマップを獲得した．同時に，フォワードモデルマップ内に入力時系列群の中間の状態を表現したネットワークを獲得できた．また，フォワードモデルマップに対応した複数の制御器を表現したコントローラマップの取得を行った．獲得したフォワードモデルマップとコントローラマップを用いた制御手法と従来制御手法との適応性能比較試験を行いその有効性を示した．水中ロボットの運動の特性の変化への適応が可能であることを示した．

6 章で述べたように，SOM のアルゴリズムは安定した結果を出すことが出来る．一方，問題に合わせて競合層のユニットやモジュールの数，競合層の次元や形状を適切に設定しなければ，望んだ結果（必要としている情報をマップから読取れる）は得られない．経験やノウハウが必要となる場合もあり，様々なシミュレーション及び実験による検証が必要である．

今後，自己組織的行動獲得システムを実現していく上で必要となる技術要素としては以下のようなものが挙げられる．環境認識システムと運動制御システムに分けて以下に述べる．

環境認識システム

- センサ情報の追加
  - ロボット自身の状態（姿勢，速度及び加速度等）
  - ロボットの位置
- ユニットの幾何学的情報の利用
  - マップを用いた目標状態の決定法
  - 経路計画法
- 追加学習法
  - ロボットが取得した情報から逐次追加学習

運動制御システム

- コントローラの評価値によるマッピング
  - 即応，省エネ
- 運動モード間の影響を考慮したマップの作成
  - 動特性及び制御器ネットワークを拡張し，表現応力を向上させる

自己組織的行動獲得システムの実現には，行動計画法，行動目標による制御器の選択などに対する解決手法の提案及び更なるシミュレーション及び実験による検証が必要であるが，本論文で示した結果は，自己組織的行動獲得システム実現に貢献できた．

Appendix

A) SOM ソースプログラム

3.2.2 項で紹介した自己組織化マップの適用例である動物の分類のソースプログラムを示す.

機能によって分割してソースを作成している．大きく分けて SOM の学習アルゴリズムの実装している部分とプログラムの実行処理を行う部分とに分かれている．プログラムのソースファイルリストを以下に，次頁にソースを示す.

- main.c
  - 学習の実行処理
  - 画面表示
- som.c
  - 入出力データファイル処理
  - 学習アルゴリズムの実装
- som.h
  - 各種パラメータ設定

```
/*******************************
        som program main.c
        S.Nishida
*******************************/
#include"som.h"
#include"SOMg.h"
#include<time.h>

#define   WindowSize        500

double    _alph;            // Learning Rate
int       _range;           // Neighboorhood Area
int       _loop;            // Learning Iteration

SOMCanvas       somc;

extern    DATA    _input[DataNum];          // input data
extern    UNIT    _unit[XDIM*YDIM];         // vector units

extern    void    _Rand_Init(void);         // Initialization of all Units randomly
extern    void    _Input_Data(void);        // Read input data
extern    void    _CalcDiff(void);          // Calculate difference between neighboors
extern    void    _Competition(void);       // Competitive Process
extern    void    _Cooperation(void);       // Cooperative Process
extern    void    _Adaptation(void);        // Adaptive Process
extern    void    _Output_Data(void);       // Save Map data

void
LabelClear(void)
{
        int       i;

        for(i = 0; i < XDIM*YDIM; i++)
                memset(_unit[i].label, 0, LabelNum*DataNum);
}

/*      Screen-Displayed Func.      */
void
DrawMap(int t)
{
        int       i, j, k;
        char      str[64];
        double    val[XDIM*YDIM] = {0.0};

        sprintf(str, "ITERATION = %d", t);
        SOMgWriteMessage(str);

        for(i = 0; i < XDIM*YDIM; i++){
                val[i] = _unit[i].diff;
                SOMgClearWithBG(i, val[i]);
                for(k = 0; k < DataNum; k++)
                        SOMgWriteText(i,k,_unit[i].label[k]);
        }
        LabelClear();
```

```
	XfFlush();
}

/*	Nighboorhood Func.	*/
int
RangeFunc(int t)
{
	int	sigma;

	sigma = (int)(r_max-r_min)*exp(-(double)t/TAU)+ r_min;		//Gauss
	return	sigma;
}

/*	Learning Rate Func.	*/
double
AlphFunc(int t)
{
	//return	alph0*(1.0 - (double)t/loop0);		//liner
	return	alph0 * exp( -( (double)t / (loop0) ));	//Gauss
}

/*	Execution Process */
int
main(int argc,char **argv)
{
	int	num;
	int	button;
	int	i, j;
	int	check = 0;
	time_t	time0, time1;


	//printf(stderr, "Check%d¥n",check++);

	_Rand_Init();
	_Input_Data();

	somc = SOMgCreateSOMCanvas(XDIM, YDIM, WindowSize, 0, HEX1, "animal");
	_Competition();
	DrawMap(_loop);
	button = XfWaitButton();
	time0 = clock();

	for(_loop=0;_loop<loop0;_loop++){
		_range = RangeFunc(_loop);
		//_alph = AlphFunc(_loop);

		_Competition();
		_Cooperation(); //_Adaptation();

		_CalcDiff();
/*
		if((_loop+1)%(loop0/5) == 0){
			fprintf(stderr,"%10d: %d¥n", loop0-_loop-1, _range);
```

```
                        DrawMap(_loop+1);
                        //button = XfWaitButton();
                }
                else if(_loop < (loop0/10)){
                        fprintf(stderr,"%10d: %d¥n", loop0-_loop-1, _range);
                        DrawMap(_loop+1);
                        //button = XfWaitButton();
                }
                else{}
*/
                fprintf(stderr,"%10d: %d¥n", loop0-_loop-1, _range);
                DrawMap(_loop+1);
                //button = XfWaitButton();
                LabelClear();
        }

        _Output_Data();
        time1 = clock();
        printf("%.2f¥n", (double)(time1, time0)/CLOCKS_PER_SEC);
        //button = XfWaitButton();
        button = XfWaitButton();

        return 0;
}
```

```
/*******************************
        som.c
        program : S.Nishida
*******************************/
#include"som.h"

double  _alph;              // Learning Rate
int     _range;             // Neighboor Area
int     _Dim[3] = {VDIM,XDIM,YDIM};
int     _n;
double  _err[XDIM*YDIM][VDIM], _h[DataNum][XDIM*YDIM];

/*      Calc Distance on Conpetitive Layer  */
/*      Hexagonal Topology                  */
float
hexa_dist(int bx, int by, int tx, int ty)
{
        float   ret, diff;

        diff = bx - tx;

        if(((by - ty) % 2) != 0){
                if((by % 2) == 0){
                        diff -= 0.5;
                }
                else{
                        diff += 0.5;
                }
        }
        ret = diff * diff;
        diff = by - ty;
        ret += 0.75 * diff * diff;
        ret = (float) sqrt((double)ret);

        return(ret);
}

/*      Calclate Euclid Distance between units      */
double
_Euclid_Dist(double *u1, double *u2)
{
        int     i;
        double  length2 = 0.0;

        for(i = 0; i < VDIM; i++)
                length2 += (u1[i]-u2[i])*(u1[i]-u2[i]);

        return  sqrt(length2);
}

/*      Calclate Dot Product between units  */
double
_Dot_Product(double *u1, double *u2)
{
```

```
	int	i;
	double	length = 0.0, sum1 = 0.0,sum2 = 0.0;
	double	nu1[VDIM], nu2[VDIM];

	for(i = 0; i < VDIM; i++){
		sum1 += u1[i];
		sum2 += u2[i];
	}
	for(i = 0; i < VDIM; i++){
		nu1[i] = u1[i]/sqrt(sum1);
		nu2[i] = u2[i]/sqrt(sum2);
	}

	for(i = 0; i < VDIM; i++)
		length += nu2[i]*nu1[i];

	return	length;
}

/*	Coordinate of Unit -> Index No. of Unit		*/
int
_UnitNo(int x, int y)
{
	if(x < 0 || y < 0 || x > (XDIM - 1) || y > (YDIM - 1))
		return	-1;
	else
		return	(x + y * XDIM);
}

/*	Calculate Distance between Neighboors		*/
void
_CalcDiff(void)
{
	int	i, j, NeighNum;
	int	x, y;
	double	DiffSum;

	for(i = 0; i < XDIM*YDIM; i++){
		x = i % XDIM;
		y = i / XDIM;

		DiffSum = 0.0;
		NeighNum = 0;

		if((j = _UnitNo(x - 1, y)) >= 0){ /* Left Unit */
			DiffSum += _Euclid_Dist(_unit[i].data, _unit[j].data);
			NeighNum++;
		}
		if((j = _UnitNo(x + 1, y)) >= 0){ /* Right Unit */
			DiffSum += _Euclid_Dist(_unit[i].data, _unit[j].data);
			NeighNum++;
		}
		if(i%2){ /* Even row */
			if((j = _UnitNo(x, y - 1)) >= 0){ /* Upper Left Unit */
```

```
                                        DiffSum += _Euclid_Dist(_unit[i].data, _unit[j].data);
                                        NeighNum++;
                                }
                                if((j = _UnitNo(x + 1, y - 1)) >= 0){ /* Upper Right Unit */
                                        DiffSum += _Euclid_Dist(_unit[i].data, _unit[j].data);
                                        NeighNum++;
                                }
                                if((j = _UnitNo(x, y + 1)) >= 0){ /* Bottom Left Unit */
                                        DiffSum += _Euclid_Dist(_unit[i].data, _unit[j].data);
                                        NeighNum++;
                                }
                                if((j = _UnitNo(x + 1, y + 1)) >= 0){ /* Bottom Right Unit */
                                        DiffSum += _Euclid_Dist(_unit[i].data, _unit[j].data);
                                        NeighNum++;
                                }
                        }
                        else{ /* Odd row */
                                if((j = _UnitNo(x, y - 1)) >= 0){ /* Upper Left Unit */
                                        DiffSum += _Euclid_Dist(_unit[i].data, _unit[j].data);
                                        NeighNum++;
                                }
                                if((j = _UnitNo(x + 1, y - 1)) >= 0){ /* Upper Right Unit */
                                        DiffSum += _Euclid_Dist(_unit[i].data, _unit[j].data);
                                        NeighNum++;
                                }
                                if((j = _UnitNo(x, y + 1)) >= 0){ /* Bottom Left Unit */
                                        DiffSum += _Euclid_Dist(_unit[i].data, _unit[j].data);
                                        NeighNum++;
                                }
                                if((j = _UnitNo(x + 1, y + 1)) >= 0){ /* Bottom Right Unit */
                                        DiffSum += _Euclid_Dist(_unit[i].data, _unit[j].data);
                                        NeighNum++;
                                }
                        }
                        _unit[i].diff = DiffSum/NeighNum;
                }

}

/*      Initialize all units randomly */
void
_Rand_Init(void)
{
        int     i,j;

        randomize();

        for(i = 0; i < XDIM*YDIM; i++){
                for(j = 0; j < VDIM; j++){
                        _unit[i].data[j] = urandom(1.0)+0.5;
                        //_unit[i].data[j] = frandom();
                }
        }
        fprintf(stderr,"_rand_init¥n");
```

```
}

/*      Read input data     */
void
_Input_Data(void)
{
        FILE    *ifp;
        int     i,j;

        ifp = fopen(INPUT_FILENAME,"r");
        for(j=0; j < DataNum; j++){
                for (i=0;i<VDIM;i++){
                        fscanf(ifp,"%lf ",&_input[j].data[i]);
                }
                fscanf(ifp,"%s",_input[j].label);
        }
        fclose(ifp);

        fprintf(stderr,"_Input_Data¥n");
}

/*      Evaluation Process          */
double
_Evaluation(int i, int j, int k)
{
        double  length;

        length = _Euclid_Dist(&_input[k].data[0],&_unit[i+j*XDIM].data[0]);

        return  length;
}

/*      Competitive Process         */
void
_Competition(void)
{
        float   length = 0.0, min = 100.0, max = -1.0;
        int     i, j, k;

        for(k = 0; k < DataNum; k++){
                for(j=0;j<YDIM;j++){
                        for(i=0;i<XDIM;i++){
                                length = _Evaluation(i, j, k);
                                if(min > length){
                                //if(max < length){
                                        _input[k].wx = i;
                                        _input[k].wy = j;
                                        min = length;
                                        //max = length;
                                }
                                else{}
                        }
                }
                i = _input[k].wx;
```

```
                j = _input[k].wy;

                memcpy(_unit[i+j*XDIM].label[_unit[i+j*XDIM].winner], _input[k].label, 32);
                //printf("winner %d %d¥t", _input[k].wx, _input[k].wy);
                _unit[i+j*XDIM].winner++;

                min = 100;
                max = -1;
        }
        for(i=0;i<XDIM*YDIM;i++)
                _unit[i].winner = 0;
}

/*      Neighborhood Function       */
double
NFunc(int x1, int y1, int x2, int y2, int sigma)
{
        double  dist;
        double  dist2, sigma2;
        double  u;

        dist = hexa_dist(x1,y1,x2,y2);
/*
        sigma2 = sigma*sigma;
        dist2 = dist*dist;
*/
        u = fabs(dist/sigma);

        return  exp(-pow(u, 1.0/(1.0 + BETA)));
}

/*      Cooperative Process         */
void
_Cooperation(void)
{
        int     i, j, k;
        double  sum = 0.0;

        for(j = 0; j < XDIM*YDIM; j++){
                for(i = 0; i < DataNum; i++){
                        _h[i][j]= NFunc(_input[i].wx, _input[i].wy, j%XDIM, j/XDIM, _range);
                        for(k = 0; k < VDIM; k++)
                                _unit[j].data[k] += alph0 * _h[i][j] * (_input[i].data[k] -
_unit[j].data[k]);
                }
        }
}

/*      Adaptive Process   */
void
_Adaptation(void)
{
        int     i,j,k;
```

```
		for(k = 0; k < DataNum; k ++)
			for(i=0;i<XDIM*YDIM;i++)
				for(j=0;j<VDIM;j++){
					_unit[i].data[j] += alph0 * _h[k][i] * (_input[k].data[j] -
_unit[i].data[j]);
				}
		memset(_err,0,XDIM*YDIM*VDIM*sizeof(double));
}

/*		Save Map file		*/
void
_Output_Data()
{
		int		i,j,k,num;
		FILE	*ofp;

		ofp = fopen(OUTPUT_FILENAME,"w");
		//fprintf(ofp,"%d hexa %d %d bubble¥n",_Dim[0],_Dim[1],_Dim[2]);

		for(i = 0; i < XDIM*YDIM; i++){
			memset(_unit[i].label,0,8*DataNum);
		}
		_Competition();

		for(j=0;j<YDIM;j++){
			for(i=0;i<XDIM;i++){
				for(k=0;k<VDIM;k++){
					fprintf(ofp,"%f ",_unit[i+j*XDIM].data[k]);
				}
				fprintf(ofp,"%s¥n",_unit[i+j*XDIM].label[0]);
			}
			fprintf(ofp,"¥n");
			fprintf(ofp,"¥n");
		}
		fclose(ofp);
		fprintf(stderr,"_Output_Data¥n");
}
```

```
/*******************************
        som.h
        program : S.Nishida
*******************************/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<random.h>

#define  INPUT_FILENAME         "input_animal.data"
#define  OUTPUT_FILENAME        "animal-online.dat"
#define  XDIM           15      // Size of Competitive Layer
#define  YDIM           15
#define  VDIM           21      // Dim of Input Vector Unit
#define  DataNum        17      // Num of Input Vector

#define  r_max  sqrt(XDIM*XDIM+YDIM*YDIM)   // Initial Size of Neighborhood Function
#define  r_min  1                           // Final Size

#define  LabelNum       32      // Label Characters
#define  loop0          300     // Iteration
#define  alph0          0.2     // Learning Rate
#define  BETA           -0.5    // range :-1 ~ infinity; -0.25: Sub, -0.5: Gaussian, 1: Super
#define  TAU            20      // Time constant value

typedef  struct
{
        double  data[VDIM];
        int     wx, wy;         // Coordinate of BMU corresponding to a input
        char    label[LabelNum];
}DATA;

typedef  struct
{
        int     x,y;            // Coordinate in Competitive Layer

        int     num;            // Index Num
        int     winner;         // winner or not
        double  data[VDIM];
        double  diff;           //Distance between Neighbors
        char    label[DataNum][LabelNum];
}UNIT;

DATA    _input[DataNum];
UNIT    _unit[XDIM*YDIM];
```

B) mnSOM ソースプログラム

3.3.3 項で紹介した mnSOM の適用例である 3 次 Legendre 正規化関数を用いた 3 次元多項式の分類に用いたプログラムのソースファイルを示す.

SOM のプログラムと同様に，機能によって分割してソースを作成している．プログラムのソースファイルリストを以下に，次頁にソースを示す.

- main.c
  - 学習の実行処理
  - 画面表示
- mnsom.c
  - mnSOM 学習アルゴリズムの実装
- bp.c
  - 誤差逆伝播法の実装
- parameter.h
  - 各種パラメータ設定
- mnsom.h
  - モジュール構造体の定義
  - クラス構造体の定義
  - mnSOM 学習プロセス関数のプロトタイプ宣言
- bp.h
  - ニューラルネットワーク構造体の定義
  - 誤差逆伝播法学習プロセス関数のプロトタイプ宣言

```
/****************
        main.c
****************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <random.h>
#include "SOMg.h"
#include "parameters.h"
#include "bp.h"
#include "mnsom.h"

#define Freq_Graph 1

void ReadData(void);
void GraphMap(int);

SOMCanvas somc;

const   double  a[6][3] =           // Teaching Class Data
{       0.5,    0.5,    0.5,
        -0.5,   -0.5,   -0.5,
        1.0,    0.0,    0.0,
        -1.0,   0.0,    0.0,
        0.0,    0.0,    1.0,
        0.0,    0.0,    -1.0
};

double  b[3][N_MODULE] = {0.0};                 // LegendreCoeffcient on k-th Module
double  PSI[N_MODULE][N_CLASS] = {0.0};         // Learning Distributed Rate

/*      Calculate Legendre Coefficient form Learning Distributed Rate     */
void
CalcLegendreCoefficient(void)
{
        int i, j, k;

        for(k = 0; k < N_MODULE; k++)
                for(j = 0; j < 3; j++)
                        for(i = 0; i < N_CLASS; i++)
                                b[j][k] += module[k].psi[i]*a[i][j];
}

/*      Save Legendre Coefficient for graph   */
void
PrintLegendreCoefficient(int t)
{
        int     i, j, k;
        int     x, y;
        double  sum = 0.0;

        for(y = 0; y < N_Y; y++){
                for(x = 0; x < N_X; x++){
                        for(j = 0; j < 3; j++){
```

```
                                        printf("%f¥t", b[j][x+y*N_X]);
                                }
                                printf("¥n");
                        }
                        printf("¥n");
                }
                printf("¥n");

                for(x = 0; x < N_X; x++){
                        for(y = 0; y < N_Y; y++){
                                for(j = 0; j < 3; j++){
                                        printf("%f¥t", b[j][x+y*N_X]);
                                }
                                printf("¥n");
                        }
                        printf("¥n");
                }
                printf("¥n");
}

/*      Execution Process */
int
main(int argc, char **argv)
{
   int t;
   int i,k;

   ReadData();
   //MNSOM_Initialize_Hex1();
   MNSOM_Initialize_Square();
   somc=SOMgCreateSOMCanvas(N_X, N_Y, 400, 0, SQUARE, "mnSOM: Map of Qubic Function
Family");
   SOMgSetGraphAreaAll(-1.0, -1.0, 1.0, 1.0);
   //for(t=0; t <= ITERATION; t++){
   for(t=0; ; t++){
      if (t%Freq_Graph==0) {
         if (XfAskButton()==3) break;
         if (XfAskButton()==1) XfWaitButton();;
         GraphMap(t);
      }
      MNSOM_Training(t);
   }
   CalcLegendreCoefficient();
   PrintLegendreCoefficient(t);
   XfWaitButton();;
}

/*      Read input data of all classes*/
void
ReadData(void)
{
   int c,d;
   double x,y;
   FILE *fp;
```

```
    char buffer[100];

    if ((fp=fopen(DATAFILE,"r"))==NULL){
        fprintf(stderr, "File not found.¥n");
        exit(1);
    }
    for(c=0; c<N_CLASS; c++){
        for(d=0; d<N_DATA; d++){
            fscanf(fp,"%lf %lf",&x,&y);
            class[c].data[d].X[0]=x;
            class[c].data[d].T[0]=y;
        }
        sprintf(class[c].label, "P%d",c+1);
    }
    fclose(fp);
}

/*          Screen Displayed Function   */
void
GraphMap(int t)
{
    int m, wc, d;
    char str[200];
    double diff[N_MODULE];
    DataVector dv;

    sprintf(str,"T=%d                              ", t);
    SOMgWriteMessage(str);

    MNSOM_CalcModuleDistance();
    for(m=0; m<N_MODULE; m++) diff[m]=module[m].NeighborDiff;
    SOMgClearAllWithBG(diff);

    for(m=0; m<N_MODULE; m++){
        SOMgSetUCanvas(m);

        dv.X[0]=-1.0;
        _FP(&module[m].mlp,dv);
        SOMgMove(dv.X[0],module[m].mlp.Y[0]);
        for(dv.X[0]=-1.0; dv.X[0]<=1.0; dv.X[0]+=0.1) {
            _FP(&module[m].mlp,dv);
            //SOMgDrawCircle(dv.X[0],module[m].mlp.Y[0],1);
            SOMgLine(dv.X[0],module[m].mlp.Y[0]);
        }
        wc=module[m].winclass;
        if (wc>=0) {
            SOMgWriteText(-1,0,class[wc].label);
            XfSetColor(RED);
            SOMgMove(class[wc].data[0].X[0],class[wc].data[0].T[0]);
            for(d=0; d<N_DATA; d++)
              //SOMgDrawCircle(class[wc].data[d].X[0],class[wc].data[d].T[0],1);
              SOMgLine(class[wc].data[d].X[0],class[wc].data[d].T[0]);
            XfSetColor(BLACK);
        }
```

```
    }
    XfFlush();
}
```

```
/****************
        mnsom.c
****************/
#define MNSOM_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <random.h>
#include "parameters.h"
#include "bp.h"
#include "mnsom.h"

/*      Calculate Num. of Neighboors        */
void
AddNeigborList(int x, int y, int *n, int list[])
{
   if (x<0 || y<0) return;
   if (x>=N_X || y>=N_Y) return;
   list[*n]=x+y*N_X;
   (*n)++;
}

/*      Evaluative Process*/
void
EvaluativeProcess()
{
        int     i, j, k;
        int     n;

        for(k = 0; k < N_MODULE; k++)
                for(i = 0; i < N_CLASS; i++)
                        module[k].err[i] = 0.0;

        for(k = 0; k < N_MODULE; k++){
                for(i = 0; i < N_CLASS; i++){
                        for(j = 0; j < N_DATA; j++){
                                module[k].err[i] += _FP(&module[k].mlp, class[i].data[j]);
                        }
                        module[k].err[i] /= N_DATA;
                }
        }
}

/*      Initialize lattice-competitive Layer    */
void
MNSOM_Initialize_Square()
{
   int m,nx,ny,n;

   randomize();
   for(m=0; m<N_MODULE; m++){
      nx=m%N_X; ny=m/N_X;
      module[m].x = nx;
      module[m].y = ny;
```

```
        module[m].winclass=-1;
        MLP_Initialize(&module[m].mlp);
        n=0;
        AddNeigborList(nx-1, ny,     &n, module[m].NeighborList);
        AddNeigborList(nx+1, ny,     &n, module[m].NeighborList);
        AddNeigborList(nx-1, ny-1, &n, module[m].NeighborList);
        AddNeigborList(nx,     ny-1, &n, module[m].NeighborList);
        AddNeigborList(nx+1, ny-1, &n, module[m].NeighborList);
        AddNeigborList(nx-1, ny+1, &n, module[m].NeighborList);
        AddNeigborList(nx,     ny+1, &n, module[m].NeighborList);
        AddNeigborList(nx+1, ny+1, &n, module[m].NeighborList);
        module[m].Neighbor_N=n;
    }
    EvaluativeProcess();
}

/*          Initialize hexagonal competitive Layer              */
void
MNSOM_Initialize_Hex1()
{
    int m,nx,ny,n;
    double ratio;

    randomize();
    ratio=sqrt(3.)/2.;
    for(m=0; m<N_MODULE; m++){
        nx=m%N_X; ny=m/N_X;
        module[m].x = (ny%2==0) ? nx : nx+0.5;
        module[m].y = ny * ratio;
        module[m].winclass=-1;
        MLP_Initialize(&module[m].mlp);
        n=0;
        AddNeigborList(nx-1, ny, &n, module[m].NeighborList);
        AddNeigborList(nx+1, ny, &n, module[m].NeighborList);
        AddNeigborList(nx, ny-1, &n, module[m].NeighborList);
        AddNeigborList(nx, ny+1, &n, module[m].NeighborList);
        if (ny%2==0){
            AddNeigborList(nx-1, ny-1, &n, module[m].NeighborList);
            AddNeigborList(nx-1, ny+1, &n, module[m].NeighborList);
        } else {
            AddNeigborList(nx+1, ny-1, &n, module[m].NeighborList);
            AddNeigborList(nx+1, ny+1, &n, module[m].NeighborList);
        }
        module[m].Neighbor_N=n;
    }
    EvaluativeProcess();
}

/*          Competitive Process             */
void
CompetitiveProcess()
{
            int         bmm;
            int         i, j, k;
```

```
	for(k = 0; k < N_MODULE; k++)
		module[k].winclass = -1;

	for(i = 0; i < N_CLASS; i++){
		bmm = 0;
		for(k = 0; k < N_MODULE; k++){
			if(module[bmm].err[i] > module[k].err[i]){
				bmm = k;
			}
			else{}
		}
		class[i].bmm = bmm;
		module[bmm].winclass = i;
	}
}

double
Sigma(int t)
{
	return	(SIGMA_MAX-SIGMA_MIN)*exp(-((double)t)/TAU) + SIGMA_MIN;
}

/*	Neighborhood Function	*/
double
NFunc(int m1, int m2, double s)
{
	double	dist2;

	if(m1 == m2)
		return	1.0;
	else{
		dist2 = (module[m1].x-module[m2].x)*(module[m1].x-module[m2].x)
			+ (module[m1].y-module[m2].y)*(module[m1].y-module[m2].y);
		return	exp(-dist2/(2*s*s));
	}
}

/*	Cooperative Process	*/
void
CooperativeProcess(int t)
{
	int	i, j, k;
	double	sum = 0.0;

	for(k = 0; k < N_MODULE; k++){
		sum = 0.0;
		for(i = 0; i < N_CLASS; i++){
			module[k].psi[i] = NFunc(class[i].bmm, k, Sigma(t));
			sum += module[k].psi[i];
		}
		for(i = 0; i < N_CLASS; i++){
			module[k].psi[i] = module[k].psi[i]/sum;
		}
```

```
		}
	}

/*		Adaptive Process and Next Evaluative Process*/
void
Adaptive_And_Evaluative_Process()
{
		int		i, j, k;
		int		n;
		double	err;

		for(n = 0; n < N_LEARN_REPEAT; n++){
			for(j = 0; j < N_DATA; j++){
				for(i = 0; i < N_CLASS; i++){
					for(k = 0; k < N_MODULE; k++){
						err = _BP_Learning(&module[k].mlp, class[i].data[j],
module[k].psi[i]);
						module[k].err[i] = (1.0-Epsilon)*module[k].err[i] +
Epsilon*err;
					}
				}
			}
		}
	}

/*		Error between module#1 and module#2		*/
double
CalcErrNeighbor(int m1, int m2)
{
   int c;
   double err=0.;
   for(c=0; c<N_CLASS; c++)
      err+=module[m1].psi[c]*module[m2].err[c];
   return err;
}

/*		Copy weights of Best Neighbor		*/
void
CopyBestNeighbor(void)
{
		int		m,i,m2,min_m,c;
		double	Err,ErrN,min_err;

		for(m=0; m<N_MODULE; m++){
			Err = CalcErrNeighbor(m,m);
			min_m = module[m].NeighborList[0];
			min_err = CalcErrNeighbor(m,min_m);
			for(i=1; i<module[m].Neighbor_N; i++){
				m2 = module[m].NeighborList[i];
				ErrN = CalcErrNeighbor(m, m2);
				if (ErrN < min_err){
					min_m = m2;
					min_err = ErrN;
				}
```

```
                        else{}
                }
                if (min_err < Gamma * Err){
                        MLP_Copy(&module[m].mlp, &module[min_m].mlp);
                        for(c=0; c<N_CLASS; c++)
                                module[m].err[c] = module[min_m].err[c];
                }
                else{}
        }
}

/*      mnSOM Training Execution*/
void
MNSOM_Training(int t)
{
   CompetitiveProcess();
   CooperativeProcess(t);
   Adaptive_And_Evaluative_Process();
   CopyBestNeighbor();
}

/*      Calculate Distance between module#1 and module#2      */
double
module_dis(MLP m1, MLP m2)
{
   int k;
   double sum=0,err;
   for(k=0; k<N3; k++){
      err=m1.Y[k]-m2.Y[k];
      sum+=err*err;
   }
   return sum/N3;
}

/*      Sigmoid Func.      */
double
sigmoid(double x)
{
   return 1./(1.+exp(-x));
}

/*      Calculate Distance between Neighbors            */
void
MNSOM_CalcModuleDistance()
{
   int m, c, d, i, m2;
   double sum1,sum2,mean,sigma;
   for(m=0; m<N_MODULE; m++)
      module[m].NeighborDiff=0.;
   for(c=0; c<N_CLASS; c++){
      for(d=0; d<N_DATA; d++){
         for(m=0; m<N_MODULE; m++)
            _FP(&module[m].mlp,class[c].data[d]);
         for(m=0; m<N_MODULE; m++)
```

```
            for(i=0; i<module[m].Neighbor_N; i++){
               m2=module[m].NeighborList[i];
               module[m].NeighborDiff
                  +=module_dis(module[m].mlp,module[m2].mlp);
            }
         }
      }
      sum1=sum2=0;
      for(m=0; m<N_MODULE; m++){
         sum1+=(module[m].NeighborDiff/=module[m].Neighbor_N);
         sum2+=module[m].NeighborDiff*module[m].NeighborDiff;
      }
      mean=sum1/N_MODULE;
      sigma=sqrt((sum2/N_MODULE)-mean*mean);
      for(m=0; m<N_MODULE; m++)
         module[m].NeighborDiff
            =sigmoid((module[m].NeighborDiff-mean)/sigma);
   }
```

```
/****************
        bp.c
*****************/
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<random.h>
#include "parameters.h"
#include "bp.h"

/*      Initilazation of MLP            */
void
MLP_Initialize(MLP *M)
{
        int     i, j, k;

        for (j = 0; j < N2; j++){
                for (i = 0; i < N1; i++){
                        M->W1[i][j] = urandom(RND_W_WIDTH);
                        M->dW1[i][j] = 0.0;
                }
                M->B2[j] = urandom(RND_B_WIDTH);
                M->dB2[j] = 0.0;
        }
        for (k = 0; k < N3; k++){
                for (j = 0; j < N2; j++){
                        M->W2[j][k] = urandom(RND_W_WIDTH);
                        M->dW2[j][k] = 0.0;
                }
                M->B3[k] = urandom(RND_B_WIDTH);
                M->dB3[k] = 0.0;
        }
#ifdef  REC
        for (j = 0; j < N2; j++){
                for (i = 0; i < N2; i++){
                        M->Wrec[i][j] = urandom(RND_W_WIDTH);
                        M->dWrec[i][j] = 0.0;
                }
        }
#endif
}

/*      Copy weights to M1 form M2      */
void
MLP_Copy(MLP *M1, MLP *M2)
{
        int     l, m, n;

        for(m = 0; m < N2; m++){
                for(l = 0; l < N1; l++){
                        M1->W1[l][m] = M2->W1[l][m];
                        M1->dW1[l][m] = 0.0;
                }
                M1->B2[m] = M2->B2[m];
```

```
                M1->dB2[m] = 0.0;
        }
        for(n = 0; n < N3; n++){
                for(m = 0; m < N2; m++){
                        M1->W2[m][n] = M2->W2[m][n];
                        M1->dW2[m][n] = 0.0;
                }
                M1->B3[n] = M2->B3[n];
                M1->dB3[n] = 0.0;
        }
}

/*      derivative of tanh   */
double
dTanh(double y)
{
   return 1.-y*y;
}

/*      Forward Propagation          */
double
_FP(MLP *M, DataVector d)
{
        int     l, m, n;
        double  Err = 0.0, err_tmp;

        for (m = 0; m < N2; m++){
                M->H[m] = M->B2[m];
                for (l = 0; l < N1; l++)
                        M->H[m] += M->W1[l][m] * d.X[l];
                M->H[m] = tanh(M->H[m]);
        }
        for (n = 0; n < N3; n++){
                M->Y[n] = M->B3[n];
                for (m = 0; m < N2; m++)
                        M->Y[n] += M->W2[m][n] * M->H[m];
                M->Y[n] = tanh(M->Y[n]);
                err_tmp = d.T[n] - M->Y[n];
                Err += err_tmp*err_tmp;
        }
        return  Err/N3/2.0;
}

/* Calculate Error between output of MLP and desired data          */
void
CalcError(MLP *M, DataVector d)
{
        int     l, m, n;
        for(n = 0; n < N3; n++)
                M->E3[n] =    (d.T[n] - M->Y[n])* dTanh(M->Y[n]);

        for(m = 0; m < N2; m++){
                M->E2[m] = 0.0;
                for(n = 0; n < N3; n++)
```

```
                        M->E2[m] += M->E3[n]*M->W2[m][n];
                M->E2[m] *= dTanh(M->H[m]);
        }
}

/*      Back Propagation  */
void
_BP(MLP *M, DataVector d, double psi)
{
        int     l, m, n;
        double  eta, alpha;

        eta = Eta * psi;
        alpha = Alpha * psi;

        for(n = 0; n < N3; n++){
                for(m = 0; m < N2; m++){
                        M->dW2[m][n] = alpha*M->dW2[m][n] + eta*M->E3[n]*M->H[m];
                        M->W2[m][n] += M->dW2[m][n];
                }
                M->dB3[n] = alpha*M->dB3[n] + eta*M->E3[n];
                M->B3[n] += M->dB3[n];
        }
        for(m = 0; m < N2; m++){
                for(l = 0; l < N1; l++){
                        M->dW1[l][m] = alpha*M->dW1[l][m] + eta*M->E2[m]*d.X[l];
                        M->W1[l][m] += M->dW1[l][m];
                }
                M->dB2[m] = alpha*M->dB2[m] + eta*M->E2[m];
                M->B2[m] += M->dB2[m];
        }
}

/*      Back Propagation Learning Execution             */
double
_BP_Learning(MLP *m, DataVector d, double a)
{
   double err;
   err=_FP(m,d);
   CalcError(m, d);
   _BP(m,d,a);
   return err;
}
```

```
/*******************************
        parameter.h
        Parameters of Data
*******************************/
#define   DATAFILE          "data"
#define   N_CLASS           6
#define   N_DATA            101

//Parameters of SOM
#define   STYLE             SQUARE          //HEX1              /* MapStyle */
#define   N_X               10          /* MapSize */
#define   N_Y               10
#define   N_MODULE          (N_X*N_Y)         /* Total Module Number */
#define   SIGMA_MAX         sqrt(N_X*N_X + N_Y*N_Y)
#define   SIGMA_MIN         2.0
#define   TAU               50
#define   N_LEARN_REPEAT        20
#define   ITERATION         10000

#define   Epsilon           0.001
#define   Gamma             0.8                 /* SafetyFactor */

/*        Parameters of MLPs          */
#define   N1        1       /*NumberofInputLayerUnits*/
#define   N2        5       /*NumberofHiddenLayerUnits*/
#define   N3        1       /*NumberofOutputLayerUnits*/

#define   RND_W_WIDTH 0.1
#define   RND_B_WIDTH  0.1
#define   Alpha             0.1
#define   Eta               0.1
```

```
/****************
        mnsom.h
*****************/
#define LABELLEN        100

typedef struct {
        char            label[LABELLEN];        /* Name of the class */
        DataVector      data[N_DATA];           /* Data Vectors */
        int             bmm;                    /* The Best Matching Module */
} ClassData;

typedef struct {
        double  x;              /*      The position of the module on the map       */
        double  y;
        MLP     mlp;            /*      MLP weight and state of the module          */
        double  err[N_CLASS];
        double  psi[N_CLASS];   /*      Learning Disributed Rate                    */
        int     winclass;
        int     Neighbor_N;
        int     NeighborList[8];
        double  NeighborDiff;
} MLP_Module;

#ifdef MNSOM_SOURCE
ClassData       class[N_CLASS];
MLP_Module      module[N_MODULE];
#else
extern ClassData    class[N_CLASS];
extern MLP_Module module[N_MODULE];
#endif

void MNSOM_Initialize_Square();
void MNSOM_Initialize_Hex1();
void MNSOM_Training(int t);
void MNSOM_CalcModuleDistance();
```

```
/*****************
        bp.h
*****************/
typedef struct {
        double  H[N2];          /* Hidden Layer */
        double  Y[N3];          /* Output Layer */
        double  W1[N1][N2];     /* Weights from Input to Hidden Layer */
        double  W2[N2][N3];     /* Weights from Hidden to Output Layer */
        double  B2[N2];         /* Bias of Hidden Units */
        double  B3[N3];         /* bias of Output Units */
        double  E2[N2];         /* Error of Hidden Units */
        double  E3[N3];         /* Error of Output Units */
        double  dW1[N1][N2];    /* Weight Change of W1 */
        double  dW2[N2][N3];    /* Weight Change of W2 */
        double  dB2[N2];        /* Weight Change of B2 */
        double  dB3[N3];        /* Weight Change of B3 */
} MLP;

typedef struct {
        double  X[N1];          /* Input vector */
        double  T[N3];          /* Desired output vector */
} DataVector;

void    MLP_Initialize(MLP *mlp);
void    MLP_Copy(MLP *m1, MLP *m2);
double  _FP(MLP *mlp, DataVector d);
void    _BP(MLP *mlp, DataVector d, double a);
double  _BP_Learning(MLP *mlp, DataVector d, double a);
```

謝辞

本論文製作にあたり，様々な方々にご指導，ご助言及びご協力をいただきここまで辿り着くことが出来た．この場で感謝の意を表したい．

はじめに，大学生活の 10 年間における約 6 年間の研究生活において公私ともに様々なお力添えを下さった九州工業大学生命体工学研究科脳情報専攻石井和男助教授に感謝申し上げたい．先生には，論文の書き方，プレゼンテーションの仕方，プログラムの書き方といった基礎的なことから，研究においての解決法やアプローチについて様々な議論を通してご指導いただいた．

脳情報専攻の古川徹生教授には，mnSOM3 ヶ月コースという他研究室での研究活動において，脳の未使用領域が開拓されるような感覚を覚える貴重な経験をさせていただいた．単なる道具として使用してきたニューラルネットワークであったが，基礎的な理論から指導していただき理解が深まった．それだけでなく，他方面からのアプローチを経験することで研究に対する視野が広がった．

九州工業大学情報工学部システム創生情報工学科の延山英沢教授，岡本卓教授，機械情報工学科の安部憲広教授及び林栄治助教授には，本論文の審査に当たって様々なご助言ご指導いただいたことに感謝したい．

研究生活および私生活は決して楽ではなかったが，こうして論文をまとめることが出来たのも石井研究室のメンバーのおかげである．脳情報専攻博士後期課程の Amir A.F.Nssiraei さんの研究に対する非常にエネルギッシュな姿勢には大きな刺激を受けた．佐藤雅紀君とは研究室での 6 年間，九州工業大学への入学が同期であるのでお互いに気の置けない仲間としてあることが出来た．石塚誠君の持つ行動力と腕力によって様々な実験やデモの準備などが円滑に行われた．松尾貴之君は石井研配属当初から研究室のマスコットとして常にあり，私を癒してくれた．大畑智海君の人を統べる眼差しは先輩である私を引っ張ってくれた．杉山公一君は異端の血として研究室の起爆剤となり研究室を盛り上げてくれた．園田隆君には本論文で提案したシステムの全体像のイメージを Fig1.6 という形で描いてくれた．王雪冰君は食事をご馳走していただき，食事中には研究の話から日中関係まで語り合う楽しい時間をすごすことが出来

た．彼らとは研究のライバルとして常に意識し切磋琢磨してきた．研究内容分野は異なるが，彼らの動向が私を刺激しここまでやって来るためのエネルギーを与えてくれた．

博士前期課程の江里口優君，神田敦司君，白石武尊君，武村泰範君，真田篤君，伴健志君，草壁亮君，望月隆吾君，西田裕也君，野畑慎伍君，徳賀健太郎君，横山健君，渡邊伸人君，情報工学部 4 年生の荒木聡君，森邦洋君達は研究室での私の弟的存在として，共に明朗快活な研究生活を送る上での不可欠な存在であった．石井研究室というひとつの家族のような団体の中で，公私共に後輩たちとともにあることで私も成長できた．皆に感謝したい．

本研究に対して，東京大学生産技術研究所浦環先生および東海大学海洋学部渡邉啓介先生に貴重なご意見を賜った．ここで謝意を表したい．

私が学問を志すきっかけとなった祖父の故久雄には生前に本論文をまとめることが出来なかったことを謝罪するとともに本論文をもって供養したい．

最後に，私の大学生活を経済的にも精神的にも支えてくれた，両親の尚生，信子に感謝したい．

参考文献

[1] けいはんな社会的知能発生学研究会, "知能の謎 認知発達ロボティクスの挑戦", 講談社, 2004.

[2] 辻三郎, "知能ロボット", 日本ロボット学会誌, Vol.1 No.1, pp.25-30, 1983.

[3] 有本卓, "ロボットの知能", 日本ロボット学会誌, Vol.21, No.2, pp131-134, 2003.

[4] 山崎哲生, "銅が危ない！－深海底鉱物資源開発の必要性と可能性－", KANRIN 日本船舶海洋工学会誌, 第 6 号, pp.64-70, 2006.

[5] 高井研, "地球最後のフロンティア地殻内に存在する微生物たち", 海と地球の情報誌 Blue Earth 通巻第 66 号, pp8-11, 2003.

[6] 映画「日本沈没」ウェブサイト, http://www.nc06.jp/special/index.html

[7] 中尾政之, "深海無人探査機「かいこう」行方不明", 失敗知識データベース-失敗百選, 科学技術振興機構.

[8] 浦環, 高川慎一, "海中ロボット総覧", 成山堂書店, 1994.

[9] 浦環, 高川慎一, "海中ロボット", 成山堂書店, 1997.

[10] 浦環, "海中に求められるロボット", 日本ロボット学会誌, Vol.22, No.6, pp692-696, 2004.

[11] T. Fujii, T. Ura, H. Chiba, Y. Nose and K. Aramaki, "Development of a versatile test-bed "Twin-Burger" toward realization of intelligent behaviors of autonomous underwater vehicles," OCEANS'93, Vol.1, pp.I186-I191, 1993.

[12] 近藤 逸人, 浦 環, "自律型海中ロボット"Tri-Dog 1 "の開発", ロボティクス・メカトロニクス講演会 2000 予稿集, 2A1-05-002, 2000.

[13] T. Maki, H. Kondo, T. Ura and T. Sakamaki, "Navigation of an Autonomous Underwater Vehicle for Photo Mosaicing of Shallow Vent Areas," OCEANS'07 Asia Pacific, CD-ROM, 2006.

[14] 浦環, 大坪新一郎, "航行型無索無人潜水艇に関する研究（その１　グライディング航行の研究）", 日本造船学会論文集, Vol.162, pp.117-124, 1987.

[15] 前田久明他, "無索無人潜水艇に働く線形流体力及び操縦応答に関する研究", 日本造船学会論文集, Vol.164, pp.211-220, 1988.

[16] 大楠丹, 柏木正, 小寺山亘, "Towed Vehicle の動力学に関する基礎的研究", 日本造船学会論文集, Vol.162, pp.99-109, 1987.

[17] J.M. Zurada, R.J. Marks II, C.J. Roinson, "Computational Intelligence, Imitating Life", IEEE Press, 1994.

[18] M. Palaniswami, Y. Attikiouzel, R.J. Marks II, David B. Fogel and T. Fukuda, "Computational Intelligence, A Dynamic System Perspective", IEEE Press, 1995.

[19] T. Ura, T. Fujii, Y. Nose and Y. Kuroda, "Self-Organizing Control System for Underwater Vehicles", Proc. of OCEANS'90, pp.76-81, 1990.

[20] 藤井輝夫, 浦 環, 黒田洋司, 能勢義昭, "自己生成型ニューラルネットコントローラシステムの開発と潜水機の運動制御への適用（その２：フォワードモデルの改良と無索テストベッドによる実験）", 日本造船学会論文集, Vol.169, pp.477-486, 1991.

[21] T. Fujii and T. Ura, "SONCS: Self-Organizing Neural-Net-Controller System for Autonomous Underwater Robots", Proc. of IEEE/INNS IJCNN'91, pp.1973-1982, 1991.

[22] K. Ishii, T. Fujii and T. Ura, "An on-line adaptation method in a neural network based control system for AUVs", IEEE Journal of Oceanic Engineering, Vol.20, No.3, pp.221-228, 1995.

[23] 浦環, 須藤拓, "自己訓練による海中ロボットの定高度航行", 日本造船学会論文集, Vol.171, pp.581-586, 1992.

[24] 浦環, 須藤拓, "自己訓練による海中ロボットの定高度航行（その２：フォワードモデルの改良）", 日本造船学会論文集, Vol.174, pp.917-924, 1993.

[25] J. Yuh, "Leaning Control for Underwater Robotic Vehicles", IEEE Control System Magazine, Vol.15, No.2, pp.39-46, 1994.

[26] J. Yuh, R. Lakshmi, "An Intelligent Control System for Remotely Operated Vehicles", IEEE J. of Oceanic Engineering, Vol.18, No.1, pp.55-62, 1993.

[27] J. S. Wang, C. S. G. Lee, J. Yuh, "An On-line Self-Organizing Neuro-Fuzzy Control for Autonomous Underwater Vehicles", Proc. of ICRA'02, pp.1095-1100, 2002.

[28] J. Guo, F. C. Chiu, C. C. Wang, "Adaptive Control of an Autonomous Underwater Vehicle Testbed Using Neural Networks", Proc. of Oceans'95, pp.1033 - 1039, 1995.

[29] J. H. Li, P. M. Lee, S. J. Lee, "Neural Net Based Nonlinear Adaptive Control for Autonomous Underwater Vehicles", Proc. of ICRA'02, pp.1075 - 1080, 2002.

[30] P. J. Kodogiannis, G. Lisboa, J. Lucas, "Neural Network Modeling and Control for Underwater Vehicles", Artificial Intelligence in Engineering, Vol.1, pp.203-212, 1996.

[31] R.S. Burns, "The Use of Artificial Neural Networks for the Intelligent Optimal Control of Surface Ships", IEEE Journal of Oceanic Engineering, Vol.20, No.1, pp.65-72, 1995.

[32] 小川原陽一, 岩本才次, 吉村学, "風外乱補償機能を付加した船舶操縦運動の学習型フィードフォワード制御方式の基礎的検討", 日本造船学会誌論文集, Vol.178, pp.321-328, 1995.

[33] I. Yamamoto, Y. Terada, T. Nagamatu and Y. Imaizumi, "Propulsion System with Flexible/Rigid Oscillating Fin", IEEE Journal of Oceanic Engineering, Vol.20, No.1, pp.23-30, 1995.

[34] T. K. Y. Lo, H. Leung, J. Litva, "Artificial Neural Network for AOA Estimation in a Multipath Environment Over the Sea", IEEE. Journal of Oceanic Engineering, Vol.19, No.4, pp.555-562, 1994.

[35] J. C. Park, R. M. Kennedy, "Remote Sensing of Ocean Sound Speed Profiles by a Perceptron Neural Network", IEEE Journal of Oceanic Engineering, Vol.21, No.2, pp.216-224, 1996.

[36] J. Healey, "A Neural Network Approach to Failure Diagnostics for Underwater Vehicles", Proc. of AUV'92, pp.131-134, 1992.

[37] M. Takai, T. Ura, "Development of a System to Diagnose the Autonomous Underwater Vehicle", International Journal of Systems Science on Unmanned Underwater Vehicle Control, Vol.30, No.9, pp.981-988, 1999.

[38] T. Kohonen, "Self-organized formation of topologically correct feature maps", Biological cybernetics, Vol.43, pp.59-69, 1982.

[39] S.S. Haykin, (1999), "Neural Networks -A Comprehensive Foundation- 2nd Edition", Prentice Hall.

[40] T. Yamakawa and K. Horio, "Self-Organizing Relationship (SOR) Network", IEICE Trans., E82-A, pp.1674-1678, 1999.

[41] K. Ishii, T. Fujii, T. Ura, "An On-line Adaptation Method in a Neural Network Based Control System for AUV's", IEEE Journal of Oceanic Engineering, Vol.20, No.3, pp.221-227, 1995.

[42] 石井和男, 浦環, 藤井輝夫, "ニューラルネットワークによる潜水艇の運動の同定（その２：学習過程の改良とコントローラ調整への適用）", 日本造船学会論文集, Vol.177, pp.429-435, 1995.

[43] 石井和男, 藤井輝夫, 浦環, 能勢義昭, "ニューラルネットによる潜水艇の運動の同定（その３：学習による外乱への適応）", 日本造船学会論文集, Vol.182, pp.469-479, 1997.

[44] K. Ishii and T. Ura, "An adaptive neural-net controller system for an underwater vehicle", Journal of IFAC Control Engineering Practice, Vol.8, pp.177-184, 2000.

[45] M. Haruno, D. M. Wolpert and M. Kawato, "Mosaic: Module selection and identification for control", Neural Computation, Vol.13, No.10, pp.2201-2220, 2001.

[46] K. Doya, K. Samejima, K. Katagiri and M. Kawato, "Multiple Model-based Reinforcement Learning", Neural Computation, Vol.14, pp.1347-1369, 2002.

[47] 川人光男, 銅谷賢治, 春野雅彦, "ヒト知性の計算神経科学 <第 4 回>多重順逆対モデル(モザイク) -その情報処理と可能性", 科学, Vol.70, No.11, pp.1099-1017, 2000.

[48] 鮫島和行, 銅谷賢治, 川人光男, "強化学習 MOSAIC: 予測性によるシンボル化と見まね学習", 日本ロボット学会誌 19 巻 5 号, pp.551-556, 2001.

[49] K. Tokunaga, T. Furukawa and S. Yasui, "Modular Network SOM: Extension of SOM to the realm of function space", WSOM'03, pp.173-178, 2003.

[50] T. Furukawa, T. Tokunaga, S. Kaneko, K. Kimotsuki and S. Yasui, "Generalized Self-Organizing Maps (mnSOM) for Dealing with Dynamical Systems," International Symposium on Nonliner Theory and its Applications, pp.231-234, 2004.

[51] T. Furukawa, K. Tokunaga, K. Moroshita and S. Yasui, "Modular Network SOM (mnSOM): From Vector Space to Function Space", International Joint Conference on Neural Networks, 2005.

[52] K. Tokunaga, T. Furukawa, "Nonlinear ASSOM Constituted of Autoassociative Neural Modules", 5th Workshop on Self-Organizing Maps, 2005.

[53] T. Minatohara, T. Furukawa, "Self-Organizing Adaptive Controllers: Application to the Inverted Pendulum", 5th Workshop on Self-Organizing Maps, pp.44-48, 2005.

[54] 西田周平, 矢野孝三, 石井和男, 浦環, "自己組織化マップを用いた障害物回避手法とその海中ロボットへの適用", ロボティクス・メカトロニクス講演会 2002 予稿集, 1A1-A08, 2002.

[55] 西田周平, 矢野孝三, 石井和男, 浦環, "自己組織化マップを用いた障害物回避手法とその海中ロボットへの適用", 第 20 回日本ロボット学会学術講演会予稿集, 1B23, 2002.

[56] S. Nishida, K. Ishii, K. Watanabe and T. Ura, "A Collision Avoidance System based on Self-Organizing Map and its application to an Underwater Vehicle," Proc. of ICRACV2002, pp.602-607, 2002.

[57] S. Nishida, K. Ishii and T. Ura, "A Navigation System for Underwater Vehicle using Self-Organizing Map (An Adaptation Method of Map with Trajectory Evaluation)," Proc. of GCUV2003, pp.101-106, 2003.

[58] 西田周平, 石井和男, "自己組織化マップを用いた水中ロボットのナビゲーションシステム", ロボティクス・メカトロニクス講演会 2003 予稿集, 2P1-2F-A5(1)-(2), 2003.

[59] 西田周平, 石井和男, 浦環, "自己組織化マップを用いた水中ロボットのための適応学習", SICE システムインテグレーション部門講演会(SI2003)予稿集, pp738-739, 2003.

[60] K. Ishii, S. Nishida and T. Ura, "An Adaptive Learning Method for SOM Based Navigation System and Its Application to an Underwater Vehicle", Proc. of WSOM2003, pp.287-292, 2003.

[61] S. Nishida, K. Ishii and T. Ura, "Adaptive Learning to Environment using Self-Organizing Map and its Application for Underwater Vehicles", Proc. of UT2004, pp223-228, 2004.

[62] K. Ishii, S. Nishida and T. Ura, "A Self-Organizing Map Based Navigation System for an Underwater Robot", Proc. of ICRA2004, pp.4466-4471, 2004.

[63] S. Nishida, K. Ishii and T. Furukawa, "An adaptive controller system using mnSOM", Proc. of Brain-inspired Information Technology, pp.181-184, 2005.

[64] 西田周平, 石井和男, 古川徹生, "mnSOM を用いたニューラルネットコントロールシステムの開発", 平成 17 年日本船舶海洋工学会秋季講演会予稿集, pp13-14, 2005.

[65] 西田周平, 石井和男, 古川徹生, "mnSOM を用いた水中ロボットの自己組織的行動決定システムの開発", SICE システムインテグレーション部門講演会(SI2005)予稿集, pp655-656, 2005.

[66] 西田周平, 石井和男, 古川徹生, "mnSOM を用いた適応制御システムの水中ロボットへの適用", ロボティクスシンポジア予稿集, pp264-269, 2006.

[67] 西田周平, 石井和男, 古川徹生, "水中ロボットにおける自己組織的行動獲得システム -第 1 報:自己組織化マップを用いた運動制御システムの提案-", 日本船舶海洋工学会論文集 第 3 号, pp.205-213, 2006.

[68] 西田周平, 石井和男, 古川徹生, "モジュラーネットワーク型自己組織化マップを用いた水中ロボットのシステム同定", ロボティクス・メカトロニクス講演会 2006 予稿集, 1P1-E23, 2006.

[69] S. Nishida, K. Ishii, T. Furukawa, "An Adaptive Neural Network Control System using mnSOM", Proc. of Oceans'06 Asia Pacific, FRI-04-4, 2006.

[70] 西田周平, 石井和男, 古川徹生, "モジュラーネットワーク型自己組織化マップを用いた水中ロボットの運動制御システム", 第 24 回日本ロボット学会学術講演会予稿集, 3D37, 2006.

[71] S. Nishida, K. Ishii, T. Furukawa, "Adaptability of mnSOM Based Control System to Changing Dynamic Property", Proc. of Brain-Inspired Information Technology, 2006.

[72] S. Nishida, K. Ishii and T. Furukawa, "Development of a Control System for Autonomous Underwater Vehicles using mnSOM", Proc. of SCIS&ISIS 2006, TH-E4 100305, 2006.

[73] S. Nishida, K. Ishii and T. Furukawa, "An Online Adaptation Control System using mnSOM", Proc. of ICONIP 2006, pp.935-942, 2006.