〔非公開〕

ＴＲ－Ｃ－００８４

利用者インタフェースのための手振り認識と理解について

大西　剛　Takeshi ONISHI
竹村治雄　Haruo TAKEMURA
岸野文郎　Fumio KISHINO

１９９３　２．８

ＡＴＲ通信システム研究所

利用者インタフェースのための手振り認識と理解について

大西 剛　　竹村 治雄　　岸野 文郎
ATR 通信システム研究所

概要

データグローブの開発などによりここ数年この分野の研究がさらに盛んになってきた．手振りを用いたユーザインタフェースには現実世界での感覚を忠実に再現したインタフェースを目指す方向と，容易に利用できることに重点を置いた方向が考えられる．我々は，後者の観点から右手による手振りを用いてディスプレイ上に表示される画像を操作する模擬実験を行ない，この条件下でのユーザインタフェースとしての手振り使用の有効性を検証した．また，この実験中に現れた手振り分析することにより，手振り認識に必要なデータとして手指の先端座標データを用いることを提案した．この仮定の下でインタラクティブなユーザインタフェースを構築するために，手形状を識別する方法，手振りの動きを評価する方法を検討した．これらの方法を用いてインタラクティブに CG オブジェクトを操作するユーザインタフェースを試作し，手振りの利用の可能性を確認した．

目次

1 はじめに **3**

2 手振り利用の模擬実験 **3**

2.1 実験条件 4

2.2 結果 5

2.3 考察 6

3 手振り認識に必要なデータ **6**

4 手形状の表現 **7**

5 静的な手形状の識別 **8**

6 試作システム **10**

6.1 概観 11

6.2 ニューラルネットを用いた手振りの動き評価 12

6.3 試作システムの機能 14

6.4 評価 16

7 画像データから手形状認識 **16**

7.1 計測環境 18

7.2 手形状識別 18

7.2.1 識別アルゴリズム 18

7.2.2 識別結果 19

7.3 計測データに欠落がある場合の対応 19

7.4 動きを含む手振りへの対応 20

8 おわりに **21**

A 試作システム構成 **24**

B プログラムソースリスト **25**

1 はじめに

各種分野でのコンピュータ利用技術の発展，一般化が進むにつれて，より使い易いユーザインタフェース構築の必要性が高まってきている．身振り・手振りは，人と人とのコミュニケーションのさまざまな場面で用いられている．我々は，身振り・手振りを利用してより豊かなマンマシンインタフェースを構築することを目指し，現在は手振りのみに限定して研究を行なっている．手振りをマンマシンインタフェースとして利用することの利点として，直観的なインタフェースとなることが期待できる，手振りにより同時に複数のパラメータをコントロールすることが可能である，等があげられる．

我々は身振り・手振りを用いて意図を伝えあっているが，実際の日常生活では身振りのみでコミュニケーションをする場合は稀である．したがって実際にユーザインタフェースを構築するためには，身振りの長所・短所を把握し，有効性を正しくとらえる必要がある． Hauptmann らは，身振りのみ，音声のみ，身振りと音声の併用で画像オブジェクトの操作を行なう模擬実験を行ない，統計的な結果およびそれぞれの場合の被験者の印象について報告している [1]．この実験では，音声がかなり重要な役割を持つことが報告されている．これは操作の対象が複雑な場合は，身振りのみで全ての操作をすることが困難となることを示唆している．

これまで Bolt らの “Put That There”， Krueger の VIDEOSPACE など手振りを用いたインタフェースが研究されてきた [2][3]．これらの研究では，限られた文脈の中で限られた手振りのみが意味を持つものとして用いられてきた．身振りは前後の文脈に依存する度合が強く，多義的な解釈が可能であると考えられるが，多くの文脈中で共通して自然な形で使えるマンマシンインタフェースとして利用するには，手振りの認識を行なう手法が確立しておらず，実際のシステムを構築する上で課題となっている [4]．

データグローブの開発などによりここ数年この分野の研究がさらに盛んになってきた [5][6]．手振りを用いたユーザインタフェースには現実世界での感覚を忠実に再現したインタフェースを目指す方向と，容易に利用できることに重点を置いた方向が考えられる．本報告では，我々が後者の観点から進めてきた手振り利用の研究について報告する．

2 節では，右手による手振りを用いてディスプレイ上に表示される画像を操作する模擬実験を行ない，この条件下でのユーザインタフェースとしての手振り使用の有効性を検証したこと， 3 節， 4 節では，この実験中に現れた手振り分析することにより，手振り認識に必要なデータとして手指の先端座標データを用いることについてそれぞれ述べる [7][8]．さらに，この仮定の下でインタラクティブなユーザインタフェースを構築するために， 5 節では手形状を識別する方法 [9][10]， 6 節では手振りの動きを評価する方法を述べる． 7 節ではこれらの方法を用いてインタラクティブに CG オブジェクトを操作するユーザインタフェースを試作し，手振りの利用の可能性を確認したことについて， 8 節では手振りデータの画像処理による獲得方法の検討について述べる [11][12][13][14]．

2 手振り利用の模擬実験

自然な手振りを入力の一手段として利用するシステムを構築するために以下の実験を行なった。この実験は、システムに組み込むことが可能な自然な手振りが存在するか、あるとすればどのような動作かを調べることを目的として行なった．

2.1 実験条件

コンピュータディスプレイ上の画像オブジェクトを操作するのにユーザはどんな手振りを使うかを調べるために以下のような条件で実験を行なった (図 1).

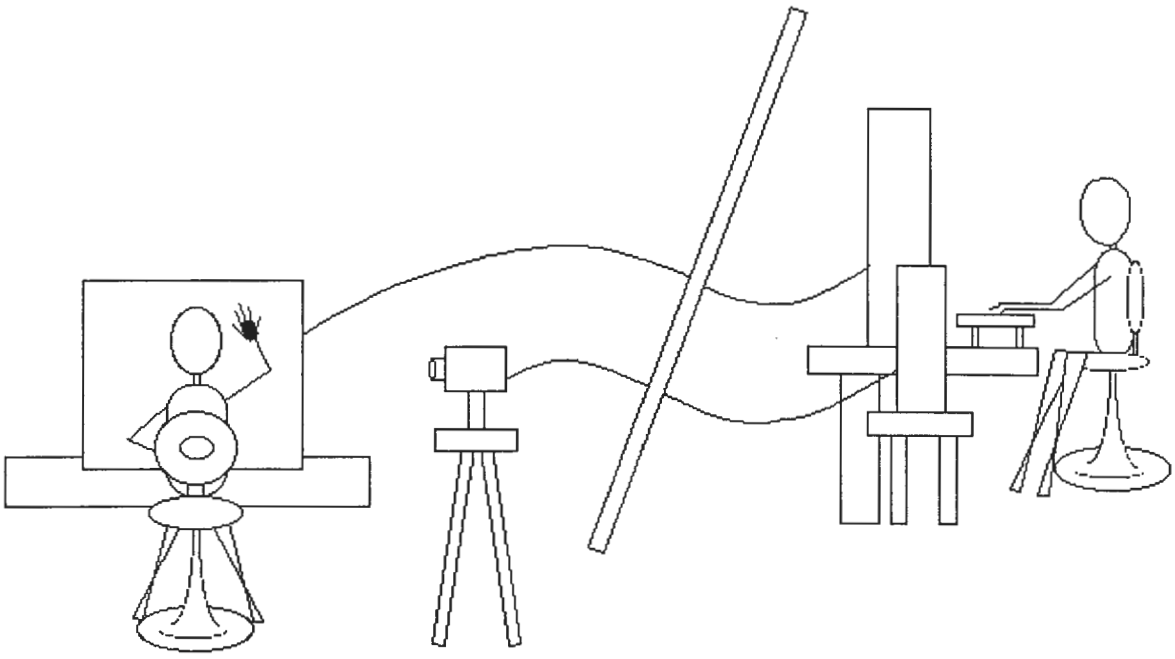


図 1 模擬実験の環境

以下に実験条件を列挙する.

- 被験者は 19 インチディスプレイの前に座り，ディスプレイ上の画像オブジェクトを操作するタスクを与えられる.
- 2 次元表示のディスプレイ上の対象オブジェクトを色の異なる目標のオブジェクトに重ねることがタスクである. このタスクには図 2 で示すように移動，回転，拡大・縮小の 3 種類がある.
- 各被験者は 3 種類のそれぞれのタスクを 5 回ずつ，合計 15 回のタスクを繰り返して行なう。
- 各オブジェクトは正方形のワイヤモデルとする.
- 被験者には，右手及びその指の形・動きのみが意味を持つことをあらかじめ伝えておくが，どのような手振りが有効であるかは教えず，被験者は思いのままに動作を行なう.
- 被験者の動作は，被験者撮影用のカメラで撮影しＶＴＲに記録するとともに，実験者が被験者の動作を判断するために用いる.
- 手の位置データ，形状データを獲得するために被験者はデータグローブを装着する.
- 実験者はカメラを通した映像を見て，被験者の動作の意図を理解できれば，マウス，ダイアルボックスを使い，オブジェクトを操作する. この操作によるオブジェクトの変化は記録される.

- カメラの位置は被験者の右脇とし，ディスプレイを撮らないようにした．これは，目標オブジェクトを知ることが実験者に影響を与えるのを避けるためである．

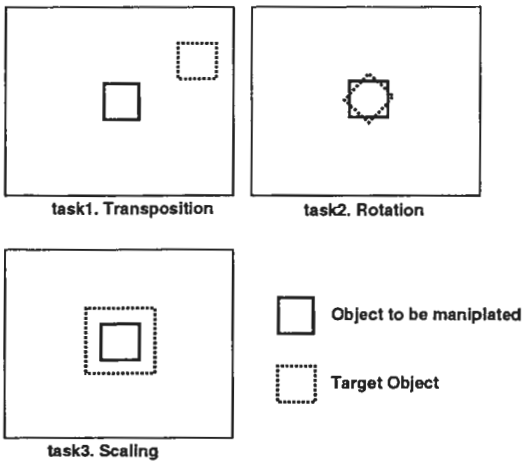


図 2 模擬実験のタスク

2.2 結果

8 人の被験者に対して上記環境で実験した結果を述べる．（被験者は年齢が 20 歳代前半から 30 歳代前半の男性 3 人女性 5 人であった．）

移動を意図したと認められる動作には，大きく分けて 2 種類あった．一つは，人差指で指差した方向に画像オブジェクトが移動するように指示するものであり，他方は手の移動にしたがってオブジェクトが移動することを期待したものである．前者の場合，指し示す方向は上下左右に限られ，「被験者の意図した場所にオブジェクトが移動するまで，手の形を変えずにおくという手振り (T1)[1]」が用いられた．後者の場合，「オブジェクトの移動位置を逐次指で指し示す動作 (T2)」と，「手全体の動きに追随してオブジェクトを移動させようとする動作 (T3)」とが確認された．

回転では，オブジェクトに対して手掌部分を向け，軽く指を伸ばした状態で，手全体を内側または外側に回転させる動作が多く見られた．このとき，「手を回転させる角度がオブジェクトの回転角度に等しくなることを期待する動作 (R1)」と，「手を回転させることが回転の方向と開始とを意味するだけで回転角度に関しては含意しない動作 (R2)」に分けられた．上記以外の動作として，「人差し指のみを伸ばした状態で回転の中心の回りを指差しながら回す動作」が見られた．この動作では，回転の方向のみを含意することが多く，わずかな角度の回転を意図する場合にのみ，人差指を回転方向に小さく移動させる動作が見られた．上に述べた 3 つの動作の内，2 番目，3 番目のものは，その動作を繰り返して行なうことが認められた．

拡大・縮小では，オブジェクトに対して手掌部分を向け，拡大ならば指を曲げた状態から伸ばした状態に，縮小ならば指を伸ばした状態から曲げた状態に変化させる動作が，多くの被験者に共通に見られた．この動作には，「指の広げ具合がオブジェクトの大きさを含意する動作 (S1)」と，「指を伸ばしたり曲げたりすることがオブジェクトの拡大・縮小の開始を意図するだけの動作 (S2)」の 2 種類があった．後者の場合，同じ動作を複数回繰り返すことが多かった．その他の動作では，「手をディスプレイに近づけたり，遠ざけたりする動作 (S3)」があった．またどのような動作を行なってよいのか戸惑う被験者も見られた．

[1]表 1 での記号を示す．

2.3 考察

各タスクに対して複数の被験者が共通の手振りを用いることがある程度確かめられた．3つのタスクに共通して手振りは2種類に分けられると考えられる．すなわち，その手振りが“バリュエータ”または“ロケータ”として量的な情報を含むものと，手振りが“ボタン”として働き以後変化が継続することを期待するか，一時的に変化することを仮定するものである [5]．どちらの場合においても，視覚的フィードバックが重要であり，ユーザはオブジェクトの反応を見ながら，自分の手の動きの解釈され方を判断し操作することが確かめられた．

また拡大・縮小のタスクでは，被験者に戸惑いが見られた．これは，現実の世界ではこのような操作をすることが少ないからであると考えられる．このことからも，手振りの直観性が検証されるが，一方その限界も示されている．

3 手振り認識に必要なデータ

これまでに，身振りを記述する方法はいくつか提案されている [16][17]．しかし，これらの方法は人が人間の動きを観察して，その動きからコードを生成したり，あらかじめ人により記述されたコードから動きを再現するものが中心で，実際の動きを人の判断を介することなくに抽出するには十分でない．また，手は体の他の部分よりもより多くの情報を担っていると考えられること，手振りとしてではなく実際にものを触ったりつかんだりするのにも使われることなど，これまでの体内部の形状をのみ考慮した身振りを記述する方法では対応できない．

手振りの認識を行なうのに必要なデータは，ある時点において視覚的に相違が認められる手の形を区別できる情報を少なくとも含んでいる必要がある．その要素として具体的には，各指の各関節がどの程度に曲がっているか，各指の他の指との接触・交差の有無等が挙げられる．

さらに，対象物への働きかけとして手振りが用いられる場合には，環境内における手と対象物の位置関係が重要な情報を含んでいる．さらに位置関係は，手全体の位置のみが重要である場合とある指の位置が重要な場合に分けられる．

手振りをコマンドとして用いる場合には，これら基礎的なデータから手振りをコード化して扱った方が便利な場合がある．例えば，前記の模擬実験の結果を示した表1で，手振りT1はオブジェクトを移動させるときに被験者の望む位置にオブジェクトが移動するまで上下左右のいずれかの方向に人差し指で指差す手振りを示している．人差し指で指差す手形状をコード化し，あるシンボルと考えれば，この手振りをコマンドとして利用する時には，手形状がこのシンボルに変換されることと手と指の方向が分かればよい．また，手振りS3では，手を開く動作がオブジェクトの拡大をうながすスイッチの意味を持った動作として用いられている．手を閉じている手形状，手を開いている手形状，その中間の手形状をそれぞれシンボルと考えれば，上記の動作はシンボル時系列としてあつかうことができる．

すなわち，ある環境内でインタラクティブな入力手段としての手振りを認識するのに必要なデータは，手振りの使用環境における手の位置，手の形，およびこれらの時間的な変化を表せなければならない．さらに手形状を上記の意味で容易にシンボルに変換できることが望ましい．

Task	Type	Gesture	Persons	information needed for recognition			
				Values			Symbol
				Hand		Finger	Symbol Stream
translation	switch	T1	1		○	○	○
	locator	T2	2	○	○	○	
		T3	5	○			
rotation	valuator	R1	6		○	○	
	switch	R2	1		○	○	○
scaling	valuator	S1	3			○	
		S3	2	○			
	switch	S2	3			○	○

表 1 模擬実験結果

4 手形状の表現

前節の要求を満たす手の形状表現手法として，手の形状を手の甲に固定したある点からの各指先端位置によって表現する方法を提案する．

人間の手を次のように骨格を中心としたモデルとして考える．骨を直線で表現し，関節をそれらの接点と考え，接点においてのみ回転運動が行なえる仮定とする．また，中手の部分は一平面上の広がりとし，この部分は変化しないと仮定する．

このモデルを用いることによって，ある時点における手の形状は，各指の関節の曲げ角度で表現できる．また手の形状は各指の関節の相対座標を用いて表すこともできる．すなわち，手の各部の大きさと各間接曲げ角を併せた情報と，手の各指の関節の相対座標がもつ情報は，ある時点の手の形状を表すことにおいて等価である．

指先端位置による手形状の表現仮想的な骨格モデルに，さらに人の手の各指の各関節がすべて独立に動かせるわけではないという制約を仮定する．この仮定を用いることにより，以下のような 3 次元直交座標系を用いて手の形状を表現することができる．座標系は，手の甲のある点を原点とし，そこから中指を自然に（たとえば，机上で）伸ばした方向を x 軸方向とする．仮想的な手の骨格モデルとして中手の部分は一平面上の広がりとして仮定したので，その平面上に x 軸と直交する y 軸を親指と逆向きを正として定めることができる．z 軸はこの平面と直交する手の背から腹への向きをに定める．手の形状はこの座標系における各指先端座標で式 (1) の $G(t)$ のように表すことができる．$G(t)$ と手の位置 $P(t)$ と方向 $A(t)$ を用いて，手振りからの基礎的なデータとすることができる．

$$G(t) = \begin{pmatrix} x_1(t), & x_2(t), & x_3(t), & x_4(t), & x_5(t) \\ y_1(t), & y_2(t), & y_3(t), & y_4(t), & y_5(t) \\ z_1(t), & z_2(t), & z_3(t), & z_4(t), & z_5(t) \end{pmatrix}$$

$$P(t) = \begin{pmatrix} x_0(t) \\ y_0(t) \\ z_0(t) \end{pmatrix}, \quad A(t) = \begin{pmatrix} \psi(t) \\ \theta(t) \\ \phi(t) \end{pmatrix} \tag{1}$$

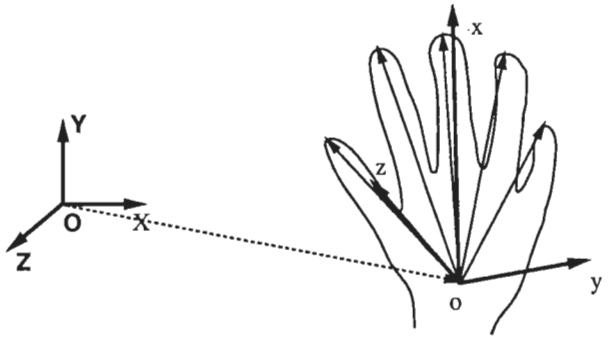


図 3 手振りデータ取得のための座標系

本手法の特徴として，第一に指相互の関係が位置関係を含む手形状を簡単に表現することができることが挙げられる．これは角度を用いて手形状を表す場合には困難であった．

第二に手以外の物体との位置関係が陽に記述できる．手をシンボルとしてではなく物理的な道具として用いるときにはこの情報が必要である．コード化を行なったあとではデータからこの情報が失われてしまう．

第三に実際にこの手法を直接測定方法として使えることがある．このとき角度を用いる場合に比して，測定部位が少ない．また，指先端という特徴的な点であるので，画像処理による非接触装置で有効なインタフェースを実現できる可能性を持っている．

5 静的な手形状の識別

動きを伴わない手の形状を識別する基礎検討として，10 種類の手の形状を判別する実験を行なった．

親指と人指し指の各先端の基準点からの相対座標を測定する．手の基準点および各指先端位置を検出するために 3 次元磁気センサを用いている．これらの磁気センサから得られるデータから環境内での位置 P(t)，3 次元空間における向き A(t)，および 2 本の指のみに限定した手の形状 G(t) を知ることができる．今回の実験では，これらのデータの中で各時点における G(t) の 6 次元データのみを対象とした．1 人の被験者が 10 種類の定められた手の形状をそれぞれ 50 回繰り返し行なった結果を用いる．

実験では，10 種類の手の形状それぞれに対して得た 50 個のサンプルデータに対して識別を行なった．識別方法としては，部分空間法を用いた．部分空間法は各類の特徴を比較的少ない次元で効率良く表現でき，識別には入力ベクトルとの内積のみを計算すれば良いという点で，高速に時系列データを処理することが必要な手振り認識に有効であると考えられる．部分空間法による手の形状の識別を行なうことにより，さまざまな手の形状をある程度の群に分けることが可能となり，知識を必要とするより細かな識別は，その群内においてのみ行なえば良いことになる．

部分空間法では，手の形状に対応したデータを各類として，その類の既知の標本パターン群の情報から最小平均二乗識別誤差の規準を用いて各類標本パターン群を近似する基底直交ベクトルを設計する．識別段階では，未知の入力パターンが識別系に入力され適当な類すなわち手の形状に識別されるか，どの類にも識別されずに棄却されることになる．以下の実験では，サンプルパターンの内 20 個を部分空間を設計するための標本パターンとし，残りの 30 個を識別されるべき未知の入力パターンとして用いた．

(1) CLAFIC 部分空間法による識別

CLAFIC 部分空間法 (Class-Featuring Information Compression) による識別結果を表 2 に示す．手の形状に関する各類 $\omega^{(i)}$, $(i = 0, \ldots, 10)$ の標本ベクトルの相関行列を $Q^{(i)}$ とすれば，各手形状に対応した部分空間は，以下の式で求められる $Q^{(i)}$ の固有ベクトル $u_j^{(i)}$ で張られることが知られている．

$$Q^{(i)} u_j^{(i)} = \lambda_j^{(i)} u_j^{(i)}, \; j = 1, \ldots, p^{(i)} \tag{2}$$

ここで部分空間の次元 $p^{(i)}$ の選択は忠実度 κ によって決定される．実験では $\kappa = 0.95$ とした．すなわち，各固有ベクトルに対応した固有値 $\lambda_j^{(i)}$ から

$$\frac{\sum_{j=1}^{p^{(i)}} \lambda_j^{(i)}}{\sum_{j=1}^{6} \lambda_j^{(i)}} \leq \kappa < \frac{\sum_{j=1}^{p^{(i)}+1} \lambda_j^{(i)}}{\sum_{j=1}^{6} \lambda_j^{(i)}} \tag{3}$$

を $i = 0, \ldots, 10$ のすべての i で満たすように定めた．識別関数は，

$$\delta^{(i)}(g) = \sum_{j=1}^{p^{(i)}} \left(g^T u_j^{(i)} \right)^2 \tag{4}$$

ただし，g は入力される 6 次元の手形状ベクトルである．
という部分空間の基底ベクトルと入力ベクトルの内積和であり，各類の基底ベクトルとの識別関数を計算し，もっとも大きな値をとる類に類別する．10 種の手形状に適用した結果，個々の部分空間の次元は 1 から 2 になった．

(2) 複合類似度法 (MSM) による識別

複合類似度 (MSM; Multi Similarity Method) を識別規準に用いた場合の結果を示す．複合類似度法は，入力ベクトルが g，類に対応した部分空間が $L^{(i)}$ のとき識別関数が

$$\delta^2(L^{(i)}, g) = \sum_{j=1}^{(i)} \frac{\lambda_j^{(i)}}{\lambda_1^{(i)}} \frac{\left(g^T u_j^{(i)} \right)^2}{g^T g} \tag{5}$$

となり，基底ベクトル $u_j^{(i)}$ への g の射影が対応する固有値の大きさを反映する点が CLAFIC 法とは異なっている．しかし，今回用いたデータでは，CLAFIC 法と複合類似度を用いた結果には大きな差は認められなかった．これは手形状に対応した各類の相関行列の最大固有値だけが他の固有値に比べて大きな値をとるためである．

(3) 平均学習部分空間法 (ALSM; Averaged Learning Subspace Method) による識別

上の 2 種類の識別方式は，各類が他の類と独立に表現されるため異なる複数の手の形状に対してうまく識別できないことがある．この問題を補う方法として，部分空間法に学習を導入し，訓練集合に対する誤識別率を最小にするように識別境界を調整する学習部分空間法がある．そのアルゴリズムは，

1. 訓練ベクトル x から初期行列

$$S_0^{(i)} = \sigma_{x \in \omega^{(i)}} x x_T, i = 1, \ldots, K \tag{6}$$

を求め，CLAFIC 法と同様に $S_0^{(I)}$ の $p^{(i)}$ 個の主要固有値に対応するベクトルを部分空間 $L_0^{(i)}$ の基底とする．（$k = 1$ とする．）

2. 部分空間 $L_{k-1}^{(1)}, \ldots, L_{k-1}^{(K)}$ と決定規則を用いて，すべての訓練ベクトルの識別を行なう．$\omega^{(i)}$ に属していて，$\omega^{(j)}$ に識別された訓練ベクトルに対して以下のように，条件付相関行列の推定値を

$$S_k^{(i,j)} = \sigma^{x \in \omega^{(i)} x \to \omega^{(j)}} x x^T \tag{7}$$

によって計算する．

3.

$$S_k^{(i)} = S_{k-1}^{(i)} + \sigma_{j \neq i} \alpha^{(i,j)} S_k^{(i,j)} - \sigma_{(j \neq i)} \beta^{(i,j)} S_k^{(j,i)} \tag{8}$$

を計算する．係数 $\alpha^{(i,j)}, \beta^{(i,j)}$ は適当に決める．たとえばある定数にする．

4. $p^{(i)}$ 個の主要固有値に対応する $S_k^{(i)}$ の固有ベクトルを計算して，$L_k^{(i)}$ の新しい基底とする．

5. k を 1 つ増して，段階 2 に行く．

である．識別実験では $\alpha^{(i,j)}, \beta^{(i,j)}$ は 0.1 に固定して行なった．

表 2 に CLAFIC 法と ALSM 法で識別実験を行なった結果を示す（MSM 法は CLAFIC 法の結果と大きな差がなかった）．3 種類の識別法の中で ALSM 法がもっとも実験結果がよかったので，ALSM 法を採用することとした．

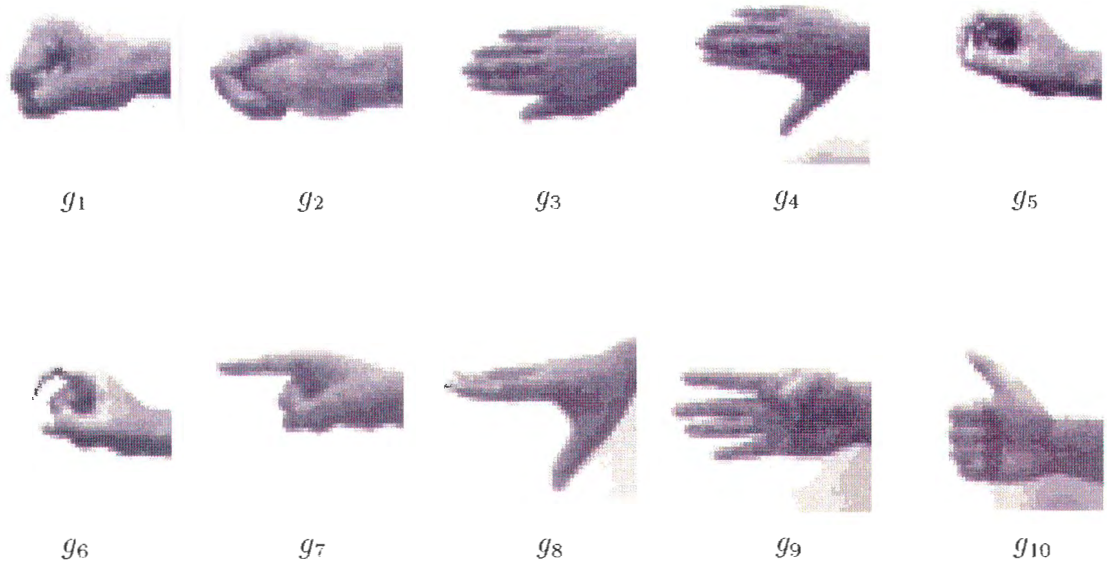


図 4 実験で用いた手形状パターン

6 試作システム

これまでに示した方法を用いて，図 7 のような手振りによる CG オブジェクト操作システムの試作について報告する．

Class	CLAFIC		ALSM	
	mis-classified	rejected	mis-classified	rejected
g_1	0	0	0	0
g_2	2	0	0	0
g_3	42	0	0	0
g_4	1	0	0	0
g_5	3	0	1	0
g_6	1	0	1	0
g_7	3	0	0	0
g_8	5	3	1	0
g_9	1	1	0	2
g_{10}	0	0	0	0

表 2 部分空間法での手形状識別実験結果

図 4 のパターンを部分空間法で識別した結果. MSM 法の結果は CLAFIC 法と大きな差はなかったので省いている. それぞれ, 50 個の標本パターン (20 個の訓練パターン, 30 個の検査パターン) における誤識別数, 棄却数を示す.

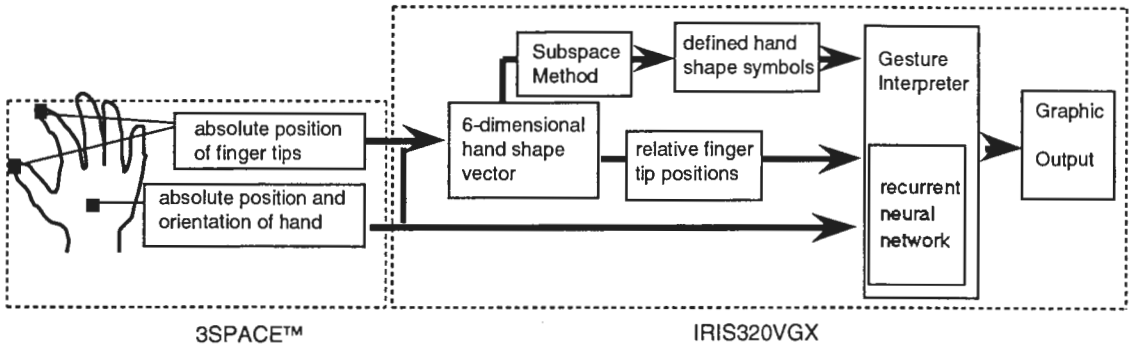


図 7 CG オブジェクト操作システム概観

6.1 概観

本システムでは CG により生成されたオブジェクトを手振りを用いて操作する. 親指・人差指の先端および基準位置として手背面に装着した 3 個の 3 次元磁気センサから得られる手の位置・方向, 指の相対位置座標を時系列データとして扱う. 各時点における指先端相対位置データをあらかじめ定めておいた形状に識別し, これによって画像オブジェクト操作の開始・終了を定める. この間の手全体の動きおよび指の動きから, 操作によりオブジェクトが受ける適切な変化量を決定し出力する. 指先位置データをあらかじめ定めた形状に変換する部分では, すでに述べた平均学習部分空間法により識別を行なっている. 動きから変化量を算出する部分は, リカレントニューラルネットワークを用いて時系列データを処理している. このニューラルネットワークの説明を次項で説明する.

6.2 ニューラルネットを用いた手振りの動き評価

手振りの動きをバリュエータとして用いることがあることが，上記の模擬実験において確認された．手振りから得られる時系列データを処理するためにリカレントニューラルネットワークを用いた [18].

入力は手の形状データ，手の位置・方向の変化量，出力は CG オブジェクトに与えるべき変化量である．

ネットワークの構造は入力層，中間層，出力層，コンテクスト層からなり，入力層の各ユニットには逐次，手の形状データ，位置・方向の変化量が入力される．コンテクスト層は 1 単位時間前の中間層の出力値を保存し，メモリの役割をはたす．中間層，コンテクスト層の各ユニットは出力層への結合を持っている．

学習過程においてそれぞれのユニット間の結合の強さは，以下のように誤差逆伝搬学習により決定される．図 5 のような多層ネットワークを仮定し，層内結合，フィードバック結合はないものとする．p 番目のパターンに対してユニット j は，前層のすべてのユニットからの出力をシナプス荷重によって重み付けした式 (9) の内部状態 net_{pj} を持ち，非線形の出力関数 $f(x)$ を作用させて式 (11) のユニットからの出力を得る．

$$net_{pj} = \sum_i w_{ji} o_{pi} \tag{9}$$

$$o_{pi} = f(net_{pi}) \tag{10}$$

各ユニットの出力関数には半線形関数のロジスティック関数

$$f(x) = \frac{1}{1 + e^{-x}} \tag{11}$$

を用いる．式 (9) ～ (11) を用いて入力層から順番にネットワーク各層の出力を計算し，その後，2 乗誤差の総和 E_p

$$E_p = \frac{1}{2} \sum_j t_{pj} - opj^2 \tag{12}$$

を小さくするように学習を行なう．そのアルゴリズムは，最初に出力層の誤差信号 δ^o_{pj} を

$$\delta^o_{pj} = (t_{pj} - y_{pj}) f'(net_{pj}) \tag{13}$$

とし，中間層－出力層の重み係数を

$$\Delta w_{ji}(n+1) = \eta(\delta^o_{pj} o_{pi}) + \alpha \Delta w_{ij}(n) \tag{14}$$

として求める．ここで，n は学習の回数を示し，η は学習の係数，α は一回前の重みの変化が今回の重みの変化に与える影響の大きさを決める定数である．次に中間層の誤差信号 δ^i_{pj} を式 (15) のように表す．

$$\delta^i_{pj} = f'(net_{pj}) \sum_k \delta^o_{pk} w_{kj} \tag{15}$$

入力－中間層の重み係数は式 (15) で得られた δ^i_{pj} に式 (14) を用いて変更される．一連のこれらの計算を繰り返して行なうことにより，学習を行なう．

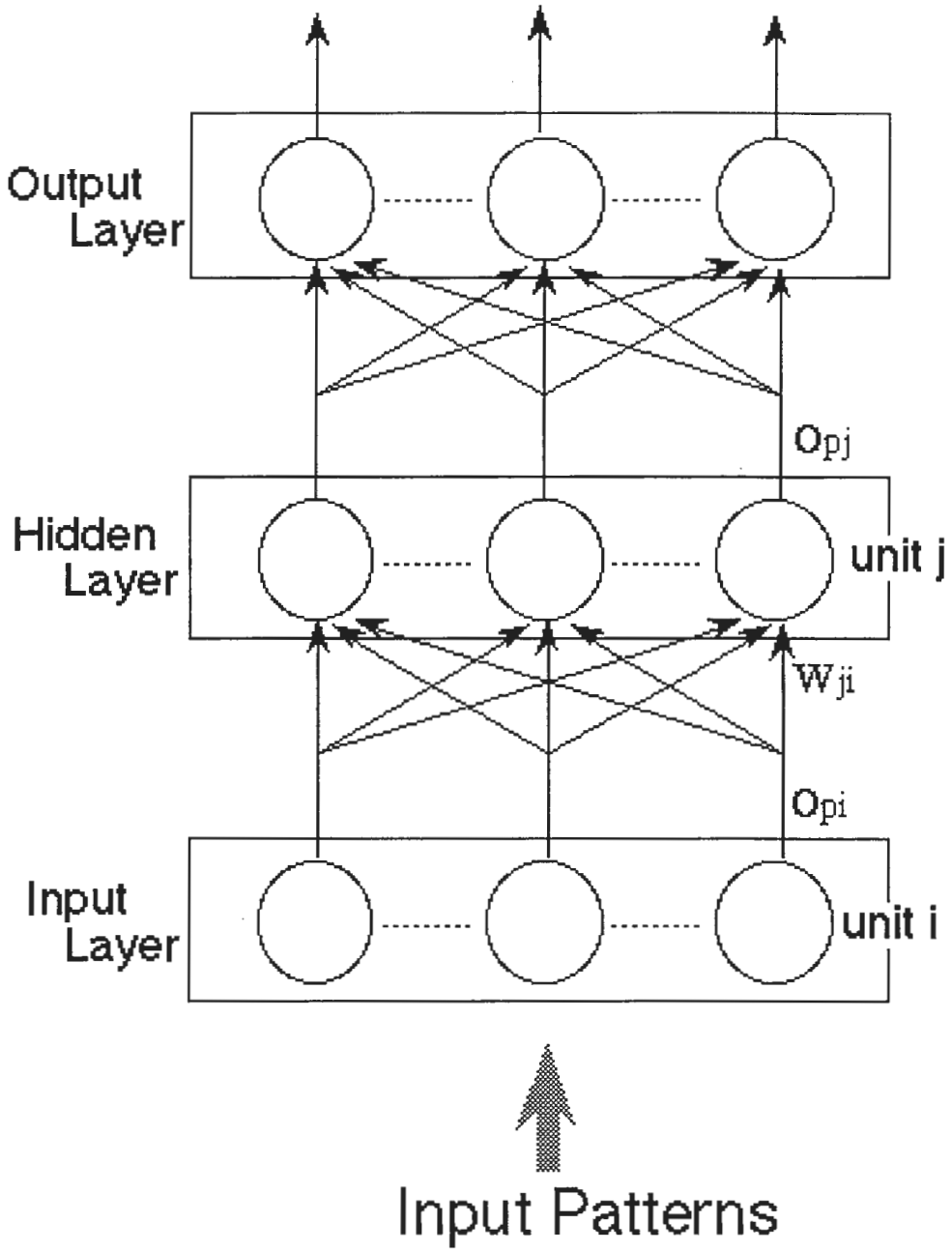


図5 多層ネットワークの構成

図6のリカレントニューラルネットワークにおいて，コンテクスト層から中間層への結合の重み係数をを上で述べた中間層－入力層の重み係数と同様に決定することによって誤差逆伝搬学習を導入する．誤差逆伝搬学習を用いるときには結合の重みを求める方法として，全パターンに対して重み変更量を求め一度に修正する方法と，逐次的に重みを変更する方法がある．ここでは，各パターンは時系列データの一部であるため，後者の方法を取って学習を行なった．

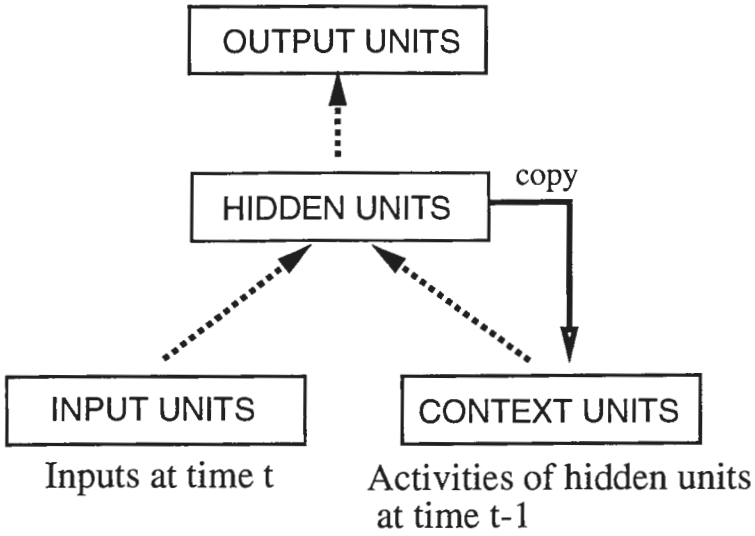


図6. リカレントニューラルネットワークの構成

試作システムでは，オブジェクトの回転および拡大・縮小の操作を行なうときにこの手法を用いている.

6.3 試作システムの機能

意味のある手振りとして“操作対象オブジェクト指定”，“移動”，“回転”，“拡大・縮小”，“操作対象オブジェクト指定の解除”の5種類の手振りを定義した.

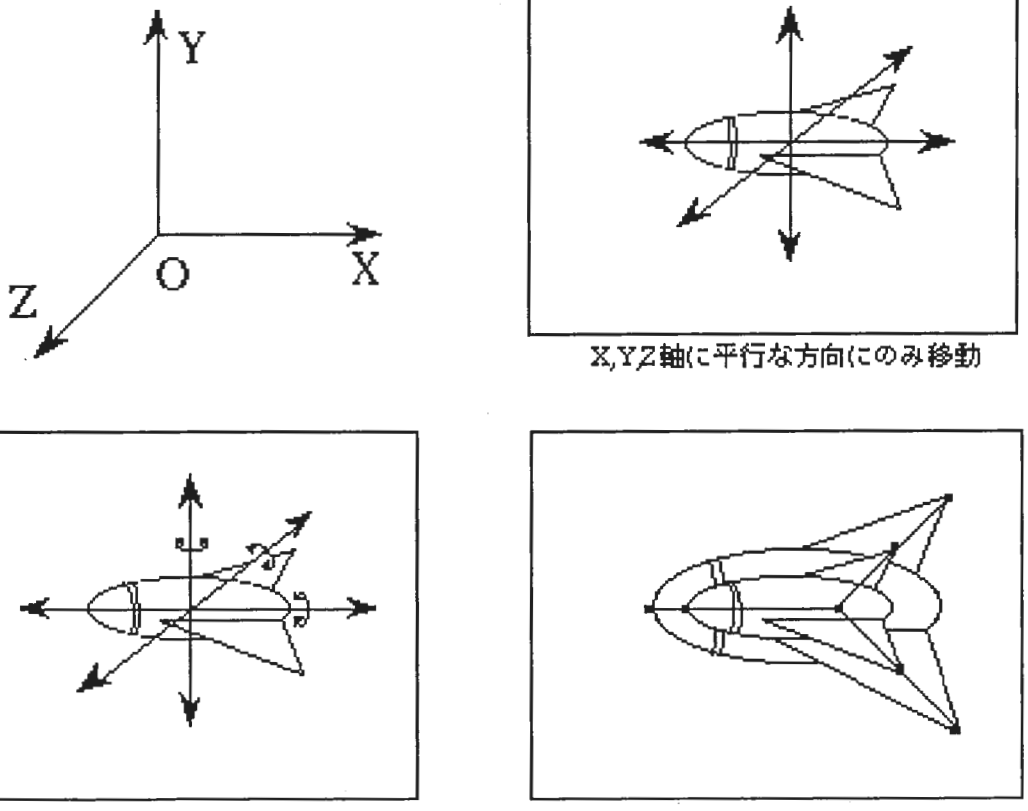


図8 操作によるCGオブジェクトの変化

オブジェクト指定の手振りでは，オブジェクトをつかむ動作によって操作対象を指定する．ここでいうつかむ動作とは，CG空間で定義されたオブジェクトの基準点の位置と手の基準点との距離がある値よりも小さくかつその時に手を閉じた形状（5節での実験中の手形状 g_1, g_2）であることをいい，その条件を満たせばそのオブジェクトを操作対象とする．したがって，つかむという自然な動作では，手を広げた状態で物体の位置まで手を移動させ，閉じる動作であるが，本システムのインプリメントではその時間的な手の変化までには対応していない．また，CGオブジェクトをボリュームのあるものとして扱っていない．同時に操作の対象となるオブジェクトは1つに限定している．

あるオブジェクトへの操作を終了する時には，親指のみを伸ばした手振り（5節の手形状 g_9）をすることによって操作を終了する．

オブジェクトを移動させるには，手を開いた状態で（5節の手形状 g_4, g_{10}）でその広げた手がなす面に垂直な方向に手を動かす．ここで、オブジェクトはCG空間中の X, Y, Z の3軸のいずれかに平行な方向にのみ移動する．この移動方向は，手内部の基準座標の z 軸と X 軸，Y 軸，Z 軸の内積の絶対値がもっとも大きい軸の方向としている．

オブジェクトの回転は，手を軽く開いた状態（5節の手形状 g_6, g_7, g_8）で，手首から先全体を回転させる．回転軸は X, Y, Z のいずれかに平行で各オブジェクトに定められた基準点を通る直線である．座標O-XYZにおいて手内部の基準点から親指先端へのベクトル$\boldsymbol{v}_t$，人差し指先端へのベクトル$\boldsymbol{v}_t$，$\boldsymbol{v}_i$を用いて表した$\boldsymbol{a}_{tmp}$

$$\boldsymbol{a}_{tmp} = \frac{2}{3}\boldsymbol{v}_t + \frac{1}{3}\boldsymbol{v}_i \tag{16}$$

が，X 軸，Y 軸，Z 軸と内積をとり，絶対値がもっとも大きい軸に平行な直線を回転軸としている．手の動きデータからオブジェクトを変化させるパラメータを決定するためにリカレントニューラルネットワークを用いている．手の基準点を通りオブジェクトの回転軸に平行な直線の回りを手の方向が何度変化したかの差分 Δa をとる．ネットワークは前節で述べたリカレントニューラルネットワークにさらにコンテクスト層を2単位時間分もつ構成をなす(図9)．入力層，出力層のユニット数はともに1であり，中間層，2つのコンテクスト層は5個のユニットからなる．Δa の絶対値が入力される．学習パターンは手の回転速度がある程度以上速くなるとオブジェクトの回転を押えるようなパターンを用いた．これによって，オブジェクトが望ましい状態になるまで回転させることを，手をある一定速度以下で回転させすばやく元の状態に手を戻すことを繰り返すことによって，可能にする．手の方向をそのままオブジェクトの回転角度に対応させると，手の構造上一定以上の変化を一度に起こすことはできないが，繰り返し動作を可能にすることによって，望みの状態にまで変化させることができる．

オブジェクトの拡大・縮小は，手を開いたり閉じたりすること(4節の手形状 g_5, g_6, g_7, g_8）で行なう．拡大・縮小はオブジェクトの基準点を中心にして等方的に行なわれる．リカレントニューラルネットワークを用いて，望みの大きさまで動作を繰り返し行なうことでオブジェクトを変化させることを実現している．ネットワークは入力層のユニット数6，中間層，2つのコンテクスト層はそれぞれ20個ずつのユニットを持ち，出力層には2個のユニットを持つ．ネットワークには式(1)の記号を用いて

$$G(t) - G(t-1)$$

が入力される．出力層の2ユニットの出力の差分 d から次式により拡大率 r を決定する．

$$r = \sqrt{(1.0 + \alpha d)} \tag{17}$$

α は経験的に得られた定数である．学習パターンには，回転のものと同様に，手を例えば広げる動作のあとにすばやく閉じる動作があるときには，この閉じる動作の後に広げる動作が繰り返されるものと仮定し，閉じるときにオブジェクトが縮小しないようなパターンを与えた．

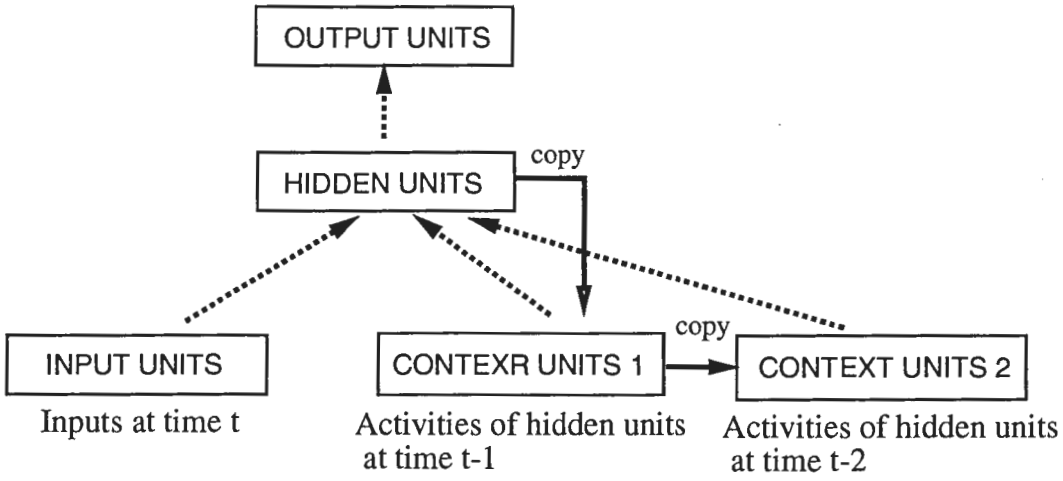


図9 コンテクスト層を2つもつリカレントニューラルネットワーク

6.4 評価

このシステムでは操作を「つかみ，移動させ，はなす」といった直接的な操作よりも抽象なものとし，また変化の方向を限定するなどの制限を加えている．操作対象オブジェクトを指定するとき以外は，操作は手と操作対象のオブジェクトの位置関係に依存しない．操作するときにオブジェクトの位置に手がある必要がないので，上記の直接的な操作よりもユーザの疲労を少なくすることができる．しかし一方で，ユーザが意図せずしてある操作を示す手の形をしているときにおいて，システムはオブジェクトを変化させることがあるなど，誤動作の増やす原因とも考えられる．

回転操作，拡大・縮小操作では，ニューラルネットワークの振舞いが重要であるが，充分望ましい段階までには達していない．特に拡大・縮小操作では，ユーザの意図とは逆にオブジェクトが変化することがあった．ユニット数，サンプルパターンの数，与え方などを検討する必要がある．

手に磁気センサを装着することは不快感をユーザに与える．非装着で手のデータを獲得する方法を実現する必要がある．

7 画像データから手形状認識

磁気センサを手に装着することなく，手の形状，位置 / 方向のデータを取得する手段として2台のカメラから得たステレオ画像を処理することの検討を行なった．その処理の概要および方針を図10に示す．

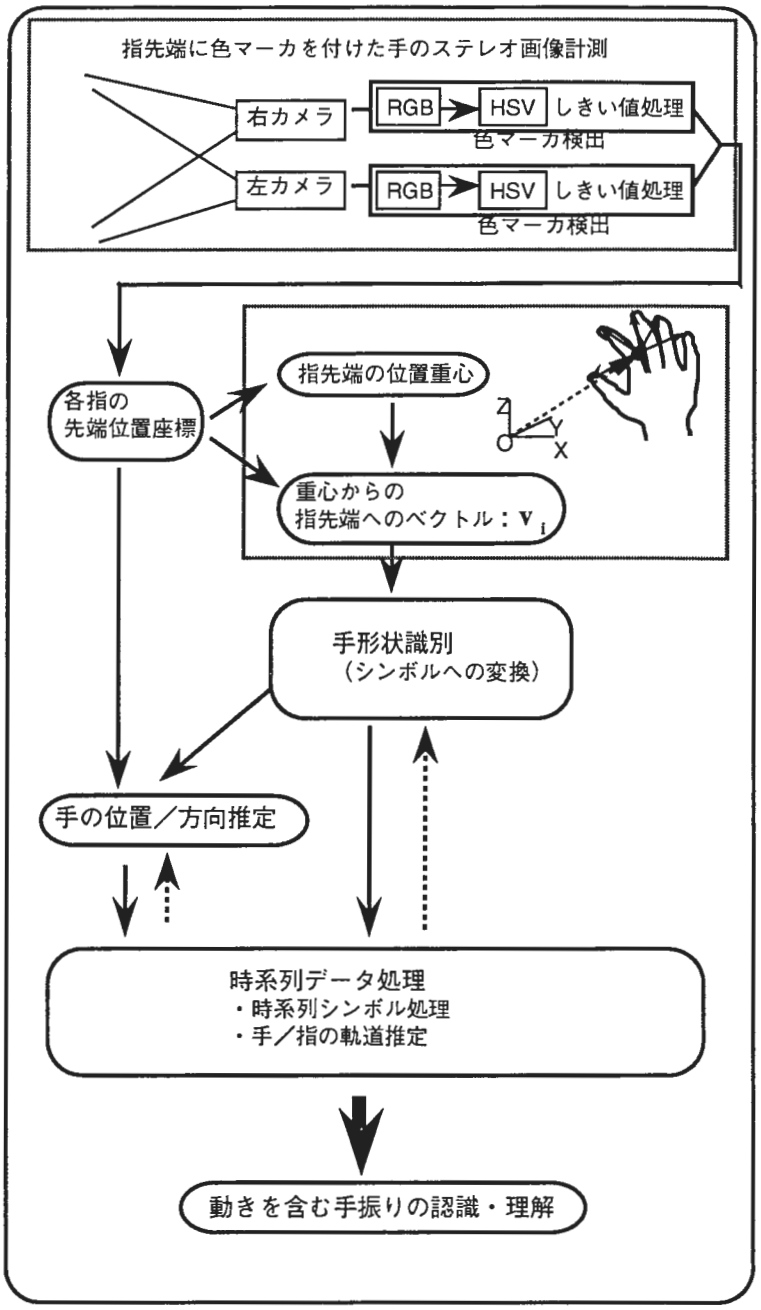


図 10 画像処理を用いた手振り認識

これまで，手振りの基礎的なデータとして以下の方法を提案してきた．すなわち，手内部に基準となる座標系を設定し，その原点のワールド座標系における位置を手の位置とし，２つの座標系のなすオイラー角を手の方向を示すデータとし，手形状は手内部の座標における指先端座標を用いて表現してきた．しかし，画像処理では指先端は特徴点として比較的求めやすいが，手の内部に基準となる点および座標を決定することは困難である．本節では，指先端にマーカを付け、ステレオ画像を処理することにより求められるワールド座標系における先端座標値を計測データとし，手の内部に基準座標を設定することなく，手形状を識別する方法について述べる．また、手の動きを許容する場合には，手掌部分や他の指に遮蔽され常に指先端を特徴点として追跡することができない．このようなオクルージョンがおこる場合の対応について述べる．

7.1 計測環境

図11で示すように2台のカメラを用いて特徴点の位置を計測する．各々のカメラから得られた画像上に特徴点が見つかれば，その3次元座標は特徴点対を三角測量して，容易に計測できる．

右手の各指先端に赤，黄色，緑，シアン，青のマーカーを付ける．各々のカメラからのRGBアナログ信号をHSVデジタル画像に変換し，2つの画像についてしきい値処理し，特徴点としてマーカを抽出する．それぞれの指に違った色のマーカを付けるため，2台のカメラからの画像を処理するときに誤対応が起こることなく，それぞれの指先端に対してワールド座標系における位置座標が求められる．

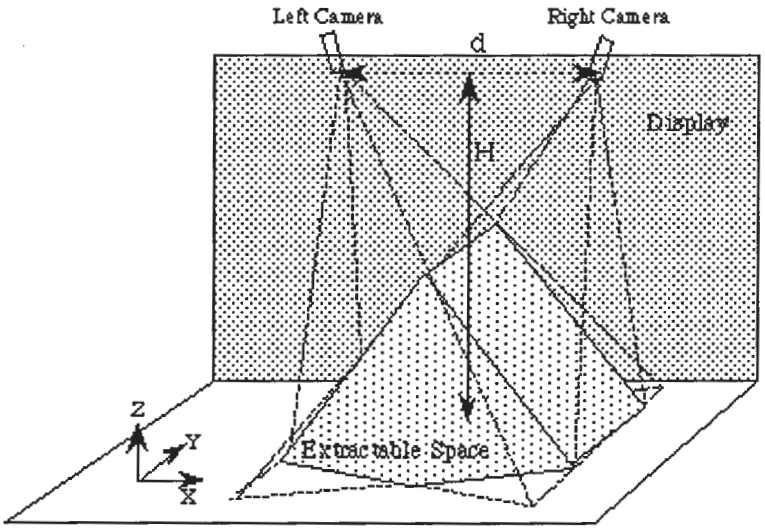


図11. 計測環境
(カメラ間距離 d=120cm, 輻輳角 30.0 °)

7.2 手形状識別

7.2.1 識別アルゴリズム

前節で求められた計測データから手の位置および向きに依存しない手法で手形状を識別する．N本(N = 3 ～ 5)の指先端座標の重心を求め，重心からの各指先端へのベクトルをとる．これらのベクトルの内積は，重心を原点とする直交座標系であれば，手の向きに依存しない．すなわち，手の位置・向きによらない手形状を示すデータであると考えられる．N = 5のときの，このデータの形式は以下のようになる(図2.)．ただし，V_iは計測データ (i = 1...5)，添字1～5は親指から小指を示す．

$$(\boldsymbol{v}_1 \cdot \boldsymbol{v}_1, \quad \boldsymbol{v}_2 \cdot \boldsymbol{v}_1, \quad \boldsymbol{v}_2 \cdot \boldsymbol{v}_2, \ldots \boldsymbol{v}_5 \cdot \boldsymbol{v}_4, \quad \boldsymbol{v}_5 \cdot \boldsymbol{v}_5)$$

$$\boldsymbol{v}_i = \boldsymbol{V}_i - \boldsymbol{g} \ (i = 1...5), \qquad \boldsymbol{g} = 1/5 \sum_{i=1}^{5} \boldsymbol{v}_i$$

識別は5節と同様に部分空間法によって行なった．部分空間法では，入力ベクトルの大きさには関係なく，ベクトル内の要素の相互関係のみに識別結果が依存する．しかし，手形状を考えるとき，データの大きさが重要な意味を持つ．このため上で述

べた内積によるデータベクトルにその大きさの情報を含むデータを1次元付け加えたベクトルを部分空間法で処理することにした.

7.2.2 識別結果

CLAFIC法, 平均学習部分空間法 (ALSM) を用いて識別を行なった. 1人の被験者が手の位置・方向を変化させながら14種類のあらかじめ定められた手形状 (g_1, ... ,g_{14}) を50回繰り返して得られたパターンをデータとし, そのうちのランダムに選んだ20パターンを訓練ベクトルとし, 残りを検査ベクトルとした.

5本の指のデータすべて用いて訓練 / 識別を行なったときの結果を表3に示す. CLAFIC法では各類に対して, 他の類との関係を考慮せず独立に部分空間を決定するために類似した手形状からの入力パターンには誤識別を起こすことがある. 学習によってそれぞれの部分空間を修正するアルゴリズムを含む平均学習部分空間法を用いた結果, 誤識別を減らすことができた. 5つの指先端座標が得られたときには, この方法により, 手の内部に基準座標を設定することなく手形状を識別できた.

	訓練ベクトル (20)		検査ベクトル (30)	
手形状	CLAFIC	ALSM	CLAFIC	ALSM
g_1	0	0	0	0
g_2	4	0	9	0
g_3	0	0	3	2
g_4	18	0	0	0
g_5	0	0	20	1
g_6	6	0	7	0
g_7	10	0	18	2
g_8	13	0	8	2
g_9	0	0	9	1
g_{10}	0	0	0	2
g_{11}	2	0	9	1
g_{12}	1	0	2	0
g_{13}	0	0	2	3
g_{14}	9	0	0	0

表 3. 部分空間法による識別結果, N = 5 のとき
(誤識別数, 棄却パターン数は0であった.)

7.3 計測データに欠落がある場合の対応

すべての指先端からデータが得られない場合を想定して, 3本の指, 4本の指からのデータを用いて識別を行なった. この実験に用いたパターンに対しては, 小指の影響は比較的少なく小指を除いた4本の指での識別結果は5本での結果に比べ, 大きな相違はなかった. しかし, その他の指が見えない場合, 識別できない類が存在することがわかった (表4). これらの類を識別できない類は, ひとまとまりのグループとして以後あつかう. たとえば, 親指のデータがとれないときの結果が表2に示

されている．誤識別が多いが，これは以下のグループ $G_1 \sim G_8$ 内で識別できなかったと考えられる．

$G_1 = \{g_1\}$, $G_2 = \{g_2, g_7, g_{10}\}$, $G_3 = \{g_3\}$, $G_4 = \{g_4\}$,
$G_5 = \{g_5, g_6, g_8, g_9, g_{14}\}$, $G_6 = \{g_{11}\}$, $G_7 = \{g_{12}, g_{13}\}$

手形状	小指のデータなし		親指のデータなし	
	訓練	検査	訓練	検査
g_1	0	0	0	1
g_2	0	0	1	12
g_3	0	4	0	1
g_4	0	0	1	2
g_5	1	4	4	10
g_6	0	1	10	17
g_7	0	0	8	13
g_8	0	2	5	3
g_9	1	2	4	10
g_{10}	0	2	2	5
g_{11}	0	1	0	0
g_{12}	0	0	2	8
g_{13}	0	2	0	18
g_{14}	0	0	6	4

表 4. 平均学習部分空間法 (ALSM) による識別結果

指が 4 本のとき（誤識別数，表中の訓練は訓練ベクトル 20 パターン，検査は検査ベクトル 30 パターンを示す．棄却されたパターンはなかった）

7.4 動きを含む手振りへの対応

手掌部分や他の指に遮蔽されことがあるため，動きのある手振りから常に 5 本の指の先端を特徴点として追跡することはできない．このような場合には手画像から指先端の位置を抽出するだけでは，充分なデータを得ることはできない．しかし，特徴点が 3 つ以上抽出可能なときには，次の簡単な仮定を導入することによって，処理を行う．抽出可能な特徴点が 2 つ以下のときには，データ不足として処理を行なわない．特徴点が 3 以上の場合には，特徴点の抽出できた指のセットごとに前節の親指に対して示したようなグループ分けを行なう．これまで N 個の特徴点が抽出されていて手形状が g_i と識別され，現在 N-1 個の特徴点が抽出されていて手形状が g_j と識別されたとき（ただし，$N = 4$ or 5),

- g_i と g_j がともにグループ G_k に含まれるとき，手形状に変化はなかったとして処理する．
- g_i と g_j がちがうグループに含まれるとき，手形状に変化があったとして識別結果をそのまま用いて処理する．

上の仮定を用いて各時点での手の形状データをシンボルに変換し，シンボル時系列を得る．このシンボル時系列をシンボルの継続時間を含む簡単な辞書データと比較し，認識を行なうことが考えられる．

8 おわりに

手振りを用いたユーザインタフェースの研究の第一歩として，CG オブジェクトを手振りによって操作する模擬実験を被験者を用いて行なった．この結果，それぞれのタスクに対して複数の被験者が共通の手振りを用いて操作を行なおうとしたことがわかり，手振りをある程度汎用的なユーザインタフェースを作る可能性を確認した．また，被験者が使う手振りを意味的な違いによって分類し，それぞれの場合について手振りからどのような情報を抽出すべきかを検討した．

これらの検討から手の形状データを指先端の 3 次元位置座標をもちいて表現することを提案し，その方法に基づくデータを用いて，手形状を識別する実験を行なった．識別実験には 3 種類の部分空間法を用いて行ない，平均部分空間法では実験サンプルのほとんどを正しく識別することができた．また，この表現法を用いることによって，指の関節角度を用いては識別が難しい指先端の接触の有無による形状の相違を簡単に識別することができた．

上記の形状識別で操作の開始，終了を決定し，その間の操作のあたえる変化をリカレントニューラルネットワークで決定する，CG オブジェクトを操作するシステムを試作した．このシステムではニューラルネットワークの振舞いが重要な意味をもつが，この試作の段階では充分望ましい振舞いを行なうに至っていない．ユニット数，サンプルパターンの数や与え方などを検討することにより，ネットワークの振舞いをより望ましいものに近付けることが必要である．

最後に画像データから手形状データを獲得する方法について述べた．指の先端にカラーマーカを付け特徴点としてこれらを追跡することによって，手形状を識別し，手の位置方向を求める．この方法では磁気センサを用いる時のように，手の内部に基準座標を設定することが困難であり，画像から抽出できる指の先端座標のみから手形状データを獲得することになる．5 本の指先の位置データがすべて取得できる時には手形状の識別はある程度可能であるが，ある指が他の指に隠れるなどしてすべての先端位置データが獲得できない時には，手のシルエット画像を利用するなどの他の手法と組み合わせた処理が必要となることがわかった．

今後の研究の方向として次のようなことが考えられる．

(1) 手振りデータの獲得手段として非接触手法すなわち画像処理による方法が一つの重要なテーマとなる．これには 8 節で述べたように，各指のラベリング問題，オクルージョン問題を解決する必要がある．これらの問題解決の指針として，手のシルエット画像や濃淡画像から手の形状を推定する方法 [21]，高速画像処理により手の動きから形状を推定する方法 [22] などを組み合わせるが考えられる．

(2) 手振りを片手に限定せず，両手による手振り，さらには体の他の部分と位置関係を考慮した身振りを取り扱う [23]．画像処理で手振りデータを獲得する時には，(1) で述べた問題はさらに複雑なものとなる．

(3) 動きのある手振りを扱う．6 節では動きのある手振りからアプリケーションに即した量的な出力を求める方法を述べた．シンボリックな動きのある手振り（例えば手話）を認識する方法はまだ課題となっている．その方針として，5 節で

述べたように各時点で手の形状位置などをコード化し，手振りをシンボル時系列としてあつかう方法がある．これはニューラルネットワークを用いる方法 [24] よりも処理データ量の面で有利であると考えられる．

(4) 音声インタフェースとの統合により使い易いユーザインタフェースを構築する．

これらの項目は手振りをユーザインタフェースとしてさらに使い易いものにして行くのに必要な要素である．これらの基礎的な研究を積み上げて行くこととともにユーザインタフェースとして用いることからのフィードバックを研究にいかして行くことが重要である．

参考文献

[1] A. G. Hauptmann: Speech and Gestures for Graphic Image Manipulation, Proceedings of CHI '89, ACM 1989

[2] R. A. Bolt: The Human Interface, Lifetime Learning Puclications, 1984

[3] M. W. Krueger: Artificial Reality II, Adidison-Wesley, 1991

[4] G. Kurtenbach, E. A. Hulteen: Gestures in Human–Computer Communication, In B. Laurel eds., The art of human-computer interface design, pp.309-317, Addison Wesley 1990

[5] D. J. Sturman, D. Zeltzer, and S. Pieper: Hands-on Interaction With Virtual Environment, Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology, pp.19-24 1989

[6] D. Weiner, S. K. Ganapathy: A Synthetic Visual Environment with Hand Gesturing and Voice Input, Proceedings of CHI '89, ACM 1989

[7] 大西，竹村，伴野，岸野：手振りを用いたユーザインタフェースに関する一検討，電子情報通信学会 HC90-22, 1990

[8] 大西，竹村，岸野：手形状記述の一提案，電子情報通信学会春季全大 A-250, 1991

[9] 大西，竹村，岸野：手振り認識のための手形状記述について，テレビジョン学会技術報告 ICS'91-32, 1991

[10] 大西，竹村，岸野：静的手形状識別の検討，電子情報通信学会秋季全大 A-130, 1991

[11] T. Onishi, H. Takemura, F. Kishino: A study of hand gesture recgnition for an interactive environment, 第 7 回 HI シンポジウム， 1991

[12] 大西，竹村，岸野：手振りを用いた CG オブジェクト操作システム，電子情報通信学会春季全大 A-292, 1992

[13] 大西，竹村，岸野：画像計測データからの静的手形状認識，情報処理学会第 45 回全国大会， 1992

[14] 大西，竹村，岸野：手画像を用いた動きを含む手振り認識，第 8 回 HI シンポジウム， 1992

[15] Erkki Oja: Subspace Methods of Pattern Recognition, John Wiley & Sons, 1983

[16] 黒川，神戸：身振り言語の記述とコンピュータ化辞書の構築，第5回 HI シンポジウム， 1989

[17] 高橋，岸野：手振り認識方法とその応用，電子情報通信学会論文誌 J73-D-II, pp.347-352, 1990

[18] G. Mourouvapin, T. Onishi, H. Takemura, F. Kishino: Hand Motion Interpetaion Using Neural Network, Proceedings of the 4th Sapporo International Computer Graphics Symposium, pp.38-42 1990

[19] D. E. Rumelhart, J. C. McClelland: Parallel Distributed Processing, MIT Press, 1986

[20] J. L. Elman: Finding structures in time, Cognitive Sience vol.14, pp.179-211, 1990

[21] 亀田，美濃，池田：シルエットを利用した手指の三次元形状推定法， MIRU '92, II-239, 1992

[22] 石淵，竹村，岸野：パイプライン型画像処理装置を用いた実時間手形状認識，計測自動制御学会 HI 部会， N&R Vol.7 pp.275-280 1992

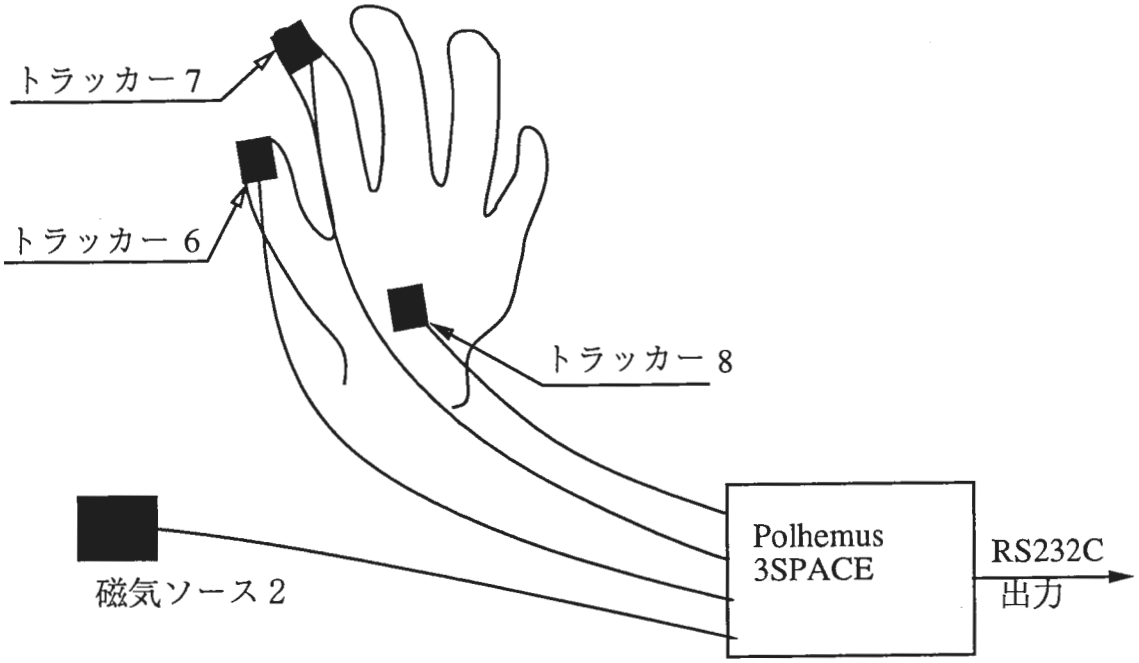
[23] 石淵，竹村，岸野：両手操作のための実時間手形状認識，電子情報通信学会秋季全大， A-149, 1992

[24] K. Murakami, H. Taguchi:Gesture Recognition using Recurrent Neural Networks, Proceedings of CHI 90, ACM 1990

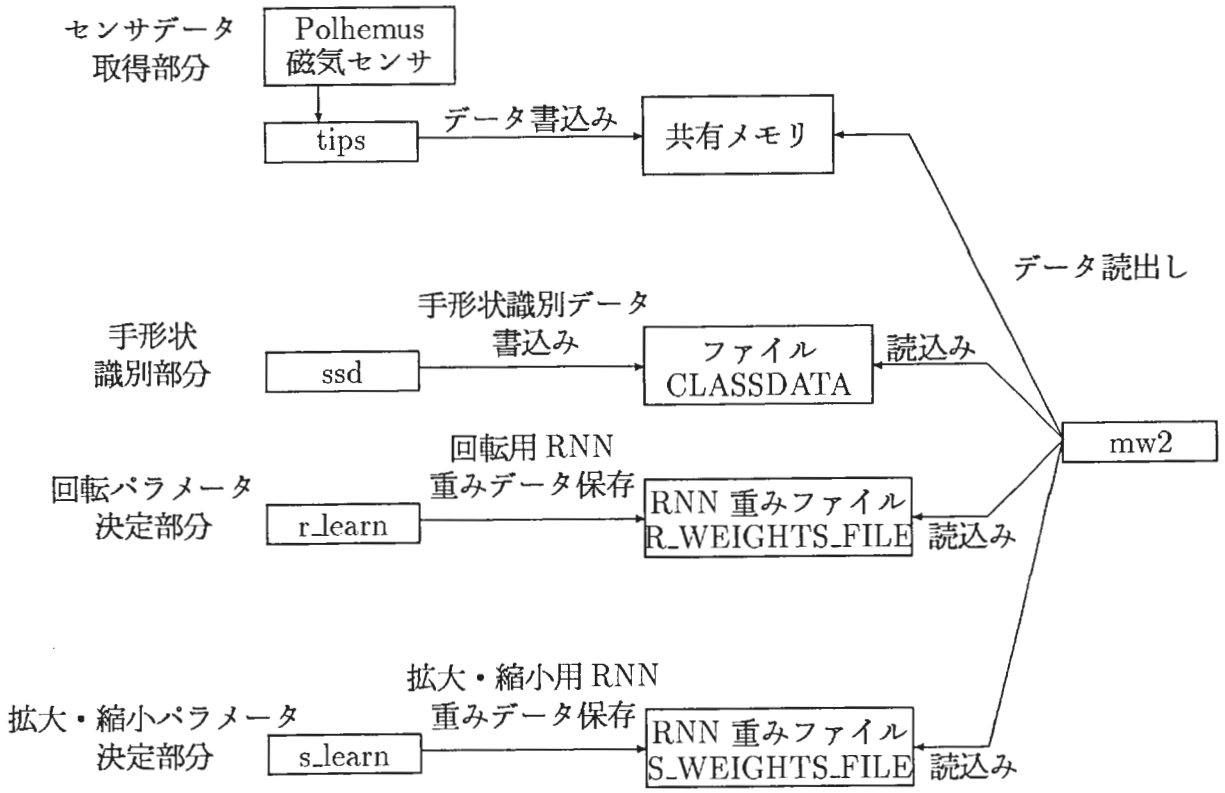
[25] 3Space, Users Manual, Polhemus, P.O. Box 560, Colchester, Vermont

A　試作システム構成

手形状データ取得部の構成



ソフトウェア構成



B　プログラムソースリスト

ワークステーション ciris2 のディレクトリ /usr1/onishi/kouobu/crab/claw/world に CG 操作システムのプログラムソースおよび実行形式ファイルがある.

それぞれの実行形式ファイルを作成するのに必要なプログラムは次の通り.

```
mw2: makeworld2.c(*) blockworld.c readmtl.c setlight.c scankw.c readobj.c
     readmap.c matrix.c(*) platform.c(*) subspace.c(*) shs.c(*) angle.c(*)
     s_network.c(*) r_network.c(*) network.c(*)
r_learn: r_learn.c(*) r_network.c(*) network.c(*)
s_learn: s_learn.c(*) s_network.c(*) network.c(*)
ssd: ssd.c(*) shs.c(*) matrix.c(*) subspace.c(*) platform.c(*)
tips: tips.c(*)
```

(*) 印をつけたプログラムソースを以下につける.

```
/*      block world debugg tools        */
/*                                      */
/*      showblock                       */
/*      displays world file on display  */
/*      This is no stereo               */
/*                                      */
/* リンクモデルで構成される世界を表示   */
/*   ためのテストプログラム             */
/*                                      */
/*   だったんだけど，わたくし大西剛が   */
/*   おろかにも自分用に変更してしまっ   */
/*   たのでした.                        */
/*   This is a main part                */
/*        of mw version 2.0.1.          */
/*        09/27/91                      */

#include <signal.h>
#include <stdio.h>
#include <errno.h>
#include "gl.h"
#include "device.h"
#include <math.h>

#include "blockworld.h"
#include "subspace.h"
#include "claw.h"
#include "network.h"
#include "get_time.h"


#define YOFFSET   (-50.0)
#define ZOFFSET   (-50.0)
#define COSD(x) cos(M_PI*(x)/180.0)
#define SIND(x) sin(M_PI*(x)/180.0)
#define ABS(x)   (((x) > 0)? (x):-(x))

extern double timeMatrix[TIMES];
extern double vpMatrix[TIMES][3];
extern double apMatrix[TIMES][3];
extern double vtMatrix[TIMES][3];
extern double viMatrix[TIMES][3];
extern double vcMatrix[TIMES][3];
extern double vrMatrix[TIMES][3];
extern double vsMatrix[TIMES][3];
extern double avtMatrix[TIMES][3];
extern double aviMatrix[TIMES][3];
extern double avcMatrix[TIMES][3];

extern void GetHandData();
extern char *DataTop();
extern void LastData();
/* extern void LearningStage(); */
extern int LoadClassData();
extern int RealTimeRecognize();
extern int EndHand();

extern struct WORLDPOINTER def_world;

int rotflag = 0;        /* dummy for blockworld */
int movflag = 0;
int sizflag = 0;
int setaxis = 0;
int eight = 0;

int porm;
int handshape = -1;
```



```
float cposx, cposy, cposz, crot, cele;
float offx, offy, offz, offrotx, offroty, offrotz;
float trx, try, trz, rotx, roty, rotz;
float otrx, otry, otrz, dtrx, dtry, dtrz;
float axx, axy, axz;
double azi, ele, rol, nazi, nele, nrol;
double x, y, z, dx, dy, dz, ox, oy, oz;
float rnetout;
float da, ovt[3], ovi[3];
float howbig = 1.0, angleaxis;

struct unit *r_pu[R_TOTAL], *s_pu[S_TOTAL];

double r_tampon1[R_NUM_CON1], r_tampon2[R_NUM_CON2];
extern double  r_input_pat[R_PATTERNS+1][R_NUM_IN];
double s_tampon1[S_NUM_CON1], s_tampon2[S_NUM_CON2];
extern double  s_input_pat[S_PATTERNS+1][S_NUM_IN];

struct BLOCKWORLD *pickobj, *pickblock(), *pickblockdist();
struct POLYGON *obcur0, *obcur1, *obcur2, *obcur3;
usage(s)
  char *s;
{
  printf("Usage: %s blockworld-file \n",s);
  exit();
}

int menu, killmenu;
char worldfile[256];
char *head;

FILE *fopen();

main(n,p)
  int n;
  char *p[];
{

  if(n!=2) usage(p[0]);

  strcpy(worldfile, p[1]);
  init_networks();

  if (LoadClassData(DEFCLASSDATA) < 0)
    exit();

  initialize();
  head = DataTop();
  loadmtldbase("master.mtl");

  readblockworld(worldfile);

  drawimage();


  for(;;)
    main_loop();
}

main_loop()
{
  drawitems();
  processinput(); /* process input */
  gestureprocess();
}
```

```
133
134 initialize()
135 {
136   struct POLYGON *readobj();
137   maxsize(XMAXSCREEN, YMAXSCREEN);
138   minsize(50,50);
139   winopen("Show World");
140
141   doublebuffer();
142   RGBmode();
143   subpixel(TRUE);
144
145   zbuffer(1);
146   backface(TRUE);
147   concave(TRUE);
148
149   gconfig();
150
151   RGBcolor((short)0, (short)0, (short)0);
152   clear();
153   swapbuffers();
154   clear();
155
156   mmode(MVIEWING);
157
158   def_light_calc("master.lgt");
159   use_light_calc("lightset");
160
161   initqueue();
162
163   killmenu = defpup("Do you want to exit h? %t|yes|no");
164   menu = newpup();
165   addtopup(menu, "Show World %t|exit program|next scene");
166
167   obcur0 = readobj("kyu0.obj", 1.0);
168   obcur1 = readobj("kyu1.obj", 1.0);
169   obcur2 = readobj("kyu2.obj", 1.0);
170   obcur3 = readobj("arrow.obj", 1.0);
171
172   cposx = cposy = cposz = crot = cele = 0.0;
173 }
174
175 /* Process input from the window manager */
176
177 processinput()
178 {
179   struct BLOCKWORLD *bw;
180   short val;
181
182   int dev;
183   int domove;
184   int menuval, kmenuval;
185
186   domove = FALSE;
187   while(qtest()){
188     dev = qread(&val);
189     switch(dev){
190     case ESCKEY:
191       fexit();
192       break;
193     case LEFTMOUSE:
194     case MIDDLEMOUSE:
195       break;
196     case RIGHTMOUSE:
197       if(val == 1){
198         menuval = dopup(menu); /* returns the menu item chosen */
```



```
199         if(menuval == 1){ /* if exit chosen, ask user */
200           kmenuval = dopup(killmenu); /* to reconfirm */
201           if(kmenuval == 1)
202             fexit();
203         }
204
205         if(menuval == 2){
206           if(readbwcontinue() == -2) exit();
207           drawimage(cposx, cposy, cposz, crot, cele);
208           }
209       }
210       break;
211     case REDRAW:          /* window manager asking for redraw. */
212       reshapeviewport(); /* this sapes the window. */
213       drawimage(cposx, cposy, cposz, 0, 0);
214       /* this redraws the image. */
215       break;
216     case SW0:
217     case SW2:
218     case SW3:
219     case SW4:
220     case SW5:
221     case SW6:
222     case SW7:
223     case DIAL0:
224     case DIAL1:
225       break;
226     case DIAL2:
227       cposy = val;
228       break;
229     case DIAL3:
230       cposz = val;
231       break;
232     case SW1:
233     default:
234         break;
235     }
236   }
237   if (getbutton(SW1))
238     howbig = 1.0;
239 }
240
241 gestureprocess()
242 {
243   struct BLOCKWORLD *block_world_pick();
244   GetHandData(head);
245   handshape = RealTimeRecognize();
246   printf("%d\n", handshape);
247   switch(handshape) {
248   case 0:
249   case 1:
250     if (pickobj == NULL)
251       pickobj = block_world_pick((float)vpMatrix[0][0],
252                                  (float)vpMatrix[0][1]+YOFFSET,
253                                  (float)vpMatrix[0][2]+ZOFFSET,
254                                  &def_world);
255     sizflag = 0;
256     eight = 0;
257   case 2:
258     rotflag = 0;
259     eight = 0;
260     break;
261   case 3:
262     movflag = 1;
263     rotflag = 0;
264     sizflag = 0;
```

```
265       eight = 0;
266       break;
267     case 4:
268       movflag = 0;
269       rotflag = 0;
270       sizflag = 1;
271       eight = 0;
272       break;
273     case 5:
274       movflag = 0;
275       rotflag = 1;
276       sizflag = 1;
277       eight = 0;
278       break;
279     case 6:
280       movflag = 0;
281       rotflag = 1;
282       sizflag = 1;
283       eight = 0;
284       break;
285     case 7:
286       movflag = 0;
287       rotflag = 1;
288       sizflag = 1;
289       eight = 0;
290       break;
291     case 8:
292       if (eight >= 10)
293         if (pickobj != NULL) {
294           pickobj = (struct BLOCKWORLD *) NULL;
295           setaxis = 0;
296           eight = 0;
297         }
298       eight++;
299       movflag = 0;
300       rotflag = 0;
301       sizflag = 0;
302       break;
303     case 9:
304       movflag = 1;
305       rotflag = 0;
306       sizflag = 0;
307       eight = 0;
308       break;
309     default:
310       movflag = 0;
311       rotflag = 0;
312       sizflag = 0;
313       eight = 0;
314       break;
315     }
316
317     if (movflag)
318       setrail();
319     else
320       dtrx = dtry = dtrz = 0;
321     otrx = vpMatrix[0][0];
322     otry = vpMatrix[0][1];
323     otrz = vpMatrix[0][2];
324     if (rotflag) {
325       if (pickobj != NULL)
326         setrotaxis();
327     }
328     else if (pickobj != NULL)
329       if (setaxis != 0)
330         setaxis = 0;
```



```
331   calcparams();
332 /*  ovt[0] = vtMatrix[0][0];
333   ovt[1] = vtMatrix[0][1];
334   ovt[2] = vtMatrix[0][2];
335   ovi[0] = viMatrix[0][0];
336   ovi[1] = viMatrix[0][1];
337   ovi[2] = viMatrix[0][2]; */
338 }
339
340
341
342 Matrix idmat = { 1.0, 0.0, 0.0, 0.0,
343                  0.0, 1.0, 0.0, 0.0,
344                  0.0, 0.0, 1.0, 0.0,
345                  0.0, 0.0, 0.0, 1.0, };
346
347 drawitems()
348 {
349   if (pickobj != NULL) {
350     mysetvalue(pickobj, pickobj->bw_mag_scale * howbig,
351                pickobj->bw_pos[0]+dtrx, pickobj->bw_pos[1]+dtry,
352                pickobj->bw_pos[2]+dtrz,
353 /*             pickobj->bw_rot[0]+rotx,
354                pickobj->bw_rot[1]+roty, pickobj->bw_rot[2]+rotz,*/
355                rotx, roty, rotz,
356                pickobj->bw_mask, pickobj->bw_id, pickobj->bw_obj);
357
358     printf("----- %d ------\n", pickobj->bw_id);
359 /*    printf("%f %f %f\n", dtrx, dtry, dtrz); */
360   }
361   drawimage(cposx, cposy, cposz, crot, cele);
362 }
363
364 drawimage(x,y,z,rot,ele)
365   float x, y, z, rot, ele;
366 {
367   long xsize, ysize;
368
369   getsize(&xsize, &ysize);
370
371   mmode(MPROJECTION);
372
373   window(-5.0, 5.0,
374          -5.0*((float)ysize/xsize), 5.0*((float)ysize/xsize), 8.0,10000.0);
375
376   lookat(0.0, y, z, 0.0, 0.0, 0.0, 0);
377   mmode(MVIEWING);
378   loadmatrix(idmat);
379
380   cpack(0xff000000);
381   clear();
382   zclear();
383
384   drawbw();
385   mydrawcursor(vpMatrix[0][0], vpMatrix[0][1]+YOFFSET, vpMatrix[0][2]+ZOFFSET,
386                apMatrix[0][0], apMatrix[0][1], apMatrix[0][2], obcur0, 0.2);
387   mydrawcursor(avtMatrix[0][0], avtMatrix[0][1]+YOFFSET,
388                avtMatrix[0][2]+ZOFFSET, 0.0, 0.0, 0.0,
389                obcur1, 0.2);
390   mydrawcursor(aviMatrix[0][0], aviMatrix[0][1]+YOFFSET,
391                aviMatrix[0][2]+ZOFFSET, 0.0, 0.0, 0.0,
392                obcur2, 0.2);
393
394
395 /*    mydrawarrow(vpMatrix[0][0], vpMatrix[0][1]+YOFFSET,
396                   vpMatrix[0][2]+ZOFFSET,  axx, axy, axz,
```

```
397                 obcur3, 1.0); */
398
399   swapbuffers();
400 }
401
402 initqueue()
403 {
404   setvaluator(DIAL0, (short)0, (short)-5000, (short)5000);
405   setvaluator(DIAL1, (short)0, (short)-100, (short)100);
406   setvaluator(DIAL2, (short)0, (short)0, (short)200);
407   setvaluator(DIAL3, (short)300, (short)0, (short)600);
408
409   qdevice(ESCKEY);
410   qdevice(MOUSE1);
411   qdevice(MOUSE2);
412   qdevice(MOUSE3);
413   qdevice(DIAL0);
414   qdevice(DIAL1);
415   qdevice(DIAL2);
416   qdevice(DIAL3);
417   qdevice(DIAL4);
418   qdevice(DIAL5);
419   qdevice(DIAL6);
420   qdevice(DIAL7);
421   qdevice(SW0);
422   qdevice(SW1);
423   qdevice(SW2);
424   qdevice(SW3);
425   qdevice(SW4);
426   qdevice(SW5);
427 }
428
429 mydrawcursor(x, y, z, rx, ry, rz, obcur, mag)
430       float x, y, z;
431       float rx, ry, rz;
432       struct POLYGON *obcur;
433       float mag;
434 {
435   pushmatrix();
436   translate((Coord)x, (Coord)y, (Coord)z);
437   rot(rx, 'x');
438   rot(ry, 'y');
439   rot(rz, 'z');
440   scale(mag, mag, mag);
441   calldrawobj(obcur);
442   popmatrix();
443 }
444
445 mydrawarrow(x, y, z, rx, ry, rz, obcur, mag)
446       float x, y, z;
447       float rx, ry, rz;
448       struct POLYGON *obcur;
449       float mag;
450 {
451   pushmatrix();
452   translate((Coord)x, (Coord)y, (Coord)z);
453   rot(ry, 'y');
454   rot(rz, 'z');
455   scale(mag, mag, mag);
456   calldrawobj(obcur);
457   popmatrix();
458 }
459
460 setrail()
461 {
462
```



```
463   double x, y, z;
464   x = ABS(COSD(apMatrix[0][0])*SIND(apMatrix[0][1])*COSD(apMatrix[0][2])
465     + SIND(apMatrix[0][0])*SIND(apMatrix[0][2]));
466   y = ABS(SIND(apMatrix[0][0])*SIND(apMatrix[0][1])*COSD(apMatrix[0][2])
467     - COSD(apMatrix[0][0])*SIND(apMatrix[0][2]));
468   z = ABS(COSD(apMatrix[0][1])*COSD(apMatrix[0][2]));
469
470
471   if (x >= y && x >= z) {
472     trx = vpMatrix[0][0];
473     dtry = dtrz = 0;
474     dtrx = trx-otrx;
475   }
476   else if (y >= z) {
477     try = vpMatrix[0][1];
478     dtry = try-otry;
479     dtrz = dtrx = 0;
480   }
481   else {
482     trz = vpMatrix[0][2];
483     dtrz = trz - otrz;
484     dtrx = dtry = 0;
485   }
486
487 }
488
489 setrotaxis()
490 {
491   double ax, ay, az;
492   double tix, tiy, tiz;
493
494   ax =  2.0*avtMatrix[0][0]/3.0 + aviMatrix[0][0]/3.0 - vpMatrix[0][0];
495   ay =  2.0*avtMatrix[0][1]/3.0 + aviMatrix[0][1]/3.0 - vpMatrix[0][1];
496   az =  2.0*avtMatrix[0][2]/3.0 + aviMatrix[0][2]/3.0 - vpMatrix[0][2];
497
498   tix = (double) avtMatrix[0][0] - aviMatrix[0][0];
499   tiy = (double) avtMatrix[0][1] - aviMatrix[0][1];
500   tiz = (double) avtMatrix[0][2] - aviMatrix[0][2];
501
502   z = atan2(tiy, tix);
503   y = atan2(tix, tiz);
504   x = atan2(tiz, tiy);
505
506   dx = 180.0*(x - ox)/M_PI;
507   dy = 180.0*(y - oy)/M_PI;
508   dz = 180.0*(z - oz)/M_PI;
509
510   axx = 0.0;
511   if (setaxis == 0) {
512     if ((ABS(ax) >= ABS(ay)) && (ABS(ax) >= ABS(az)))
513       setaxis = 1;
514     else if (ABS(ay) >= ABS(az))
515       setaxis = 2;
516     else
517       setaxis = 3;
518   }
519
520   switch (setaxis) {
521   case 1:
522     da = (float) dx;
523     axy = axz = .0;
524     break;
525   case 2:
526     setaxis = 2;
527     da = (float) dy;
528     axy = .0;
```

```
529       axz = 90.0;
530       break;
531     case 3:
532       setaxis = 3;
533       da = (float) dz;
534       axy = 90.0;
535       axz = .0;
536       break;
537     defaults:
538       break;
539     }
540     if (da < 0) {
541       porm = 1;
542       da *= -1;
543     }
544     else
545       porm = 0;
546     ox = x;
547     oy = y;
548     oz = z;
549 }
550
551 fexit()
552 {
553     LastData(head);
554     EndHand();
555     exit(0);
556 }
557
558 calcparams() /* for recurrent neural network */
559 {
560     double netx, nety, netz;
561     if (pickobj == NULL)
562       return;
563     if (rotflag) {
564       r_movement(r_pu, da/10.0);
565       switch(setaxis) {
566       case 1:
567         netx = M_PI*rnetout/180.0;
568         xrotangle(azi, ele, rol, &nazi, &nele, &nrol, netx);
569         break;
570       case 2:
571         nety = M_PI*rnetout/180.0;
572         yrotangle(azi, ele, rol, &nazi, &nele, &nrol, nety);
573         break;
574       case 3:
575         netz = M_PI*rnetout/180.0;
576         zrotangle(azi, ele, rol, &nazi, &nele, &nrol, netz);
577          break;
578       defaults:
579         break;
580       }
581       rotx = (float)180.0*nazi/M_PI;
582       roty = (float)180.0*nele/M_PI;
583       rotz = (float)180.0*nrol/M_PI;
584
585       fprintf(stdout, "%f\n", rnetout);
586       azi = nazi;
587       ele = nele;
588       rol = nrol;
589
590     }
591
592     if (sizflag)
593       s_movement(s_pu, s_tampon1, s_tampon2);
594     else
```



```
595       howbig = 1.0;
596     if (movflag)
597       ;
598 }
599
600 init_networks()
601 {
602     r_create_processing_units(r_pu, r_tampon1, r_tampon2);
603     s_create_processing_units(s_pu, s_tampon1, s_tampon2);
604     r_load_weights(r_pu);
605     s_load_weights(s_pu);
606 }
607
608 r_movement(pu, da)
609       struct unit *pu[];
610       float da;
611 {
612     r_input_pat[R_PATTERNS][0] = (double) da;
613     r_recongnize(pu);
614 }
615
616 r_recongnize(pu)
617       struct unit *pu[];
618 {
619     double angle;
620
621     r_init_input_units(pu, R_PATTERNS);
622
623     r_propagate(pu);
624
625     angle = 10 * pu[R_OUT_UID(0)]->output;
626     if (angle < 1 && angle > -1)
627       angle = 0;
628     else if (porm)
629       angle *= -1.0;
630
631     rnetout = (float) angle;
632 }
633
634 s_movement(pu, tampon1, tampon2)
635       struct unit *pu[];
636       double tampon1[], tampon2[];
637 {
638
639     s_input_pat[S_PATTERNS][0] = vtMatrix[0][0]-vtMatrix[1][0];
640     s_input_pat[S_PATTERNS][1] = vtMatrix[0][1]-vtMatrix[1][1];
641     s_input_pat[S_PATTERNS][2] = vtMatrix[0][2]-vtMatrix[1][2];
642     s_input_pat[S_PATTERNS][3] = viMatrix[0][0]-viMatrix[1][0];
643     s_input_pat[S_PATTERNS][4] = viMatrix[0][1]-viMatrix[1][1];
644     s_input_pat[S_PATTERNS][5] = viMatrix[0][2]-viMatrix[1][2];
645 /*
646     s_input_pat[S_PATTERNS][0] = 0;
647     s_input_pat[S_PATTERNS][1] = 0;
648     s_input_pat[S_PATTERNS][2] = 0;
649     s_input_pat[S_PATTERNS][3] = 0;
650     s_input_pat[S_PATTERNS][4] = 0;
651     s_input_pat[S_PATTERNS][5] = 0;
652 */
653     s_recognize(pu, tampon1, tampon2);
654 }
655
656 s_recognize(pu, tampon1, tampon2)
657       struct unit *pu[];
658       double tampon1[], tampon2[];
659 {
660     double sfactor;
```

```
661
662   s_init_input_units(pu, tampon1, tampon2, S_PATTERNS);
663
664   s_propagate(pu, tampon1, tampon2);
665
666   sfactor = pu[S_OUT_UID(0)]->output - pu[S_OUT_UID(1)]->output;
667
668
669   sfactor -= .000653;
670   if (sfactor > 0) sfactor *= 2.0;
671   if (sfactor < 0) sfactor *= 5.0;
672   if (sfactor <= -1) sfactor = 0;
673   if (sfactor < .001 && sfactor > -.000001)
674     sfactor = 0;
675   printf("%lf %f %f \n", sfactor, pu[S_OUT_UID(0)]->output, pu[S_OUT_UID(1)]->ou
    tput);
676   howbig = (float) sqrt(1+sfactor);
677 }
678
679
680
681
682
683
```

```
 1 /* angle.c */
 2 #include <math.h>
 3
 4 xrotangle(a, e, r, na, ne, nr, theta)
 5       double a, e, r, *na, *ne, *nr, theta;
 6 {
 7   double alpha, beta, gamma, epsilon;
 8
 9   alpha = acos(sin(e));
10   *ne = asin(cos(alpha+theta));
11
12   beta = acos(sin(a)*cos(e));
13   epsilon = cos(beta+theta)/cos(*ne);
14   if (epsilon > 1.0) epsilon = 1.0;
15   if (epsilon < -1.0) epsilon = -1.0;
16   *na = acos(epsilon);
17
18   gamma = acos(cos(e)*cos(r));
19   epsilon = cos(gamma+theta)/cos(*ne);
20   if (epsilon > 1.0) epsilon = 1.0;
21   if (epsilon < -1.0) epsilon = -1.0;
22   *nr = acos(epsilon);
23 }
24
25
26 yrotangle(a, e, r, na, ne, nr, theta)
27       double a, e, r, *na, *ne, *nr, theta;
28 {
29   double alpha, beta, gamma, epsilon;
30
31   alpha = acos(sin(e));
32   *ne = asin(cos(alpha+theta));
33   beta = acos(cos(a)*cos(e));
34   epsilon = cos(beta+theta)/cos(*ne);
35   if (epsilon > 1.0) epsilon = 1.0;
36   if (epsilon < -1.0) epsilon = -1.0;
37   *na = acos(epsilon);
38
39   gamma = acos(cos(e)*cos(r));
40   epsilon = cos(gamma+theta)/cos(*ne);
41   if (epsilon > 1.0) epsilon = 1.0;
42   if (epsilon < -1.0) epsilon = -1.0;
43   *nr = acos(epsilon);
44 }
45
46 zrotangle(a, e, r, na, ne, nr, theta)
47       double a, e, r, *na, *ne, *nr, theta;
48 {
49   double alpha, beta, gamma, delta, epsilon;
50
51   alpha = acos(cos(a)*cos(e));
52   beta = acos(sin(a)*cos(e));
53   *na = atan(cos(beta+theta)/cos(alpha+theta));
54   epsilon = cos(alpha+theta)/cos(*na);
55   if (epsilon > 1.0) epsilon = 1.0;
56   if (epsilon < -1.0) epsilon = -1.0;
57   *ne = acos(epsilon);
58
59   gamma = acos(cos(a)*sin(e)*sin(r)-sin(a)*cos(r));
60   delta = acos(cos(a)*cos(r)+sin(a)*sin(e)*sin(r));
61   epsilon = cos(*na)*cos(delta+theta)-sin(*na)*cos(gamma+theta);
62   if (epsilon > 1.0) epsilon = 1.0;
63   if (epsilon < -1.0) epsilon = -1.0;
64   *nr = acos(epsilon);
65 }
66
```


67

```
/*  subspace.c                                                          */
/*  ``パターン認識と部分空間法'' エルッキ・オヤ著 を参考にしたライブラリ */
/*  This program is written by Takeshi Onishi and partially             */
/*  based on ``Subspace Methods of Pattern Recognition'' Erkki Oja.     */
/*                                                                      */
/*  コメント中の SMPR は上記の文献を指す.                               */
/*                                                                      */
/*                                                      04/04/91        */
/*                                                                      */

#include <stdio.h>
#include <math.h>
#include "subspace.h"

extern int amean(), vari(), corr(), jacoji();
extern int mmul(), mtral();
extern void mprintf();
extern int OrthoVector();

int ClassCorelationMatrix();


void SortEugenValues();
void SortEvIn();
void SwapEvIn();
void SwapEugenValues();
void SwapEugenVectors();
int MulSimMethod();
struct Class *SetClass();
void ReleaseClass();


/* データの類相関行列を求める                                           */
/*                                                                      */
/* 入出力 a     double型の配列                                          */
/*              配列 a[m][l] に m個のl次元のベクトルを格納する.         */
/*              演算後は各ベクトルが正規化されている                    */
/*                                                                      */
/* 入力   l     int 各ベクトルの次元数                                  */
/*        m     int ベクトルの個数                                      */
/*                                                                      */
/* 出力   b     double型の配列                                          */
/*              行列 b は 行列 a の転置行列                             */
/*                                                                      */
/*        c     double型の配列                                          */
/*          (正規化された行列 a の逆行列)  x (正規化された行列 a)       */
/*                                                                      */
/*                                                                      */

int ClassCorelationMatrix(a, b, c, l, m)
     double a[];
     double b[], c[];
     int    l, m;
/* each vector has l dimensions and there are m vectors. */
{
  int i, j, k;
  double  tmp, tmpp;

  for (i = 0; i < m; i++) {
    k = i * l;
    tmp = 0;
    for (j = k; j < l + k; j++)
      tmp += a[j] * a[j];
    tmpp = sqrt(tmp);
    if (tmp == 0.0) {
      fprintf(stderr, "%dth vector is a zero vector.\n", i);
```



```
      return(-1);
    }
    for (j = k; j < l + k; j++)
      a[j] /=  tmpp;
  } /* これですべてのベクトルが長さ1に正規化された  */
  mtral(a, b, l, m, m, l); /* a の対称行列を bとする */
  mmul(b, a, c,  m, l, l, l, m, l);
    /* 行列 c のトレース は mに等しい */
  for (i = 0; i < l * l; i++)
    c[i] /= m;

  return(0);
}
```

```
 81
 82 /*                                                                    */
 83 /* 固有値とそれに対応した固有ベクトルを入力し，固有値の大きい順に    */
 84 /* 固有ベクトルとともに並べ直す.                                      */
 85 /*                                                                    */
 86 /* 入出力    EugenValues[]  double型の配列                            */
 87 /*                          EugenValues[1][1] を仮定.                 */
 88 /*                          対角成分に固有値が格納されている.         */
 89 /*                          演算後は，固有値の大きい順に列ベクトルを並べ直し，*/
 90 /*                          EVal[j][0]に固有値を格納する.             */
 91 /*                              注意:演算前はこの配列（行列）は対称.  */
 92 /*                                   演算後は非対称.                  */
 93 /*                                                                    */
 94 /*          EugenVectors[]  doubei型の配列                            */
 95 /*                          EugenVectors[1][1] を仮定.                */
 96 /*                          固有値EugenValues[j][j]に対応する固有ベクトルは，*/
 97 /*                  EugenVectors[j]に格納される.                      */
 98 /*                                                                    */
 99 /*        EVec[0][0] … … Evec[0][1-1]    <= これが EVal[0][0] に対応  */
100 /*            :              :                    （演算前の）        */
101 /*            :              :                                        */
102 /*        EVec[j][0] … … EVec[j][1-1]    <= これが EVal[j][j] に対応  */
103 /*            :              :                    （演算前の）        */
104 /*        EVec[1-1][0] …  EVec[1-1][1-1] <= これが EVal[1-1][1-1] に対応*/
105 /*                                                （演算前の）        */
106 /*                                                                    */
107 /* 入力      1     固有値数，固有ベクトル数，または固有ベクトルの次元 */
108 /*                                                                    */
109 /* 出力      EValue 構造体 EvIn のポインタへの配列                     */
110 /*                          もちろん 1個.                             */
111 /*                                                                    */
112
113 /*   構造体 EvIn                                                      */
114 /*                                                                    */
115 /*         固有値とその固有値がソートする前に何番目の固有値であったか */
116 /*         を記憶するため構造体                                       */
117 struct EvIn {
118   double value;
119   int number;
120 };
121
122 void SortEugenValues(EugenValues, EugenVectors, l)
123      double EugenValues[];
124      double EugenVectors[];
125      int l;
126 {
127   struct EvIn **EValue;
128
129   int p, q;
130
131   EValue = (struct EvIn **) malloc(sizeof(struct EvIn *) * l);
132
133   for (p = 0, q = 0; p < l * l ; p += (l+1), q++) {
134     *(EValue+q) = (struct EvIn *) malloc(sizeof(struct EvIn));
135     (*(EValue+q))->value = EugenValues[p];
136     (*(EValue+q))->number = q;
137   }
138
139   (void) SortEvIn(EValue, l);
140   (void) SwapEugenValues(EValue, EugenValues, l);
141   (void) SwapEugenVectors(EValue, EugenVectors, l);
142
143 /* これ以降は，メモリの解放してるだけ */
144
145   free((struct EvIn **) EValue);
146
```



```
147   for (q = 0; q < l; q++)
148     free((struct EvIn *) (*(EValue+q)));
149 }
150
151 void SortEvIn(Value, n)
152 struct EvIn *Value[];
153 int n;
154 {
155   int i, j;
156
157   for (i = 0; i < n; i++)
158     for (j = i + 1; j < n; j++)
159       if (Value[i]->value < Value[j]->value)
160         (void) SwapEvIn(Value[i], Value[j]);
161 }
162
163 void SwapEvIn(EvInA, EvInB)
164 struct EvIn *EvInA, *EvInB;
165 {
166   struct EvIn *swap;
167
168   swap = (struct EvIn *) malloc(sizeof(struct EvIn));
169   swap->value = EvInA->value;
170   swap->number = EvInA->number;
171   EvInA->value = EvInB->value;
172   EvInA->number = EvInB->number;
173   EvInB->value = swap->value;
174   EvInB->number = swap->number;
175   free((struct EvIn *)swap);
176 }
177
178
179 void SwapEugenValues(Value, Values, l)
180      struct EvIn *Value[];
181      double Values[];
182      int l;
183 {
184   int i, j,  k;
185   double *hate, *top_hate;
186
187   top_hate = (double *) malloc(sizeof(double) * l * l);
188   for (i = 0 , hate = top_hate ; i < l * l; i++, hate++) {
189       *hate = Values[i];
190   }
191   for (i = 0; i < l; i++) {
192     k = Value[i]->number;
193     hate = top_hate;
194     for (j = 0; j < l; j++)
195       Values[i * l + j] = hate[k * l +j];
196     Values[i * l] = Values[i * l + k];
197   }
198   free((double *)top_hate);
199 }
200
201
202 void SwapEugenVectors(Value, Vectors, l)
203      struct EvIn *Value[];
204      double Vectors[];
205      int l;
206 {
207   int i, j,  k;
208   double *hate, *top_hate;
209
210   top_hate = (double *) malloc(sizeof(double) * l * l);
211   for (i = 0 , hate = top_hate ; i < l * l; i++, hate++) {
212     *hate = Vectors[i];
```

```
213   }
214   for (i = 0; i < l; i++) {
215     k = Value[i]->number;
216     hate = top_hate;
217     for (j = 0; j < l; j++)
218       Vectors[i * l + j] = hate[k * l +j];
219   }
220   free((double *)top_hate);
221 }
```



```
222
223
224 /*                                                              */
225 /* 忠実度に対応した部分空間の次元数を返す                       */
226 /*                                                              */
227 /* 入力 EugenValues double型の配列                              */
228 /*                        EugenValues[l]を仮定．ソートされた固有値を格納 */
229 /*                        注意：SortEugenValuesで使った同名の配列とは違う */
230 /*                                                              */
231 /*      l           int  固有値数，あるいは生データのベクトルの次元数.  */
232 /*      fidellty    double  忠実度  (SMPR の p.73を参照)         */
233 /* 関数値   int     忠実度に対応した類部分空間の次元数           */
234 /*                                                              */
235
236 int DimPartialSpace(EugenValues, l, fidelity)
237     double EugenValues[];
238     int    l;
239     double fidelity;
240 {
241   int i;
242   double plus;
243
244   plus = 0;
245
246   for (i = 0; i < l; i++) {
247     plus += EugenValues[i];
248     if (plus >= fidelity)
249       return(i+1);
250   }
251
252   return -1;
253 }
```

```
254
255
256
257 /*                                                              */
258 /*  構造体 Class に次元数，固有値，基底ベクトルを格納.          */
259 /*                                                              */
260 /*                                                              */
261
262 struct Class *SetClass(id, dimension, l, values, vectors)
263      int  id, dimension, l;
264      double values[];
265      double vectors[];
266 {
267   int i;
268   struct Class *class;
269   double *tmp;
270
271   class = (struct Class *) malloc(sizeof(struct Class));
272
273   class->ClassId = id;
274   class->DimClass = dimension;
275   class->ClassVectorDim = l;
276   class->ClassEugenValues = (double *) malloc(sizeof(double) * dimension);
277   class->ClassBasis = (double *) malloc(sizeof(double) * dimension * l);
278   tmp = class->ClassEugenValues;
279   for (i = 0; i < dimension; i++)
280     *tmp++ = values[i];
281   tmp = class->ClassBasis;
282   for (i = 0; i < dimension * l; i++)
283     *tmp++ = vectors[i];
284   return((struct Class *) class);
285 }
286
287 void ReleaseClass(class)
288      struct Class *class;
289 {
290   free((double *) class->ClassEugenValues);
291   free((double *) class->ClassBasis);
292   free((struct Class *) class);
293 }
294
295 int BasicClassification(classes, vector, discriminant,  k, l, threshold)
296      struct Class *classes[];
297      double vector[], discriminant[], threshold;
298      int k, l;
299 {
300   int i, j, m, n, x;
301
302   double *tmp0, *tmp1, *tmp2, *top,  plus, max, plusplus, euplusplus;
303   double lev; /* もっとも大きな固有値 */
304
305   max = threshold;
306   x = -1;
307
308   tmp2 = (double *) malloc(sizeof(double) * l);
309   for (i = 0, top = tmp2 ;i < l; i++)
310     *(tmp2++) = *(vector++);
311   tmp2 = top;
312   OrthoVector(tmp2, l);
313
314   for (i = 0; i < k; i++) {
315     tmp0 = classes[i]->ClassBasis;
316     tmp1 = classes[i]->ClassEugenValues;
317     m = classes[i]->DimClass;
318     plus = 0;
319     plusplus = 0;
```



```
320     euplusplus = 0;
321
322     for (n = 0, lev = *tmp1; n < m ; n++, tmp1++) {
323       plus = 0;
324 #ifdef MOREDEBUG_EX
325       for (j = 0, tmp2 = top; j < l; j++, tmp0++, tmp2++) {
326         plus += (*tmp0) * (*tmp2);
327         printf("% lf\t% lf\t% lf\n", *(tmp0), *(tmp2), plus);
328       }
329 #else
330       for (j = 0, tmp2 = top; j < l; j++, tmp0++, tmp2++)
331         plus += (*tmp0) * (*tmp2);
332 #endif
333       plusplus = plus * plus;
334       euplusplus += plusplus;
335 #ifdef MOREDEBUG_EX
336       printf("plusplus = % lf\t euplusplus = % lf\n", plusplus, euplusplus);
337 #endif
338     }
339
340 #ifdef MOREDEBUG
341     printf("class %d euplusplus = %lf\n", i, euplusplus);
342     getchar();
343 #endif
344     discriminant[i] = euplusplus;
345     if (euplusplus > max) {
346       max = euplusplus;
347       x = i;
348     }
349   }
350 #ifdef DEBUG
351   for (i = 0; i < k; i++)
352     fprintf(stdout, "discriminant function value of %d:  % lf\n",
353             i, discriminant[i]);
354 #endif
355   return(x);
356 }
357
358 int MulSimMethod(classes, vector, discriminant,  k, l, threshold)
359      struct Class *classes[];
360      double vector[], discriminant[], threshold;
361      int k, l;
362 {
363   int i, j, m, n, x;
364
365   double *tmp0, *tmp1, *tmp2, *top,  plus, max, plusplus, euplusplus;
366   double lev; /* もっとも大きな固有値 */
367
368   max = threshold;
369   x = -1;
370
371   tmp2 = (double *) malloc(sizeof(double) * l);
372   for (i = 0, top = tmp2 ;i < l; i++)
373     *(tmp2++) = *(vector++);
374   tmp2 = top;
375   OrthoVector(tmp2, l);
376
377   for (i = 0; i < k; i++) {
378     tmp0 = classes[i]->ClassBasis;
379     tmp1 = classes[i]->ClassEugenValues;
380     m = classes[i]->DimClass;
381     plus = 0;
382     plusplus = 0;
383     euplusplus = 0;
384
385     for (n = 0, lev = *tmp1; n < m ; n++, tmp1++) {
```

```
386       plus = 0;
387 #ifdef MOREDEBUG
388       for (j = 0, tmp2 = top; j < l; j++, tmp0++, tmp2++) {
389         plus += (*tmp0) * (*tmp2);
390         printf("*tmp0++ =\% lf\n", *(tmp0));
391       }
392 #else
393       for (j = 0, tmp2 = top; j < l; j++, tmp0++, tmp2++)
394         plus += (*tmp0) * (*tmp2);
395 #endif
396       plusplus = plus * plus;
397       euplusplus += ((*tmpl)/lev) * plusplus;
398     }
399
400
401 #ifdef MOREDEBUG
402     printf("class %d euplusplus = %lf\n", i, euplusplus);
403     getchar();
404 #endif
405     discriminant[i] = euplusplus;
406     if (euplusplus > max) {
407       max = euplusplus;
408       x = i;
409     }
410   }
411 #ifdef DEBUG
412   for (i = 0; i < k; i++)
413     fprintf(stdout, "discriminant function value of %d:  % lf\n",
414             i, discriminant[i]);
415 #endif
416   return(x);
417 }
418
```

```
/* shs.c */

#include <stdio.h>
#include <math.h>
#include <device.h>
#include "subspace.h"
#include "claw.h"
#ifndef TRUE
#define TRUE 1
#endif


extern void mprintf();
extern int  mtra2();
extern int jacobi();
extern void ClassCorelationMatrix();
extern void SortEugenValues();
extern struct Calss *SetClass();
extern void ReleaseClass();
extern int MulSimMethod();
extern int OrthoVector();

extern double timeMatrix[TIMES];
extern double vpMatrix[TIMES][3];
extern double apMatrix[TIMES][3];
extern double vtMatrix[TIMES][3];
extern double viMatrix[TIMES][3];
extern double vcMatrix[TIMES][3];
extern double vrMatrix[TIMES][3];
extern double vsMatrix[TIMES][3];
extern double avtMatrix[TIMES][3];
extern double aviMatrix[TIMES][3];
extern double avcMatrix[TIMES][3];
extern double avrMatrix[TIMES][3];
extern double avsMatrix[TIMES][3];

extern void GetHandData();
extern char *DataTop();
extern void LastData();
extern int shmid;

void  GetMatrixData();
void GetVector();
void LearningStage();
void RecognizingStage();
void RealTimeRecognize();
void EndHand();
int SaveClassData();
int LoadClassData();
void SaveClassDataSub();
void LoadClassDataSub();

FILE *datafile;
char *head;
int l = 6, m;
struct Class *class[10];

void LearningStage()
{
  char dname[128];

  int i, j, k;
  int p, q, nr;

  int dimension[10];
  double a[20][6];
```



```
  double c[6][6], b[6][20], v[6][6];
  double s[10][10][6][6], ss[10][6][6],  oss[10][6][6];
  double pf[6][6], mf[6][6];
  double funv[10], max, tmp, eps;
  double ev[6];
  double vec[6], ovec[6], vecvec[6][6];
  char hoge[16];
  l = 6; m =20;
  for (p = 0; p < 10; p++) {
    sprintf(dname, "/usr1/onishi/koubou/crab/claw/scale/kaku/z%d", p);
    if ((datafile = fopen(dname, "r")) == NULL) {
      fprintf(stderr, "Can't open file :%s\n", dname);
      exit(0);
    }

    GetMatrixData(a, p);
    fclose(datafile);

    (void) ClassCorelationMatrix(a, b, c, l, m);

    for (i = 0; i < l; i++)
      for (j = 0; j < l; j++)
        ss[p][i][j] = c[i][j];

    eps = 1.0e-6;
    nr = 50;

    i = (int) jacobi(c, v, l, l, &nr, eps);

    (int) mtra2(v, l, l);

    (void) SortEugenValues(c, v, l);

    for (j = 0; j < l; j++)
      ev[j] = c[j][0];
    k = (int) DimPartialSpace(ev , l, .97);
    dimension[p] = k;
    printf("Dimension of Partial Space is %d.\n", k);
    class[p] = (struct Class *) SetClass(p, k, l, ev, v);
  }

  while(1) {
    for (p = 0; p < 10; p++)
      for (k = 0; k < 10; k++)
        msub(s[p][k], s[p][k], s[p][k], 6, 6, 6, 6, 6, 6);
    for (p = 0; p < 10; p++) {
      for (i = 0; i < 6; i++)
        for (j = 0; j < 6; j++)
          c[i][j] = ss[p][i][j];

      i = (int) jacobi(c, v, l, l, &nr, eps);

      (int) mtra2(v, l, l);

      (void) SortEugenValues(c, v, l);

      for (j = 0; j < l; j++)
        ev[j] = c[j][0];
/*
      k = (int) DimPartialSpace(ev , l, .95);
      dimension[p] = k;
      printf("Dimension of Partial Space is %d.\n", k);
*/
      class[p] = (struct Class *) SetClass(p, dimension[p], l, ev, v);

      sprintf(dname, "/usr1/onishi/koubou/crab/claw/scale/kaku/z%d", p);
```

```
133       if ((datafile = fopen(dname, "r")) == NULL) {
134         fprintf(stderr, "Can't open file :%s\n", dname);
135         exit(0);
136       }
137       for (q = 0; q < 20; q++) {
138         GetVector(vec);
139         k = BasicClassification(class, vec, funv, 10, 1, .5);
140
141         if (p != k) {
142           fprintf(stdout, "%d %d %lf %lf %lf %lf %lf %lf \n",
143                   p, k, vec[0], vec[1], vec[2], vec[3], vec[4], vec[5]);
144         }
145         for(i = 0, max = (p==0)? 1:0, tmp = 0; i < 10; i++)
146           if (i != p)
147             if (funv[i] > tmp) {
148               tmp = funv[i];
149               max = i;
150             }
151         if (k != p) {
152           for(i = 0; i < 6; i++)
153             ovec[i] = vec[i];
154           OrthoVector(ovec, 6);
155           mmul(ovec, ovec, vecvec, 1, 6, 6, 6, 1, 6);
156           madd(s[p][k], vecvec, s[p][k], 6, 6, 6, 6, 6, 6);
157         }
158       }
159       fclose(datafile);
160     }
161     printf("Continue learning: ");
162     scanf("%s", hoge);
163     if (hoge[0] != 'y'&& hoge[0] != 'Y')
164       break;
165
166     msub(pf, pf, pf, 6, 6, 6, 6, 6, 6);
167     msub(mf, mf, mf, 6, 6, 6, 6, 6, 6);
168     for (i = 0; i < 10; i++) {
169       for (j = 0; j < 10; j++) {
170         madd(pf, s[i][j], pf, 6, 6, 6, 6, 6, 6);
171         madd(mf, s[i][j], mf, 6, 6, 6, 6, 6, 6);
172       }
173
174       mti(.1, pf, pf, 6, 6);
175       mti(.1, mf, mf, 6, 6);
176       madd(ss[i], pf, ss[i], 6, 6, 6, 6, 6, 6);
177       madd(ss[i], mf, ss[i], 6, 6, 6, 6, 6, 6);
178 /*      mprintf(ss[i], 6, 6, 6); */
179     }
180
181     for (p = 0; p < 10; p++)
182       for (i = 0; i < 6; i++)
183         for (j = 0; j < 6; j++)
184           oss[p][i][j] = ss[p][i][j];
185 /*    printf("Continue learning: ");
186     scanf("%s", hoge);
187     if (hoge[0] != 'y'&& hoge[0] != 'Y')
188       break;
189 */
190   }
191 }
192
193 int SaveClassData(s)
194      char s[];
195 {
196   int i;
197
198   if ((datafile = fopen(s, "w")) == NULL)
```



```
199     return -1;
200   for (i = 0; i < 10; i++)
201     SaveClassDataSub(i);
202   fclose(datafile);
203   (void) EndHand();
204   return 0;
205 }
206
207 void SaveClassDataSub(p)
208 {
209   int i;
210   double *x;
211
212 /*  fprintf(datafile, "#%d %s\n", p, class[p]->ClassName); */
213   fprintf(datafile, "%d\n", class[p]->ClassId);
214   fprintf(datafile, "%d\n", class[p]->DimClass);
215   fprintf(datafile, "%d\n", class[p]->ClassVectorDim);
216
217   for (i = 0, x = (double *) class[p]->ClassEugenValues;
218        i < class[p]->DimClass; i++)
219     fprintf(datafile, "%lf ", *x++);
220   fprintf(datafile, "\n");
221   for (i = 0, x = (double *) class[p]->ClassBasis;
222        i < class[p]->DimClass*l; i++)
223     fprintf(datafile, "%lf ", *x++);
224   fprintf(datafile, "\n");
225   fflush(datafile);
226 }
227
228 int LoadClassData(s)
229      char s[];
230 {
231   int i;
232
233   if ((datafile = fopen(s, "r")) == NULL)
234     return -1;
235   for (i = 0; i < 10; i++)
236     LoadClassDataSub(i);
237   fclose(datafile);
238   return(0);
239 }
240
241 void LoadClassDataSub(i)
242      int i;
243 {
244   int id, dim, vdim;
245   int j;
246   double *tmp, y;
247
248   class[i] = (struct Class *) malloc(sizeof(struct Class));
249   fscanf(datafile, "%d\n", &id);
250   class[i]->ClassId = id;
251   fscanf(datafile, "%d\n", &dim);
252   class[i]->DimClass = dim;
253   fscanf(datafile, "%d\n", &vdim);
254   class[i]->ClassVectorDim = vdim;
255   class[i]->ClassEugenValues = (double *) malloc(sizeof(double) * dim);
256   tmp = (double *)class[i]->ClassEugenValues;
257   for (j = 0; j < dim; j++, tmp++) {
258     fscanf(datafile, "%lf", &y);
259     *tmp = y;
260     printf("%lf %lf\n", y, *tmp);
261   }
262   class[i]->ClassBasis = (double *) malloc(sizeof(double) * dim * vdim);
263   tmp = (double *)class[i]->ClassBasis;
264   for (j = 0; j < (dim*vdim); j++, tmp++) {
```

```
265     fscanf(datafile, "%lf", &y);
266     *tmp = y;
267     printf("%lf %lf\n", y, *tmp);
268   }
269 }
270
271 void RecognizingStage()
272 {
273   int p, q, k;
274   char dname[128];
275   double funv[10];
276   register double vec[6];
277
278   for (p = 0; p < 10; p++) {
279     sprintf(dname, "/usr1/onishi/koubou/crab/scale/kakuz%d", p);
280     if ((datafile = fopen(dname, "r")) == NULL) {
281       fprintf(stderr, "Can't open file :%s\n", dname);
282       exit(0);
283     }
284     fprintf(stdout, "Actual Class : %d\n", p);
285     for (q = 0; q < 50; q++) {
286       if (q == 20) printf("\n");
287       GetVector(vec);
288       k = BasicClassification(class, vec, funv, 10, 1, .7);
289       if (p != k) {
290         fprintf(stdout, "%d %d %lf %lf %lf %lf %lf %lf \n",
291                 p, k, vec[0], vec[1], vec[2], vec[3], vec[4], vec[5]);
292       }
293     }
294     fclose(datafile);
295   }
296 }
297
298
299 void RealTimeRecognize()
300 {
301   int i;
302   double t, funv[10];
303   register double vec[6];
304   int k[20];
305
306   GetHandData(head);
307   if (t != timeMatrix[0]) {
308     vec[0] = vtMatrix[0][0];
309     vec[1] = vtMatrix[0][1];
310     vec[2] = vtMatrix[0][2];
311     vec[3] = viMatrix[0][0];
312     vec[4] = viMatrix[0][1];
313     vec[5] = viMatrix[0][2];
314
315     k[0] = BasicClassification(class, vec, funv, 10, 1, .5);
316     /*    fprintf(stdout, "%lf \n", timeMatrix[0]);
317           fprintf(stdout, "%d %lf %lf %lf %lf %lf %lf \n",
318             k[0], vec[0], vec[1], vec[2], vec[3], vec[4], vec[5]);
319     for (i = 19; i > 0; i--)
320       k[i] = k[i-1];
321     fprintf(stdout, "% d % d % d % d % d % d \n", k[0], k[1], k[2], k[3],k[4]);
322     fprintf(stdout, "% d % d % d % d % d % d \n", k[5], k[6], k[7], k[8],k[9]);
323     fprintf(stdout, "% d % d % d % d % d % d \n", k[10], k[11], k[12], k[13],k[1
4]);
324     fprintf(stdout, "% d % d % d % d % d % d \n", k[15], k[16], k[17], k[18],k[1
9]);
325     fprintf(stdout, "\n"); */
326
327     t = timeMatrix[0];
328     return k[0];
```



```
329   }
330   else
331     return k[1];
332 }
333
334
335 #include <sys/time.h>
336
337 void GetMatrixData(a, p)
338      double a[20][6];
339      int p;
340 {
341   int i, j, flg, lnum[20];
342   long seed;
343   char line[256];
344   double t, thumbx, thumby, thumbz, indexx, indexy, indexz;
345   struct timeval tp;
346   struct timezone tzp;
347
348   for(i = 0; i < 20; i++) {
349     flg = 0;
350       gettimeofday(&tp, &tzp);
351       seed = tp.tv_usec;
352       srandom(seed);
353       lnum[i] = (int) random()%50;
354       for (j = 0; j < i; j++)
355         while (lnum[i] == lnum[j])
356           lnum[i] += 1;
357
358     fseek(datafile, 0L, SEEK_CUR);
359     j = 0;
360     while(j++<=lnum[i])
361       fgets(line, 256, datafile);
362     sscanf(line, "%lf %lf %lf %lf %lf %lf %lf",
363             &t, &thumbx, &thumby, &thumbz, &indexx, &indexy, &indexz);
364     a[i][0] = thumbx;
365     a[i][1] = thumby;
366     a[i][2] = thumbz;
367     a[i][3] = indexx;
368     a[i][4] = indexy;
369     a[i][5] = indexz;
370   }
371 }
372
373 void GetVector(vector)
374      double vector[];
375 {
376   char line[256];
377   double t, thumbx, thumby, thumbz, indexx, indexy, indexz;
378
379   fgets(line, 256, datafile);
380     sscanf(line, "%lf %lf %lf %lf %lf %lf %lf",
381             &t, &thumbx, &thumby, &thumbz, &indexx, &indexy, &indexz);
382   vector[0] = thumbx;
383   vector[1] = thumby;
384   vector[2] = thumbz;
385   vector[3] = indexx;
386   vector[4] = indexy;
387   vector[5] = indexz;
388 }
389
390 void EndHand()
391 {
392   int p;
393   for (p = 0; p < 10; p++)
394     (void) ReleaseClass(class[p]);
```

```
395 }
396
```

```
 1 /* r_learn.c */
 2 #include <stdio.h>
 3 #include <string.h>
 4 #include <math.h>
 5 #include <sys/types.h>
 6 #include <fcntl.h>
 7 #include "network.h"
 8
 9 char name[40];
10
11 FILE *fp;
12
13 struct unit *r_pu[R_TOTAL];
14
15 double r_tampon1[R_NUM_CON1], r_tampon2[R_NUM_CON2];
16
17 main()
18 {
19      r_create_processing_units(r_pu, r_tampon1, r_tampon2);
20      r_create_in_out_links(r_pu);
21      r_learn(r_pu);
22      r_save_weights(r_pu);
23 }
24
```

```
 1 /* s_learn.c */
 2 #include <stdio.h>
 3 #include <string.h>
 4 #include <math.h>
 5 #include <sys/types.h>
 6 #include <fcntl.h>
 7 #include "network.h"
 8
 9 char name[40];
10
11 FILE *fp;
12
13 struct unit *s_pu[S_TOTAL];
14
15 double s_tampon1[S_NUM_CON1], s_tampon2[S_NUM_CON2];
16
17 main()
18 {
19      s_create_processing_units(s_pu, s_tampon1, s_tampon2);
20      s_create_in_out_links(s_pu, s_tampon1, s_tampon2);
21      s_learn(s_pu, s_tampon1, s_tampon2);
22      s_save_weights(s_pu, s_tampon1, s_tampon2);
23 }
24
```

```
 1 /* network.c */
 2 #include "network.h"
 3 #include <math.h>
 4 #include <stdio.h>
 5 #include <ctype.h>
 6
 7 extern double m_out_f(), r_out_f(), s_out_f();
 8 extern double m_delta_f_out(), r_delta_f_out(), s_delta_f_out();
 9 extern double m_delta_f_hid(), r_delta_f_hid(), s_delta_f_hid();
10 int iterations = DEFAULT_ITER;
11 /* double rdm();  */
12
13 /* return a random number between -1.0 and 1.0 */
14 /*
15 double
16 rdm()
17 {
18   return((rand() % 50000 - 25000)/ 25000.0);
19 }
20 */
21
22 struct unit *
23 create_unit(uid, label, output, out_f, delta, delta_f,thd)
24 int  uid;
25 char *label;
26 double   output, delta,thd;
27 double   (*out_f)(), (*delta_f)();
28 {
29   struct unit  *unitptr;
30
31   if (!(unitptr = (struct unit *)malloc(sizeof(struct unit)))) {
32     fprintf(stderr, "create_unit: not enough memory\n");
33     exit(1);
34   }
35   /* initialize unit data */
36   unitptr->uid = uid;
37   unitptr->label = label;
38   unitptr->output = output;
39   unitptr->unit_out_f = out_f;  /* ptr to output function */
40   unitptr->delta = delta;
41   unitptr->unit_delta_f = delta_f;
42   unitptr->threshold = thd;
43   return (unitptr);
44 }
45
46 struct link *
47 create_link(start_inlist, to_uid, start_outlist, from_uid, label, wt, data)
48 struct link *start_inlist, *start_outlist;
49 int     to_uid, from_uid;
50 char *  label;
51 double  wt, data;
52 {
53   struct link  *linkptr;
54
55   if (!(linkptr = (struct link *)malloc(sizeof(struct link)))) {
56     fprintf(stderr, "create_link: not enough memory\n");
57     exit(1);
58   }
59   /* initialize link data */
60   linkptr->label = label;
61   linkptr->from_unit = from_uid;
62   linkptr->to_unit = to_uid;
63   linkptr->weight = wt;
64   linkptr->data = data;
65   linkptr->next_inlink = start_inlist;
66   linkptr->next_outlink = start_outlist;
```



```
67   return(linkptr);
68 }
69
70 char
71 get_command(s)
72 char    *s;
73 {
74   char  command[BUFSIZE];
75
76   fputs(s, stdout);
77   fflush(stdin); fflush(stdout);
78   fgets(command, BUFSIZE, stdin);
79   return((command[0]));          /* return 1st letter of command */
80 }
```

```
 1 /* matrix.c */
 2 #include <stdio.h>
 3 #include <math.h>
 4
 5 int mtral(), mtra2(), jacoji(), mmul(), minver(), madd(), mti();
 6 void mprintf();
 7
 8
 9 /* 行列の転置 その１                                                    */
10 /*                                                                      */
11 /* 入力  a[]      double型の配列                                        */
12 /*                        a[m][la]を仮定                                */
13 /*       la, lb   int    配列 a, b の第２添字                          */
14 /*                        la >= n, lb >= m                              */
15 /*       m, n     int    配列 a, b の内，演算対象となる行列数          */
16 /*                                                                      */
17 /* 出力  b[]      double型の配列                                        */
18 /*                        転置した結果の行列b[n][lb]                    */
19
20 int mtral(a, b, la, lb, m, n)
21      double a[], b[];
22      int la, lb, m, n;
23 {
24   int i, j, k;
25
26   if (m < 1 || n < 1)
27     return (-1);
28
29   for (i = 0; i < m; i++) {
30     k = i * la;
31     for (j = 0; j < n; j++)
32       b[j * lb + i] = a[k+j];
33   }
34   return(0);
35 }
```



```
36
37
38
39 /* 行列の転置 その２                                                    */
40 /*                                                                      */
41 /* 入力  a[]      double型の配列                                        */
42 /*                        a[m][l]を仮定                                 */
43 /*       l        int    配列 a の第２添字                             */
44 /*                        l >= m                                        */
45 /*       m        int    配列 a の内，演算対象となる行列数             */
46 /*                                                                      */
47 /* 出力  a[]      double型の配列                                        */
48 /*                        転置した結果の行列 a[l][m]                    */
49
50 int mtra2(a, l, m)
51      double a[];
52      int l, m;
53 {
54   int i, j, k, mml, ipl, s, t;
55   double w;
56
57   if (m < 2)
58     return(-1);
59   mml  = m - 1;
60   for (i = 0; i < mml; i++) {
61     k = i * l;
62     ipl = i + 1;
63     for ( j = ipl; j < m; j++) {
64       s = k + j;
65       t = j * l + i;
66       w = a[s];
67       a[s] = a[t];
68       a[t] = w;
69     }
70   }
71   return(0);
72 }
```

```
 73
 74
 75
 76 /* 行列の乗算                                                    */
 77 /*                                                              */
 78 /* 入力 a[], b[]    double型の配列                              */
 79 /*                        a[m][la], b[k][lb] を仮定             */
 80 /*      la, lb, lc  int   配列 a, b, c の第２添字の値          */
 81 /*      m, n, k     int   配列 a, b, c の内，演算の対象となる行列数 */
 82 /*                                                              */
 83 /* 出力 c[]         double型の配列                              */
 84 /*                        結果の行列. c[m][lc] を仮定           */
 85
 86 int mmul(a, b, c, la, lb, lc, m, n, k)
 87      double a[], b[], c[];
 88      int la, lb, lc, m, n, k;
 89 {
 90   int i, j, s, t, u;
 91   double w;
 92
 93   if (m < 1 || n < 1 || k < 1) return(-1);
 94
 95   for (i = 0; i < m; i++) {
 96     t = i * la;
 97     u = i * lc;
 98     for ( j = 0; j < k; j++) {
 99       w = .0;
100       for (s = 0; s < n; s++)
101         w += a[t + s] * b[s * lb + j];
102       c[u+j] = w;
103     }
104   }
105   return(0);
106 }
```



```
107
108
109
110 /* Jacobi法により，実対称行列の固有値と固有ベクトルを求める          */
111 /*                                                              */
112 /* 入出力   a[]   double型の配列                                */
113 /*                        配列 a[m][l]で行列を格納する          */
114 /*                        演算後は，対角要素 a[i][i](i = 0,…,m-1)に */
115 /*                        固有値が得られる                      */
116 /*          nr    int型へのポインタ                            */
117 /*                        回転を行なう最大数を与える．          */
118 /*                        演算後は回転数が得られる．            */
119 /* 入力     l     int     配列 a の第２添字                     */
120 /*          m     int                                           */
121 /*          eps   double  収束判定値                            */
122 /*                                                              */
123 /* 出力     v     double型の配列                               */
124 /*                        結果の固有ベクトルが配列 v[m][l]で得られる */
125 /*                        j番目の固有値に対する固有ベクトルは，v[i][j] */
126 /*                        (i = 0,…,m-1)である．                 */
127 /*                                                              */
128
129 int jacobi(a, v, l, m, nr, eps)
130      double a[], v[], eps;
131      int l, m, *nr;
132 {
133   int  n, mml, ipl, i, j, r, c, kl, k2, s1, s2, s3, s4, s5, s6;
134   double wmax, w, a1, a2, a3, a4, a5,  a6, t1, t2, t3, t4,  ta,  si, co;
135   extern double fabs(), sqrt();
136
137   if ( m < 2 || *nr < 1 || eps <= .0) return(-1);
138
139   n  = 0;
140   mml = m - 1;
141   for(i = 0; i < mml ; i++) {
142     kl = i * l;
143     ipl = i + l;
144     for (j = ipl; j < m; j++) ;
145 /*       if (a[kl+j] != a[j * l+i])  */
146 /*       return(-1); */
147     }
148
149   for ( i = 0; i < m; i++) {
150     kl= i * l;
151     for (j  = 0; j < m; j++)
152       if (i != j) v[kl+j] = 0.0;
153     v[kl+i] = 1.0;
154   }
155   while(1) {
156     wmax = 0.0;
157     for (i = 0; i < m; i++) {
158        kl = i * l;
159        ipl = i + l;
160        for (j = ipl; j < m; j++) {
161          w = fabs(a[kl+j]);
162          if (w > wmax) {
163            wmax  = w;
164            r = i;
165            c = j;
166          }
167        }
168      }
169     if (wmax <= eps) {
170       *nr = n;
171       return(0);
172     }
```

```
173     if (n >= *nr) {
174       *nr = n;
175       return(1);
176     }
177     k1 = r * l;
178     k2 = c * l;
179     s1 = k1 + r;
180     s2 = k2 + c;
181     s3 = k1 + c;
182     n++;
183     a1 = a[s1];
184     a2 = a[s2];
185     a3 = a[s3];
186     t1 = fabs(a1-a2);
187     t2 = t1 * t1;
188     t3 = 4.0 * a3 * a3;
189     ta = 2.0 * a3 / (t1 + sqrt(t2 + t3));
190     if (a1 < a2) ta = -ta;
191     t4 = ta * ta + 1.0;
192     co = sqrt(1.0 / t4);
193     si = ta * co;
194     for (i = 0; i < m; i++) {
195       s4 = i * l;
196       s5 = s4 + r;
197       s6 = s4 + c;
198       w = v[s5];
199       v[s5] = w * co + v[s6] * si;
200       v[s6] = -w * si + v[s6] * co;
201       if (i != r && i != c) {
202         w = a[s5];
203         a[s5] = w * co + a[s6] * si;
204         a[s6] = -w * si + a[s6] * co;
205         a[k1+i] = a[s5];
206         a[k2+i] = a[s6];
207       }
208     }
209     a[s1] = a1 * co * co + a2 * si *si + 2.0 * a3 * co * si;
210     a[s2] = a1 + a2 - a[s1];
211     a[s3] = 0.0;
212     a[k2+r] = 0.0;
213   }
214 }
```



```
215
216
217
218 /* 正方行列の逆行列と行列式の値を Gauss-Jordan法により求める.                */
219 /* 掃き出す際, 軸(Pivot)として列要素の絶対値最大を選択するために            */
220 /* 行入れ換えを行なう.                                                      */
221 /*                                                                          */
222 /* 入出力    a[]    double型の配列. a[m][l]を仮定.                          */
223 /*                                    演算後は逆行列が得られる              */
224 /*                                                                          */
225 /*                                                                          */
226 /*                  a[0][0] ....  a[0][m-1]                                 */
227 /*                  a[1][0] ....  a[1][m-1]                                 */
228 /*                     :             :                                      */
229 /*                     :             :                                      */
230 /*                  a[l-1][0] ... a[l-1][m-1]                               */
231 /*                                                                          */
232 /*                  この行列の内, m列(a[m-1][?])までが演算される            */
233 /*                                                                          */
234 /* 入力      l      int ,                 配列 aの第2添字                  */
235 /*           m      int ,                 配列 aの内, 演算対象となる行数   */
236 /*           eps    double型へのポインタ  収束判定値                       */
237 /*                                                                          */
238 /* 出力      det    行列式の値                                              */
239 /*                                                                          */
240
241 int minver(a, l, m, eps, det)
242      double a[], eps, *det;
243      int l, m;
244 {
245   int work[500], i, j, k, r, iw, s, t, u, v;
246   double w, wmax, pivot, api, w1;
247   extern double fabs();
248
249   if (m < 2 || m > 500 || eps < .0) return(-1);
250
251   w1 = 1.0;
252
253   for (i = 0; i < m; i++)
254     work[i] = i;
255   for (k = 0; k < m; k++) {
256     wmax = .0;
257     for (i = k; i < m; i++) {
258       w = fabs(a[i*l+k]);
259
260       if (w > wmax) {
261         wmax = w;
262         r = i;
263       }
264     }
265     pivot = a[r*l+k];
266     api = fabs(pivot);
267     if ( api <= eps) {
268       *det = w1;
269       return (1);
270     }
271     w1 *= pivot;
272     u = k * l;
273     v = r * l;
274     if (r != k) {
275       w1 = -w1;
276       iw = work[k];
277       work[k] = work[r];
278       work[r] = iw;
279       for (j = 0; j < m; j++) {
280         s = u + j;
```

```
281         t = v + j;
282         w = a[s];
283         a[s] = a[t];
284         a[t] = w;
285       }
286     }
287     for ( i = 0; i < m; i++)
288       a[u+i] /= pivot;
289
290     for (i = 0; i < m; i++) {
291       if (i != k) {
292         v = i * l;
293         s = v + k;
294         w = a[s];
295         if (w != .0) {
296           for (j = 0; j < m; j++)
297             if (j != k) a[v+j] -= w * a[u+j];
298           a[s] = -w / pivot;
299         }
300       }
301     }
302     a[u+k] = 1.0 / pivot;
303   }
304   for ( i= 0; i < m; i++) {
305     while(1) {
306       k =  work[i];
307       if (k == i) break;
308       iw= work[k];
309       work[k] = work[i];
310       work[i] = iw;
311       for (j = 0;  j < m; j++) {
312         u = j * l;
313         s = u + i;
314         t = u + k;
315         w = a[s];
316         a[s] = a[t];
317         a[t] = w;
318       }
319     }
320   }
321   *det = wl;
322   return(0);
323 }
```



```
324
325
326
327 /* 行列の表示                                                    */
328 /*                                                              */
329 /* 入力     x[]   double型の配列    x[l][m]を仮定               */
330 /*          l     int               行列の列数                  */
331 /*          m     int               行列の行数                  */
332 /*          n     int               何列目までを出力するか      */
333 /* 出力                                                         */
334 /*                                                              */
335 /*          x[0][0]  .....   x[0][n-1]  … x[0][l-1]             */
336 /*          x[1][0]          x[1][n-1]  … x[1][l-1]             */
337 /*             :                :                               */
338 /*             :                :                               */
339 /*          x[m-1][0] ...    x[m-1][n-1] …x[m-1][l-1]           */
340 /*                                  ^ここまで表示               */
341 /*          普通は l = n                                        */
342
343 void mprintf(x, l, m, n)
344      double x[];
345      int l, m, n;
346 {
347   int i, j;
348
349   for (i = 0; i < m; i++) {
350     for (j = 0; j < n; j++)
351       printf(" % 7.4e ", x[i * l + j]);
352     printf("\n");
353   }
354   printf("\n");
355 }
```

```
356
357
358 /* ベクトルを正規化する */
359
360 int OrthoVector(vector, dimension)
361      double vector[];
362      int dimension;
363 {
364   int i;
365   double sumsum, sqsumsum;
366
367   sumsum = 0;
368
369   for (i = 0; i < dimension; i++)
370     sumsum += (vector[i] * vector[i]);
371   if (sumsum == 0) {
372     fprintf(stderr, "zero vector\n");
373     return(-1);
374   }
375
376   sqsumsum = sqrt(sumsum);
377
378   for (i = 0; i < dimension; i++)
379     vector[i] /= sqsumsum;
380   return(dimension);
381 }
382
383 /* 行列の加減算                                                    */
384 /*                                                                 */
385 /* 入力 a[], b[]    double型の配列                                 */
386 /*                        a[m][la], b[k][lb] を仮定                */
387 /*      la, lb, lc  int   配列 a, b, c の第2添字の値                */
388 /*      m, n, k     int   配列 a, b, c の内，演算の対象となる行列数 */
389 /*                                                                 */
390 /* 出力 c[]         double型の配列                                 */
391 /*                        結果の行列. c[m][lc] を仮定              */
392
393
394 int madd(a, b, c, la, lb, lc, m, n)
395      double a[], b[], c[];
396      int la, lb, lc, m, n;
397 {
398   int i, j, k, s, t;
399
400   if (m < 1 || n < 1) return(-1);
401   for (i = 0; i < m; i++) {
402     k = i * la;
403     s = i * lb;
404     t = i * lc;
405     for (j = 0; j < n; j++)
406       c[t+j] = a[k+j] + b[s+j];
407   }
408
409   return(0);
410 }
411
412 int msub(a, b, c, la, lb, lc, m, n)
413      double a[], b[], c[];
414      int la, lb, lc, m, n;
415 {
416   int i, j, k, s, t;
417
418   if (m < 1 || n < 1) return(-1);
419   for (i = 0; i < m; i++) {
420     k = i * la;
421     s = i * lb;
```



```
422     t = i * lc;
423     for (j = 0; j < n; j++)
424       c[t+j] = a[k+j] - b[s+j];
425   }
426
427   return(0);
428 }
429
430
431 /* 行列の定数倍
432 /*                                                                 */
433 /* 入力 a[]         double型の配列                                 */
434 /*                        a[m][la]          を仮定                 */
435 /*      la,     lc  int   配列 a       の第2添字の値              */
436 /*      m,      k   int   配列 a,    c の内，演算の対象となる行列数 */
437 /*                                                                 */
438 /* 出力 c[]         double型の配列                                 */
439 /*                        結果の行列. c[m][la] を仮定              */
440
441
442 int mti(alpha, a, c, la, m)
443      double alpha, a[], c[];
444      int la, m;
445 {
446   int i;
447
448     for (i = 0; i < (la*m); i++)
449       c[i] = alpha * a[i];
450   return(0);
451 }
```

```
/* platform.c */
#include <stdio.h>
#include "claw.h"

void GetHandData();
char *DataTop();
void LastData();


double timeMatrix[TIMES];
double vpMatrix[TIMES][3];
double vtMatrix[TIMES][3];
double viMatrix[TIMES][3];
double vcMatrix[TIMES][3];
double vrMatrix[TIMES][3];
double vsMatrix[TIMES][3];
double apMatrix[TIMES][3];
double avtMatrix[TIMES][3];
double aviMatrix[TIMES][3];
double avcMatrix[TIMES][3];
double avrMatrix[TIMES][3];
double avsMatrix[TIMES][3];

int shmid;
char *shmat();

void GetHandData(head)
     char *head;
{
  int i, j;
  double vp[3], vt[3], vi[3], vc[3], vr[3], vs[3];
  double avt[3], avi[3], avc[3], avr[3], avs[3];
  double ap[3];

  double t, old_t;

  sscanf(head,
         "%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %l
f %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf",
         &t, &(vp[0]), &(vp[1]), &(vp[2]), &(ap[0]), &(ap[1]), &(ap[2]),
         &(vt[0]), &(vt[1]), &(vt[2]), &(vi[0]), &(vi[1]), &(vi[2]),
         &(vc[0]), &(vc[1]), &(vc[2]), &(vr[0]), &(vr[1]), &(vr[2]),
         &(vs[0]), &(vs[1]), &(vs[2]), &(avt[0]), &(avt[1]), &(avt[2]),
         &(avi[0]), &(avi[1]), &(avi[2]), &(avc[0]), &(avc[1]), &(avc[2]),
         &(avr[0]), &(avr[1]), &(avr[2]), &(avs[0]), &(avs[1]), &(avs[2]));

  if (t == old_t)
    return NULL;
  old_t = t;

  for (i = TIMES - 1; i > 0; i--) {
    timeMatrix[i] = timeMatrix[i-1];
    for (j = 0; j < 3; j++) {
      vpMatrix[i][j] = vpMatrix[i-1][j];
      apMatrix[i][j] = apMatrix[i-1][j];
      vtMatrix[i][j] = vtMatrix[i-1][j];
      viMatrix[i][j] = viMatrix[i-1][j];
      vcMatrix[i][j] = vcMatrix[i-1][j];
      vrMatrix[i][j] = vrMatrix[i-1][j];
      vsMatrix[i][j] = vsMatrix[i-1][j];
      avtMatrix[i][j] = avtMatrix[i-1][j];
      aviMatrix[i][j] = aviMatrix[i-1][j];
      avcMatrix[i][j] = avcMatrix[i-1][j];
      avrMatrix[i][j] = avrMatrix[i-1][j];
      avsMatrix[i][j] = avsMatrix[i-1][j];
    }
```



```
  }

  timeMatrix[0] = t;
  for (j = 0; j < 3; j++) {
    vpMatrix[0][j] = vp[j];
    apMatrix[0][j] = ap[j];
    vtMatrix[0][j] = vt[j];
    viMatrix[0][j] = vi[j];
    vcMatrix[0][j] = vc[j];
    vrMatrix[0][j] = vr[j];
    vsMatrix[0][j] = vs[j];
    avtMatrix[0][j] = avt[j];
    aviMatrix[0][j] = avi[j];
    avcMatrix[0][j] = avc[j];
    avrMatrix[0][j] = avr[j];
    avsMatrix[0][j] = avs[j];
  }
  /* v?Matrix には過去のデータを入れておく */
  /* v?Matirx[0] が最新のデータ */
  /* v?Matrix[TIMES-1] がもっとも古い */

#ifdef DEBUG
    printf("%lf %lf %lf %lf %lf %lf %lf %lf %lf\n",
           vp[0], vp[1], vp[2], vt[0], vt[1], vt[2], vi[0], vi[1], vi[2]);
#endif
}

void LastData(head)
     char *head;
{
  if (shmdt(head) == EOF) {
    perror("shmdt");
    exit();
  }
  if (shmctl(shmid, IPC_RMID) == EOF) {
    perror("shmctl");
    exit();
  }
}

char *DataTop()
{
  char *head;
  if ((shmid = shmget(SHM_KEY, DATASIZE, SHM_MODE | IPC_ALLOC)) == EOF)
    if ((shmid = shmget(SHM_KEY, DATASIZE, SHM_MODE | IPC_CREAT)) == EOF) {
      perror("shmget");
      return(NULL);
    }
  if (*(head = shmat(shmid, 0, 0)) == EOF) {
    perror("shmat");
    return(NULL);
  }
  return(head);
}
```

```
 1 /* tips.c                                                              */
 2 /* このプログラムはNo.5, No.6 No.7 のtracker から生データを取得し これら */
 3 /* のデータから No.6, No.7 の位置を No.5を基準とした相対位置データに変換 */
 4 /* する. この変換されたデータと元の No.5のデータおよびその時の時間を    */
 5 /* shared memory に書き込む. そのプロトコルは, ??                       */
 6 /*                                        04/08/91                     */
 7 /*                                                                     */
 8
 9 #include <stdio.h>
10 #include <math.h>
11 #include <device.h>
12 #include "claw.h"
13 #include "get_time.h"
14
15 #define COSD(x) cos(M_PI*(x)/180.0)
16 #define SIND(x) sin(M_PI*(x)/180.0)
17
18 int bitc();
19 int shmid;
20 char *head, *shmat();
21
22 main(argc, argv)
23      int argc;
24      char *argv[];
25 {
26   int i, m, n, loop;
27   double x, y, z, a, r, e;     /* トラッカーからのデータの一時的格納用 */
28   double vp[3], vt[3], vi[3]; /* vp 手の甲の3D絶対位置 */
29                                /* vt 手の甲に対する親指の先端位置 */
30                                /* vi 手の甲に対する人指し指の先端位置 */
31   double ap[3];                /* ap 手の甲の3D空間内の角度 */
32   double ca, sa, ce, se, cr, sr;
33   double pa, pb, pc, pd, pe, pf, pg, ph, pi, pj, pk, pl;
34                                /* c?, s?, p?                          */
35                                /* vt, vi を求めるための変数  */
36   double current_t, old_t, delay;    /* 時間制御用 */
37   double dummy[3];             /* ダミー */
38
39   if (argc < 2)
40     delay = 0.098;             /* サンプリングタイムのdefault */
41                                /* は, 10Hz                            */
42   else
43     delay = 1.0/atof(argv[1]) - 0.02;
44                                /* 一つ目の引数はHz */
45   trinit();                    /* トラッカー初期化 */
46   trset(112);                  /* トラッカーの5, 6, 7番の使用を宣言 */
47   trcalib("3space.dat");       /* ファイル ./3space.dat 中の座標系を使用 */
48
49   loop = bitc(112);
50   trroton();
51
52
53   if ((shmid = shmget(SHM_KEY, DATASIZE, SHM_MODE | IPC_ALLOC)) == EOF)
54       if ((shmid = shmget(SHM_KEY, DATASIZE, SHM_MODE | IPC_CREAT)) == EOF) {
55         perror("shmget");
56         exit();
57       }
58   if (*(head = shmat(shmid, 0, 0)) == EOF) {
59     perror("shmat");
60     exit();
61   }
62
63   for (;;) {
64     while (((current_t = get_double_time()) - old_t) < delay)
65       ; /* 10Hz */
66     trposin();
```



```
67     for (m = 0; m < loop; m++) {
68       n = trposrotrd(&x, &y, &z, &a, &e, &r);
69       if ( n == 5) {
70         /* 5番のトラッカーを手の甲に */
71         vp[0] = pa = x;
72         vp[1] = pb = y;
73         vp[2] = pc = z;
74         ap[0] = pj = a;
75         ap[1] = pk = e;
76         ap[2] = pl = r;
77       }
78       else if (n == 6) {
79         /* 6番のトラッカーを親指に */
80         pd = x;
81         pe = y;
82         pf = z;
83       }
84       else if (n ==  7) {
85         /* 7番のトラッカーを人指し指に */
86         pg = x;
87         ph = y;
88         pi = z;
89       }
90     }
91
92     ca = COSD(pj); sa = SIND(pj);
93     ce = COSD(pk); se = SIND(pk);
94     cr = COSD(pl); sr = SIND(pl);
95
96     vt[0] = (pd-pa)*ca*ce + (pe-pb)*sa*ce + (pf-pc)*(-1*se);
97     vt[1] = (pd-pa)*(ca*se*sr-sa*cr)
98           + (pe-pb)*(ca*cr+sa*se*sr)
99           + (pf-pc)*ce*sr;
100    vt[2] = (pd-pa)*(ca*se*cr+sa*sr)
101          + (pe-pb)*(sa*se*cr-ca*sr)
102          + (pf-pc)*ce*cr;
103    /* これで間違いなく親指の先端位置が求められた！？ */
104
105    vi[0] = (pg-pa)*ca*ce + (ph-pb)*sa*ce + (pi-pc)*(-1*se);
106    vi[1] = (pg-pa)*(ca*se*sr-sa*cr)
107          + (ph-pb)*(ca*cr+sa*se*sr)
108          + (pi-pc)*(ce*sr);
109    vi[2] = (pg-pa)*(ca*se*cr+sa*sr)
110          + (ph-pb)*(sa*se*cr-ca*sr)
111          + (pi-pc)*(ce*cr);
112    /* again I hope 間違いなく人指し指の位置が求められたことを */
113
114    sprintf(head,
115            "%6.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %
   7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2
   f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f",
116            current_t, vp[0], vp[1], vp[2], ap[0], ap[1], ap[2],
117            vt[0], vt[1], vt[2], vi[0], vi[1], vi[2],
118            dummy[0], dummy[1], dummy[2], dummy[0], dummy[1], dummy[2],
119            dummy[0], dummy[1], dummy[2], pd, pe, pf,
120            pg, ph, pi, dummy[0], dummy[1], dummy[2],
121            dummy[0], dummy[1], dummy[2], dummy[0], dummy[1], dummy[2]);
122
123 #ifdef DEBUG
124    fprintf(stderr, "%s\n", head);
125 #endif
126
127    old_t = current_t;
128
129    /*   if (vp[0] < -10.0)  This way is not smart. */
130
```

```
131     if (getbutton(SW0))
132       /* 逃げる手段も入れておく */
133         break;
134   }
135
136
137   if (shmdt(head) == EOF)
138     perror("shmdt");
139
140   if (shmctl(shmid, IPC_RMID, 0) == EOF)
141     perror("shmctl");
142
143 }
144
145 int bitc(m)
146      int m;
147 {
148   int n, c, mask;
149
150   for (c = 0, mask = 1, n = 1; n < 9; mask <<= 1,  n++)
151     if(mask&m)
152       c++;
153   return c;
154 }
```

```
/* ssd.c */

#include <stdio.h>
#include <gl.h>
#include <device.h>
#include <sys/types.h>
#include <fcntl.h>
#include "subspace.h"
#include "gObject.h"
#include "claw.h"
#include "get_time.h"

extern double timeMatrix[TIMES];
extern double vpMatrix[TIMES][3];
extern double apMatrix[TIMES][3];
extern double vtMatrix[TIMES][3];
extern double viMatrix[TIMES][3];
extern double vcMatrix[TIMES][3];
extern double vrMatrix[TIMES][3];
extern double vsMatrix[TIMES][3];
extern double avtMatrix[TIMES][3];
extern double aviMatrix[TIMES][3];
extern double avcMatrix[TIMES][3];
extern double avrMatrix[TIMES][3];
extern double avsMatrix[TIMES][3];

extern void GetHandData();
extern char *DataTop();
extern void LastData();
extern void LearningStage();
extern void RecognizingStage();
extern int RealTimeRecognize();
extern int EndHand();

extern int SaveClassData();
extern int LoadClassData();

extern int shmid;
extern double vec[6];
extern struct Class *class[10];

void do_what();

int handshape;
char *head;

main(argc, argv)
     int argc;
     char *argv[];
{
  double t, ot;
  int p = 0;
  head = DataTop();

  if (argc == 2)
    if ( ! strcmp(argv[1],"-p"))
      p = 1;
  LearningStage();
  SaveClassData(DEFCLASSDATA);
  (void) EndHand();
  LoadClassData(DEFCLASSDATA);
  while(1) {
    while((t=get_double_time())-ot < 0.09)
      ;
    GetHandData(head);
    handshape = RealTimeRecognize();
```



```
    if (p) {
      fprintf(stdout, "%f %f %f %f %f %f\n",
              vtMatrix[0][0], vtMatrix[0][1], vtMatrix[0][2],
              viMatrix[0][0], viMatrix[0][1], viMatrix[0][2]);
      fflush(stdout);
    }
  printf("%d %f %f %f %f %f %f\n",
              handshape, vtMatrix[0][0], vtMatrix[0][1], vtMatrix[0][2],
              viMatrix[0][0], viMatrix[0][1], viMatrix[0][2]);

    if (getbutton(LEFTMOUSE))
      break;
    ot = t;
  }
  LastData(head);
 /*  EndHand(); */
}
```

```
/* r_network.c */
#include "network.h"
#include <math.h>
#include <stdio.h>
#include <ctype.h>
#include "r_network.h"

extern double r_tampon1[R_NUM_CON1], r_tampon2[R_NUM_CON2];

/* Input Patterns */
/*
double  r_input_pat[R_PATTERNS+1][R_NUM_IN] = {
{0.524000},
{0.524000},
{0.444000},
{0.373000},
{0.352000},
{0.282000},
{0.207000},
{0.116000},
{0.074000},
{0.103000},
{0.134000},
{0.212000},
{0.330000},
{0.490000},
{0.652000},
{0.652000},
{0.647000},
{0.597000},
{0.478000},
{0.340000},
{0.098000},
{-0.014000},
{-0.053000},
{-0.055000},
{-0.016000},
{0.073000},
{0.277000},
{0.435000},
{0.637000},
{0.754000},
{0.776000},
{0.727000},
{0.556000},
{0.394000},
{0.197000},
{0.021000},
{-0.075000},
{-0.068000},
{-0.042000},
{0.018000},
{0.207000},
{0.377000},
{0.610000},
{0.752000},
{0.752000},
{0.758000},
{0.613000},
{0.470000},
{0.325000},
{0.168000},
{0.168000},
{0.004000},
{0.011000},
{0.105000},
```



```
{0.319000},
{0.522000},
{0.713000},
{0.736000},
{0.716000},
{0.641000},
};
*/
/* Target Patterns */
/*
double  r_target_pat[R_PATTERNS][R_NUM_OUT] = {
{0.000000},
{0.000000},
{0.100000},
{0.200000},
{0.500000},
{0.700000},
{0.800000},
{0.450000},
{0.300000},
{0.100000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
{0.100000},
{0.400000},
{0.750000},
{0.750000},
{0.300000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
{0.100000},
{0.750000},
{0.750000},
{0.300000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
{0.300000},
{0.500000},
{0.600000},
{0.400000},
{0.300000},
{0.000000},
{0.000000},
{0.000000},
{0.000000},
```

```
{0.000000},
{0.000000},
{0.000000},
{0.200000},
{0.300000},
};
*/

double r_out_f(), r_delta_f_hid(), r_delta_f_out(), r_pattern_error();
double r_pattern_err[R_PATTERNS];
double r_rdm();

/*
 * create input, hidden, output units (and threshold or bias unit)
 */
r_create_processing_units(pu)
struct unit *pu[];
{
  int   id;                     /* processing unit index */
  struct unit *create_unit();

  for (id = R_IN_UID(0); id < R_IN_UID(R_NUM_IN); id++)
    pu[id] = create_unit(id, "input", 0.0, NULL, 0.0, NULL, 0.0);
  for (id = R_CON_UID1(0); id < R_CON_UID1(R_NUM_CON1); id++)
    pu[id] = create_unit(id, "context1", 0.0, NULL, 0.0, NULL, 0.0);
  for (id = R_CON_UID2(0); id < R_CON_UID2(R_NUM_CON2); id++)
    pu[id] = create_unit(id, "context2", 0.0, NULL, 0.0, NULL, 0.0);
  for (id = R_HID_UID(0); id < R_HID_UID(R_NUM_HID); id++)
    pu[id] = create_unit(id, "hidden", 0.0, r_out_f, 0.0, r_delta_f_hid, r_rdm()
);
  for (id = R_OUT_UID(0); id < R_OUT_UID(R_NUM_OUT); id++)
    pu[id] = create_unit(id, "output", 0.0, r_out_f, 0.0, r_delta_f_out,r_rdm())
;
  for (id = 0; id < R_NUM_CON1; id++) {
    r_tampon1[id] = .5;
    r_tampon2[id] = .5;
  }
}

/*
 * create links - fully connected for each layer
 *              note: the bias unit has one link to ea hid and out unit
 */
r_create_in_out_links(pu)
struct unit *pu[];
{
  int   i, j, k;                /* i == to and j == from unit id's */
  struct link *create_link();

  /* fully connected units */
  for (i = R_HID_UID(0); i < R_HID_UID(R_NUM_HID); i++) { /* links to hidden */
    for (j = R_IN_UID(0); j < R_IN_UID(R_NUM_IN); j++) /* from input units */
      pu[j]->outlinks =
        pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->outlinks, j,
                                     (char *)NULL, r_rdm(), 0.0);
    for (j = R_CON_UID1(0); j < R_CON_UID1(R_NUM_CON1); j++)
      pu[j]->outlinks =
        pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->outlinks, j,
                                     (char *)NULL, r_rdm(), 0.0);
    for (j = R_CON_UID2(0); j < R_CON_UID2(R_NUM_CON2); j++)
      pu[j]->outlinks =
        pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->outlinks, j,
                                     (char *)NULL, r_rdm(), 0.0);
  }
  for (i = R_OUT_UID(0); i < R_OUT_UID(R_NUM_OUT); i++) {       /* links to outp
ut */
```



```
    for (j = R_HID_UID(0); j < R_HID_UID(R_NUM_HID); j++) /* from hidden units *
/
      pu[j]->outlinks =
        pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->outlinks, j,
                                     (char *)NULL, r_rdm(), 0.0);
  }
}

r_learn(pu)
struct unit *pu[];
{
  int j, k;
  register i, temp;
  char   tempstr[BUFSIZE];
  extern int    iterations;
  static char prompt[] = "Enter # iterations (default is 8000) => ";
  static char quotel[] = "I should do more learning.\n";

  printf(prompt);
  fflush(stdin); fflush(stdout);
  gets(tempstr);
  if (temp = atoi(tempstr))
    iterations = temp;

  printf("\nLearning ");
  for (i = 0; i < iterations; i++) {
    if ((i % NOTIFY) == 0) {
      printf(".");
      fflush(stdout);
    }
    if (i > iterations - R_PATTERNS)
      r_bp_learn(pu, 1);
    else
      r_bp_learn(pu, 0);
  }
  printf(" Done\n\n");
  for (j = 0; j < R_PATTERNS; j=j+2)
    printf("Error for pattern %d = \t%lf\tError for pattern %d =\t%lf\n", j, r_p
attern_err[j], j+1, r_pattern_err[j+1]);

  for (j = 2; j < R_PATTERNS; j++) {
    if (r_pattern_err[j] > 0.05) {
      printf("\nI don't know pattern %d very well.\n%s",j, quotel);
    }
    if (j == R_PATTERNS - 1) {
      printf("\nOK. I know all the patterns quite well, now.\n");
      break;
    }
  }
}

/* back propagation learning */
r_bp_learn(pu, save_error)
struct unit *pu[];
int    save_error;
{
  int i;
  static int count = 0;
  static int pattern = 0;
  extern int iterations;
  extern double r_pattern_err[R_PATTERNS];

  r_init_input_units(pu, pattern); /* initialize input pattern to
                                                        learn */
  r_propagate(pu);                 /* calc outputs to check versus
                                                        targets */
```

```
260   if (save_error)
261     r_pattern_err[pattern] = r_pattern_error(pattern, pu);
262   r_bp_adjust_weights(pattern, pu);
263   if (pattern < R_PATTERNS - 1)
264     pattern++;
265   else
266       pattern = 0;
267   count++;
268
269   if (count == iterations) {
270      for (i = 0; i < R_NUM_CON1; i++) {
271         r_tampon1[i] = 0.5;
272         r_tampon2[i] = 0.5;
273      }
274      count = 0;
275   }
276 }
277
278 /* initialize the input units with a specific input pattern to learn */
279 r_init_input_units(pu, pattern)
280 struct unit *pu[];
281 int    pattern;
282 {
283   int   id;
284
285   for (id = R_IN_UID(0); id < R_IN_UID(R_NUM_IN); id++)
286     pu[id]->output = r_input_pat[pattern][id];
287   for (id = R_CON_UID1(0); id < R_CON_UID1(R_NUM_CON1); id++)
288     pu[id]->output = r_tampon1[id - R_CON_UID1(0)];
289   for (id = R_CON_UID2(0); id < R_CON_UID2(R_NUM_CON2); id++)
290     pu[id]->output = r_tampon2[id - R_CON_UID2(0)];
291 }
292
293 /* calculate the activation level of each unit */
294 r_propagate(pu)
295 struct unit *pu[];
296 {
297   int   id;
298
299   for (id = R_HID_UID(0); id < R_HID_UID(R_NUM_HID); id++)
300     (*(pu[id]->unit_out_f))(pu[id], pu);
301   for (id = R_OUT_UID(0); id < R_OUT_UID(R_NUM_OUT); id++)
302     (*(pu[id]->unit_out_f))(pu[id], pu);
303   for (id = 0; id < R_NUM_CON1; id++)
304     r_tampon2[id] = r_tampon1[id];
305   for (id = R_HID_UID(0); id < R_HID_UID(R_NUM_HID); id++)
306     r_tampon1[id - R_HID_UID(0)] = pu[id]->output;
307 }
308
309 /* function to calculate the activation or output of units */
310 double
311 r_out_f(pu_ptr, pu)
312 struct unit *pu_ptr, *pu[];
313 {
314   double sum = 0.0, exp();
315   struct link *tmp_ptr;
316
317   tmp_ptr = pu_ptr->inlinks;
318   while (tmp_ptr) {
319     /* sum up (outputs from inlinks times weights on the inlinks) */
320     sum += pu[tmp_ptr->from_unit]->output * tmp_ptr->weight;
321     tmp_ptr = tmp_ptr->next_inlink;
322   }
323   sum += pu_ptr->threshold;
324   pu_ptr->output = 1.0/(1.0 + exp(-sum));
325   return(pu_ptr->output);
```



```
326 }
327
328 /* half of the sum of the squares of the errors of the
329    output versus target values */
330 double
331 r_pattern_error(pat_num, pu)
332 int     pat_num;        /* pattern number */
333 struct unit *pu[];
334 {
335   int           i;
336   double        temp, sum = 0.0;
337
338   for (i = R_OUT_UID(0); i < R_OUT_UID(R_NUM_OUT); i++) {
339     temp = r_target_pat[pat_num][R_TARGET_INDEX(i)] - pu[i]->output;
340     sum += temp * temp;
341   }
342   return (sum/2.0);
343 }
344
345 r_bp_adjust_weights(pat_num, pu)
346 int     pat_num;        /* pattern number */
347 struct unit *pu[];
348 {
349   int           i;              /* processing units id */
350   double        temp1, temp2, delta, error_sum;
351   struct link   *inlink_ptr, *outlink_ptr;
352
353   /* calc deltas */
354   for (i = R_OUT_UID(0); i < R_OUT_UID(R_NUM_OUT); i++) /* for each output unit
    */
355     (*(pu[i]->unit_delta_f))(pu, i, pat_num); /* calc delta */
356   for (i = R_HID_UID(0); i < R_HID_UID(R_NUM_HID); i++) /* for each hidden unit
    */
357     (*(pu[i]->unit_delta_f))(pu, i);      /* calc delta */
358   /* calculate weights and thresholds */
359   for (i = R_OUT_UID(0); i < R_OUT_UID(R_NUM_OUT); i++) {       /* for output un
    its */
360     inlink_ptr = pu[i]->inlinks;
361     while (inlink_ptr) {        /* for each inlink to output unit */
362       temp1 = LEARNING_RATE * pu[i]->delta *
363         pu[inlink_ptr->from_unit]->output;
364       temp2 = MOMENTUM * inlink_ptr->data;
365       inlink_ptr->data = temp1 + temp2; /* new delta weight */
366       inlink_ptr->weight += inlink_ptr->data;   /* new weight */
367       inlink_ptr = inlink_ptr->next_inlink;
368     }
369     pu[i]->threshold += LEARNING_RATE * pu[i]->delta;    /* new threshold */
370   }
371   for (i = R_HID_UID(0); i < R_HID_UID(R_NUM_HID); i++) { /* for each hid unit *
    /
372     inlink_ptr = pu[i]->inlinks;
373     while (inlink_ptr) {        /* for each inlink to output unit */
374       temp1 = LEARNING_RATE * pu[i]->delta *
375         pu[inlink_ptr->from_unit]->output;
376       temp2 = MOMENTUM * inlink_ptr->data;
377       inlink_ptr->data = temp1 + temp2; /* new delta weight */
378       inlink_ptr->weight += inlink_ptr->data;   /* new weight */
379         inlink_ptr = inlink_ptr->next_inlink;
380     }
381     pu[i]->threshold += LEARNING_RATE * pu[i]->delta;    /* new threshold */
382   }
383 }
384
385 /* calculate the delta for an output unit */
386 double
387 r_delta_f_out(pu, uid, pat_num)
```

```
388 struct unit *pu[];
389 int    uid, pat_num;
390 {
391   double        temp1, temp2, delta;
392
393   /* calc deltas */
394   temp1 = (r_target_pat[pat_num][R_TARGET_INDEX(uid)] - pu[uid]->output);
395   temp2 = (1.0 - pu[uid]->output);
396   delta = temp1 * pu[uid]->output * temp2; /* calc delta */
397   pu[uid]->delta = delta; /* store delta to pass on */
398 }
399
400 /* calculate the delta for a hidden unit */
401 double
402 r_delta_f_hid(pu, uid)
403 struct unit *pu[];
404 int    uid;
405 {
406   double        temp1, temp2, delta, error_sum;
407   struct link   *inlink_ptr, *outlink_ptr;
408
409   outlink_ptr = pu[uid]->outlinks;
410   error_sum = 0.0;
411   while (outlink_ptr) {
412     error_sum += pu[outlink_ptr->to_unit]->delta * outlink_ptr->weight;
413     outlink_ptr = outlink_ptr->next_outlink;
414   }
415   delta = pu[uid]->output * (1.0 - pu[uid]->output) * error_sum;
416   pu[uid]->delta = delta;
417 }
418
419 /* save weight of all the connections and threshold of hidden and output units *
    /
420 r_save_weights(pu)
421 struct unit *pu[];
422 {
423   int i, j, k = 0, l = 0;
424   double table_weights[(R_NUM_IN + R_NUM_CON1 + R_NUM_CON2)*R_NUM_HID + R_NUM_HI
    D*R_NUM_OUT + R_NUM_HID + R_NUM_OUT];          /* size = total number of links
    + number of thresholds; only */
425                         /* hidden and output units have a threshold
        */
426   FILE *fiche, *errf;
427   static char file_name[] = "R_WEIGHTS_FILE";
428   static char err_file[] = "R_ERROR_FILE";
429   struct link *tmp_ptr;
430
431
432   for (i = R_HID_UID(0); i < R_HID_UID(R_NUM_HID); i++) {
433     tmp_ptr = pu[i]->inlinks;
434     table_weights[k*(R_NUM_IN + R_NUM_CON1 + R_NUM_CON2 + 1)] = pu[i]->threshold
    ;
435     j = (k + 1)*(R_NUM_IN + R_NUM_CON1 + R_NUM_CON2 + 1) - 1;
436     while (tmp_ptr) {
437         table_weights[j] = tmp_ptr->weight;
438         j--;
439         tmp_ptr = tmp_ptr->next_inlink;
440     }
441     k++;
442   }
443
444   for (i = R_OUT_UID(0); i < R_OUT_UID(R_NUM_OUT); i++) {
445     tmp_ptr = pu[i]->inlinks;
446     table_weights[k*(R_NUM_IN + R_NUM_CON1 + R_NUM_CON2 + 1) + l*(R_NUM_HID + 1)
    ] = pu[i]->threshold;
447     j = k*(R_NUM_IN + R_NUM_CON1 + R_NUM_CON2 + 1) + (l + 1)*(R_NUM_HID + 1) -1;
```



```
448     while (tmp_ptr) {
449       table_weights[j] = tmp_ptr->weight;
450         j--;
451         tmp_ptr = tmp_ptr->next_inlink;
452     }
453     l++;
454   }
455
456   fiche = fopen(file_name, "w");
457   for (i = 0; i < (R_NUM_IN + R_NUM_CON1 + R_NUM_CON2)*R_NUM_HID + R_NUM_HID*R_N
    UM_OUT + R_NUM_HID + R_NUM_OUT; i++)
458       fprintf(fiche, "%lf\n", table_weights[i]);
459   fclose(fiche);
460   errf = fopen(err_file, "w");
461   for (i = 0; i < R_PATTERNS; i++)
462     fprintf(errf, "%lf\n", r_pattern_err[i]);
463   fclose(errf);
464
465 }
466
467 r_load_weights(pu)
468 struct unit* pu[];
469 {
470   int i, j;
471   double x;
472   FILE *fp;
473   static char name[] = "R_WEIGHTS_FILE";
474
475   if ((fp = fopen(name, "r")) == NULL) {
476      fprintf(stderr, "\nCan't load the file -[%s]\n", name);
477      exit(1);
478   }
479
480   for (i = R_HID_UID(0); i < R_HID_UID(R_NUM_HID); i++) {
481    fscanf(fp, "%lf", &x);
482    pu[i]->threshold = x;
483
484    for (j = R_IN_UID(0); j < R_IN_UID(R_NUM_IN); j++) {
485     fscanf(fp, "%lf", &x);
486     pu[j]->outlinks = pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->out
    links, j,
487                                                    (char *)NULL, x, 0.0);
488    }
489
490    for (j = R_CON_UID1(0); j < R_CON_UID1(R_NUM_CON1); j++) {
491     fscanf(fp, "%lf", &x);
492     pu[j]->outlinks = pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->out
    links, j,
493                                                    (char *)NULL, x, 0.0);
494    }
495
496    for (j = R_CON_UID2(0); j < R_CON_UID2(R_NUM_CON2); j++) {
497     fscanf(fp, "%lf", &x);
498     pu[j]->outlinks = pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->out
    links, j,
499                                                    (char *)NULL, x, 0.0);
500    }
501   }
502
503   for (i = R_OUT_UID(0); i < R_OUT_UID(R_NUM_OUT); i++) {
504    fscanf(fp, "%lf", &x);
505    pu[i]->threshold = x;
506
507    for (j = R_HID_UID(0); j < R_HID_UID(R_NUM_HID); j++) {
508     fscanf(fp, "%lf", &x);
509     pu[j]->outlinks = pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->out
```

```
    links, j,
510                                                 (char *)NULL, x, 0.0);
511     }
512   }
513
514   fclose(fp);
515 }
516
517 double
518 r_rdm()
519 {
520   return((rand() % 50000 - 25000)/ 25000.0);
521 }
```

```
/* s_network.c */
#include "network.h"
#include <math.h>
#include <stdio.h>
#include <ctype.h>

/* extern double s_tampon1[S_NUM_CON1], s_tampon2[S_NUM_CON2]; */

/* Input Patterns */
/*
double  s_input_pat[S_PATTERNS+1][S_NUM_IN] = {
{0.524000, -0.543000, 1.067000, 0.821000, -0.356000, 1.069000},
{0.524000, -0.543000, 1.067000, 0.821000, -0.356000, 1.069000},
{0.444000, -0.522000, 1.139000, 0.724000, -0.348000, 1.179000},
{0.373000, -0.507000, 1.177000, 0.626000, -0.317000, 1.229000},
{0.352000, -0.503000, 1.158000, 0.590000, -0.312000, 1.201000},
{0.282000, -0.444000, 1.206000, 0.580000, -0.286000, 1.200000},
{0.207000, -0.596000, 1.196000, 0.764000, -0.206000, 1.155000},
{0.116000, -0.703000, 1.146000, 0.992000, -0.097000, 0.978000},
{0.074000, -0.802000, 1.089000, 1.158000, -0.078000, 0.728000},
{0.103000, -0.842000, 1.048000, 1.250000, -0.124000, 0.498000},
{0.134000, -0.861000, 1.016000, 1.292000, -0.165000, 0.397000},
{0.212000, -0.853000, 0.989000, 1.327000, -0.204000, 0.473000},
{0.330000, -0.771000, 0.985000, 1.254000, -0.236000, 0.620000},
{0.490000, -0.714000, 0.905000, 1.209000, -0.286000, 0.731000},
{0.652000, -0.691000, 0.817000, 1.083000, -0.425000, 0.787000},
{0.652000, -0.694000, 0.827000, 0.962000, -0.558000, 0.750000},
{0.647000, -0.656000, 0.889000, 0.951000, -0.531000, 0.833000},
{0.597000, -0.580000, 1.005000, 0.889000, -0.432000, 1.017000},
{0.478000, -0.477000, 1.147000, 0.752000, -0.323000, 1.215000},
{0.340000, -0.448000, 1.219000, 0.684000, -0.231000, 1.267000},
{0.098000, -0.526000, 1.257000, 0.748000, -0.126000, 1.216000},
{-0.014000, -0.669000, 1.163000, 0.916000, -0.060000, 1.096000},
{-0.053000, -0.770000, 1.105000, 1.056000, -0.071000, 0.906000},
{-0.055000, -0.833000, 1.067000, 1.166000, -0.106000, 0.682000},
{-0.016000, -0.860000, 1.052000, 1.233000, -0.138000, 0.513000},
{0.073000, -0.884000, 1.021000, 1.292000, -0.154000, 0.470000},
{0.277000, -0.759000, 0.986000, 1.270000, -0.168000, 0.650000},
{0.435000, -0.675000, 0.919000, 1.127000, -0.240000, 0.712000},
{0.637000, -0.617000, 0.824000, 1.096000, -0.302000, 0.811000},
{0.754000, -0.648000, 0.762000, 1.075000, -0.467000, 0.694000},
{0.776000, -0.651000, 0.792000, 1.062000, -0.508000, 0.724000},
{0.727000, -0.584000, 0.942000, 1.012000, -0.417000, 0.951000},
{0.556000, -0.473000, 1.107000, 0.859000, -0.352000, 1.079000},
{0.394000, -0.535000, 1.163000, 0.881000, -0.256000, 1.110000},
{0.197000, -0.614000, 1.193000, 0.932000, -0.186000, 1.065000},
{0.021000, -0.691000, 1.155000, 1.038000, -0.109000, 0.925000},
{-0.075000, -0.775000, 1.100000, 1.122000, -0.083000, 0.729000},
{-0.068000, -0.819000, 1.075000, 1.189000, -0.092000, 0.530000},
{-0.042000, -0.818000, 1.065000, 1.228000, -0.101000, 0.402000},
{0.018000, -0.839000, 1.039000, 1.267000, -0.180000, 0.343000},
{0.207000, -0.815000, 0.986000, 1.280000, -0.228000, 0.462000},
{0.377000, -0.744000, 0.939000, 1.239000, -0.282000, 0.542000},
{0.610000, -0.699000, 0.837000, 1.236000, -0.340000, 0.628000},
{0.752000, -0.699000, 0.727000, 1.175000, -0.437000, 0.612000},
{0.752000, -0.699000, 0.727000, 1.175000, -0.437000, 0.612000},
{0.758000, -0.577000, 0.882000, 1.169000, -0.227000, 0.978000},
{0.613000, -0.519000, 1.079000, 1.085000, -0.280000, 0.869000},
{0.470000, -0.624000, 1.131000, 1.067000, -0.358000, 0.690000},
{0.325000, -0.720000, 1.128000, 1.044000, -0.429000, 0.481000},
{0.168000, -0.793000, 1.104000, 0.984000, -0.505000, 0.251000},
{0.168000, -0.793000, 1.104000, 0.984000, -0.505000, 0.251000},
{0.004000, -0.865000, 1.048000, 0.831000, -0.613000, -0.031000},
{0.011000, -0.877000, 1.032000, 0.786000, -0.653000, -0.080000},
{0.105000, -0.855000, 1.013000, 1.244000, -0.149000, 0.586000},
{0.319000, -0.784000, 0.954000, 1.203000, -0.263000, 0.645000},
```



```
{0.522000, -0.742000, 0.832000, 1.183000, -0.342000, 0.697000},
{0.713000, -0.694000, 0.751000, 1.095000, -0.477000, 0.713000},
{0.736000, -0.685000, 0.764000, 1.047000, -0.531000, 0.723000},
{0.716000, -0.655000, 0.859000, 1.000000, -0.482000, 0.867000},
{0.641000, -0.544000, 1.002000, 0.903000, -0.375000, 1.045000},
{0.488000, -0.516000, 1.131000, 0.879000, -0.254000, 1.128000},
{0.268000, -0.577000, 1.194000, 0.906000, -0.106000, 1.161000},
{0.051000, -0.658000, 1.171000, 0.976000, -0.033000, 1.062000},
{-0.033000, -0.730000, 1.121000, 1.085000, -0.037000, 0.859000},
{-0.069000, -0.803000, 1.071000, 1.151000, -0.044000, 0.682000},
{-0.051000, -0.835000, 1.052000, 1.182000, -0.074000, 0.559000},
{-0.013000, -0.860000, 1.031000, 1.228000, -0.127000, 0.498000},
{0.195000, -0.799000, 0.998000, 1.253000, -0.192000, 0.594000},
{0.405000, -0.723000, 0.927000, 1.217000, -0.264000, 0.655000},
{0.622000, -0.652000, 0.761000, 1.106000, -0.385000, 0.629000},
{0.727000, -0.677000, 0.670000, 1.020000, -0.532000, 0.568000},
{0.734000, -0.708000, 0.689000, 1.021000, -0.563000, 0.611000},
{0.733000, -0.776000, 0.768000, 1.039000, -0.639000, 0.753000},
{0.684000, -0.667000, 0.918000, 0.964000, -0.490000, 0.960000},
{0.684000, -0.667000, 0.918000, 0.964000, -0.490000, 0.960000},
{0.416000, -0.557000, 1.206000, 0.894000, -0.144000, 1.089000},
{0.158000, -0.698000, 1.188000, 0.983000, 0.194000, 0.978000},
{0.032000, -0.842000, 1.078000, 1.049000, 0.266000, 0.786000},
{0.017000, -0.950000, 0.993000, 1.124000, 0.189000, 0.591000},
{0.013000, -0.976000, 0.962000, 1.168000, 0.095000, 0.464000},
{0.065000, -0.986000, 0.937000, 1.245000, -0.054000, 0.392000},
{0.334000, -0.829000, 0.904000, 1.295000, -0.189000, 0.509000},
{0.552000, -0.733000, 0.795000, 1.201000, -0.382000, 0.556000},
{0.711000, -0.758000, 0.672000, 1.082000, -0.683000, 0.473000},
{0.742000, -0.723000, 0.719000, 1.040000, -0.717000, 0.536000},
{0.735000, -0.651000, 0.823000, 1.049000, -0.625000, 0.706000},
{0.679000, -0.514000, 0.962000, 1.017000, -0.473000, 0.861000},
{0.524000, -0.510000, 1.155000, 1.118000, -0.372000, 0.848000},
{0.306000, -0.614000, 1.182000, 1.205000, -0.280000, 0.694000},
{0.071000, -0.695000, 1.177000, 1.207000, -0.223000, 0.511000},
{-0.026000, -0.743000, 1.149000, 1.182000, -0.188000, 0.411000},
{0.043000, -0.763000, 1.108000, 1.209000, -0.197000, 0.589000},
{0.324000, -0.674000, 1.067000, 1.158000, -0.332000, 0.789000},
{0.512000, -0.608000, 1.000000, 1.052000, -0.412000, 0.823000},
{0.607000, -0.555000, 0.982000, 0.996000, -0.415000, 0.898000},
{0.576000, -0.470000, 1.097000, 0.966000, -0.376000, 0.960000},
{0.433000, -0.543000, 1.199000, 1.080000, -0.334000, 0.918000},
{0.276000, -0.688000, 1.170000, 1.174000, -0.317000, 0.730000},
{0.136000, -0.826000, 1.088000, 1.197000, -0.294000, 0.512000},
{0.030000, -0.931000, 1.026000, 1.185000, -0.269000, 0.358000},
{0.005000, -0.962000, 1.004000, 1.169000, -0.260000, 0.284000},
{0.014000, -0.980000, 0.994000, 1.178000, -0.255000, 0.299000},
{0.204000, -0.919000, 1.007000, 1.239000, -0.247000, 0.557000},
{0.414000, -0.747000, 1.010000, 1.197000, -0.278000, 0.786000},
{0.526000, -0.650000, 0.966000, 1.041000, -0.376000, 0.853000},
{0.600000, -0.629000, 0.908000, 0.964000, -0.441000, 0.861000},
{0.618000, -0.622000, 0.909000, 0.912000, -0.481000, 0.872000},
{0.589000, -0.589000, 0.956000, 0.874000, -0.440000, 0.953000},
{0.544000, -0.545000, 1.007000, 0.826000, -0.406000, 1.023000},
{0.480000, -0.465000, 1.110000, 0.811000, -0.297000, 1.129000},
{0.376000, -0.559000, 1.163000, 0.968000, -0.242000, 1.071000},
{0.191000, -0.713000, 1.159000, 1.124000, -0.172000, 0.858000},
{0.044000, -0.838000, 1.096000, 1.155000, -0.131000, 0.591000},
{0.010000, -0.893000, 1.059000, 1.157000, -0.129000, 0.426000},
{0.096000, -0.871000, 1.064000, 1.167000, -0.153000, 0.397000},
{0.096000, -0.871000, 1.064000, 1.167000, -0.153000, 0.397000},
{0.376000, -0.671000, 1.077000, 1.171000, -0.219000, 0.841000},
{0.564000, -0.612000, 0.948000, 1.079000, -0.349000, 0.826000},
{0.685000, -0.642000, 0.841000, 1.011000, -0.519000, 0.761000},
{0.685000, -0.642000, 0.841000, 1.011000, -0.519000, 0.761000},
{0.637000, -0.578000, 0.940000, 0.938000, -0.428000, 0.943000},
```

```
{0.569000, -0.471000, 1.092000, 0.864000, -0.287000, 1.124000},
{0.569000, -0.471000, 1.092000, 0.864000, -0.287000, 1.124000},
{0.133000, -0.618000, 1.210000, 0.985000, -0.021000, 1.022000},
{-0.030000, -0.767000, 1.117000, 1.067000, -0.006000, 0.784000},
{-0.009000, -0.834000, 1.075000, 1.142000, -0.050000, 0.579000},
{0.050000, -0.878000, 1.045000, 1.198000, -0.124000, 0.436000},
{0.113000, -0.879000, 1.029000, 1.256000, -0.180000, 0.414000},
{0.233000, -0.876000, 0.993000, 1.316000, -0.245000, 0.514000},
{0.416000, -0.773000, 0.934000, 1.252000, -0.296000, 0.713000},
{0.599000, -0.690000, 0.810000, 1.080000, -0.443000, 0.710000},
{0.715000, -0.707000, 0.739000, 1.043000, -0.588000, 0.659000},
{0.707000, -0.711000, 0.753000, 1.009000, -0.577000, 0.704000},
{0.689000, -0.675000, 0.821000, 0.980000, -0.534000, 0.824000},
{0.638000, -0.588000, 0.932000, 0.905000, -0.383000, 1.034000},
{0.529000, -0.496000, 1.047000, 0.751000, -0.225000, 1.163000},
{0.170000, -0.471000, 1.274000, 0.751000, -0.006000, 1.232000},
{-0.024000, -0.622000, 1.165000, 0.933000, 0.057000, 1.056000},
{-0.067000, -0.762000, 1.062000, 1.090000, 0.009000, 0.803000},
{-0.097000, -0.914000, 0.952000, 1.171000, -0.071000, 0.555000},
{-0.086000, -0.999000, 0.904000, 1.210000, -0.161000, 0.377000},
{-0.033000, -1.024000, 0.894000, 1.233000, -0.242000, 0.290000},
{0.193000, -0.997000, 0.923000, 1.304000, -0.322000, 0.506000},
{0.426000, -0.930000, 0.894000, 1.286000, -0.354000, 0.705000},
{0.594000, -0.871000, 0.712000, 1.035000, -0.579000, 0.648000},
{0.677000, -0.911000, 0.666000, 1.051000, -0.725000, 0.596000},
{0.746000, -0.980000, 0.620000, 1.170000, -0.830000, 0.570000},
{0.800000, -0.926000, 0.761000, 1.294000, -0.725000, 0.836000},
{0.735000, -0.798000, 0.987000, 1.191000, -0.524000, 1.143000},
{0.619000, -0.618000, 1.169000, 1.046000, -0.345000, 1.350000},
{0.436000, -0.528000, 1.238000, 1.015000, -0.293000, 1.360000},
{0.182000, -0.630000, 1.192000, 1.129000, -0.179000, 1.229000},
{-0.094000, -0.767000, 1.065000, 1.217000, -0.195000, 0.894000},
{-0.153000, -0.863000, 1.000000, 1.263000, -0.235000, 0.646000},
{-0.115000, -0.913000, 0.977000, 1.293000, -0.275000, 0.460000},
{-0.064000, -0.969000, 0.917000, 1.296000, -0.299000, 0.288000},
{-0.001000, -0.998000, 0.841000, 1.227000, -0.291000, 0.077000},
{0.148000, -1.081000, 0.806000, 1.148000, -0.286000, -0.004000},
{0.463000, -1.235000, 0.743000, 1.075000, -0.248000, 0.137000},
{0.871000, -1.133000, 0.536000, 1.390000, -0.688000, 0.205000},
{1.013000, -1.034000, 0.528000, 1.666000, -0.686000, 0.325000},
};*/
double  s_input_pat[S_PATTERNS+1][S_NUM_IN] = {
{-0.450000, 0.220000, 0.200000, -0.719999, 0.250000, 0.440000},
{-0.240000, 0.000000, 0.000000, -0.350000, 0.000000, 0.180000},
{-0.240000, -0.170000, 0.000000, -0.220000, -0.170000, 0.000000},
{-0.140000, 0.000000, 0.000000, 0.000000, 0.140000, 0.130000},
{0.000000, 0.190000, 0.000000, 0.240000, 0.340000, 0.000000},
{0.000000, 0.000000, 0.000000, 0.410000, 0.240000, 0.000000},
{0.000000, -0.200000, 0.000000, 0.000000, -0.130000, -0.340000},
{0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
{0.000000, -0.310000, 0.000000, 0.460000, 0.220000, -0.120000},
{0.130000, 0.000000, 0.000000, 0.400000, 0.100000, 0.000000},
{0.000000, -0.170000, -0.150000, 0.170000, 0.000000, -0.330000},
{0.000000, -0.270000, 0.000000, 0.270000, 0.000000, -0.250000},
{0.000000, -0.260000, 0.000000, 0.350000, 0.170000, 0.110000},
{0.100000, 0.000000, 0.000000, 0.000000, 0.000000, -0.370000},
{0.000000, -0.200000, -0.110001, 0.200000, -0.150000, -0.690000},
{0.000000, -0.380000, 0.000000, 0.410001, 0.000000, -0.410000},
{-0.100000, -0.470000, 0.000000, 0.000000, -0.320000, -1.460000},
{-0.230000, -0.570000, 0.000000, 0.360000, 0.640000, -0.820001},
{0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
{-0.260000, -0.320000, 0.000000, -0.120000, 0.280000, -0.849999},
{-0.350000, -0.450000, -0.150000, -0.870000, 0.390000, -3.090000},
{-0.630000, -0.800000, -0.440001, -1.060000, 0.590000, -3.230000},
{0.000000, -0.460000, -0.110000, -0.170000, 0.000000, -1.320000},
{0.160000, -0.120000, 0.210000, 0.390000, -0.410000, 0.000000},
```



```
{1.870000, 2.020000, 0.639999, 3.720000, -1.180000, 10.350000},
{0.590000, 1.960000, -0.179999, -2.230000, -1.280000, 2.170000},
{0.000000, 0.380000, 0.000000, -1.120000, -0.440000, 0.190000},
{-0.190000, 0.000000, 0.120000, -0.490000, 0.200000, 0.360001},
{0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
{-0.230000, 0.210000, -0.150000, -0.450000, 0.330000, 0.000000},
{-0.110000, 0.130000, 0.000000, -0.250000, 0.230000, 0.190001},
{-0.100000, 0.110000, 0.100000, 0.000000, 0.280000, 0.000000},
{-0.160000, -0.130000, 0.179999, 0.000000, 0.150000, 0.000000},
{0.000000, 0.000000, 0.000000, 0.000000, 0.210000, -0.280000},
{-0.140000, -0.460000, 0.000000, 0.610001, 0.430000, 0.000000},
{0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
{0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
{0.000000, 0.000000, 0.000000, 0.180000, -0.140000, -0.380000},
{0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
{-0.280000, -0.800000, 0.000000, 0.000000, -0.130000, -0.209999},
{-0.400000, -0.780000, -0.140000, 1.450000, 0.430000, -3.370001},
{0.000000, -0.680000, -0.110000, 0.330000, 0.220000, -0.950000},
{0.000000, -0.290000, -0.130000, -0.110000, 0.000000, -1.440000},
{0.000000, -0.230000, -0.260000, -0.560000, 0.000000, -2.680000},
{-0.200000, -0.680000, -0.139999, -1.050000, 0.300000, -2.880000},
{0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
{0.000000, -0.410000, -0.150001, -0.380000, 0.180000, -1.390000},
{0.120000, 0.000000, 0.130000, 0.840000, -0.800000, 0.600000},
{1.200000, 1.469999, 0.390000, 4.590000, 0.770000, 10.940000},
{0.880000, 1.980000, -0.140000, -3.190000, -1.930000, 1.450000},
{0.000000, 0.520000, 0.170000, -1.090000, -0.100000, 0.139999},
{-0.260000, 0.130000, 0.170000, -0.530001, 0.140000, 0.340000},
{-0.270000, 0.110000, 0.100000, -0.360000, 0.290000, 0.290000},
{-0.240000, 0.000000, 0.000000, -0.360000, 0.000000, 0.110001},
{0.000000, 0.220000, 0.000000, 0.000000, 0.390000, 0.000000},
{0.000000, 0.000000, 0.000000, 0.120000, 0.230000, 0.000000},
{0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
{-0.180000, -0.230000, 0.000000, -0.110000, 0.000000, -0.440001},
{-0.230000, -0.430000, 0.000000, 0.350000, 0.000000, -0.610000},
{-0.230000, -0.640000, 0.000000, 0.559999, 0.260000, -1.060000},
{-0.250000, -0.620000, 0.000000, 0.830000, 0.830000, -0.480000},
{0.000000, -0.440000, -0.120000, 0.290000, 0.000000, -0.730000},
{0.000000, -0.400000, -0.130000, 0.000000, -0.130000, -1.509999},
{0.000000, -0.240000, 0.000000, 0.000000, 0.000000, -1.790000},
{-0.110000, -0.450000, -0.140000, -0.230000, -0.110000, -1.850000},
{0.000000, -0.290000, -0.129999, -0.290000, 0.390000, -1.900000},
{0.000000, -0.240000, -0.230000, -0.740001, 0.130000, -1.830000},
{0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
{0.000000, -0.360001, 0.000000, -0.580000, 0.000000, -1.310000},
{-0.100000, -0.219999, 0.000000, -0.309999, -0.370000, -0.350000},
{0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
{0.230000, 0.000000, 0.000000, 0.400000, 0.000000, 0.210000},
{0.190000, 0.380000, 0.000000, -0.120000, 0.000000, 0.000000},
{0.000000, 0.000000, 0.000000, 0.130000, -0.100000, 0.240000},
{0.000000, 0.000000, 0.000000, 0.000000, -0.180000, 0.130000},
{0.000000, 0.160000, 0.000000, 0.110000, 0.000000, 0.260000},
{0.250000, 0.280000, 0.000000, 0.360000, 0.000000, 0.450000},
{0.220000, 0.410000, 0.000000, 0.000000, -0.130000, 0.290000},
{0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
{0.000000, 0.170000, 0.000000, 0.190001, 0.000000, 0.420000},
{0.000000, 0.160000, 0.000000, 0.270000, 0.000000, 0.600000},
{0.000000, 0.140000, 0.000000, 0.000000, 0.000000, 0.170000},
{-0.140000, 0.000000, 0.000000, 0.250000, 0.160000, 0.450000},
{0.140000, 0.180000, 0.000000, 0.320000, -0.380000, 0.920000},
{0.180000, 0.270000, 0.000000, 0.000000, -0.300000, 0.660000},
{0.000000, 0.190000, 0.000000, 0.100000, 0.000000, 0.700000},
{0.130000, 0.270000, 0.130000, 0.210000, 0.000000, 1.060000},
{0.110000, 0.190000, 0.000000, 0.140000, -0.240000, 1.080000},
{0.000000, 0.210000, 0.000000, 0.190000, -0.340000, 1.760000},
{0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
```

```
265 {0.000000, 0.240000, 0.000000, -0.370000, -0.190000, 1.520000},
266 {0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
267 {-0.190000, 0.570000, 0.000000, -0.200000, 0.680000, 0.370000},
268 {0.000000, 0.100000, 0.000000, -0.920000, -0.720000, 1.700001},
269 {0.000000, 0.000000, 0.000000, -0.390000, -0.160000, 0.190000},
270 {0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
271 {0.000000, 0.000000, 0.000000, -0.610000, -0.270000, 0.110000},
272 {0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
273 {0.000000, 0.190000, 0.000000, 0.000000, 0.190000, 0.000000},
274 {0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
275 {-0.700000, -2.750000, -0.310000, -1.210000, 2.090000, -14.230000},
276 {-0.760000, -1.490000, -0.280000, 0.160000, 0.200000, 0.140000},
277 {-0.270000, -0.540000, -0.190000, -0.610000, -0.340000, -0.710000},
278 {0.000000, 0.000000, 0.000000, 0.240000, 0.430000, 0.000000},
279 {0.210000, 0.270000, 0.230000, 0.500000, -0.620000, 0.830000},
280 {0.190000, 0.500000, 0.330000, 0.570001, -0.450000, 1.180000},
281 {0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
282 {0.210000, 0.310000, 0.220000, 1.530000, -0.140000, 2.600000},
283 {0.300000, 0.610000, 0.000000, 0.190000, -0.530000, 1.150000},
284 {0.000000, 0.300000, 0.000000, 0.290000, 0.240000, 0.600000},
285 {0.000000, 0.240000, 0.000000, 0.250000, 0.000000, 1.090000},
286 {0.000000, 0.000000, 0.000000, 0.000000, 0.150000, 0.480000},
287 {0.140000, 0.180000, 0.000000, 0.350000, 0.000000, 1.310000},
288 {0.150000, 0.570000, 0.000000, 0.000000, 0.000000, 1.230000},
289 {0.000000, 0.340000, 0.000000, 0.000000, 0.170000, 1.000000},
290 {0.210000, 0.550000, 0.000000, -0.390000, -0.330000, 0.160000},
291 {0.000000, -0.160000, 0.000000, 0.190000, 0.160000, 0.840000},
292 {0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
293 {0.000000, 0.110000, 0.000000, -0.290000, -0.100000, 0.550000},
294 {0.000000, 0.260000, 0.000000, 0.000000, 0.000000, 0.690000},
295 {0.000000, 0.260000, 0.000000, -0.220000, -0.190000, 0.570001},
296 {0.000000, 0.120000, 0.000000, -0.590000, -0.380000, 0.460000},
297 {0.000000, 0.000000, -0.110000, -0.170000, 0.000000, 0.000000},
298 {0.000000, 0.160000, 0.000000, -0.290000, 0.000000, 0.000000},
299 {0.000000, 0.000000, 0.000000, -0.179999, -0.140000, 0.139999},
300 {0.000000, -0.260000, 0.000000, -0.240001, -0.210000, 0.100000},
301 {0.000000, 0.000000, 0.000000, 0.000000, 0.220000, 0.240000},
302 {0.000000, 0.000000, 0.000000, 0.000000, 0.260000, 0.100000},
303 {0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
304 {0.000000, 0.000000, 0.000000, 0.000000, 0.000000, -0.110000},
305
306 {0.230000, 0.000000, 0.000000, 0.400000, 0.000000, 0.210000},
307 {0.190000, 0.380000, 0.000000, -0.120000, 0.000000, 0.000000},
308 {0.000000, 0.000000, 0.000000, 0.130000, -0.100000, 0.240000},
309 {0.000000, 0.000000, 0.000000, 0.000000, -0.180000, 0.130000},
310 {0.000000, 0.160000, 0.000000, 0.110000, 0.000000, 0.260000},
311 {0.250000, 0.280000, 0.000000, 0.360000, 0.000000, 0.450000},
312 {0.220000, 0.410000, 0.000000, 0.000000, -0.130000, 0.290000},
313 {0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
314 {0.000000, 0.170000, 0.000000, 0.190001, 0.000000, 0.420000},
315 {0.000000, 0.160000, 0.000000, 0.270000, 0.000000, 0.600000},
316 {0.000000, 0.140000, 0.000000, 0.000000, 0.000000, 0.170000},
317 {-0.140000, 0.000000, 0.000000, 0.250000, 0.160000, 0.450000},
318 {0.140000, 0.180000, 0.000000, 0.320000, -0.380000, 0.920000},
319 {0.180000, 0.270000, 0.000000, 0.000000, -0.300000, 0.660000},
320 {0.000000, 0.190000, 0.000000, 0.100000, 0.000000, 0.700000},
321 {0.130000, 0.270000, 0.130000, 0.210000, 0.000000, 1.060000},
322 {0.110000, 0.190000, 0.000000, 0.140000, -0.240000, 1.080000},
323 {0.000000, 0.210000, 0.000000, 0.190000, -0.340000, 1.760000},
324 {0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
325 {0.000000, 0.240000, 0.000000, -0.370000, -0.190000, 1.520000},
326 {0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
327 {-0.190000, 0.570000, 0.000000, -0.200000, 0.680000, 0.370000},
328 {0.000000, 0.100000, 0.000000, -0.920000, -0.720000, 1.700001},
329 {0.000000, 0.000000, 0.000000, -0.390000, -0.160000, 0.190000},
330 {0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
```



```
331 {0.000000, 0.000000, 0.000000, -0.610000, -0.270000, 0.110000},
332 {0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
333 {0.000000, 0.190000, 0.000000, 0.000000, 0.190000, 0.000000},
334 {0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000},
335 {-0.700000, -2.750000, -0.310000, -1.210000, 2.090000, -14.230000},
336 };
337
338
339 /* Target Patterns */
340
341 double  s_target_pat[S_PATTERNS][S_NUM_OUT] = {
342 {0.000000, 0.000000},
343 {0.000000, 0.000000},
344 {0.000000, 0.000000},
345 {0.000000, 0.000000},
346 {0.000000, 0.000000},
347 {0.000000, 0.000000},
348 {0.000000, 0.000000},
349 {0.000000, 0.000000},
350 {0.000000, 0.000000},
351 {0.000000, 0.000000},
352 {0.020000, 0.000000},
353 {0.030000, 0.000000},
354 {0.040000, 0.000000},
355 {0.050000, 0.000000},
356 {0.050000, 0.000000},
357 {0.050000, 0.000000},
358 {0.060000, 0.000000},
359 {0.070000, 0.000000},
360 {0.050000, 0.000000},
361 {0.060000, 0.000000},
362 {0.100000, 0.000000},
363 {0.300000, 0.000000},
364 {0.300000, 0.000000},
365 {0.500000, 0.000000},
366 {0.300000, 0.000000},
367 {0.100000, 0.000000},
368 {0.000000, 0.000000},
369 {0.000000, 0.000000},
370 {0.000000, 0.000000},
371 {0.000000, 0.000000},
372 {0.000000, 0.000000},
373 {0.000000, 0.000000},
374 {0.000000, 0.000000},
375 {0.000000, 0.000000},
376 {0.000000, 0.000000},
377 {0.000000, 0.000000},
378 {0.000000, 0.000000},
379 {0.000000, 0.000000},
380 {0.000000, 0.000000},
381 {0.000000, 0.000000},
382 {0.000000, 0.000000},
383 {0.100000, 0.000000},
384 {0.100000, 0.000000},
385 {0.300000, 0.000000},
386 {0.400000, 0.000000},
387 {0.400000, 0.000000},
388 {0.400000, 0.000000},
389 {0.300000, 0.000000},
390 {0.200000, 0.000000},
391 {0.000000, 0.000000},
392 {0.000000, 0.000000},
393 {0.000000, 0.000000},
394 {0.000000, 0.000000},
395 {0.000000, 0.000000},
396 {0.000000, 0.000000},
```

```
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.040000, 0.000000},
{0.100000, 0.000000},
{0.100000, 0.000000},
{0.100000, 0.000000},
{0.150000, 0.000000},
{0.150000, 0.000000},
{0.300000, 0.000000},
{0.200000, 0.000000},
{0.200000, 0.000000},
{0.200000, 0.000000},
{0.100000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.050000},
{0.000000, 0.040000},
{0.000000, 0.030000},
{0.000000, 0.050000},
{0.000000, 0.040000},
{0.000000, 0.070000},
{0.000000, 0.050000},
{0.000000, 0.060000},
{0.000000, 0.100000},
{0.000000, 0.100000},
{0.000000, 0.100000},
{0.000000, 0.150000},
{0.000000, 0.200000},
{0.000000, 0.300000},
{0.000000, 0.300000},
{0.000000, 0.400000},
{0.000000, 0.300000},
{0.000000, 0.300000},
{0.000000, 0.150000},
{0.000000, 0.200000},
{0.000000, 0.200000},
{0.000000, 0.100000},
{0.000000, 0.100000},
{0.000000, 0.200000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},

{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.100000, 0.000000},
{0.100000, 0.000000},
{0.200000, 0.000000},
{0.300000, 0.000000},
{0.300000, 0.000000},
{0.250000, 0.000000},
{0.300000, 0.000000},
{0.200000, 0.000000},
{0.300000, 0.000000},
{0.300000, 0.000000},
{0.250000, 0.000000},
{0.100000, 0.000000},
{0.200000, 0.000000},
{0.200000, 0.000000},
```



```
{0.200000, 0.000000},
{0.200000, 0.000000},
{0.100000, 0.000000},
{0.050000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.050000},
{0.000000, 0.040000},
{0.000000, 0.030000},
{0.000000, 0.050000},
{0.000000, 0.040000},
{0.000000, 0.070000},
{0.000000, 0.050000},
{0.000000, 0.060000},
{0.000000, 0.100000},
{0.000000, 0.100000},
{0.000000, 0.100000},
{0.000000, 0.150000},
{0.000000, 0.200000},
{0.000000, 0.300000},
{0.000000, 0.300000},
{0.000000, 0.400000},
{0.000000, 0.300000},
{0.000000, 0.300000},
{0.000000, 0.150000},
{0.000000, 0.200000},
{0.000000, 0.200000},
{0.000000, 0.100000},
{0.000000, 0.100000},
{0.000000, 0.200000},
{0.000000, 0.000000},
{0.000000, 0.000000},
{0.000000, 0.000000},
};
/*
double  s_target_pat[S_PATTERNS][S_NUM_OUT] = {
{0.00000, 0.00000},
{0.00000, 0.00000},
{0.00000, 0.00000},
{0.100000, 0.00000},
{0.200000, 0.00000},
{0.500000, 0.00000},
{0.700000, 0.00000},
{0.800000, 0.00000},
{0.450000, 0.00000},
{0.300000, 0.00000},
{0.100000, 0.00000},
{0.00000, 0.00000},
{0.00000, 0.100000},
{0.00000, 0.500000},
{0.00000, 0.750000},
{0.00000, 0.300000},
{0.00000, 0.100000},
{0.00000, 0.00000},
{0.100000, 0.00000},
{0.400000, 0.00000},
{0.750000, 0.00000},
{0.750000, 0.00000},
{0.300000, 0.00000},
```

```
529 {0.00000, 0.00000},
530 {0.00000, 0.300000},
531 {0.00000, 0.600000},
532 {0.00000, 0.750000},
533 {0.00000, 0.600000},
534 {0.00000, 0.300000},
535 {0.00000, 0.00000},
536 {0.00000, 0.00000},
537 {0.100000, 0.00000},
538 {0.750000, 0.00000},
539 {0.750000, 0.00000},
540 {0.300000, 0.00000},
541 {0.00000, 0.00000},
542 {0.00000, 0.00000},
543 {0.00000, 0.200000},
544 {0.00000, 0.500000},
545 {0.00000, 0.500000},
546 {0.00000, 0.750000},
547 {0.00000, 0.750000},
548 {0.00000, 0.500000},
549 {0.00000, 0.300000},
550 {0.00000, 0.00000},
551 {0.00000, 0.00000},
552 {0.300000, 0.00000},
553 {0.500000, 0.00000},
554 {0.600000, 0.00000},
555 {0.400000, 0.00000},
556 {0.300000, 0.00000},
557 {0.00000, 0.00000},
558 {0.00000, 0.00000},
559 {0.00000, 0.300000},
560 {0.00000, 0.500000},
561 {0.00000, 0.750000},
562 {0.00000, 0.800000},
563 {0.00000, 0.300000},
564 {0.00000, 0.00000},
565 {0.200000, 0.00000},
566 {0.300000, 0.00000},
567 {0.500000, 0.00000},
568 {0.750000, 0.00000},
569 {0.600000, 0.00000},
570 {0.500000, 0.00000},
571 {0.300000, 0.00000},
572 {0.00000, 0.00000},
573 {0.00000, 0.200000},
574 {0.00000, 0.800000},
575 {0.00000, 0.00000},
576 {0.00000, 0.00000},
577 {0.00000, 0.00000},
578 {0.00000, 0.00000},
579 {0.200000, 0.00000},
580 {0.300000, 0.00000},
581 {0.400000, 0.00000},
582 {0.300000, 0.00000},
583 {0.200000, 0.00000},
584 {0.00000, 0.00000},
585 {0.00000, 0.00000},
586 {0.00000, 0.300000},
587 {0.00000, 0.300000},
588 {0.00000, 0.200000},
589 {0.00000, 0.00000},
590 {0.00000, 0.00000},
591 {0.200000, 0.00000},
592 {0.500000, 0.00000},
593 {0.500000, 0.00000},
594 {0.200000, 0.00000},
```



```
595 {0.00000, 0.00000},
596 {0.00000, 0.200000},
597 {0.00000, 0.500000},
598 {0.00000, 0.200000},
599 {0.00000, 0.00000},
600 {0.300000, 0.00000},
601 {0.500000, 0.00000},
602 {0.400000, 0.00000},
603 {0.200000, 0.00000},
604 {0.00000, 0.00000},
605 {0.00000, 0.00000},
606 {0.00000, 0.300000},
607 {0.00000, 0.400000},
608 {0.00000, 0.300000},
609 {0.00000, 0.100000},
610 {0.00000, 0.00000},
611 {0.00000, 0.00000},
612 {0.00000, 0.00000},
613 {0.00000, 0.00000},
614 {0.00000, 0.00000},
615 {0.500000, 0.00000},
616 {0.400000, 0.00000},
617 {0.00000, 0.00000},
618 {0.00000, 0.00000},
619 {0.00000, 0.00000},
620 {0.00000, 0.00000},
621 {0.00000, 0.750000},
622 {0.00000, 0.500000},
623 {0.00000, 0.00000},
624 {0.00000, 0.00000},
625 {0.00000, 0.00000},
626 {0.00000, 0.00000},
627 {0.00000, 0.00000},
628 {0.750000, 0.00000},
629 {0.400000, 0.00000},
630 {0.300000, 0.00000},
631 {0.200000, 0.00000},
632 {0.00000, 0.00000},
633 {0.00000, 0.00000},
634 {0.00000, 0.00000},
635 {0.00000, 0.300000},
636 {0.00000, 0.200000},
637 {0.00000, 0.100000},
638 {0.00000, 0.100000},
639 {0.00000, 0.00000},
640 {0.300000, 0.00000},
641 {0.750000, 0.00000},
642 {0.600000, 0.00000},
643 {0.450000, 0.00000},
644 {0.300000, 0.00000},
645 {0.00000, 0.00000},
646 {0.00000, 0.00000},
647 {0.00000, 0.600000},
648 {0.00000, 0.750000},
649 {0.00000, 0.300000},
650 {0.00000, 0.200000},
651 {0.00000, 0.00000},
652 {0.00000, 0.00000},
653 {0.00000, 0.00000},
654 {0.150000, 0.00000},
655 {0.450000, 0.00000},
656 {0.600000, 0.00000},
657 {0.750000, 0.00000},
658 {0.100000, 0.00000},
659 {0.00000, 0.00000},
660 {0.00000, 0.100000},
```

```
661 {0.00000, 0.200000},
662 {0.00000, 0.450000},
663 {0.00000, 0.600000},
664 {0.00000, 0.300000},
665 {0.00000, 0.200000},
666 }; */
667
668 double s_out_f(), s_delta_f_out(), s_delta_f_hid(), s_pattern_error();
669 double s_pattern_err[S_PATTERNS];
670 double s_rdm();
671 /*
672  * create input, hidden, output units (and threshold or bias unit)
673  */
674 s_create_processing_units(pu, s_tampon1, s_tampon2)
675 struct unit *pu[];
676 double s_tampon1[], s_tampon2[];
677 {
678   int   id;                       /* processing unit index */
679   struct unit *create_unit();
680
681   for (id = S_IN_UID(0); id < S_IN_UID(S_NUM_IN); id++)
682     pu[id] = create_unit(id, "input", 0., NULL, 0., NULL, 0.);
683   for (id = S_CON_UID1(0); id < S_CON_UID1(S_NUM_CON1); id++)
684     pu[id] = create_unit(id, "context1", 0., NULL, 0., NULL, 0.);
685   for (id = S_CON_UID2(0); id < S_CON_UID2(S_NUM_CON2); id++)
686     pu[id] = create_unit(id, "context2", 0., NULL, 0., NULL, 0.);
687   for (id = S_HID_UID(0); id < S_HID_UID(S_NUM_HID); id++)
688     pu[id] = create_unit(id, "hidden", 0., s_out_f, 0., s_delta_f_hid, s_rdm());
689   for (id = S_OUT_UID(0); id < S_OUT_UID(S_NUM_OUT); id++)
690     pu[id] = create_unit(id, "output", 0., s_out_f, 0., s_delta_f_out, s_rdm());
691   for (id = 0; id < S_NUM_CON1; id++) {
692     s_tampon1[id] = 0.5;
693     s_tampon2[id] = 0.5;
694   }
695 }
696
697 /*
698  * create links - fully connected for each layer
699  *                note: the bias unit has one link to ea hid and out unit
700  */
701 s_create_in_out_links(pu, s_tampon1, s_tampon2)
702 struct unit *pu[];
703 double s_tampon1[], s_tampon2[];
704 {
705   int   i, j, k;                  /* i == to and j == from unit id's */
706   struct link *create_link();
707
708   /* fully connected units */
709   for (i = S_HID_UID(0); i < S_HID_UID(S_NUM_HID); i++) { /* links to hidden */
710     for (j = S_IN_UID(0); j < S_IN_UID(S_NUM_IN); j++) /* from input units */
711       pu[j]->outlinks =
712         pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->outlinks, j,
713                                      (char *)NULL, s_rdm(), 0.);
714     for (j = S_CON_UID1(0); j < S_CON_UID1(S_NUM_CON1); j++)
715       pu[j]->outlinks =
716         pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->outlinks, j,
717                                      (char *)NULL, s_rdm(), 0.);
718     for (j = S_CON_UID2(0); j < S_CON_UID2(S_NUM_CON2); j++)
719       pu[j]->outlinks =
720         pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->outlinks, j,
721                                      (char *)NULL, s_rdm(), 0.);
722   }
723   for (i = S_OUT_UID(0); i < S_OUT_UID(S_NUM_OUT); i++) {       /* links to outp
    ut */
724     for (j = S_HID_UID(0); j < S_HID_UID(S_NUM_HID); j++) /* from hidden units *
    /
```



```
725       pu[j]->outlinks =
726         pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->outlinks, j,
727                                      (char *)NULL, s_rdm(), 0.);
728   }
729 }
730
731
732 s_learn(pu, s_tampon1, s_tampon2)
733 struct unit *pu[];
734 double s_tampon1[], s_tampon2[];
735 {
736   int j, k;
737   register i, temp;
738   char   tempstr[BUFSIZE];
739   extern int    iterations;
740   static char prompt[] = "Enter # iterations (default is 8000) => ";
741   static char quote1[] = "I should do more learning.\n";
742
743   printf(prompt);
744   fflush(stdin); fflush(stdout);
745   gets(tempstr);
746   if (temp = atoi(tempstr))
747     iterations = temp;
748
749   printf("\nLearning ");
750   for (i = 0; i < iterations; i++) {
751     if ((i % NOTIFY) == 0) {
752       printf(".");
753       fflush(stdout);
754     }
755     if (i > iterations - S_PATTERNS)
756       s_bp_learn(pu, s_tampon1, s_tampon2, 1);
757     else
758       s_bp_learn(pu, s_tampon1, s_tampon2, 0);
759   }
760   printf(" Done\n\n");
761   for (j = 0; j < S_PATTERNS; j=j+2)
762     printf("Error for pattern %d = \t%lf\tError for pattern %d =\t%lf\n", j, s_p
    attern_err[j], j+1, s_pattern_err[j+1]);
763
764   for (j = 2; j < S_PATTERNS; j++) {
765     if (s_pattern_err[j] > 0.5) {
766       printf("\nI don't know pattern %d very well.\n%s",j, quote1);
767     }
768     if (j == S_PATTERNS - 1) {
769       printf("\nOK. I know all the patterns quite well, now.\n");
770       break;
771     }
772   }
773 }
774
775 /* back propagation learning */
776 s_bp_learn(pu, s_tampon1, s_tampon2, save_error)
777 struct unit *pu[];
778 double s_tampon1[], s_tampon2[];
779 int    save_error;
780 {
781   int i;
782   static int count = 0;
783   static int pattern = 0;
784   extern int iterations;
785   extern double s_pattern_err[S_PATTERNS];
786
787   s_init_input_units(pu, s_tampon1, s_tampon2, pattern); /* initialize input pat
    tern to
788                                                          learn */
```

```
789   s_propagate(pu, s_tampon1, s_tampon2);                    /* calc outputs to ch
    eck versus
790                                                    targets */
791   if (save_error)
792     s_pattern_err[pattern] = s_pattern_error(pattern, pu);
793   s_bp_adjust_weights(pattern, pu);
794   if (pattern < S_PATTERNS - 1)
795     pattern++;
796   else
797       pattern = 0;
798   count++;
799
800   if (count == iterations) {
801      for (i = 0; i < S_NUM_CON1; i++) {
802         s_tampon1[i] = 0.5;
803         s_tampon2[i] = 0.5;
804      }
805      count = 0;
806   }
807 }
808
809 /* initialize the input units with a specific input pattern to learn */
810 s_init_input_units(pu, s_tampon1, s_tampon2, pattern)
811 struct unit *pu[];
812 double s_tampon1[], s_tampon2[];
813 int    pattern;
814 {
815   int   id;
816
817   for (id = S_IN_UID(0); id < S_IN_UID(S_NUM_IN); id++)
818     pu[id]->output = s_input_pat[pattern][id];
819   for (id = S_CON_UID1(0); id < S_CON_UID1(S_NUM_CON1); id++)
820     pu[id]->output = s_tampon1[id - S_CON_UID1(0)];
821   for (id = S_CON_UID2(0); id < S_CON_UID2(S_NUM_CON2); id++)
822     pu[id]->output = s_tampon2[id - S_CON_UID2(0)];
823 }
824
825 /* calculate the activation level of each unit */
826 s_propagate(pu, s_tampon1, s_tampon2)
827 struct unit *pu[];
828 double s_tampon1[], s_tampon2[];
829 {
830   int   id;
831
832   for (id = S_HID_UID(0); id < S_HID_UID(S_NUM_HID); id++)
833     (*(pu[id]->unit_out_f))(pu[id], pu);
834   for (id = S_OUT_UID(0); id < S_OUT_UID(S_NUM_OUT); id++)
835     (*(pu[id]->unit_out_f))(pu[id], pu);
836   for (id = 0; id < S_NUM_CON1; id++)
837     s_tampon2[id] = s_tampon1[id];
838   for (id = S_HID_UID(0); id < S_HID_UID(S_NUM_HID); id++)
839     s_tampon1[id - S_HID_UID(0)] = pu[id]->output;
840 }
841
842 /* function to calculate the activation or output of units */
843 double
844 s_out_f(pu_ptr, pu)
845 struct unit *pu_ptr, *pu[];
846 {
847   double sum = 0., exp();
848   struct link *tmp_ptr;
849
850   tmp_ptr = pu_ptr->inlinks;
851   while (tmp_ptr) {
852     /* sum up (outputs from inlinks times weights on the inlinks) */
853     sum += pu[tmp_ptr->from_unit]->output * tmp_ptr->weight;
```



```
854     tmp_ptr = tmp_ptr->next_inlink;
855   }
856   sum += pu_ptr->threshold;
857   pu_ptr->output = 1./(1. + exp(-sum));
858 }
859
860 /* half of the sum of the squares of the errors of the
861    output versus target values */
862 double
863 s_pattern_error(pat_num, pu)
864 int     pat_num;        /* pattern number */
865 struct unit *pu[];
866 {
867   int           i;
868   double        temp, sum = 0.;
869
870   for (i = S_OUT_UID(0); i < S_OUT_UID(S_NUM_OUT); i++) {
871     temp = s_target_pat[pat_num][S_TARGET_INDEX(i)] - pu[i]->output;
872     sum += temp * temp;
873   }
874   return (sum/2.);
875 }
876
877 s_bp_adjust_weights(pat_num, pu)
878 int     pat_num;        /* pattern number */
879 struct unit *pu[];
880 {
881   int           i;              /* processing units id */
882   double        temp1, temp2, delta, error_sum;
883   struct link   *inlink_ptr, *outlink_ptr;
884
885   /* calc deltas */
886   for (i = S_OUT_UID(0); i < S_OUT_UID(S_NUM_OUT); i++) /* for each output unit
    */
887     (*(pu[i]->unit_delta_f))(pu, i, pat_num); /* calc delta */
888   for (i = S_HID_UID(0); i < S_HID_UID(S_NUM_HID); i++) /* for each hidden unit
    */
889     (*(pu[i]->unit_delta_f))(pu, i);      /* calc delta */
890   /* calculate weights and thresholds */
891   for (i = S_OUT_UID(0); i < S_OUT_UID(S_NUM_OUT); i++) {       /* for output un
    its */
892     inlink_ptr = pu[i]->inlinks;
893     while (inlink_ptr) {        /* for each inlink to output unit */
894       temp1 = LEARNING_RATE * pu[i]->delta *
895         pu[inlink_ptr->from_unit]->output;
896       temp2 = MOMENTUM * inlink_ptr->data;
897       inlink_ptr->data = temp1 + temp2; /* new delta weight */
898       inlink_ptr->weight += inlink_ptr->data;   /* new weight */
899       inlink_ptr = inlink_ptr->next_inlink;
900     }
901     pu[i]->threshold += LEARNING_RATE * pu[i]->delta;    /* new threshold */
902   }
903   for (i = S_HID_UID(0); i < S_HID_UID(S_NUM_HID); i++) { /* for each hid unit *
    /
904     inlink_ptr = pu[i]->inlinks;
905     while (inlink_ptr) {        /* for each inlink to output unit */
906       temp1 = LEARNING_RATE * pu[i]->delta *
907         pu[inlink_ptr->from_unit]->output;
908       temp2 = MOMENTUM * inlink_ptr->data;
909       inlink_ptr->data = temp1 + temp2; /* new delta weight */
910       inlink_ptr->weight += inlink_ptr->data;   /* new weight */
911         inlink_ptr = inlink_ptr->next_inlink;
912     }
913     pu[i]->threshold += LEARNING_RATE * pu[i]->delta;    /* new threshold */
914   }
915 }
```

```
916
917 /* calculate the delta for an output unit */
918 double
919 s_delta_f_out(pu, uid, pat_num)
920 struct unit *pu[];
921 int     uid, pat_num;
922 {
923   double        temp1, temp2, delta;
924
925   /* calc deltas */
926   temp1 = (s_target_pat[pat_num][S_TARGET_INDEX(uid)] - pu[uid]->output);
927   temp2 = (1. - pu[uid]->output);
928   delta = temp1 * pu[uid]->output * temp2; /* calc delta */
929   pu[uid]->delta = delta; /* store delta to pass on */
930 }
931
932 /* calculate the delta for a hidden unit */
933 double
934 s_delta_f_hid(pu, uid)
935 struct unit *pu[];
936 int     uid;
937 {
938   double        temp1, temp2, delta, error_sum;
939   struct link   *inlink_ptr, *outlink_ptr;
940
941   outlink_ptr = pu[uid]->outlinks;
942   error_sum = 0.;
943   while (outlink_ptr) {
944     error_sum += pu[outlink_ptr->to_unit]->delta * outlink_ptr->weight;
945     outlink_ptr = outlink_ptr->next_outlink;
946   }
947   delta = pu[uid]->output * (1. - pu[uid]->output) * error_sum;
948   pu[uid]->delta = delta;
949 }
950
951 /* save weight of all the connections and threshold of hidden and output units *
    /
952 s_save_weights(pu)
953 struct unit *pu[];
954 {
955   int i, j, k = 0, l = 0;
956   double table_weights[(S_NUM_IN + S_NUM_CON1 + S_NUM_CON2)*S_NUM_HID + S_NUM_HI
    D*S_NUM_OUT + S_NUM_HID + S_NUM_OUT];          /* size = total number of links
    + number of thresholds; only */
957                         /* hidden and output units have a threshold
       */
958   FILE *fiche, *errf;
959   static char file_name[] = "S_WEIGHTS_FILE";
960   static char err_file[] = "S_ERROR_FILE";
961   struct link *tmp_ptr;
962
963
964   for (i = S_HID_UID(0); i < S_HID_UID(S_NUM_HID); i++) {
965     tmp_ptr = pu[i]->inlinks;
966     table_weights[k*(S_NUM_IN + S_NUM_CON1 + S_NUM_CON2 + 1)] = pu[i]->threshold
    ;
967     j = (k + 1)*(S_NUM_IN + S_NUM_CON1 + S_NUM_CON2 + 1) - 1;
968     while (tmp_ptr) {
969         table_weights[j] = tmp_ptr->weight;
970         j--;
971         tmp_ptr = tmp_ptr->next_inlink;
972     }
973     k++;
974   }
975
976   for (i = S_OUT_UID(0); i < S_OUT_UID(S_NUM_OUT); i++) {
```



```
977     tmp_ptr = pu[i]->inlinks;
978     table_weights[k*(S_NUM_IN + S_NUM_CON1 + S_NUM_CON2 + 1) + l*(S_NUM_HID + 1)
    ] = pu[i]->threshold;
979     j = k*(S_NUM_IN + S_NUM_CON1 + S_NUM_CON2 + 1) + (l + 1)*(S_NUM_HID + 1) -1;
980     while (tmp_ptr) {
981       table_weights[j] = tmp_ptr->weight;
982         j--;
983         tmp_ptr = tmp_ptr->next_inlink;
984     }
985     l++;
986   }
987
988   fiche = fopen(file_name, "w");
989   for (i = 0; i < (S_NUM_IN + S_NUM_CON1 + S_NUM_CON2)*S_NUM_HID + S_NUM_HID*S_N
    UM_OUT + S_NUM_HID + S_NUM_OUT; i++)
990       fprintf(fiche, "%lf\n", table_weights[i]);
991   fclose(fiche);
992   errf = fopen(err_file, "w");
993   for (i = 0; i < S_PATTERNS; i++)
994     fprintf(errf, "%lf\n", s_pattern_err[i]);
995   fclose(errf);
996
997 }
998
999 s_load_weights(pu)
1000 struct unit* pu[];
1001 {
1002   int i, j;
1003   double x;
1004   FILE *fp;
1005   static char name[] = "S_WEIGHTS_FILE";
1006
1007   if ((fp = fopen(name, "r")) == NULL) {
1008      fprintf(stderr, "\nCan't load the file -[%s]\n", name);
1009      exit(1);
1010   }
1011
1012   for (i = S_HID_UID(0); i < S_HID_UID(S_NUM_HID); i++) {
1013    fscanf(fp, "%lf", &x);
1014    pu[i]->threshold = x;
1015
1016    for (j = S_IN_UID(0); j < S_IN_UID(S_NUM_IN); j++) {
1017     fscanf(fp, "%lf", &x);
1018     pu[j]->outlinks = pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->out
    links, j,
1019                                                    (char *)NULL, x, 0.);
1020    }
1021
1022    for (j = S_CON_UID1(0); j < S_CON_UID1(S_NUM_CON1); j++) {
1023     fscanf(fp, "%lf", &x);
1024     pu[j]->outlinks = pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->out
    links, j,
1025                                                    (char *)NULL, x, 0.);
1026    }
1027
1028    for (j = S_CON_UID2(0); j < S_CON_UID2(S_NUM_CON2); j++) {
1029     fscanf(fp, "%lf", &x);
1030     pu[j]->outlinks = pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->out
    links, j,
1031                                                    (char *)NULL, x, 0.);
1032    }
1033   }
1034
1035   for (i = S_OUT_UID(0); i < S_OUT_UID(S_NUM_OUT); i++) {
1036    fscanf(fp, "%lf", &x);
1037    pu[i]->threshold = x;
```

```
1038
1039     for (j = S_HID_UID(0); j < S_HID_UID(S_NUM_HID); j++) {
1040      fscanf(fp, "%lf", &x);
1041      pu[j]->outlinks = pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->out
     links, j,
1042                                                     (char *)NULL, x, 0.);
1043     }
1044    }
1045
1046    fclose(fp);
1047 }
1048
1049 double
1050 s_rdm()
1051 {
1052    return((rand() % 50000 - 25000)/ 25000.0);
1053 }
```