画像認識の適用性向上のための自動知識獲得と並列化に関する研究

有田, 大作
九州大学システム情報知能システム工学

https://doi.org/10.11501/3166851

出版情報：九州大学, 1999, 博士（工学）, 課程博士
バージョン：
権利関係：

画像認識の適応性向上のための自動知識獲得と並列化に関する研究

有田大作

画像認識の適応性向上のための
自動知識獲得と並列化に関する研究

平成 12 年 2 月

有田 大作

目 次

図目次

表 目 次

第 1 章

序論

1.1 はじめに

「百聞は一見に如かず」ということわざがあるように，人間は外部からの情報の大部分を視覚によって獲得している．このことから，計算機によって視覚情報処理を実現しようという画像認識の研究は 1960 年代から行われてきた．しかし，この研究にはいまだ多くの研究者が取り組んでいる．画像認識を困難にしている点は多くあるが，そのうちの二つの問題として知識構築と計算量の問題が挙げられる．

知識構築の問題 画像認識のためには対象物に関する知識が必要であり，この知識をどうやって構築するのかという問題である．従来の画像認識システムでは，知識の構築をシステム設計者が行っていたが，画像認識システムの適応範囲を拡げるたびに，対象物に関するを構築しなければならず，そのコストが大きなものになっていた．

計算量の問題 画像は情報量が大きいため，それを処理するための計算量が多くなってしまうという問題である．高度で複雑な画像処理を行おうとすればするほど，計算量は多くなってしまい，現実的な処理時間では処理が終了しなくなってしまう．

このような背景から，知識の自動獲得や画像処理の並列化に関する研究が行われてきた．本研究も画像認識のこれらの問題点の解決法を示すものである．

本章では，まず画像認識におけるこれらの問題点を明らかにするために，画像認識における処理のレベルについて述べる．つぎに，画像認識における知識獲得と並列画像処理の

それぞれについて，考え方，現状，応用例，問題点について述べる．そして，その上で本研究の位置付けについて述べる．

1.2 画像認識における画像処理のレベル

画像認識に必要な画像処理は，情報の抽象度や処理方法により以下の三つに大別される．

低レベル画像処理 画素を処理対象とし，画像またはトークンを出力する画像処理を低レベル画像処理と呼ぶ．画像処理におけるトークンとは，領域やエッジ点などの画像中の意味のある要素である．画像を出力する画像処理の例としては，平滑化処理，先鋭化処理，オプティカルフロー処理などが挙げられる．また，トークンを出力する画像処理の例としては，2 値化処理，領域分割処理，エッジ検出処理などが挙げられる．さらに，エッジ点をまとめて直線や曲線を出力するようなトークンを処理対象としてトークンを出力する画像処理も低レベル画像処理に含める．

中間レベル画像処理 トークンを処理対象とし，トークンの特徴を抽出しシンボルを出力する画像処理を中間レベル画像処理と呼ぶ．画像処理におけるシンボルとは，あらかじめ計算機が保持しているシンボルの特徴を基にシンボルのラベルをトークンに付けたものである．トークンの特徴を抽出する画像処理の例としては，細線化，骨格化，モーメント計算などの領域を対象としたもの，直線抽出，曲率の計算，チェイン符号の計算のなどのエッジを対象としたものが挙げられる．シンボルを出力とする画像処理はテンプレートマッチングやパターン認識などの認識処理であり，計算機があらかじめ保持しているシンボルの特徴とトークンの特徴とのマッチングを行い，トークンにシンボルのラベルを付与する．

高レベル画像処理 シンボルを処理対象とし，シンボル間の関係を利用して，対象物の認識を行う画像処理を高レベル画像処理と呼ぶ．中間レベル画像処理によって得られたシンボル情報にはあいまいな部分や誤った部分も含まれることが多いので，高レベル処理と中間レベル処理がインタラクションを行うことによってあいまい性や誤りを減らす必要がある．

次節以降では，以上の処理レベルの分類に基づき，知識獲得や並列化のポイントを整理する．

1.3 画像認識のための知識獲得

まず，画像認識のために必要な知識のうち，設計者が何を明示的にシステムに与え，何をシステムが自動的に獲得するかという点が重要である．言い換えれば，対象物に依存する知識と依存しない知識の切り分けを明確にすることであるが，システムの適用範囲にもよるため一般的な議論は容易ではない．そこで，まず，1.2 節で述べた画像認識の階層性に基づいて画像認識に必要な知識について考察する．

低レベル画像処理における知識 このレベルでは，画素からトークンを抽出する．このためには以下の知識が必要である．

(知識 a) 何をトークンとするのか

これは，対象物を構成するトークンとしては何が良いのかという知識である．対象物によって，領域が良いものもあれば，エッジ点が良いものもある．したがって，この知識は対象物によって異なる知識である．一方，システム設計の観点から考えると，トークンは画像処理によって画像から抽出するため，利用するトークンが増えると，必要な画像処理の計算量が増大するという問題を引き起こす．したがって，利用トークンについては，なんらかの制限を加える必要がある．[1]

中間レベル画像処理 このレベルでは，トークンとシンボルの対応をとる．このためには以下の知識が必要である．

(知識 b) どのような特徴情報が必要なのか

これは，画像処理によって得られたトークンが，対象物を構成するシンボルであるかどうかを判断する特徴情報についての知識である．シンボルによって，例えば色情報が良いものもあれば，形状情報が良いものもある．したがって，この知識は対象物によって異なる知識である．一方，低レベル画像処理の知識と同様に，さまざまな特徴情報を計算すると計算量が増大するため，何らかの制限が必要である．

[1] そもそも抽出処理プログラムが用意されているトークンしか利用できないという本質的な問題もあるが，ここではその点については議論しない．

(知識 c) 対象物を構成するのはどのようなトークンか

これは，対象物を構成する各トークンの属性を示す知識である．対象物によって，どのような属性値を持つトークンで構成されるか異なるので，対象物によって異なる知識である．

高レベル画像処理 このレベルでは，シンボル間の関係を利用して対象物の認識を行う．このためには以下の知識が必要である．

(知識 d) シンボル間の関係はどのようになっているのか

これは，対象物を構成するシンボルの構造についての知識である．通常，シンボルの構造は対象物によって異なるので，この知識は対象物によって異なる知識となる．ただし，文字認識のような単純な対象についてはトークンの識別だけで認識を達成できることもあり，その場合は高レベル画像処理は必要ない．

従来の画像認識における知識獲得の研究では，**(知識 c)** のみを獲得するものがほとんどで，**(知識 a)** と **(知識 b)** はシステム設計者が暗黙的に与え，**(知識 d)** は対象としていなかった．また，人工神経回路網を利用した画像認識では，これらすべての知識を自動的に獲得することを目指している．しかし，**(知識 a)** と **(知識 b)** に制限を与えていないのでそれらの組み合わせ数が膨大になってしまい，計算量が膨大になってしまうだけでなく，局所解に陥ることで正しくない知識を獲得してしまうことも多い．

1.4 画像処理の並列化

並列処理を行う際のポイントは，「何を並列化の対象とするか」ということである．並列化の対象によって，処理対象を並列するデータ並列処理方式と処理手順を並列化するプログラム並列処理方式が考えられる．プログラム並列処理方式は更に，単一のデータの流れに対して複数のプログラムを連続的に実行するパイプライン並列処理方式と，単一のデータをブロードキャストして複数のプログラムによって並列処理する機能並列処理方式に分けられる．並列化によって処理を高速化するためには，目的の処理に最適な並列処理方式を選択することが重要である．以下では，並列処理の各方式について，効率的に実装可能な画像認識処理について述べる．

1.4.1 データ並列処理方式

データ並列処理方式により効率的に高速化できる画像認識の各レベルの処理を整理すると以下のようになる.

低レベル画像処理

- フィルタ処理
  フィルタ処理は，注目画素そのもの，またはその近傍の画素を基に行う画像処理であり，平滑化，先鋭化，2 値化，エッジ検出，モルフォロジー演算などが含まれる．処理対象が画素であり，さらに全画素をなぞるように処理する形態のため，画素を処理単位とするデータ並列処理手法による並列処理に向いている．すでに画像処理プロセッサとして商用化されているものも多い.
- 領域分割処理
  領域分割処理には大きく分けて，処理開始時は画像全体を一つの領域としてそこから領域を分割していくトップダウン的な方法と，処理開始時は各画素を一つの領域としてそこから領域を統合していくボトムアップ的な方法がある．その中で並列処理に適しているのはボトムアップ的な手法である．並列処理単位を画素とし，領域に変化がなくなるまで繰り返し全画素をなぞるように処理することで領域分割を行う.

中間レベル画像処理

- トークンを並列処理単位とする画像処理
  トークンの特徴を計算する処理，例えば領域のモーメント計算，細線化や，曲線のチェインコードや曲率の計算などは，トークンを処理単位として並列化できる.
- シンボルを並列処理単位とする画像処理
  トークンとシンボルの対応をとるとき，シンボルが複数ある場合は，シンボルを処理単位として並列化できる [1].
- ハフ変換
  ハフ変換は点をトークンとし，トークンの並びの特徴を抽出する処理である．この場合，トークンを処理単位とした並列化も可能であるが，それ以外にも求め

ようとしているパラメータを処理単位とした並列化も可能であり，しかも後者の方が高性能である場合が多い [2].

高レベル画像処理

このレベルの画像処理は，処理対象はシンボルであり，シンボルを処理単位として並列化することができる．ただし，中間レベル画像処理とのインタラクションのことを考慮する必要がある．

また，複数のセンサーを複数の計算機に接続し画像処理を行う場合は，複数のセンサーで同時に獲得された画像データを各画像ごとに処理していると考えることができるので，これもデータ並列処理方式による並列処理である．この場合，並列処理の単位は画像であり，その間の独立性が高いため，効率的な並列処理が可能な場合が多い．具体的にはステレオ処理 (視差計算)，視体積交差法 [3, 4] などの画像処理が挙げられる．

1.4.2 パイプライン並列処理方式

パイプライン並列処理方式では，処理データが連続的に生成される必要があるため，動画像処理の並列化に利用されるのが一般的である．動画像処理の各処理段階をパイプライン状に並べ，その上を動画像データが流れていくように処理が行われる．各処理段階では独立した画像処理を行うので，各処理段階で 1.4.1 節で述べたような並列化を行うことも可能である．

1.4.3 機能並列処理方式

機能並列処理方式による並列化では，並列に実行される処理間の独立性が高いため，並列化は容易である．しかし，最も時間のかかる処理で全体の処理時間が決まってしまうので，各プロセッサの負荷にかたよりがあると効率的な並列化ができない．機能並列処理方式による並列画像処理の例としては以下のようなものが考えられる．

- 同じカテゴリーの処理を並列に実行し，処理結果を統合することで，精度の向上を目指す処理．例えば，エッジ検出処理と領域分割処理，テンプレートマッチングベースのステレオ処理と特徴点ベースのステレオ処理を同じデータに対して同時に実行し，結果を統合する処理が考えられる．

- 異なるカテゴリーの処理を並列に実行し，処理結果を統合することで，より多くの情報獲得を目指す処理．例えば，物体の形状情報を獲得する処理と色情報を獲得する処理を並列に実行し，結果を統合することで，物体の形状情報と色情報を持った物体モデルを構築する処理が考えられる．

1.5 研究の目的

これまで述べてきたように，画像認識のための知識獲得の研究は，複雑な構造を持たない単純なパターン分類型のものであった．しかし，これでは複雑な構造を持つ対象物については，対象物に関する適切な知識を構築することはできない．本研究の第1の目的は，自動獲得する知識とシステムに与える事前知識の切り分けについて以下のように考え，例題を与えることによって複雑な構造を持つ対象物の知識を獲得する，汎用的かつ高精度な画像認識を実現することである．

知識 (a) 対象物を構成するトークンを領域に限定する．

知識 (b) あらかじめシステム設計者によって与えられた特徴情報の中から，そのトークンに必要なものを自動的に選択する．

知識 (c) 例題から自動的に獲得する．

知識 (d) 例題から自動的に獲得する．

一方，並列画像処理についてはさまざまな研究が行われてきており，すでに解決している部分も多い．しかし，以下に挙げる画像処理の並列化については未解決である．したがって，本研究ではこれらの画像処理の並列化の手法とそれに適した並列処理環境を示すことを第2の目的とする．

高レベル画像処理の並列化 高レベル画像処理は，中間レベル画像処理からの出力であるシンボルを入力として処理を行う．しかし，このシンボルの情報にあいまいさが含まれることが多い．そのため，高レベル画像処理と中間レベル画像処理とがインタラクションを行いながらこのあいまいさを減らす必要がある．つまり，中間レベル画像処理によるトークンとシンボルの対応をボトムアップの制約，高レベル画像処理によるシンボル間の整合をトップダウンの制約とし，それらの制約を満たす解を探す必要が

ある．このような処理を実現するために，本研究ではマルチエージェント型の並列処理を導入する．これは，トークンやシンボルをエージェントとし，それらを単位とする並列処理を行うことで，全体として解を求めるというものである．

実時間画像処理の並列化 実時間画像処理には，次々と獲得される画像データを処理するために，高速実行が要求される．1台の計算機の計算能力は限られたものなので，高度な処理を行うためには並列処理が必要である．さらに，近年要求が高まってきている複数のセンサーを利用した実時間画像処理のためには，計算能力だけではなく，I/O能力も要求される．このため，複数のセンサーを複数の計算機に接続することでI/Oの負荷を分散させることが可能な，分散型並列計算機が注目されている．本研究では，分散型並列計算機上での実時間画像処理について，そのために必要な機能の実現を目指す．具体的には，高速データ転送機構，同期機構，エラー処理機構である．また，これらの機能をアプリケーションプログラマが容易に利用できるように，プログラミング環境の開発も行う．

1.6 論文の構成

本論文は全6章から成る．第2章では，提案する知識獲得法について述べる．そして，第3章では，高レベル画像処理の並列化の例として，第2章の処理の一部を並列化し，その有効性を示す．また，第4章で，実時間画像処理の並列化の実現に必要な機能について述べ，第5章でそれらの機能を提供する並列動画像処理プログラミング環境について述べる．最後に，第6章で本研究のまとめと将来の展望について述べる．

第 2 章

画像認識における知識獲得

2.1 目的

画像中のどこに対象物があるかを探索したり，画像中の対象物が何であるかを認識したりするシステムを構築するためには，対象物に関する知識をあらかじめシステムが持っている必要がある．これまで提案されている多くの画像認識システムでは，この知識をシステム開発者が構築している．しかし，画像認識をさまざまな分野に利用しようとしたとき，システム開発者がひとつひとつの対象物に関する知識を構築することは困難である．そこで，対象物に関する知識をシステムが自動的に獲得するというアプローチが重要となってくる．本章では，この問題についての解決法を示し，知識の記述法，知識の獲得法，知識を利用した対象物探索法について述べる [5, 6, 7, 8].

画像認識に必要な知識を自動的に獲得する方法として，例題をシステムに与えることにより知識を獲得することが考えられる．そこで本研究では，ユーザが例題として「対象物の名前」および「画像上での対象物の位置」を入力することによって，システムがその対象物に関する知識を獲得し，その知識を利用して対象物の探索・認識を行う方式を開発することを目指す.

本研究では 3 次元物体が写っているカラー画像を対象として，例として示される複数の画像 (例示画像) から対象物に関する知識 (対象物モデル) を獲得する．具体的には，1 枚の例示画像から一つの対象物モデル (これをデータモデルと呼ぶ) を作成し，複数のデータモデルを比較することでより一般的な対象物モデル (これを知識モデルと呼ぶ) を獲得する．つまり知識モデルは，対象物クラスに一般化されたものになる．また，この対象物モデルを

利用して，カラー画像中から対象物を探索する．対象物クラスの対象物モデルを利用しているので，知識獲得時には与えていない個体についても探索可能である．例示画像は，対象物のみが存在している (実際は，画像中の対象物の領域をユーザが指定する) ことを仮定するが，探索時には対象物が切り出されていることを仮定しない．

2.2 他研究との比較

対象物に関する知識を自動的に獲得するシステムとしてはいくつか方法が提案されているが，それぞれ以下のような問題がある．

- Connell らの研究 [9]，秋山らの研究 [10]
  これは，主に 2 次元の線画や輪郭線画像を対象とするものであり，トークンとしては線で囲まれた領域を用いている．したがって，3 次元物体の実画像に対しては，トークンを安定に抽出することが保証できず，適用し難い面がある．本研究は実画像における領域をトークンとして用い，領域分割における問題点を解決した階層的な対象物モデルを作成している．

- Harvey らの SPAM[11]
  これは，領域をトークンとし，ユーザが画像上でトークンを指し示し，対象物を構成するシンボルを教える．このようにして対象物を構成するシンボルの構造に関する知識を獲得し，対象物を認識できるようになるものである．しかし，この方法では知識獲得時にユーザが指定したトークンが，認識時の領域分割において単一領域として抽出されなければならず，この条件を満たすようなトークンを選択することが非常に難しいという問題点がある．本研究では，ユーザの作業を最小限に抑え負担が少ないだけでなく，システムが必要なトークンを自動的に選択することによって，上記の問題がないトークンを選ぶことが可能になる．

- 村瀬らの研究 [12]
  これは，画像を固有ベクトル空間上で表現し，認識・学習しようとするものであり，単純な方法で，ある特定の対象の姿勢の認識に比較的良好な結果を得ている興味深いものである．ここでは，トークンは画像そのものであり，トークンの特徴情報は画像の固有ベクトルである．トークンの記述力を高めることで，トークンの識別のみで対

象物を認識できるようにし，高レベルの画像処理を不要としている．しかし，これは個々の対象物の認識を目指すものであり，本研究のように対象物に関する知識を一般化し対象物クラスの認識を目指すものではない．

2.3 対象物モデル

本研究における対象物モデルは，トークンとして領域を利用し，領域の階層構造を表す分割木によってシンボルの構造を記述する．トークンの特徴情報としては領域の色特徴，位置・大きさ特徴，形状特徴を利用する．これらの領域特徴は，各領域が持つ特徴テーブルに保持されている．本節ではこの分割木と特徴テーブルについて述べる．

2.3.1 分割木

画像の領域とは「画像上で隣接している画素値が類似している画素の集合」のことであり，領域分割は，画像を画素値が類似している画素の集合に分けることである．画像中の対象物は一つ以上の領域から成っており，領域の特徴と隣接関係によって対象物を認識しようという試みが数多く行われてきた．しかし，領域分割を行うときには，「画素値が類似している」かどうかを判断しなければならない，すなわち，領域内にある画素が持つ値の均一性をどの程度にするかを定義しなければならない．このために，均一性の評価方法と均一かどうか判断するための閾値を設定することになる (この閾値を本論文では分割度と呼ぶことにする)．ある画像に対して領域分割を行うとき，分割度を大きく設定すると，領域の均一性が高い，つまり，画素値が非常に近いものから領域が成り立つことになり，小さな領域が生成されることになる．反対に分割度を小さく設定すると，領域内の画素の均一性が低い，つまり，画素値がある程度離れたものまで一つの領域に含まれることになり，面積の大きな領域が生成されることになる．このように，領域分割の結果は分割度の影響を受けてしまうため，対象物モデルを作成するのに適した分割度をどのように設定すれば良いかという難しい問題が生じ，一般的な解決法は分かっていない．

そこで，ある一定の分割度による領域分割結果を利用するのではなく，分割度による領域分割結果の変化を利用する手法を新たに提案する．分割度を変化させたときの図 2.1 (a) の領域分割結果は図 2.1 (b) の右のようになる．分割度を徐々に小さくすることにより，それまで分かれていた領域が一つの領域に結合されていく (例えば領域 2 は領域 4，5，6，7

が結合したものである). これは，領域が複数のより小さな領域から成り立っていることを示す．つまり，領域にはこのような包含関係による階層構造があることがわかる．

本研究では，この領域の階層構造に，領域分割手法の特性を反映した対象物の本質的な情報が含まれていると考え，領域をノード，包含関係をリンクとする木 (図 2.1 (b) の左の木) によって，領域の階層構造を表現する．この木を分割木と呼び，これを利用して対象物モデルを構成する．

2.3.2 領域特徴の表現

分割木の各ノードには，そのノードに対応する領域の 2 次元的な特徴 (色，大きさ，位置などに関する特徴，領域特徴と呼ぶ) についての情報を保持する特徴テーブルを持たせる．画像中の対象物の大きさはあらかじめ分かっている訳ではないので，拡大縮小の影響を受けない特徴である必要がある．

データモデルは一つの例題画像から作成されたものであり，その特徴テーブルに保持される領域情報は分割木の各ノードに対応するユニークな領域の特徴を表したものである．一方，知識モデルは複数の例題画像から得られたデータモデルを統合したものである．したがって，その特徴テーブルに保持されている領域情報は例題画像集合中で対応づけられた領域の持つ特徴をまとめたものであり，例題画像が与えられる度に更新される．また，対象物モデルの更新時，および，対象物の探索時には，二つの分割木のノードの領域情報どうしを比較し，ノードの特徴の類似性を評価する．このため，領域情報はデータを更新しやすく，また，類似性を評価しやすい形式である必要がある．

特徴テーブルに保持される領域情報は以下の特徴である．

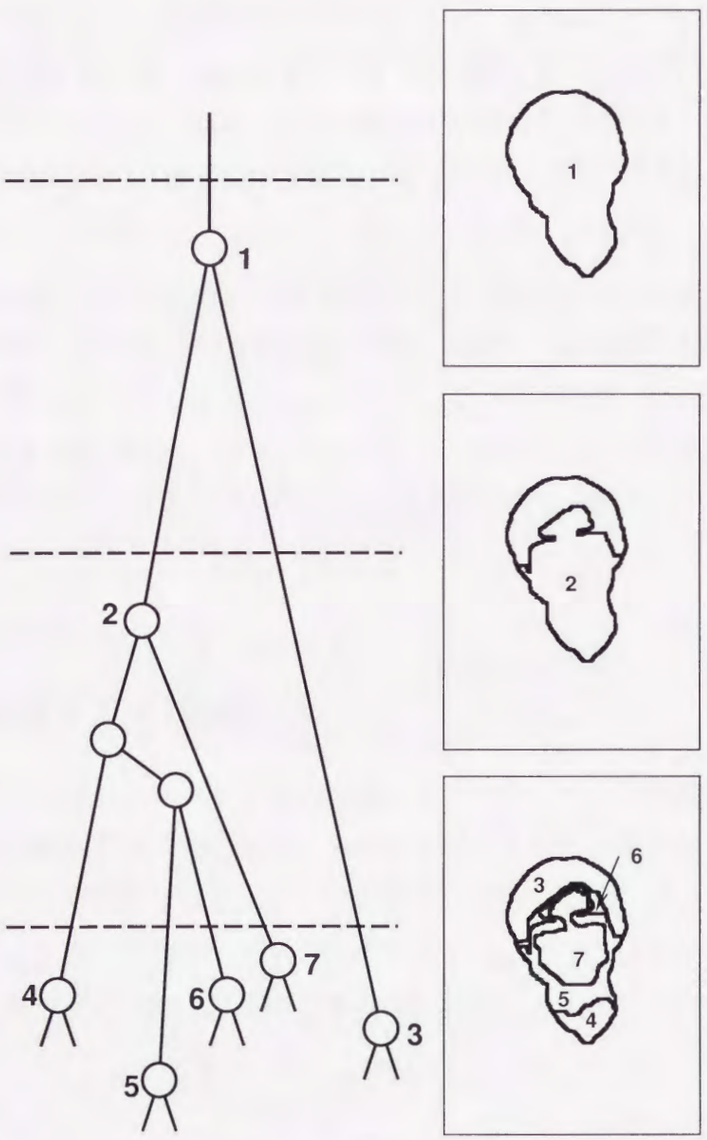
1. 色に関する特徴

   色に関する特徴として，データモデルでは，領域内の画素についての RGB それぞれの平均値と分散値を用いる．一方，知識モデルでは，例題画像集合中で対応づけられた領域のもつ上記の特徴量の頻度分布 (ヒストグラム) を特徴として用いる．ヒストグラムの階級数は，ここでは 15 にしている．

2. 位置・大きさに関する特徴

   位置と大きさに関する特徴は，拡大縮小の影響を受けないようにするため，データモデルでは，対象物全体の領域の外接矩形に対する，その領域の外接矩形の相対的位置

(a) 原画像



13

(b) 分割木

図 2.1: 分割木

(上辺，下辺，左辺，右辺) と大きさ (幅，高さ) を用いる．一方，知識モデルでは，例題画像集合中で対応づけられた領域の位置・大きさの特徴データを，すべてリストにして保持している．色の場合と異なり，ヒストグラムを作成しないのは，対象物探索処理における対象物全体の領域の位置の推定のためである．これについては 2.6 章で説明する．また，対象物モデル獲得時には，これらの特徴はヒストグラム形式に変換され，色情報と同様に処理される．この変換は，対応づけられた領域の位置・大きさの情報がすべてリストに保持されているので，簡単に頻度分布が計算できる．

3. 形に関する特徴

   形に関する特徴 (形状特徴) は，データモデルでは，ノードに対応する領域内の画素を 1，外の画素を 0 で示す 2 次元のビットパターン (領域ビットマップと呼ぶ) で表現している．ただし，画像全体の画素について領域内/外を示すわけではなく，領域の外接矩形中の画素についてのみ 1/0 のパターンで表現する．また，形状のみに注目するために，領域ビットマップの外接矩形の長辺が一定の長さになるように拡大縮小することによって，領域ビットマップの大きさを正規化している．

   知識モデルにおける形状特徴は，図 2.2に示すように，例題画像集合中で対応付けられた領域の領域ビットマップを重ねあわせた 2 次元の頻度分布として表す．これを領域ヒストグラムと呼ぶ．領域ヒストグラムは正方形であり，辺の長さは領域ビットマップの長辺と同じである．領域ビットマップと領域ヒストグラムの重ね合わせの位置は，最もよく重なり合うと考えられる位置 (2.5.3 節の **Match** の値が得られるときの位置) とする．

2.4 対象物モデルの獲得

対象物モデル獲得処理は，ユーザがある対象物クラスについての例題画像を複数与えることにより，その対象物クラスの対象物モデルを獲得する処理である．このとき，複数の例題画像を同時に与える必要はなく，ユーザが例題画像を与えるたびに，対象物モデルは，対象物クラスのより一般的なものに更新されていく．

対象物モデル獲得処理の手順は以下のとおりである．

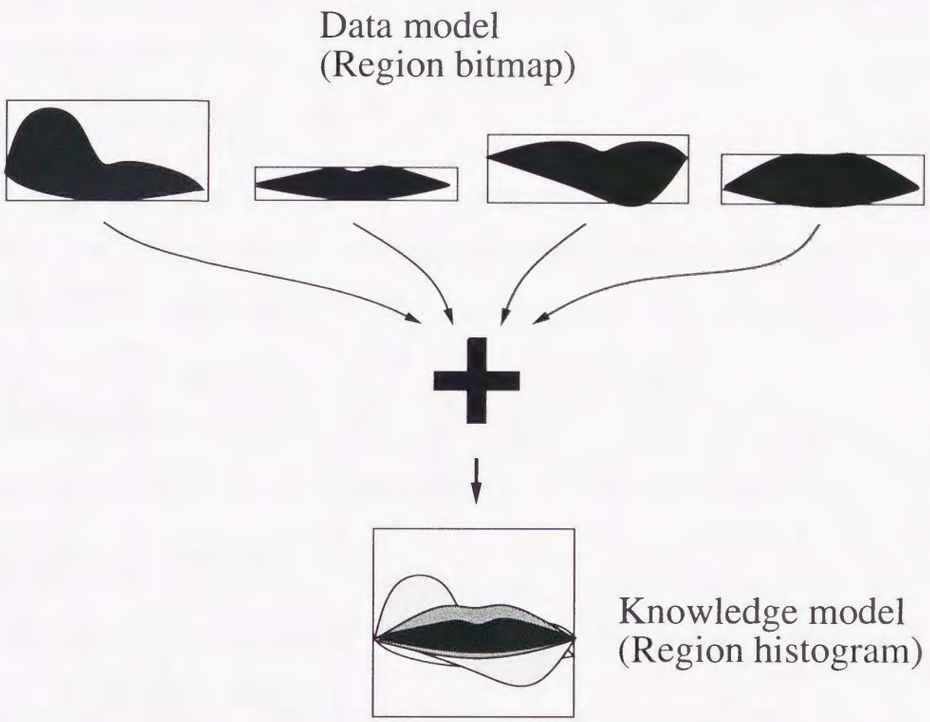


図 2.2: 領域ヒストグラム

1. システムはユーザから与えられた画像全体を領域分割し，画像全体についての分割木を作成する．
2. 1. で得られた分割木からユーザが対象物内部の領域を必要なだけ選択することで，対象物全体の領域を指定する．
3. 対象物内部の各領域に対応するノードの特徴テーブルを作成し，それ以外のノードを削除することで，対象物領域についてのデータモデルを作成する．
4. データモデルと知識モデルの間でノードマッチングを行い，二つの分割木のノード間での対応をとる．
5. 知識モデルを更新する．

この手順を複数の例示画像に対して行い，知識モデルを一般化する．ただし，最初の例題画像を与えたときには，知識モデルが存在しないので，手順 3で作成したデータモデルがそのまま知識モデルとなる．以下，手順 1，4，5，について詳しく説明する．

2.4.1 領域分割

画像全体を画素結合法 [13] により領域分割を行う．画素結合法は，

1. 各画素を一つの領域と考え
2. 隣接する領域のうち最も類似度の大きいものから順に結合し
3. ある閾値よりも大きな類似度の隣接領域がなくなると処理を終了する

という領域分割手法である．このとき，類似度の閾値ををを非常に小さく (実際は 0 に) 設定すると，処理終了時には領域が一つになる．この領域の結合の過程を木で表すことにより，分割木が得られる．また，結合した二つの領域の類似度が，生成された領域の分割度になる．

領域分割における領域間の類似度としては，以下の尺度 **Seg_level** を用いた．

$$\mathbf{Seg\_level} = \frac{1}{\log_{10}(\Delta S_{pq}+1)+1} \tag{2.1}$$

$$\Delta S_{pq} = \frac{n_p n_q}{n_p+n_q}\sum_{j=1}^{m}(\bar{x}_j^p - \bar{x}_j^q)^2 \tag{2.2}$$

ここで，n_p，n_q は領域 p，q の画素数，m は画像のバンド数（カラーの場合は 3），j は各バンドの画像に対応している．ΔS_{pq} は領域 p，q 間の分散であり，ΔS_{pq} が小さい，つまり **Seg_level** が大きいものほど，領域間の類似性が高いとした．

2.4.2 ノードマッチング

ノードマッチングとは，知識モデルの分割木の各ノードに対して，データモデルの分割木の中から対応するノードを探索することである．このとき，問題になるのは,「ノードの画像特徴の整合性」と「分割木全体の整合性」を考慮して，どのように最適解を探すかということである．

- ノードの特徴の対応
  知識モデルの各ノードについて，特徴が最も類似しているノードをデータモデル中から選び，それらを対応させる．特徴が類似しているかどうかを判断するために，照合度を利用する (2.5 節参照)．

- 分割木全体の整合性
  ノードの対応が分割木における親子関係と整合しているかをチェックする．実際には，ノードの特徴の類似により対応したノード対について，分割木の構造と矛盾が生じている場合は，照合度のより大きいノード対を優先させ，そうでないノード対を破棄することで解決する．対応が破棄されたノードは次に照合度の大きいノードとの対応を試みる．

知識モデルの各ノードが「データモデルのノードと矛盾なく対応している」か「矛盾なく対応するデータモデルのノードがない」となったとき，ノードマッチングを終了する．

ノードマッチングを画像認識における処理のレベルに当てはめると，画像特徴 (特徴テーブルに保持されている領域情報) を利用してシンボル (知識モデルのノード) とトークン (データモデルのノード) の対応をとるという中間レベル画像処理と，トークンの構造 (データモデルの分割木の構造) とシンボルの構造 (知識モデルの分割木の構造) に矛盾のない対応をとるという高レベル画像処理とがインタラクションを行いながら同時に実行される処理であると言える．

2.4.3 対象物モデルの更新

知識モデルの対応がとれたノードについては特徴テーブルに保持されている領域特徴を更新し，対応がとれなかったノードについてはそのノードを削除し，上位と下位のノードを接続する．

このように，知識モデルの更新では，対応のとれたノードのみを残しているので，知識モデルは，対象物に不可欠で，しかも，利用している領域分割手法によって必ず抽出される領域[1] から構成されることになる．また，特徴テーブルに保持されている領域特徴は，より多くの例題から作成されることになるので，そのノードのより一般的な特徴を表すことができるようになる．

2.5 照合度

照合度とは，知識モデルとデータモデルの対応するノードの特徴がどの程度類似しているかを表す度合である．照合度は，対応ノードの領域特徴どうしを比較することによって求める．この手順は以下のとおりである．

1. 各特徴から二つのノードが対応するかどうかを表す基本確率を計算する．
2. すべての特徴の基本確率から結合確率を求め，これからノードどうしの照合度を計算する．

以下，本研究で用いる確率則について説明し，具体的な基本確率の計算法とそれに基づく照合度の計算法について述べる．

2.5.1 Dempster & Shafer の確率則

上述した基本確率は Dempster & Shafer の確率則 [14] における確率であり，これは以下の三つの確率で表される．

- 肯定確率 $m(A)$
- 否定確率 $m(\bar{A})$

[1] これらの領域は，ここで定めた尺度に基づいて選択されたものであり，人間の直観に合わないように見えることもある

- 無知確率 $m(A \cup \bar{A})$.
- $m(A) + m(\bar{A}) + m(A \cup \bar{A}) = 1$

この基本確率は，知識モデルのノードとデータモデルのノードがどの程度類似しているかをある 1 種類の領域特徴だけで評価したものと考えることができる．

Dempster & Shafer の確率則における無知確率とは「その証拠からは A とも $\bar{A}$ ともいえない」確率を表す．したがって，本システムにおける無知確率は「その領域特徴からはノード対が対応するかどうか判断できない」確率を表すことになる．つまり，ある領域特徴がそのノードを特徴付けるものである場合，言い替えるとそのノードであるかどうかを判断するために有効な領域特徴である場合に，無知確率は小さくなり，肯定か否定かをはっきりと判断することになる．この無知確率の存在が Bayes の確率則との違いであり，あいまいさを含む証拠を扱う場合には，Dempster & Shafer の確率則を用いる方が自然に確率を表現できる．

また，基本確率 m_1 と m_2 の結合確率 $m_{1,2}$ を求める Dempster の結合則は (2.3) 式で与えられる．

$$m_{1,2}(z) = \frac{\sum_{x \cap y = z} m_1(x) m_2(y)}{1 - \sum_{x \cap y = \emptyset} m_1(x) m_2(y)} \tag{2.3}$$

ここで，$x, y, z = \{A, \bar{A}, A \cup \bar{A}\}$ ，$m_{1,2}(\emptyset) = 0$ である．

また，三つ以上基本確率の結合確率は，(2.4) 式を帰納的に適用すれば求めることができる．

$$m_{1,\cdots,i}(x) = m_{(1,\cdots,i-1),i}(x) \tag{2.4}$$

ここで，$2 \leq i \leq n$ ，n は基本確率数である．

2.5.2 ヒストグラムからの基本確率の計算

知識モデルの領域特徴がヒストグラムで表現されている場合，データモデルの領域特徴 (ここではその値を x で表す) が与えられたときの基本確率を以下のように定める．まず，基本確率は以下の条件を満たすものとする．

- 初期状態 (ヒストグラムがすべて 0) では $m(A) = m(\bar{A}) = 0$.
- 与えた例題画像数が多いほど，$m(A \cup \bar{A})$ が小さくなる．

これらの条件を満たす関数として，以下を定義する．

$$m(A:x) = \min(\frac{c}{Ms}h(x), 1) \tag{2.5}$$

$$m(\bar{A}:x) = \begin{cases} \min(\frac{cN}{Ms}, 1) & (h(x)=0 \text{ のとき}) \\ 0 & (h(x)\neq 0 \text{ のとき}) \end{cases} \tag{2.6}$$

$$m(A\cup\bar{A}) = 1 - m(A) - m(\bar{A}) \tag{2.7}$$

ここで，$h(x)$ は階級 x でのヒストグラムの度数，N は与えた例題画像数，M はヒストグラムの階級数 (後述の実験では $M=15$)，c は $h(x)=0$ である階級の数，s はスケーリングのための定数 (後述の実験では $s=20$) である．

c/M が大きくなると，$m(A)$ と $m(\bar{A})$ の値が大きくなり，$m(A\cup\bar{A})$ の値は小さくなる．これは分布に偏りがあるときには，その特徴がノード対応判定に有効であると考えられるからである．また，それまでその領域特徴が値 x をとったことがない場合に，$m(\bar{A})$ は正の値をとる．これは N の値が大きくなるほど意味が重くなるので，$m(\bar{A})$ を N に比例させている．

2.5.3 形状特徴からの基本確率の計算

形状特徴として保持されている領域ヒストグラムは，領域ビットマップを加算したものである．形状が不変なノードでは加算される部分と，加算されない部分に分かれることになり，領域ヒストグラムの値は 0 付近と例題数付近の両極端に分かれることになる．したがって，領域ヒストグラムを画像とみなしたときの濃度ヒストグラムを作成し，その分散を基に形状特徴の重要性を表す状態指数 (**Cond**) を (2.8) 式で求める．

$$\mathbf{Cond} = \frac{Var}{Mean(N-Mean)} \tag{2.8}$$

ただし，それぞれ領域ヒストグラムの濃度分布の Var : 分散値と $Mean$: 平均値であり，N は与えた例題画像数である．(2.8) 式の分母は，平均値が $Mean$ のときの Var の最大値である．このため，**Cond** の値は，領域ヒストグラムの濃度分布が平均値付近に集中しているときには 0，逆に 0 付近と N 付近の両端に偏っているときには 1 に近くなる．

これから，無知確率 $m(A\cup\bar{A})$ を (2.9) 式で求める．

$$m(A\cup\bar{A}) = 1 - \mathbf{Cond}\times(1-\frac{1}{N^2+1}) \tag{2.9}$$

ここで，例題画像数が少ないと，知識モデルの情報も信用性が低いので，無知確率を上げるために，重要性を表す **Cond** を例題画像数によって調整している．

次に，$m(\bar{A})$ と $m(A)$ を求めるために，領域ヒストグラムと領域ビットマップの一致度を表す値 (**Match**) を (2.10) 式によって計算する．一般に，領域ビットマップは領域ヒストグラムよりも小さいので，領域ビットマップを短辺方向にずらしながら **Match** を求め，最大の値を採用する．

$$\mathbf{Match} = \max_{i} \sum_{x} \sum_{y} W(x, y, i) \tag{2.10}$$

ただし，x 方向にずらすときは (y 方向にずらす場合も同様)，

$$W(x,y,i) = \begin{cases} 2 \times RHist(i+x, y) - N \\ \qquad (\ RMap(x,y) = 1 \text{ のとき}) \\ -(2 \times RHist(i+x, y) - N) \\ \qquad (\ RMap(x,y) = 0 \text{ のとき}) \end{cases}$$

ここで，i : ずれの大きさ，$RMap$: データモデルの領域ビットマップ，$RHist$: 知識モデルの領域ヒストグラムである．領域ビットマップ内のすべての画素について $W(x,y,i)$ を計算し，その合計を求める．その上で，i を変化させたときの合計の最大値が **Match** となる．この式において，領域ヒストグラムの度数の高い部分と領域ビットマップの 1 の画素が重なるところ，および，度数の低い部分と 0 の画素が重なるところでは，$W(x,y,i)$ は正になり，それ以外では負になる．したがって，領域ヒストグラムの度数の高い部分の形状と領域ビットマップの画素値が 1 の部分の形状が正確に重なるほど **Match** の値は大きくなる．

そして，(2.11)，(2.12) 式によって，$m(\bar{A})$ と $m(A)$ を求める．

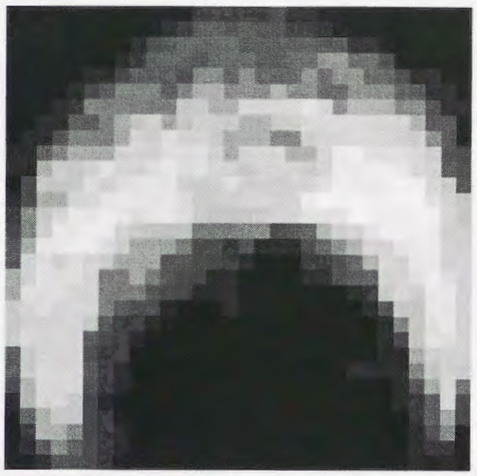
$$m(\bar{A}) = \frac{N \times RHistSize^2 - \mathbf{Match}}{2 \times N \times RHistSize^2} \times (1 - m(A \cup \bar{A})) \tag{2.11}$$

$$m(A) = 1 - m(\bar{A}) - m(A \cup \bar{A}) \tag{2.12}$$

(2.11) 式の分母は分子のとり得る最大値であり，その分子の第 1 項は **Match** のとり得る最大値である．したがって，領域ヒストグラムの **Cond** の値が大きく，**Match** の値も大きく，例題画像数が多い場合に，$m(\bar{A})$ は 0 に近づくことになる．

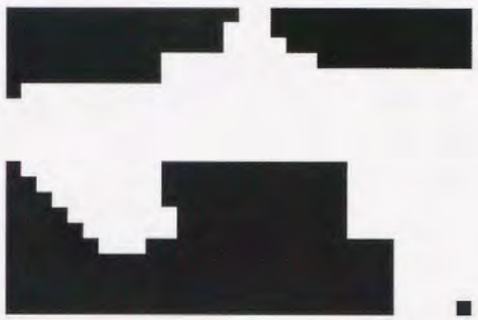
図 2.3に示す形状特徴 (黒の部分が 0 を示す) に対して，基本確率を計算したものを表 2.1 に示す．領域ヒストグラム H1，H2 はどちらも 15 枚の例題画像から作成したものであり，

領域ヒストグラム H1



領域ヒストグラム H2

領域ビットマップ B1



領域ビットマップ B2

図 2.3: 形状特徴の例

表 2.1: 形状特徴からの基本確率の例

	H1–B1	H1–B2	H2–B1	H2–B2
Var	38.3	38.3	19.1	19.1
Cond	0.686	0.686	0.397	0.397
Match	9646	1482	1604	6936
$m(A)$	0.584	0.378	0.221	0.299
$m(\bar{A})$	0.097	0.303	0.174	0.096
$m(A \cup \bar{A})$	0.319	0.319	0.605	0.605

これらと，ある画像のデータモデルの領域ビットマップ B1，B2 が対応づけられる基本確率を求めたものが表 2.1である．H1 との組合せについては，無知確率が小さくなっており，H2 との組合せは，無知確率が大きくなっている．これは，H1 では，領域の形が一定であるのに対し，H2 では領域の形が一定ではないことによる．また，対応づけられるべき組合せである H1–B1 の肯定確率は大きくなっており，その他の組合せについては肯定確率が小さくなっていることもわかる．

2.5.4 基本確率から照合度の計算

各特徴から確率を求めた後，(2.3)，(2.4) 式の Dempster の結合則によって結合確率を求め，その値から照合度 **Sim** を (2.13) 式を用いて計算する．

$$\mathbf{Sim} \quad = \quad m_{1,\cdots,n}(A) + m_{1,\cdots,n}(A \cup \bar{A}) \tag{2.13}$$

2.6 対象物の探索

対象物探索処理は，ユーザが与えた画像からユーザが指示した対象物の領域を抽出する処理である．対象物の領域は画像全体の部分領域であるので，対象物を表す分割木は画像全体についての分割木の部分木となる．したがって，画像全体のデータモデルを作成し，その分割木の中から対象物の知識モデルの分割木と対応する部分木を探せばよい．具体的には，以下の手順で対象物の探索を行う．

1. ユーザはシステムに対し探索の対象とする画像と対象物名を入力する.

2. システムは画像全体を領域分割し，画像全体についての分割木，および，それぞれのノードの特徴テーブルを作成する．これをデータモデルとする.

3. データモデルと知識モデルの間でノードマッチングを行い，二つの分割木のノード間での対応をとる.

4. 探索された対象物の領域を出力する.

以下，手順 (3)，(4) について説明する.

2.6.1 ノードマッチング

対象物モデル獲得処理と同様に，データモデルと知識モデルをノード間の照合度と分割木の整合性を基に対応させる．異なっているのは，対象物全体の領域が指定されていないために，照合度計算における位置と大きさに関する特徴が利用できないことであり，以下の手順によって照合度を求める.

1. 知識モデルとデータモデル間のノード対のすべての組合せについて，以下の処理を行う.

   (a) 位置・大きさ特徴を使わずに照合度を計算する.

   (b) 知識モデルのノードの持っている対象物全体の領域との相対位置特徴 (2.3.2 節 (2) 参照) と，データモデルのノードに対応する領域の位置特徴から，探索画像中の対象物全体の領域の位置を推定する．この位置を推定全体領域位置と呼び，四つのパラメータ (対象物全体の領域の外接矩形の上辺，下辺，左辺，右辺の位置) によって表す[2].

   (c) 推定全体領域位置を表す四つのパラメータによってはられる 4 次元空間に，推定全体領域位置を投票する[2]．このとき，(1a) で求めたノード対の照合度を投票の重みとする.

[2]知識モデルのノードの持つ相対位置情報は，知識獲得時に与えられた例題の数だけ存在している．したがって，(1b) と (1c) における推定全体領域位置はその数だけ存在することになる.

この結果，全体領域である可能性の高い位置ほど多くの票を得ることになり，この得票数が，ある位置が領域全体の位置である可能性がどの程度あるかを示すことになる．

2. 各ノード対の照合度を以下のように計算する．

   (a) そのノード対による推定全体領域位置での得票数[3]を調べる．これは，そのノード対が対応したと考えたときに推定される対象物全体の領域の位置・大きさの妥当性を示すものであり，したがって，そのノード対の位置・大きさ特徴の合致の程度を示すことになる．

   (b) (1a) で求めた照合度と，上で求めた得票数を積算することにより，最終的な照合度を求める．

2.6.2 探索された対象物領域の出力

画像中に対象物が存在している場合には，対象物を表す分割木が，データモデルの分割木の部分木となっている．したがって，知識モデルに含まれているノードは必須のものだけであるから，それらのすべてがデータモデルの分割木の部分木のノードと対応がとれたときに，指定された対象物が探索されたことになる．このとき，この部分木のルートノードの表す領域が，対象物全体の領域として出力される．

2.7 実験と考察

2.7.1 対象物モデル獲得実験

提案した方法による対象物モデル獲得実験を行った．実験では，例題画像として 15 枚の人間の頭部の写った実画像を用いた (図 2.4，カラー，120 × 160 画素)．対象物モデルの分割木のノード数は，データモデルでは 60 から 80 個程度であり，知識モデルでは例題を与えるにつれて図 2.5のように減少し，11 枚の例題を与えた時点で 6 個になった．これは，この 6 個のノードがすべてのデータモデルに存在したことを意味し，これらに対応する領域が対象物に必須の領域であるということを表している．

[3]実際には複数の推定全体領域位置があるので，それぞれの得票数の平均をとる．

図 2.4: 例題画像

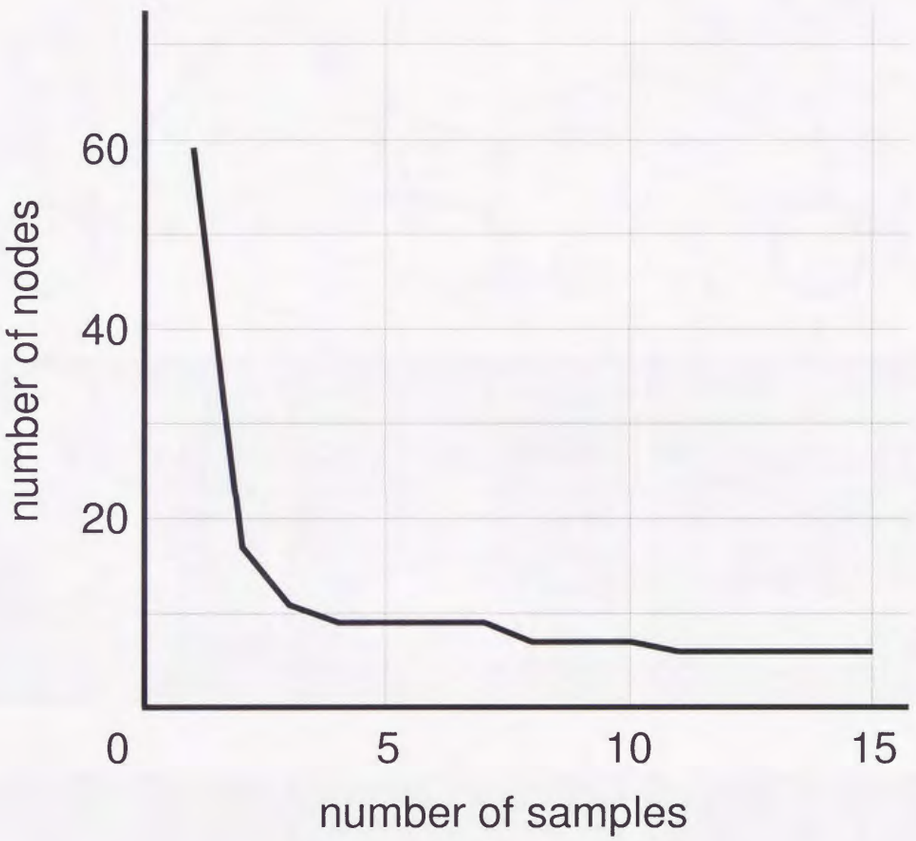


図 2.5: 分割木のノード数

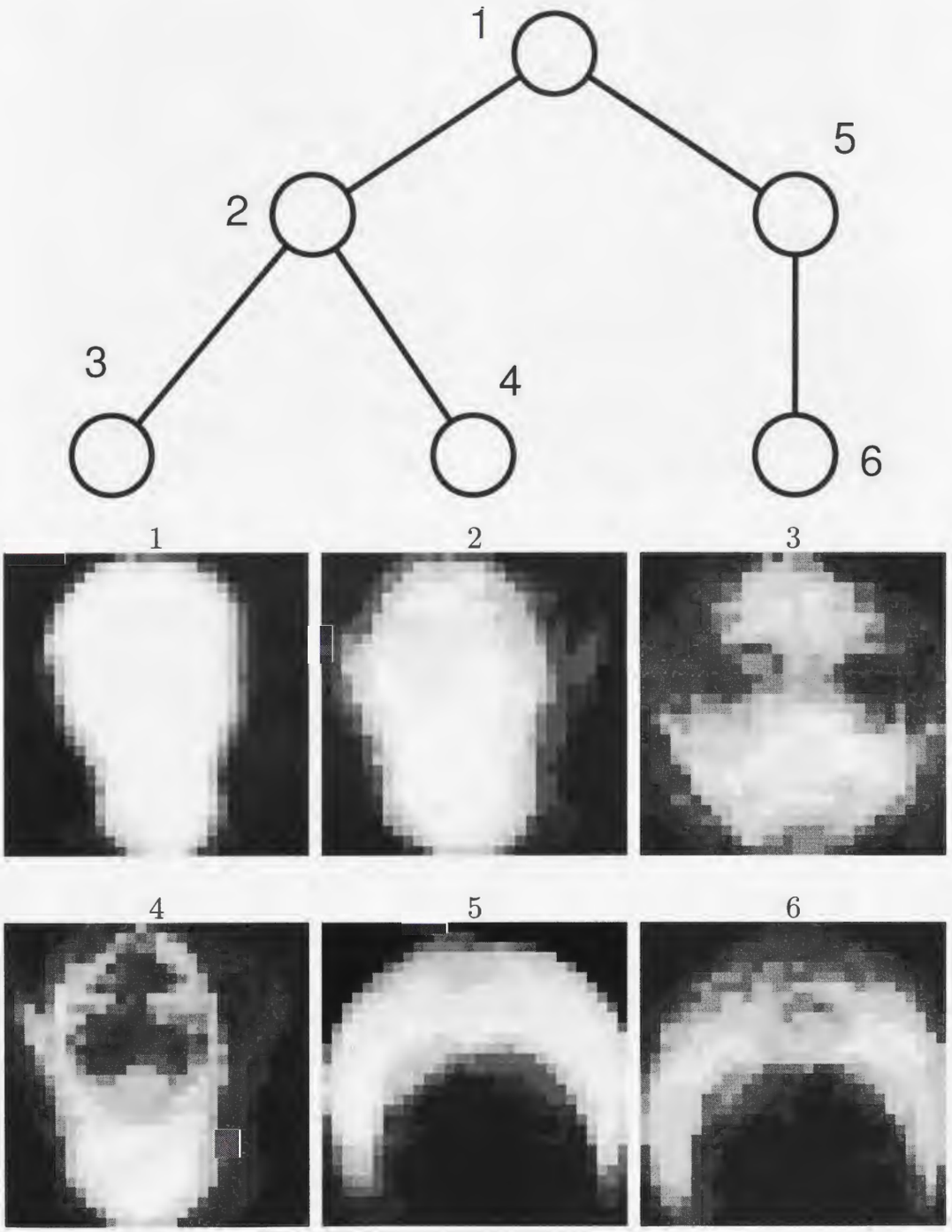


図 2.6: 獲得された分割木と領域ヒストグラム

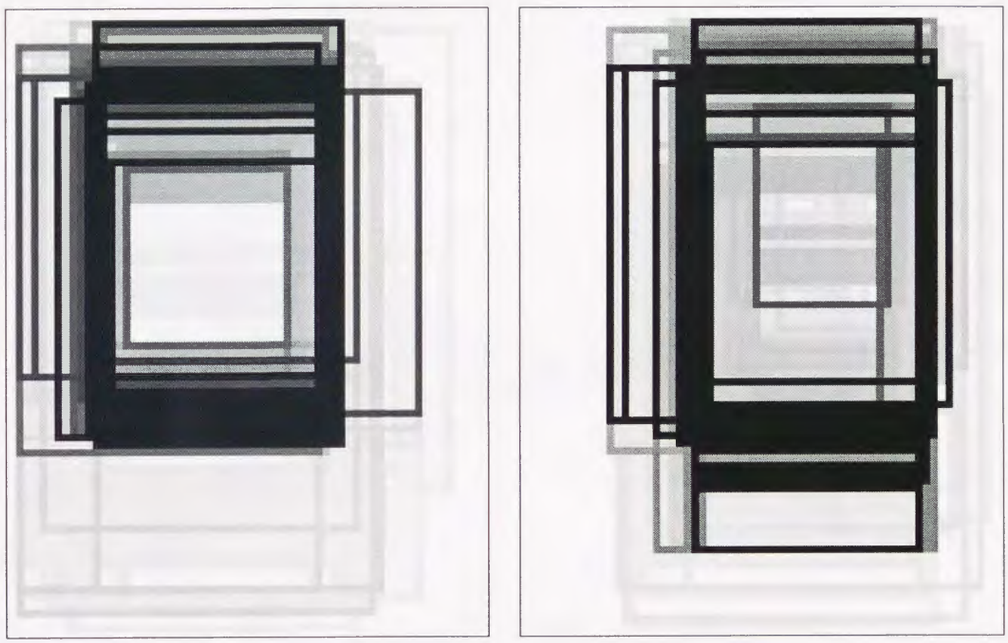
(a) (b)

図 2.7: 探索画像

このときの対象物モデルの分割木とその各ノードの領域ヒストグラムを図 2.6に示す．そのノードの番号は領域ヒストグラムの番号と対応している．ノード 1 は頭部全体，ノード 2 は顔全体，ノード 3 は顔の中心部，ノード 4 は顔の周辺部，ノード 5 は髪，ノード 6 は髪の一部 (照明によって光っていない部分) を表している．

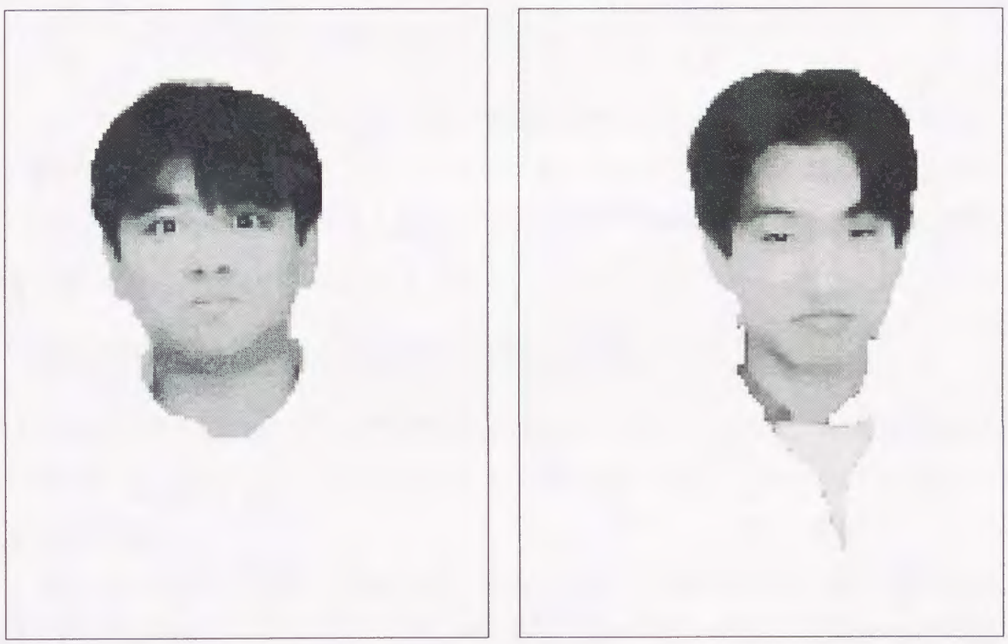
2.7.2 対象物探索実験

前節で獲得した対象物モデルを利用し，探索画像から人間の頭部の領域を抜き出す実験を行った．実験では，図 2.7を探索画像として与えた．これらの画像は対象物モデル獲得の例題としては与えていない．照合度計算のための対象物全体の領域の位置についての投票結果が図 2.8である．濃い部分ほど多くの票を集めたことを示す．これを利用して照合度を求めノードマッチングを行った結果，抜き出された領域が図 2.9である．



(a) (b)

図 2.8: 対象物全体の領域の位置



(a) (b)

図 2.9: 探索結果

2.7.3 考察

獲得した対象物モデルのノード数は，例題画像を与えるごとに減少していったが，11 個目の例題画像で 6 個になってからは減少しなくなった．これより，人間の頭部に必須である 6 個の領域から成る対象物モデルが獲得されたと考えられる．また，対象物探索の結果は良好であった．しかし，対象物の背景による領域分割誤りを考慮していないために，(b) の結果のように背景の一部を含んだ領域が出力されたり，探索がうまくいかない場合があった．

獲得した対象物モデル中に，目や口の領域がないのは，ある程度よりも小さな領域は，データモデルを作成する段階から削除しているからである．これは，小さい領域を扱うために生じる以下のような問題を避けるための処置である．

- ディジタル画像において，小さすぎる領域は形状特徴が不正確になる．また，画像を領域分割し分割木を作成するとき，小さい領域ほど多く生成される．これにより，小さくて似たような (領域特徴が似ている) 領域が多数存在することによる誤対応の問題が生じる．

- 領域数の増大による処理量の増加の問題が生じる．

目や口の領域を含む対象物モデルを獲得できるようにするためには，画像の解像度を上げ，それらの領域の面積を大きくするか，より小さな領域についても精度の高い特徴を利用することが必要である．

次に，本手法の有効性について考える．本手法では，対象物の 2 次元的な「見え方」を基に対象物モデルを作成している．ここでいう見え方とは，色による領域分割の結果得られる分割木の構造と各ノードの持つ特徴量ということであり，一つの対象物クラスではこれらに大きな変化がないということが，本手法の前提となっている．本章で示した実験では，簡単な RGB の強さの均一性を基にした領域分割の手法により分割木を作成したため，このままでは光源等の環境が大幅に変化したり，鏡面反射成分が強く現れている例題に対してはうまく動かない可能性がある．しかし，それらの問題に対応した手法 [15, 16, 17] を基に領域分割を行えば，基本的な枠組みはこのままで対応できるものと考えている．また，通常のカメラからの入力だけでなく，レンジセンサのからの入力を用いた結果についても同じようなことが可能であり，その場合は，対象物の表面形状の類似性に基づいて知識モデルが作成されることになる．この手法の一番の問題点は，3 次元的な情報を基にしていないので，視点が大幅に異なれば，同じような分割木が構成されず，知識モデルが生成でき

ないという点である．また，機能的には同じことを果たすが，見かけが大幅に異なるような抽象度の高い対象物クラス (例えば水道の蛇口) についてもユニークな知識モデルを構成することができない．これらの問題に対しては，一つの対象物クラスに対して複数の対象物モデルを保持するような仕掛けを構築することによって対処できると考えているが，その点は今後の課題と言えよう．

最後に処理時間について述べる．本手法の処理には，最大 5 時間 (知識モデルとデータモデルのノード数によって大きく変化する) とかなりの時間がかかっている．本手法では，データモデルを作成し，それと知識モデルの間でノードマッチングを行い，知識モデルを更新 (または対象物領域を出力) する，という処理の段階があるが，そのなかでノードマッチングが処理時間のほとんどを占めている．これは非常に多くのノードの対応をとる処理のため，組み合わせ的に計算量が増えているからである．この問題を解決するには，ノードの数を減らすか，ノードマッチングを並列に行う必要がある．まずノードの数について考える．今回提案した分割木の作成法は，領域統合の課程を忠実に構造化している．そのため，例えばある領域とそれに隣接する領域 (たとえそれが 1 画素からなる領域であっても) とが統合され領域が生成されると，その新しい領域に対応するノードが作られる．このため，ほぼ同じ領域特徴を持つ領域が多数生成されてしまい，ノード数の増加の原因になっている．これを改善して，領域統合によって生成された領域がもとになった領域と領域特徴が大きく異なる場合のみ，ノードを作成するようにすれば，ノード数を大きく減らすことができる．また，ノードマッチングの並列化については第 3 章で述べる．

第 3 章

高レベル画像処理における並列処理

3.1 目的

2.4.2 節で述べたように，ノードマッチングは高レベル画像処理と中間レベル画像処理を同時に実行する処理である．これは，シンボルの構造というトップダウン知識による「全体構造の整合」と，トークンとシンボルの対応というボトムアップ知識による「部分特徴の整合」という二つの制約を満たす解を求める処理である．

通常の木どうしのマッチングならば，ルートノードを出発点としてそこからリーフノードに向かって順番にノードの対応を取っていくことによって，二つの制約を満たすマッチングを実現できる．しかも，対応が取れたノードをルートノードとする部分木については，ほかの部分木と独立に処理を行うことが可能なので，並列処理を容易に導入することができる．

しかし，本研究におけるノードマッチングでは，

- ノードどうしが対応するかどうかは，他のノードの対応付けにも影響されるため，それらのノードだけでは判断できない．
- すべてのノードに対応するノードが存在するとは限らない．

という理由により，木の上位ノードから順番に対応をとる方法では正しく対応付けができない．

そこで本研究では，各ノードが並列に，分割木全体の整合性を保ちながら自分と対応するノードを探索する処理の実現を目指す．2.7.3 節で示したように，この処理には多くの計

算時間を必要とするため，マルチエージェントモデルによる並列処理を導入し，その高速化を図るが [18]，ここでは負荷分散が特に大きな問題になる．低レベル画像処理における並列化では，画像上のデータ参照の局所性が高く，画像を一様に処理するため処理量の偏りも少ないため，容易に負荷分散可能である．これに対し高レベル画像処理では，データ参照は大域的に行われ，処理量の偏りも大きいため，負荷分散が難しい．以下本章では，各エージェントの機能，エージェント全体でのふるまい，マルチエージェント型並列処理の実現方法，実験結果について述べる．

3.2 エージェントの機能

本研究では知識モデルノードエージェント (KNA)，データモデルノードエージェント (DNA)，知識モデルリンクエージェント (KLA)，マスターエージェント (MA) の 4 種類のエージェントを導入する．これらのエージェントは他のエージェントからのメッセージを受信することで動作し，その結果必要ならば他のエージェントにメッセージを送信する．このように各エージェントが自律的に動作し，メッセージのやりとりをすることにより，ノードマッチングを実現する．

3.2.1 知識モデルノードエージェント (KNA)

知識モデルの分割木の各ノードと 1 対 1 に対応し，そのノードと対応するデータモデルの分割木のノードを探索する．他のエージェントからのメッセージを受信することにより以下の動作を行う．

- 機能
  - 照合度一覧 (後述) の上位の DNA へのプロポーズ
  - KLA へ親子関係制約確認の依頼
  - DNA からのプロポーズを受理するかどうかの判断
- 保持するデータ
  - 特徴テーブル
  - 各 DNA との照合度一覧

- 自分に関係する KLA へのポインタ
- 自分に対応する DNA へのポインタ

- メッセージに対する動作

初期設定 すべての DNA との組合せについて照合度を計算し，照合度があるしきい値以上の DNA について，DNA とその照合度を一覧表に保持する．対象物モデル獲得処理においては，知識モデルとデータモデルの分割木のルートノードどうしが対応することが分かっているので，照合度をすぐに計算できる．対象物探索処理においては，知識モデルの分割木のルートノードがデータモデルの分割木のどのノードと対応するのかが最初は分からないので，2.6.1 節で説明した通り，2 段階の処理によって照合度を計算する．

対応 DNA 探索 一覧表の上位の DNA から順にプロポーズする．プロポーズが受理されると，KLA に親子関係のチェックを依頼する．一覧表のすべての DNA にプロポーズを拒否されたときは動作を停止する．

KLA からの対応関係の撤回 一覧表の次の DNA から順にプロポーズ処理をおこなう．

KLA からの対応関係の承認 動作を停止する．

DNA からのプロポーズ プロポーズを受理するかどうかを決定する．受理する場合は，KLA に親子関係のチェックを依頼し，確定したら DNA に受理のメッセージを送り，動作を停止する．

3.2.2 データモデルノードエージェント (DNA)

データモデルの分割木の各ノードと 1 対 1 に対応し，そのノードと対応する知識モデルの分割木のノードを探索する．他のエージェントからのメッセージを受信することにより以下の動作を行う．

- 機能
  - KNA からのプロポーズを受理するかどうかの判断
  - 照合度一覧の上位の KNA にプロポーズ
- 保持するデータ

- 特徴テーブル
- それまでプロポーズのあった KNA についての照合度一覧
- 自分の親子の DNA へのポインタ
- 自分に対応する KNA へのポインタ

- メッセージに対する動作
  データモデルの各ノードに対応する．KNA からのメッセージに対応して以下のように動作する．

  照合度計算の依頼 特徴テーブルを比較して照合度を計算し，結果を KNA に伝える．

  KNA からのプロポーズ 対応する KNA がない場合は，無条件に受理する．すでに，対応する KNA がある場合は，照合度を比較し，大きい方を選ぶ．新しい KNA が選ばれた場合は，これに受理のメッセージを送り，古い KNA にキャンセルのメッセージを送る．古い KNA が選ばれた場合は，新しい KNA に拒否のメッセージを送る．また，プロポーズのあった KNA について，照合度の一覧表を作成する．

  KNA からのキャンセル 一覧表の上位の KNA から順にプロポーズする．プロポーズが受理されるか，一覧表のすべての KNA に拒否されるかすると，動作を停止する．

3.2.3 知識モデルリンクエージェント (KLA)

知識モデルの分割木の各リンクと 1 対 1 に対応し，両端のノードの親子関係の整合性をチェックする．他のエージェントからのメッセージを受信することにより以下の動作を行う．

- 機能
  - マッチングにおけるノードの親子関係の整合性をとる
- 保持するデータ
  - 自分の両端の KNA へのポインタ

- メッセージに対する動作

  KNA からの親子関係のチェックの依頼に対して，自分の両端の KNA に対応する DNA について，KNA とおなじ親子関係 (先祖と子孫の関係) が成り立つかどうかを調べる．成り立つ場合は，承認のメッセージを依頼してきた KNA に送る．成り立たない場合は，照合度の小さい方の KNA の対応関係を不適切とし，この KNA に対応関係の撤回のメッセージを送る．

3.2.4 マスターエージェント (MA)

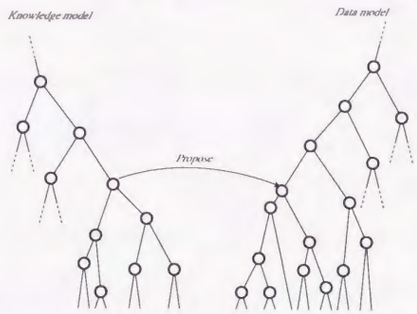
処理の開始と終了判定を行う特別なエージェントであり，このエージェントがメッセージを受信することはない．MA は他のエージェントの動作状態を監視しており，動作しているエージェントがなくなると，処理が終了したと判断する．MA の動作は以下の通りである．

- 処理開始時にすべての KNA に対し初期設定開始メッセージを送る．
- 初期設定処理が終了すると，次にすべての KNA に対し対応 DNA 探索開始メッセージを送る．
- 対応 DNA 探索処理が終了すると，ノードマッチングを終了させる．

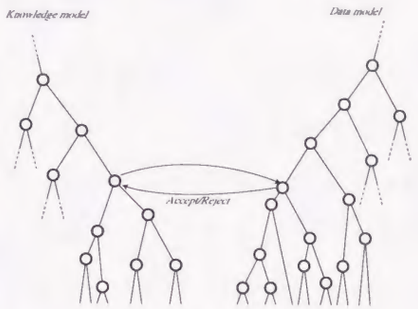
3.2.5 エージェント全体としてのふるまい

処理が始まると，各 KNA について対応する DNA が探索されていく (図 3.1参照)．このとき，より照合度が大きくなるように，各エージェントは動作する．したがって，照合度の大きなノード対から対応が確定していくことになる．さらに，このようなノード対を基に親子関係の整合性を利用してノードの対応がとれていく．最終的には以下の条件を満たすモデルマッチングが実現される．

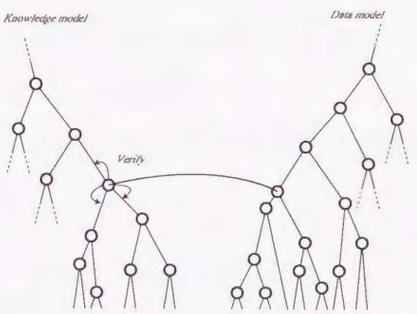
- 親子関係の整合性がとれている．
- 各 KNA,DNA について，これ以上照合度が大きくかつ対応ノードのない対応候補は存在しない．



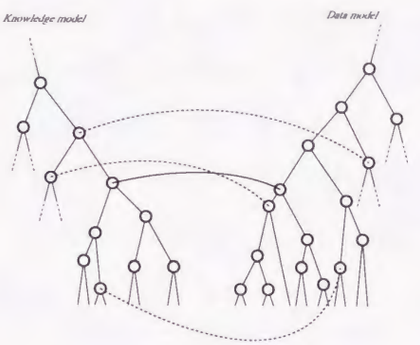
(a) 左が知識モデルの分割木の一部，右がデータモデルの分割木の一部とする．まず，KNA が DNA にプロポーズのメッセージを送る．



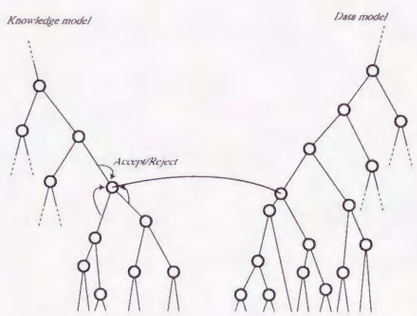
(b) DNA が KNA に受理か拒否のメッセージを送る．拒否の場合は (a) に戻る．



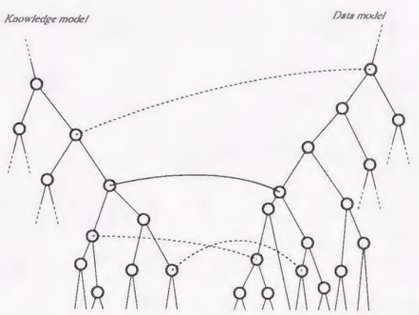
(c) プロポーズが受理された場合，KNA は周囲の KLA に親子関係のチェック依頼のメッセージを送る．



(d) このような対応関係であれば親子関係に矛盾がない．



(e) このような対応関係が一つでもあれば親子関係に矛盾があることになる．



(f) 矛盾がない場合は，KLA は KNA に承認のメッセージを送る．矛盾がある場合は，KLA は矛盾する対応関係のうち照合度が小さいものに対応関係撤回のメッセージを送る．

図 3.1: ノードマッチングの流れ

3.3 マルチエージェントモデル型並列処理の実現

マルチエージェントモデル型並列処理を実現する場合，最も素直な方法は，各エージェントを処理の実体 (プロセス) とし，それらをそれぞれ一つのプロセッサ上で実行することである．しかし，本システムでは対象物探索におけるデータモデルの分割木のノード数は画像の面積の約 2 倍であり，例えば 320 × 240 の画像の場合 $320 \times 240 \times 2 - 1 = 153599$ 個にもなる．DNA だけでもこれだけの数になってしまうため，すべてのエージェントが並列に動作するだけの十分な数のプロセッサを用意するのは難しい．同様に，十分な数のプロセスを起動・管理できる OS もない．

そこで本研究では，ワーカをプロセスとして起動し，各エージェントはワーカ上で実行されるようにする．ワーカの数を変更することで，必要なプロセッサの数と起動されるプロセスの数をユーザが調整できるようになる．すべてのエージェントはメッセージを受信することによって動作するので，すべてのメッセージを一括管理するメッセージキューを用意し，各ワーカが以下の処理を繰り返すことで，マルチエージェントモデル型並列処理を実現する．

1. ワーカはメッセージキューからメッセージを取り出す．
2. そのメッセージを受信するべきエージェントの動作を実行する．
3. 送信メッセージをメッセージキューに入れる．

このとき，共有メモリ型並列計算機と分散メモリ型並列計算機のどちらを利用するかにより，以下のような利点と欠点が生じる．

共有メモリ型並列計算機 すべてのワーカがすべてのエージェントの動作を実行することができるので，ワーカ間の動的負荷分散を効率的に実現できる．しかし，メッセージキューのデータの出し入れには排他制御が必要なため，そこがボトルネックになってしまう．

分散メモリ型並列計算機 通信のオーバヘッドを考えると，各エージェントの動作を実行するワーカを固定しなければならないため，負荷分散が難しい．しかし，このため一つのワーカに一つのメッセージキューを用意することができるため，メッセージキューがボトルネックになることがない．

本研究では，共有メモリ型並列計算機を利用する．これは，

- どのエージェントとどのエージェントがメッセージのやりとりをするか予測できない．
- どのエージェントの負荷が大きいか予測できない．

ことより，動的負荷分散を効率的に実行できることによる利点が大きいという点が第 1 の理由である．また，メッセージキューがボトルネックにならないように，つまり，排他制御のための待ち時間が少なくなるように以下の工夫を行うことで，上記の欠点を克服することができたことが第 2 の理由である．

- メッセージキューを複数用意する．
- メッセージキューへの書き込みとメッセージキューからの読み出しのそれぞれに，どのメッセージキューに対してメッセージの出し入れをするべきかを示す大域変数を用意する．
- ワーカがメッセージキューに対してメッセージの出し入れをするときは，大域変数の値によって出し入れするメッセージキューを決定し，そのあと大域変数の値を次のメッセージキューを示すように変更する．
- 大域変数の参照と変更については排他制御を行わない．このため大域変数の参照と変更のタイミングによっては，同じメッセージキューへに対するメッセージの出し入れが連続する場合がある．しかし，メッセージの出し入れを完全に振り分けることを目的としているわけではない (ある程度分散されれば十分である) ため，問題ない．
- メッセージキューに対するメッセージの出し入れについては，排他制御を行う．

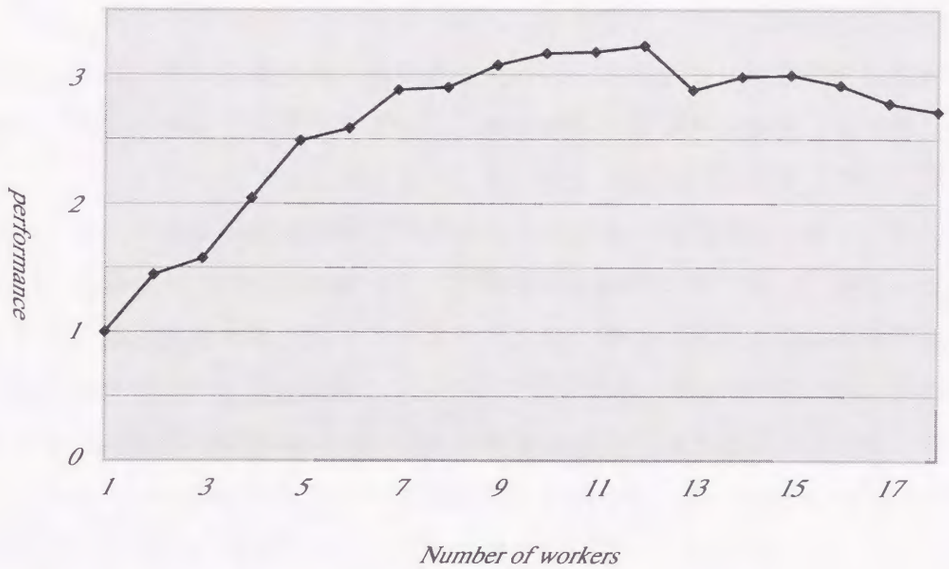


図 3.2: ワーカ数による処理速度の変化

3.4 実験と考察

本章で提案したノードマッチング手法を共有メモリ型並列計算機である SUN UltraEnterprise(CPU として UltraSparc を 24 個搭載) 上に実装し，対象物モデル獲得実験を行った．使用 CPU 数による処理速度の変化は図 3.2の通りである．ただし，CPU を 1 個使ったときの速度を 1 とした．この結果から，高レベル画像処理としてはまずまずの高速化が実現されているのが分かる．なお，CPU を 1 個使ったときの実際の処理時間は，対象物モデル獲得時は 25 秒程度から 2 秒程度に変化する (知識モデルのノード数の減少にしたがって処理時間も減少する)．速度が CPU 数に比例しないことについては以下の原因が考えられる．

メモリアクセスの競合 これは共有メモリ型並列計算機の欠点であり，キャッシュの大容量化や先読によってキャッシュヒット率を向上させる必要がある．

排他制御のオーバヘッド メッセージキューの構造を工夫することにより，排他制御による待ち時間は大幅に削減されたが，排他制御自体のオーバヘッドは減少していない．ワーカを利用したマルチエージェントモデルによる並列化を行うためには，このメッセージキューの排他制御は必須である．このため，対策としては排他制御自体を高速化する必要がある．

本章で提案したノードマッチング手法のようにメモリアクセスの局所性が低く，負荷の見積りも難しい高レベル画像処理を実現するためには共有メモリ型並列計算機が有効である．しかし，実験の考察でも述べたように，共有メモリ型並列計算機ではメモリアクセスや排他制御がボトルネックになってしまうため，さまざまな画像処理を同時に実行するような場合は，それぞれの画像処理の間でのメモリアクセスの局所性が高く，負荷の見積りも容易比較的容易なので，分散メモリ並列計算機が有効である．特に，多数のカメラを用いて得られる多視点画像情報の解析アルゴリズムや，多くの画像処理結果を統合する協調型アルゴリズムを実行させる大規模なシステムではその有効性が顕著であると考えられる．このような場合の並列処理方式については，第 4 章以降で取りあげる．また，低レベル画像処理から高レベル画像処理までを同時に実行する場合は，高レベル画像処理を実行するための共有メモリ型並列計算機が，各画像処理を実行するための分散メモリ型並列計算機の一部となっているような階層的な並列計算機が有効であると考えられる．

第 4 章

分散型並列計算機による実時間画像処理

4.1 目的

近年，広域監視 [19, 20]，自動講義撮影 [21, 22]，自動走行車 [23]，モーションキャプチャ [24, 25] などの動画像処理を実時間で行う必要があるアプリケーションの研究が盛んに行われるようになってきた．これは計算機の低価格化と能力の向上によるところが大きい．しかし，実時間画像処理は，静止画像処理よりもさらに計算量が多く時間的な制約も生じるため，研究が進むにつれてさらに計算能力向上の必要性が高まってきている．

また，これらのアプリケーションでは複数のカメラを利用することも多い．複数のカメラを利用する場合，物理的な制約および I/O 能力の限界により，1 台の計算機に接続できるカメラの台数が制限されてしまうという問題がある．

これらの問題を解決するために，計算機とカメラから成る観測ステーションをネットワークによって接続した分散型並列計算機を構築し，実世界の様々な状況を把握しようという研究が進められている [26, 27, 28, 29].

近年の計算機の処理能力の向上と価格の低下はめざましく，このような分散並列計算機を利用することにより，性能，コストの両面ですぐれたによるシステムを構築することができるようになってきており，分散型並列計算機のためのプログラミング環境が提案されている [30, 31]．このような環境を利用することにより，容易に分散並列計算機上でシステムを構築することができ，また，これらの環境を利用できるのであればどのような分散並列計算機上でも構築したシステムを動作させることができるという利点がある．

また，汎用の計算機を利用することにより，画像キャプチャボードやビデオグラフィッ

クスカードなどの様々な周辺機器を接続することができる．さらに，OS の選択肢も広いため，システム設計の自由度が高いという利点もある．さらに，カメラの台数の増減にも，計算機の台数を増減によって容易に対応できる．

分散型並列計算機には以上のような利点があるが，しかし，これまでの分散型並列計算機は，通常の並列計算機に比べて通信速度が遅く，大量のデータ通信を伴なう画像処理には向かなかった．しかし，近年の，ネットワーク技術の進歩により 1Gbps を超える高速ネットワークを低コストで利用できるようになってきた．これは非圧縮動画像のリアルタイム転送が可能な転送速度である．

このような背景のもと，本研究では，汎用計算機である PC を利用し，それらを高速ネットワークで接続した PC クラスタ上で実時間画像処理を行うことを目指す [32, 33, 34]．このためには，高速データ転送機構，同期機構，エラー処理機構が必要であり，特別なハードウェアを用いないことを前提としているので，これらの機能をソフトウェアによって実現する．

4.2 他研究との比較

本研究のように複数のカメラを複数の計算機に接続して動画像処理を行う研究としては以下のものが挙げられる．

- Kanade らの 3D Dome[35]
  これは 49 台の同期のとれたカラーカメラと 17 台の PC を利用し，動画像を撮影するものである．このシステムは複数視点の動画像を記録するためのものであり，オンラインで動画像処理を行うものではない．

- Davis らの Keck Laboratory[4]
  これは 64 台の同期のとれたカメラ (48 台の濃淡カメラと 16 台のカラーカメラ) と 17 台の PC を利用し，オンラインで動画像処理を行うものである．しかし，このシステムは高速ネットワークを利用しておらず，動画像の実時間転送はできない．

- 亀田らの研究 [36]
  これは 4 台のカラーカメラと 9 台のワークステーションを利用し，オンラインで動画像処理を行うものである．このシステムは高速ネットワークの一種である ATM を利

用しており，動画像を転送しながら処理を行っているが，処理遅れに対応していないので実時間性を保証できない．また，カメラ間の同期を行っていないため，撮影時刻のずれが生じ，アプリケーションによっては十分な精度が出ないことが考えられる．

これに対し，本研究のPCクラスタシステムは同期のとれたカラーカメラとPCを利用し，高速ネットワークによって動画像を転送しながら実時間並列動画像処理を行うことができる．さらに，処理遅れへの対処を実現することにより，実時間性を保証している．

4.3 PCクラスタシステムの概要

本研究で用いるPCクラスタシステムの概要について述べる．

4.3.1 システム構成

本研究で用いるPCクラスタは，14台のノードPCからなっている．各PCの主な各PCは2個のCPUを搭載しているAT互換機であり，OSにはLinuxを用いている．各PCは高速ネットワークMyrinetによってで相互に結合されている (図 4.1)．Myrinetは，アメリカのMyricom社が開発した，スイッチングハブを中心としたスター型の構成をとるギガビットLANの一種である．Myrinet上の通信にはPM通信ライブラリ [37] を利用している．また，6台のCCDカメラがビデオキャプチャボードによりノードPCに接続されている．

以下に具体的はハードウェア構成を以下に列挙する．

- CPU: Intel PentiumIII 450MHz × 2
- メインメモリ: 384MB SDRAM
- Myrinetインターフェイス: Myricom M2M-PCI32
- イーサネットインターフェイス: Intel EtherExpress Pro/100
- ビデオキャプチャボード: Imaging Technology IC-PCI
- CCDカメラ: Victor KY-F57
- 同期信号発生器： リーダー電子 410BB

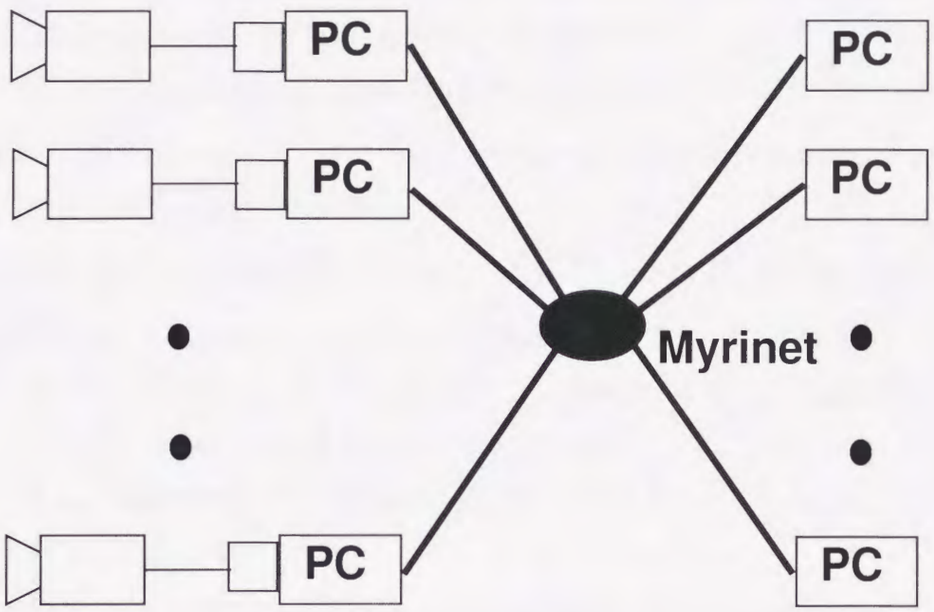


図 4.1: PC クラスタの構成

4.3.2 PC クラスタの利点と欠点

通常の並列計算機と比較したときの，分散型並列計算機，その中でも特に PC クラスタの利点と欠点について挙げると以下のようになる．

- PC クラスタの利点

  拡張性 ノード PC を増やすことで，容易にシステムを拡張できる．これにより，接続可能なカメラを増やすことも容易になる．

  コストパフォーマンス 近年の PC の高性能化および低価格化は著しい．

  柔軟性 汎用部品を利用しているので，最新の部品への更新が容易である．

  豊富な周辺機器 画像入力やグラフィクスに関係する周辺機器が豊富にある．

  OS の選択 Windows，UNIX からリアルタイム OS まで，さまざまな OS を選択できる．

- PC クラスタの欠点

ネットワーク速度 (特に共有メモリ型並列計算機と比べると) 計算機間のデータ転送能力が劣るので，細粒度並列処理や大規模なブロードキャストが必要な処理のようにデータ転送量が多い処理の実現は難しくなる．

稼働率 1 台の計算機よりも多くの部品から構成されることになるので，故障する可能性は高くなるおそれがある．

保守管理 複数の計算機を運用することになるので，その分保守管理の手間がかかる．

必要スペース．エネルギー 各 PC には直接画像処理には必要のない部品 (フロッピィディスクドライブ，CD-ROM ドライブ，音源カードなど) や PC 間で重複している部品 (ハードディスクドライブ，ビデオカードなど) が含まれていることが多く，装置の大きさや消費エネルギーの点で無駄が発生する．

4.4 分散型並列計算機における実時間画像処理の枠組み

実時間画像処理システムの性能を評価する場合，入力データが処理されてその結果が出力に出てくるまでの遅れ (レイテンシー) と単位時間あたりの処理量 (スループット．通常，1 秒間に何フレーム処理できるかを示す fps:frame per second で示すことが多い) の二つを考える必要がある．並列化の方式によって，どちらの性能が向上するか異なってくるので，並列化の容易性や用途に応じて，適切な並列化を行う必要がある．分散型並列計算機で実時間画像処理を行う場合は，第 1章で述べたように，以下の方式が考えられる．

パイプライン並列処理 図 4.2 (a) に示すように，処理の段階ごとに 1 台の計算機を割り当てる並列処理であり，実時間画像処理では，一つのフレームに対する処理を各計算機で順々に行うという方式をよく用いる．この場合，ある時刻では，各計算機は異なったフレームに対する処理を並列に行っていることになる．従って，計算機台数を増やすと，並列に処理するフレーム数が増えるため，スループットは向上するが，あるフレームが入力されて，その結果が出力されるまでに必要な時間は，計算機台数分のフレーム数だけ遅れることになりレイテンシーが増大する．各計算機の処理量が同じであると仮定すると，N 台の計算機によりスループットは N 倍になるものの，レイテンシーは N 倍になる．

データ並列処理 図 4.2 (b) に示すように，データを分割し複数の計算機で同時に処理する並列処理であり，実時間画像処理では，一つのフレームのデータを分割して各計算機で処理を行うという方式を用いることが多い．計算機の台数を増やすと，各計算機で処理すべきデータの量が減り，その結果，スループットは増大し，レイテンシーは減少する．各計算機の処理量が同じであると仮定すると，N 台の計算機によりスループットは N 倍になり，レイテンシーは $1/N$ になる．言い換えれば，レイテンシーを固定すると，計算機を増やすほど，それに比例して可能な計算量が増えることになる．ただし，データを相互の処理結果に非依存に分割しなければならないので，処理の内容によってはこの並列処理を実現できるとは限らないし，更に，データ分割と処理結果の統合のためのオーバーヘッドが発生する．また，データを分割するのではなく，複数のカメラによって同時に獲得されたデータに対して複数の計算機で同時に処理する方式もデータ並列処理の一つである (図 4.2 (c) 参照)．

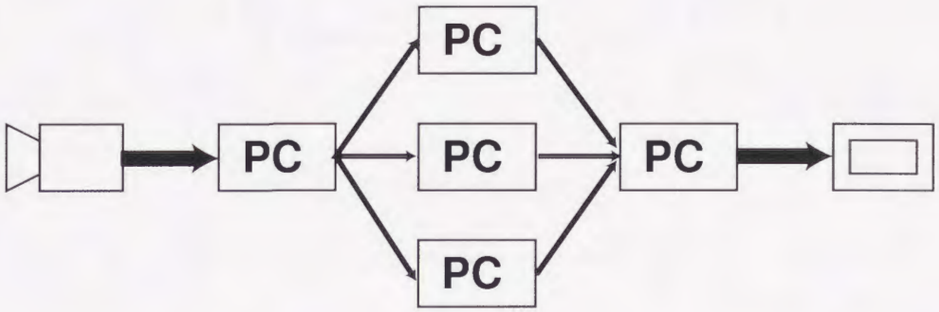
機能並列処理 図 4.2 (d) に示すように，同じフレームのデータに複数の計算機で同時に異なる処理を行う並列処理であり，原理的には，データ並列処理と同様の効果がある．ただし，同時に異なる処理を行わなければならないので，処理の内容によってはこの並列処理を実現できるとは限らない．また，計算機間で処理量が異なると，処理量の最も多い計算機の処理時間に，全体の処理時間が拘束され，計算機台数分の処理性能の向上が得られない．

ここで想定する実時間画像処理は，入力としてビデオカメラのように一定のフレーム速度[1]で次々とフレームデータが与えられるものに対し，各フレームデータに対する処理を滞ることなく実行するものであり，上述のような並列方式として実現するには，以下の機構を実現する必要がある．

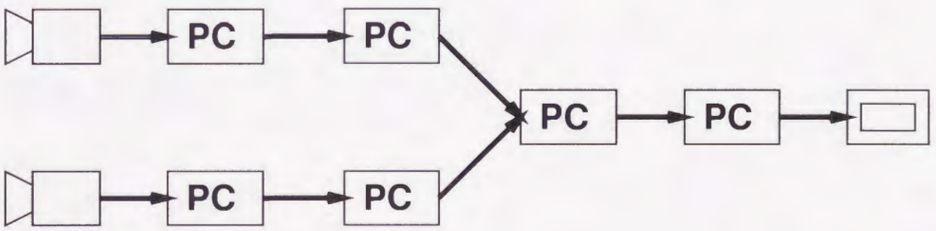
- 各計算機内の処理が決められた時間内に終了する．これは各計算機における処理量を調節することにより実現できる．
- 計算機間の通信が決められた時間内に終了する．このためには，高速なネットワークとそれを効率良く使う機構が必要である．
- 処理遅れ，通信遅れが起こった場合はエラー処理が起動される．このためには同期機構およびエラー処理機構が必要である．

[1]NTSC に準拠したカメラを利用しているため，フレーム速度は 30fps である．

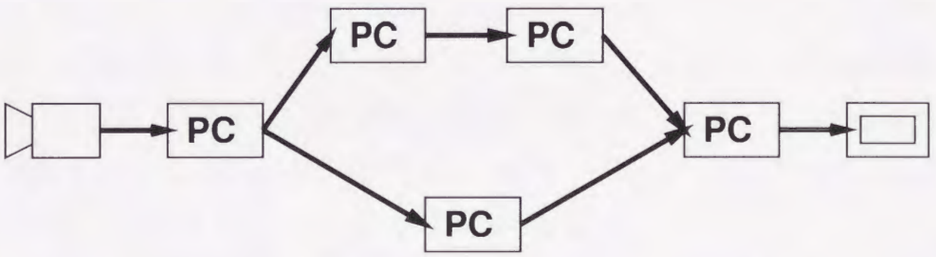
(a) パイプライン並列処理



(b) データ並列処理



(c) 複数カメラのデータ並列処理



(d) 機能並列処理

図 4.2: 並列処理方式

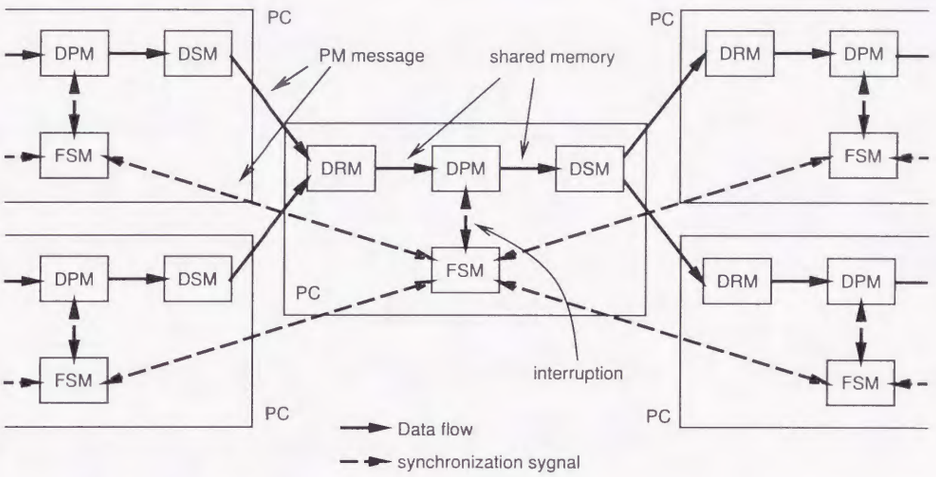


図 4.3: モジュール構成

以降の節では，これらの機構を実現する方法を含む，PC クラスタ上での実時間画像処理システムの詳細について説明する．

4.5 実装方式

4.5.1 モジュール構成

並列動画像処理を効率的に実行するために，本システムのノード PC 内では，プロセスとして (図 4.3) のような四つのモジュールが動作している．このようにプロセスを複数動作させることにより，データの受信，処理，送信を並行に行うことができ，外部からのデータの受信にも即座に対応できるようになっている．また，データの送受信のためにはバッファが必要であり，これは複数のモジュールからアクセスできるように共有メモリ上に実現され，複数のモジュールが共有するバッファ管理オブジェクトにより管理されている．以下，各モジュールと各オブジェクトについて説明する．

受信バッファオブジェクト: Receive Buffer Object (RBO) 他の PC から受信したデータを保持しておくバッファを管理するオブジェクト．

- 書き込み可能な受信バッファ内の位置を DRM に知らせる．この情報を基に，転送元 PC から受信したデータはこの位置に書き込まれる．

- DPM の要求に従って，DPM が次に処理すべきデータへのポインタを返す.
- 処理終了の知らせを受け取って，そのデータの位置を書き込み可能にする.

送信バッファオブジェクト: Send Buffer Object (SBO) 他の PC へ送信するデータを保持しておくバッファを管理するオブジェクト.

- 書き込み可能な送信バッファ内の位置を DPM に知らせる．この情報を基に，DPM の出力データはこの位置に書き込まれる.
- DSM の要求に従って，DSM が次に送信すべきデータへのポインタを返す.
- 送信終了の知らせを受け取って，そのデータの位置を書き込み可能にする.

データ受信モジュール: Data Receiving Module (DRM) データ受信に関する情報のやりとりを行うモジュール.

- 転送元 PC からの各フレーム分のデータ転送が終了した知らせを受け取り，RBO に伝える.
- 書き込み可能になった受信バッファの位置をチェックし，転送元 PC にその位置を伝える.

データ処理モジュール: Data Processing Module (DPM) 実際の画像処理を行うモジュール.

- 処理開始の割り込み信号を受けとると，受信バッファからデータを読み込み，そのデータを入力として画像処理を行い，処理結果を送信バッファに書き込む.
- 処理が終了したデータについて RBO に伝える.
- 書き込みが終了したデータについて SBO に伝え，また，書き込みが終了したことを割り込み信号として DSM に送る.

データ送信モジュール: Data Sending Module (DSM) データ送信に関する情報のやりとりを行うモジュール.

- 処理モジュールからの各フレーム分のデータ書き込みが終了した知らせを割り込み信号として受け取り，そのデータを転送先 PC に送信する.

- 送信が終了したデータについてSBOに伝える

フレーム同期モジュール: Frame Synchronizing Module (FSM) PC間の同期をとるためのモジュール.

- FSMどうしの通信によって，4.5.4節で述べるフレーム同期信号 (Frame Synchronization Sigral(FSS)) をデータの流れにそって上流から下流へ向けて送受信する.
- FSSを受け取ると，処理開始を通知する割り込み信号をDPMに送る.

4.5.2 データ転送機構

データ転送にはRWCPによって開発されたMyrinet上の通信ライブラリPM[37]を用いている．以下にPMライブラリの主な特徴を挙げる.

- 7.5 μsの低レイテンシ
- 118MByte/秒の高スループット
- ゼロコピー通信をサポート
- 割込み受信をサポート

非圧縮フルサイズカラー画像 (640 × 480,1画素4バイト) を30fpsで転送するためには，毎秒 $640 \times 480 \times 4 \times 30 = 36864000 \simeq 35$Mバイトのスループットが必要であるが，PMライブラリはこの条件を満たしている.

また，ゼロコピー通信と割込み受信のサポートにより，受信側のPCはCPUに負荷をかけることなくデータの受信を行うことが可能である．一方，データ転送の単位が最大8Kバイトのため，画像データを分割して送信する必要があるので，送信側のPCはデータ送信にCPUを利用することになってしまう．このため，CPUを一つしか持たないPCを利用する場合は,

$$(\text{データ処理時間}) + (\text{データ送信時間}) \leqq (1\text{フレーム時間})$$

となる必要がある．これではデータ送信先が複数ある場合，データ送信時間が長くなり，十分なデータ処理時間が得られなくなってしまう．一方，本システムのようにCPUを二つ持っているPCを利用する場合は

(データ処理時間) ≦ (1 フレーム時間)
(データ送信時間) ≦ (1 フレーム時間)

を満たせばよいので，データ送信の量に関係なく一定のデータ処理時間を確保できる．また，データ送信先が複数ある場合も 1 フレーム時間まではデータ送信時間を延ばすことができる．

4.5.3 同期データ転送と非同期データ転送

本システムがサポートするデータ転送は，受信時の同期 (4.5.5 節参照) のとりかたによって，同期データ転送と非同期データ転送に分けられる．

同期データ転送 動画像データおよびその処理結果などのように，フレームごとにデータを転送するものであり，毎フレーム届かなければならないデータを転送するのに利用される．同期データ転送は各フレームの処理の開始時に，必要なデータが到着しているかどうかチェックする (データ転送同期．4.5.5 節参照)．特に，同期データ転送で転送されるデータを同期データと呼ぶことにする．DPM は各フレームの処理開始時に同期データを受け取る．

非同期データ フレームタイミングとは関係なくデータを転送するものであり，1 フレームに何回もデータ転送されたり，まったくデータ転送されなくても構わないデータを転送するのに利用される．通常は，フィードバックデータや協調処理のための制御信号などに利用される．特に，非同期データ転送で転送されるデータを非同期データと呼ぶことにする．DPM は処理における任意の段階で最新の非同期データを受け取ることができる．

4.5.4 時刻管理

分散並列計算機環境において実時間画像処理を行う場合，時刻の管理が重要である．これは，複数のカメラによって撮影された画像が同一時刻のものでなければならず，また各計算機でのデータの処理と計算機間の同期データの転送を同じ時間間隔で行わなければならないからである．このために，以下のことを行っている．

- すべてのカメラに外部同期信号を与えることによって，すべてのカメラを完全に同期させる．これにより，まったく同じ時刻の画像を撮影することができる．

- カメラが接続された各 PC で同時刻に画像獲得を開始する．時刻を合わせるために NTP[38] を導入し，PC の内部時計を一致させている．NTP を使うことによって，数ミリ秒以下の精度[2]で PC の時刻を合わせることができるので，33 ミリ秒ごとの画像獲得に対しては十分な精度である．また，データにはフレーム番号が付けられている．これにより，同一フレーム番号の画像は同じ時刻の画像であることを保証する．

- カメラが接続された PC は，カメラからの垂直同期信号のタイミングに合わせて，フレーム同期信号 (FSS) を発生する．FSS は同期データの流れに沿って上流の PC から下流の PC へと転送される．この FSS によって各 PC 間のデータ処理とデータ転送の同期をとることができる (同期機構の詳細は 4.5.5 節を参照)．

4.5.5 同期機構

本システムでは，実時間でデータを受信，処理，送信することが要求される．これらのデータ転送，処理を滞りなく実行するために，PC 間の同期をとることが必要である．そこで，本システムでは，

- データ転送同期

- データ処理同期

の 2 種類の同期機構を提供し，同期の実現のために FSS を利用している．

FSS は DPM に処理のタイミングを知らせるための信号であり，カメラの垂直同期信号に合わせてすべての PC に送信される．これを実現するために以下の手順で処理を行う．

1. FSM が前段の PC の FSM から FSS を受け取る．

2. 次段の PC の FSM に対し FSS を送信する．同時に，DPM に対し処理開始を通知する割り込み信号を発信する．

3. DPM は割り込み信号を受信すると，次のフレームの処理を開始する．

この FSM からの割り込み信号によって，DPM は処理遅れ，同期データの未受信といったエラーの検出をすることができる．

[2]実際には PC クラスタのような通信遅延が小さい環境では 1 ミリ秒以下の精度と言われている．

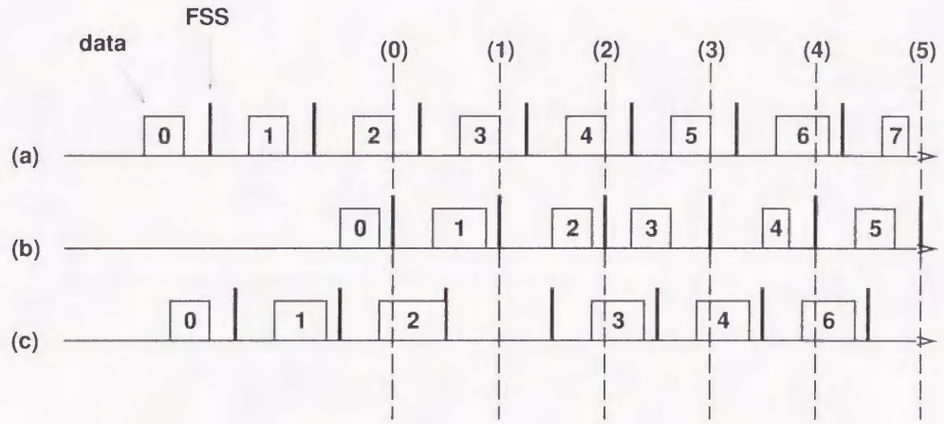


図 4.4: 複数の PC からのデータ受信

さらに，複数の PC からデータを受信する場合，同時刻に処理されるべき全てのデータを待たなければならない．図 4.4において，注目する PC は 3 台の PC(a),(b),(c) からデータを受信している．それぞれの PC からの FSS の受信時刻は，処理経路の長さの違いから異なることがある．よって，同時刻に画像キャプチャボードにより発せられた FSS が全て受信されるまで待機し，それらがすべて受信されると (つまりこの例の場合は (b) からの FSS を受信すると)，FSM は次段の PC に対して FSS を送信し，処理モジュールに処理開始を通知する割り込み信号を発信する．このような処理のために，FSS はフレーム番号を保持している．

データ転送同期は，DPM に同期データが到着しているかどうかをチェックする同期である．この同期により，前段の PC からの同期データ転送の遅れや前段までの PC における同期データのデータ落ちを検出することができる．図 4.5は連続した二つの PC のタイミングチャートである．上半分と下半分がそれぞれ一つの PC の動作を表しており，それぞれの PC の中では，上から順に DRM におけるデータ受信時間，DPM におけるデータ処理時間，DSM におけるデータ送信時間を表している．また，上段の PC から下段の PC へと同期データが転送されている．DRM が受け取る同期データは，正常時は二つの FSS の間に受信される．そのため，FSS を受信したときに DPM は処理を開始することができる．しかし，(a) でデータ転送の遅れが起こっており，処理開始時に同期データが到着していない．このため，DPM は処理を開始することができず，プログラマによって指定されたエラー処理が実行されることになる．

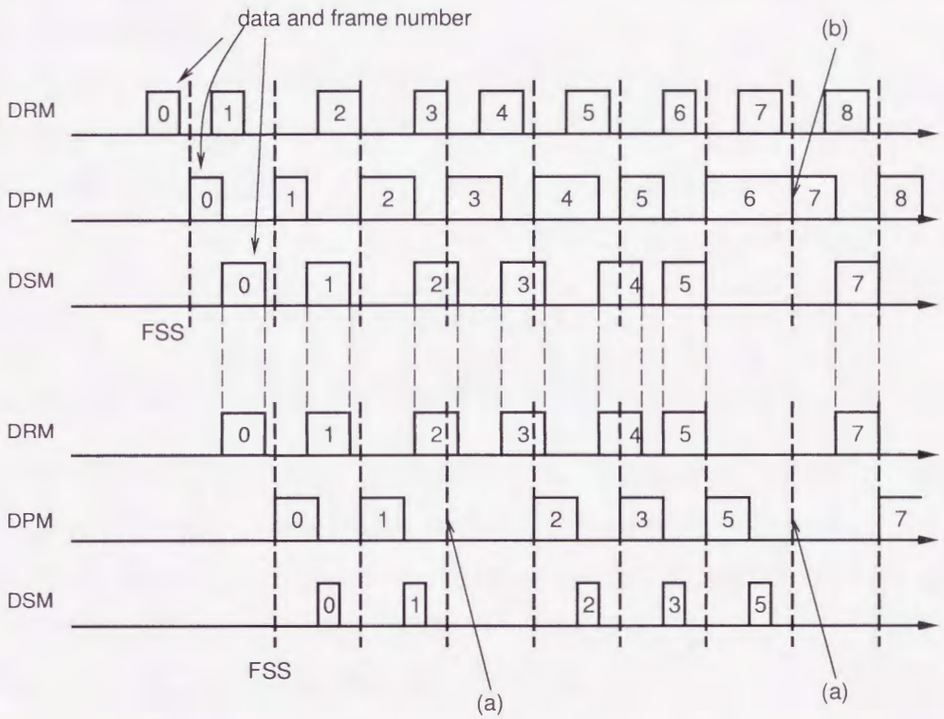


図 4.5: データ転送同期とデータ処理同期

また，複数 (N 個とする) の PC からの同期データを受信する PC では，同時に処理すべき同期データが揃うのを待つ必要がある．このため，必要な同期データの数 $n_{barrier}(1 \leq n_{barrier} \leq N)$ をプログラマは指定する．$n_{barrier}$ の値は通常は N が指定されることになるが，処理の内容によっては N よりも小さな値を指定しても良い．例えば，多眼ステレオ視 [39] のように複数のデータからロバスト推定を行う処理では，必要な画像サンプル数を指定し，その数以上の画像サンプルが集まった場合は処理を実行できるようなことが考えられる．この場合，必要な画像サンプル数を $n_{barrier}$ とすることで，$n_{barrier}$ 個以上の同期データが揃った場合には処理を実行することができる．なお，処理開始に間に合わなかった同期データは破棄される．

さらに，複数のフレームの同期データを使用する処理 (フレーム間差分処理など) の場合も，同時に処理すべき同期データが揃っているかどうかをチェックする必要がある．このときはデータ落ち (4.5.6 節参照) によって欠落したフレームデータがあったときへの対応によって，以下のような対応が考えられ，プログラマがどちらかを選択することができる．

- データ落ちしたフレームデータが関係する処理はまったく行わない．
- データ落ちしたフレームデータの前のフレームデータを代用する．

一方，データ処理同期は，DPM の処理が 1 フレーム時間以内に終了しているかどうかをチェックする同期である．図 4.5において，(b) でデータ処理の遅れが起こっており，処理開始時になっても前のフレームのデータ処理が終了していない．このため，そのままでは DPM は処理を開始できず，また，DSM はデータを送信することができない．この場合もプログラマが指定したエラー処理が実行されることになる．

4.5.6 同期機構におけるエラー処理

2 種類の同期機構におけるエラー処理として，以下の方法が考えられる．

(a) データ転送の遅れ

1. データが到着するのを待って，処理を開始する．
2. 次の FSS まで処理を行わない．次の処理では最新のデータを処理する．

(b) データ処理の遅れ

1. 処理を打ち切り，次のフレームのデータに対する処理を開始する．
2. 終了するまで処理を継続する．次の処理では最新のデータを処理する．

これらのエラー処理の組み合わせにより，以下の 3 種類のエラー処理モードを用意し，アプリケーションプログラマはその中から適切なものを選択することができる．なお，(a) の 2は次のフレームのデータに対する処理を行うために現在のフレームのデータを落とすエラー処理であり，(b) の 1は次のフレームのデータに対する処理を行うために現在のフレームのデータに対する処理を途中で打ち切るエラー処理であるから，これらのエラー処理の組み合わせは効果が重複しており，無意味なので考えない．

データ落ち型 (a) は 2，(b) は 2の組み合わせ (図 4.6 (a) 参照)．完全なデータだけを出力するが，エラーのときはデータ落ちが起こる．この方式は，処理が遅れたときに決められた時間以内に出力が得られないことになるので，厳密には実時間システムとは言えなくなる．しかし，データを落とすことによってすぐに遅れの状態から正常な状態に復帰できるので，実際の動画像処理としては最も利用される方式である．

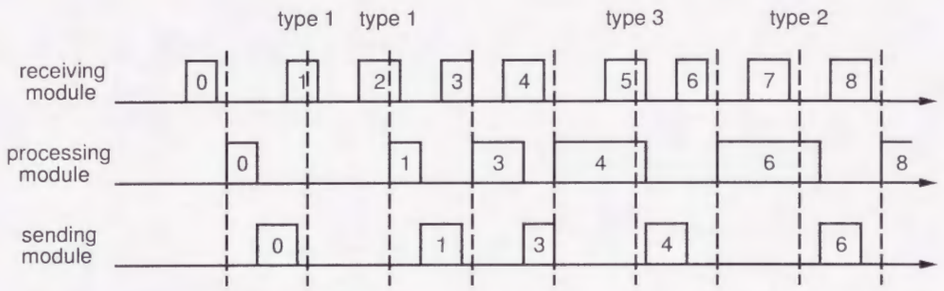
不完全データ転送型 (a) は 1，(b) は 1の組み合わせ (図 4.6 (b) 参照)．データ落ちは起こならいが，データ処理が遅れたときは不完全なデータが出力される．この方式のみが厳密な意味での実時間処理となる．ただし，計算機の処理能力と比べて処理量が多くなりすぎると，完全な出力がまったく得られなくなってしまうので，プログラマが負荷分散を慎重に行う必要がある．なお，このとき出力されるデータは，以下の中からアプリケーションプログラマが選択する．

1. 処理途中のデータ
   これを選択するとよい処理としては，幅優先探索処理，繰り返し型の最適化処理，粗精型のコンピュータグラフィックス生成処理のように，徐々に結果が良くなっていく処理などが挙げられる．この場合，打ち切り回数などを指定することなく，時間がきたら処理を打ち切って結果を出力するということができる．
2. 前フレームの処理結果
   これを選択するとよい処理としては，移動物体の位置計算のように，前後のフレームの処理結果の相関が強い処理などが挙げられる．
3. あらかじめ指定されたデータ
   特別なデータを出力することで，処理が途中で打ち切られたことを後段の PC に

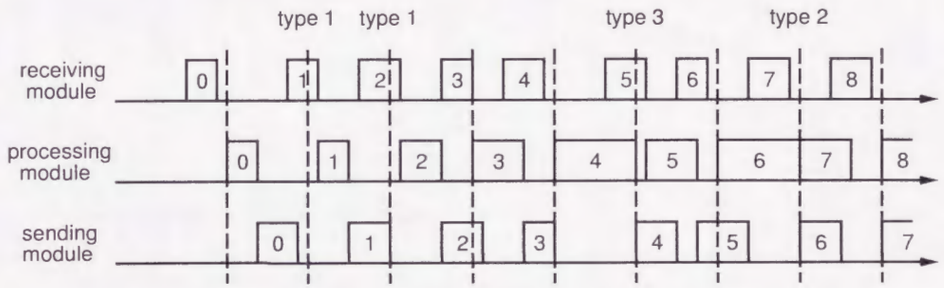
知らせることができる．

完全保持型 (a) は 1，(b) は 2の組み合わせ (図 4.6 (c) 参照)．データ落ちも起こらず，完全なデータだけを出力する．しかし，この方式は，一度エラーが起こるとそれ以降の実時間性の保証ができず，さらに最悪の場合は受信バッファが溢れてしまう．

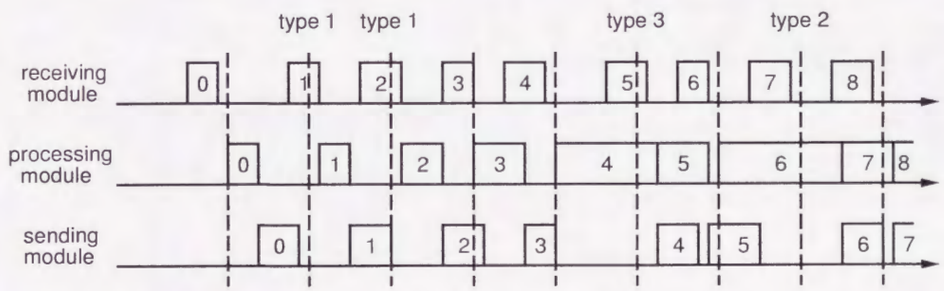
データ落ち型のエラー処理については，無駄なデータ処理を回避するための機構を導入している．データ落ちが起こった PC(PC0 とする) よりも下流に同期データの統合を行う PC(PC2 とする) がある場合，その落ちたフレーム番号の同期データに対する処理が PC2 上では行われない可能性がある (行われるかどうかは，PC2 の $n_{barrier}$ の値によって決まる)．処理が行われないのであれば，PC2 の上流の PC(PC1 とする) においてそのフレーム番号の同期データを処理することは無駄な処理である．この無駄な処理のために，図 4.7 (a) のようなことが起こる可能性がある．ここでは，PC0 では 1，3，5 フレーム目が，PC1 では 2，4 フレーム目の同期データが落とされてしまい，それらの同期データが統合される PC2 ($n_{barrier} = 2$ とする) では 0 フレーム目と 6 フレーム目の同期データの処理しか行うことができない．このような場合，PC0 でデータ落ちが起こった時点で PC2 でそのフレーム番号の同期データに対する処理が行われないことが確定するので，その時点で PC1 でのそのフレーム番号の同期データに対する処理をやめさせればよい．これを実現するために，ある PC でデータ落ちが生じた場合，そのフレーム番号の同期データに対する処理のキャンセル信号を近傍の PC に送信する．図 4.7 (b) では，データ落ちの情報が周辺の PC に以下のように伝えられる．(a) 処理時間超過の検出 (b)1 フレーム目の同期データが落とされることを伝える．(c)1 フレーム目の同期データに対する処理が不要であることを伝える．(d)1 フレーム目の同期データに対する処理を中断させる．(e)(f)(g)(h) は 3 フレーム目の同期データに対する (a)(b)(c)(d) と同様の処理である．(i)(j) も (a)(b) と同様であるが，PC1 からも同様に (k)(l) が届くので，5 フレーム目の同期データに対する処理は終了する．(m)(n)(o)(p) は 7 フレーム目の同期データに対する (a)(b)(c)(d) と同様の処理である．このように無駄な処理をやめさせることによって，PC2 は 0，2，4，6 フレーム目の同期データを処理することができる．



(a) データ落ち型

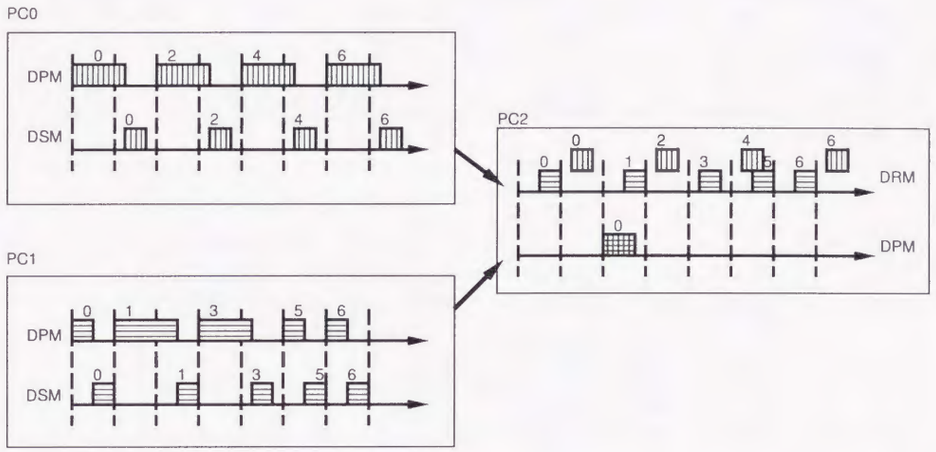


(b) 不完全データ転送型

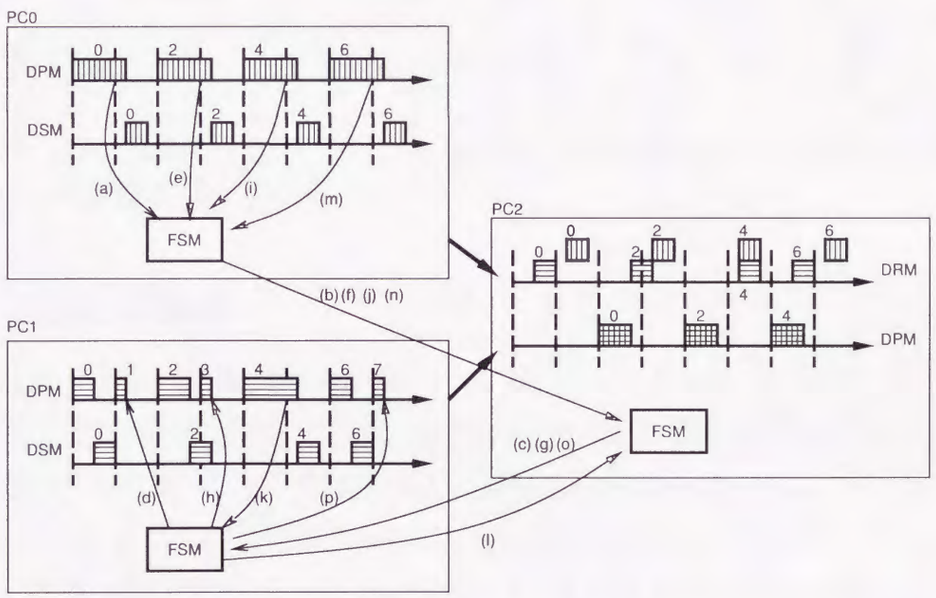


(c) 完全保持型

図 4.6: 同期におけるエラー処理方式



(a) 無駄な処理をやめさせない場合



(b) 無駄な処理をやめさせる場合

図 4.7: データ落ちによる無駄な処理への対応

表 4.1: 画像 1 枚あたりの転送時間 (単位はミリ秒)

画像サイズ	最小時間	平均時間	最大時間
640 × 480 (24 ビット/画素)	8.9	9.1	9.2
320 × 240 (24 ビット/画素)	2.2	2.3	2.3

表 4.2: 画像 1 枚あたりのブロードキャスト転送時間 (単位はミリ秒)

受信 PC 数	最小時間	平均時間	最大時間
2	18.3	18.3	18.4
3	27.1	27.2	27.4
4	35.8	35.9	36.1
5	44.5	44.6	44.8

4.6 実験

本節では，4.3.1節で述べた PC クラスタ上に，本章で提案した機能を実現し，それについて実験による評価と考察を行う．

4.6.1 性能評価実験

まず，画像データの転送能力は表 4.1のようになった．これは示されたサイズの画像を同期データ転送によって転送するときの，送信側 PC における画像 1 枚あたりの送信時間である．この結果から本システムは 30fps でデータ転送可能な速度があり，転送時間も安定していることから，実時間画像処理システムとして利用できることが分かる．

つぎに，複数の PC へのブロードキャスト転送の能力は表 4.2のようになった．これは，640 × 480 (24 ビット/画素) の画像を示された台数の PC に同期データ転送によってブロードキャスト転送したときの総送信時間である．この結果から，本システム上では 3 台以下の PC に実時間 (30fps) でブロードキャスト転送できることが分かる．

最後に，1 台の PC の処理能力を表 4.3に示す．これは 640 × 480 の画像に対する，示された画像処理アルゴリズムの実行時間である．この結果が示すように，実時間画像処理

表 4.3: 画像処理アルゴリズムの実行時間 (単位はミリ秒)

画像処理アルゴリズム	ビット/画素	処理時間
平均値フィルタ (3×3)	24	107.2
背景差分	24	48.0
テンプレートマッチング (8×8)	24	2,529.1
テンプレートマッチング (8×8,MMX 利用)	24	1,318.0
2 値化 (大津の方法)	8	12.7
エッジ検出 (Sobel)	8	111.2

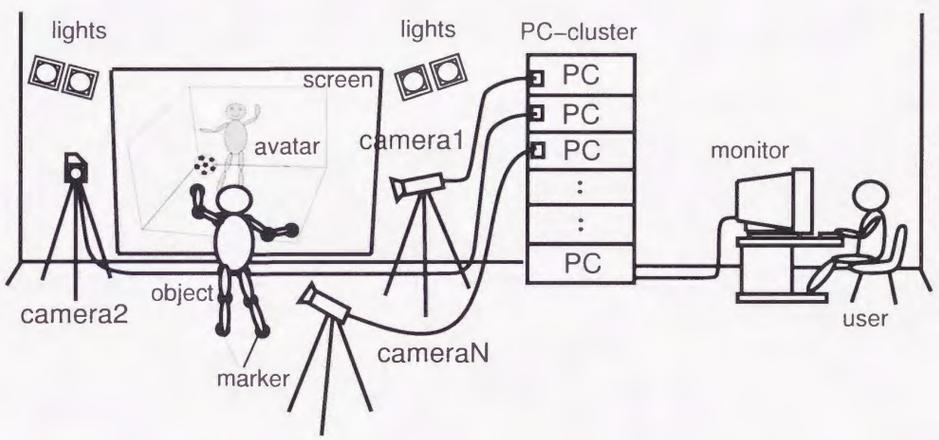


図 4.8: モーションキャプチャシステムの外観

(30fps) を行うためには現在の PC の性能は必ずしも十分ではない．しかし，計算量削減の戦略を導入することにより実時間画像処理アプリケーションの実現は可能である．この具体的な事例について 4.6.2 節で述べる．

4.6.2 実アプリケーションによる実験

ここでは，本システムを利用した実アプリケーションとして実装した，オンラインモーションキャプチャシステムについて述べる [24, 25]．このシステムは図 4.8のような装置を利用している．N 台のカメラ (実際は 2 台) で撮影した動画像を PC クラスタ上のモーションキャプチャシステムによって被験者の姿勢を推定し，プロジェクタによってスクリーン上

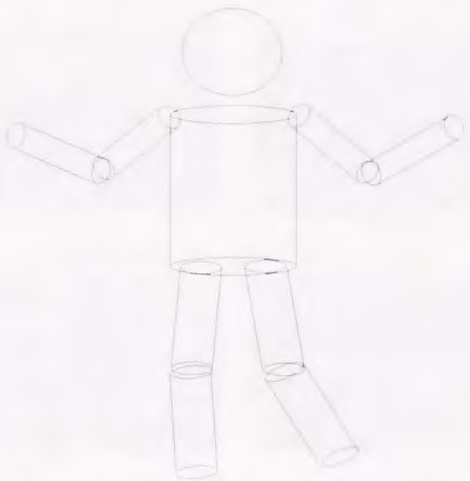


図 4.9: 多関節人体モデル

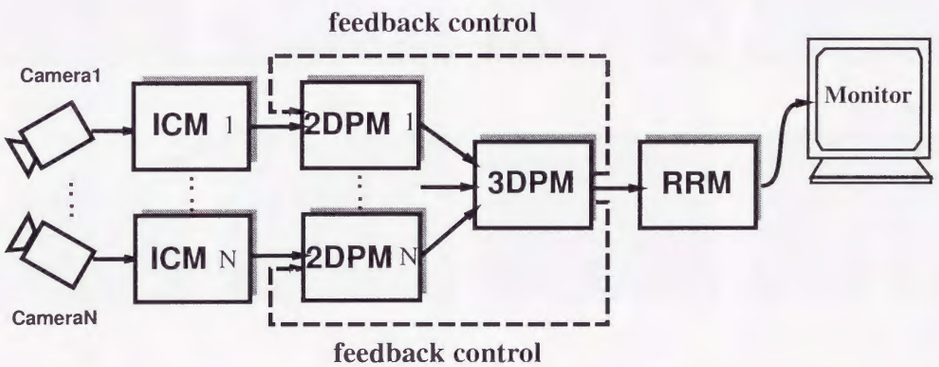


図 4.10: モーションキャプチャシステムの構成

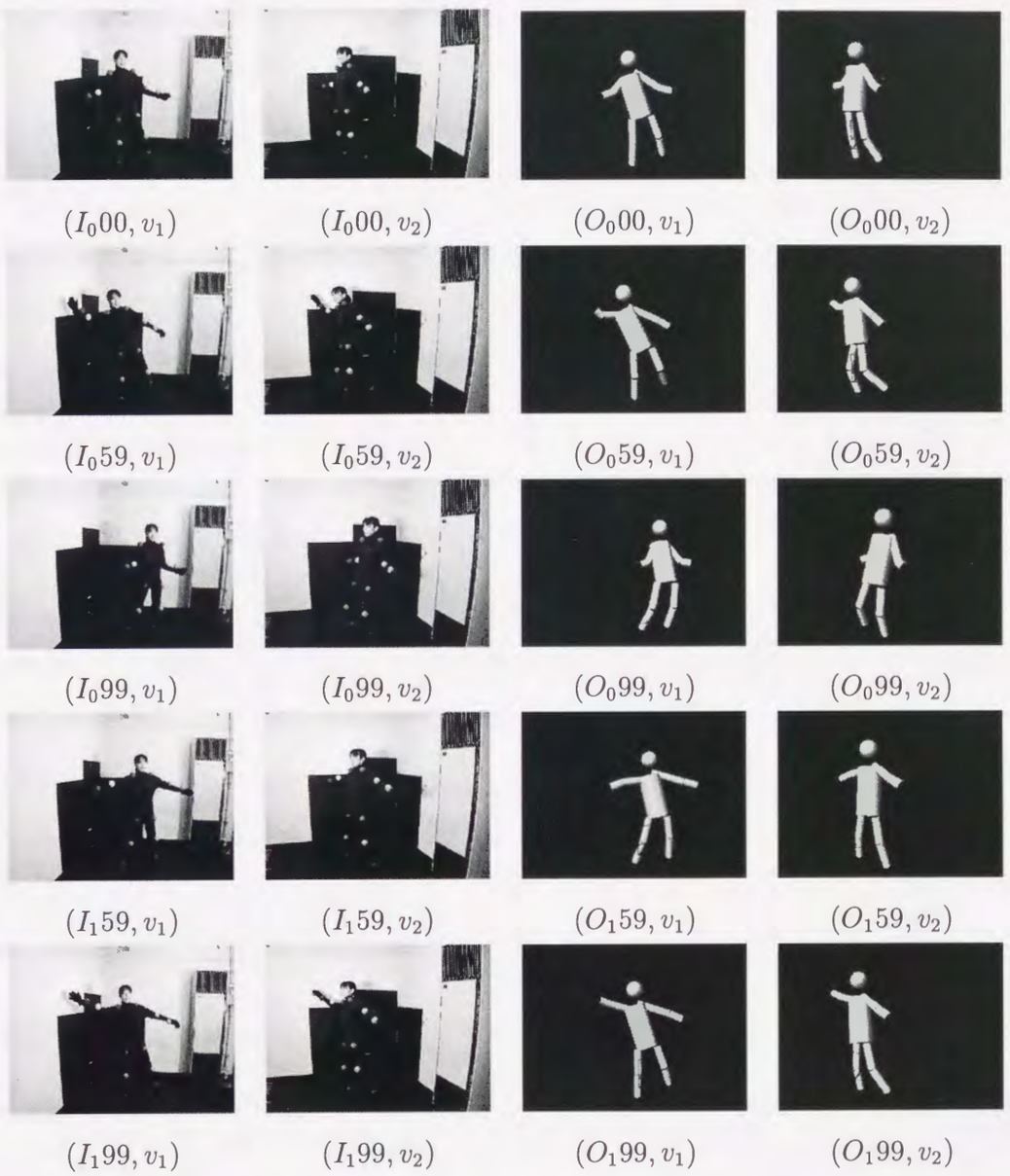


図 4.11: モーションキャプチャの入力画像と出力画像

に投影される．被験者は図 4.9のようなモデルによって表現されており，その関節に 12 個のカラーマーカを装着している．これらのマーカを追跡することで関節の位置を計算できる．そして，その位置をモデルに当てはめることで被験者の姿勢を推定することができる．このような処理を実現するために，各 PC の DPM では以下の処理を行っている (図 4.10 参照)．

Image Capturing Module(ICM) 画像獲得と画像のリサイズ (640×480 から 320×240) を行う．

2D Processing Module(2DPM) 2 次元画像上でのカラーマーカの位置を計算する．具体的には，背景差分により探索範囲を限定し，色情報を基にカラーマーカを抽出する．そして，カラーマーカの 2 次元画像上での重心の位置を出力する．

2D Processing Module(3DPM) 3 次元空間上でのカラーマーカの位置を計算する．具体的には，各 2 次元画像上での位置より，ステレオ法によってカラーマーカの 3 次元空間上での位置を計算し，その結果を出力する．

Real-time Rendering Module(RRM) 仮想空間を表示する．具体的には，カラーマーカの 3 次元空間上での位置より，人体モデルの位置姿勢計算を行い，仮想空間のレンダリングを行う．

このシステムの入力動画像と出力動画像を図 4.11 に示す．この処理は 30fps で実現されており，入力から出力までのレイテンシは 0.2 秒程度である．

4.7 考察

性能評価実験の結果からは，データ転送の性能は十分であるが，データ処理の性能はまだ必ずしも十分ではないことが分かった．しかしながら，階層的粗精サーチや階層的処理，前フレームの処理結果を利用した処理範囲の限定など，処理量を削減するための戦略を利用すれば，モーションキャプチャシステムのような実用的な実時間画像処理アプリケーションを構築することが可能であることも判明した．さらに，近年の CPU の性能向上はめざましく，データ処理の性能は急速に改善されることが期待される[3]．特に，性能評価実験の結

[3]実際，すでに 800MHz の PentiumIII が発表されており，実験時よりも CPU の速度は約 2 倍になっている．

果から分かるようにマルチメディア用拡張命令セットを利用することで大幅な高速化が実現されており，このような拡張命令セットを利用することによってもデータ処理の性能は大幅に改善されると見込まれる[4]．また，多視点画像処理でのPCクラスタの優位性は，現在のデータ処理性能でも揺らぐものではなく，本システムの有効性は今後ますます高まっていくと考えられる．

今後の課題としては，1台のPCに複数のCPUを搭載していることを利用したデータ処理性能の向上が挙げられる．本システムでは1台のPCに搭載された2個のCPUのうち1個だけをデータ処理に利用し，もう1個のCPUはデータ送信にのみ利用している．しかし，4.6.1節の性能評価実験からデータ処理能力よりもデータ送信能力に余裕があり，データ送信を行うCPUは休止している時間が長いことが分かった．そこで，以下の2通りの方法によって，両方のCPUをデータ処理に利用し，データ処理性能を向上させることが考えられる．

1. パイプライン並列処理方式による利用
   2個のCPUを利用してパイプライン並列処理を行う．パイプラインの2段目のCPUはデータ処理とデータ送信の両方を行う．データ送信のみを行っていた2段目のCPUでデータ処理も行うことになるので，データ処理能力を向上させることができる．

2. データ並列処理方式または機能並列処理方式による利用
   2個のCPUで同じフレームのデータに対する処理を並列に行い，データ処理時間を短縮する．データ送信用のCPUがないため，1個のCPUを搭載したPCを利用するときと同様に

   $$(\text{データ処理時間}) + (\text{データ送信時間}) \leqq (1\ \text{フレーム時間})$$

   を満たす必要があるが，データ送信時間が短い場合は処理能力を向上させることがでる．

ただし，パイプライン並列処理方式ではレイテンシが増加するので注意が必要である．方法1では，PC内のパイプライン1段につき1フレーム時間のレイテンシが生じるのでレイテンシは2フレーム時間となり本論文で述べた実装方式と変わらないが，方法2では，1

[4]PentiumIIIから導入されたストリームSIMD拡張命令セットを利用するようになれば，さらにデータ処理性能，特に3次元座標計算のような浮動小数点演算性能が大幅に向上すると考えられる．

フレーム時間内に処理と送信が終了するのでレイテンシを 1 フレーム時間に抑えることができる．1 台の PC に 3 個以上の CPU を搭載した場合も上記の方法によってデータ処理性能を向上させることができるが，方法 1では CPU の数に比例してレイテンシが大きくなることに注意しなければならない．

第 5 章

プログラミングツール RPV

5.1 目的

本章で述べるプログラミングツール RPV[40, 41, 42] は分散型並列計算機の一種である PC クラスタ上の実時間画像処理プログラミングツールである．分散型並列計算機上の汎用的なプログラミングツールとしては，PVM[30] と MPI[31] がよく知られている．さらに，分散型並列計算機上の画像処理プログラミングツールとしては，谷口らの PRIME[43]，松尾らの VIOS-III[44]，Lückenhaus らの研究 [45]，Kim らの研究 [46]，Rodin らの iRos[47]，Squyres らの研究 [48]，Gennart らの CAP[49] など数多く行われている．しかし，これらは実時間画像処理のためのツールではないので，第 4 章で述べた実時間データ転送機構，同期機構，エラー処理機構が実現されていない．そのため，アプリケーションプログラマがこれらの機能を実現しなければならないが，これはハードウェア，OS，ネットワークなどのシステムの詳細まで熟知している必要があり非常に困難な作業である．また，MPI/RT[50] は MPI を基にした実時間並列処理プログラミングツールである．しかし，これは通信の QoS(Quality of Service) を確保することに主眼がおかれており，データ処理同期やそのエラー処理は考慮されていない．この点については IEEE1394 規格 [51] を利用した実時間通信も同様である．したがって，やはりアプリケーションプログラマが実時間画像処理に必要な機能を実現しなければならないことになる．これに対して，本研究では実時間画像処理に必要な機能をアプリケーションプログラマから隠蔽し，PC クラスタ上で実時間画像処理プログラミングを簡便に記述することができるような環境を開発することを目指す．

5.2 RPV の概要

RPV は実時間並列画像処理アプリケーション構築のためのプログラミングツールであり，実時間並列画像処理に必要なデータ転送機構，同期機構，エラー処理機構といった機能を提供するものである．そのため，RPV を利用することにより，アプリケーションプログラマは容易に実時間画像処理アプリケーションを構築することができる．このとき，アプリケーションプログラマが記述するのは，PC 間のデータフロー情報と各 PC 上で実行されるデータ処理タスクのみである．RPV は C++上のクラスライブラリとして実装されており，アプリケーションプログラマは RPV が提供するクラス `RPV_Connection` と関数 `RPV_Invoke` を利用してそれらを記述する．

各 PC ではその PC で行うデータ処理タスクに合わせて関数 `RPV_Invoke` を起動しなければならないので，Multiple Programs Multiple Data(MPMD) モデルによるプログラミングを行うことになる．しかし，RPV では PC にごとに起動する関数 `RPV_Invoke` を切り替えることもできるので (5.4 節参照)，Single Program Multiple Data(SPMD) モデルによるプログラミングも可能である．SPMD モデルによるプログラミングは一つの実行ファイルしか必要としないので，大規模なシステムになっても実行ファイルの管理が容易であるという利点がある．

この RPV では，以下の手順でモジュールの生成と初期化が行われる．

1. クラス `RPV_Connection` にデータフロー情報を設定する．
2. 関数 `RPV_Invoke` が起動される．
3. 関数 `RPV_Invoke` の中でフォークが実行され，DRM，DPM，DSM，FSM の各モジュールが生成される．
4. 各モジュールのデータフローに関する設定を `RPV_Connection` によって初期化する．
5. データ転送を行う DSM と DRM の間で必要な情報を交換し，データ転送の準備を行う．
6. DPM は前処理関数 `pre_func`(5.4 節参照) を実行する．

すべての PC でこれらの初期化が終了したのち，データ処理が開始される．以下の節ではクラス `RPV_Connection` によるデータフロー情報の記述法と関数 `RPV_Invoke` によるデー

```
class RPV_Connection{
   int myPC_no;              自 PC 番号
   char* keyword;            データ処理のキーワード
   int input_PC_num;         入力同期データの数
   int* input_PC;            入力同期データの転送元 PC 番号
   int* input_data_size;     入力同期データの大きさ
   int input_frame_num;      1 回の処理に使用する入力同期データフレーム数
   int output_PC_num;        出力同期データの数
   int* output_PC;           出力同期データの転送先 PC 番号
   int* output_data_size;    出力同期データの大きさ
   int ainput_PC_num;        入力非同期データの数
   int* ainput_PC;           入力非同期データの転送元 PC 番号
   int* ainput_data_size;    入力非同期データの大きさ
   int ainput_data_num;      1 回の処理に使用する入力非同期データフレーム数
   int connect_PC_num;       使用する PC の数
   int* connect_PC;          使用する PC 番号
};
```

図 5.1: クラス RPV_Connection

タ処理タスクの記述法について述べる.

5.3 データフローの記述

クラス RPV_Connection は，各 PC におけるデータ送受信の情報を持つクラスであり，図 5.1のように定義される．このクラスの持つ情報をもとに，各 PC は PC 間のデータの送受信を行う．したがって，原理的にはクラス RPV_Connection の定義は PC ごとに異なってくるが，通常は PC に共通のファイル (コネクションファイルと呼ぶ) によって初期化されるようになっている．つまり，アプリケーションプログラマはコネクションファイルによって PC クラスタ全体のデータフローを記述することになる．このように一つのファイルによって記述することで，データフロー情報の無矛盾性の確認が容易になる (実際にコネクションファイルを読み込むときに矛盾がないかどうか確認される)．また，データフロー情報はファイルにまとめて記述されるため，ユーザはプログラムの修正，再コンパイルを

行うことなく，利用するPCと各PCで行う処理を変更することができる．コネクションファイルの形式は以下の通りである．

- 一つの行が一つのPCのクラスRPV_Connectionを初期化する．
- 各行は空白かタブで区切られた以下の列から成り，それぞれの列がクラスRPV_Connectionの各メンバに対応する．

  PCno PCクラスタ内のPCに付けられた番号

  keyword PCで行うデータ処理を指定する文字列

  i_PC 入力同期データの転送元PC番号

  i_size 入力同期データの大きさ

  i_num 1回の処理に使用する入力同期データフレーム数

  o_PC 出力同期データおよび出力非同期データの転送先PC番号

  o_size 出力同期データおよび出力非同期データの大きさ

  ai_PC 入力非同期データの転送元PC番号

  ai_size 入力非同期データの大きさ

  ai_num 1回の処理に使用する入力非同期データフレーム数

この中で，入力同期データと入力非同期データは区別するが出力同期データと出力非同期データは区別せずにまとめて記述する理由は，

- データ転送同期はデータの受信についてのみ必要であり，送信については同期データ転送と非同期データ転送の区別をする必要がないから
- 同じ出力データを同期データと非同期データの両方にブロードキャストできるようになるから

である．

```
void RPV_Invoke(
   RPV_Connection* connect,                    データフロー情報
   struct RPV_FSM sync_mode,                   同期モード
   int frame_num,                              処理する総フレーム数
   void* (*pre_func)(void*),                   前処理関数
   void* pre_func_arg,                         pre_func の引数
   void* (*user_func)(RPV_Input*, RPV_Output*,
            RPV_Ainput*, void*),               毎フレーム実行されるユーザ関数
   void* user_func_arg,                        user_func の引数
   void* (*post_func)(void*),                  後処理関数
   void* post_func_arg                         post_func の引数
);
```

図 5.2: 関数 RPV_Invoke

```
class RPV_Input{
   void*** data_ptr;    入力同期データへのポインタ
   int input_PC_num;    入力同期データの数
   int* input_PC;       入力同期データの送信元 PC 番号
   int frame_num;       1 回の処理に使用する入力同期データフレーム数
   int* frame_no;       入力同期データのフレーム番号
   int* data_size;      入力同期データの大きさ
};
```

図 5.3: クラス RPV_Input

```
class RPV_Output{
   void** data_ptr;     出力同期データへのポインタ
   int output_PC_num;   出力同期データの数
   int* output_PC;      出力同期データの送信先 PC 番号
   int* data_size;      出力同期データの大きさ
};
```

図 5.4: クラス RPV_Output

```
class RPV_Ainput{
   void*** data_ptr;       入力非同期データへのポインタ
   int ainput_PC_num;      非同期入力データの数
   int* ainput_PC;         非同期入力データの送信元 PC 番号
   int* frame_no;          非同期入力データのフレーム番号
   int* data_size;         非同期入力データの大きさ
   int ainput_data_num;    1 回の処理に使用する入力非同期データの数
};
```

図 5.5: クラス RPV_Ainput

5.4 データ処理タスクの記述

図 5.2に示す関数 RPV_Invoke が，各 PC 上で起動され，処理終了まで全体の制御を行う関数である．関数 RPV_Invoke の引数によってその PC で実行されるデータ処理関数が指示される．通常，クラス RPV_Connection のメンバ keyword にしたがって関数 RPV_Invoke の引数として渡すデータ処理関数を定める．このようにすることで，プログラムの修正，再コンパイルを行うことなくコネクションファイルを修正するだけで，各データ処理タスクを実行する PC を変更できる．これにより，PC クラスタ内に性能の異なる PC が存在するときに，その性能に見合ったデータ処理タスクを実行させることが容易になる．また，第 2 引数の sync_mode で，同期機構に関する選択を行う．この構造体により，処理遅れなどのエラーが生じた際のエラー処理モード，エラー処理で代替データの提供を選択した場合の代替データを選択する．これによりアプリケーションプログラマは処理に適した同期機構を選択することができる (4.5.5 節参照)．

関数 RPV_Invoke の動作は以下の通りである．

1. 起動後まず，フォークを実行することによって，DRM，DPM，DSM，FSM の各モジュールを生成する．このうち，DRM，DSM，FSM は RPV_Connection の設定にしたがって処理終了まで動作する．

2. DPM は RPV_Invoke の引数にしたがって，指定された関数を起動しながら処理終了まで動作する．具体的には以下の動作を行う (図 5.6参照)．

(a) 前処理関数 pre_func を実行する.

(b) 処理開始割り込み信号を待つ.

(c) ユーザ関数 user_func を実行し，1 フレーム分のデータを処理する.

(d) frame_num フレーム分のデータを処理が済んでいなければ，2bへ戻る.

(e) 後処理関数 post_func を実行し，処理を終了する.

3. すべての PC で処理が終了したことを確認し，RPV 全体の処理を終了する.

ユーザ関数 user_func の引数は，四つの入出力データへのポインタと一つのアプリケーションプログラマが自由に利用できる引数から成る．入力同期データ引数 (図 5.3参照) と入力非同期データ引数 (図 5.5参照) には，システムによって自動的に必要なデータが渡され，ユーザ関数はそれらのデータを読み込むことができる．また，出力データ引数 (図 5.4参照) には，ユーザ関数から書き込み可能であり，書き込みが終了すると自動的にそれらのデータは送信される．このようにユーザ関数は，データの送受信や同期について考慮する必要がなく，静止画像を入力とし，それを処理し，結果を出力する静止画像処理と同じ形式で記述可能である．そのため，動画像処理ということを意識することなく，容易にユーザ関数をプログラミングできる．また，クラス RPV_Connection において，ある PC のデータ転送元 PC とデータ転送先 PC にその PC を指定することで，その PC の出力データを次のフレームにおけるその PC の入力データにすることができる．これにより，過去の処理結果がユーザ関数の引数として渡されるので，過去の処理結果を利用するデータ処理タスクについても容易にプログラミング可能である.

5.5 RPV 標準ライブラリ

RPV では標準的なデータ形式をクラスライブラリとして用意する．これにより，プログラマの負担の軽減のほか，プログラマ間でのデータやプログラムの交換も容易になる．また，RPV では，標準的な画像処理を関数ライブラリとして用意する．用意する関数はマルチメディア拡張命令セットを利用するなどして高速化を行う．これにより，プログラマは高速な画像処理システムを低負担で実現できる．本節では RPV 標準ライブラリが用意するデータ形式と関数について述べる.

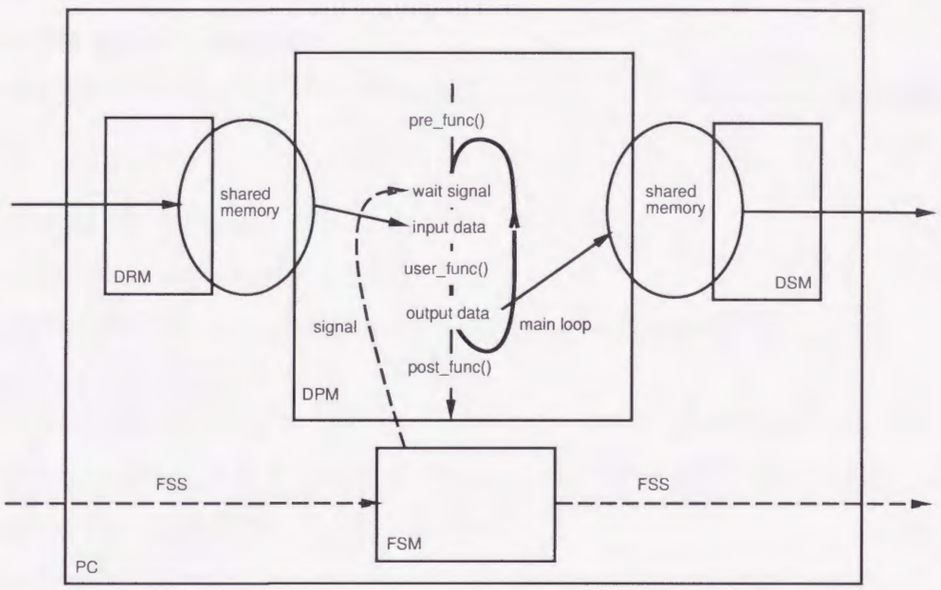


図 5.6: 関数 RPV_Invoke の動き

クラスライブラリ　RPV は以下のデータ形式のクラスライブラリを用意する．5.4 節で述べたように，RPV ではデータ処理を各フレームデータごとに処理するように記述するので，用意する画像形式も動画像形式ではなく静止画像形式である．

カラー画像 (24 ビット)

`RPV_RGB24<width, height>`

説明:24 ビット (RGB 各 8 ビットの) の画素から成る横 `widht`，縦 `height` のカラー画像

カラー画像 (32 ビット)

`RPV_RGB32<width, height>`

説明:32 ビット (RGB 各 8 ビットとダミー 8 ビット) の画素から成る横 `widht`，縦 `height` のカラー画像

濃淡画像 (8 ビット)

`RPV_GRAY8<width, height>`

説明:8 ビットの画素から成る横 `widht`，縦 `height` の濃淡画像

2 値画像 (8 ビット)

RPB_BIN8<width, height>

説明:8 ビット (下位 1 ビットのみを使用) の画素から成る横 widht，縦 height の 2 値画像

一般画像 (画素の型は Type)

RPV_IMAGE<width, height, Type>

説明:Type 型の画素から成る横 widht，縦 height の一般画像

関数ライブラリ RPV の用意する画像処理の関数ライブラリの例を挙げる．なお，入出力画像として一般画像型を代表として挙げているが，他の画像型用の関数も用意する．また，これら以外の関数も順次追加していく予定である．

平均値フィルタ

RPV_IMAGE<width, height, Type>* RPV_AverageFilter(RPV_IMAGE<width, height, Type>*) im, int size

第 1 引数: 入力画像

第 2 引数: 近傍領域の大きさ

返り値: 出力画像

説明: 画像の平滑化の一種．size × size 近傍の画素値の平均を注目画素の新しい画素値とする処理.

中間値フィルタ

RPV_IMAGE<width, height, Type>* RPV_MedianFilter(RPV_IMAGE<width, height, Type>*) im, int size

第 1 引数: 入力画像

第 2 引数: 近傍領域の大きさ

返り値: 出力画像説明: 画像の平滑化の一種．size × size 近傍の画素値の中間値を注目画素の新しい画素値とする処理.

ガウシアンフィルタ

RPV_IMAGE<width, height, Type>* RPV_GaussianFilter(RPV_IMAGE<width, height, Type>* im, int size, float S)

第 1 引数: 入力画像

第 2 引数: 近傍領域の大きさ

第 3 引数: 分散値

返り値: 出力画像説明: 画像の平滑化. `size` × `size` 近傍に分散値 S のガウシアンをたたみこむ処理.

Sobel フィルタ

`RPV_IMAGE<width, height, Type>* RPV_SobelFilter(RPV_IMAGE<width, height, Type>*)`

第 1 引数: 入力画像

返り値: 出力画像説明: 画像のエッジ検出. 3 × 3 近傍からエッジ強度を求める処理.

ラプラシアンフィルタ

`RPV_IMAGE<width, height, Type>* RPV_LaplacianFilter(RPV_IMAGE<width, height, Type>*)`

第 1 引数: 入力画像

返り値: 出力画像説明: 画像のエッジ検出. 3 × 3 近傍からエッジ強度を求める処理.

画像間差分

`RPV_IMAGE<width, height, Type>* RPV_Subtraction(RPV_IMAGE<width, height, Type>* im1, RPV_IMAGE<width, height, Type>* im2)`

第 1 引数: 入力画像 1

第 2 引数: 入力画像 2

返り値: 出力画像説明: 画像 `im1` と画像 `im2` の画素値の差の絶対値を求める処理.

5.6 プログラム例

ここでは, RPV プログラミングツールを用いたプログラム例を示す. 図 5.7に, システム例の概要を示す. このシステムは, 12 個のカラーマーカをつけた人間を 3 台のカメラで撮影した動画像から, マーカの 3 次元位置を計算し, 結果を表示する処理を実時間で行うシステムである. この例では, PC0, 1, 2 で 3 台のカメラから動画像の獲得, 平滑化を行い, PC3, 4, 5 でマーカの 2 次元位置を計算し, PC6 でマーカの 3 次元位置を計算し,

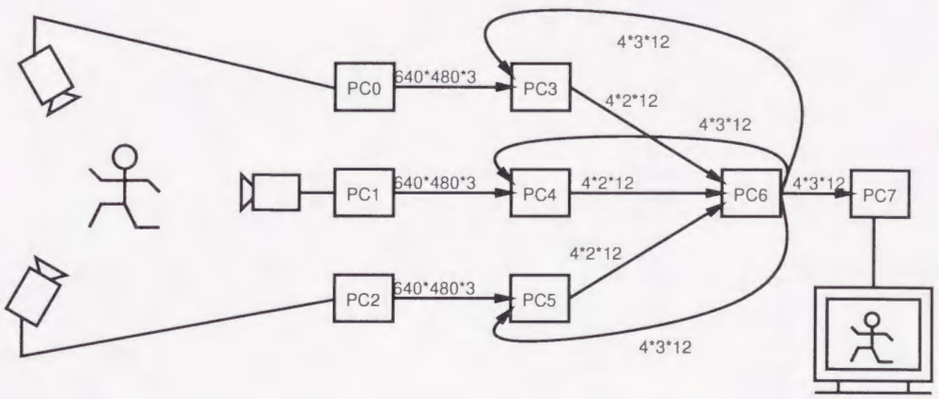


図 5.7: システム例

#PCno	keyword	i_PC	i_size	i_num	o_PC	o_size	a_PC	a_size	a_num
0	smooth	c	640*480*3	1	3	640*480*3	-	-	-
1	smooth	c	640*480*3	1	4	640*480*3	-	-	-
2	smooth	c	640*480*3	1	5	640*480*3	-	-	-
3	calc2D	0	640*480*3	1	6	4*2*12	6	4*3*12	1
4	calc2D	1	640*480*3	1	6	4*2*12	6	4*3*12	1
5	calc2D	2	640*480*3	1	6	4*2*12	6	4*3*12	1
6	calc3D	3,4,5	4*2*12,4*2*12,4*2*12	1	7,3,4,5	4*3*12,<,<,<	-	-	-
7	display	6	4*3*12	1	-	-	-	-	-

図 5.8: コネクションファイル例

PC7 でアニメーションを作成している．

図 5.8は，クラス RPV_Connection を初期化するコネクションファイルである．i_PC 列の c はカメラからの画像データを受け取ることを表し，o_size 列の < は左のデータをブロードキャストすることを表す．keyword 列によって，各 PC でのデータ処理タスクが指示されている．

図 5.9が全ての PC で起動されるメイン関数である．8 行目の RPV_Init で，ファイルからデータフロー情報をクラス RPV_Connection に読み込む．その際，ファイル内に矛盾がないか検査も行う．14 行目から 33 行目にかけては，クラス RPV_Connection のメンバ keyword にしたがって適切なユーザ関数を引数とする RPV_Invoke を起動している．例えば，keyword が “calc2D” の場合，前処理関数 ReadBackgraoud と，画像処理関数 Calculatre2D が引数として渡される．このようにユーザが一つのプログラムを各 PC で起動すると，指定された keyword にしたがって適切なユーザ関数が起動される．

各 PC で起動する関数は，図 5.10のように定義する．ここでは，10 行目から 19 行目ま

```
 1  #include <stdio.h>
 2  #include <rpv.h>
 3
 4  int main(void)
 5  {
 6      FILE *fp;
 7      fp = fopen("connect.cnt", "r");
 8      RPV_Connection* connect = RPV_Init(fp);
 9      fclose(fp);
10
11      struct RPV_FSM miss_FSM;
12      miss_FSM.FSM = RPV_DATA_MISSING;
13
14      if (strcmp(connect->keyword, "smooth") == 0) {
15          RPV_Invoke(connect, miss_FSM, 0, NULL, NULL, &Smooth, NULL, NULL,
 NULL);
16      }
17      else if (strcmp(connect->keyword, "calc2D") == 0) {
18          ReadBackgroundArg read_background_arg(BACKGROUND_FILE_NAME);
19          RPV_Invoke(connect, miss_FSM, 0,
20                     &ReadBackground, &read_background_arg, &
21                     Calculate2D, &read_background_arg.background,
22                     NULL, NULL);
23      }
24      else if (strcmp(connect->keyword, "calc3D") == 0) {
25          ReadCalibrationArg read_calibration_arg(CALIBRATION_FILE_NAME);
26          RPV_Invoke(connect, miss_FSM, 0,
27                     &ReadCalibration, &read_calibration_arg,
28                     &Calculate3D, &read_calibration_arg.calibration_data,
29                     NULL, NULL);
30      }
31      else if (strcmp(connect->keyword, "display") == 0) {
32          RPV_Invoke(connect, miss_FSM, 0, NULL, NULL, &Display, NULL, NULL,
 NULL);
33      }
34      return 0;
35  }
36
```

図 5.9: プログラム例 (メイン関数)

でが前処理関数であり，21 行目以下の Calculate2D が画像処理関数である．Calculate2D では，640 × 480 画素のカラー画像と，後段の PC からのフィードバックを入力として，画像平面上のマーカの 2 次元位置を計算し，その座標を出力している．具体的には，23，24 行目で受信バッファ中のデータ位置へのポインタ，26，27 行目で送信バッファ中の書き込み位置へのポインタ，28 行目で受信バッファ中のフィードバックデータ位置へのポインタをそれぞれ取得し，29，30 行目でユーザ引数を読み込んでいる．そして，背景差分を行った後，差分後の画像とフィードバックデータを入力として，マーカの 2 次元座標位置を計算している．

5.7 考察

RPV を以下の 3 点について評価する．

プログラミングの容易さ

一般のプログラマにとって，割り込み，プロセス間通信，排他制御などのシステムコールは難解なものである．しかし，これらのシステムコールを使いこなさなければ実時間並列画像処理を実現することはできない．RPV では，これらのシステムコールを利用する部分は完全に隠蔽されており，アプリケーションプログラマはそれらを意識することなく実時間並列画像処理システムを構築することが可能である．

さまざまな画像処理を実現できる記述力

(i) 各データ処理タスクは関数として記述され，入出力データはその関数の引数として与えられること，(ii) アプリケーションプログラマが自由に利用できる引数も用意されていること，(iii) 複数系列のデータや複数フレームのデータを利用することもできること，これらから，データを入力，処理，出力することをフレームごとに繰り返すデータ処理，および，そのようなデータ処理の組み合わせ処理を記述することが可能である．また，非同期データ転送をサポートしているため，分散観測ステーションどうしのインタラクションのように，イベント発生時のみ行われる通信も記述できる．さらに，非同期データのみを送受信するデータ処理タスクも記述可能であるため，例えば必要なときのみ参照される知識データベースのようにフレームごとの繰り返し処理ではないデータ処理タスクも実現できる．このように，RPV は非常に高い画像処理の記述力を持っていると言うことができる．

```
 1  #include <fstream.h>
 2  #include <rpv.h>
 3
 4  struct ReadBackgroundArg {
 5      char* filename;
 6      RPV_RGB24<IMAGE_WIDTH,IMAGE_HEIGHT> background;
 7      ReadBackgroundArg(const char* s);
 8  };
 9
10  void* ReadBackground(void* a)
11  {
12      ReadBackgroundArg* arg = (ReadBackgroundArg*)a;
13
14      ifstream fs(arg->filename);
15      arg->background.Read(fs);
16      fs.close();
17
18      return NULL;
19  }
20
21  void* Calculate2D(const RPV_Input* id, RPV_Output* od, const RPV_Ainput*
aid, void* a)
22  {
23      const RPV_RGB24<IMAGE_WIDTH,IMAGE_HEIGHT>* i_data
24  = (const RPV_RGB24<IMAGE_WIDTH,IMAGE_HEIGHT>*)id->data_ptr[0][0];
25      Positions2D<MARKER_NUM>* o_data
26        = (Positions2D<MARKER_NUM>*)od->data_ptr[0];
27      const Positions3D<MARKER_NUM>* a_data
28        = (const Positions3D<MARKER_NUM>*)ad->data_ptr[0][0];
29      const RPV_RGB24<IMAGE_WIDTH,IMAGE_HEIGHT>* background
30  = (RPV_RGB24<IMAGE_WIDTH,IMAGE_HEIGHT>*)a;
31
32      RPV_RGB24<IMAGE_WIDTH,IMAGE_HEIGHT>* sub_data
33        = RPV_Subtraction(i_data, background);
34
35      // Searching for markers from the subtracted image
36      // using previous 3D positions of markers
37      // for robustness against occlusion.
38      SearchMarker(sub_data, a_data, o_data);
39
40      return NULL;
41  }
42
```

図 5.10: プログラム例 (Calculate2D)

さまざまな並列処理を実現できる記述力

PC を単位とした並列化については，1.4 節で述べた並列手法すべてを記述することができる．

これらから，RPV は実時間並列画像処理プログラミングツールとしては高い能力を持っていることが分かる．

今後の課題としては，まず 4.7 節で述べたような複数の CPU を搭載した PC への対応が挙げられる．現在の RPV では，PC とデータ処理タスクが 1 対 1 で対応するしなければならないので，複数の CPU を搭載した PC でも 1 個の CPU しかデータ処理を行うことができない．また，二つ目の課題として，負荷分散の問題が挙げられる．現在の RPV では，各 PC へのデータ処理タスクの割り振りをアプリケーションプログラマが行うことで静的負荷分散を行っている．しかし，各 PC でのデータ処理の負荷や PC 間のデータ転送の負荷を正確に見積ることは難しく，アプリケーションプログラマの試行錯誤によって負荷分散が適切になるようにデータ処理タスクの割り振りを行うことになる．さらに，複数の CPU を搭載した PC に対応すると負荷分散がより複雑になり，アプリケーションプログラマの負担がさらに増してしまう．これらの課題を解決する方法としては，アプリケーションプログラマは PC ではなく，仮想的な分散並列要素 (Distributed Component:DC) 上でデータ処理タスクを実行するシステムを作成し，DC を PC に割り当てることによってデータ処理タスクを各 PC に配置する．複数の DC を 1 台の PC に割り当てることによって，複数の CPU を搭載した PC を有効に利用したり，負荷が小さいタスクをまとめたりすることができる．また，一つの DC を複数の PC に割り当てることによって，データ並列処理を行うことができる．このとき，負荷分散が最適になるように PC と DC の割り当てを自動的に行うツールを提供できれば，アプリケーションプログラマの負担が軽減される．このような方法が可能であるのは，データ処理タスクはデータ処理アルゴリズムのみを記述したものであり，データ通信とは独立に記述されているので，データ処理タスクを PC に配置するときにデータ通信のことを考慮しなくてもよいからである．

三つ目の課題として，データ並列処理方式の拡張が挙げられる．本研究で実現したデータ並列処理方式は一つのフレーム内で分割したデータを単位として並列化する空間方向のデータ並列処理方式である．この他に，フレームを単位として並列化する時間方向のデータ並列処理方式が考えられる．例えば図 5.11のように，1 フレーム目は PC1 が，2 フレーム目は PC2 が，3 フレーム目は PC3 が，4 フレーム目はまた PC1 が処理するという並列

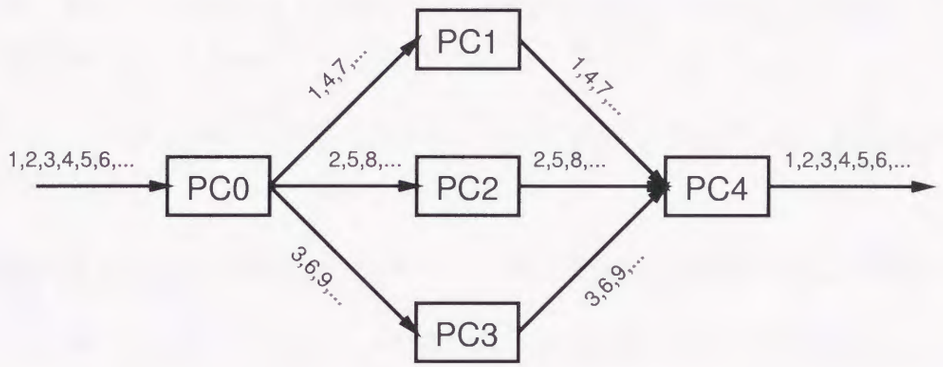


図 5.11: 時間方向のデータ並列処理

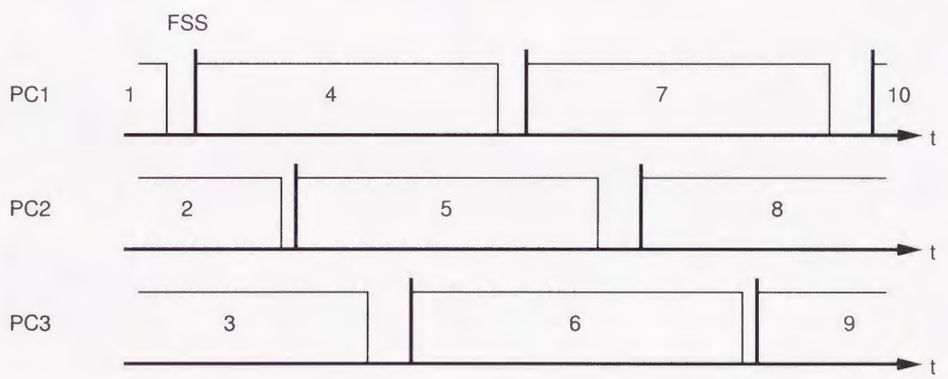


図 5.12: 時間方向のデータ並列処理時のデータ処理のタイムテーブル

化である．現在の RPV ではこのデータ並列処理方式を実現することはできない．なぜなら，すべてのデータ処理とデータ転送が FSS のタイミングによって 1 フレーム時間を基準として行われているので，このデータ並列処理方式のように複数フレーム時間 (この例では 3 フレーム時間) にまたがるようなデータ処理に対応できないからである．この課題は以下の機能拡張によって解決できる．

- 一つのストリームのデータをフレームごとに複数の PC に振り分ける機能を DSM に追加すること
- 複数の PC からのデータを一つのストリームにまとめる機能を DRM に追加すること
- FSS を複数の PC にフレームごとに振り分ける機能を FSM に追加すること
- 複数の PC からの FSS を一連の FSS にまとめる機能を FSM に追加すること
- このデータ並列処理方式を指定できるようにコネクションファイルの仕様を拡張すること

これにより，上の例では，PC1，PC2，PC3 には 3 フレームに 1 回だけ FSS が到着し，それに合わせて同期をとることで，図 5.12に示すタイムテーブルのように DPM 上のデータ処理が実行される．

第 6 章

結論

6.1 おわりに

本研究では，画像認識をさまざまな領域に適用するときの問題点として対象物知識の構築の問題と計算量の問題を挙げ，これらの問題点に対して以下の解決法を提案した．

画像認識に必要な知識を例題画像から獲得する方法 この方法は，画像を領域分割することによって得られる領域の木構造を基にして，知識を構成するシンボルの構造を表現するものであり，領域分割の特性に合った知識を構築することが可能である．また，複数の例題画像を与えることによって，知識が一般化されることも本手法の特徴である．

高レベル画像処理の並列化 高レベル画像処理では，トークンとシンボルの対応というボトムアップの処理と，シンボルの構造の整合性というトップダウンの処理を融合する必要がある．このため，マルチエージェントモデルによる並列処理を導入した．また，低レベル画像処理と違い，高レベル画像処理ではデータ参照の局所性が低く処理量の偏りが大きい．このため，共有メモリ型並列計算機を利用し，動的負荷分散を導入することによって，マルチエージェントモデルによる並列処理を効率的に実現した．

分散型並列計算機による実時間画像処理 実時間画像処理を分散並列環境で実行するためには，高速データ転送機構，同期機構，エラー処理機構が必要であり，これらの機能を実現した．また，分散型並列計算機として PC クラスタを利用し，この上でこれらの機能を実現した．

分散型並列計算機による実時間画像処理のためのプログラミングツール RPV RPV を使うことによって，実時間並列画像処理に必要なさまざまな機能が隠蔽され，PC 間のデー

タフローと各 PC におけるデータ処理タスクのみを記述することによりアプリケーションを構築できる.

6.2 今後の課題

今後の課題としては以下の点が挙げられる.

対象物知識の 3 次元化 本研究では, 対象物をある一つの向きから見た画像から対象物モデルを構築するので, それ以外の向きかた見た画像では対象物を探索できない. この問題を解決するためには, さまざまな向きから見た画像から対象物モデルを構築する必要がある. そこで, 主な方向から見た画像から対象物モデルを構築し, それらの対象物モデルを統合することが必要である. このとき, 異なる向きから見た対象物モデルの分割木のノードどうしの対応をとる方法, 見る向きによる領域特徴の変化の表現法などについて検討しなければならない.

対象物探索の動画像への応用 静止画像ではまず領域分割を行って領域を抽出したあとで, ノードマッチングによって対象物探索を行っている. しかし, 動画像では前後のフレームの相関が高いので, 一度対象物領域を探索できれば, その後のフレームではその領域を追跡できればよく, 画像を領域分割する必要はない. 動画像に対するこのような制約に基づいた効率的な対象物追跡を考えていく必要がある.

分散型並列計算機での負荷分散の方法 アプリケーションプログラマの負担をさらに減らすため, 以下の機能を実現し, 負荷分散ツールを作成する.

- データ処理タスクのプロセッサへの配置に応じたデータフローの自動作成
- 複数のデータ処理タスクをプロセッサに配置したときの, それらのタスク間のデータ通信
- データ処理およびデータ通信の負荷に基づいた, データ処理タスクのプロセッサへの配置変更

広域分散型並列画像処理環境への適応 本研究で提案した PC クラスタでは均一な性能を持つ高速ネットワークの利用を前提としていた. しかし, 広域分散型並列計算機環境ではネットワークの性能は均一ではない. また, 故障などで一部の通信ができなくなる

可能性もある．このような環境における実時間並列画像処理についての議論は十分ではないので，今後検討する必要がある．

謝辞

本研究の機会と研究に対する御指導を頂いた雨宮真人教授に深く感謝致します．本研究を進めていく上で様々な助言を頂いた谷口倫一郎教授に深く感謝致します．本論文をまとめるにあたって，貴重な御意見を頂いた長谷川隆三教授に深く感謝致します．日頃より討議に参加して頂いた菅沼明助教授，福岡大学の鶴田直之講師，米元聡君，峯恒憲助教授，日下部茂助教授，富安洋史助手に感謝致します．筆者と共に本研究を進めて頂いた山田善之君，白方貴史君，濱田義雄君，松本明日香君，松尾泰治君，吉本廣雅君にお礼を申し上げます．

また，筆者を画像認識分野の研究者として導いて頂いた京都大学の池田克夫教授，美濃導彦教授，松山隆司教授に感謝致します．

最後に，筆者の長い間の学生生活を支えて頂いた両親と妹に感謝の言葉を送ります．

参考文献

[1] 大上, 杉本, 北村, 角, 富田. ネットワーク型並列計算機環境における物体認識. 信学会論文誌 (D-II), Vol. J82-D-II, No. 12, pp. 2307–2351, 1999.

[2] 松山, 和田. Hough 変換 : 投票と多数決原理に基づく幾何学的対象の検出 と識別. 松山, 久野, 井宮（編）, コンピュータビジョン 技術評論と将来展望, 第 10 章, pp. 149–165. 新技術コミュニケーションズ, 1998.

[3] S. Tokai, T. Wada, and T. Matsuyama. Real time 3D shape reconstruction using PC cluster system. In *Proc. of 3rd International Workshop on Cooperative Distributed Vision*, pp. 171–187, 1999.

[4] L. Davis, E. Borovikov, R. Cutler, D. Harwood, and T. Horprasert. Multi-perspective analysis of human action. In *Proc. of 3rd International Workshop on Cooperative Distributed Vision*, pp. 189–223, 1999.

[5] D. Arita, N. Tsuruta, R. Taniguchi, and M. Amamiya. Model generation method for object recognition task by pictorial examples. In *Proc. of ICIP'94 International Conference on Image Processing*, Vol. I, pp. 233–237, 1994.

[6] D. Arita, N. Tsuruta, R. Taniguchi, and M. Amamiya. An object recognition system based on a model generated from image examples. In *Proc. of Second Asian Conference on Computer Vision*, Vol. 3, pp. 161–165, 1995.

[7] 有田, 白方, 鶴田, 谷口, 雨宮. 知識獲得機能をもった画像認識システム －画像領域の輪郭情報の利用－. 九州大学大学院システム情報科学研究科報告, Vol. 2, No. 1, pp. 87–92, 1997.

[8] 有田, 鶴田, 谷口, 雨宮. 画像認識における階層的対象物モデルの獲得. 映像情報メディア学会誌, Vol. 51, No. 8, pp. 1240–1248, 1997.

[9] J. H. Connell and M. Brady. Generating and generalizing models of visual objects. *Artificial Intelligence*, Vol. 31, pp. 159–183, 1987.

[10] 秋山, 荒井. 対象の幾何学的構造に関する知識を学習により記述するモデル駆動型画像理解システム. 信学技報 PRU93-29, pp. 33–40, 1993.

[11] W. Harvey, M. Diamond, and D. Mckeown Jr. Tools for acquiring spatial and functional knwoledge in aerial image analysis. In *Proc. of the DARPA Image Understanding Workshop*, pp. 857–873, 1992.

[12] 村瀬, S. Nayar. 2 次元照合による 3 次元物体認識ーパラメトリック固有空間法ー. 信学会論文誌 (D-II), Vol. J77-D-II, No. 11, pp. 2179–2187, 1994.

[13] 高木, 下田（編）. 画像解析ハンドブック. 東京大学出版会, Tokyo, 1991.

[14] Glenn Shafer. *A mathematical theory of evidence.* Prinston University Press, Princeton, 1976.

[15] 富永昌治. コンピュータビジョンにおけるカラー情報の表現と利用. 松山, 久野, 井宮（編）, コンピュータビジョン 技術評論と将来展望, 第 5 章, pp. 64–79. 新技術コミュニケーションズ, 1998.

[16] 富永昌治. カラー画像の色分類と分割. 情処論文誌, Vol. 31, No. 11, pp. 1589–1598, 1998.

[17] 岩瀬, 山村, 田中, 大西. 映り込み分離カメラシステム. 信学会論文誌 (D-II), Vol. J81-D-II, No. 6, pp. 1224–1232, 1998.

[18] 有田, 鶴田, 谷口, 雨宮. 領域の階層構造を利用した画像認識システムにおけるモデルマッチング. 情報処理学会第 4 9 回全国大会, 第 2 巻, pp. 153–154, 1994.

[19] L. Davis, R. Chellappa, Y. Yacoob, and Q. Zheng. Visual surveillance and monitoring of human and vehicular activity. In *Proc. of Image Understanding Workshop*, pp. 19–23, 1997.

[20] T. Kanade, R. T. Collins, A. J. Lipton, P. Burt, and L. Wixson. Cooperative multi-sensor video surveillance. In *Image Understanding Workshop*, pp. 3–10, 1997.

[21] 宮崎, 亀田, 美濃. 複数のカメラを用いた複数ユーザに対する講義の実時間映 像化法. 信学会論文誌 (D-II), Vol. J82-D-II, No. 10, pp. 1598–1605, 1999.

[22] M. Minoh and Y. Kameda. Distance learning environment based on the interpretation of dynamic situation of lecture room. In *Proc. of 3rd International Workshop on Cooperative Distributed Vision*, pp. 283–301, 1999.

[23] Dean Pomerleau. Visibility estimation from a moving vehicle using the ralph vision system. In *Proc. of Image Understanding Workshop*, pp. 339–344, 1997.

[24] S. Yonemoto, A. Matsumoto, D. Arita, and R. Taniguchi. A real-time motion capture system with multiple camera fusion. In *Proc. of 10th International Conference on Image Analysis and Processing*, pp. 600–605, 1999.

[25] 米元, 有田, 谷口. 多視点動画像処理による実時間全身モーションキャプチャ システム— 視覚に基づく仮想世界とのインタラクション—. 映像メディア学会誌, Vol. 54, No. 3, 2000(発表予定).

[26] Takashi Matsuyama. Cooperative distributed vision. In *Proc. of 1st International Workshop on Cooperative Distributed Vision*, pp. 1–28, 1997.

[27] 松山, 浅田, 美濃, 和田. 分散協調視覚プロジェクトー分散協調視覚研究, システム開発の概要ー. 情処研報 CVIM103-4, pp. 25–34, 1997.

[28] Takashi Matsuyama. Cooperative distributed vision — integration of visual perception, action, and communication —. In *Proc. of Image Understanding Workshop*, pp. 1–39, 1998.

[29] 松山隆司. 分散協調視覚ー視覚・行動・コミュニケーション機能の統合による知能の創発ー. 画像の認識・理解シンポジウム, 第 I 巻, pp. 343–352, 1998.

[30] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine — A Users' Guide and Tutorial for Networked Parallel Computing.* The MIT Press, 1994.

[31] Message Passing Interface Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, Vol. 8, No. 3, 1994.

[32] D. Arita, N. Tsuruta, and R. Taniguchi. Real-time parallel video image processing on PC-cluster. In *Proc. of SPIE'98 Parallel and Distributed Methods for Image Processing II*, Vol. 3452, pp. 23–32, 1998.

[33] D. Arita, Y. Hamada, and R. Taniguchi. A real-time distributed video image processing system on PC-cluster. In *Proc. of 4th International Conference of the Austrian Center for Parallel Computation(ACPC)*, pp. 296–305, 1999.

[34] D. Arita, R. Taniguchi, S. Yonemoto, and Y. Hamada. A real-time multi-view image processing system on PC cluster. In *Proc. of the 4th Asian Conference on Computer Vision*, pp. 270–275, 2000.

[35] T. Kanade, H. Saito, and S. Vedula. The 3D room: Digitizing time-varying 3D events by synchronized multiple video streams. Technical Report CMU-RI-TR-98-34, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 1998.

[36] 亀田, 太尾田, 角所, 美濃. 時空間の分割とビデオ画像のパイプライン処理による高速三次元再構成. 情処論文誌, Vol. 40, No. 1, pp. 13–22, 1999.

[37] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato. Pm: An operating system coordinated high performance communication library. In P. Sloot and B. Hertzberger, editors, *High-Performance Computing and Networking*, pp. 708–717. 1225 of Lecture Notes in Computer Science,Springer-Verlag, 1997.

[38] David L. Mills. Internet time synchronization: the network time protocol. *IEEE Trans. Communications*, Vol. COM-39, No. 10, pp. 1482–1493, 1991.

[39] 奥富正敏. ステレオ視. 松山, 久野, 井宮（編）, コンピュータビジョン 技術評論と将来展望, 第 8 章, pp. 123–137. 新技術コミュニケーションズ, 1998.

[40] D. Arita, Y. Hamada, and R. Taniguchi. Programming tool for real-time image processing on a distributed system. In *Proc. of SPIE'99 Parallel and Distributed Methods for Image Processing III*, pp. 28–39, 1999.

[41] 濱田, 有田, 谷口. 実時間並列動画像処理ツール RPV. 情処研報コンピュータビジョンとイメージメディア (99-CVIM-119), pp. 9–16, 1999.

[42] D. Arita, Y. Hamada, S. Yonemoto, and R. Taniguchi. RPV: A programming environment for real-time parallel vision — specification and programming methodology—. In *Proc. of Workshop on Parallel and Distributed Computing in Image processing, Video Processing and Multimedia*, 2000(To be appeared).

[43] R. Taniguchi, Y. Makiyama, N. Tsuruta, S. Yonemoto, and D. Arita. Software platform for parallel image processing and computer vision. In *Proc. of SPIE'97 Parallel and Distributed Methods for Image Processing*, Vol. 3166, pp. 1–10, 1997.

[44] H. Matsuo, K. Nakada, and A. Iwata. A distributed image processing environment vios iii and it's performance evaluation. In *Proc. of 14th International Conference on Pattern Recognition*, Vol. II, pp. 1538–1542, 1998.

[45] M. Lückenhaus and W. Eckstein. A thread concept for automatic task parallelization in image analysis. In *Proc. of SPIE'98 Parallel and Distributed Methods for Image Processing II*, Vol. 3452, pp. 34–44, 1998.

[46] K. N. Kim, T. S. Choi, and R. S. Ramakrishna. A new parallel vision environment in heterogeneous networked computing. In *Proc. of SPIE'98 Parallel and Distributed Methods for Image Processing II*, Vol. 3452, pp. 45–56, 1998.

[47] V. Rodin, F. Harrouet, P. Ballet, and J. Tisseau. oris: Multiagents approach for image processing. In *Proc. of SPIE'98 Parallel and Distributed Methods for Image Processing II*, Vol. 3452, pp. 57–68, 1998.

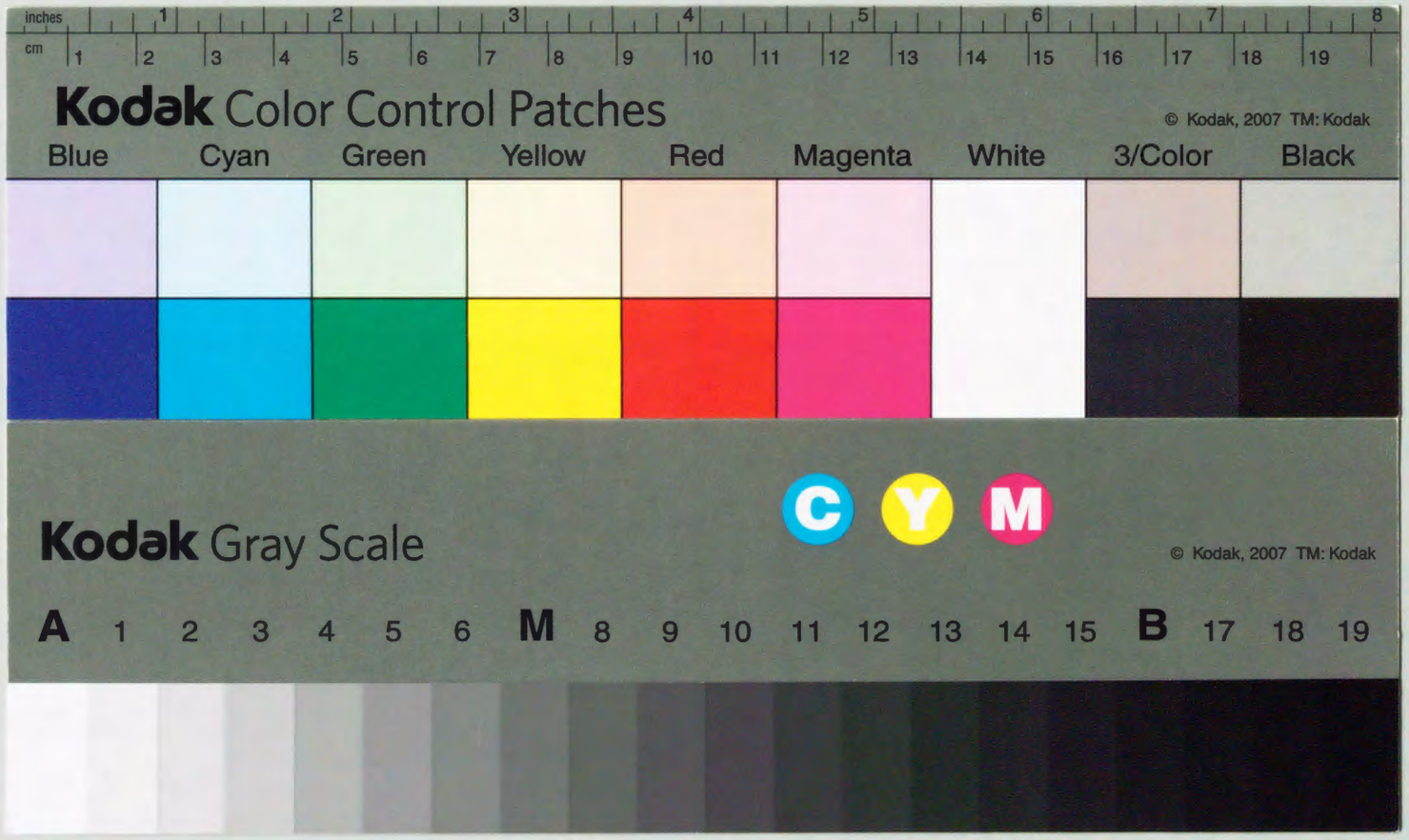
[48] J. M. Squyres, A. Lumsdaine, and R. L. Stevenson. A toolkit for parallel image processing. In *Proc. of SPIE'98 Parallel and Distributed Methods for Image Processing II*, Vol. 3452, pp. 69–80, 1998.

[49] B. Gennart and R. D. Hersch. Computer-aided synthesis of parallel image processing applications. In *Proc. of SPIE'99 Parallel and Distributed Methods for Image Processing III*, Vol. 3817, pp. 48–61, 1999.

[50] A. Kanevsky, A. Skjellum, and A. Rounbehler. MPI/RT —an emerging standard for high-performance real-time systems. In *Proc. of 31st Hawaii International Conference on System Sciences*, Vol. 3, pp. 157–166, 1998.

[51] L. Böszörményi, G. Hölzl, and E. Pirker. Parallel cluster computing with IEEE1394-1995. In *Proc. of 4th International Conference of the Austrian Center for Parallel Computation(ACPC)*, pp. 522–532, 1999.

索 引


inches
cm
Kodak Color Control Patches
© Kodak, 2007 TM: Kodak
Blue
Cyan
Green
Yellow
Red
Magenta
White
3/Color
Black
Kodak Gray Scale
C
Y
M
© Kodak, 2007 TM: Kodak
A 1 2 3 4 5 6 M 8 9 10 11 12 13 14 15 B 17 18 19