分散並列計算機による実時間モーションキャプチャ

有田，大作
九州大学大学院システム情報科学研究院知能システム学部門

米元，聡
九州大学大学院システム情報科学研究院知能システム学部門

谷口，倫一郎
九州大学大学院システム情報科学研究院知能システム学部門

https://doi.org/10.15017/4777974

出版情報：九州大学情報基盤センター年報. 1, pp.9-23, 2001-10. 九州大学情報基盤センター
バージョン：
権利関係：

情報基盤センター年報第1号（2001年）
九州大学情報基盤センター

分散並列計算機による実時間モーションキャプチャ
Real-time Motion Capture on a Distributed Parallel Computer

有田 大作　　米元 聡　　谷口 倫一郎
Daisaku Arita　Satoshi Yonemoto　Rin-ichiro Taniguchi

九州大学大学院システム情報科学研究院知能システム学部門
Department of Intelligent Systems, Kyushu University

要旨　近年，複数のカメラによって撮影した画像から対象物や環境についての情報を獲得しようという研究が広く行われるようになってきている．本研究では，複数のPCを高速ネットワークで結合したPCクラスタを利用してこのようなシステムを構築するためのプログラミング環境RPV(Real-time Parallel Vision)を提案する．さらに，RPVを利用した実時間マーカレスモーションキャプチャシステムを紹介し，RPV上で実用的なアプリケーションを開発できることを示す．

Abstract　Recently, image analysis using multiple cameras has been extensively researched. In this paper, we propose RPV(Real-time Parallel Vision), a programming environment for developing image analysis system with multiple cameras on PC cluster which consists of multiple PCs and high speed network. And we propose a real-time marker-less motion capture system to show that we can develop practical applications using RPV.

1　はじめに

近年，コンピュータビジョンの分野では，複雑な対象や環境を理解するために，複数のカメラから得られた多視点情報を利用するという研究が盛んに行われている．複数のカメラを用いる場合の問題点，特に，実時間の画像処理を行おうとする場合の問題点として，1台の計算機に接続できるカメラ台数の制限が挙げられる．これは，計算機のI/O能力の限界や接続可能な装置数の制限といった物理的な制約によるものであり，多数のカメラを利用しようとする場合大きな問題となってくる．また，視点数の増加に伴って必要とされる計算能力も大きくなってくる．これに対して，計算機とカメラを対にした観測ステーションをネットワークで接続し，分散計算システムとしてコンピュータビジョンシステムを構築しようという研究が進められている[1]．このような分散計算システムでは，カメラの台数に特に制約はなく，必要に応じて観測ステーションを増加させるだけで視点数を増やすことができ，観測ステーションの台数に見合っただけの計算能力が得られるという大きな利点がある．このような背景から，我々は，汎用的なPCをネットワーク接続したPCクラスタ上での実時間画像処理に関する研究を行っている．汎用のPCを利用することにより，コストパフォーマンスが高く，柔軟性に富んだシステムを構築することが可能である．

従来より，数値計算やデータベースの世界では，ネットワーク接続された汎用のワークステーションやPCを分散型並列計算機として利用することが行われてきており，並列プログラムの記述性や移植性を高めるために，標準的なプログラミング環境も提案されている[2, 3]．また，画像処理に関しても静止画を対象とした分散処理システムもいくつか開発されている[4, 5, 6]．しかし，実時間画像処理では，フレーム単位で次々と到着する画像データを滞りなく処理し，結果を出力することが要求されるため，その分散並列計算機上での実現は容易ではない．すなわち，実時間画像処理を分散並列計算機上で実現するためには，各通信経路における1フレーム分のデータ転送，および，各ノードPCにおける1フレーム分のデータ処理が1フレーム時間（通常よく使われるカメラでは1/30秒）で終了することを保証しなければならない．このためには

- 高速かつ低負荷なデータ転送機構
- 1フレーム時間内にデータ転送とデータ処理が終了していることを保証する同期機構
- データ転送やデータ処理に遅れが生じたときに正常な状態に復帰するためのエラー処理機構

が必要である．また，応用システムを開発，維持する上では，これらの実時間処理機構の複雑なプログラミングも問題となってくる．本論文では，これらの機能を提供することにより実時間並列画像処理をPCクラスタ上で容易に開発できるプログラミング環境RPV（Real-time Parallel Vision）を提案し，その概要，実時間処理機能の実現方法について述べる．さらに，RPVを使った実アプリケーションとして構築した実時間マーカレスモーションキャプチャシステムについて概説する．

2 分散並列計算機上の実時間多視点画像処理環境 RPV

2.1 RPV の概要

2.1.1 システム構成

本研究で用いるPCクラスタシステム（**図 1**）は，14台のノードPCからなっている．その内，6台にCCDカメラが接続されており，すべてのCCDカメラは同期信号発生装置により同期がとられている．各ノードPCはCPUを2個搭載しているAT互換機であり，OSにはLinuxを用いている．また，各ノードPCはギガビットLANの一種であるMyrinetによって相互に結合されおり，Myrinet上の通信にはRWCPによって開発されたPM通信ライブラリ[7]を利用している．Myrinet上のPM通信ライブラリの特徴として，

- 7.5 μs の低レイテンシ
- 118MByte/秒の高スループット
- 全二重通信をサポート
- ゼロコピー通信をサポート
- 割込み受信をサポート

といった点が挙げられる．このデータからも分かるように，非圧縮フルサイズカラー画像（640×480画素/フレーム，1画素4バイト）を30fpsで転送するための条件，すなわち毎秒 $640 \times 480 \times 4 \times 30 = 36864000 \simeq 35$M バイト以上のスループット，を十分満たしている．また，ゼロコピー通信と割込み受信のサポートにより，受信側のノード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．

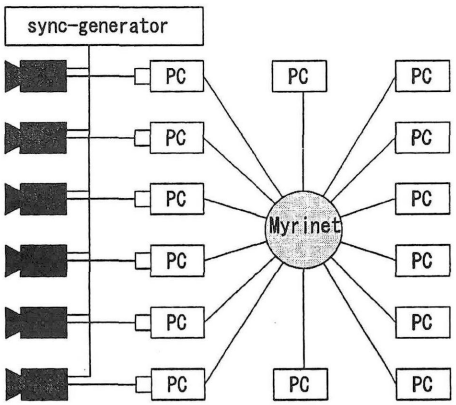


図 1: PC クラスタシステム

2.1.2 並列処理方式

RPVでは，以下の並列処理方式を実現している．

- パイプライン並列処理
  図 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，データ処理能力は向上するが，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は増大してしまう．

- データ並列処理
  図 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以下の二つの方式が考えられる．

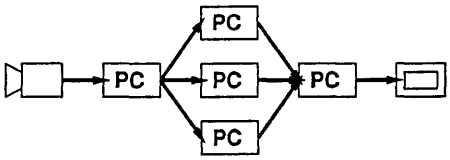
  - 空間方向データ並列処理
    各フレームのデータを分割して各計算機で処理を行う方式．画像を分割して並列に処理を行うなど，画像処理ではよく利用される並列処理である．また，フレームデータを分割するのではなく，複数のカメラによって同時に獲得されたデータに対して複数の計算機で同時に処理する方式も空間方向

データ並列処理の一つである（図 2 (c) 参照）.

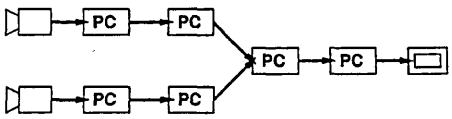
- 時間方向データ並列処理
  フレームごとに異なる計算機で処理を行う方式. 視体積交差法のような分割後の処理の負荷に偏りが生ずるデータ処理タスクや, 画像領域分割のようにフレームデータの分割が難しいデータ処理タスクを並列化するときに利用できる.

- 機能並列処理
  図 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, 空間方向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くなってしまう.

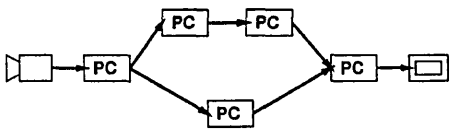
(a) Pipeline parallel processing



(b) Data parallel processing



(c) Parallel processing of multiple cameras' data



(d) Function parallel processing

図 2: 並列処理方式

2.1.3 プログラミングの概要

実時間並列画像処理アプリケーションを構築するとき, アプリケーションによって異なるのは, ノードPC間のデータフローと各ノードPCにおけるデータ処理タスクのみである. これら以外の機能, 具体的には

- データ転送機構
- 同期機構
- エラー処理機構

についてはすべてのアプリケーションで共通に利用できる. これらの機能を実現するためには, ハードウェア, OS, ネットワークについての多くの知識が必要であり, プログラミングは容易ではない. そこで, RPVではこれらの機能を C++のクラスライブラリとして提供することで, アプリケーションを容易に記述できるようにしている. これにより, アプリケーションプログラマはデータフロー情報とデータ処理タスクを記述するだけでよいことになる.

ノードPC間のデータフロー情報の記述 図 3で示したクラス `RPV_Connection` は, ノードPC間のデータフロー情報を持つクラスである. 各ノードPCは, このクラスの持つ情報をもとに, ノードPC間のデータの送受信を行う. 通常このクラスは, ノードPC間のデータフロー情報を記述したファイル（以下コネクションファイル）によって初期化される. このコネクションファイルにはPCクラスタ全体のデータフローを記述する. このようにデータフロー情報は 一つのファイルにまとめて記述されるため, データフロー情報の管理が容易であるだけでなく, プログラムの修正, 再コンパイルを行うことなく, 利用するノードPCと各ノード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は, 1行に1台のノードPCのデータフロー情報を記述し, 各行には以下の項目が記述される.

PCno PCクラスタ内のノードPCに付けられた番号

```
class RPV_Connection{
   int myPC_no;              自ノードPC番号
   char* keyword;            処理内容のキーワード
   int input_PC_num;         受信する同期データの数
   int* input_PC;            受信同期データの転送元ノードPC番号
   int* input_data_size;     受信する同期データのサイズ
   int input_frame_num;      同時に使用する受信同期データのフレーム数
   int input_buf_num;        受信同期データ用バッファの数
   int output_PC_num;        送信データの数
   int* output_PC;           送信データの転送先ノードPC番号
   int* output_data_size;    送信データのサイズ
   int output_buf_num;       送信バッファの数
   int ainput_PC_num;        受信非同期データの数
   int* ainput_PC;           受信非同期データの転送元ノードPC番号
   int* ainput_data_size;    受信非同期データのサイズ
   int ainput_data_num;      同時に使用する受信非同期データのフレーム数
   int connect_PC_num;       使用するノードPCの数
   int* connect_PC;          使用するノードPC番号列
};
```

図 3: クラス RPV_Connection

keyword ノードPCで行うデータ処理を指定する文字列

i_PC 受信同期データの転送元ノードPC番号

i_size 受信同期データのサイズ

i_num 同時に使用する受信同期データの数

o_PC 送信データの転送先ノードPC番号

o_size 送信データサイズ

ai_PC 受信非同期データの転送元ノードPC番号

ai_size 受信非同期データのサイズ

ai_num 同時に使用する受信非同期データの数

各ノードPCでのデータ処理アルゴリズムの記述 RPVにおけるデータ処理は図4に示す流れで行われる．まず，初期化1でコネクションファイルの読み込みが行われ，クラス RPV_Connection が初期化される．つぎに，変数 keyword によって分岐を行い，ノードPCごとに引数の異なる関数 RPV_Invoke を起動する．このようにするのは，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ノードPCを変更できるようにするためである．図5に示すのが関数 RPV_Invoke の仕様であり，関数 RPV_Invoke の引数によってそのノードPCで実行されるデータ処理タスクが指示される．また，第2引数の sync_mode で，同期機構に関する選択を行う（2.2.4参照）．

関数 RPV_Invoke の動作は以下の通りである．

1. 初期設定として，モジュールの起動（2.2.1参照），通信路の初期化を行う．
2. 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．
   (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. すべてのノード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，三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．受信同期データ引数と受信非同期データ引数には，システムによって自

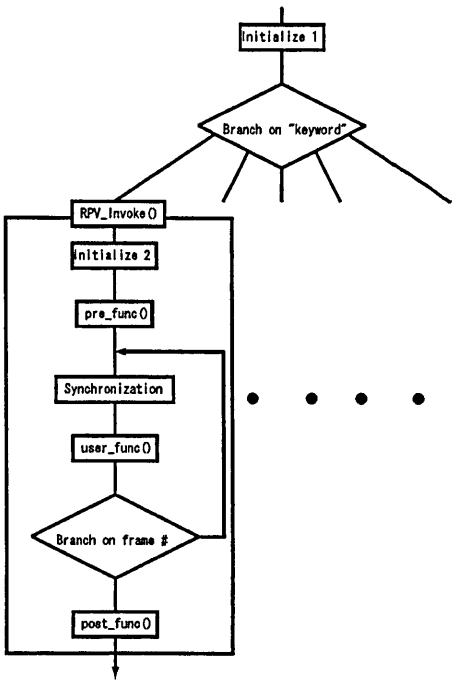


図 4: 処理の流れ

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，送信データ引数には，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，1 枚の画像を入力とし，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，実時間動画像処理ということをほとんど意識することなく，容易にユーザ関数をプログラミングできる．

2.2 RPV の実装

本節では，RPV におけるデータ転送機構，同期機構，エラー処理機構の実現方法について述べる．

2.2.1 実装の基本方式

並列動画像処理を効率的に実行するために，本システムの各ノード PC では，図 6に示す四つのモジュールが，UNIX プロセスとして並行動作している．これにより，データの受信，処理，送信を並行に行うこと

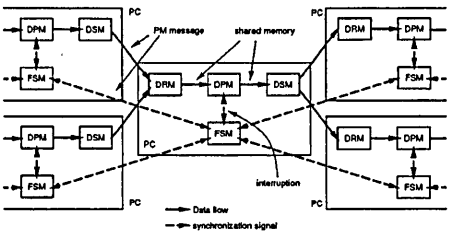


図 6: 各ノード PC のモジュール構成

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のバッファは，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ている．以下，各モジュールの動作について説明する．

- データ受信モジュール（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．
  - 転送元ノード 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，データが書き込まれた受信バッファの領域を書き込み不可（読み込み可能）にする．
  - 処理が済んでデータ書き込み可能になった受信バッファの領域をチェックし，転送元ノード PC に伝える．
- データ処理モジュール（DPM）
  実際の画像処理を行うモジュール．
  - 最初に関数 `pre_func` を，最後に関数 `post_func` を呼び出す．
  - 処理開始の割り込み信号を受けとると，受信バッファの読み込み可能領域からデータを読み込んで画像処理を行う（実際は関数 `user_func` を呼び出す）．処理結果は送信バッファの書き込み可能領域に書き込み，その領域を書き込み不可とする．
  - 処理が終了すると，当該の受信バッファ領域を書き込み可能にする．
  - 処理結果の書き込みが終了すると，書き込みの終了を通知する割り込み信号を DSM に送る．

```
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,                              処理するフレーム数
   void* (*pre_func)(void*),                   ループ前に実行される前処理関数
   void* pre_func_arg,                         pre_func の引数
   void* (*user_func)(RPV_Input*, RPV_Output*,
                RPV_Ainput*, void*),           ループ中に実行される関数
   void* user_func_arg,                        user_func の引数
   void* (*post_func)(void*),                  ループを出た後に実行される後処理関数
   void* post_func_arg                         post_func の引数
);
```

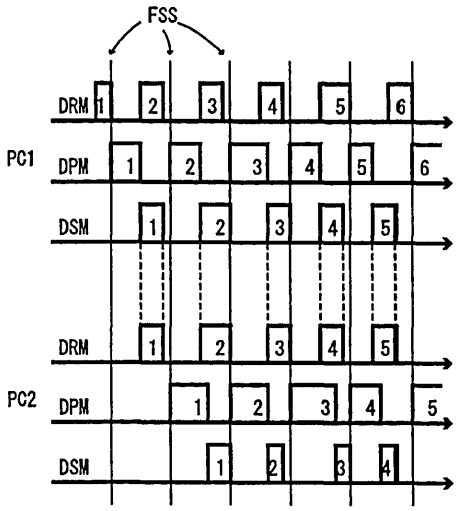
図 5: 関数 RPV_Invoke

- データ送信モジュール（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ノードPCに送信する．
  - 送信が終了したデータの格納されていた領域を書き込み可能とする．
- フレーム同期モジュール（FSM）
  ノード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，**2.2.3**で述べるフレーム同期信号（Frame Synchronization Signal: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．

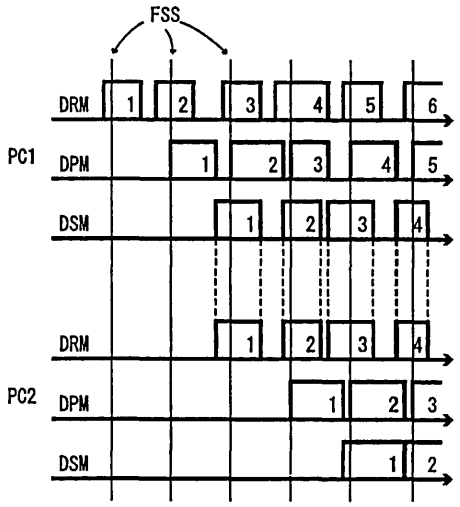
2.2.2 ノードPCにおける処理の並列化

システム全体としての性能を向上させるためには，各ノードPCでの性能を向上させる必要がある．そこで，ここでは，ノードPC内のモジュールの並列化について考察する．**2.2.1**で述べたモジュールのうちCPUを多く利用するものは，DPM以外にDSMがある．これはPM通信ライブラリではデータ転送の単位が最大8Kバイトのため，画像のような大きなデータは分割して送信する必要があり，その制御にCPUが必要なためである．その他のモジュールはほとんどCPUを利用しない．また，Myrinetは全二重通信可能であるため，データ受信とデータ送信が競合することもない．したがって，データの転送，処理に直接関与するモジュールであるDRM，DPM，DSMでの処理は，一般に**図7 (a)**のようになる．この図で，上半分と下半分がそれぞれ一つのノードPCの動作を表しており，それぞれのノード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，上段のノードPCから下段のノードPCへと同期データが転送されている．この図のように，DPMによるデータ処理の終了後，DSMによるデータ送信が行われるという排他的な処理になる．

このとき画像の実時間処理を実現するためには，DPMによるデータ処理時間とDSMによるデータ送信時間の合計が1フレーム時間以内となることが必要となる．しかし，1台のノードPCに2個のCPUを搭載すれば，DPMとDSMを並列動作させることが可能となり，**図7 (b)**のように，t番目のフレームのデータ処理と$t-1$番目のフレームのデータ送信をオーバラップさせることが可能となる．つまり，1フレーム時間内で画像データの処理にCPUを多く割り当てられるようになる．このような点を考慮し，本研究のPCクラスタでは各ノードPCに2個のCPUを搭載している．ただし，データ処理とデータ送信をオーバラップさせない場合に比べてオーバラップさせる場合は，1フレーム時間あたりの処理可能量が増加する（スループットが増大する）かわりにレイテンシが2倍になってしまう．したがって，どちらの方式を採用するかをアプリケーションの計算量やノードPCの台数などを考慮してアプリケーションプログラマが選択できるようになっている．



(a) Without overlaping DPM and DSM



(b) With overlaping DPM and DSM

図 7: モジュールのタイムチャート

2.2.3 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ノード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[8]を導入し，ノード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でノード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おり，これを比較することにより，異なるカメラからの画像の獲得タイミングが同じであるかどうかを判断できる．
- カメラが接続されたノード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，FSSを発生する．FSSは同期データの流れに沿って上流のノードPCから下流のノードPCへと転送される．このFSSによって各ノード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**2.2.4**を参照）．

2.2.4 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，ノード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．

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，前段のノードPCからの同期データ転送の遅れや前段までのノード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8**は連続した二つのノードPCのタイムチャートである．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

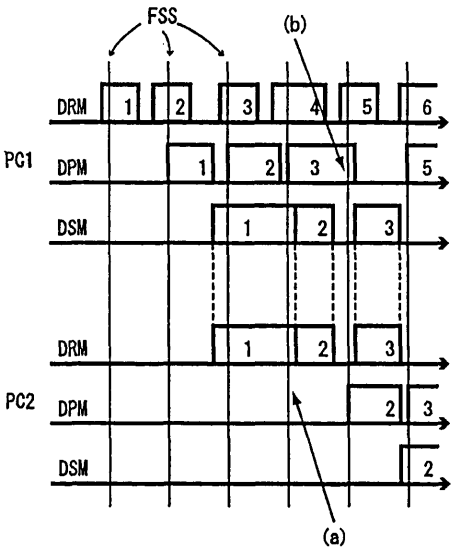


図 8: データ転送同期とデータ処理同期

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．

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 8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．

なお，本システムでは，ここで述べた同期機構に基づいた同期データ転送以外に，非同期データ転送もサポートしている．非同期データ転送は，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，データは 1 フレームに何回も転送されても，あるいはまったく転送されなくてもよい．通常は，フィードバックや協調処理のための制御信号などに利用される．アプリケーションプログラマは関数 user_func 中の任意のタイミングで最新の非同期データを転送することができる．

2.2.5 エラー処理機構

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．

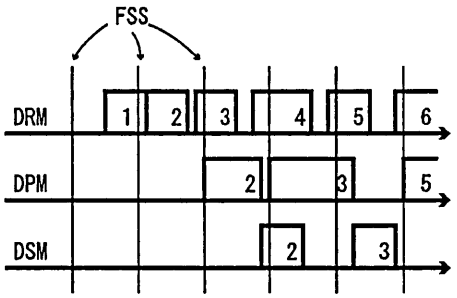
- データ落ち型 (Data missing)
  (a) は (2)，(b) は (2) の組み合わせ (図 9 (a) 参照)．完全なデータだけを出力するが，エラーのときはデータ落ちが起こる．この方式は，処理が遅れたときに決められた時間以内に出力が得られないことになるので，厳密には実時間システムとは言えなくなる．しかし，データを落とすことによってすぐに遅れの状態から正常な状態に復帰できるので，実際の動画像処理としては最も利用される方式である．

- 不完全データ転送型 (Incomplete data)
  (a) は (1)，(b) は (1) の組み合わせ (図 9 (b) 参照)．データ落ちは起こらないが，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

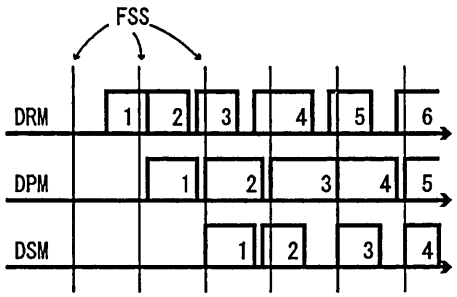
分散を慎重に行う必要がある．なお，このとき出力されるデータは，以下の中からアプリケーションプログラマが選択する．

1. 処理途中のデータ
2. 前フレームの処理結果
3. あらかじめ指定されたデータ

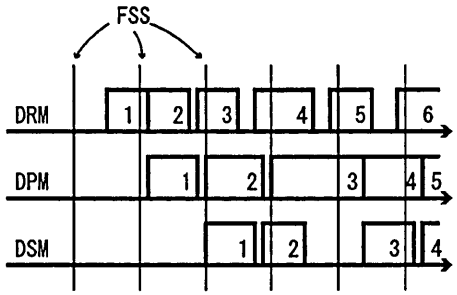
- 完全保持型（Complete queuing）
(a) は (1)，(b) は (2) の組み合わせ（図9 (c) 参照）．データ落ちも起こらず，完全なデータだけを出力する．しかし，この方式は，一度エラーが起こるとそれ以降の実時間性の保証ができず，さらに最悪の場合は受信バッファが溢れてしまう．



(a) Data missing



(b) Incomplete data



(c) Complete queuing

図 9: 同期におけるエラー処理方式

3 多視点画像処理を利用した実時間マーカレスモーションキャプチャシステム

本節では，RPV を利用して構築した実時間マーカレスモーションキャプチャシステム [9, 10] について説明する．

3.1 目的

近年は *SmartRoom*[11] など，接触型の器具を用いることなく，ビデオカメラを用いて人間の行為を知覚するような知的なサイバースペース生成の研究が進んでいる．これは非接触な方式を実現することが，コンピュータビジョンの達成すべき研究課題であるばかりでなく，仮想空間への入口となる計測空間と日常の生活空間とをシームレスに繋げるために有効と考えられているからであり，本研究もこのようなセンサを装着することによる制約を打破する非接触型の知覚系の構築を目指すものである．

3.2 人体動作推定の問題

人体動作の推定問題は，なるべく制約のない，リアリティの高い推定を実時間で行う方式の開発が課題であると言える．一般に，人体動作の推定問題は，図 10 で示すような階層からなっている．第一段階は，画像から人間を抽出するのに必要な画像特徴を抽出するレベルである．第二段階はそれらの画像特徴を統合し，人体の３次元的な姿勢を抽出するレベルであり，基本的には物理的な情報を取り扱う世界であるり，現在までの研究のほとんどは，この段階までにとどまっている．第三段階は，第二段階で得られた結果を基に，さらに詳細な情報や時系列情報を獲得・統合して，行動や意図，感情などの抽象化された概念を抽出するレベルである．第三段階を実現するためにも，第二段階でリアリティの高い人体動作を推定・再現する必要があ

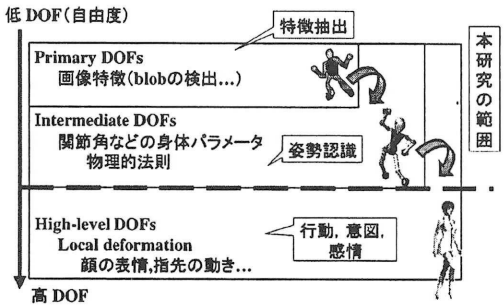


図 10: 人体動作のモデル

るので，本研究では，以下の課題を実現するシステムを構築する．

- 少ない画像特徴からリアリティの高い人体動作を再現する．
- 実時間で動作可能であり，ユーザ介在の後処理を必要としないなどオンライン性がある．
- 計測範囲を拡張し，様々な姿勢を認識できるよう，多視点情報を利用する．

また，本システムでは前述の *Pfinder* システムと同様に，画像特徴として色領域 blob を用いる．その理由は，画像上で手先や足先の追跡が他の画像特徴に比べ安定で，抽出精度の人体姿勢依存性が低いと考えるからである．

3.3 Blob 特徴の追跡

以下では，手や顔に相当する色領域 blob を追跡し，その 3 次元位置を求める方法について述べる．

3.3.1 領域の検出

背景差分と外接矩形の検出 前処理として，背景差分を施し，人体領域を囲む外接矩形を検出する．

色識別 本手法では，入力画像中に観測される肌色領域は手，顔のいづれかの領域であると解釈される．色識別の方法に，照明変化に比較的ロバストな方法として，パラメトリックな色モデルによる識別手法を用いている [9]． 各画素の色特徴 (r,g,b) が，次のような濃淡値 i をパラメータとした 2 次形式で表せると仮定する．

$$\begin{aligned} R &= R_2 i^2 + R_1 i, \\ G &= G_2 i^2 + G_1 i, \\ B &= B_2 i^2 + B_1 i \end{aligned} \tag{1}$$

ここで，6 つのモデルパラメータ係数 R_1, $\cdots$, B_2 は，予め学習用実画像から推定したものである．

色識別においては，以下の式により，各画素について観測される色特徴 (r,g,b) とモデル色特徴間のマッチングを行い，しきい値処理を行う．

$$error = (\hat{r} - \hat{R})^2 + (\hat{g} - \hat{G})^2 \tag{2}$$

ここで，$\hat{r} = r/(r+g+b)$, $\hat{g} = g/(r+g+b)$, $\hat{R} = R/(R+G+B)$, $\hat{G} = G/(R+G+B)$.

3.3.2 Blob の定義

領域特徴の 1 次モーメントまで考慮したガウシアン blob 特徴は，楕円形状モデル $\mathbf{e} = (\mu, \mathbf{\Sigma})$ である (平均 μ，共分散 $\mathbf{\Sigma}$)．しかし，肌色領域のように，同じ色特性の領域を複数個追跡する場合，このモデルでは画素単位での識別に誤りが生じる恐れがある．領域追跡を正確に行うためには，より正確な領域形状，領域境界を把握する必要がある．

そこで，本研究では，blob 特徴に正確な領域境界を表現可能なように局所的な変形 $\mathbf{d}$ を追加する．

$$\mathbf{p}(s) = \mathbf{e}(s) + \mathbf{d}(s) \tag{3}$$

ここで，s は楕円周のパラメータを表す．以下，「blob」を，式 (3) の $\mathbf{p}$ と定義する．

3.3.3 画素の識別

初期ラベリング まず，外接矩形内の領域を各ガウシアン blob 領域へとクラスタリングする．どの部位に相当するか決定する尺度としては，前フレームまでに推定された 2 次元重心位置 (あるいは予測位置) との近接性のみを用いる．これにより，初期の推定位置 (μ', $\mathbf{\Sigma}$') が求まる．

変形楕円による輪郭境界の探索 次のステップは，初期推定位置 (μ',$\mathbf{\Sigma}$') を用いて領域境界を決定する，すなわちどの画素が各 blob に属するかを厳密に決定することである．境界探索のための探索領域を，初期楕円の周上に配置する (図 11(上) 参照)．各探索点における局所窓 W から，尤もらしい領域境界を決定する．

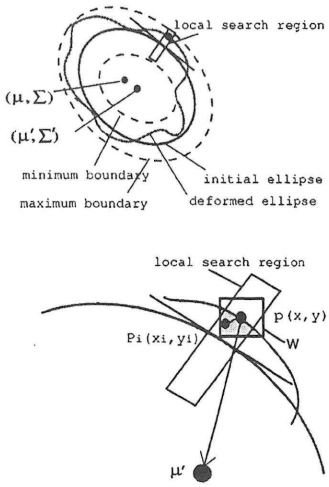


図 11: (上) 変形楕円モデル．(下) 境界探索の領域．

通常，境界を決定する尺度としてエッジ勾配などが用いられるが，本手法では，衣服のエッジの影響も考えられるため，以下のような色の分離境界を定める分離度 $J(\mathbf{p})$ をその尺度としている．

$$if \sum_{(x_i,y_i)\in W} n(x_i,y_i) > 0$$
$$J(\mathbf{p}) = |N_W/2 - \sum_{(x_i,y_i)\in W} p(x_i,y_i|skin) + \varepsilon|^{-1} \quad (4)$$

ここで，$p(x_i,y_i|skin)$ は肌色の尤度，N_W は W 内の画素数，$n(x_i,y_i) = \mathbf{p}_i\vec{\mathbf{p}} \cdot \mu'\vec{\mathbf{p}}$(図 11(下) 参照)．したがって，$\mathbf{d}(\mathbf{s}) = \arg\max J(\cdot)$.

推定した変形楕円内で対応画素の重心 μ を計算することにより，blob の 2 次元位置を求める．

3.3.4 3 次元位置の復元

2 視点の画像から blob 特徴が観測される時，その 3 次元位置をステレオ視により計算することが可能である．予め得られるキャリブレーション情報により，カメラ座標原点と blob 中心位置を結ぶ視線を各視点について計算し，それら視線の交差位置を求める．

視線 1 が，$\mathbf{T}_1 = \mathbf{o}_1 + t_1\mathbf{d}_1$ と表現され (t_1 は媒介変数，$\mathbf{o}_1$ はその視点のカメラ座標原点，$\mathbf{d}_1$ はその単位視線ベクトル)，視線 2 が，$\mathbf{T}_j = \mathbf{o}_2 + t_2\mathbf{d}_2$($t_2$ は媒介変数，$\mathbf{o}_2$ はカメラ座標原点，$\mathbf{d}_2$ は単位視線ベクトル)

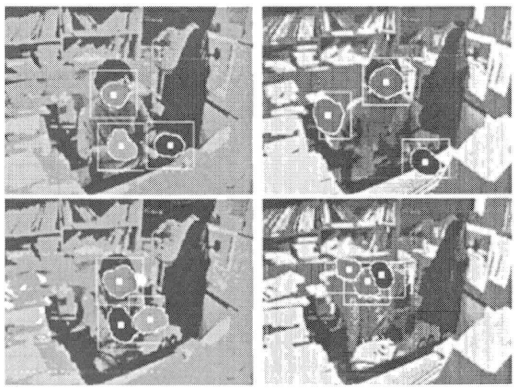


図 12: 推定した変形楕円領域の例．

と表現される時，視線 1 上にあり，視線 2 から最小距離である点 $\mathbf{T}$ は，t_1 をパラメータとして求めると以下のようになる．

$$\mathbf{T} = \mathbf{o}_1 - \frac{(\mathbf{d}_1 \times \mathbf{m}_2, \mathbf{o}_1 \times \mathbf{m}_2 - \mathbf{n}_2)}{||\mathbf{d}_1 \times \mathbf{m}_2||^2}\mathbf{d}_1 \quad (5)$$

ただし

$$\mathbf{m}_j = \frac{\mathbf{d}_j}{\sqrt{1 + ||\mathbf{o}_j \times \mathbf{d}_j||^2}},$$
$$\mathbf{n}_j = \frac{\mathbf{o}_j \times \mathbf{d}_j}{\sqrt{1 + ||\mathbf{o}_j \times \mathbf{d}_j||^2}}.$$

この点 $\mathbf{T}$ が 3 次元 blob 位置である．

3.3.5 隠れの検出

人体領域の追跡においては，顔領域が一定位置にあり，手領域が顔領域を隠す状況や，片方の手領域がもう一方の手領域を一時的に隠し，横切るというような状況が起こり得る．この場合，blob 領域の可視性が重要である．推定したモデルの 3 次元予測位置を再投影することにより，どの blob が可視であるかを判定することが可能である．このモデルベースの予測を「画像生成による解析」[12] と呼ぶ．具体的には，まず，カメラ座標原点からの距離を Z バッファ法により計算し，次フレームでの blob の可視性を求める．それをもとに，隠れが起こっている blob についてはその影響を最小限にするため，画素識別の際に隠れた境界の推定を行わず，初期楕円弧に境界を固定するように制限する (図 13参照)．

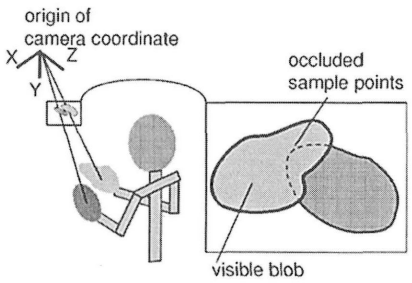


図 13: 隠れの検出.

3.4 人体モデルの動き生成

3.4.1 推定 blob の人体モデルへのあてはめ

知覚モジュールにより得られるのは胴体重心・頭部・両手・両足の計 6 つの 3D blob 位置 (以後，これを知覚データと呼ぶ) のみである．輪郭解析などを行い肘・膝などの関節位置を推定する方法 [13][14] もあるが，不安定な結果しか得られず推定可能な条件も厳しくなる．したがって，限られた少数の知覚データを用いて人体モデルの姿勢を生成するため，以下の逆運動学 (インバースキネマティクス) を用いてこの問題を解決する．なお人体モデルの表現としては，知覚データより求まる位置に両膝・両肘を加えた部位数 14，自由度 23(詳細は後述) を考えた多関節構造モデルとする (図 14).

3.4.2 実時間逆運動学

従来の手法　逆運動学問題は一般に，ロボティクス，コンピュータグラフィクス [15] の分野で用いられる．解法のアプローチとしては，解析的に解く方法 [17]，数値解析で解く方法 [16] があり，また，ヤコビ法，最適化法などが提案されている．また，腕のみであるが，実際の人体の動きデータを収集しそれらから導出した線形関係式を用いるユニークな方法などもある [18]．ロボティクスおよびその関連分野では，生体に近い腕の到達運動など研究されているが，実際は動力学と併用されることが多く，質量，弾性など対象に関する多くの事前情報が必要とされる．一方，コンピュータグラフィクスの分野でも近年，実時間での逆運動学問題解法が開発されつつある．しかしながら，利用用途がモデリングツールであり，ユーザとのインタラクションを前提としているため，時系列データを必要としない最適化手法を用い，局所解に陥った場合はユーザが修正するようなアプローチが用いられている．

提案手法　我々の目的においては逆運動学に関し，上述の一般的手法と異なる以下の性質が望まれる．

- 実時間で両腕両脚の計 4 つの接続リンクに関する逆運動学について同時に解が得られる
- 知覚モジュールにより入力されるゴール (3-D blob) 位置は精度が悪いということを前提とする
- 連続的かつ自然な人体の動きを与える

本論文では，上記の性質を考慮したアプローチとして解析的に解く方法を提案する．まず逆運動学におけるゴールとして様々なものが考えられるが，我々のような目的の場合，知覚モジュールから入力される 3-D blob 位置が正確でないことが起こりうる．特に，ゴール位置が解の存在範囲を越えて設定された場合，連続性が満たされなくなる．よって，実際のゴールを方向およびその距離 (絶対値) と定義することで対処する．つまり，ゴールの方向は必ず満たし，その距離は設定ゴール位置が解の存在範囲内である時のみ 3-D blob 位置に一致させる (図 14の左上参照)．一般に位置姿勢表現はロール・ピッチ・ヨーなどの回転・並進自由度が用いられ，それらが多関節構造の相互依存関係にあるため，このままでは解析的に解くことは困難である．そこで解析的に解けるよう自由度を適切に分ける方法が必要である．

解法　2 つの可変リンクに関する運動学方程式を以下のように定義する．

$$\begin{aligned}\mathbf{g}_i = \quad & \mathbf{T}^b\mathbf{R}^b(0, {R_y}^b, {R_x}^b)\mathbf{R}^{b'}({R_z}^{b'}, 0, 0) \\ & \mathbf{T}^{l_1}\mathbf{R}^{l_1}({R_z}^{l_1}, {R_y}^{l_1}, 0)\mathbf{R}^{l'_1}({R_z}^{l'_1}, 0, {R_x}^{l'_1}) \\ & \mathbf{T}^{l_2}\mathbf{R}^{l_2}(0, 0, {R_x}^{l_2})\ \mathbf{t}^e\end{aligned} \tag{6}$$

ここで，$\mathbf{g}_i = (g_x, g_y, g_z, 1)^T$ は，ゴール $i(1, \cdots, 4)$ を表す位置ベクトル，$\mathbf{T}^b$ および $\mathbf{R}^b$，$\mathbf{R}^{b'}$ は胴体に関する相対並進，回転行列，$\mathbf{T}^{l_1}$ および $\mathbf{R}^{l_1}$，$\mathbf{R}^{l'_1}$，$\mathbf{T}^{l_2}$ および $\mathbf{R}^{l_2}$ はリンク 1，2 に関する相対並進，回転行列である ($'$ のものは解析的に解くために分けられた回転行列を表す)．$\mathbf{t}^e$ はリンク 2 の末端のゴール位置への局所座標系での並進ベクトルである．また $\mathbf{R}(R_z, R_y, R_x)$ の回転自由度 R_z, R_y, R_x はそれぞれロール・ピッチ・ヨーを表す．

すると，逆運動学方程式の解析解としては，その幾何学的な性質より

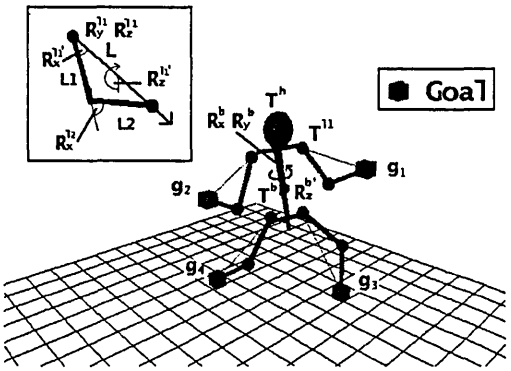


図 14: 人体モデルの構造およびゴール方向の定義

$$R_y{}^{l_1'} = -\arccos\left(\frac{g_z - {}^wT_z{}^{l_1}}{\|\mathbf{g} - {}^w\mathbf{T}^{l_1}\|}\right)$$

$$R_z{}^{l_1} = -\arctan\left(\frac{g_y - {}^wT_y{}^{l_1}}{g_x - {}^wT_x{}^{l_1}}\right)$$

$$R_x{}^{l_1'} = \arccos\left(\frac{{L_1}^2 + L^2 - {L_2}^2}{2L_1L}\right)$$

$$R_x{}^{l_2} = \arccos\left(\frac{{L_1}^2 + {L_2}^2 - L^2}{2L_1L_2}\right) - \pi$$

$$(|L_1 - L_2| \le L \le L_1 + L_2)$$

が得られる．ここで，L_1,L_2,L はそれぞれリンク 1，リンク 2 の長さ，リンク 1 の原点からゴールまでの距離を表し，${}^wT^*$ はワールド座標を表す (図 14参照)．

また，胴体の方向に関しては，胴体位置 ${}^w\mathbf{T}^b$ および頭部位置 ${}^w\mathbf{T}^h$ により ${}^w\mathbf{T}^h - {}^w\mathbf{T}^b$ の方向に体軸が平行になるように決定する．

$$R_x{}^b = -\arcsin\left(\frac{{}^wT_y{}^h - {}^wT_y{}^b}{\|{}^w\mathbf{T}^h - {}^w\mathbf{T}^b\|}\right)$$

$$R_y{}^b = -\arctan\left(\frac{{}^wT_x{}^h - {}^wT_x{}^b}{{}^wT_z{}^h - {}^wT_z{}^b}\right)$$

なお，$R_z{}^{l_1'}$ は (6) 式により，両腕両脚のそれぞれに設定されるパラメータであり，腕に関しては，肘の上げ下げ，脚に関しては膝の開き具合に対応する，より人間らしい動きを与えるための回転角である．これらは逆運動学の解からは直接得られないが，人体の姿勢表現には必要なパラメータであり，ここでは実際の測定値から実験的な定数値を当てはめた．また，$R_z{}^{b'}$ は人体の向き (パン) を定める回転角であり，これも逆運動学の解には直接影響を与えない，人体の姿勢表現には必要なパラメータであるが，人体の向き (パン) を 6 つの blob 位置のみから推定することは難しいため固定値にしており，その自動推定法の開発は今後の課題である．

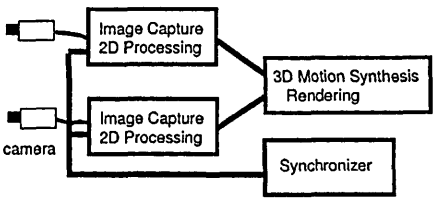


図 15: 多視点動画像処理の PC クラスタへの実装．

本手法の特徴をまとめると以下のようになる．

- 解析的な方法であるため実時間計算可能
- 少数の知覚データから人体モデルの姿勢を定める目的に特化しているため 2 つの可変リンクのみ適用可能
- 逆運動学の解には直接現れないが，人体の姿勢には影響を与えるパラメータを設定できるため多くの動作表現が実現可能

3.5 実験結果

3.5.1 システム構成

以下の実験では，2 ～ 6 台の PC が画像獲得・2 次元画像処理を行い，1 台の PC が 2 次元画像処理を行った PC からの情報を受信し，3 次元の復元処理，人体モデルの動作生成，人体モデルの描画を行う．さらに 1 台の PC が 2 次元画像処理を行う PC に同期信号を送信し，PC 間の同期を管理する (図 15)．入力画像のサイズは 320 × 240 画素 (YUV:422) で 30fps で動作可能である[1]．

3.5.2 多視点カメラの利用による計測範囲の拡張

本手法においては，人体動作の計測は正面像の対に対してのみ適用可能である．したがって，多視点カメラを用いて正面像の選択を行うことができれば，計測範囲の拡張を行うことが可能となる．しかしながら，

[1]DFW-V500 の仕様上，外部トリガにより同期をとると，最大 15fps の速度でしか画像を取り込めないが，システムの性能としては 30fps のスループットが得られている

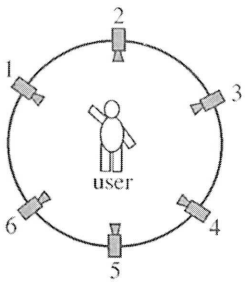


図 16: 6視点カメラの均等配置.

高精度に人体の向き (パン角) を求めるには胴体を正確に測定する必要があり，現在のアプローチでは困難である．そこで本実験では簡単に，両足先のなす角度により，正面像の視点対を選択することを考える．すなわち，以下の手順で視点選択を行う．

- 鉛直上方からの仮想カメラ座標を想定し，左右足先の投影点 $\mathbf{p}_l$, $\mathbf{p}_r$ を結ぶ2次元ベクトルと，各視点のカメラ座標原点の投影点 $\mathbf{p}_c$ ，ワールド座標原点の投影点 $\mathbf{p}_w$ を結ぶ2次元ベクトルを求める．
- それらを正規化し，ベクトルのなす角度を計算することにより，最適なすなわち直角に近い角度の隣り合う視点対を選択する．

本実験では，6つのカメラを同等の視野を保ちつつ60度の均等間隔で円周上に配置し (図 16)，直立状態の動作に関しては，どの向きでも正面像を観測可能な観測環境を想定した．図 17に，その入力画像および人体モデルの再構成像を示す．また，選択された視点対を太枠で示している．再投影による blob の予測位置を用いているので，滑らかな視点の切替えを行うことができ，推定位置の変動の影響は少ないことが確認できた．6つのカメラ程度でも，あらゆる人体の向きについての安定な正面像を選択可能であった．

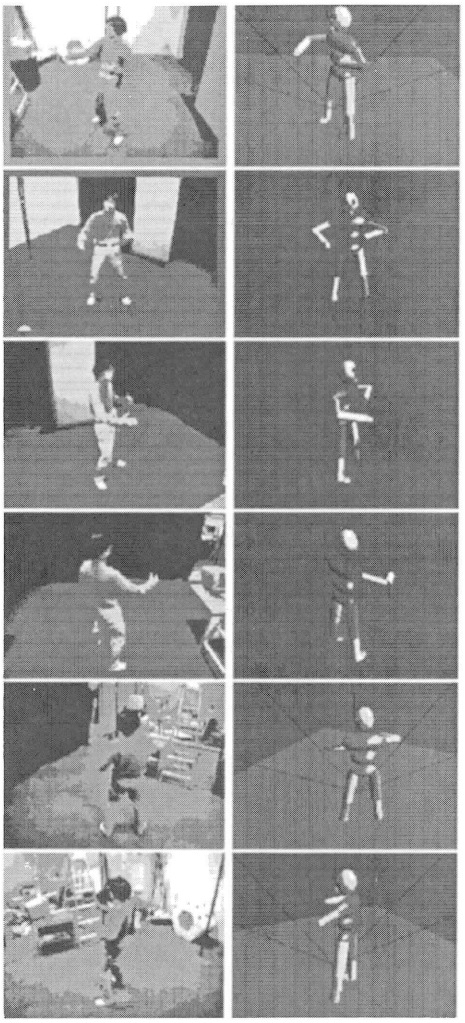


図 17: オンライン視点選択による全身動作の再構成の例 (左: 入力画像 (視点 1-6)． 右: 同一視点からの人体モデル再構成像).

4 おわりに

本論文では，実時間並列画像処理アプリケーションを作成するためのプログラミング環境である RPV について述べた．RPV を用いると，ユーザは，データ転送や同期，エラー処理と言った分散システム上で実時間処理を行う際の問題に悩まされることなく，データフロー情報とデータ処理アルゴリズムを記述するだけでプログラミングを行うことができる．また，RPV を利用した実時間マーカレスモーションキャプチャシステムを構築した．これにより，RPV を利用して実用的なシステムを構築することができることを示した．

今後の課題としては

- ノード PC とネットワークについての負荷分析を利用した半自動負荷分散ツールの構築
- オーバヘッドの解析によるシステム性能の向上

- 力学的あるいは感性的な動作フィルタリングを施すことによる，より人間らしい動作の生成や，知覚データに大きく影響されない安定な動作の再現

が挙げられる．

参考文献

[1] 松山, "分散協調視覚 —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—," 画像の認識・理解シンポジウム, 分冊 I, pp.343–352, July, 1998.

[2] 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.

[3] Message Passing Interface Forum, "MPI: A message-passing interface standard", Int. J. Supercomputer Applications and High Performance Computing, vol.8, no.3, pp.159–416, 1994.

[4] R. Taniguchi, Y. Makiyama, N. Tsuruta, S. Yonemoto, D. Arita, "Software platform for parallel image processing and computer vision," SPIE'97, Parallel and Distributed Methods for Image Processing, pp.1–10, July, 1997.

[5] H. Matsuo, K. Nakada, and A. Iwata, "A Distributed Image Processing Environment VIOS III and it's Performance Evaluation," Int. Conf. on Pattern Recognition, vol.II, pp.1538–1542, August, 1998.

[6] 松山, 浅田, 青山, 浅津, "再帰トーラス結合アーキテクチャにおける並列対象認識のためのデータレベル並列プロセスの構成," 情処学論, vol.36, no.10, pp.2310–2320, 1995.

[7] H. Tezuka, A. Hori, Y. Ishikawa, M. Sato, "PM: an operating system coordinated high performance communication library," High-Performance Computing and Networking, P. Sloot and B. Hertzberger, Springer-Verlag, pp.708–717, 1997.

[8] D. L. Mills, "Internet time synchronization: the network time protocol," IEEE Trans. Communications, vol.39, no.10, pp.1482–1493, 1991.

[9] 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ー視覚に基づく仮想世界とのインタラクションー," 映像情報メディア学会誌, Vol.54, No.3, pp.409–416, 2000.

[10] 谷口, 米元, "多視点カメラを用いた 3 次元人体動作の実時間推定," 情処研報, 2001-CVIM-128, pp.65–72, 2001.

[11] C. Wren, A. Azarbayejani, T. Darrell, A. Pentland, "Pfinder: Real-Time Tracking of the Human Body," IEEE Trans. Pattern Analysis and Machine Intelligence, Vol.19, No.7, pp.780–785, 1997.

[12] S. Yonemoto, N. Tsuruta, R. Taniguchi, "Tracking of 3D Multi-Part Objects Using Multiple Viewpoint Time-varying Sequences," Proc. of 14th Int. Conf. on Pattern Recognition, pp.490–494, 1998.

[13] Y. Azoz, L. Devi, R. Sharma "Reliable Tracking of Human Arm Dynamics by Multiple Cue Integration and Constraint Fusion," Proc. of Conf. on Computer Vision and Pattern Recognition, pp.905–910, 1998.

[14] 岩澤, 海老原, 竹松, 坂口, 大谷, "Shall We Dance?"の構築," 信学技報, PRMU98-114, pp.15–22, 1998.

[15] C. Welman, "Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation, MSc Thesis," CS, Simon Frazer University, 1993.

[16] J. Zhao, N. Badler, "Inverse Kinematics positioning using nonlinear programming for highly articulated figures," Trans. Computer Graphics, Vol.13, No.4, pp.313–336, 1994.

[17] D. Tolani, "An Inverse Kinematics Toolkit for human modeling and simulation," PhD Thesis, University of Pennylvania, 1998.

[18] Y. Koga, "Planning Motions with Intentions," Proc. of SIGGRAPH'94, pp.24–29, 1994.