NEDO（新エネルギー・産業技術総合開発機構）
即効型地域新生コンソーシアム研究開発

# 柔軟変形物ハンドリング用ビジョンチップの研究開発

# 報告書

平井 慎一　　立命館大学ロボティクス学科
有本 卓　　立命館大学ロボティクス学科

大西 弘之　　グローリー工業株式会社 研究開発センター
山本 真也　　グローリー工業株式会社 研究開発センター
柳内 さゆり　グローリー工業株式会社 研究開発センター
本林 満英　　グローリー工業株式会社 イメージ開発部

山本 武久　　NKE 株式会社

本報告書は NEDO（新エネルギー・産業技術総合開発機構）平成 13 年 3 月 29 日付け委託契約に基づく開発項目「地域新生コンソーシアム研究開発　即効型地域新生コンソーシアム　柔軟変形物ハンドリング用ビジョンチップの研究開発」(平成 13 年 3 月 29 日～平成 14 年 3 月 31 日）に関する研究成果をまとめたものである.

## 要約

本プロジェクトの目標は，柔軟物ハンドリングのために，ビデオフレームレートで平面運動物体の位置と姿勢を検出することができるビジョンチップを開発することである．運動物体の位置と姿勢を検出することは，様々な場所に置かれている物体のハンドリングや変形を伴う物体のハンドリングなど，高度なハンドリングの基盤となる技術である．多くのビジョンアルゴリズムは，計算量が多い半面，並列性が高いという特性を持っており，ビジョンアルゴリズムを ASIC 化することは性能の面で効果が高い．一方，ビジョンシステムは，ASIC 化がコスト的に優位なほど，多くの場所で使われることはないのが現状である．さらに，アプリケーションに応じて，アルゴリズムのチューニングや修正が必要になる場合が多い．したがって，ブロックマッチング等を除くと，ビジョンアルゴリズムの ASIC 化はコスト的に困難な点が多い．そこで本プロジェクトでは，ビジョンアルゴリズムを FPGA(Field Programmable Gate Array) 上に実装する．FPGA とは，論理回路を書き換えることが可能な VLSI である．ユーザが設計した論理回路を FPGA に書き込むことにより，FPGA はユーザが設計した回路として動作する．論理回路をユーザが設計できるというソフトウェアの長所と，論理回路として高速に動作するというハードウェアの長所を併せ持つ．

本プロジェクトでは，片側ラドン変換法とフーリエ・ラドン変換法を FPGA 上に実装し，ビデオフレームレートで平面運動物体の位置と姿勢を検出することができるビジョンシステムを構築する．さらに，開発したビジョンシステムを用いた物体ハンドリングを実現する．本プロジェクトの成果を，以下に列挙する．

- FPGA ベースのリアルタイムビジョンシステムを構築した．200 万ゲート規模の FPGA（Xilinx 社 Vertex-E），ビデオデコーダー，ビデオエンコーダー，SRAM，PC インターフェースを実装し，システムを構築した．
- 片側ラドン変換法アルゴリズムを，システムコンパイラで記述した．さらに，ビデオデコーダー，ビデオエンコーダー，SRAM のシミュレーション記述を，システムコンパイラで行った．位置・姿勢の検出のビデオレート処理 (33msec) を実現した．
- マッチトフィルター法のフーリエ変換の計算回数を削減するため，ハフ・フーリエ変換法の原理を利用したフーリエ・ラドン変換法を開発した．フーリエ・ラドン変換法を PC 上において C 言語で実装し，運動物体の位置・姿勢の検出が可能であることを検証した．

- FPGA ベースのリアルタイムビジョンシステムを構築した．200 万ゲート規模の FPGA（Xilinx 社 Virtex-E），外部 RAM，PC インターフェースを実装し，システムを構築した．
- Handel-C によりマッチトフィルター法の論理設計を行い，FPGA へ実装し，位置・姿勢の検出の処理速度は，ビデオレート処理 (33msec) を実現した．
- 視覚情報に基づき，リアルタイムで軌道を修正する手法として，速度追従法を開発した．
- スカラアームと 7 軸マニピュレータをビジョンシステムと接続し，バラ積み運動物体の整列を実現した．

本プロジェクトの成果は，高速物体ハンドリング，画像検査への応用が可能である．

## SUMMARY

In this project, we will develop realtime vision chips to detect the position and the orientation of planar motion objects for their dynamic handling. Current vision systems are categorized into two; software-based approach and ASIC-based approach. Vision algorithms are implemented on programs, which are performed on a general-purpose MPU, in a software-based approach. This approach can perform various algorithms and can construct a system in a low cost. On the other hand, it often requires much computation time and fails in realtime processing. Logic circuits specialized to individual vision algorithms are designed and are implemented on LSI's in an ASIC-based approach. Fast computation can be performed in the ASIC-based approach but it requires huge time and cost to design and to implement logic circuits on ASIC's. Consequently, software-based approach has good flexibility but lacks realtimeness while ASIC-based approach has good realtimeness but lacks flexibility. To overcome this dilemma, we will develop a vision system based on FPGA's. FPGA's are special VLSI's where users can configure logic circuits. We can design a logic circuit specialized to a vision algorithm and can implement the designed logic circuit on an FPGA. This implies that realtimeness of a vision algorithm can be realized by implementing the algorithm in a logic circuit and flexibility of a vision system can be realized by redesigning and reconfiguring the logic circuit on an FPGA.

In this project, we will implement two vision algorithms on FPGA's; one-sided Radon transform method and Fourier-Radon transform method. The developed FPGA-based vision system can detect the position and the orientation of a planar motion object in the video-frame rate. Moreover, we will realize the object handling using the vision system. The results of this project are summarized as follows:

- We have developed a prototype of the FPGA-based realtime vision, which consists of Xilinx Vertex 2000-E, video decoder, video encoder, SRAM, and PC interface.
- We have introduced SystemCompiler that provides a design of logic circuits using C/C++ language instead of hardware description languages. We have implemented the one-sided Radon transform method on an FPGA using the SystemCompiler and have found

that the detection of the position and the orientation of a planar motion object can be performed in the video-frame rate.

- Fourier-Radon transform method has been developed to reduce the computation in the matched filter approach. We have implemented this method on a PC to find that it can detect the position and the orientation of a planar motion object.

- We have developed an FPGA-based realtime vision system consisting of FPGA (Xilinx Vertex-E), SRAM, and PC interface.

- We have introduced DK1 that provides a design of logic circuits using C/C++ language. We have implemented the matched filter on an FPGA using the DK1. We have shown that the position and the orientation of a planar motion object can be detected in the video-frame rate.

- Velocity-following method has been proposed to update the motion of a handling device according to the visual information of a handling object.

- We have prototyped a handling system consisting of a scalar arm, a 7-DOF manipulator, and the realtime vision system to perform the sorting of randomly-located, moving objects successfully.

The results of this project can be applied to fast object handling and object inspection.

# 目 次

# 図 目 次

# 表 目 次

# 第1章　FPGAベース
# リアルタイムビジョン

## 要旨

**1. 目的**

ビデオフレームレートで平面運動物体の位置と姿勢を検出することができるビジョンチップを FPGA をベースに開発する．

**2. 成果**

**(1) 柔軟変形物の運動を認識するアルゴリズムの開発**

・ 片側ラドン変換法を PC 上で改良し，FPGA で効率の良いアルゴリズムを開発した．並列性が高く，メモリアクセスが可能な限りシーケンシャルであるように，アルゴリズムを再構成した．

・ 片側ラドン変換法の二つのアルゴリズム，簡略版ならびに完全版ともに，並列アルゴリズムにより，変形運動物体の運動を検出できることを，PC 上で確認した．

**(2) ビジョンアルゴリズムの FPGA 上への実装**

・ FPGA ベースのリアルタイムビジョンシステムを構築した．200 万ゲート規模の FPGA（Xilinx 社 Vertex-E），ビデオデコーダー，ビデオエンコーダー，SRAM，PC インターフェースを実装し，システムを構築した．

・ システムコンパイラを導入し，C レベルの論理回路設計が可能な環境を構築した．

・ 片側ラドン変換法アルゴリズムを，システムコンパイラで記述した．さらに，ビデオデコーダー，ビデオエンコーダー，SRAM のシミュレーション記述を，システムコンパイラで行った．

・ 位置・姿勢の検出のビデオレート処理 (33msec) を実現した．

**3. 今後の課題**

・ 物体変形に対応できるアルゴリズムの開発

## 1.1　はじめに

運動物体の位置と姿勢を検出することは，様々な場所に置かれている物体のハンドリングや変形を伴う物体のハンドリングなど，高度なハンドリングの基盤となる技術である．たとえば，食品産業においては，多くの食品が人手によりハンドリングされている．食品の衛生を保つために，ハンドリングの自動化が望まれている．しかしながら，食品はランダムな位置や姿勢で置かれていることが多く，ハンドリングの自動化は困難である．したがって，食品の位置や姿勢を検出することは，自動ハンドリングのキーとなる技術である．また，リサイクルにおいては，分解部品の分別の多くが人手により成されており，分別精度の向上のために自動化が望まれている．このとき，分解部品の位置と姿勢を検出することが要求される．

物体ハンドリングにおいては，対象物の運動を計測し，ハンド等の運動を制御する必要がある．一般的に対象物は平面運動を行うので，平面運動における位置と姿勢を検出できれば十分である．平面運動を検出する方法は，1) 正規化相関法，2) 輪郭法，3) 一般化ハフ変換法 [1]，4) ラドン・フーリエ変換法 [2] など，多くの手法が提案されている．正規化相関法は，物体の並進運動を検出できる反面，物体の回転運動を検出できない．輪郭法は，物体の並進運動と回転運動を検出することができる．一方，凹物体や穴を有する物体など，輪郭関数を定義できない場合や，物体の輪郭が検出できない場合には適用できない．一般化ハフ変換法とラドン・フーリエ変換法は，物体の並進運動と回転運動を検出することが可能であり，どのような物体にも適用することができる．すなわち，重心や輪郭が不明確な場合でも，物体の位置と姿勢を計測することができる．これら二つの手法の計算量は極めて大きいが，計算の並列性が高く，システム LSI 化することにより高速に実行することが期待できる．本プロジェクトでは，特にラドン・フーリエ変換法に焦点を当て，ラドン・フーリエ変換チップを試作するとともに，チップを用いた柔軟物ハンドリングを実現することを目指す．

著者らは，片側ラドン変換を用いて，平面運動物体の位置と姿勢を検出する三つのアルゴリズムを提案した [3]．これらのアルゴリズムは，CCD カメラからの入力画像とあらかじめ撮影した参照画像を比較することによ

り，任意形状の物体の位置と姿勢を算出することができる．アルゴリズムの一つは物体の画像重心の計算を必要とせず，他の二つは画像重心の計算を必要とする．前者のアルゴリズムにおいては，対象物体を背景から分離する必要がないため，対象物体が他の物体と重なっていたり，対象物体の一部が画像の外にある場合でも，位置と姿勢の検出が可能である．しかしながら，このアルゴリズムは，片側ラドン変換の計算，パワースペクトルの計算，パワースペクトルの二次元マッチングから構成されており，多くの計算時間を要する．一方，これらの計算は並列性が高く，並列演算により計算時間を短縮できる可能性が高い．

本章では，平面運動物体の位置と姿勢をビデオフレームレートで検出するために，FPGA ベースリアルタイムビジョンを開発する．まず，FPGA を用いたビジョンシステムを構築する．次に，片側ラドン変換を用いて，物体の位置と姿勢を検出するアルゴリズムについて説明する．次に，片側ラドン変換の並列演算について考察し，投票の概念に基づく並列アルゴリズムを提案する．最後に，片側ラドン変換アルゴリズムを FPGA ベースリアルタイムビジョンに実装する．

## 1.2　ビジョンチップの例

視覚信号は，二次元の時系列信号であり，その情報量は極めて大きい．したがって，画像処理には，多くの演算を要求される．たとえば，512×512 ピクセルの画像に，サイズ $3 \times 3$ の線形フィルタを適用する場合，$2 \times 10^6$ 回以上の乗算が必要である．画像がビデオフレームレート，すなわち 30 frame/s (fps) で得られる場合，演算速度は 70 MFLOPS を越える．また，フーリエ変換やハフ変換など，画像処理に有効であることが示されているが，演算量の大きいアルゴリズムが多い．一方，画像処理は一般的に並列性が高く，処理の高速化のためにはシステム LSI 化が有効な手段である．特に，ロボティクスにおけるビジョンにおいては，リアルタイムで画像処理を実行する必要があり，ビジョンチップの開発が進められてきた．また，画像における変換は，計算時間を要する処理であり，フーリエ変換の並列演算 [4]，ハフ変換の並列演算 [5] に関する研究が進められた．本節では，ビジョンチップ開発の例をいくつか紹介する．

**相関演算チップ**　対象物を発見し追跡することは，ロボットビジョンにおける最も基本的な機能であり，ターゲットトラッキングとよばれる．ターゲットトラッキングを実現する一つの方法は，あらかじめ与えられる参照画像と類似度の高い領域を，入力画像内から探索する手法である．入力画像内で，参照ウインドウと同じサイズを有する領域を，候補ウインドウとよぶ．類似度を評価する一つの規範は，二つの画像領域間の相関関数 (Correlation) である．相関関数は 0 以上 1 以下の値を持ち，二つのウインドウ内の画像が一致しているとき相関関数の値は 1 である．ウインドウのサイズを $M \times N$ で表し，参照ウインドウ $RW$ 内の画素値を $R_{0,0}, R_{0,1}, \cdots, R_{M-1,N-1}$，候補ウインドウ $CW$ 内の画素値を $C_{0,0}, C_{0,1}, \cdots, C_{M-1,N-1}$ とする．このとき相関関数 $Cor$ は，

$$Cor = \frac{E_{RC}}{\sqrt{E_{RR}E_{CC}}} \tag{1.1}$$

で与えられる．ここで

$$E_{RC} = \sum_{i=1}^{M-1}\sum_{j=1}^{N-1} R_{i,j}C_{i,j},$$

$$E_{RR} = \sum_{i=0}^{M-1}\sum_{j=1}^{N-1} R_{i,j}^2, \quad E_{CC} = \sum_{i=0}^{M-1}\sum_{j=1}^{N-1} C_{i,j}^2.$$

すべての候補ウインドウに対して相関関数の値を計算し，相関関数の値が最大となる候補ウインドウの位置を物体の位置とする．以上の手法を，相関法とよぶ．

上式に示す相関関数は積の個数が多く，演算時間を要する除算と平方根を含む．そこで，相関関数に近い性質を持ち，計算が単純な規範が使用されることがある．このような規範の一つは，二つの画像領域間の絶対差分和 (Sum of Absolute Differences, SAD) である．絶対差分和は，絶対誤差平均 (Mean Abosule Difference, MAD) とも言われる．SAD は 0 または正の値を取り，二つのウインドウ内の画像が一致しているとき SAD の値は 0 である．すなわち，SAD の値が小さいほど，相関度が高い．絶対差分和 SAD は，

$$SAD = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} |R_{i,j} - C_{i,j}| \tag{1.2}$$

で与えられる．すべての候補ウインドウに対して SAD の値を計算し，SAD の値が最小となる候補ウインドウの位置を物体の位置とする．

相関法の計算は単純であり並列性が高いので，VLSI 化に向いている．SAD の計算を行うチップを用いて実時間ビジョンシステムが構築され [6]，マイクロプロセッサのコプロセッサに適したチップが開発された [7]．また，SAD と正規化相関関数の計算アルゴリズムが VLSI 化され，それを用いたビジョンシステムがトラッキングビジョンの名称で市販されている [8]．

**ステレオビジョンチップ**　ステレオビジョンの原理は三角測量であり，ステレオビジョンにおいて三次元情報を得るためには，複数の画像間の対応点を求める必要がある．図 1.1 に示すように，対応点の算出では一般に，一方の画像に参照ウインドウを設定し，それに対応する領域を他方の画像内から探索する．三次元空間内の点と二つのカメラの投影中心により定まる面をエピポーラ面，エピポーラ面と画像平面との交線をエピポーラ線とよぶ．画像間の対応を求めるときには，エピポーラ線上で対応点を探索すれば十分であることが知られている [9]．したがって，エピポーラ線上に複数の候補ウインドウを設定し，参照ウインドウと最も相関度の高い候補ウインドウを求める．このような探索を，複数の参照ウインドウに対して実行することにより，画像間の対応点を求める．以上のように，ステレオビジョンにおいて対応点を求める手続きは，計算量が多く，リアルタイムで実行することが困難である．そこで，対応点の算出を高速かつ正確に実行するために，ステレオビジョン VLSI チップが試作されている．

対応点を評価する一つの規範は，(1.2) 式に示す絶対差分和である．エピポーラ線上のすべての領域で SAD を計算し，SAD の最小値を求めることにより，対応点を決定する．対応点探索においては，図 1.2-(a) に示すように，(1.2) 式の $MN$ 個の絶対差分 (AD) を並列に計算できる．また，図 1.2-(b) に示すように，エピポーラ線上の複数の候補ウインドウに対して，並列に SAD を評価することができる．このように，対応点探索のアルゴリズムは並列性が高いため，アルゴリズムを VLSI 化することにより，高速に対応点を求めることができる．

対応点を求めるときの問題点は，ウィンドウのサイズである．ウインドウが小さいと，相関度の高い領域が複数存在する可能性が高くなる．すなわち，対応の一意性が失われる．一方，ウインドウサイズが大きいと，相

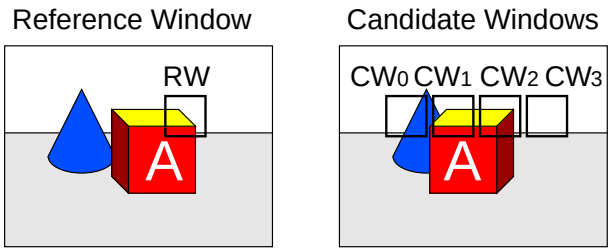
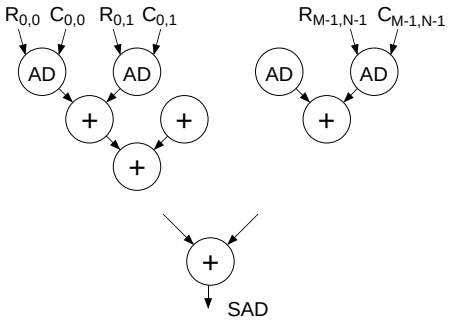
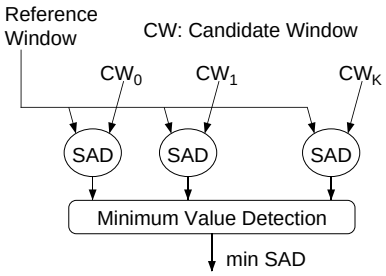



図 1.1: ステレオビジョンにおける対応点




(a) Pixel-parallelism




(b) Window-parallelism

図 1.2: 対応点マッチングにおける並列性

関度が全般的に低くなる．すなわち，ウインドウの中央部より遠い部分の画素が考慮されるため，対応点の信頼性が低下する．したがって，対応点を一意に決定することができ，可能な限りサイズが小さいウインドウを設定することが望ましい．そこで，ウインドウサイズを適応的に変化させるアルゴリズムが提案されている [10, 11]．張山らは，ウインドウサイズを適応的に変化させるアルゴリズムを，VLSI 上に実現している [12, 13]．このステレオビジョン VLSI においては，エピポーラ線上のすべての領域で SAD を計算する．SAD の最小値を $f_1$，二番目に小さい極小値を $f_2$ とする．ウィンドウサイズが $W \times W$ の場合，SAD の最小値のユニーク性を，$S_w = (f_1 - f_2)/W^2$ で評価する．この評価値を用いることにより，適切なウインドウサイズを自動的に決定している．このとき，図 1.2-(a) に示す AD の並列演算は演算部の稼働率が低くなるため採用せず，図 1.2-(b) に示す候補ウインドウに対する SAD の並列計算を採用している．対応点探索部には Module Based Array とよばれるゲートアレイを，入力画像に対する LOG フィルタと最小値検出部には FPGA(Field Programmable Gate Array) を用いている．

**高速ビジョンチップ**　ビデオ入出力で使用されている NTSC 信号のレートは 30fps である．これは，人間に動画を提示するときに十分なレートとして定められた規格である．一方，メカニカルシステムの安定な制御のためには，サンプリング周波数 1000Hz，すなわちサンプリング時間 1msec が必要である．したがって，ビジョン信号をメカニカルシステムのフィードバック制御に用いるためには，画像を 1000fps で取得し，フィードバック信号を求めることが望まれる．結局，NTSC 信号は，人間への画像の提示が目的であり，メカニカルシステムのフィードバック制御においてはレートが不足していることがわかる．そこで，メカニカルシステムの制御を目的として，高速で画像を取得し，信号処理を実行できるビジョンシステムが研究されている．

石川らは，ビジョンチップのアーキテクチャとして $S^3$PE (Simple and Smart Sensory Processing Elements) アーキテクチャを提唱し，フィードバックレート 1000Hz を有するビジョンシステムを開発している [14, 15]．このアーキテクチャにしたがって，小室らは，ターゲットトラッキング用

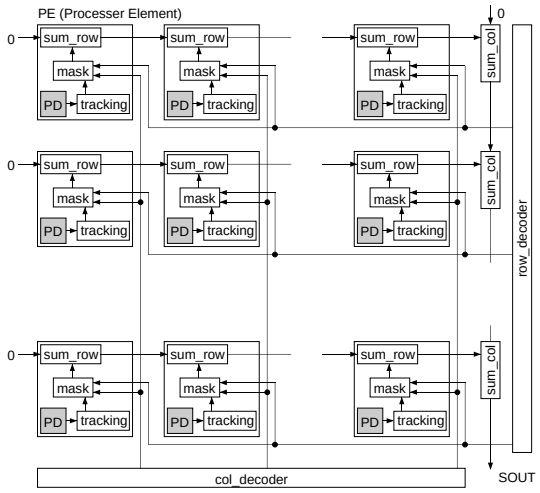



図 1.3: ターゲットトラッキングチップの構成

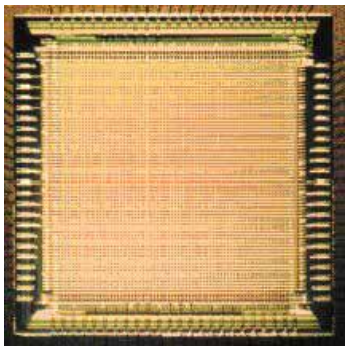


図 1.4: ターゲットトラッキングチップ

の高速ビジョンチップを開発している [16, 17]．図 1.3 に示すように，光検出器 (PD) と信号処理部を一体化した処理要素 (PE) を並列に配置する．PE は，トラッキング回路と行総和演算回路から構成される．各行の右端には，列総和演算回路が取り付けられている．行方向，列方向には，それぞれ制御ラインが構成されている．行方向，列方向の制御ラインへの信号は，命令をもとに行方向デコーダと列方向デコーダが生成する．チップの総和計算機能を用いることにより，画像の 0 次モーメントと 1 次モーメントを計算することができる．画像サイズ 256×256 の場合，0 次モーメントを 8$\mu s$ で，1 次モーメントを 66$\mu s$ で計算することができる．したがって，ターゲットの重心を，約 270$\mu s$ で求めることができる．0.6$\mu m$ CMOS プ

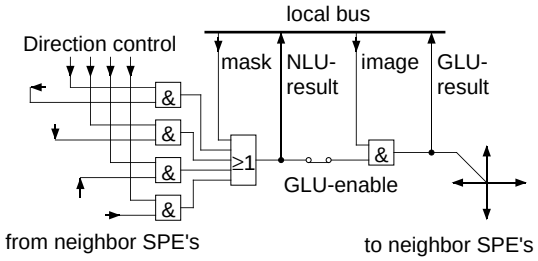



図 1.5: グローバル論理ユニット

ロセスを使用して，64 × 64 の要素を集積したチップを試作したところ，図 1.4 に示すチップが得られ，フレームレート 2.4msec でターゲットを追跡できると報告されている．

Eklund らは，Near-Sensor Image Processing (NSIP) の概念に基づき，ビジョンチップを開発している [18]．このビジョンチップでは，フォトダイオードと信号処理部を一体化した要素 SPE (Sensory Processing Element) を，二次元的に配置する．すべての SPE に，図 1.5 に示すグローバル論理ユニット (GLU) が接続されている．各 GLU は，隣接する SPE に対応する GLU と接続されている．GLU は画像全体で並列に作用し，演算結果を隣接 4 方向に伝搬させることができる．この機能を用いることにより，領域を囲む最小長方形の計算や連結領域の抽出等，広域的な演算を実現することができる．

## 1.3　FPGA を用いたビジョンシステム

前節で述べたように，従来様々なビジョンチップが開発されている．多くのビジョンアルゴリズムは，計算量が多い半面，並列性が高いという特性を持っており，ビジョンアルゴリズムを ASIC 化することは性能の面で効果が高い．一方，ビジョンシステムは，ASIC 化がコスト的に優位なほど，多くの場所で使われることはないのが現状である．さらに，アプリケーションに応じて，アルゴリズムのチューニングや修正が必要になる場合が多い．したがって，ブロックマッチング等を除くと，ビジョンアルゴリズムの ASIC 化はコスト的に困難な点が多い．現在の技術を見ると，CCD カメラと汎用のキャプチャボードを用いることにより，画像の入力は安価に実現できることがわかる．したがって，ディジタル画像の処理を高速かつ安価に実現する手段があれば，リアルタイムビジョンシステムを安価に実現できる．そこで本プロジェクトでは，ディジタル画像処理を FPGA(Field Programmable Gate Array) 上に実装する．FPGA とは，論理回路を書き換えることが可能な VLSI である．ユーザが設計した論理回路を FPGA に書き込むことにより，FPGA はユーザが設計した回路として動作する．論理回路をユーザが設計できるというソフトウェアの長所と，論理回路として高速に動作するというハードウェアの長所を併せ持つ．もちろん，ソフトウェアより自由度は低く，また ASIC より速度が遅く，消費電力が多い．しかしながら，ビジョンシステムに要求される特性を考えると，FPGA によりリアルタイムビジョンを構成する利点は大きいと判断する．

本研究では，実際のハンドリングで使用されるビジョンシステムを構築することを目的とする．物体ハンドリングにおいては，高いハンドリングスピードが要求されるため，ビジョンシステムには高速性が要求される．一方，対象とする物体や環境は多様であり，多様な物体ハンドリングに適用できる汎用性が要求される．高速性と汎用性を満たすために，FPGA を用いたビジョンシステムを構築する．FPGA においては，論理回路を書き換えることができるため，多様な物体ハンドリングに適用できる汎用性を実現できる．一方，ビジョンアルゴリズムを並列に実行することにより，高速性を実現することができる．試作した FPGA ボードを図 1.6 に示す．

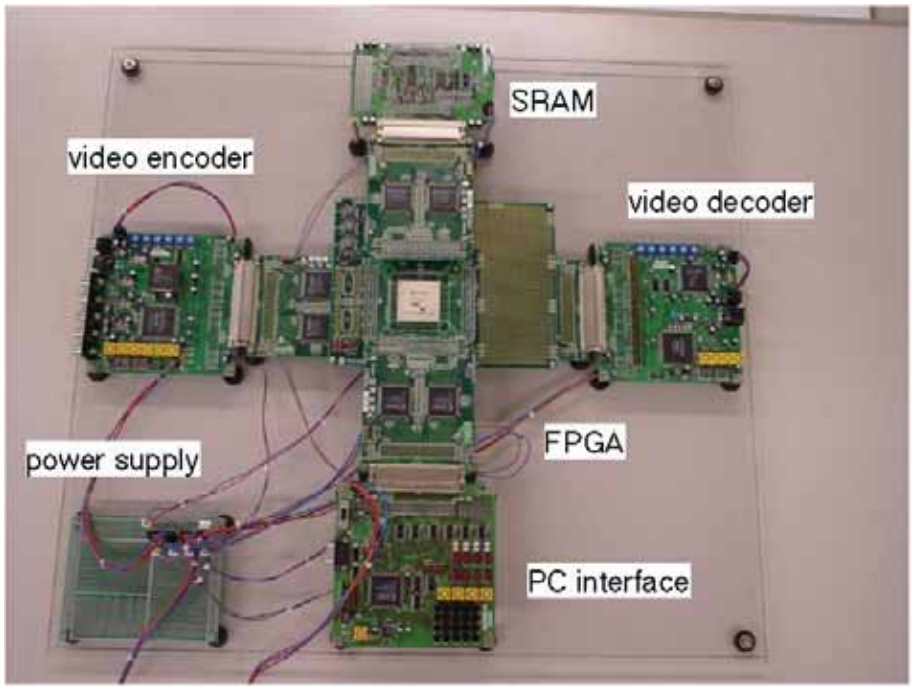



図 1.6: FPGA ベースリアルタイムビジョン

図に示すように，FPGA の周辺には，ビデオデコーダ，ビデオエンコーダ，SRAM が周辺に配置されている．画像の入力には，CCD カメラと Bt シリーズに代表されるビデオキャプチャチップを用いる．このとき，CCD カメラからの NTSC 信号は，順次 A/D 変換され，後段の画像処理部に送られる．画像処理部においては，ディジタル画像を扱うので，論理回路が構成できれば十分である．そこで，FPGA 上に，画像処理用のハードウェアを実現する．ただし，ラドン・フーリエ変換法では，ラドン変換やフーリエ変換の結果を保存しておくメモリが必要である．そこで，SRAM 内に変換の結果を保存し，必要に応じて参照する．使用している FPGA は，Xilinx Vertex-E 2000 である．FPGA の外観を図 1.7 に，仕様を表 1.1 に示す．

ハードウェア記述では，Verilog HDL や VHDL に代表されるハードウェア記述言語を使うのが，現在の主流である．一方，ロボティクスにおけるアルゴリズムの開発には，C/C++が広く用いられている．本プロジェクトにおけるラドン・フーリエ変換法のアルゴリズムも，C/C++で記述さ

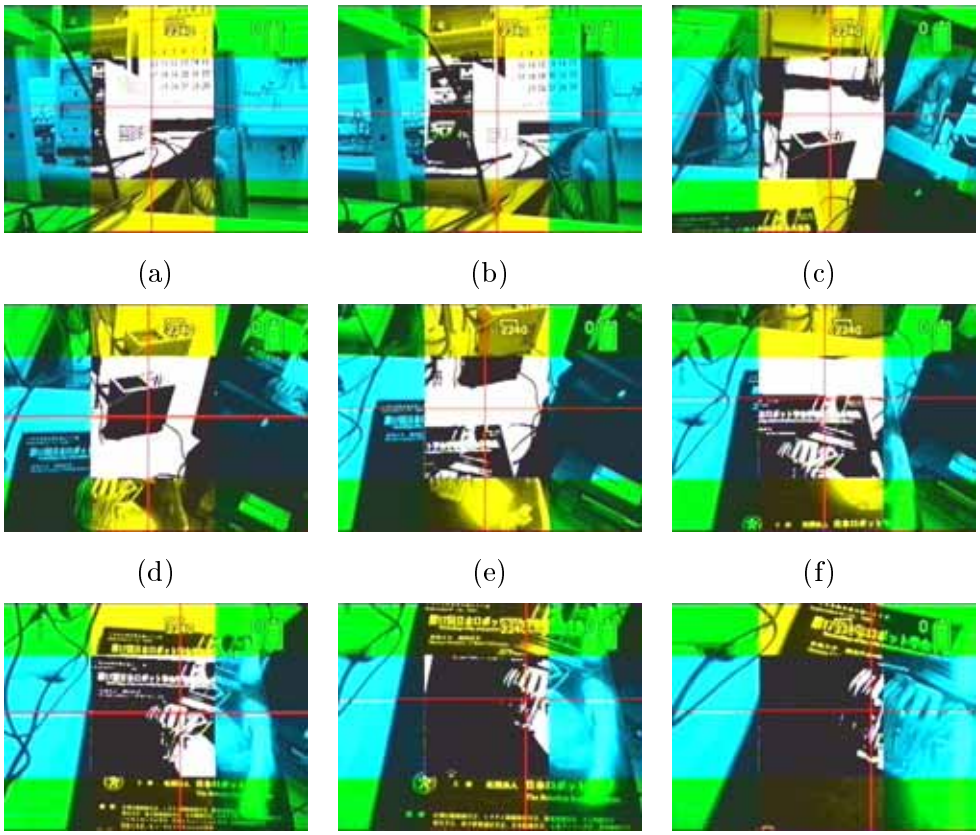

図 1.7: FPGA の外観

表 1.1: FPGA の仕様

| | | |
|---|---|---|
| メーカー | | Xilinx |
| 製品名 | | VertexE 2000 |
| 型番 | | MCV2000E BG560-8 |
| 仕様 | システムゲート数 | 200 万 |
| | パッケージ | 560 Ball Grid Array |
| | 外形寸法 | 42.5×42.5 [mm] |
| | 電源電圧 | DC+1.8V |
| | 製造プロセス | 0.18$\mu$ |
| | | 6 層メタルシリコンプロセス |

れている．アルゴリズムを VLSI に実装するためには，C/C++で記述したアルゴリズムをハードウェア記述言語で書き換える必要がある．しかしながら，アルゴリズムの規模が大きく，複雑になるにしたがって，アルゴリズムの開発とハードウェアの開発を繰り返し行う必要性が高くなる．特に，知能ロボットにおける信号処理アルゴリズムや運動制御アルゴリズムは，開発と検証を繰り返し行うことが多い．この繰り返しに要する時間を短縮するため，C/C++をベースとするハードウェア記述が提案されている．C/C++ ベースのハードウェア記述言語として，SystemC，SpecC，CycleC，Handel-C 等がある．本研究開発では，CycleC (C Level Design, Inc.) を用いて，ラドン・フーリエ変換のロジックを記述する．CycleC は，ANSI C/C++の文法のサブセットである．サブセットである CycleC の記述スタイルに従って，ハードウェアを記述する．論理回路の挙動は C++ のクラスとメソッドで記述する．実際の論理回路は，インスタンスに対応する．CycleC により記述されたロジックは，通常のコンパイルを行うことにより，ソフトウェアとして実行することができる．結局，Cycle C で記述した論理回路は，PC 上でコンパイルし，論理回路の挙動を評価することができる．一方，CycleC により記述されたロジックは，トランスレータにより HDL へ変換することができる．すなわち，Cycle C による記述されたロジックを，HDL に変換し，FPGA の論理回路を構成する．以上のように，通常のコンパイルと実行によるアルゴリズムの検証と，HDL への変換が両立しており，アルゴリズムの開発とハードウェアの記述を効率よく進めることが期待できる．

試作 FPGA ボードの性能を確認するために，重心計算を実装した．試作 FPGA ボードによる重心計算の結果を，図 1.8 に示す．FPGA では，1)NTSC 信号による画像の入力，2) 画像の SRAM への格納，3) 二値化重心の計算，4) 重心を示すカーソルの描画，5)NTSC 信号による画像の出力を行っている．画面中央部に設定したウインドウに対して，二値化重心を計算する．図より，一連の処理をビデオフレームレートで実行できることがわかる．



(a)　(b)　(c)

(d)　(e)　(f)

(g)　(h)　(i)

図 1.8: 試作 FPGA ボードによる重心計算

## 1.4　片側ラドン変換を用いた位置と姿勢の検出

本節では，片側ラドン変換を用いて，平面運動物体の位置と姿勢を検出するアルゴリズムを紹介する．さらに，片側ラドン変換の並列処理について述べる．

### 1.4.1　片側ラドン変換とその性質

図 1.9 に示すように，グレースケール画像に座標系 $O-xy$ を設定する．点 $(x,y)$ における画素値を $g(x,y)$ で表す．図 1.3-(a) に示すように，原点からの距離 $\rho$，$x$ 軸と成す角 $\theta$ の直線に沿って，画素値 $g(x,y)$ を線積分する．この積分を，ラドン変換とよぶ．ラドン変換においては，直線群に沿う線積分を計算する．原点回りに直線群を角度 $\pi$ 回転させると，もとの直線群に一致する．ラドン変換がこのような性質を有するため，ラドン変換を用いるアルゴリズムでは，角度 $\alpha$ の回転と角度 $\alpha+\pi$ の回転を区別することが困難である．そこで，角度 $\alpha$ の回転と角度 $\alpha+\pi$ の回転を区別するために，積分路として直線の代わりに半直線を導入する．すなわち，図 1.3-(b) に示すように，原点からの距離 $\rho$，$x$ 軸と成す角 $\theta$ の半直線に沿って，画素値 $g(x,y)$ を線積分する．この積分を，片側ラドン変換とよび，次式で表される．

$$U[g](\rho,\theta)=\int_0^{\infty} g(\xi\cos\theta-\rho\sin\theta,\xi\sin\theta+\rho\cos\theta)d\xi. \tag{1.3}$$

図 1.11-(a) に示す画像のラドン変換と片側ラドン変換を，それぞれ図

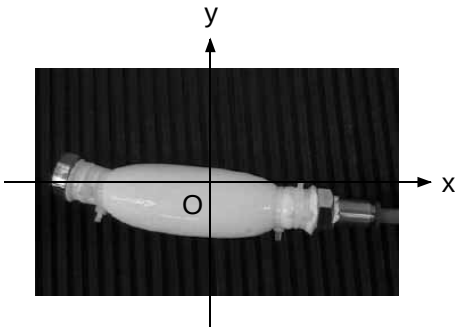
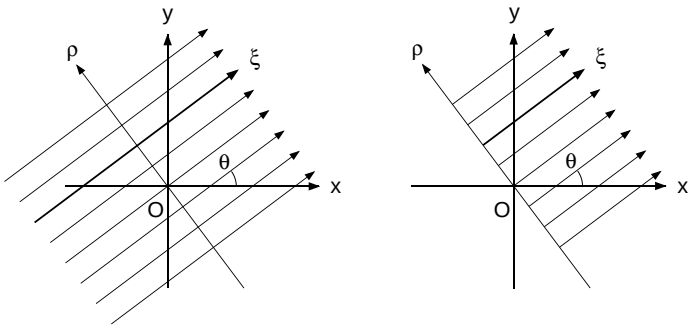



図 1.9: 画像座標系




(a)　(b)

図 1.10: ラドン変換と片側ラドン変換の積分路

1.11-(b),(c) に示す．ラドン変換は，変数 $\theta$ に関して周期 $\pi$ の関数であるのに対し，片側ラドン変換は，変数 $\theta$ に関して周期 $2\pi$ の関数である．したがって，片側ラドン変換を用いることにより，物体の姿勢を周期 $2\pi$ で計測することができる．

図 1.12-(a)，(b) に示すように，原画像を $g_0$，原画像を原点回りに角度 $\alpha$ 回転させて得られる画像を $g_1$ とする．画像 $g_0$ の片側ラドン変換 $U_0(\rho,\theta)$ と $g_1$ の片側ラドン変換 $U_1(\rho,\theta)$ は，次式を満たす．

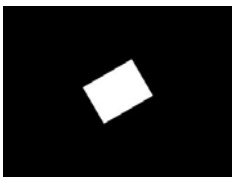
$$U_0(\rho,\theta) \equiv U_1(\rho,\theta+\alpha), \quad \forall\rho,\theta. \tag{1.4}$$

図 1.12-(c) に示すように，原画像を角度 $\beta$ の方向に距離 $d_0$ 並進移動させて得られる画像を $g_2$，その片側ラドン変換を $U_2(\rho,\theta)$ とする．原画像に写る物体の内部にあるように，座標原点を選ぶと，

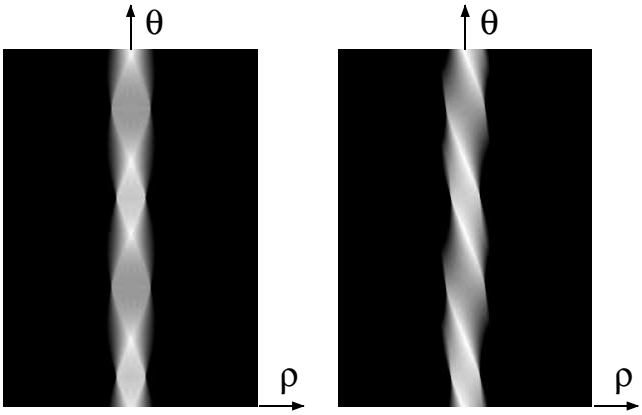
$$\begin{aligned} U_0(\rho,\theta) &\equiv U_2(\rho - d_0\sin(\theta-\beta),\theta), \\ \forall\rho, &\quad \theta+\frac{\pi}{2}=\beta. \end{aligned} \tag{1.5}$$

図 1.12-(d) に示すように，原画像に回転角 $\alpha$，移動方向 $\beta$，移動距離 $d_0$ で与えられる平面運動を与えた画像を $g_3$，その片側ラドン変換を $U_3(\rho,\theta)$ とする．このとき，

$$\begin{aligned} U_0(\rho,\theta-\alpha) &\equiv U_3(\rho - d_0\sin(\theta-\beta),\theta), \\ \forall\rho, &\quad \theta+\frac{\pi}{2}=\beta. \end{aligned} \tag{1.6}$$

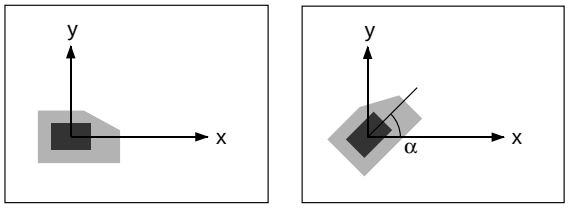


(a) image

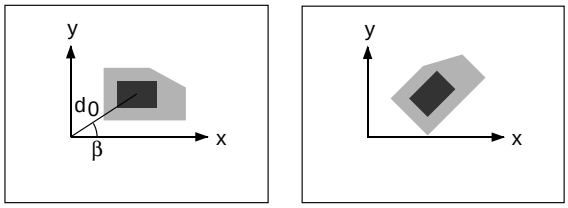



(b) Radon　(c) one-sided Radon

図 1.11: ラドン変換と片側ラドン変換の例




(a) original image $g_0$　(b) rotation $g_1$




(c) translation $g_2$　(d) planar motion $g_3$

図 1.12: 剛体の平面運動

片側ラドン変換 $U(\rho,\theta)$ の $\rho$ に沿うパワースペクトルを $\mathcal{F}(f,\theta)$ で表す．ラドン変換のパワースペクトルは周期 $\pi$ の周期関数である．一方，片側ラドン変換 $U(\rho,\theta)$ と $U(\rho,\theta+\pi)$ は一般に異なるので，片側ラドン変換のパワースペクトルは周期 $2\pi$ の周期関数である．片側ラドン変換 $U_0(\rho,\theta)$ と $U_3(\rho,\theta)$ のパワースペクトルをそれぞれ $\mathcal{F}_0(f,\theta)$，$\mathcal{F}_3(f,\theta)$ とすると，次式が成り立つ．

$$\mathcal{F}_0(f,\theta-\alpha) \equiv \mathcal{F}_3(f,\theta) \quad \forall f, \quad \theta+\frac{\pi}{2}=\beta. \tag{1.7}$$

片側ラドン変換の性質を利用して，平面運動物体の位置と姿勢を検出するアルゴリズムを構成する．参照画像と入力画像を比較して，運動を検出する．参照画像を $g_{sample}$，入力画像を $g_{input}$ で表す．それらの片側ラドン変換をそれぞれ $U_{sample}(\rho,\theta)$，$U_{input}(\rho,\theta)$ で表す．片側ラドン変換のパワースペクトルをそれぞれ $\mathcal{F}_{sample}(f,\theta)$，$\mathcal{F}_{input}(f,\theta)$ で表す．まず，パワースペクトル $\mathcal{F}_{sample}(f,\theta)$ と $\mathcal{F}_{input}(f,\theta)$ を比較し，回転角 $\alpha$ と並進移動方向 $\beta$ を算出する．次に，片側ラドン変換 $U_{sample}(\rho,\theta)$ と $U_{input}(\rho,\theta)$ を比較し，並進移動距離 $d_0$ を求める．以上のアルゴリズムは，次のようにまとめられる．

**Full Algorithm**

**Step 1** Compute $U_{sample}(\rho,\theta)$ and $U_{input}(\rho,\theta)$.

**Step 2** Compute $\mathcal{F}_{sample}(f,\theta)$ and $\mathcal{F}_{input}(f,\theta)$.

**Step 3** Find $\theta_{sample}$ and $\theta_{input}$ that minimize
$\| \mathcal{F}_{sample}(f,\theta_{sample}) - \mathcal{F}_{input}(f,\theta_{input}) \|$.

**Step 4** $\alpha = \theta_{input} - \theta_{sample}$
$\beta = \theta_{input} + \pi/2$

**Step 5** Find $d_0$ that satisfies
$U_{sample}(\rho,\theta_{sample}) \equiv U_{input}(\rho+d_0,\theta_{input}) \quad \forall\rho.$

片側ラドン変換のパワースペクトルは周期 $2\pi$ であるので，パワースペクトルの比較により，回転角を求めることができる．ただし，このアルゴリ

ズムは，パワースペクトルの二次元マッチングを含んでおり，多くの計算時間を要する．

物体と背景は分離できると仮定し，多くの計算時間を要する上述のアルゴリズムを簡略化する．まず，物体の位置を，画像重心から求める．画像重心まわりの $\rho = 0$ に対応する片側ラドン変換を求める．物体の姿勢は，$U_{sample}(0, \theta)$ と $U_{input}(0, \theta)$ のマッチングにより求めることができる．以上のアルゴリズムは，次のようにまとめられる．

**Simplified Algorithm**

**Step 1** Compute gravity centers $G_{sample}$ and $G_{input}$.

**Step 2** Compute $U_{sample}(0, \theta)$ and $U_{input}(0, \theta)$
around individual gravity centers.

**Step 3** Find $\alpha$ that satisfies
$U_{sample}(0, \theta - \alpha) \equiv U_{input}(0, \theta) \quad \forall\theta$.

片側ラドン変換 $U(0, \theta)$ は周期 $2\pi$ であるので，変換を比較することにより，回転角を求めることができる．このアルゴリズムを簡略版アルゴリズムとよぶ．片側ラドン変換のマッチングは 1 次元であり，計算時間は短い．

### 1.4.2　片側ラドン変換の並列処理

本節では，片側ラドン変換の並列計算について考察する．グレースケール画像が CCD カメラにより撮影され，NTSC 形式で処理部に送られるとする．画像の幅を $W$，高さを $H$ で表す．グレースケール画像は，格子点 $P_{i,j}$ $(i = 0, 1, \cdots, W-1, j = 0, 1, \cdots, H-1)$ における画素値 $g_{i,j}$ により与えられる．オリジナルのアルゴリズムでは，画素値 $g(x, y)$ の値を，周囲の格子点における画素値を補間することにより求める．すなわち，

$$\begin{aligned} g(x,y) = (1-r_x)(1-r_y)g_{i,j} + r_x(1-r_y)g_{i+1,j} \\ +(1-r_x)r_y g_{i,j+1} + r_x r_y g_{i+1,j+1} \end{aligned}$$

ここで $i = [x/W]$，$j = [y/H]$，$r_x = (x - iW)/W$，$r_y = (y - jH)/H$ である．(1.3) 式で与えられる片側ラドン変換の計算においては，格子点へのアクセスに規則性がない．したがって，格子点の画素値をメモリに保存しなければならない上に，メモリへのランダムアクセスが必要である．これは，計算時間とメモリ領域の増加を引き起こす．

計算時間とメモリ領域を減らすために，片側ラドン変換の並列計算を構成する．まず，格子点へのアクセス順序を変更する．格子点における画素値は，処理部に一つずつ順に送られる．そこで，ハフ変換における投票の概念を導入する．格子点 $P_{i,j}$ の座標を $(x_{i,j}, y_{i,j})$ で表す．角度の範囲 $[0, 2\pi]$ を $K$ 分割し，$\Delta\theta = 2\pi/K$，$\theta_k = k\Delta\theta$ $(k = 0, 1, \cdots, K-1)$ とする．パラメータ $\rho$ の間隔を $\Delta\rho$ で表し，$\rho_h = h\Delta\rho$ $(h = 0, \pm 1, \pm 2, \cdots)$ とする．片側ラドン変換 $U(\rho, \theta)$ を，$\theta = \theta_k$，$\rho = \rho_h$ のみで，離散的に求める．積分路を定めるパラメータ $\rho, \theta, \xi$ を用いて，座標 $x, y$ は次式のように表される．

$$\begin{aligned} x &= \xi\cos\theta - \rho\sin\theta, \\ y &= \xi\sin\theta + \rho\cos\theta. \end{aligned}$$

ただし，$\xi$ の値は正または 0 でなくてはならない．上式より

$$\begin{aligned} \xi &= x\cos\theta + y\sin\theta, \\ \rho &= -x\sin\theta + y\cos\theta. \end{aligned}$$

表 1.2: 片側ラドン変換における投票

initialize array $U(\theta_k, \rho_h)$
for $(i,j) = (0,0), (0,1), \cdots, (W-1, H-1)$ do
　compute $x_{i,j}$ and $y_{i,j}$
　for $k = 0, 1, \cdots, K-1$ do
　　$\theta_k = k\Delta\theta$, $C_k = \cos\theta_k$, $S_k = \sin\theta_k$
　　$\xi = x_{i,j}C_k + y_{i,j}S_k$
　　$\rho = -x_{i,j}S_k + y_{i,j}C_k$
　　if $\xi \geq 0$ then
　　　$h = \rho/\Delta\rho$
　　　increase $U(\theta_k, \rho_h)$ by pixel value $g_{i,j}$
　　end
　end
end

座標 $x, y$ と角度 $\theta$ の値が与えられると，上式により $\xi$ と $\rho$ の値を計算することができる．計算した $\xi$ の値が正または 0 である場合，画素値 $g(x,y)$ の値が積分値 $U(\rho,\theta)$ に加算されている．したがって，$\xi$ の値が正または 0 であるならば，積分値 $U(\rho,\theta)$ を画素値 $g(x,y)$ の値だけ増加させればよい．結局，片側ラドン変換における投票は，Table 1.2 に示すようにまとめられる．

Table 1.2 に示す投票の過程では，格子点の画素値は一つずつ順番にアクセスされる．CCD カメラは，格子点の画素値を一つずつ処理部に送る．したがって，各々の画素値が順番に処理部に送られている限り，画素値をメモリ上に記憶させる必要はなく，Table 1.2 に示す投票により，片側ラドン変換を計算することができる．

また，各々の角度に関して並列に，投票を実行することができる．角度 $\theta_k$ に関する投票においては，配列 $U(\rho,\theta)$ は，$\theta = \theta_k$ のおいてのみアクセスされる可能性がある．すなわち，配列 $U(\rho,\theta)$ へのアクセスにおい

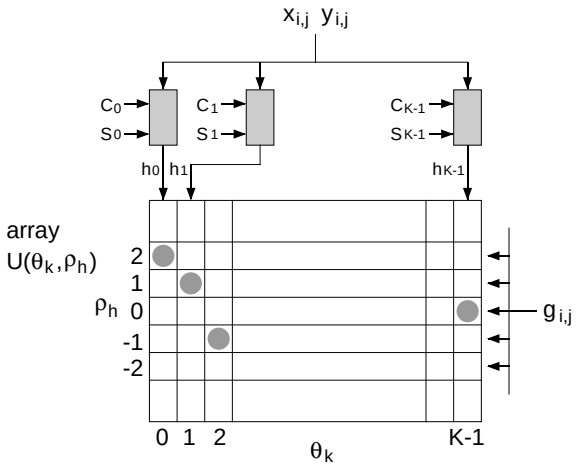



図 1.13: 片側ラドン変換における並列投票

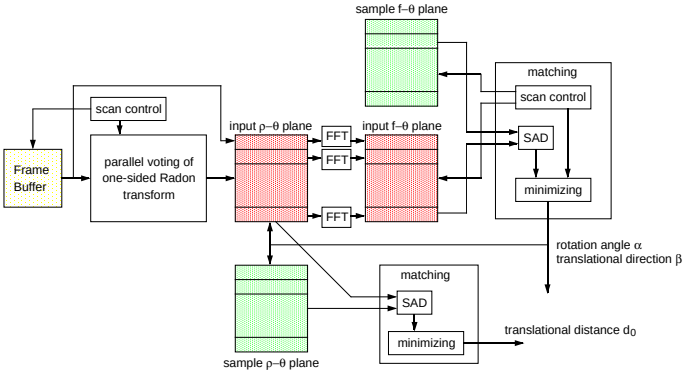



図 1.14: 並列アルゴリズムにおけるデータの流れ

ては，異なる角度に関する投票はたがいに干渉しない．座標 $x_{i,j}, y_{i,j}$ と $C_k = \cos\theta_k$，$S_k = \sin\theta_k$ から，角度 $\theta_k$ に対応する添字 $h$ を並列に計算することができる．画素値 $g_{i,j}$ を増やす操作も，角度 $\theta_0, \theta_1, \cdots, \theta_{K-1}$ に対して並列に実行することができる．したがって，図 1.13 に示すように，投票を並列に実行することができる．

並列アルゴリズムにおけるデータの流れを，図 1.14 に示す．パワースペクトルの二次元マッチングにより回転角と移動方向を，片側ラドン変換の一次元マッチングにより移動距離を算出する．

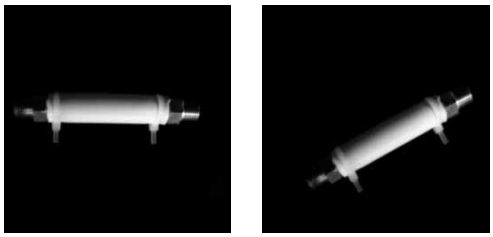
### 1.4.3　並列アルゴリズムの評価

本節では，オリジナルアルゴリズムと比較することにより，並列アルゴリズムを評価する．二つのアルゴリズムを，CPU が Pentium III 800 MHz，メモリが 256MB の PC 上に実装した．PC の OS は Vine Linux である．プログラムは C で記述し，GCC ver.2.7.2.3 でコンパイルした．図 1.15-(a)，(b) に参照画像と入力画像の例を示す．画像は 8 [bit] グレースケール画像である．サイズは 256 [pixel] × 256[pixel] であり，画像の中央を座標原点とする．図 1.15-(c) に，それぞれの画像に対してオリジナルアルゴリズムで計算した片側ラドン変換を示す．図 1.15-(d) に，それぞれの画像に対して並列アルゴリズムで計算した片側ラドン変換を示す．パラメータ $\theta$ の間隔を $\Delta\theta = 1$ [degree] とする．パラメータ $\rho$ の範囲を $[-125, 125]$ [pixel] とし，その間隔を $\Delta\rho = 1$ [pixel] とする．図 1.15-(c) と (d) を比較すると，並列アルゴリズムは片側ラドン変換を正確に計算していることがわかる．

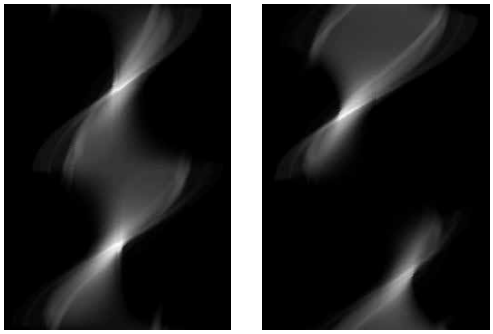
物体の位置と姿勢を算出するために，片側ラドン変換のパワースペクトルを計算する．この例では，周波数 $f_h = h/125$ $(h = -125, -124, \cdots, 125)$ におけるパワースペクトルを求めている．周波数 $f_0$ に近い低周波の領域では，パワースペクトルが安定に計算できるのに対し，高周波の領域ではノイズが多く，数値的に不安定である．そこで，パワースペクトルのマッチングにおいては，次式のノルムを用いる．

$$\begin{aligned} &\parallel \mathcal{F}_{sample}(f_h, \theta_{sample}) - \mathcal{F}_{input}(f_h, \theta_{input}) \parallel \\ &= \sum_{h=0}^{40} |\mathcal{F}_{sample}(f_h, \theta_{sample}) - \mathcal{F}_{input}(f_h, \theta_{sample})|. \end{aligned}$$

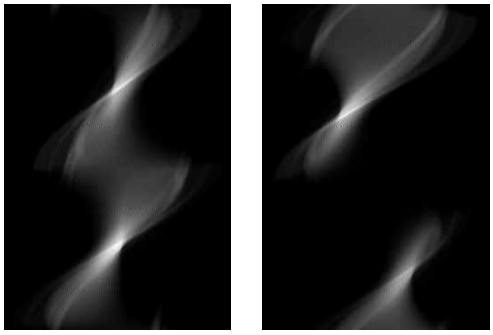
図 1.15 は，参照画像に対応するパワースペクトルと入力に対応するパワースペクトルのノルムを表す．図 1.15-(a) においては，オリジナルアルゴリズムによる計算結果を用い，図 1.15-(b) においては並列アルゴリズムによる計算結果を用いている．ノルム $\parallel \mathcal{F}_{sample}(f_h, \theta_{sample}) - \mathcal{F}_{input}(f_h, \theta_{input}) \parallel$ を，$\theta_{sample} = 0, 1, \cdots, 359$ [degree]，$\theta_{input} = 0, 1, \cdots, 359$ [degree] に対して計算する．図は，得られた $360 \times 360$ 個の計算結果を一列に並べて得られるグラフである．図 1.16-(a) より，ノルムは $\theta_{sample} = 292$ [degree]，$\theta_{input} = 322$ [degree] において最小値を取ることがわかる．したがって，



(a) sample image　(b) input image

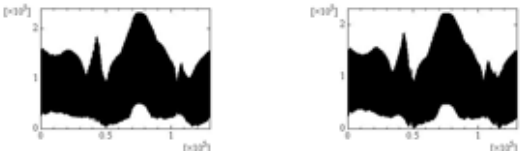


(c) transforms by original algorithm



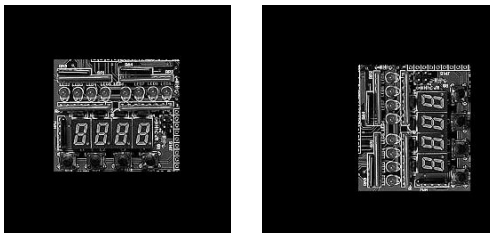
(d) transforms by parallel algorithm

図 1.15: チューブ画像に対する片側ラドン変換の計算結果

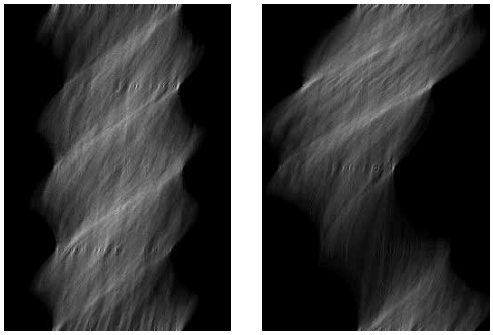


(a) original algorithm　(b) parallel algorithm

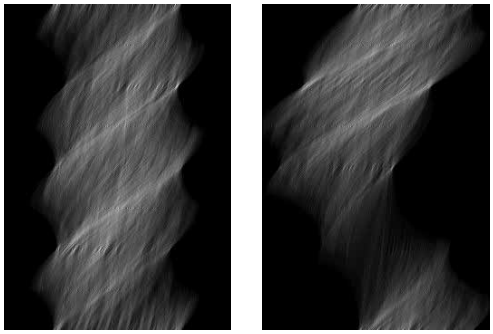
図 1.16: チューブ画像に対応するパワースペクトルのマッチング



(a) sample image　(b) input image

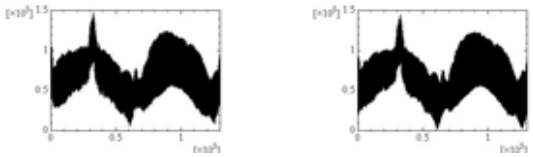


(c) transforms by original algorithm



(d) transforms by parallel algorithm

図 1.17: ボード画像に対する片側ラドン変換の計算結果



(a) original algorithm　(b) parallel algorithm

図 1.18: ボード画像に対応するパワースペクトルのマッチング

図 1.19: エアーテーブル上のサンプル物体

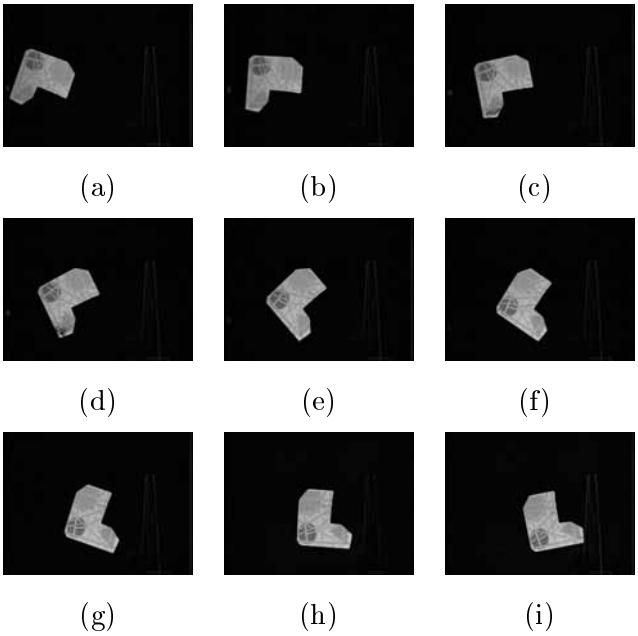
$\alpha = 30$ [degree] ならびに $\beta = 52$ [degree] である．さらに $d_0 = 40$ [pixel] が得られる．図 1.16-(b) より，ノルムは $\theta_{sample} = 113$ [degree]，$\theta_{input} = 143$ [degree] において最小値を取ることがわかる．したがって，$\alpha = 30$ [degree] ならびに $\beta = 233$ [degree] である．さらに $d_0 = -39$ [pixel] が得られる．結局，並列アルゴリズムにより計算された物体の位置と姿勢は，オリジナルアルゴリズムによる計算結果と一致していることがわかる．

図 1.17-(a)，(b) に別の例を示す．画像は 8 [bit] グレースケール画像であり，サイズは 256 [pixel] × 256 [pixel] である．オリジナルアルゴリズムで計算した片側ラドン変換を図 1.17-(c) に，並列アルゴリズムで計算した片側ラドン変換を図 1.17-(d) に示す．図 1.17-(c) と (d) を比較すると，並列アルゴリズムは片側ラドン変換を正確に計算していることがわかる．図 1.17-(c) に示す片側ラドン変換に対して計算されたパワースペクトルのノルムを，図 1.18-(a) に示す．図より，$\theta_{sample} = 258$ [degree]，$\theta_{input} = 349$ [degree] においてノルムが最小になることがわかる．したがって，$\alpha = 91$ [degree] ならびに $\beta = 79$ [degree] である．さらに，$d_0 = 49$ [pixel] が得られる．図 1.17-(d) に示す片側ラドン変換に対して計算されたパワースペクトルのノルムを，図 1.18-(b) に示す．図より，$\theta_{sample} = 259$ [degree]，$\theta_{input} = 349$ [degree] においてノルムが最小になることがわかる．したがって，$\alpha = 90$ [degree] ならびに $\beta = 79$ [degree] である．さらに，$d_0 = 49$ [pixel] が得られる．この例においても，並列アルゴリズムにより計算された物体の位置と姿勢は，オリジナルアルゴリズムによる計算結果と一致していることがわかる．

次に，実環境においてオリジナルアルゴリズムと並列アルゴリズムを評価する．図 1.19 にエアーテーブル上の物体の例を示す．画像の幅は 320 [pixel]，高さは 240 [pixel] である．画像の中央を座標原点とする．パラメータ $\theta$ の間隔を $\Delta\theta = 1$ [degree] とする．パラメータ $\rho$ の範囲を $[-256, 256]$

表 1.3: 回転角度の計算結果

| | original | | parallel | |
|---|---|---|---|---|
| actual | computed | error | computed | error |
| 0° | −1° | −1° | −2° | −2° |
| 30° | 30° | 0° | 30° | 0° |
| 60° | 64° | 4° | 64° | 4° |
| 90° | 93° | 3° | 93° | 3° |
| 120° | 122° | 2° | 119° | −1° |
| 150° | 148° | −2° | 149° | −1° |
| 180° | 181° | 1° | 180° | 0° |
| 210° | 219° | 9° | 218° | 8° |
| 240° | 238° | −2° | 245° | 5° |
| 270° | 266° | −4° | 277° | 7° |
| 300° | 294° | −6° | 299° | −1° |
| 330° | 324° | −6° | 326° | −4° |



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

図 1.20: エアーテーブル上を動く物体

[pixel] とし，その間隔を $\Delta\rho = 2$ [pixel] とする．CCD カメラにより得られる 8 [bit] グレースケール画像を $g_{camera}$，あらかじめ撮影された背景画像を $g_{back}$ とする．入力画像 $g_{input}$ を，以下に示す手続きで求める．まず，カメラ画像と背景画像の絶対差分を計算する．すなわち，

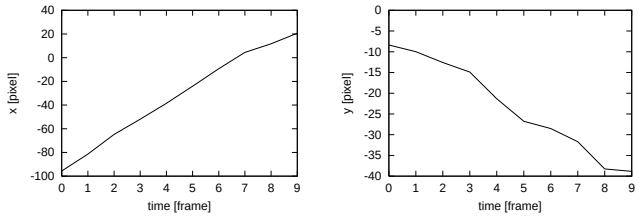
$$g_{ad}(x,y) = \mid g_{camera}(x,y) - g_{back}(x,y) \mid .$$

入力画像 $g_{input}$ は，絶対差分画像 $g_{ad}$ より次式にしたがって計算される．

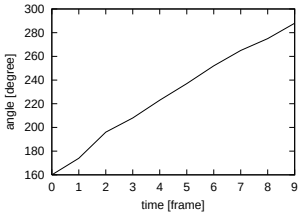
$$g_{input}(x,y) = \begin{cases} g_{ad}(x,y) & g_{af}(x,y) \geq 90 \\ 0 & otherwise \end{cases} .$$

姿勢の計算結果の精度を評価する．図 1.19 に示す物体を，テーブル上に正確な姿勢で置く．実際の姿勢角度と二つのアルゴリズムにより計算される姿勢角度を，Table 1.3 に示す．Table 1.3 より，並列アルゴリズムにより得られる姿勢角度は，オリジナルアルゴリズムにより得られる角度とほぼ一致していることがわかる．ただし，絶対誤差は 10 [degree] 近くに達する．

図 1.20-(a)〜(h) に，エアーテーブル上の物体の運動を示す．これらの画像はビデオフレームレートで取得している．物体とテーブル間の摩擦は無視できるので，物体はテーブル上を等速度・等角速度で運動する．これらの画像から，オリジナルアルゴリズムで計算した物体の位置と姿勢を図 1.21 に，並列アルゴリズムで計算した位置と姿勢を図 1.22 に示す．並列アルゴリズムによる計算結果は，オリジナルアルゴリズムによる計算結果とほぼ一致している．さらに，物体の速度と角速度は，ほぼ一定であることがわかる．したがって，並列アルゴリズムは，オリジナルアルゴリズムと同程度に物体の平面運動を検出できることがわかる．

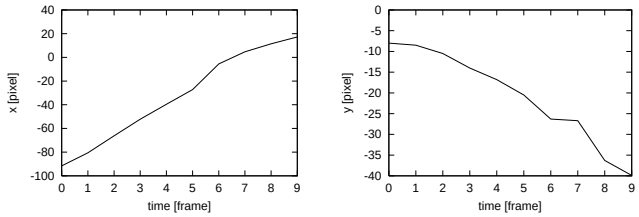



(a) position

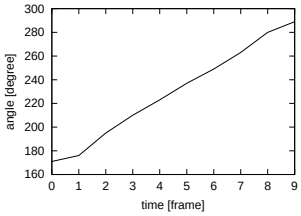



(b) orientation

図 1.21: オリジナルアルゴリズムで計算した位置と姿勢




(a) position




(b) orientation

図 1.22: 並列アルゴリズムで計算した位置と姿勢

## 1.5　ビジョンアルゴリズムの FPGA への実装

### 1.5.1　論理回路設計言語の選択

本プロジェクトでは，FPGA の回路設計に SystemCompiler(C Level Design) を用いる．SystemCompiler の最大の特徴は，C/C++言語ベースで論理回路設計が可能な点にある．C/C++言語をベースに論理回路設計を行う設計ツールは，二通りのアプローチに分けられる．一つは，C/C++言語に，並列演算構文や柔軟なビット操作など，VLSI 設計に必要な機能を言語仕様に追加するタイプである．もう一つは，C/C++言語の言語仕様はそのままにしておき，書式のフォーマットを制限するというタイプである．前者の，言語仕様追加タイプは，記述が理解しやすく簡潔に書けるというメリットがある．一方，追加された仕様のため，通常のコンパイラではコンパイルできなくなるというデメリットがある．後者の書式制限タイプは，通常のコンパイラでもコンパイルできるというメリットがある．デメリットとしては，ビット操作や並列動作構造などの記述を，通常の C/C++言語の仕様内で記述するために，記述が複雑になり判りにくくなるという点が挙げられる．

本プロジェクトでは，書式制限タイプであり，通常のコンパイラでもコンパイルできるという特徴と，直接回路データを出力するのではなく HDL を出力するため，特定のデバイスやツールに依存しないという特徴を評価して，SystemCompiler を選択する．通常のコンパイラでもコンパイルできるという特徴は，今回のように画像処理 VLSI を設計する場合には重要である．画像処理 VLSI は，扱う対象のデータ量が大きいため，元々あまり速くない RTL レベルでのシミュレーションでは，シミュレーションに多大の時間を要する．SystemCompiler では，CycleC と呼ばれる C/C++言語のサブセット的な書式で記述したソースファイルを HDL に変換することができる．CycleC で記述したソースは，通常のコンパイラでもコンパイル可能で，HDL と同じ動作が保証されるため，高速にシミュレーションを行うことができる．また，テスト用の映像データ・画像データの準備にも，PC の豊富なリソースを利用することができる．ディスク中の静止画像ファイル，ムービーファイルや，ビデオキャプチャーカード等のリソースも利用可能となる．

### 1.5.2　ロジックレベル記述

SystemCompiler を用いて，PC 上で動作するシミュレーションプログラムを製作する．このシミュレーションプログラムは，アルゴリズムレベルより細かいロジックレベルまで，ビデオフレームレート画像処理 VLSI と同様に動作する．以下，各部位について説明する．以下の説明では，クラス，モジュールという言葉が出てくる．クラスは，SystemCompiler の CycleC でのクラスを指し，C++言語でのクラスと互換性がある．モジュールは，回路構造での機能単位の事である．VerilogHDL のモジュールに相当する．CycleC では，クラスを使ってモジュールを記述する．クラスインスタンスが，そのままモジュールインスタンスとなる．

**映像入力**　映像入力部は，ビデオフレームレート画像処理 VLSI の外の部分であり，実際のハードウェアでは，MU200-VDEC というアナログ NTSC/PAL・デジタル変換ボードである．MU200-VDEC の仕様を表 1.4 に，外観を図 1.23 に示す．

図 1.23: MU200-VDEC の外観

シミュレーションでは，ディスク上の静止画像ファイル (BMP,PPM,PGM) あるいはムービーファイル (AVI) を読み込み，ソフトウェアで生成した垂

表 1.4: MU200-VDEC の仕様

| | | |
|---|---|---|
| メーカー | | 三菱電機マイコン機器ソフトウェア株式会社 |
| 製品名 | | 電子回路設計・ラピッドプロトタイピングキット<br>PowerMedusa シリーズ<br>NTSC/PAL ビデオデコードコンポーネント |
| 型番 | | MU200-VDEC |
| 仕様 | 入力端子 | RCA ピンジャック (NTSC/PAL コンポジット信号)<br>S 端子 (MiniDin4,NTSC/PAL Y/C 信号) |
| | 出力端子 | 240 ピンコネクタ (メス)<br>DS01R-240R(ケル株式会社製レセプタクル コネクタ) |
| | 電源 | DC+5V (最大約 1.4A) |
| | 最大外形寸法 | 165×150×17mm |
| | 入力ビデオ信号 | NTSC コンポジット信号 (アナログ 75Ω 終端 1Vp-p)<br>NTSC Y/C 信号 (アナログ 75Ω 終端 1Vp-p)<br>PAL コンポジット信号 (アナログ 75Ω 終端 1Vp-p)<br>PAL Y/C 信号 (アナログ 75Ω 終端 1Vp-p) |
| | 出力ビデオ信号 | 24 ビット RGB 信号 (デジタル TTL レベル)<br>16 ビット RGB 信号 (デジタル TTL レベル)<br>15 ビット RGB 信号 (デジタル TTL レベル)<br>24 ビット YCrCb 信号 (デジタル TTL レベル)<br>16 ビット YCrCb 信号 (デジタル TTL レベル) |
| | ビデオデコード用 IC | Brooktree 社 Bt812 (A/D 変換,Y/C 分離,<br>色復調, 同期分離,PLL, 各種制御信号発生) |
| | パラメータ設定用 IC | ALTERA 社 EPF81500AQC240-4<br>ALTERA 社 EPC1PC8 |

直同期信号，水平同期信号等と組み合わせてシミュレーション回路本体に送り出し，CCD カメラと MU200-VDEC をシミュレートする．各種同期信号を，下記のコードで生成する．

```
typedef struct {
    UBYTE    dot_clock;        // ドットクロック
    UBYTE    vertical_sync;    // 垂直同期信号
    UBYTE    horizontal_sync;  // 水平同期信号
    UBYTE    composite_sync;   // コンポジット同期信号
    UBYTE    x_field_even;     // 偶数・奇数フレーム識別信号
    UBYTE    active;           // アクティブ
} SYNC_SIGNAL;

const SYNC_SIGNAL* generate_sync_signal(void){
    static SYNC_SIGNAL signal = {true,true,true,true,true,false};
    static DWORD x_cnt = 0;    // 水平カウンタ
    static DWORD y_cnt = 0;    // 垂直カウンタ
    static bool EQ = false;
    static bool VS = false;

    if(x_cnt<779){
        x_cnt++;
    } else {
        x_cnt=0;
        // y_cnt
        if(y_cnt<525){
            y_cnt++;
        } else {
            y_cnt=0;
        }
        // 垂直同期信号,x_field_even,active
        switch(y_cnt){
        case   0:    signal.x_field_even = true;    break;
        case   3:    signal.vertical_sync = false;  break;
        case   6:    signal.vertical_sync = true;   break;
        case  20:    signal.active = true;          break;
        case 260:    signal.active = false;         break;
        case 262:    signal.x_field_even = false;   break;
        case 265:    signal.vertical_sync = false;  break;
        case 268:    signal.vertical_sync = true;   break;
        case 282:    signal.active = true;          break;
        case 522:    signal.active = false;         break;
        }
    }
    // 水平同期信号,EQ,VS,
    switch(x_cnt){
    case   0:
        switch(y_cnt){
        case   1:    EQ = true;    break;
        case   4:    VS = true;    break;
```

```
        case   7:    VS = false;   break;
        case  10:    EQ = false;   break;
        } break;
    case  18:    signal.horizontal_sync = false;  break;
    case  76:    signal.horizontal_sync = true;   break;
    case 390:
        switch(y_cnt){
        case 263:    EQ = true;   break;
        case 266:    VS = true;   break;
        case 269:    VS = false;  break;
        case 272:    EQ = false;  break;
        } break;
    }
    // composite_sync
    if( (x_cnt==350) ||
        (x_cnt==740) ||
        (x_cnt== 76 && !EQ) ||
        ((x_cnt==47 || x_cnt==437) && EQ && !VS))
            signal.composite_sync = true;
    else
    if((x_cnt== 18) || (x_cnt==408 && EQ))
            signal.composite_sync = false;
    return &signal;
}
```

シミュレーション回路本体へ同期信号と画素データを送る部分を，下記に示す．シミュレーション回路は，8bpp グレイスケールで画像を受け取るため，RGB 信号から輝度を計算する．クラス CTOP が，シミュレーション回路本体のトップレベルモジュールである．関数 process_receive_signal 関数は，シミュレーション回路から出てきた同期信号と色信号から，PC のメモリ上に画像を構築する．実際のハードウェアでは，MU200-VENC というデジタル信号・NTSC/PAL 変換ボードに相当する．

```
static void process_send_image(CTOP* pTop,COLORREF c){
    const SYNC_SIGNAL* p = generate_sync_signal();

    UBYTE Y = ((UWORD)(0.299*256)*GetRValue(c) +
               (UWORD)(0.587*256)*GetGValue(c) +
               (UWORD)(0.114*256)*GetBValue(c) )>>8;

    static uint1  VENC_CLKx1_OUT, VENC_CLKx2_OUT;
    static uint8  VENC_R,VENC_G,VENC_B;
    static uint1  VENC_HD_OUT, VENC_VD_OUT, VENC_FE_OUT, VENC_ACT_OUT;
    static uint16 SRAM1_DATA,SRAM2_DATA;
    static uint18 SRAM1_ADRS,SRAM2_ADRS;
    static uint1  SRAM1_xCE, SRAM2_xCE;
    static uint1  SRAM1_xOE, SRAM2_xOE;
```

```
    static uint1  SRAM1_xWE, SRAM2_xWE;
    static uint1  SRAM1_BLE, SRAM2_BLE;
    static uint1  SRAM1_BHE, SRAM2_BHE;

    pTop->run(  p->dot_clock,
                1,
                Y,
                p->horizontal_sync,
                p->vertical_sync,
                p->x_field_even,
                p->active,

                VENC_CLKx1_OUT, VENC_CLKx2_OUT,
                VENC_R,    VENC_G,    VENC_B,
                VENC_HD_OUT,  VENC_VD_OUT,
                VENC_FE_OUT,  VENC_ACT_OUT,

                SRAM1_DATA,   SRAM2_DATA,
                SRAM1_ADRS,   SRAM2_ADRS,
                SRAM1_xCE,    SRAM2_xCE,
                SRAM1_xOE,    SRAM2_xOE,
                SRAM1_xWE,    SRAM2_xWE,
                SRAM1_BLE,    SRAM2_BLE,
                SRAM1_BHE,    SRAM2_BHE
            );

    process_receive_signal( VENC_CLKx1_OUT,
                            VENC_CLKx2_OUT,
                            VENC_R,
                            VENC_G,
                            VENC_B,
                            VENC_HD_OUT,
                            VENC_VD_OUT,
                            VENC_FE_OUT,
                            VENC_ACT_OUT);
    return;
}
```

ファイル等から取得した静止画像を 1 画素ずつ送り出す部分を，下記に示す．バッファCOLORREF* p には，640×480×32bpp RGB フルカラーの静止画像のデータが格納される．

```
void send_image(COLORREF* pReceivedBitmap, CTOP* pTop,
                const CImage* pImage){
    s_received_image_bitmap = pReceivedBitmap;
    COLORREF* p = pImage->GetBitmap();

    // 垂直ブランキング期間
    for(UWORD y=0;y<22;y++){
        for(UWORD x=0;x<780;x++){
```

```
            process_send_image(pTop,0x0000ff);
        }
    }
    // 奇数フレーム 1,3,5,7,...
    UDWORD pitch = pImage->GetWidth();
    COLORREF* p2 = p;
    for(;y<22+240;y++){
        // 水平ブランキング期間
        for(UWORD x=0;x<47;x++){
            process_send_image(pTop,0x0000ff);
        }
        // 映像送信
        for(;x<47+640;x++){
            process_send_image(pTop,*p2++);
        }
        // 水平ブランキング期間
        for(;x<780;x++){
            process_send_image(pTop,0x0000ff);
        }
        p2 += pitch;
    }
    // 垂直ブランキング期間
    for(;y<262+22;y++){
        for(UWORD x=0;x<780;x++){
            process_send_image(pTop,0x0000ff);
        }
    }
    // 偶数フレーム 2,4,6,8,...
    p2 = p + pitch;
    for(;y<262+22+240;y++){
        // 水平ブランキング期間
        for(UWORD x=0;x<47;x++){
            process_send_image(pTop,0x0000ff);
        }
        // 映像送信
        for(;x<47+640;x++){
            process_send_image(pTop,*p2++);
        }
        // 水平ブランキング期間
        for(;x<780;x++){
            process_send_image(pTop,0x0000ff);
        }
        p2 += pitch;
    }
    // 垂直ブランキング期間
    for(;y<525;y++){
        for(UWORD x=0;x<780;x++){
            process_send_image(pTop,0x0000ff);
        }
    }
    delete[] p;
```

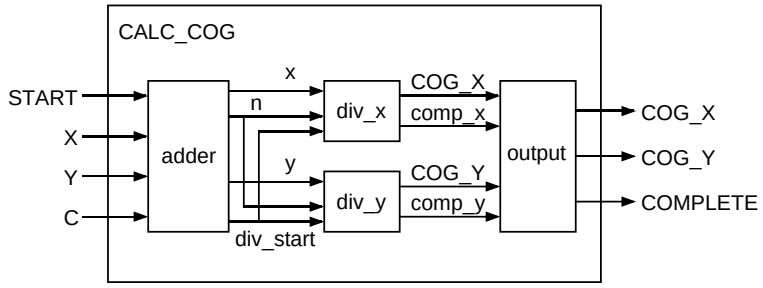



図 1.24: 2 値化重心計算モジュール

}

**2 値化重心計算**　2 値化重心計算を行うモジュールは，図 1.24 に示すように，4 つの主要なモジュール adder，div_x，div_y，output から構成される．2 値化重心モジュールには，開始信号 START，座標値 X と Y，画素情報 C が入力される．入力された画素値と座標の積は，モジュール adder で加算される．最後の画素まで加算が終了した後，モジュール div_x と div_y において，計算された積和値を画素値の合計で割って，重心を求める．モジュール output は，div_x と div_y における除算が終了するのを待って，外部に結果を出力する．モジュール adder を下記に示す．

```
void CCALC_COG_ADDER::run(  uint1    CLK,
                            uint1    xRST,
                            uint8    X,
                            uint8    Y,
                            uint8    C,
                            uint31&  SUM_X,
                            uint31&  SUM_Y,
                            uint24&  SUM_N,
                            uint1&   DIV_START
                          ){
    // xx yy nn state
    if(infer_clock(CLK) || infer_reset(!xRST) ){
        if(!xRST){
            xx = 0;
            yy = 0;
            nn = 0;
            state = 0;
        } else {
            uint8 cc = C & 0x80;
            if(state==2){
                state = 0;
```

```
            } else if(X==0 && Y==0){
                xx = 0;    yy = 0;    nn = cc;
                state=1;
            } else if(state==1){
                xx += uint31(cc)*uint31(X);
                yy += uint31(cc)*uint31(Y);
                nn += uint24(cc);
                if(X==255 && Y==255){
                    state = 2;
                }
            }
        }
    }

    SUM_X = xx;
    SUM_Y = yy;
    SUM_N = nn;
    DIV_START = state==2;
}
```

モジュール div_x，div_y は，同じ除算クラスの別インスタンスである．この除算クラスは，様々なビット幅の除算クラスを簡単に設定できるように，マクロを利用しており，下記のようになっている．モジュール div_x, div_y においては，被除数が 31 ビット，除数が 24 ビットである．

```
#define CDIVIDER(dividend_bit,divisor_bit) \
class CDIVIDER_##dividend_bit##divisor_bit { \
public: \
    CDIVIDER_##dividend_bit##divisor_bit(){ \
        tmp_QUOTIENT =0; \
        tmp_ODD      =0; \
        tmp_COMPLETE =0; \
 \
        count =0; \
        bufA  =0; \
        bufB  =0; \
        bufQ  =0; \
        tmpOdd=0; \
    } \
 \
    void run(   uint1                CLK, \
                uint1                xRST, \
                uint##dividend_bit   DIVIDEND, \
                uint##divisor_bit    DIVISOR, \
                uint##dividend_bit & QUOTIENT, \
                uint##divisor_bit &  ODD, \
                uint1                START, \
                uint1&               COMPLETE \
            ){ \
```

```
    if( infer_clock(CLK) || infer_reset(!xRST) ){ \
        if(!xRST){ \
            bufQ = 0; \
            tmpOdd = 0; \
        } else { \
            if(count!=( ##dividend_bit +1)){ \
                if(count){ \
                    uint##dividend_bit tmpA; \
                    set_slice(bufQ, ##dividend_bit -1,1, \
                              get_slice(bufQ, ##dividend_bit -2,0)); \
                    tmpA = bitvec_concat( \
                              get_slice(tmpOdd, ##dividend_bit -2,0), \
                              get_bit(bufA, ##dividend_bit -1), \
                              ##dividend_bit ,1); \
                    if(tmpA>=bufB){ \
                        set_bit(bufQ,0,1); \
                        tmpOdd = tmpA - bufB; \
                    } else { \
                        set_bit(bufQ,0,0); \
                        tmpOdd = tmpA; \
                    } \
                } else if(START){ \
                    set_slice(tmpOdd,##dividend_bit -2,0,0); \
                } \
            } \
        } \
    } \
\
    if( infer_clock(CLK) || infer_reset(!xRST) ){ \
        if(!xRST)            bufA = 0; \
        else \
        if(count!=( ##dividend_bit +1)){ \
            if(count){ \
                set_slice(bufA, ##dividend_bit -1,1, \
                get_slice(bufA, ##dividend_bit -2,0)); \
            } else if(START) bufA = DIVIDEND; \
        } \
    } \
\
    if( infer_clock(CLK) || infer_reset(!xRST) ){ \
        if(!xRST) bufB = 0; \
        else if(START) bufB = DIVISOR; \
    } \
\
    if( infer_clock(CLK) || infer_reset(!xRST) ){ \
        if(!xRST) tmp_QUOTIENT = 0; \
        else \
        if(count==( ##dividend_bit +1)) tmp_QUOTIENT = bufQ; \
    } \
\
    if( infer_clock(CLK) || infer_reset(!xRST) ){ \
```

```
        if(!xRST) tmp_ODD = 0; \
        else \
        if(count==( ##dividend_bit +1)){ \
            tmp_ODD = get_slice(tmpOdd, ##divisor_bit -1,0); \
        } \
    } \
 \
    if( infer_clock(CLK) || infer_reset(!xRST) ){ \
        if(!xRST) tmp_COMPLETE = 0; \
        else if(count==( ##dividend_bit +1)){ \
            tmp_COMPLETE = 1; \
        } \
        else if((!(count)) && START) tmp_COMPLETE = 0; \
    } \
 \
    if( infer_clock(CLK) || infer_reset(!xRST) ){ \
        if(!xRST)      count = 0; \
        else if(count==( ##dividend_bit +1)) count = 0; \
        else if(count) count = count+1; \
        else if(START) count = 1; \
    } \
 \
    QUOTIENT = tmp_QUOTIENT; \
    ODD      = tmp_ODD; \
    COMPLETE = tmp_COMPLETE; \
}; \
 \
private: \
    uint##dividend_bit tmp_QUOTIENT; \
    uint##divisor_bit  tmp_ODD; \
    uint1              tmp_COMPLETE; \
 \
    uint##divisor_bit  count; \
    uint##dividend_bit bufA; \
    uint##dividend_bit bufB; \
    uint##dividend_bit bufQ; \
    uint##dividend_bit tmpOdd; \
}
```

モジュール output を下記に示す．モジュール output は，div_x と div_y の両方の除算終了を待つ．ただし，現在の構成では，二つの除算モジュールは同じクロック数で演算終了の信号を出すので，特に必要ではない．

```
void CCALC_COG_OUTPUT::run( uint31 QUO_X,
                            uint31 QUO_Y,
                            uint1  DIV_COMP_X,
                            uint1  DIV_COMP_Y,
                            uint8& COG_X_O,
                            uint8& COG_Y_O,
                            uint1& COG_COMP
```

```
                        ){
    COG_COMP = DIV_COMP_X && DIV_COMP_Y;
    COG_X_O  = QUO_X;
    COG_Y_O  = QUO_Y;
}
```

2 値化重心計算モジュールのトップレベルモジュールを下記に示す．4 つのモジュールのインスタンスは，トップレベルモジュールのメンバとして定義されている．トップレベルモジュールは，論理回路は含まずに，各モジュール間の接続だけを記述する．

```
void CCALC_COG::run(    uint1   CLK,
                        uint1   xRST,
                        uint8   X,
                        uint8   Y,
                        uint8   C,
                        uint1&  COMPLETE,
                        uint8&  COG_X,
                        uint8&  COG_Y
                    ){
    adder.run(  CLK,
                xRST,
                X,
                Y,
                C,
                cc_out(x),
                cc_out(y),
                cc_out(n),
                cc_out(div_start)
            );

    divider_x.run(  CLK,
                    xRST,
                    cc_in(x),
                    cc_in(n),
                    cc_out(cog_x_w),
                    cc_out(cog_x_odd),
                    cc_in(div_start),
                    cc_out(div_comp_x)
                );

    divider_y.run(  CLK,
                    xRST,
                    cc_in(y),
                    cc_in(n),
                    cc_out(cog_y_w),
                    cc_out(cog_y_odd),
                    cc_in(div_start),
                    cc_out(div_comp_y)
```

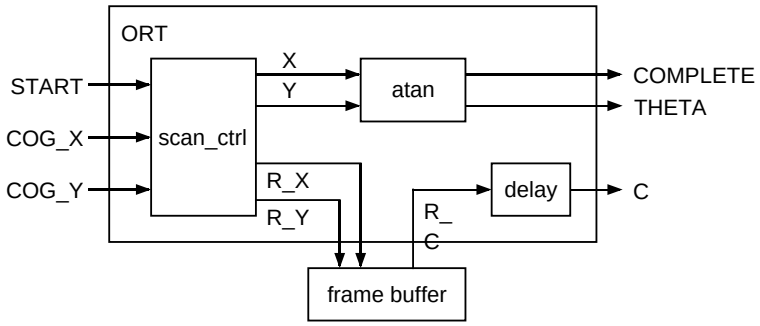



図 1.25: 片側ラドン変換モジュール

```
                );

    output.run( cc_in(cog_x_w),
                cc_in(cog_y_w),
                cc_in(div_comp_x),
                cc_in(div_comp_y),
                COG_X,
                COG_Y,
                COMPLETE
              );
}
```

**片側ラドン変換**　片側ラドン変換を行う回路を図 1.25 に示す．簡略版アルゴリズムでは，重心を原点として片側ラドン変換を行うため，重心座標と動作開始信号が入力される．モジュール scan_ctrl は，開始信号を受けると，各画素を一巡する座標を順に生成し，それを元にフレームバッファから画素情報を取得する．同時に，重心から座標への方向が水平方向と成す角 $\theta$ を，モジュール atan を使って計算する．変換結果は外部のモジュールで保存する．モジュール delay は，各モジュール間で，結果が出るまでにかかるクロックの違いを補正する．モジュール scan_ctrl を，以下に示す．二つのカウンタを利用して，ピクセルを巡回する動作を作り出す．モジュール atan が計算に多数のクロックを必要とするために，画素値がモジュール atan をスムーズに流れるように，cnt1 より cnt2 の進行を遅くしてある．

```
void CORT_SCAN_CTRL::run(   uint1   CLK,
                            uint1   xRST,
                            uint1   START,
                            uint8   COG_X,
```

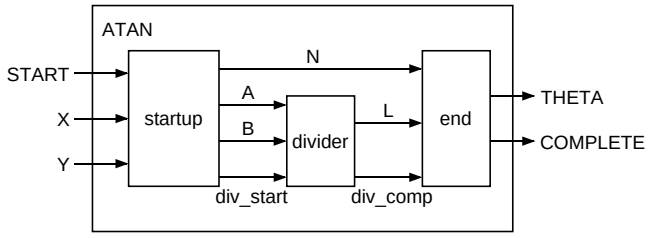



図 1.26: モジュール atan

```
                            uint8   COG_Y,
                            uint1&  ATAN_START,
                            uint9&  X,
                            uint9&  Y,
                            uint8&  R_X,
                            uint8&  R_Y
                        ){
    // cnt1,cnt2
    if(infer_clock(CLK) || infer_reset(!xRST) ){
        if(!xRST){
            cnt1 = 0;
            cnt2 = 0;
        } else {
            if(START || cnt1 || cnt2){
                if(cnt1<9){
                    cnt1++;
                } else {
                    cnt1 = 0;
                    cnt2++;
                }
            }
        }
    }

    // R_X,R_Y
    R_X = get_slice(cnt2,15,8);
    R_Y = get_slice(cnt2, 7,0);

    // X,Y
    X = (COG_X < R_X) ? uint8(R_X)-uint8(COG_X) :
                        (1<<8) | ( uint8(COG_X)-uint8(R_X) );
    Y = (COG_Y < R_Y) ? uint8(R_X)-uint8(COG_X) :
                        (1<<8) | ( uint8(COG_X)-uint8(R_X) );

    // ATAN_START
    ATAN_START = (cnt1==0) && cnt2;
}
```

モジュール atan は，除算回路と線形補間を用いて，図 1.26 に示すように

構成する．モジュール startup は，動作開始トリガ信号と，符号付座標値 X と Y を受け取る．出力 N は，求める角度 $\theta$ の上位 3 ビットの値である．これは，X と Y の正負ならびにこれらの絶対値の大小関係から判別できる．座標値の絶対値を信号 A と B に出力する．ただし，A≥B となるよう，必要に応じて絶対値を入れ替える．信号 L は，31A を B で除算したときの商である．モジュール end では，L を線形補間した値と N の値を合成して，角度 $\theta$ を算出する．モジュール startup を下記に示す．モジュール divider は，被除数と除数のビット数が違うだけで，先に説明したマクロを使って生成している．

```
void CATAN_STARTUP::run(uint1   CLK,
                        uint1   xRST,
                        uint1   START,
                        uint9   X,
                        uint9   Y,
                        uint13& A,
                        uint8&  B,
                        uint3&  N,
                        uint1&  DIV_START
                      ){
    if(infer_clock(CLK) || infer_reset(!xRST) ){
        if(!xRST){
            tmp_A = 0;
            tmp_B = 0;
            tmp_N = 0;
            tmp_DIV_START = 0;
        } else {
            if(START){
                uint1 flag = get_slice(X,7,0) < get_slice(Y,7,0);
                tmp_N = (flag<<2) | (get_bit(X,8)<<1) | get_bit(Y,8);
                if(flag)tmp_A = get_slice(Y,7,0);
                else    tmp_A = get_slice(X,7,0);
                tmp_A <<= 5;
                if(flag){
                    tmp_A -= get_slice(Y,7,0);
                    tmp_B = get_slice(X,7,0);
                } else {
                    tmp_A -= get_slice(X,7,0);
                    tmp_B = get_slice(Y,7,0);
                }
                tmp_DIV_START = 1;
            } else {
                tmp_DIV_START = 0;
            }
        }
    }
```

```
    A = tmp_A;
    B = tmp_B;
    N = tmp_N;
    DIV_START = tmp_DIV_START;
}
```

モジュール end を次に示す.

```
void CATAN_END::run(    uint1   CLK,
                        uint1   xRST,
                        uint1   DIV_COMP,
                        uint3   N,
                        uint13  L,
                        uint8&  THETA,
                        uint1&  COMPLETE
                ){
    if(infer_clock(CLK) || infer_reset(!xRST) ){
        if(!xRST){
            tmp_THETA = 0;
            tmp_COMPLETE = 0;
        } else {
            if(DIV_COMP){
                uint8 tmp = N;
                tmp_THETA = (tmp<<5) | get_slice(L,4,0);
                tmp_COMPLETE = 1;
            } else {
                tmp_COMPLETE = 0;
            }
        }
    }
    COMPLETE = tmp_COMPLETE;
    THETA = tmp_THETA;
}
```

モジュール atan のトップレベル接続を次に示す.

```
void CATAN::run(    uint1   CLK,
                    uint1   xRST,
                    uint1   START,
                    uint9   X,
                    uint9   Y,
                    uint8&  THETA,
                    uint1&  COMPLETE
                ){
    start.run(  CLK,
                xRST,
                START,
                X,
                Y,
                cc_out(a),
```

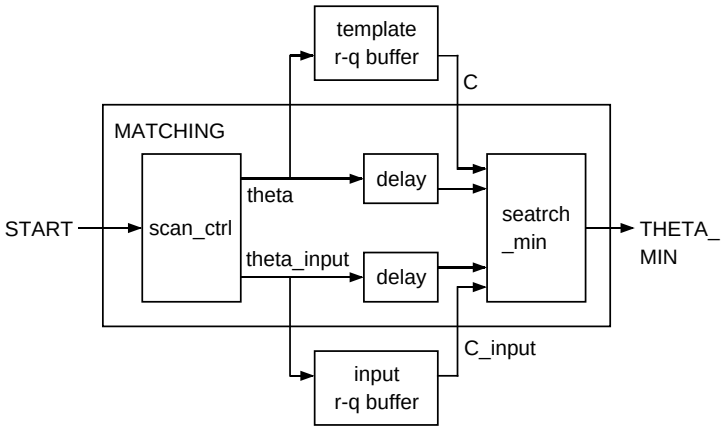



図 1.27: マッチングモジュール

```
                cc_out(b),
                cc_out(n),
                cc_out(div_start)
            );
    divider.run(    CLK,
                    xRST,
                    cc_in(a),
                    cc_in(b),
                    cc_out(l),
                    cc_out(odd_dmy),
                    cc_in(div_start),
                    cc_out(div_comp)
                );

    end.run(    CLK,
                xRST,
                cc_in(div_comp),
                cc_in(n),
                cc_in(l),
                THETA,
                COMPLETE
            );
}
```

**マッチング**　片側ラドン変換の結果からマッチングを行うために，誤差関数

$$S(\tau) = \sum_{\theta} \mid U_{input}(\theta + \tau) - U_{template}(\theta) \mid$$

を計算し，誤差関数の最小値を求める．この計算は，図 1.27 に示す回路で実現できる．モジュール scan_ctrl が $\theta$ と $\theta_{input} = \theta + \tau$ を生成し，対

応する片側ラドン変換の値をバッファから取得する．角度と変換値は，遅延を同一にしてからモジュール search_min へと送られ，絶対誤差の合計が最小になる角度を求める．モジュール scan_ctrl を下記に示す．

```
void CMATCHING_SCAN_CTRL::run(  uint1   CLK,
                                uint1   xRST,
                                uint1   START,
                                uint8&  THETA,
                                uint8&  THETA_INPUT
                             ){
    if(infer_clock(CLK) || infer_reset(!xRST) ){
        if(!xRST){
            cnt = 0;
        } else {
            if(START)   cnt = 0;
            else        cnt++;
        }
    }
    THETA = get_slice(cnt,7,0);
    THETA_INPUT = get_slice(cnt,7,0) + get_slice(cnt,15,8);
}
```

モジュール search_min を下記に示す．

```
void CMATCHING_SEARCH_MIN::run( uint1   CLK,
                                uint1   xRST,
                                uint8   THETA_A,
                                uint8   C_A,
                                uint8   THETA_B,
                                uint8   C_B,
                                uint8&  THETA_MIN
                              ){
    if(infer_clock(CLK) || infer_reset(!xRST) ){
        if(!xRST){
            diff_min = 0;
            tmp_THETA_MIN = 0;
        } else {
            uint8 d;
            if(C_A < C_B)   d = C_B - C_A;
            else            d = C_A - C_B;

            if(THETA_A==0) diff_min = 0xffff;
            if(THETA_B==0) diff=0;
            diff += d;
            if(THETA_B==0xff && (diff < diff_min) ){
                diff_min = diff;
                tmp_THETA_MIN = THETA_A;
            }
        }
```

```
    }
    THETA_MIN = tmp_THETA_MIN;
}
```

マッチングモジュールのトップレベル接続を次に示す.

```
void CMATCHING::run(    uint1   CLK,
                        uint1   xRST,
                        uint1   START,
                        uint8&  THETA_TEMP,
                        uint8   C_TEMP,
                        uint8&  THETA_INPUT,
                        uint8   C_INPUT,
                        uint8&  THETA_MIN
                    ){
    scan_ctrl.run(  CLK,
                    xRST,
                    START,
                    cc_out(theta),
                    cc_out(theta_input)
                );
    delay_a.run(    CLK,
                    xRST,
                    cc_in(theta),
                    cc_out(theta_a)
                );
    delay_b.run(    CLK,
                    xRST,
                    cc_in(theta_input),
                    cc_out(theta_b)
                );
    search_min.run( CLK,
                    xRST,
                    cc_in(theta_a),
                    C_TEMP,
                    cc_in(theta_b),
                    C_INPUT,
                    THETA_MIN
                );
    THETA_TEMP = cc_out(theta);
    THETA_INPUT = cc_out(theta_input);
}
```

**映像出力**　実際のハードウェアでは，ビデオフレームレート画像処理 VLSI はデバッグ用にデジタル映像信号を出力している．デジタル映像出力を MU200-VENC を通して NTSC 信号へ変換し，モニターにデバッグ用映像を出力する．MU200-VENC の仕様を表 1.5 に，外観を図 1.28 に示す.

表 1.5: MU200-VENC の仕様

| | | |
|---|---|---|
| メーカー | | 三菱電機マイコン機器ソフトウェア株式会社 |
| 製品名 | | 電子回路設計・ラピッドプロトタイピングキット<br>PowerMedusa シリーズ<br>NTSC/PAL ビデオエンコードコンポーネント |
| 型番 | | MU200-VENC |
| 仕様 | 入力端子 | 240 ピンコネクタ (メス)<br>DS01R-240R(ケル株式会社製レセプタクル コネクタ) |
| | 出力端子 | RCA ピンジャック (NTSC/PAL コンポジット信号)<br>S 端子 (MiniDin4,NTSC/PAL Y/C 信号) |
| | 電源 | DC+5V (最大約 1.3A) |
| | 最大外形寸法 | 165×150×17mm |
| | 入力ビデオ信号 | 24 ビット RGB 信号 (デジタル TTL レベル)<br>16 ビット YCrCb 信号 (デジタル TTL レベル)<br>(各種制御信号も入力が必要) |
| | 出力ビデオ信号 | NTSC コンポジット信号 (アナログ 75Ω 終端 1Vp-p)<br>NTSC Y/C 信号 (アナログ 75Ω 終端 1Vp-p)<br>NTSC RGB 信号 (アナログ 75Ω 終端 1Vp-p)<br>PAL コンポジット信号 (アナログ 75Ω 終端 1Vp-p)<br>PAL Y/C 信号 (アナログ 75Ω 終端 1Vp-p)<br>PAL RGB 信号 (アナログ 75Ω 終端 1Vp-p) |
| | ビデオエンコード用 IC | Brooktree 社 Bt856<br>(D/A 変換,Y/C 混合, 色変調) |
| | パラメータ設定用 IC | ALTERA 社 EPF81500AQC240-4<br>ALTERA 社 EPC1PC8 |

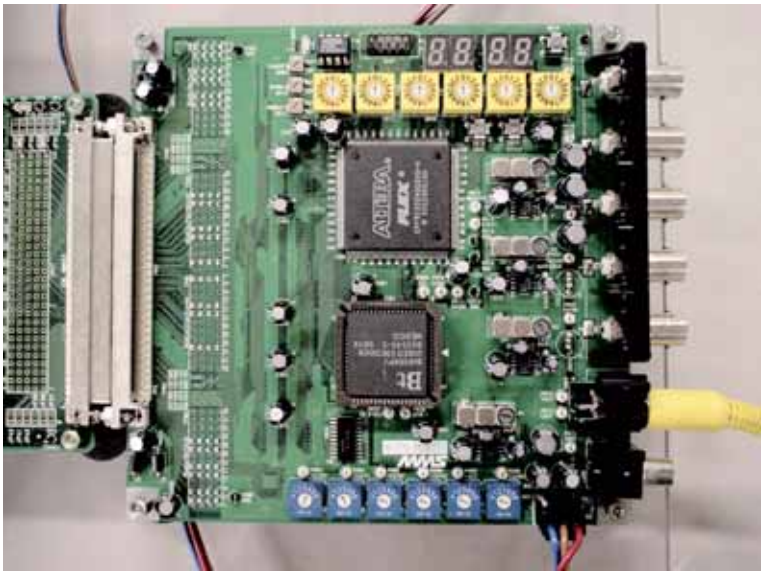


図 1.28: MU200-VENC の外観

シミュレーションでは，MU200-VENC とモニターの代わりに，下記の process_receive_signal 関数で映像信号を受け取って，画像として構築する．

```
static void process_receive_signal( uint1 VENC_CLKx1_OUT,
                                    uint1 VENC_CLKx2_OUT,
                                    uint8 VENC_R,
                                    uint8 VENC_G,
                                    uint8 VENC_B,
                                    uint1 VENC_HD_OUT,
                                    uint1 VENC_VD_OUT,
                                    uint1 VENC_FE_OUT,
                                    uint1 VENC_ACT_OUT){
    static UWORD x=0;
    static UWORD y=0;
    static uint1 last_HD = 1;
    static uint1 last_VD = 1;

    x++;
    if(last_HD==1 && VENC_HD_OUT==0){ x=0; y+=2; }
    if(last_VD==1 && VENC_VD_OUT==0) y = VENC_FE_OUT==0;

    UWORD xx = x-X0; UWORD yy = y-Y0;
    if(xx<(X1-X0) && yy<(Y1-Y0))
        s_received_image_bitmap[ xx + yy*s_received_image_pitch ] =
            RGB(VENC_R,VENC_G,VENC_B);

    last_HD = VENC_HD_OUT;
    last_VD = VENC_VD_OUT;
```

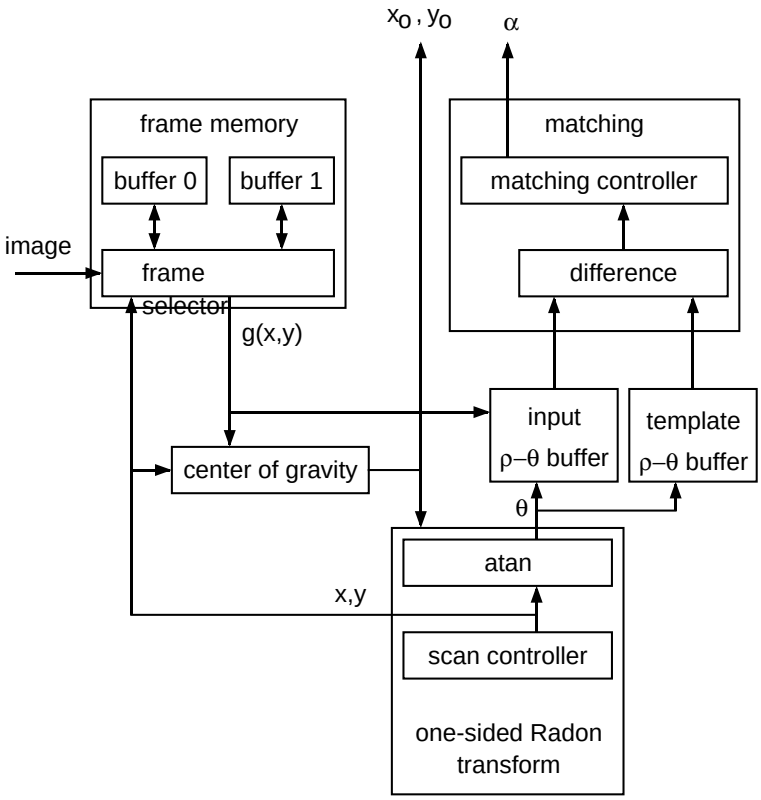
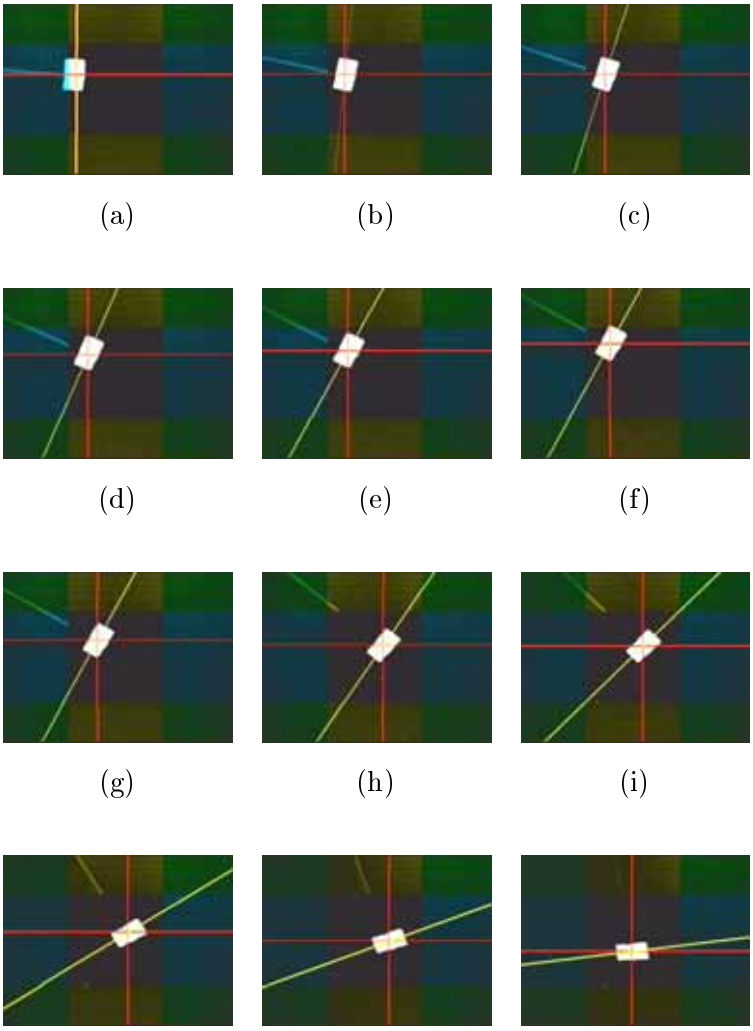



図 1.29: 回路全体の構成

}

### 1.5.3　実装回路

回路全体の構成を，図 1.29 に示す．入力画像より平面運動物体の位置 $x_0, y_0$ と姿勢 $\alpha$ を算出する．実装回路での映像入力は，MU200-VDEC を使用する．今回の画像処理は色情報は用いず，輝度情報を使用するので，出力画像のフォーマットは，24 ビット YCrCb 信号を選択した．映像のソースには，CCD カメラと市販のデジタルカメラを使用した．画像処理に使用する回路は，第 5 章 2 節で述べた CycleC での記述を，SystemCompiler を用いて HDL に変換し，それを論理合成して FPGA にコンフィグレーションした．映像出力は，MU200-VENC を用いて NTSC 信号へと変換し，モニタに出力した．



(a)　(b)　(c)

(d)　(e)　(f)

(g)　(h)　(i)

(j)　(k)　(l)

図 1.30: FPGA ビジョンシステムによる位置と姿勢の検出

FPGA ビジョンシステムを用いて位置と姿勢を検出した例を，図 1.30 に示す．十字カーソルの交点が物体の位置を，斜めの線が物体の姿勢を示す．物体の位置と姿勢がビデオフレームレートで検出できることを確認した．

## 1.6　おわりに

本章では，FPGA ベースリアルタイムビジョンを開発し，平面運動物体の位置と姿勢をビデオフレームレートで検出した．まず，FPGA を用いてビジョンシステムを構成することの優位性を指摘し，FPGA ベースリアルタイムビジョンのプロトタイプを試作した．次に，片側ラドン変換を用いて物体の位置と姿勢を検出するアルゴリズムを述べ，アルゴリズムの並列性を指摘した．次に，並列投票の概念を導入し，片側ラドン変換の並列アルゴリズムを構築した．さらに，並列アルゴリズムをオリジナルアルゴリズムと比較した．その結果，並列アルゴリズムは，オリジナルアルゴリズムと同程度に平面運動物体の位置と姿勢を検出できることがわかった．ただし，絶対誤差が 10 [degree] 近くに達しており，アルゴリズムの改良が必要である．最後に，片側ラドン変換アルゴリズムを FPGA ベースリアルタイムビジョンに実装した．平面運動物体の位置と姿勢を，ビデオフレームレートで検出できることを確認した．

今後の課題として，1) 物体の変形に対応できるビジョンアルゴリズムの構築，2) 一般化ハフ変換法など別のビジョンアルゴリズムの実装が挙げられる．

# 関連図書

[1] Ballard, D., H., *Generalizing the Hough Transform to Detect Arbitrary Shapes*, Pattern Recognition, Vol. 13, No. 2, pp.111–122, 1981

[2] Onishi, H. and Suzuki, H., *Detection of Rotation and Parallel Translation Using Hough and Fourier Transforms*, Proc. 1996 Int. Conf. on Image Processing, Vol.3, pp.827–830, 1996

[3] Tsuboi, T., Masubuchi, A., Hirai, S., Yamamoto, S., Ohnishi, K., and Arimoto, S., *Video-frame Rate Detection of Position and Orientation of Planar Motion Objects using One-sided Radon Transform*, Proc. IEEE Int. Conf. on Robotics and Automation, Vol. 2, pp.1233–1238, Seoul, May, 2001

[4] Thompson, C. D., *Fourier Transforms in VLSI*, IEEE Trans. on Computers, Vol.C-32, No.11, pp.1047–1057, 1983

[5] Maresca, M., Lavin, M., and Li, H., *Parallel Hough Transform Algorithms on Polymorphic Torus Architecture*, Levialdi, S. eds., *Multicomputer Vision*, Academic Press, pp.9–21, 198

[6] 井上, 稲葉, 森, 立川, 局所相関演算に基づく実時間ビジョンシステムの開発, 日本ロボット学会誌, Vol.13, No.1, pp.134–140, 1995

[7] Bugeja, A. and Yang, W., *A Reconfigurable VLSI Coprocessing System for the Block Matching Algorithm*, IEEE Trans on VLSI Systems, Vol.5, No.3, pp.329–337, 1995

[8] 岡林, 沢崎, 内山, 佐藤, 組込型トラッキングビジョンの開発, 第 5 回ロボティクスシンポジア, pp.369–374, 2000

[9] 徐, コンピュータビジョンにおけるエピポーラ幾何, 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, 新技術コミュニケーションズ, pp.80–96, 1998

[10] Kanade, T. and Okutomi, M., *A Stereo Matching Algorithm with an Adaptive Window: Theory and Experiment*, IEEE Trans on Pattern Analysis and Machine Intelligence, Vol.16, No.9, pp.920–932, 1989

[11] Okutomi, M. and Kanade, T., *A Locally Adaptive Window for Signal Matching*, Int. J. of Computer Vision, Vol.7, No.2, pp.143–162, 1990

[12] Hariyama, M., Takeuchi, T., and Kameyama, M., *VLSI Processor for Reliable Stereo Matching Based on Adaptive Window-Size Selection*, Proc. 2001 IEEE Int. Conf. on Robotics and Automation, pp.1168–1173, Seoul, May, 2001

[13] 張山, 竹内, 亀山, 高性能ステレオビジョン VLSI プロセッサの試作, 日本機械学会ロボティクスメカトロニクス講演会 CD-ROM, 2001

[14] 石川, 超並列・超高速ワンチップビジョンとその応用, 日本ロボット学会誌, Vol.13, No.3, pp.335–338, 1995

[15] 小室, 鈴木, 石井, 石川, 汎用プロセッシングエレメントを用いた超並列・超高速ビジョンチップの設計, 電子情報通信学会論文誌, Vol.J81-D-I, No.2, pp.70–76, 1998

[16] 小室, 石井, 石川, 吉田, 高速対象追跡用ビジョンチップの設計と試作, 日本機械学会ロボティクスメカトロニクス講演会 CD-ROM, 2001

[17] 小室, 石井, 石川, 吉田, 高速対象追跡ビジョンチップ, 電子情報通信学会論文誌, Vol.J84-D-II, No.1, pp.75–82, 2001

[18] Eklund, J.-E., Svensson, C., and Aström, A., *VSLI Implementation of a Focal Plane Image Processor – A Realization of the Near-Sensor*

*Image Processing Concept*, IEEE Trans. on VLSI Systems, No.4, Vol.3, pp,322–335, 1996

## 成果発表および特許等の状況

1. Tsuboi, T., Masubuchi, A., Hirai, S., Yamamoto, S., Ohnishi, K., and Arimoto, S., "Video-frame Rate Detection of Position and Orientation of Planar Motion Objects using One-sided Radon Transform", Proc. IEEE Int. Conf. on Robotics and Automation, Vol. 2, pp.1233–1238, Seoul, May, 2001

2. 平井慎一, "知能ロボットとシステム LSI", 計測と制御, Vol.40, No.12, pp.857–864, 2001

3. 坪井, 平井, "片側ラドン変換を用いた平面運動物体の識別と位置・姿勢検出", ロボティクス・メカトロニクス'01 講演会予稿集 CD-ROM, 2001

4. 座光寺, 平井, "リアルタイムビジョンのための片側ラドン変換法の並列化", 第 19 回日本ロボット学会学術講演論文集 CD-ROM, 2001

5. 坪井, 平井, "片側ラドン変換を用いたビデオフレームレートでの平面運動計測とその実験的評価", 2001 年電子情報通信学会情報・システムソサイエティ大会講演論文集, p.225, 2001

6. 平井, 坪井, 増渕, 座光寺, "ラドン変換に基づくビジョンアルゴリズムの並列化と FPGA への実装", 電子通信情報学会技術研究報告 DSP2001-114, Vol. 101, No. 384, pp.31–38, 2001

7. 増渕, 平井, "片側ラドン変換アルゴリズムの FPGA 実装", 計測自動制御学会システムインテグレーション部門学術講演会予稿集, pp.277–278, 2001

8. 平井, 坪井, 増渕, 座光寺, "片側ラドン変換に基づく平面運動検出アルゴリズムの並列化", ロボティクスシンポジア予稿集, pp.257–262, 2002

# 第2章　FPGAを用いた画像マッチング

## 要旨

**1. 目的**

ビデオフレームレートでハンドリング対象物を識別し，その運動を検知することができるビジョンチップを， FPGA をベースに開発する．

**2. 成果**

**(1) 柔軟変形物の運動を認識するアルゴリズムの開発**

・ PC 上においてマッチトフィルター法を C 言語で実装し，運動物体の位置・姿勢の検出が可能であることを検証した．さらに，マッチトフィルター法にラプラシアンフィルターを組合わせることにより，運動物体の位置・姿勢検出能力が向上することを検証した．

・ マッチトフィルター法のフーリエ変換の計算回数を削減するため，ハフ・フーリエ変換法の原理を利用したフーリエ・ラドン変換法を開発した．フーリエ・ラドン変換法を PC 上において C 言語で実装し，運動物体の位置・姿勢の検出が可能であることを検証した．

**(2) ビジョンアルゴリズムの FPGA 上への実装**

・ Handel-C や SpecC などの C 言語ベース LSI 設計ツールに関する調査を行った結果，Handel-C が有効であると判断し，Handel-C FPGA 開発環境 DK1 を導入した．

・ Handel-C によりマッチトフィルター法の論理設計を行い，FPGA へ実装し，動作確認を完了した．

・ FPGA ベースのリアルタイムビジョンシステムを構築した．200 万ゲート規模の FPGA（Xilinx 社 Virtex-E），外部 RAM，PC インターフェースを実装し，システムを構築した．

**(3) ビジョンアルゴリズムの最適化**

・ 位置・姿勢の検出の処理速度は，ビデオレート処理 (33msec) を実現した．

**3. 今後の課題**

・ 変形物体の認識アルゴリズムの開発

・ ハンドリングシステムとビジョンシステムとの協調動作アルゴリズムの開発

## 2.1　はじめに

食品産業におけるハンドリングロボットや生産ラインでの組み立て用ロボットを実用化するためには，人間の目の代わりとなるビジョン（視覚）システムを実現する必要がある [9][13][35]．生産現場における外観検査をビジョンシステムに代用させようとした場合，画像処理を行い検査結果を出力するまでの時間は，生産ラインのコンベア等のスピードにより決定される．そのため，ビジョンシステムの高速処理は重要な課題である．

市販の CPU(Central Processing Unit) で構成したビジョンシステムの処理能力が足りない場合，ASIC(Application Specific Integrated Circuit) を設計して対応することになる．しかし，外観検査等では，検査項目の変更・追加が頻繁に起こるため，パラメータの調整だけでは対応できない場合が多く，新たに画像処理アルゴリズムの追加が必要となる．そのとき，ASIC では新たに設計しなおす必要があり，膨大な労力とコストが問題となる．そこで，我々はハードウエアの高速性とソフトウエアの柔軟性を兼ね備えた FPGA(Field Programmable Gate Array) を利用したビジョンシステムの実現を，本研究の主要課題とした．

近年，FPGA のゲート数の向上は目覚しく，数百万システムゲートを超える FPGA が出現し，ゲート数の点からは大規模なビジョンアルゴリズムの搭載が可能となってきた．FPGA を用いて，並列処理，パイプライン処理，分散処理を組合わせて設計することにより，ASIC 並みの高速処理が可能となる．また，FPGA の回路はプログラムと同じように設計の追加・変更が容易であるので，ASIC のような問題もない．また，CPU を用いた場合には，処理を追加するごとに処理時間が増えるが，FPGA では，並列処理，パイプライン処理，分散処理を利用することにより，処理時間の増加を抑えることが可能である．

ビジョンシステムのアルゴリズムは，位置合わせ（パターンマッチング，テンプレート画像と検査対象画像間の拡大縮小・回転・平行移動の検出）と検査・照合・分類の 2 つから構成される．検査・照合・分類のアルゴリズムは対象に依存するが，位置合わせのアルゴリズムは対象に依存しない．よって，本研究は，ビジョンシステムの中でも，特に画像位置合わせの高速化を目的とした．

2 枚の画像間の拡大縮小，回転，平行移動の変換パラメータを検出する代表的な従来方式として特徴点法，正規化相関法，マッチトフィルタ法，$\theta - \rho$ ハフ変換法，一般化ハフ変換法の 5 つの方式が提案されている．

特徴点方式は，画像からコーナーや円弧等を検出し，その特徴点を 2 枚の画像間で対応付けして変換パラメータを算出する方法である [27]．この方式は，画像が高品質な場合には高速に処理できるため，数多く製品化されている．しかし，画像が低品質な場合には特徴点の検出ができなかったり，偽の特徴点を検出したりするので，対応付けができなくなることがある．また，特徴点の数が多くなってくると対応付けに要する処理時間が膨大となる．

平行移動の変換を受けた図形を画像上から検出する方式として，正規化相関法が最も一般的に用いられている [11]．正規化相関法は，入力画像上のすべての位置において，参照画像と入力画像間の正規化相関係数を計算し，その値が最大となる位置を探索することにより，平行移動量を求める方式である．正規化された相関係数を利用することで，画像のリニアな濃度変動には影響されないため，外観検査装置などによく用いられている．しかし，入力画像上のすべての点において正規化相関係数を計算するため，計算量が多くなり時間がかかるという問題がある．この点に関して，粗密探索等の利用やハードウエア化等で高速化が行われているが，拡大縮小と回転のパラメータを含めて検出しようとすると実用化は困難となる．

また，光学情報処理によるマッチトフィルタや計算機による 2 次元フーリエ変換を用いた方式が提案されている [3][4][6][13][17][30]．マッチトフィルタは上記の正規化相関法を周波数平面上で計算する方式である．そのため，正規化相関法のもつ画像のリニアな濃度変動には影響されないというロバスト性を保持している．この方式は，2 次元フーリエ変換に FFT(Fast Fourier Transform) を用いることにより，効率よく計算できるので，正規化相関法に比べて計算コストが小さくなる．しかし，2 次元フーリエ変換の計算コストは大きいため，リアルタイム処理を実現するためには，専用 ASIC 等を用いる必要がある．

$\theta - \rho$ ハフ変換平面 [7][10][18][19] を用いて円・楕円・2 次曲線や任意の図形を検出する方式が提案されている [5][8][12][25][26][29]．$\theta - \rho$ ハフ変換平面から極大点等の特徴点を用いて拡大縮小，回転，平行移動のパラメー

タを検出している．しかし，対象となる図形が複雑な場合や画像が低品質になった場合には，$\theta-\rho$ ハフ変換平面は複雑になり，特徴点が多くなったり検出できなくなったりするので適用が困難となる．これらの問題を解決する方式として．Hough−Fourier 法が提案されている [20, 21, 22, 23, 24]．この方式は，O'Gorman らが提案した [19]$\theta-\rho$ ハフ変換平面と 1 次元フーリエ変換を組合わせることにより，回転角検出精度の高いマッチトフィルターを実現している．しかし，$\theta-\rho$ ハフ変換を行う際に，画像からエッジ検出する必要があるため，コントラストがゆるやかな画像に対しては適用できない．また，$\theta-\rho$ ハフ変換を生体の視覚のハイパーコラムのモデルとして平行移動量の検出方式が提案されていてたいへん興味深い [12]．

また，直線検出用の $\theta-\rho$ ハフ変換の考え方を任意の図形を検出する方式に一般化した一般化ハフ変換 [2][14][15][16][33] が提案されている．この方式は，局所的な仮説により計算した座標変換パラメータをパラメータ空間に投票するので，ノイズに対してたいへんロバストである．しかし，局所的な仮説にエッジ点を用いると，エッジ数が少ない画像に対しては処理時間が少なくてよいが，エッジ数が多い画像では，処理時間が膨大となる．また，拡大縮小，回転，平行移動用のパラメータ空間は 4 次元となり膨大なメモリが必要となる．そのため，複数のエッジ点をランダムに選びながら投票して，途中で投票値を評価し，停止したり，4 次元パラメータ空間を 2 次元等に射影したりする Randomized Hough [34] が提案されている．しかし，低品質な画像を対象とする場合には，多くの点を選ぶ必要があるので処理時間が膨大となり，本質的な解決にはなっていない．

以上の考察より，拡大縮小，回転，平行移動のパラメータを検出する場合には，画像劣化に対するロバスト性と処理時間の観点からマッチトフィルタが有効である．

## 2.2　マッチトフィルタを利用した位置合わせ

### 2.2.1　画像位置合わせ

本研究では画像の位置合わせを，参照画像 $I_0(x,y)$ と入力画像 $I_1(x,y)$ 間で拡大縮小，回転，平行移動の変換がある場合に，拡大率，回転角，平行移動量を検出することとする．参照画像 $I_0(x,y)$ と入力画像 $I_1(x,y)$ 間には，

$$I_1(x,y) = I_0(\tilde{x},\tilde{y}), \tag{2.1}$$

$$\begin{pmatrix}\tilde{x}\\ \tilde{y}\end{pmatrix} = s_0 \begin{pmatrix}\cos\theta_0 & \sin\theta_0\\ -\sin\theta_0 & \cos\theta_0\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix} + \begin{pmatrix}u_0\\ v_0\end{pmatrix} \tag{2.2}$$

の関係があるとする．ここに，$s_0$ は拡大率，$\theta_0$ は回転角，$(u_0, v_0)$ は平行移動量である．

### 2.2.2　マッチトフィルタの原理

マッチトフィルタ (matched spatial filter) 法は，1960 年に G.L.Turin[31][32] が，RF(radio frequency) 信号の中からレーダパルスのエコーを検出するために開発した手法である．マッチトフィルタを用いることにより，ノイズに埋もれた信号の中から，所定の信号とノイズの SNR(signal-to-noise-ratio) を最大にして，所定の信号の位置を検出することができる．D.Casasent 等 [3][4] は，光学系を用いてリアルタイムにマッチトフィルタの計算を行った．さらに，メリン変換を導入し，平行移動した信号だけでなく，拡大縮小と回転した信号にも適用できるようにした．

マッチトフィルタ法は，2 つの信号間の相互相関関数を周波数領域で計算する方式である．周波数領域での計算は，光学系を用いることにより容易に計算できるため，光学情報処理 [30] の分野で数多く研究されてきた．しかし，近年，FPAG(Field Programable Gate Array)，DSP(Digital signal processor) や CPU(Central processing unit) の目覚しい進歩により，フーリエ変換が FFT(Fast Fourier Transform) アルゴリズムを利用して高速に計算できるようになった．そのため，FFT を利用したマッチトフィルタ法 [6][17] が提案されている．以下にマッチトフィルタ法の基本

原理を示す.

拡大率 $s_0 = 1$，回転角 $\theta_0 = 0$ の場合，すなわち

$$I_1(x, y) \quad = \quad I_0(x - u_0, y - v_0) \tag{2.3}$$

のように平行移動のみが存在する $I_0(x, y)$ と $I_1(x, y)$ をそれぞれフーリエ変換すると，

$$F_0(\xi, \eta) \quad = \quad \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I_0(x, y)\ e^{-j2\pi(\xi x+\eta y)} dxdy, \tag{2.4}$$

$$F_1(\xi, \eta) \quad = \quad \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I_1(x, y)\ e^{-j2\pi(\xi x+\eta y)} dxdy \tag{2.5}$$

となる．ここで，$F_0(\xi, \eta)$ と $F_1(\xi, \eta)$ は，フーリエ変換のシフト定理 [1] より

$$F_1(\xi, \eta) = e^{-j2\pi(\xi u_0+\eta v_0)} \cdot F_0(\xi, \eta) \tag{2.6}$$

の関係にある．さらに，$F_1(\xi, \eta)$ と $F_0'(\xi, \eta)$ の積を逆フーリエ変換すると

$$C(x, y) \quad = \quad \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F_1(\xi, \eta) \cdot F_0^{'*}(\xi, \eta) \cdot e^{j2\pi(x\xi+y\eta)} d\xi d\eta, \tag{2.7}$$

$$= \quad \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-j2\pi(\xi u_0+\eta v_0)} \mid F_0'(\xi, \eta) \mid^2 \cdot e^{j2\pi(x\xi+y\eta)} d\xi d\eta, \tag{2.8}$$

$$= \quad \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mid F_0(\xi, \eta) \mid^2 \cdot e^{j2\pi((x-u_0)\xi+(y-v_0)\eta)} d\xi d\eta \tag{2.9}$$

ここで，$*$ は複素共役を示す．式 (2.9) は，Wiener-Khinchine の定理 [1] から

$$C(x, y) \quad = [I_0(x, y) \star I_0(x, y)] \otimes \delta(x - u_0, y - v_0) \tag{2.10}$$

と置き換えることができる．ここで，$\star$ は相互相関，$\otimes$ は畳み込み演算を示す．式 (2.10) から，$C(x, y)$ は $I_0$ の自己相関関数が $(u_0, v_0)$ だけ平行移動していることを示している．よって，$I_1$ は，$I_0$ に対して $(u_0, v_0)$ だけ平行移動していることを検出することができる．

### 2.2.3　マッチトフィルタを利用した回転角と平行移動量の検出

D.Casasent 等 [3][4] は，光学系を用いてリアルタイムにマッチトフィルタの計算を行った．さらに，メリン変換を導入し，平行移動した信号だけでなく，拡大縮小と回転した信号にも適用できるようにした．

以下にマッチトフィルタを利用して回転角と平行移動量を検出する手順を示す．まず，画像のパワースペクトルを計算して平行移動不変平面を求める．拡大率 $s_0 = 1$ の場合，$I_0(x, y)$ と $I_1(x, y)$ をそれぞれフーリエ変換すると，

$$F_0(\xi, \eta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I_0(\tilde{x}, \tilde{y})\ e^{-j2\pi(\xi\tilde{x}+\eta\tilde{y})} d\tilde{x}d\tilde{y}, \tag{2.11}$$

$$F_1(\xi, \eta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I_1(x, y)\ e^{-j2\pi(\xi x+\eta y)} dxdy \tag{2.12}$$

となる．ここで，$F_0(\xi, \eta)$ と $F_1(\xi, \eta)$ は，フーリエ変換のシフト定理より

$$F_1(\xi, \eta) = e^{-j2\pi(\xi u_0+\eta v_0)} \cdot F_0(\xi\cos\theta_0 + \eta\sin\theta_0, -\xi\sin\theta_0 + \eta\cos\theta_0) \tag{2.13}$$

の関係にある．さらに，それぞれのパワースペクトル $P_0(\xi, \eta)$,$P_1(\xi, \eta)$ は，

$$P_1(\xi, \eta) = P_0(\xi\cos\theta_0 + \eta\sin\theta_0, -\xi\sin\theta_0 + \eta\cos\theta_0) \tag{2.14}$$

となる．次に，回転の影響を平行移動に変換するために，$(\theta, \rho)$ の極座標に変換すると

$$P_1(\theta, \rho) = P_0(\theta - \theta_0, \rho) \tag{2.15}$$

となり，$P_1$ は $P_0$ に対して，回転角に対応して $\theta$ 軸方向に $\theta_0$ だけ平行移動していることになる．この平行移動量を求めるために，1 次元マッチトフィルタを用いる．

$P_0$，$P_1$ の $\xi$ 軸の 1 次元フーリエ変換を $Q_0$，$Q_1$ とすると

$$Q_1(\xi, \rho) = e^{-j2\pi\xi\theta_0} \cdot Q_0(\xi, \rho) \tag{2.16}$$

となり，その積を逆フーリエ変換すると，

$$C(\theta, \rho) = \int_{-\infty}^{\infty} Q_1(\xi, \rho) \cdot Q_0^*(\xi, \rho) \cdot e^{j2\pi\theta\xi} d\xi, \tag{2.17}$$

$$= \int_{-\infty}^{\infty} e^{-j2\pi\xi\theta_0} \mid Q_0(\xi, \rho) \mid^2 \cdot e^{j2\pi(\theta\xi)} d\xi, \tag{2.18}$$

$$= \int_{-\infty}^{\infty} \mid Q_0(\xi, \rho) \mid^2 \cdot e^{j2\pi(\theta-\theta_0)\xi} d\xi \tag{2.19}$$

となる．ここで，$*$ は複素共役を示す．式 (2.19) は，Wiener-Khinchine の定理 [1] から

$$C(\theta, \rho) = [P_0(\theta, \rho) \star P_0(\theta, \rho)] \otimes \delta(\theta - \theta_0,\ \rho) \tag{2.20}$$

と置き換えることができる．ここで，$\star$ は相互相関，$\otimes$ は畳み込み演算を示す．式 (2.20) から，$C(\theta,\rho)$ は，各 $\rho$ において，$P_0$ の自己相関関数が $\theta$ 軸方向に $\theta$ だけ平行移動していることを示している．さらに，求めた $C(\theta,\rho)$ を $\rho$ 軸方向に加算すると，

$$C'(\theta) = \int_{-\infty}^{\infty} C(\theta,\rho)d\rho \tag{2.21}$$

となり，その最大値の位置より，回転角を求めることがきる．

回転角と平行移動量が求まったならば，$I_1(x,y)$ と回転角を補正した $I_0'(x,y)$ は

$$I_1(x,y) \quad = \quad I_0'(x-u_0, y-v_0) \tag{2.22}$$

となる．ここで，再度，$I_1(x,y)$ と $I_0'(x,y)$ 間の平行移動量を検出するために，マッチトフィルタを用いる．$I_1(x,y)$ と $I_0'(x,y)$ をそれぞれフーリエ変換すると，

$$F_1(\xi,\eta) = e^{-j2\pi(\xi u_0+\eta v_0)}F_0'(\xi,\eta) \tag{2.23}$$

の関係になり，$F_1(\xi,\eta)$ と $F_0'(\xi,\eta)$ の積を逆フーリエ変換すると

$$\begin{aligned}
C'(x,y) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F_1(\xi,\eta)\cdot F_0^{'*}(\xi,\eta)\cdot e^{j2\pi(x\xi+y\eta)}d\xi d\eta, && (2.24)\\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-j2\pi(\xi u_0+\eta v_0)} \mid F_0'(\xi,\eta)\mid^2 \cdot e^{j2\pi(x\xi+y\eta)}d\xi d\eta, && (2.25)\\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mid F_0(\xi,\eta)\mid^2 \cdot e^{j2\pi((x-u_0)\xi+(y-v_0)\eta)}d\xi d\eta, && (2.26)\\
&= [I_0'(x,y)\star I_0'(x,y)]\otimes\delta(x-u_0, y-v_0) && (2.27)
\end{aligned}$$

となり，$C'(x,y)$ の最大値の位置が平行移動量となる．以上で，回転角と平行移動量を検出することができる．

### 2.2.4　マッチトフィルタを利用した拡大率・回転角・平行移動量の検出

以下にマッチトフィルタとメリン変換を利用して拡大率と回転角と平行移動量を検出する手順を示す．まず，画像のパワースペクトルを計算して

平行移動不変平面を求める．$I_0(x,y)$ と $I_1(x,y)$ をそれぞれフーリエ変換すると，

$$F_0(\xi,\eta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I_0(\tilde{x},\tilde{y})\ e^{-j2\pi(\xi\tilde{x}+\eta\tilde{y})}d\tilde{x}d\tilde{y}, \tag{2.28}$$

$$F_1(\xi,\eta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I_1(x,y)\ e^{-j2\pi(\xi x+\eta y)}dxdy \tag{2.29}$$

となる．ここで，$F_0(\xi,\eta)$ と $F_1(\xi,\eta)$ は，フーリエ変換のシフト定理より

$$F_1(\xi,\eta) = \frac{1}{\mid s_0 \mid}e^{-j2\pi(\xi u_0+\eta v_0)}F_0(\frac{\xi\cos\theta_0+\eta\sin\theta_0}{s_0},\frac{-\xi\sin\theta_0+\eta\cos\theta_0}{s_0}) \tag{2.30}$$

の関係にある．さらに，それぞれのパワースペクトル $P_0(\xi,\eta)$,$P_1(\xi,\eta)$ は，

$$P_1(\xi,\eta) = P_0(\frac{\xi\cos\theta_0+\eta\sin\theta_0}{s_0},\frac{-\xi\sin\theta_0+\eta\cos\theta_0}{s_0}) \tag{2.31}$$

となる．ただし，$1/s_0^2$ の影響は無視する．

次に，拡大縮小と回転の影響を平行移動に変換するために，$(\theta,\rho)$ の極座標に変換し，さらに $\rho$ 軸を対数座標変換すると

$$P_1(\theta,\log\rho) = P_0(\theta-\theta_0,\log\rho-\log s_0) \tag{2.32}$$

となる．さらに，$\lambda=\log\rho$,$\kappa=\log s_0$ とすると，

$$P_1(\theta,\lambda) = P_0(\theta-\theta_0,\lambda-\kappa) \tag{2.33}$$

となり，$P_1$ は $P_0$ に対して，回転角と拡大率に対応した $(\theta_0,\kappa)$ だけ平行移動していることになる．この平行移動量を求めるために，マッチトフィルタを用いる．$P_0$,$P_1$ のフーリエ変換を $Q_0$,$Q_1$ とすると

$$Q_1(\xi,\eta) = e^{-j2\pi(\xi\theta_0+\eta\kappa)}Q_0(\xi,\eta) \tag{2.34}$$

となり，その積を逆フーリエ変換すると，

$$C(\theta,\lambda) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} Q_1(\xi,\eta)\cdot Q_0^*(\xi,\eta)\cdot e^{j2\pi(\theta\xi+\lambda\eta)}d\xi d\eta, \tag{2.35}$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-j2\pi(\xi\theta_0+\eta\kappa)}\mid Q_0(\xi,\eta)\mid^2\cdot e^{j2\pi(\theta\xi+\lambda\eta)}d\xi d\eta \tag{2.36}$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mid Q_0(\xi,\eta)\mid^2\cdot e^{j2\pi((\theta-\theta_0)\xi+(\lambda-\kappa)\eta)}d\xi d\eta \tag{2.37}$$

となる．ここで，$*$ は複素共役を示す．式 (2.37) は，Wiener-Khinchine の定理 [1] から

$$C(\theta,\lambda) \quad = \quad [P_0(\theta,\lambda) \star P_0(\theta,\lambda)] \otimes \delta(\theta-\theta_0,\ \lambda-\kappa) \tag{2.38}$$

と置き換えることができる．ここで，$\star$ は相互相関，$\otimes$ は畳み込み演算を示す．式 (2.38) から，$C(\theta,\lambda)$ は $P_0$ の自己相関関数が $(\theta_0,\kappa)$ だけ平行移動していることを示している．よって，その最大値の位置より，回転角と拡大率を求めることがきる．

回転角と平行移動量が求まったならば，$I_1(x,y)$ と回転角と拡大率を補正した $I_0'(x,y)$ は

$$I_1(x,y) \quad = \quad I_0'(x-u_0, y-v_0) \tag{2.39}$$

となる．ここで，再度，$I_1(x,y)$ と $I_0'(x,y)$ 間の平行移動量を検出するために，マッチトフィルタを用いる．$I_1(x,y)$ と $I_0'(x,y)$ をそれぞれフーリエ変換すると，

$$F_1(\xi,\eta) = e^{-j2\pi(\xi u_0+\eta v_0)} F_0'(\xi,\eta) \tag{2.40}$$

の関係になり，$F_1(\xi,\eta)$ と $F_0'(\xi,\eta)$ の積を逆フーリエ変換すると

$$\begin{aligned}
C'(x,y) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F_1(\xi,\eta)\cdot F_0^{'*}(\xi,\eta)\cdot e^{j2\pi(x\xi+y\eta)} d\xi d\eta, && (2.41)\\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-j2\pi(\xi u_0+\eta v_0)} \mid F_0'(\xi,\eta)\mid^2 \cdot e^{j2\pi(x\xi+y\eta)} d\xi d\eta, && (2.42)\\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mid F_0(\xi,\eta)\mid^2 \cdot e^{j2\pi((x-u_0)\xi+(y-v_0)\eta)} d\xi d\eta, && (2.43)\\
&= [I_0'(x,y)\star I_0'(x,y)] \otimes \delta(x-u_0, y-v_0) && (2.44)
\end{aligned}$$

となり，$C'(x,y)$ の最大値の位置が平行移動量となる．以上で，拡大率，回転角，平行移動量すべてを検出することができる．

## 2.3　パソコン上への実装

2.2.3 節で説明した平行移動量と回転角を検出するマッチトフィルタのアルゴリズムをパソコン上に C 言語 (Visual C++, Microsoft) で実装した．用いたパソコンは，CPU に Pentium1.7[GHz] を搭載した OptiPlex-GX400(DELL) であった．以下，マッチトフィルタアルゴリズムの計算過程及び結果を説明する．

### 2.3.1　平行移動不変平面

図 2.1 に参照画像 $f_0(x, y)$ と入力画像 $f_1(x, y)$ を示す．まず，参照画像と入力画像を式 (2.11),(2.12) に従って，それぞれ 2 次元フーリエ変換 $F_0(\xi, \eta)$ と $F_1(\xi, \eta)$ を求める．

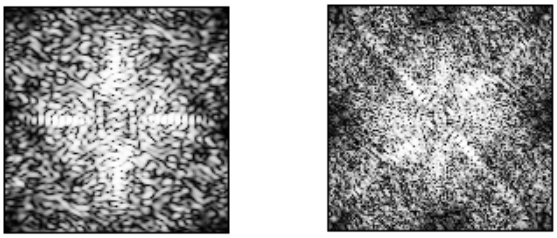
ここで，2 次元フーリエ変換を FFT(Fast Fourier Transform) アルゴリズムを用いて効率良く計算するため，画像サイズを 2 の N 乗にする必要がある．そこで，一般的には画像サイズを 2 の N 乗にするために画像の端には，0 を埋める．その結果，参照画像は，元の画像の背景との境界が長方形となり，4 角形のパターンを探索していることになる．そのため，本研究では，入力画像と参照画像に対して，ガウスラプラシアンフィルターを適用した．濃淡画像にガウスラプラシアンフィルターを適用することにより，画像の直流成分はなくなるため，背景値は 0 になり，2 の N 乗のサイズにするための 0 の埋め込みの影響を受けない効果がある．さらに，フーリエ変換のパワーを計算して，それぞれパワースペクトル平面を求める．求めた結果を図 2.2 に示す．図 2.2 に示すフーリエパワースペクトルは式 (2.14) より，参照画像と入力画像間の回転の影響を残して，平行移動の影響が消去されている．

### 2.3.2　回転角の検出

次に，参照画像と入力画像のパワースペクトル間の回転角の影響を平行移動に変換するために，それぞれ極座標変換する．その結果，式 (2.15) に示すように，入力画像のパワースペクトル $P_1(\theta, \rho)$ は，参照画像のパワースペクトル $P_0(\theta, \rho)$ に対して，回転角分だけ，$\theta$ 軸方向に平行移動してい

(a) Reference image　(b) Input image

図 2.1: 参照画像と入力画像

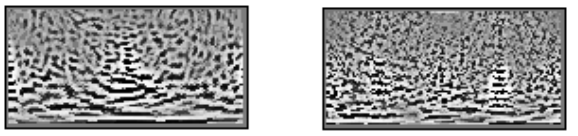


(a) Power spectrum of the reference image　(b) Power spectrum of the input image

図 2.2: 2 次元フーリエパワースペクトル

ることになる．図 2.3 にパワースペクトルの極座標変換を示す．次に，画像上に長い直線が存在すると，その周波数成分が強くなりパワースペクトルの特定の $\theta$ 軸上の値が高くなる傾向ある．そこで，パワースペクトル上において，$\theta$ 軸方向に 1 次元ガウスラプラシアンフィルターを掛けて，その影響を消去しておくことにより，画像上の長い直線の影響を排除することができる．

ここで，回転角の相当する平行移動量を検出するために，式 (2.16)~(2.21) に従って，$P_0(\theta, \rho)$，$P_1(\theta, \rho)$ 間で，$\theta$ 軸に関する 1 次元マッチトフィルタを計算することにより，$C'(\theta)$ を得ることができ，その最大値の位置の $\theta_k$ が求める回転角となる．$C'(\theta)$ の結果を図 2.4 に示す．



(a) Polar plane of the reference power spectrum　(b) Polar plane of the input power spectrum

図 2.3: パワースペクトルの極座標変換

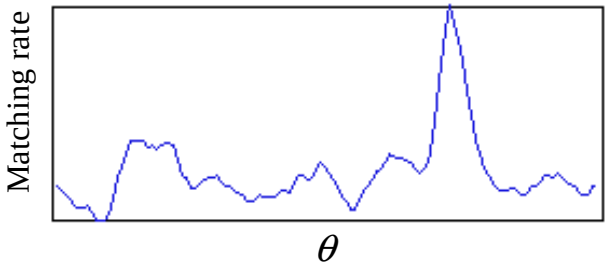



図 2.4: 回転角の検出

### 2.3.3　平行移動量の計算

検出した回転角 $\theta_k$ で補正した参照画像と入力画像間では，式 (2.22) に示すように平行移動のみが存在するので，その平行移動量を式 (2.23)∼(2.27) に従って，マッチトフィルタで検出する．

画像を $\theta_k$ 度回転させた画像のパワースペクトルと $\theta_k + 180$ 度回転させたパワースペクトルは同じになる．そのため，パワースペクトルからは，真の回転角が，前節で求めた回転角 $\theta_k$ 或いは $\theta_k + 180$ のどちらかはわからない．

そこで，$\theta_k$ 度回転させた参照画像の 2 次元フーリエ変換と $\theta_k + 180$ 度回転させた参照画像の 2 次元フーリエ変換に対して，入力画像のフーリエ変換とのマッチトフィルタを計算して，その最大値の位置を求め，その値が大きい方を真の回転角とし，その最大値の位置を平行移動量とする．図 2.5 に $\theta_k$ 度と $\theta_k + 180$ 度の回転角の場合のマッチトフィルタの結果を示す．以上の方式で求めた回転角と平行移動量は，309 度と $(112, 100)$ であった．図 2.6 に入力画像中の参照画像の位置・姿勢を検出した結果を示

す．全体の処理時間は，約 150[msec] であった．

### 2.3.4　Radon 変換を用いた高速化の検討

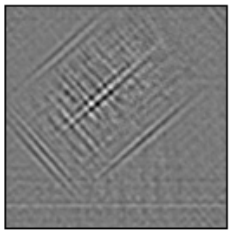
2.3.3 節では，平行移動量を求めるために，回転角の候補である $\theta_k$ 度と $\theta_k + 180$ 度に対して，2 回の 2 次元逆フーリエ変換を計算する必要があった．そこで，ラドン変換と 2 次元フーリエ変換の関係を利用して，2 次元フーリエ変換を用いないで平行移動量を求める方式を検討した．

まず，求めた回転角 $\theta_k$ 度と $\theta_k + 180$ 度のどちらが正しい回転角であるかを調べる．回転角 $\theta_k$ について，調べる場合，$F_0(\theta, \rho)$ と $F_1(\theta+\theta_k, \rho)$ 間の各 $\theta$ の $\rho$ 軸方向の 1 次元データについて，それぞれの積をとり，1 次元逆フーリエ変換した結果を $R_0(\theta, \rho)$ 平面に記憶する．$R_0(\theta, \rho)$ 平面の各 $\theta_i$ において，最大値の位置 $A_i(\theta_i, \rho_i)$ を求める．同様にして $\theta_k + 180$ の最大値の位置を求める．図 2.7 に回転角 $\theta_k$ 度と $\theta_k + 180$ 度の場合の最大値の位置を示す．

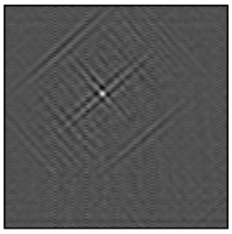
次に，求めた $A_i(\theta_i, \rho_i)$ から回転角を一意に決定する．もし，回転角 $\theta_k$ が真の回転角ならば，求めた $A_i(\theta_i, \rho_i)$ は，$R_0(\theta, \rho)$ 平面上でサインカーブ，

$$\rho = \sqrt{x_\Delta^2 + y_\Delta^2}\ \cos\left(\theta - \tan^{-1}\frac{y_\Delta}{x_\Delta}\right) \tag{2.45}$$

を描く [20, 21, 22, 23, 24]．偽の回転角ならば，ランダムな点列になる．よって，点 $A_i$ が連続している方の回転角が真の回転角となる．連続性は，連続する 2 つの $A_i$ と $A_{i-1}$ 間の距離が一定範囲内に存在する $A_i$ の個数 $V_0$



(a) Inverse fourier transform at angle k



(b) Inverse fourier transform at angle k+180

図 2.5: 逆フーリエ変換による平行移動量の検出

図 2.6: 位置合わせ結果

を求めることにより計算することができる．同様にして，$\theta_k + 180$ のときの $V_1$ を求める．$V_0$ が $V_1$ より大きいならば，$\theta_k$ が真の回転角となる．さもなければ，$\theta_k + 180$ が真の回転角となる．

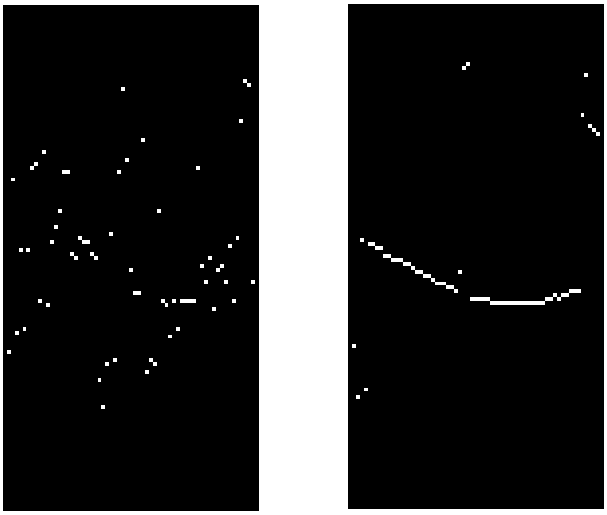
次に平行移動量を求めるために，$A_i(\theta_i, \rho_i)$ 上から 2 点ずつ，$A_i$ と $A_j$ を選びながら，連立方程式，

$$\rho_i = x \cdot \cos(\theta_i) + y \cdot \sin(\theta_i) \tag{2.46}$$

$$\rho_j = x \cdot \cos(\theta_j) + y \cdot \sin(\theta_j) \tag{2.47}$$

の解を記憶し，それらをクラスタリングし，クラス内の成分数の最大値のクラスターの位置の $(x, y)$ を平行移動量とする．

以上の方式で求めた回転角と平行移動量は，309 度と $(112, 100)$ であった．その結果を図 2.8 に示す．



(a) $R_0(\theta, \rho)$　　(b) $R_1(\theta, \rho)$

図 2.7: ラドン変換平面の最大値

図 2.8: 位置合わせ結果

## 2.4　FPGA への実装

近年，数百万システムゲートの大規模 FPGA が入手可能になっている．また，C 言語を用いて ASIC や FPGA のロジックを設計するためのツールもその数を増しつつある．このような状況下で，複雑なアルゴリズムを FPGA へ実装し，高速に処理を行うことが容易になりつつある．特に，C 言語を用いて開発を行うことにより，C 言語で記述される場合が多いソフトウェアのハードウェアへの移行が容易に行えることが期待される．また，画像処理などのアルゴリズムを対象とした場合，C 言語で設計可能であるということは，コードの可読性・保守性の面においても有利であると思われる．

一般的に FPGA ロジックの動作周波数は数十〜数百 MHz 程度であり，パソコンの CPU の数 GHz という動作周波数よりも低い．しかし，FPGA では，並列化・パイプライン化・メモリアクセスの最適化などによって処理クロック数の削減を行うことにより，比較的低い動作周波数においてでも特定の処理をパソコンの CPU よりも高速に行える可能性が高い．多数の互いに独立な処理を FPGA 上に並列に実装し，大幅な処理速度の向上を行った例も報告されている [36]．本プロジェクトでは扱うデータ量が大きく，また大規模な処理の並列化も困難であるため，主に処理のパイプライン化により処理速度の向上を図っている．

FPGA は，一般的に量産化された同規模の ASIC と比較した際の単価は高いが，システムのパフォーマンスを低下させることなくロジックの変更・追加・削除が行えるうえ，将来的にはより大規模・高性能な FPGA へのロジックの移植が可能であり，搭載アルゴリズムの改良や処理速度の向上なども比較的少ない費用で行うことが可能である．

本プロジェクトでは，Handel-C と呼ばれるロジック開発用の C 言語を重点的に用い，マッチトフィルタアルゴリズムを 1 つの FPGA と 4 つの同期 SSRAM を搭載したボード上で動作させた．その際，パイプライン化及び高性能な IP コアの使用により，パソコンよりも高速な処理を実現した．本章では，FPGA ロジック開発ツール・機材，FPGA 処理の高速化，アルゴリズムの FPGA への実装について説明する．

| 名 称 | 用 途 |
|---|---|
| Handel-C 開発環境 DK1 | Handel-C 言語コードのネットリストへの変換 |
| FPGA 開発ツール Foundation ISE | VHDL コードのネットリストへの変換<br>IP コアの生成<br>ネットリストの配置配線 |
| VHDL シミュレータ ModelSim SE | VHDL コードのシミュレーション |
| FPGA 解析ツール ChipScope ILA | FPGA 内部信号の解析 |
| FPGA 開発ボード ADM-XRC | FPGA の動作確認，実験 |
| プログラミング環境 Visual C++ | ADM-XRC 制御プログラム作成<br>アルゴリズム検証 |
| FPGA 設計・動作確認用パソコン | FPGA ロジック生成，FPGA 開発ボード制御 |
| 数値解析プログラム MATLAB | データ解析・検証 |
| 検証・解析用パソコン | データ解析・検証，アルゴリズム検証 |

表 2.1: FPGA ロジック設計ツール及び機材

### 2.4.1　開発ツール・機材

マッチトフィルタアアルゴリズムの FPGA への実装及び動作確認は，下記のツール及び機材を用いて行った．

上述の項目の中で，特に重要なツール及び機材に関して説明する．

#### Handel-C 開発環境 DK1

Celoxica 社の DK1 は，ANSI-C 言語に対してハードウェア設計用に拡張を施した Handel-C 言語の統合開発環境である．Handel-C 言語では，並列処理，パイプライン処理，ビット操作，インターフェースなどを取り扱うための言語拡張やマクロの強化などがなされており，また，基本的に代入処理の記述に対してフリップフロップが生成され，クロックバウンダリが生成される．本プロジェクトでは，主要な FPGA ロジック設計に Handel-C 言語及び DK1 を用いた．

**FPGA 開発ボード ADM-XRC**

ALPHA DATA parallel systems ltd の ADM-XRC は，Xilinx 社の約 200 万システムゲートの FPGA である Virtex2000-E-6，4 系統の 2MB(512K*36bit) 外部 ZBT SSRAM(Zero Bus Turnaround Synchronous Static RAM)，PCI コントローラー，２系統のプログラマブルクロックジェネレータなどを搭載した FPGA 開発ボードである．同ボードには，パソコンでボード制御を行うためのドライバ及び SDK が付属しており，ホストパソコンから PCI バス経由での FPGA コンフィギュレーション及び FPGA とのデータ転送が行える．なお，ZBT SSRAM とは，書き込みと読み出しの切り替えを，ウェイトを入れることなく行うことが可能な同期スタティックRAM である．同ボードの外観 (図 2.9，図 2.10) 及び主要機能ブロック図 (図 2.11) を示す．

図 2.9: ADM-XRC 外観

### 2.4.2　処理の高速化

処理の高速化のためには，処理に要するクロック数の削減と，動作クロック数の引き上げが有効である．処理に要するクロック数の削減には，並列化，外部メモリアクセスの最適化，パイプライン化が有効である．また，動作クロック数を引き上げるためには，遅延時間の制御が必要である．さらに，高速動作可能な IP コアなどのモジュールの活用も，処理に要す

図 2.10: ADM-XRC 外観 (PCI キャリアボードに装着)

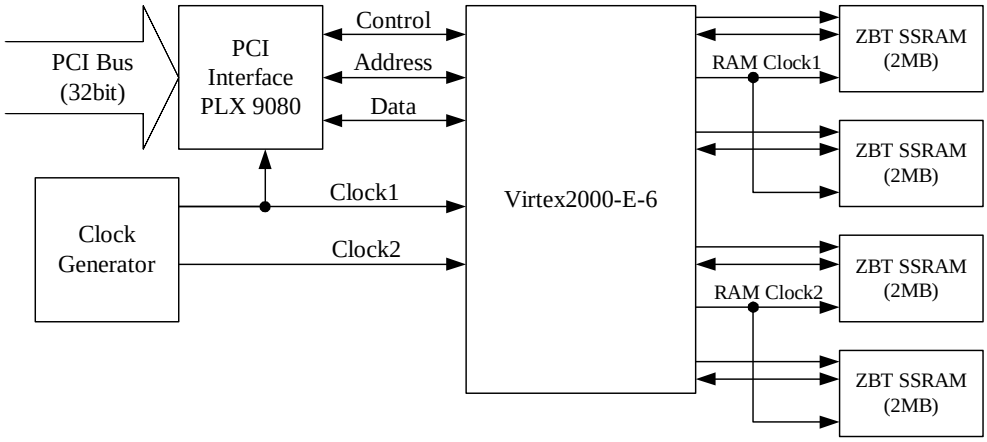



図 2.11: ADM-XRC 主要機能ブロック図

るクロック数の削減，動作クロック数の引き上げに有効である．

### 高速化の手法

**並列化**　FPGA では，処理を並列化し，同時に複数の処理を行うことが可能である．並列化を行う際には，リソースやデータの競合が発生しないよう注意する必要がある．図 2.12 に乗算の並列化の例を示す．この例では，4 重の並列化を行っているため 4 つの乗算器を用意する必要があるが，並列化無しで 4 クロック要する処理を 1 クロックで完了することが可能である．

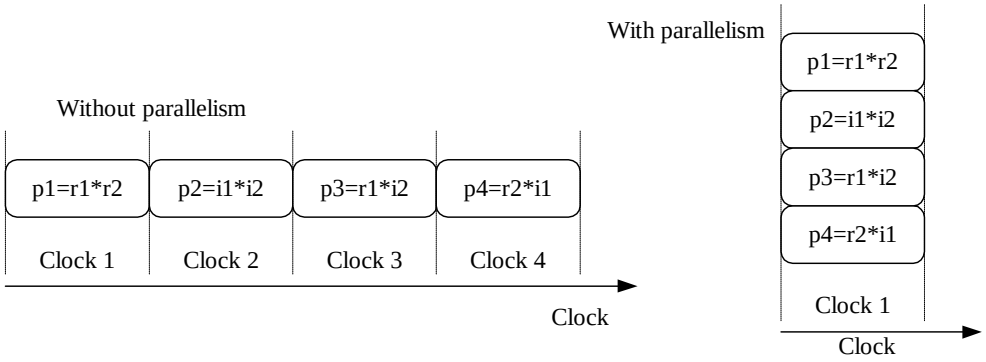



図 2.12: 並列化の例

**メモリアクセスの最適化**　ADM-XRC の 4 個の外部 ZBT SSRAM は，独立なアクセスが可能であり，同一クロック内で各々の RAM に対してデータの読み出しと書き込みを行うことが可能である．このことにより，後述するような，外部 RAM に対するデータの読み出しと書き込みを同時に行うパイプライン処理が可能となる．

また，Virtex2000-E 内部には，Block RAM と呼ばれる 4096bit の dual-port RAM が 160 個搭載されている．このメモリをバッファとして使用し一時的にデータを保持することにより，外部 RAM へのアクセスを軽減することが可能である．例えば，画像に対して 3x3 のフィルタリング処理を行う場合，バッファを用いなければ各画素を処理するために 9 回の外部 RAM リードが必要であるが，2 ライン分のバッファを用意することで，1 回の外部 RAM リードにより各画素のフィルタリングを行うことが可能となる [37].

**パイプライン化**　大量のデータに対して同じ処理を行う場合，パイプライン化によって全体処理クロック数の削減が行える可能性が高い．パイプライン化を行うことにより，パイプライン充填後，1 クロック毎に処理結果を得ることが可能になる．複数の異なる処理をパイプライン化することによりステージ数が大きいパイプライン処理を構築し，大幅な処理クロック数の削減を実現することも可能である．4 ステージパイプライン処理の例を図 2.13 に示す.

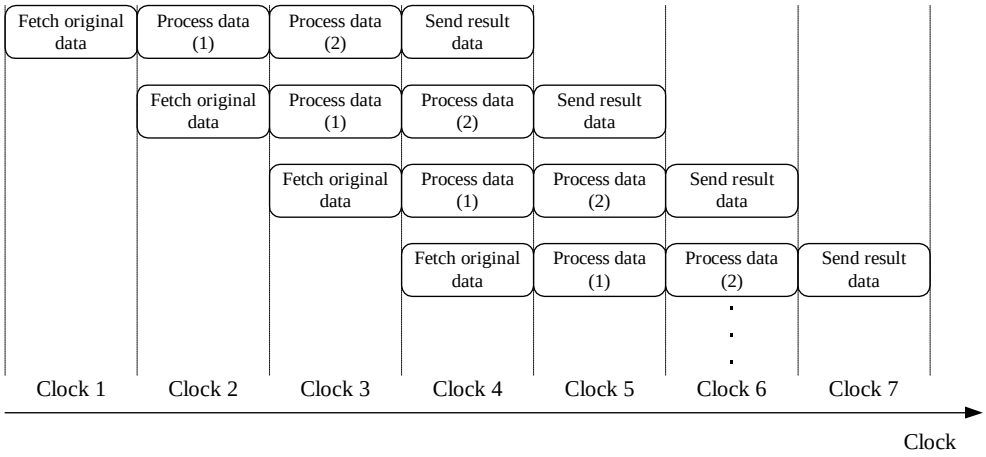



図 2.13: 4 ステージのパイプライン処理

**高速動作可能なモジュールの活用**　Xilinx 社では，グレードが高い開発ツールのユーザーに対して，無償で使用出来る高性能なハード IP コアの提供を行っている．これらの IP コアは，自作したロジックよりも最高動作周波数，処理クロック数，回路面積に関して優れていることが多い．本プロジェクトでは，これらの IP コアを活用し処理速度の向上を図った．特に FFT コアは，マッチトフィルタ処理の中で繰り返し使用している．また，本プロジェクトで使用した，60MHz 以上で動作可能な VHDL モジュール数点の作成と，square root IP コアの供給を，東京エレクトロンデバイス株式会社に依頼した．

**遅延時間の制御**　同期回路設計においては，フリップフロップ同士・フリップフロップ-外部デバイス間の遅延時間を目標値よりも短く保たねばならない．そのため，必要に応じてフリップフロップを追加し，遅延時間を短くする必要がある．ただし，フリップフロップを追加すことは処理クロック数及び消費ロジック数の増大につながる．また，演算精度が必要以上に高い部分は，消費ロジック数及び遅延の増大を招くため，遅延の増大を防ぐためには演算精度を必要最小限に留めることが望ましい．

**高速化の例**

高速化の例として，2 つの外部 RAM に別々に保存されている 2 つの 2 次元複素数データの乗算を行い，その結果を 3 番目の外部 RAM に書き込むという処理について説明する．図 2.14 に，この例のクロックと処理内容の関係を示す．この例では，並列化，メモリアクセスの最適化，パイプライン化，ロジック・配線遅延の制御を行っている． この例では，処理開始から 9 クロック目で最初の演算結果を外部 RAM に対して書き込み，以降 1 クロック毎に演算結果を外部 RAM に対して書き込む．今回の設計では，縦 256 点・横 256 点のデータの 1 行単位でパイプライン処理を行っているため，1 行の処理に要するクロック数は，256+8=264 クロックとなる．パイプライン化を行わなければ，1 行の処理に 256x9=2,304 クロックが必要である．なお，1 行ではなく画像全体を対象としたパイプライン化を行うことも可能であるが，設計工数と効果のバランスを考慮し，行単位のパイプライン化に留めた．

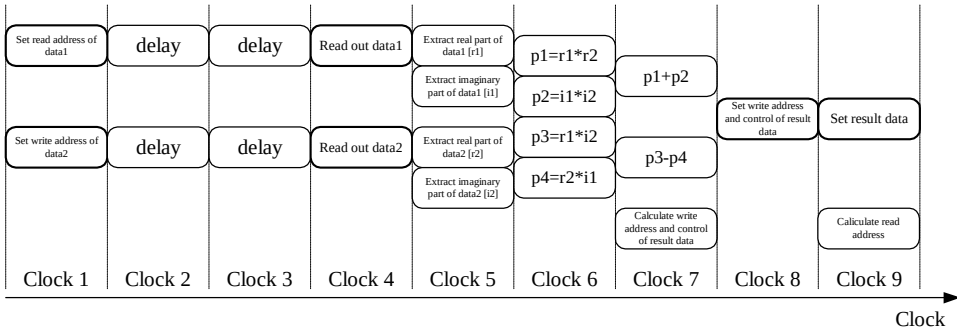



図 2.14: 複素乗算の高速化

### 2.4.3　アルゴリズムの FPGA への実装

今回のプロジェクトでは，IP コア，VHDL モジュール以外の主要な処理を Handel-C で記述した．前節で述べた高速化の大部分は Handel-C のソースコード上で記述することにより行った．

なお，DLL やクロックドメインの異なる Handel-C モジュールの結合な

どを行うための top level モジュールの記述は，VHDL で行った (図 2.15)．マッチングモジュールの 50MHz 以上での動作と，PCI コントローラ制御モジュールの 40MHz 以下での動作を同時に行うため，マッチングモジュールと PCI コントローラ制御モジュールを異なるクロックドメインに配置し，両者の間を FIFO を用いて結合した．両モジュール間のデータ転送には，コマンド /レスポンス方式を使用した．外部 RAM はマッチングモジュールに同期させ，外部 RAM へのアクセスはマッチングモジュールからのみ行うように設計した．また，マッチングモジュールが使用するクロックと外部 RAM が使用するクロックの位相差を除去するため，Virtex-E に搭載されている DLL(Delay Locked Loop) を使用した．

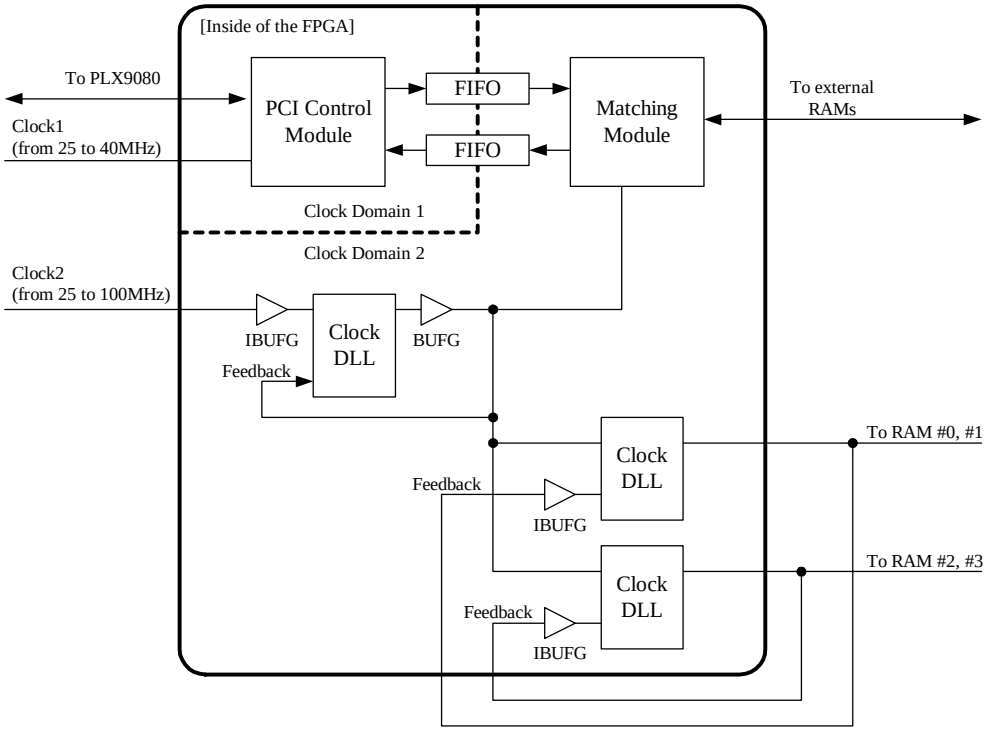



図 2.15: top level モジュール概略図

FPGA へのアルゴリズムの実装時，データ型は 8bit 実数の入力画像を除いて 32bit の固定小数点複素数 (実部 16bit，虚部 16bit) を用いた．オーバーフローなどを回避するため，データ有効桁の調整を適宜行っている．外部 RAM の 2 次元データを読み込み，その処理結果を読み込みに使用し

ていない外部 RAM へ書き込む処理をひとつの処理単位として設計を行った．1 次元 FFT 以外の処理単位に関しては，1 点のデータを 1 クロックで処理するパイプライン化を行った．さらに，特に効果が大きい箇所に関しては，複数の処理に対して連続したパイプライン化を施したものを 1 つの処理単位とした．ロジックの動作確認は，まず各処理単位ごとに行い，次に全体に対して行うことを繰り返した．

Handel-C では，メモリに対して 3〜4 倍速のクロックをメモリに供給することによって見かけ上 1 クロックでメモリアクセスを行うことも可能であるが，今回は外部 RAM の最高動作周波数 100MHz の半分である 50MHz 以上でのメモリアクセスを実現するため，Handel-C を用いて独自にパイプライン メモリアクセスの記述を行った．

マッチング実行の際は，ホストパソコンから PCI バス経由でボード上のメモリに登録画像特徴量，入力画像有効領域フラグ画像及び入力画像の転送が完了後，ホストパソコンからのコマンドで FPGA が処理を開始するよう設計を行った．なお，データ転送量削減のため，登録画像の特徴量及び入力画像有効領域のフラグ画像はリアルタイム処理を開始する前に転送を済ませておく．リアルタイム処理時の入力画像転送には DMA 転送を用いた．

### FPGA 処理詳細

FPGA 実装時の，処理単位レベルでの処理フローを図 2.16 に示し，以下に詳細を説明する．

**ガウスラプラシアンフィルタリング (Gauss Laplacian filtering)**　外部 RAM に保存されている入力グレー画像及び入力画像有効領域フラグ画像を読み込みながら，フィルタリング結果にデータ幅の調整を施し，読み込みで使用していない外部 RAM へ書き出しを行う．なお，本処理には，ライン メモリを用いてガウスラプラシアンフィルタリングを行う VHDL モジュールを使用した．

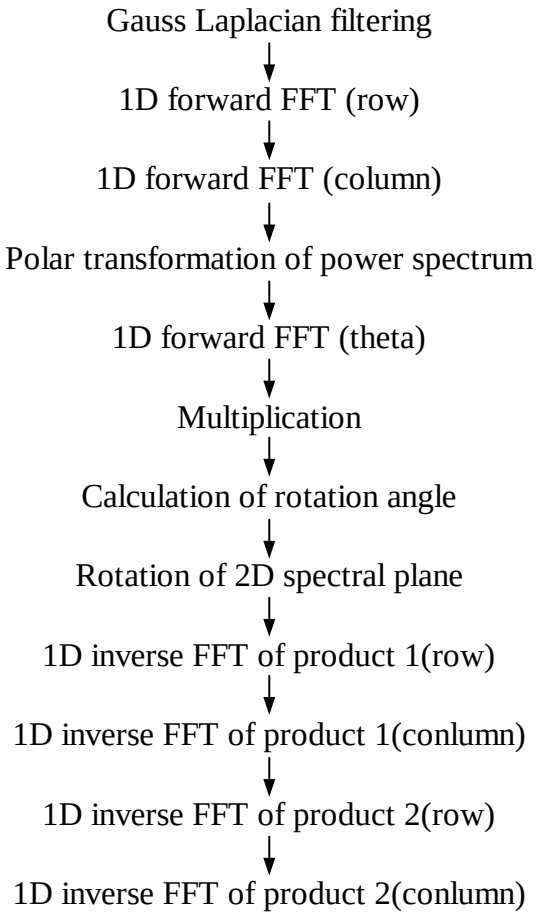



図 2.16: 処理フロー

**行方向 1 次元フーリエ変換 (1D forward FFT)**　フィルタリングの結果に対し，2 次元 FFT を施すが，今回は 1 次元 FFT IP コアを用いて row-column 法による 2 次元 FFT を行っている．本処理単位は，row-column 法の第一段階で，2 次元データの各行に対し，水平方向の 1 次元 FFT を施す．本プロジェクトでは，FPGA 内に 1 つの FFT コアを実装し，それを繰り返し使用している．FFT コアの動作モードは，最も処理クロック数が少なくて済む TMS モードを使用した．

**列方向 1 次元フーリエ変換 (1D forward FFT)**　Row 方向 FFT の結果に対して，今度は垂直方向に FFT を適用する．

**フーリエパワースペクトルの極座標変換 (Polar transformation of power spectrum)**　2 次元フーリエ平面に対して，極座標変換，パワーの算出，パワーの周波数方向の微分を順に施し，その結果の外部 RAM への書き出しのパイプライン処理を行う．極座標変換の結果，横軸が角度，縦軸が周波数の平面が得られる．極座標変換にはアドレス変換用 VHDL モジュールを，また，パワー算出の際の平方根演算には square root IP コアを使用した．

**θ 方向 1 次元フーリエ変換 (1D forward FFT)**　極座標変換されたフーリエパワースペクトルに対し θ 方向に 1 次元 FFT を施す．

**乗算 (Multiplication)**　入力画像に対してこれまでの処理を行った結果の複素共役と，前もって作成しておいた参照画像に対して同じ処理を行った結果との複素乗算のパイプライン処理を行う．

**回転角の算出 (Calculation of rotation angle)**　複素乗算結果に対して θ 方向 1 次元逆 FFT を施し，さらに結果の半径方向の和の算出及び半径方向の和が最大となる θ (回転角) の検出を逆 FFT 結果の取り出しとパイプライン化して行う．なお，今回使用した FFT コアは，制御ピンを用いて FFT の方向 (順・逆) の切り替えが行える．

**2 次元フーリエ平面の回転 (Rotation of 2D spectral plane)**　前もって作成しておいた参照画像の 2 次元フーリエ平面を，回転角 θ 及び θ +180 度回転させたものを別々の RAM に書き出す (それぞれを，フーリエ平面 1，フーリエ平面 2 と呼ぶ)．回転変換の際，Sine-Cosine Look-Up Table コアを使用した．

**積の行方向 1 次元逆 FFT(1D inverse FFT of product 1)**　フーリエ平面 1 と入力の 2 次元フーリエ平面との複素共役乗算と，FFT コアへのデータ転送とをパイプライン処理し，さらに行方向の逆 FFT 処理を行う．

**列方向 1 次元逆 FFT(1D inverse FFT of product 1)**　行方向の逆 FFT 処理結果に対して，列方向逆 FFT を施す．FFT コアからのデータ

| | |
|---|---|
| Handel-C コードからネットリストへの変換時間<br>(マッチングモジュール)<br>(PCI モジュール) | 約 14 分<br>(約 13 分)<br>(約 1 分) |
| 全ネットリストの配置配線時間 | 約 32 分 |
| 最高動作周波数 (worst case) | 約 50MHz(周期：約 32.3ms) |
| 最高動作周波数 (実動作) | 約 62MHz(周期：約 26.1ms) |
| 使用率 | 約 36%(7,005/19,200 Slices) |

表 2.2: ロジック生成結果

取り出しの際，最大値の検出もパイプライン処理で行う．

**行方向 1 次元逆 FFT(1D inverse FFT of product 2)**　フーリエ平面 2 と入力の 2 次元フーリエ平面との複素共役乗算と，FFT コアへのデータ転送とをパイプライン処理し，さらに行方向の逆 FFT 処理を行う．

**列方向 1 次元逆 FFT(1D inverse FFT of plane 2)**　行方向の逆 FFT 処理結果に対して，列方向逆 FFT を施す．FFT コアからのデータ取り出しの際，最大値の検出もパイプライン処理で行う．

### ロジック生成結果

Gateway GP8-1300(Pentium4 1.3GHz, 1280MB RDRAM) を用いた際のロジック生成結果を示す．

ロジック生成の際の PERIOD 制約として 50MHz 動作に要求される 20ns を指定した．また，外部 RAM 素子の動作タイミング及び FPGA-RAM 間の配線遅延を考慮し，OFFSET IN BEFORE 及び OFFSET OUT BEFORE 制約は，それぞれ 11.0ns，3.0ns を指定した [38]．配置配線の結果，これらの制約は満たされていたので，マッチングモジュール及び外部 RAM の 50MHz での動作が worst case においても保証されている．また，実動作の最高周波数は，マッチングモジュールの動作周波数を上げていった際に，同一テスト入力データに対して途中経過・出力結果の変化及び動作異常が無いことを確認することにより求めた．なお，ロジックの生成には，DK1

version 1.1 と，Foundation ISE 4.2 を用いた．

## 2.5　実験及び結果

### 2.5.1　実験システム

開発した FPGA 上のマッチトフィルタの位置合わせ能力を実画像を用いて検証した．実験システムは，システムの汎用性及び構築の容易さを考慮し，画像入力ボードを用いてカメラ画像をパソコンのメモリへ取り込み，パソコンで画像サイズの調整を行った後，FPGA ボードに入力画像を転送する構成を取った．実験システムに用いた機材を表 2.3 に，実験システムの接続図を，図 2.17 に示す．

ビデオレート (33[msec]) でのトラッキングを実現するためには，画像取込との同期，画像サイズ調整，入力画像転送とマッチングの全てを 33[msec] 以内に行う必要がある．そのため，DMA による FPGA ボードへの PCI データ転送と，約 60[MHz] での FPGA マッチングモジュールの駆動 (処理時間 27[msec]) を行った．パソコンへの画像取込の際，ダブルバッファリングと呼ばれる手法を用いて画像取込と画像処理を並列に行っている．また，実験用の OS として，リアルタイム OS ではないがユーザ処理の時間のばらつきが比較的少ない，Windows NT Embedded 4.0(Standard No Network, No Page File) を用いた．なお，今回の実験システムでは，カメラ画像の取込に 33[msec] を要するため，実際にカメラのシャッターが切られてからマッチングが完了するまでの時間は約 66[msec] となっている．

実験システムを，図 2.18 に示す．ターンテーブルに対象物体を乗せて，回転させながらビデオレートで画像入力し，開発したマッチトフィルタを実装した FPGA により位置・姿勢を計算しながら，トラッキングを行った．

| 名 称 | 製造元 |
|---|---|
| FPGA 開発ボード ADM-XRC | ALPHA DATA parallel systems |
| 画像取込ボード Meteor II/MC | Matrox |
| パソコン Precision 340 | DELL |
| OS Windows NT Embedded 4.0 | Microsoft |
| モノクロ CCD カメラ XC77RR | SONY |

表 2.3: 実験システム機材

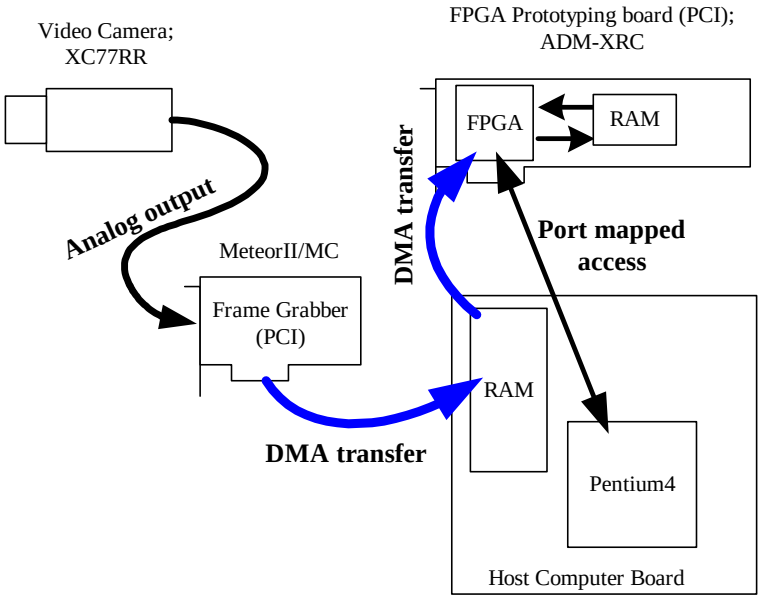



図 2.17: 実験システムの接続図

CCD カメラ (SONY 製，XC77RR) から，画像サイズが 256 × 256 画素 (256 × 240 の上下に 0 を付け加えた）の 256 階調の画像をビデオレート (33[msec]) で入力した．実験に用いた対象物体を，図 2.19，2.20 に示す．

### 2.5.2　実画像を用いた対象物のトラッキング

図 2.19，2.20 をテンプレートとして，ターンテーブル上の対象物体をトラッキングした結果を図 2.21∼2.24 に示す．図 2.19，2.20 の対象物体に関して，それぞれ約 10[sec] トラッキング実験をした．図 2.21 は，図 2.19 を 33[msec] 間隔でトラッキングした結果である．図 2.22 は，約 10[sec] 間のトラッキング結果を 200[msec] 間隔で表示した結果である．図 2.23，2.24 も同様である．どちらの対象物体に関しても，正確にトラッキングできていることがわかる．

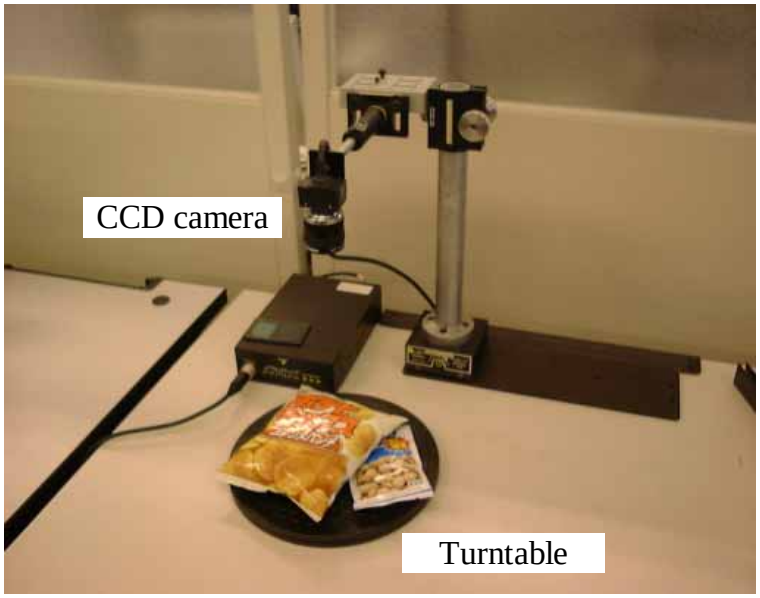



図 2.18: 実験システム


図 2.19: パターン 1 のテンプレート画像

図 2.20: パターン 2 のテンプレート画像

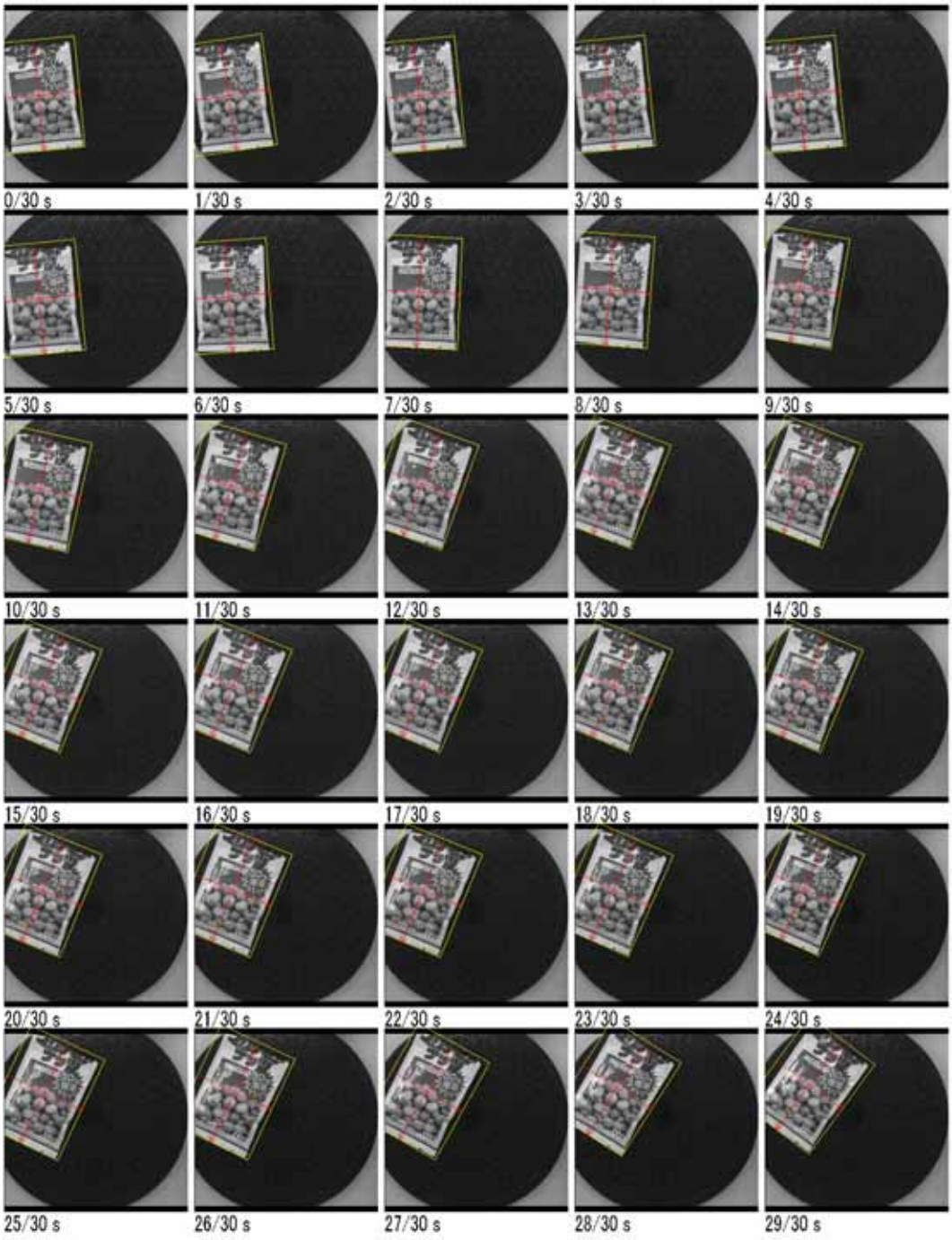



図 2.21: パターン 1 のトラッキング (1[sec])

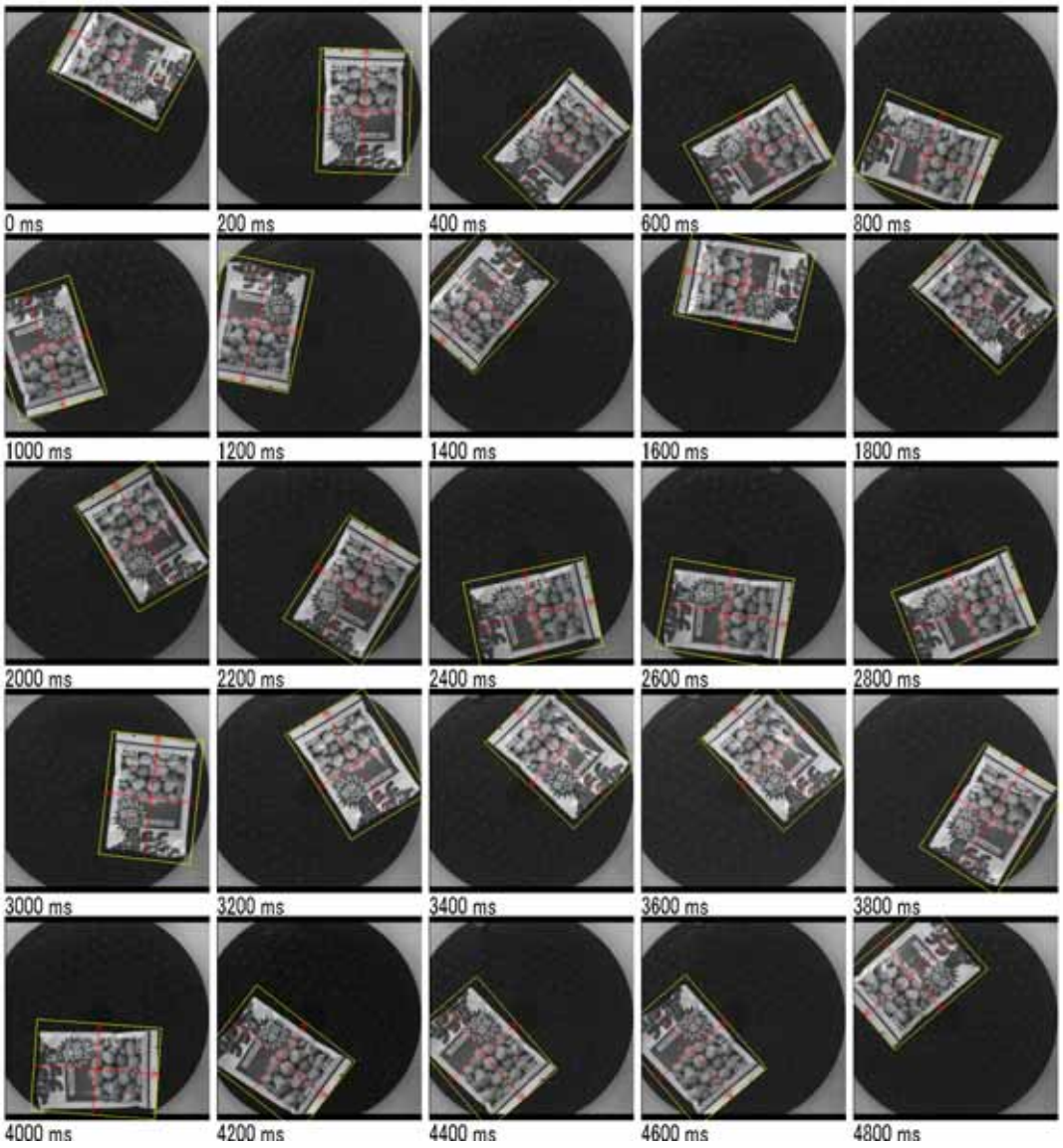



図 2.22: パターン 1 のトラッキング (4.8[sec])

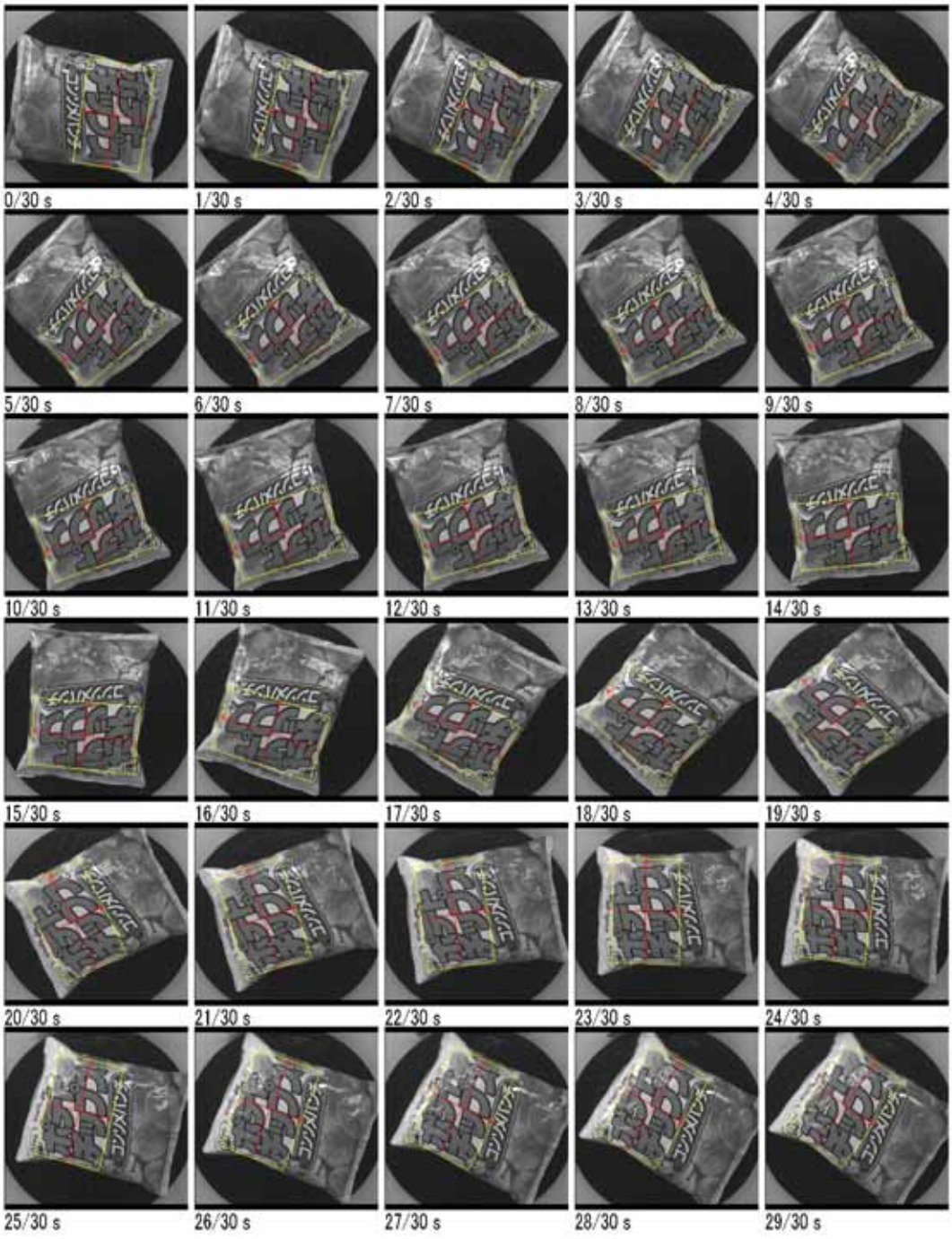



図 2.23: パターン 2 のトラッキング (1[sec])

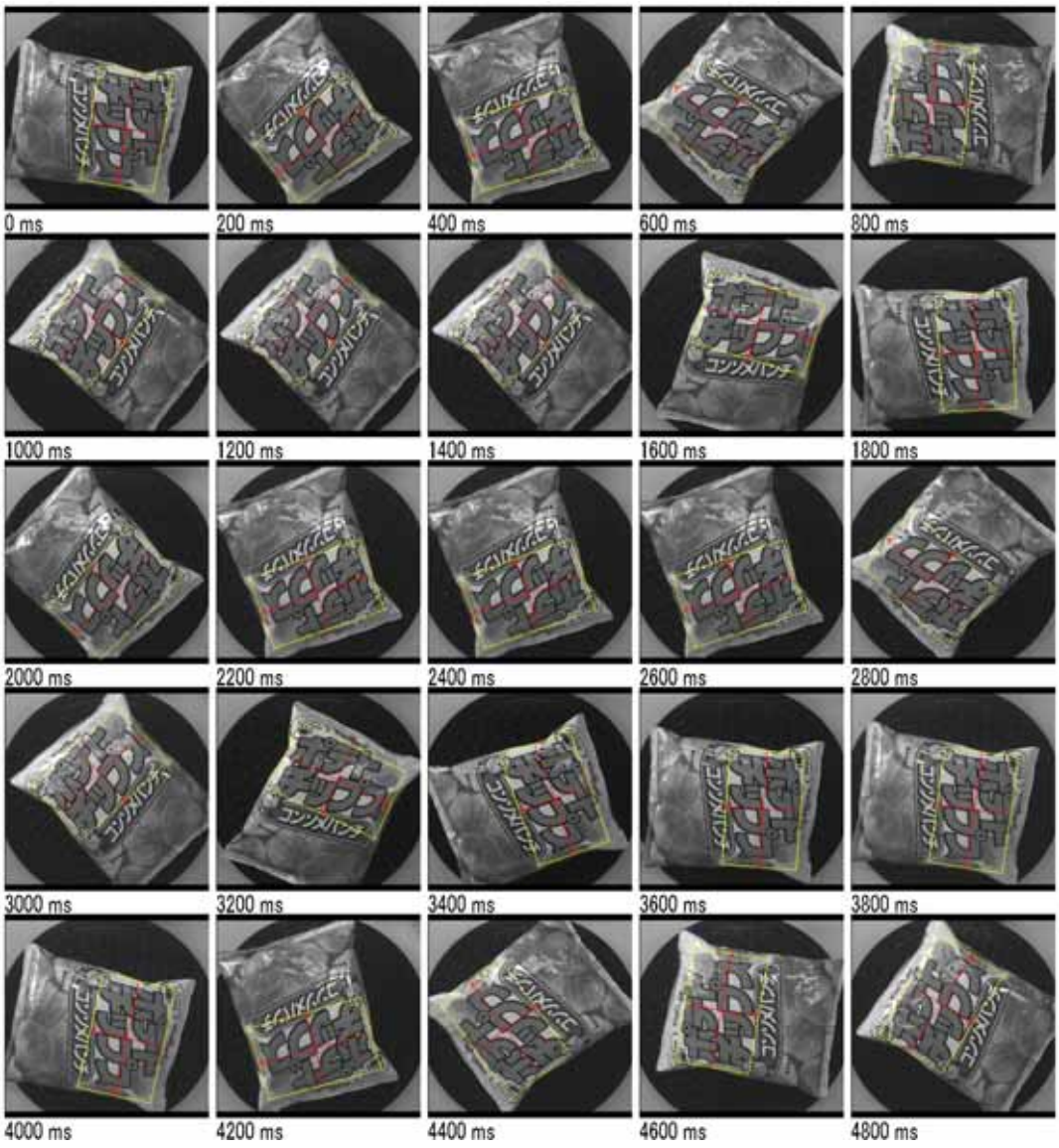



図 2.24: パターン 2 のトラッキング (4.8[sec])

## 2.6　画像検査への適用

### 2.6.1　印刷文字の検査

図 2.25，2.26 に示すように，食品の袋には，賞味期限，会社名，内容物等を示す各種文字・模様が印刷されている．印刷産業および食品業界においては，これらの印刷ミスを防ぐため，いろいろな工程において正常に印刷されているかどうかの検査が行われており，これらの工程の効率化が大きな課題となっている．袋の印刷工程においては，一部，画像処理を用いて検査の自動化が行われている．しかし，食品等の袋詰や搬送段階において印刷が汚れたり消えたりすることが起こるため，出荷直前の最終工程において検査をする必要があるものも多い．しかし，袋詰された食品等の印刷は，図 2.25，2.26 に示すように予め登録したパターンに対して歪みが生じているため，実用化が困難となっている．

### 2.6.2　パターンマッチングを利用した画像検査

本研究で開発した FPGA によるパターンマッチング機能を利用することにより，高速に食品の位置合わせを行うことができる．さらに，変形したパターン間の対応点を検出できる変形パターンマッチングを適用することにより，袋等の印刷された文字の歪みを考慮した検査を行うことが可能となる．

図 2.25(a) は位置合わせ用の参照画像である．参照画像と図 2.25(b) 間に開発した FPGA によるマッチトフィルタを適用して 33[msec] 以内に平行移動量と回転角を検出する．その結果，FPGA によるパターンマッチング機能に示すように入力画像を参照画像の座標系と同じになるように補正することができる．

文字検査の基準画像は，図 2.25(d) である．図 2.25(c) から対象となる文字領域を切り出すと図 2.25(e) となる．ここで，図 2.25(d) を変形を許さないでマッチングさせると図 2.25(g) のようになり，袋の歪みの影響で検査が困難なことがわかる．

そこで，変形を考慮した変形パターンマッチングを適用すると図 2.25(f) にように袋の歪みを考慮したマッチングが可能となることがわかる．図

2.25(d) 上の格子と図 2.25(e) 上の歪んだ格子は，それぞれ対応点を示している．図 2.26 も同様である．現状，FPGA によるパターンマッチング機能を用いて平行移動量と回転角を検出するところは実用レベルに達しているが，変形パターンマッチングはパソコン上の C 言語により実装されている．そのため，アルゴリズムの最適化と処理の高速化は今後の課題である．

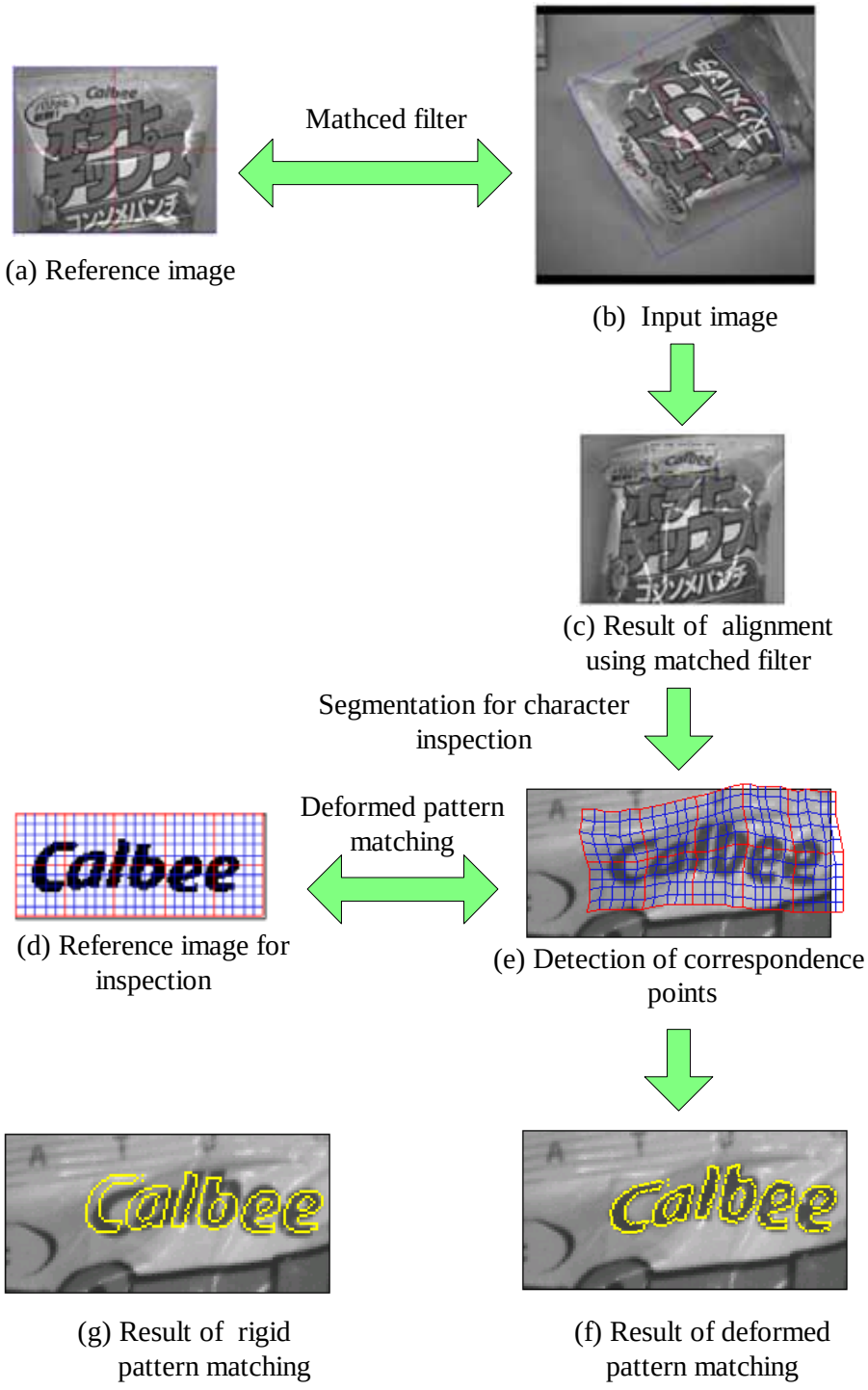

Mathced filter
(a) Reference image
(b) Input image
(c) Result of alignment using matched filter
Segmentation for character inspection
Deformed pattern matching
(d) Reference image for inspection
(e) Detection of correspondence points
(g) Result of rigid pattern matching
(f) Result of deformed pattern matching

図 2.25: 文字検査例 1

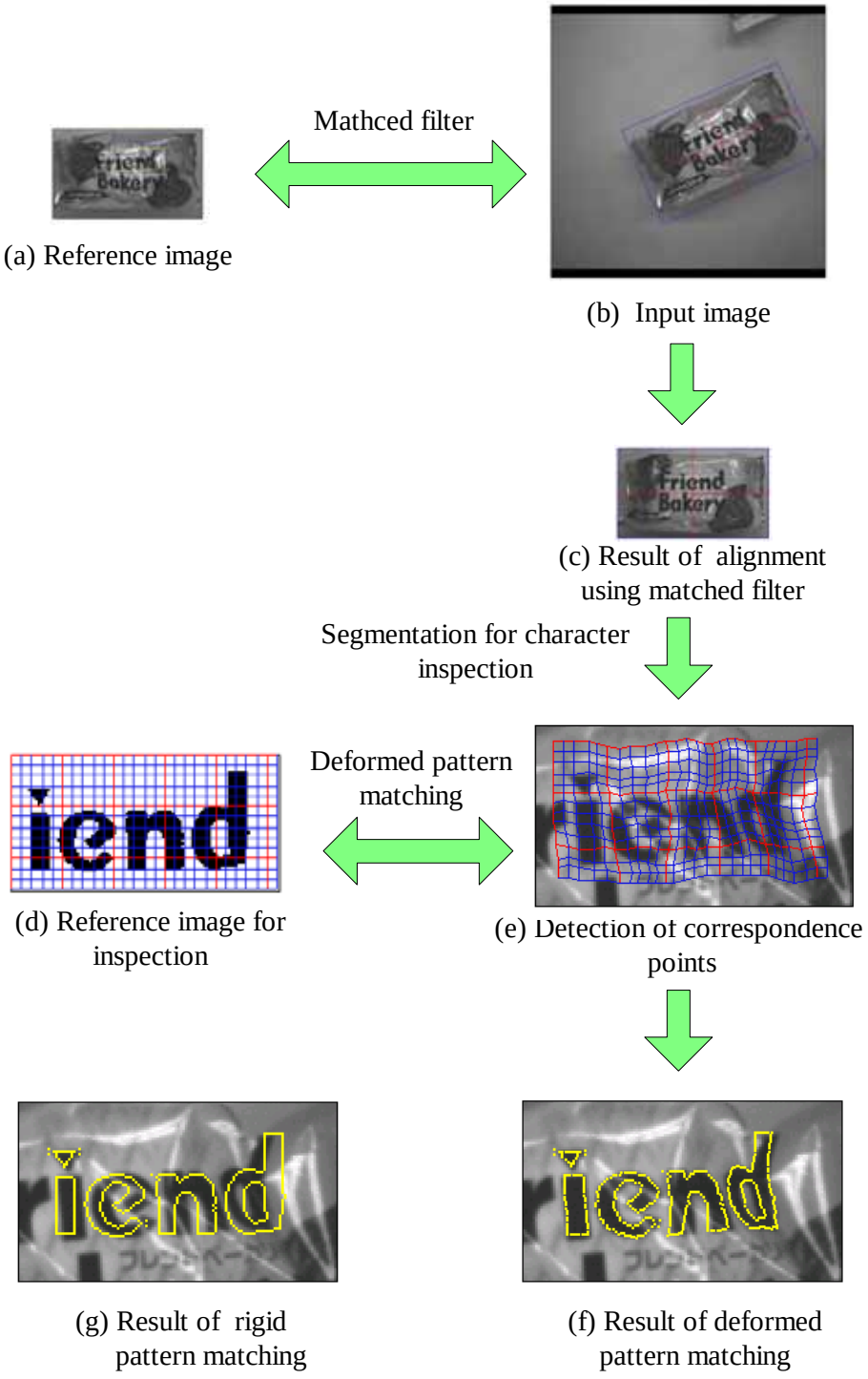

Mathced filter
(a) Reference image
(b) Input image
(c) Result of alignment using matched filter
Segmentation for character inspection
Deformed pattern matching
(d) Reference image for inspection
(e) Detection of correspondence points
(g) Result of rigid pattern matching
(f) Result of deformed pattern matching

図 2.26: 文字検査例 2

## 2.7　おわりに

本研究では，FPGA への実装に適したパターンマッチング手法として，マッチトフィルタを適用し，その最適化したアルゴリズムをパソコン上で検証した後，FPGA へ実装した．まず，マッチトフィルターを回転・平行移動したパターンのマッチングアルゴリズムとして，最適化を行い，パソコン上へ C 言語で実装し，位置・姿勢検出能力を検証した．さらに，ガウスラプラシアンフィルターを適用することにより，マッチトフィルターの能力が向上することを確認した．また，マッチトフィルター法のフーリエ変換の計算回数を削減するため，ハフ・フーリエ変換法の原理を利用したフーリエ・ラドン変換法を開発した．フーリエ・ラドン変換法を PC 上において C 言語で実装し，運動物体の位置・姿勢の検出が可能であることを検証した．

次に，PC 上で開発したパターンマッチングアルゴリズムを FPGA 上へ実装した．Handel-C や SpecC などの C 言語ベース LSI 設計ツールに関する調査を行った結果，Handel-C が有効であると判断し，Handel-C FPGA 開発環境 DK1 を導入した．Handel-C によりマッチトフィルター法の論理設計を行い，FPGA へ実装し，動作確認を完了した．FPGA ベースのリアルタイムビジョンシステムを構築した．200 万ゲート規模の FPGA（Xilinx 社 Virtex-E），外部 RAM，PC インターフェースを実装し，システムを構築した．

また，PC 上の画像入力装置と FPGA ボードを組合わせることにより，位置・姿勢の検出をビデオレート処理 (33msec) で実現した．

さらに，パターンマッチングアルゴリズムを実装した FPGA を用いて，画像検査への応用展開の可能性についても検証した．

最後に，今後の課題としては，変形物体の認識アルゴリズムの開発とハンドリングシステムとビジョンシステムとの協調動作アルゴリズムの開発が上げられる．

# 関連図書

[1] 有本 卓 : “信号・画像のデジタル処理,” 産業図書株式会社, 1980.

[2] D. H. Ballard : “Generalizing the Hough transform to detect arbitrary shapes,” Pattern Recognition, Vol. 12, No. 1, pp. 111–122, 1981.

[3] D. Casasent and D. Psaltis : “Position, rotation and scale-invariant optical correlation,” Applied Optics, Vol. 15, pp. 1793–1799, 1976.

[4] D. Casasent and D. Psaltis : “New optical transforms for pattern and recognition,” Proceedings of IEEE, Vol. 65, No. 1, pp. 77–84, January 1977.

[5] D. Casasent and R. Krishnapuram : “Curved object location by Hough transformations and inversions,” Pattern Recognition, Vol. 2, No. 2, pp. 181–188, 1987.

[6] Q. Chen, M. Defries, and F. Deconinck : “Symmetric phase-only matched filtering of Fourier-Mellin transforms for image registration and recognition,” IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 16, No. 12, pp. 1156–1166, December 1994.

[7] R. O. Duda and P. E. Hart : “Use of the Hough transformation to detect lines and curses in pictures,” Communications of the ACM, Vol. 15, No. 1, pp. 11–15, 1972.

[8] 藤本公三, 岩田剛治, 仲田周次：“$\theta - \rho$ ハフ変換平面からの 2 次曲線のパラメータ抽出,” 電子情報通信学会論文誌, Vol. J74–D–II, No. 9, pp. 1184–1191, 1991.

[9] 秦清治：“ロボットビジョンの産業への応用,” 日本ロボット学会誌, Vol. 10, No. 2, pp. 185–189, 1991.

[10] P. V. C. Hough： “Method and means for recognizing complex patterns,” U. S. Patent, 3069654, 1962.

[11] 加藤敦彦：“グレースケール画像処理の動向とその応用,” 電子技術, pp. 1057–1064, 1990.

[12] 川上進, 岡本浩明：“局所運動を検出する大脳 Magnocelluar 系視路細胞のモデル,” 電子情報通信学会論文誌, Vol. J78–D–II,No. 1, pp. 147–157, 1995.

[13] 黒木義博, 長田康行：“視覚ロボットによる位置決め,” 日本ロボット学会誌, Vol. 9, No. 3, pp. 369–373, 1991.

[14] 輿水大和：“Hough 変換に関する最近の研究動向,” 情報処理学会研究会資料, CV51–1, pp. 1–8, 1987.

[15] V. F. Leavers： “Survey： Which Hough transform ?,” CVGIP： Image Understanding, Vol. 58 , No. 2, pp. 250–264, 1993.

[16] 松山隆司, 輿水大和：“Hough 変換とパターンマッチング,” 情報処理学会論文誌, Vol. 30, No. 9, pp. 1035–1046, 1989.

[17] 中山亨, 河田聡, 南茂夫：“フーリエ位相相関法を用いたパターン識別,” 第 22 回画像工学コンファレンス論文集, pp. 89–92, 1991.

[18] 沼田宗敏, 輿水大和：“Gradient 型超高速型 Hough 変換アルゴリズム,” 情報処理学会研究会資料, CV51-2, pp. 1–8, 1987.

[19] F. O’Gorman and M. B. Clowes： “Finding picture edges through collinearity of feature points,” IEEE Transactions on Computers, Vol. 25, No. 4, pp. 449–456, April 1976.

[20] H. Onishi and H. Suzuki : “Detection of rotation and parallel translation using Hough and Fourier transforms,” Proceedings of 1996 IEEE International Conference on Image Processing(ICIP-96), Vol. 3, pp. 827–830, 1996.

[21] H. Onishi and H. Suzuki : “Detection of magnification, rotation, and parallel translation using Hough and Fourier-Mellin transforms,” in Optical Pattern Recognition VIII, David P. Casasent, Tien-Hsin Chao, Editors, Proc. SPIE 3073, pp. 244–254, 1997.

[22] 大西弘之, 鈴木 寿 : “$\theta - \rho$ ハフ変換とフーリエ変換を用いたパターンマッチング,” 画像の認識・理解シンポジウム (MIRU'96) 講演論文集, Vol. 1, pp. 355–360, 1996.

[23] 大西弘之, 鈴木 寿, 有本 卓 : “ハフおよびフーリエ変換を用いた回転と平行移動の検出,” 電子情報通信学会論文誌, Vol. J80–D–II, No. 7, pp. 1668–1675, 1997.

[24] 大西弘之, 鈴木 寿, 有本 卓 : “ハフおよびフーリエ変換を用いた拡大・回転・平行移動検出法の部品位置決めへの応用,” 日本ロボット学会誌, Vol. 16, No. 2, pp. 98–106, 1998.

[25] D. C. W. Pao, H. F. Li, and R. Jayakumar : “Detecting parametric curves using the straight line Hough transform,” Proceedings of Intelligence Conference on Pattern Recognition, pp. 620–625, 1990.

[26] D. C. W. Pao, H. F. Li, and R. Jayakumar : “Shapes recognition using the straight line Hough transform: Theory and generalization,” IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 14, No. 11, pp. 1076–1089, November 1992.

[27] T.L. Peretz : “Recognition and localization of overlapping parts from sparce data,” 2nd ISRR, pp. 28–33, 1984.

[28] 斎藤文彦 : “一般化ハフ変換による半導体ウエハ識別番号認識,” 精密工学会誌, Vol. 61, No. 10, pp. 1470–1474, 1995.

[29] S. Tuji and F. Matsumoto : “Detection of ellipses by a modified Hough transformation,” IEEE Transactions on Computers, Vol. c-27, No. 8, pp. 777–781, August 1978.

[30] 辻内順平, 村田和美 : “光学情報処理,” 朝倉書店, 1974.

[31] G. L. Turin : “An introduction to matched filters,” IRE Transactions Information Theory, IT–6(3), pp. 311–329, 1960.

[32] G. L. Turin : “An introduction to digital matched filters,” Proceedings of the IEEE, Vol. 64, No. 7, pp. 1093–1112, July 1977.

[33] 和田俊和, 松山隆司 : “Hough 変換に基づく図形検出法の新展開,” 情報処理学会論文誌, Vol. 36, No. 3, pp. 253–263, 1995.

[34] L. Xu and E. Oja : “Randomized Hough transform(RHT) : Basic mechanisms, algorithms, and computational complexities,” CVGIP : Image Understanding, Vol. 57 , No. 2, pp. 131–154, 1993.

[35] 山田啓一, 中野倫明, 山本新 : “照明変動にロバストな部品位置決め用ロボット視覚システム,” 日本ロボット学会誌, Vol. 14, No. 6, pp. 911–914, 1996.

[36] 安永 守利, 高見 知親, 吉原 郁夫 : “FPGA を用いたナノ秒オーダー画像認識ハードウェア,” 電子情報通信学会論文誌, Vol. J84–D–II, No. 10, pp. 2280–2292, 2001.

[37] Celoxica Ltd. : “Sobel Edge Detect program,” RC100 User Manual Version 1.2, pp. 34–46, 2001.

[38] 東京エレクトロンデバイス株式会社 : “UCF について,” アドバンスド Virtex/E・Spartan2 コース Rev.5.0, pp. 5-1–5-11, 2001.

## 成果発表および特許等の状況

以下，画像マッチング方式関して特許出願済み.

１．出願国 ：　日本

２．出願に係る工業所有権の種類　：　特許権

３．発明等の名称　：　画像マッチング方法、画像マッチング装置
および画像マッチングプログラム

４．発明者　：　大西弘之

５．出願日　：　平成 14 年 3 月 1 日

６．出願番号　：　特願 2002 － 56653

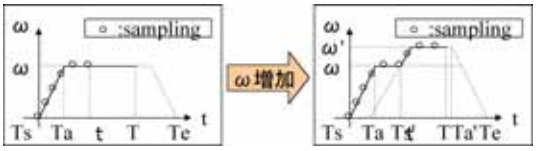
７．出願人　：　グローリー工業株式会社

# 第3章　ハンドリングシステムの構築

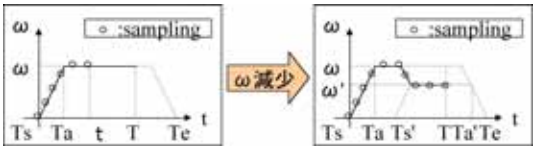
## 3.1　はじめに

本章では，視覚情報を用いたバラ積み運動物体のハンドリングシステムを構築する．前章で述べたビジョンシステムを用いることにより，運動物体の位置と姿勢をビデオフレームレートで検出することができる．検出情報をハンドリング装置に送ることにより，ハンドリング装置がバラ積み運動物体に追従し，把持することが可能になる．本章では，ハンドリング装置をバラ積み運動物体に追従させる手法として，速度追従法を提案する．速度追従法の概略を述べるとともに，バラ積み運動物体のハンドリングを実現した結果を示す．

## 3.2　速度追従法によるリアルタイム軌道計画




(a) increasing angular velocity




(b) decreasing angular velocity

図 3.1: 速度追従法による角速度軌道の計画

本節では，リアルタイムで計測される運動物体の位置と姿勢に応じて，マニピュレータの運動軌道をリアルタイムで計画する手法を提案する．ま

ず，マニピュレータが物体を把持する時刻 $T$ を設定する．時刻 $T$ を設定することにより，ビジョンシステムから得られる移動物体の位置と姿勢から，把持時に移動物体が有する位置・姿勢と速度・角速度を計算できる．片側ラドン変換を実装したビジョンシステムを用いて，移動物体の位置・姿勢 $P_{obj}(t)$ と 速度・角速度 $V_{obj}(t)$ を，ビデオフレームレートで計算する．移動物体が等速度・等角速度運動を行うと仮定すると，時刻 $T$ における移動物体の位置・姿勢 $P_{obj}(T)$ と速度・角速度 $V_{obj}(T)$ は，次式により計算できる．

$$P_{obj}(T) = P_{obj}(t) + V_{obj}(t)(T-t),$$
$$V_{obj}(T) = V_{obj}(t).$$

把持時にハンドと移動物体との間に発生する衝撃を小さくするために，時刻 $T$ において，ハンドの速度・角速度 $V_{hand}(T)$ が移動物体の速度・角速度 $V_{obj}(T)$ に追従するように制御する．さらに，計算された $P_{obj}(T)$ から，時刻 $T$ におけるハンドの位置・姿勢 $P_{hand}(T)$ を求める．マニピュレータの関節角度を $\theta(t)$，関節角速度を $\omega(t)$ で表す．ハンドの位置・姿勢 $P_{hand}(T)$ とハンドの速度・角速度 $V_{hand}(T)$ から，時刻 $T$ における関節角度 $\theta(T)$ と関節角速度 $\omega(T)$ を求めることができる．

時刻 $T$ における関節の角度 $\theta(T)$ と角速度 $\omega(T)$ が与えられるので，角速度軌道 $\omega(t)$ を計画する．角速度計画では等脚台形を用いる．等脚台形を生成するためには，台形の高さ $\omega(T)$，台形の傾き $\alpha = \omega(T)/T_a$，台形の始点 $T_s$，台形の終点 $T_e(\geq T)$ の 4 個のパラメータを決定すればよい．ここで，$T_a$ は加速時間であり，台形の傾き $\alpha$ は関節角の角加速度に相当する．画像情報より新たに $\omega(T)$ と $\theta(T)$ が算出されると，台形の高さ $\omega(T)$ が更新される．ここで，簡単のために台形の傾きを一定とすると，残り時間 $T-t$ における角度の移動量 $\theta(T)-\theta(t)$ と底辺の長さの関係より，$T_s$ が求められる．更新された 4 個のパラメータが，残り作業時間 $T-t$ における速度台形を生成する．この角速度台形の更新を画像情報が獲得されるたびに行い，リアルタイムに角速度台形を更新する．たとえば，時刻 $t$ において画像情報を取得した結果，時刻 $T$ における角速度 $\omega(T)$ が増加した場合，図 3.1-(a) に示すように角速度軌道が更新される．一方，時刻 $T$ における角速度 $\omega(T)$ が減少した場合，図 3.1-(b) に示すように角速度軌道

が更新される．以上の手法を，速度追従法とよぶ．

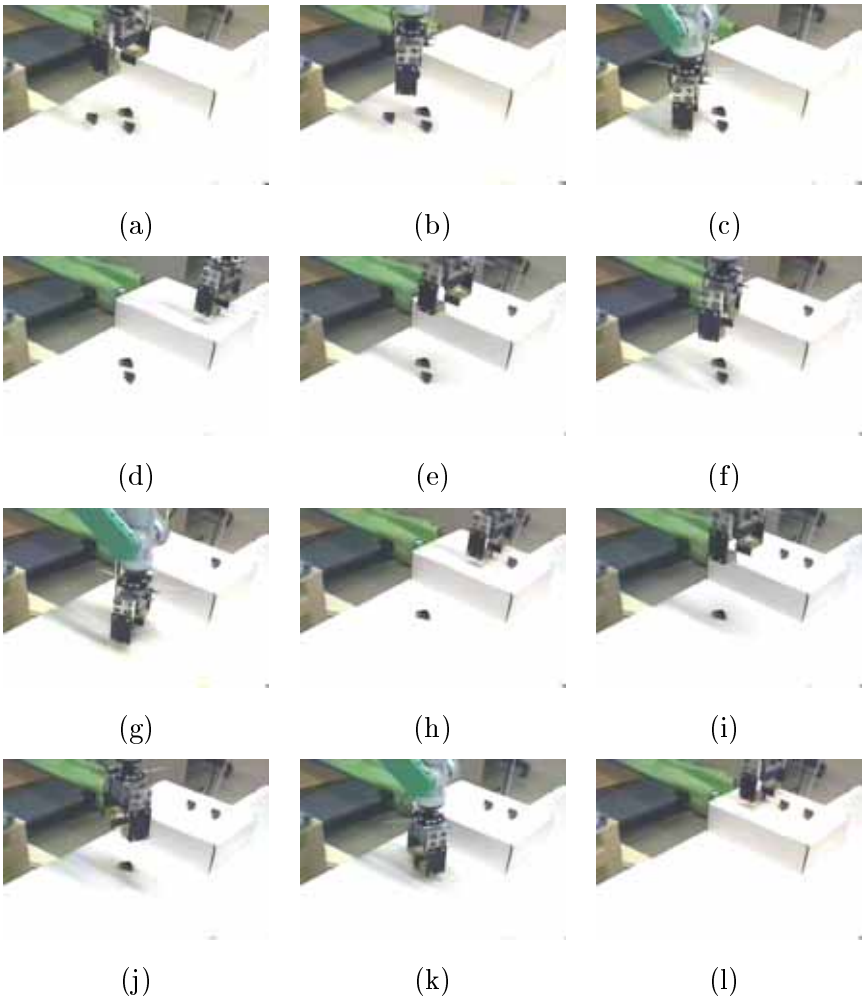



図 3.2: バラ積み物体のピックアンドプレース

## 3.3　バラ積み平面運動物体のハンドリング

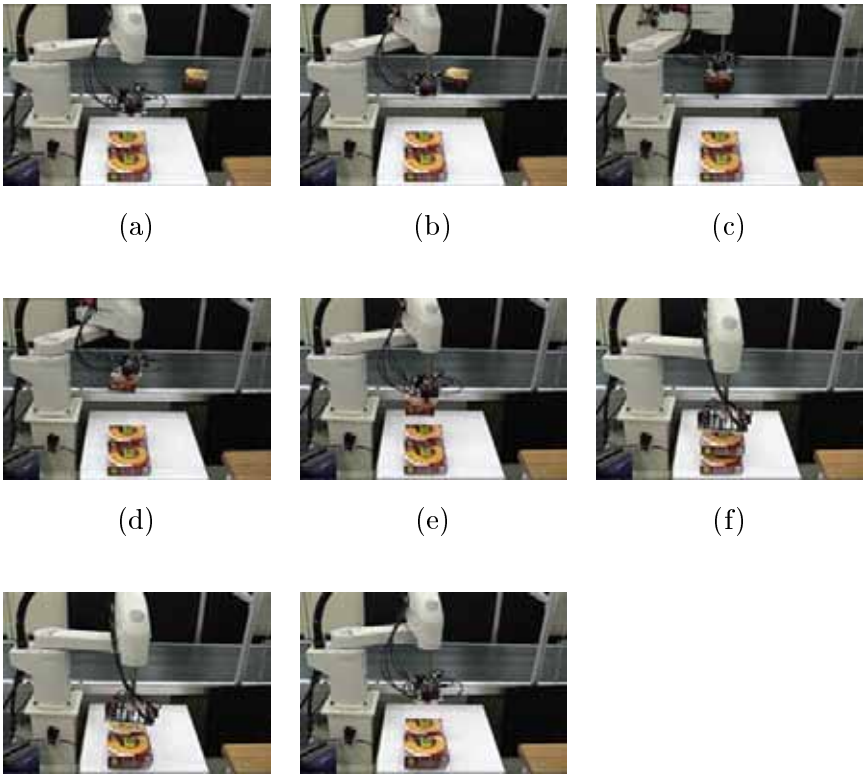
片側ラドン変換法による物体の平面運動計測と速度追従法によるリアルタイム軌道計画を適用することにより，バラ積み物体のハンドリングを実現することができる．図 3.2 に，その例を示す．この例では，7 軸マニピュレータ PA-10 の 4 軸を用いて，スカラロボットと同等の自由度を実現している．各関節をトルク指令モードで駆動する．速度追従法で計算さ

れた各関節の角度軌道に対して，重力補償付き PD 制御を適用し，各関節のトルク指令を計算している．図に示すように，位置と姿勢が異なる物体を把持し，物体を整列させることができる．

図 3.3 に，バラ積み運動物体の整列を実現した例を示す．この例では，4 軸スカラロボット RH-5AH55 を用いている．スカラロボット RH-5AH55 は，軌道指令モードで駆動されるので，速度追従法で算出した軌道をサンプリングして，与える指令軌道を生成する．物体の運動をベルトコンベアで与える．物体は等速度・等角速度運動をすると仮定し，検出した物体の位置・姿勢から速度と角速度を推定している．特に，物体の姿勢が 45° の場合を図 3.4 に，物体の姿勢が 150° の場合を図 3.5 に示す．物体の姿勢に応じてスカラロボットのハンド部が回転し，物体の整列を実現していることがわかる．

## 3.4　おわりに

本章では，視覚情報を用いたバラ積み運動物体のハンドリングシステムを構築し，バラ積み運動物体の整列を実現した．今後の課題は，1) ハンドリングの高速化，2) 物体の速度と角速度が変化する場合への拡張である．

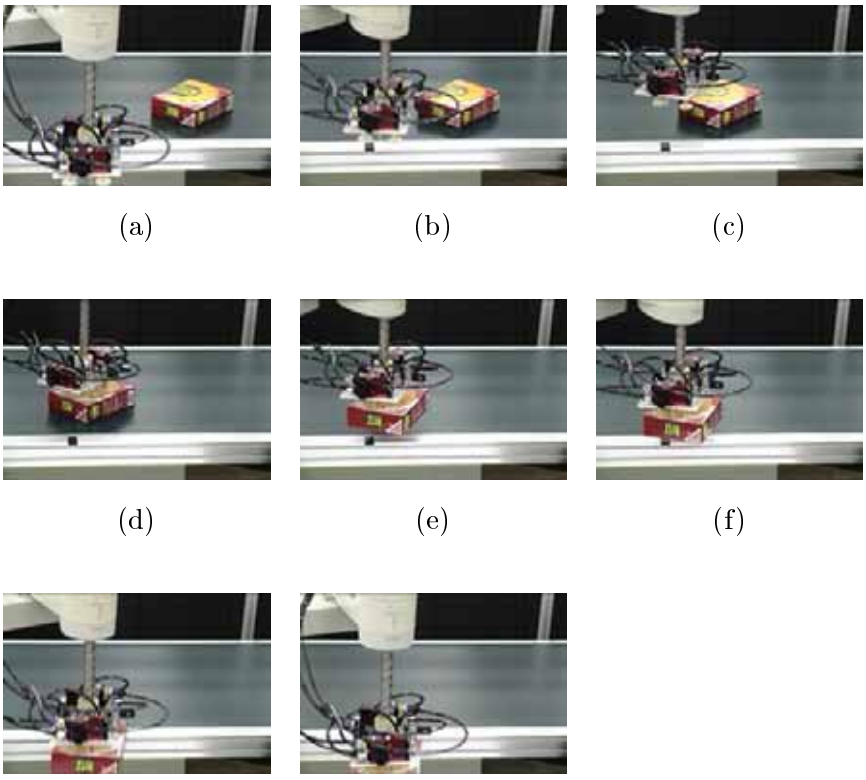


(a)　(b)　(c)

(d)　(e)　(f)

(g)　(h)

図 3.3: バラ積み運動物体の整列

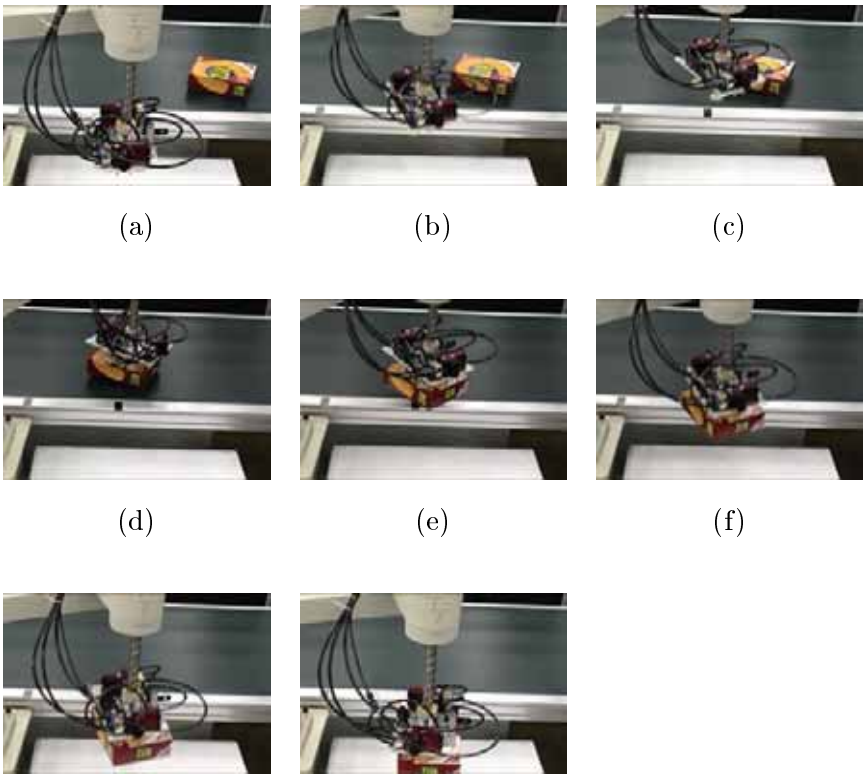


(a)　(b)　(c)

(d)　(e)　(f)

(g)　(h)

図 3.4: 物体の整列 (姿勢 45°)



(a)　(b)　(c)

(d)　(e)　(f)

(g)　(h)

図 3.5: 物体の整列 (姿勢 150°)

## 成果発表および特許等の状況

1. 増渕, 平井, ”ロボット搭載用ビジョンモジュールを用いた平面運動物体ハンドリング”, ロボティクス・メカトロニクス’01 講演会予稿集 CD-ROM, 2001

2. 柴田, 平井, ”視覚ベースハンドリングのための速度追従法によるリアルタイム軌道計算”, 計測自動制御学会システムインテグレーション部門学術講演会予稿集, pp.141–142, 2001