

Matched Filter を基にしたビジョンアルゴリズムの FPGA 実装

山本 紘督 森原 浩之 平井 慎一 (立命館大学)

Implementation of Vision Algorithm based on Matched Filter on FPGA

*Kosuke Yamamoto, Hiroyuki Morihara, Shinichi Hirai Ritsumeikan Univ.

Abstract- This paper, we will implement a vision algorithm based on matched filter on FPGA's. A vision algorithm based on matched filter can detect the position and the orientation of planer motion object, and can detect robustly against occlusion and background of an object image. But this algorithm requires much computation time for 2D-FFT and polar transformation involved in the algorithm. Its implementation on an FPGA is expected to reduce the computation. Thus, we will implement this algorithm on an FPGA to develop a real time vision chip. We will design a logic circuit of the algorithm and will verify the designed circuit.

Keywords: FPGA, matched filter, real time, vision

1 はじめに

生産ライン等におけるハンドリング制御を、画像処理技術を用いるビジョンシステムで実現するには、運動物体の位置と姿勢を検出する技術が必要となる。このようなビジョンシステムを実現させるためには、運動物体の位置と姿勢を検出する処理の高速性と物体の環境に対応するロバスト性が要求される。画像情報の入力にビデオ信号を用いる場合、リアルタイム性を保つには 33ms 以内での画像処理が求められる。一般に画像処理の演算量と性能はトレードオフの関係にあり、性能を追求するビジョンシステムでは演算量が多くリアルタイム性が犠牲になっていることが多い [1]。本研究で用いる Matched Filter を用いたアルゴリズムもその一つであり、二次元フーリエ変換や極座標変換等、演算量が多い処理を含んでいる。一方、これらの処理は並列性が高いため、並列処理が可能である FPGA に実装することで、性能を保持したまま処理演算時間の短縮と高速化を実現することが期待できる。そこで本研究ではこのアルゴリズムを FPGA に実装し、性能とリアルタイム性を両立したビジョンシステムの開発を目指す。

2 画像処理アルゴリズム

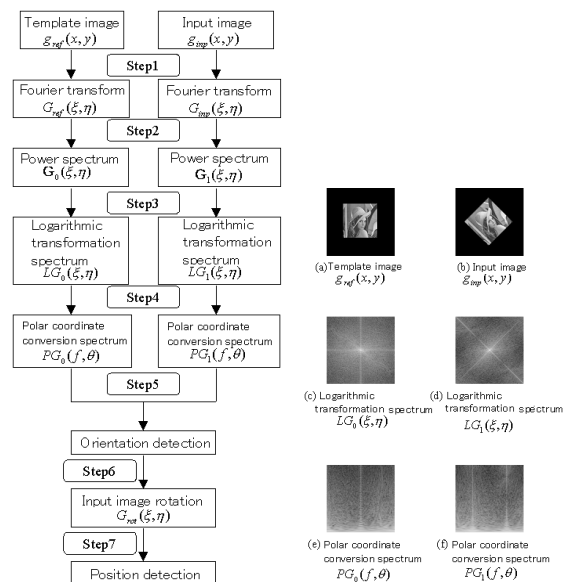
Matched Filter は画像の運動変位を求める手法であり、回転変位と位置変位を求めることができる [2]。本節では、Matched Filter を用いたアルゴリズムの流れに関して述べる。Fig.1 にアルゴリズムのフローと、実際のフローにおけるスペクトルの様子を示す。

Algorithm

Step1 テンプレート画像 $g_{ref}(x, y)$ と入力画像 $g_{inp}(x, y)$ を 2 次元高速フーリエ変換し、 $G_{ref}(\xi, \eta)$ 、 $G_{inp}(\xi, \eta)$ を求める。

Step2 2次元フーリエ変換 $G_{ref}(\xi, \eta)$ 、 $G_{inp}(\xi, \eta)$ からパワースペクトル $G_0(\xi, \eta)$ 、 $G_1(\xi, \eta)$ を得る。

Step3 パワースペクトル $G_0(\xi, \eta)$ 、 $G_1(\xi, \eta)$ の DC 成分が画像の中心にくるように配置し、対数変換を行い $LG_0(\xi, \eta)$ 、 $LG_1(\xi, \eta)$ を得る。



(a) Vision algorithm flow (b) Spectrum image

Fig.1: Vision algorithm based on matched filter

Step4 $LG_0(\xi, \eta)$ 、 $LG_1(\xi, \eta)$ の DC 成分を中心に 0 度 ~ 180 度の範囲で極座標変換を行い $PG_0(f, \theta)$ 、 $PG_1(f, \theta)$ を得る。

Step5 $PG_0(f, \theta)$ 、 $PG_1(f, \theta)$ について、1 次元 Matched Filter を用いて θ 方向のマッチングを行う。すなわち、各 f に対して、 $PG_0(f, \theta)$ と $PG_1(f, \theta)$ の θ に関する 1 次元フーリエ変換を求め、それらの商を 1 次元逆フーリエ変換し、角度 θ の移動量を求める。すべての周波数 f に対して、角度 θ の移動量を求め、投票により投票数が最大となる角度 α を求める。角度 α が回転角度である。

Step6 入力画像 $g_{inp}(x, y)$ に対して、検出された回転角度分をテンプレート画像と同じ姿勢になるように回転させ、その画像に 2 次元高速フーリエ変換を行い、 $G_{rot}(\xi, \eta)$ を得る。

Step7 $G_{rot}(\xi, \eta)$ 、 $G_{ref}(\xi, \eta)$ に対し 2 次元 Matched Filter を用いてマッチングを行う。 $G_{rot}(\xi, \eta)$ を $G_{ref}(\xi, \eta)$ で除

算し、2次元逆高速フーリエ変換を行い並進移動量を検出する。

この画像処理アルゴリズムは画像の姿勢（回転角度）を検出し、位置（並進移動量）を検出する。上記のアルゴリズムの流れのうち、Step 2からStep 5が回転角度検出部、Step 6とStep 7が並進移動量検出部である。

3 回路設計

前述の Matched Filter を基にした画像処理アルゴリズムを FPGA に実装するために機能単位（モジュール）ごとに設計する。設計はハードウェア記述言語である Verilog-HDL および System Compiler を用いた C++言語ベースで行う [3]。System Compiler とは C++言語ソースを Verilog-HDL へと変換するソフトウェアである。C++ベース設計の特徴は、通常のコンパイラでコンパイルが可能であり、Verilog-HDL と同じ動作が保証される点にある。したがって、演算量が多い処理に対して高速にシミュレーションを行うことが可能であり、テストパターンの生成にも PC の既存のソースを用いることができる。設計した画像処理アルゴリズムの機能ごとのモジュール構成を Fig.2 に示す。入力画像は SRAM フレームメモリへ、1フレームごとに交互に格納される。一方の SRAM フレームメモリに入力画像が格納されている間、他方の SRAM フレームメモリに格納された1フレーム前の画像に対して処理が行われ、位置と姿勢が求められる構成である。テンプレート画像のデータである2次元フーリエ変換と極座標変換スペクトルは、内部のメモリに保持しておく。

3.1 対数変換モジュール

対数変換モジュールは、入力 x に対して、出力 $\log(1+x)$ を求める。対数の値は、折れ線近似により計算する。すなわち、入力 x が区間 $[x_k, x_{k+1}]$ に含まれるとき、対数の値を $a_k x + b_k$ で近似する。ただし、各区間に対する a_k と b_k の値は、あらかじめ求めておく。シミュレーション結果を Fig.3 に示す。

3.2 極座標変換モジュール

極座標変換モジュールは入力画像に対して極座標変換を行い変換結果を出力する。入力値として極座標系の r と θ を入力し、それに対応する直交座標系の x, y を出力する。また、極座標変換で用いる \sin と \cos の値は、テーブルルックアップにより求める。

3.3 幾何変換モジュール

幾何変換モジュールは入力画像に対して幾何変換を行い、画像を回転させ変換結果を出力する。極座標変換モジュールと同様に \sin と \cos の値は、テーブルルックアップにより求める。

3.4 高速フーリエ変換モジュール

1次元高速フーリエ変換モジュールはIPコアとして用意されている256点FFTコアとDual Port RAMを用いて設計する。しかし、今回画像サイズ 128×128 を処理するので128点FFTコアが必要であるが、IPとして用意されていないため、256点FFTコアを用いて128点FFTモジュールを設計する。128点高速フーリエ変換の結果の値は、256点高速フーリエ変換の結果の偶数番目の成分に一致する。そこで、256点高速フーリエ変換結果の偶数番目のみを抽出することで、128

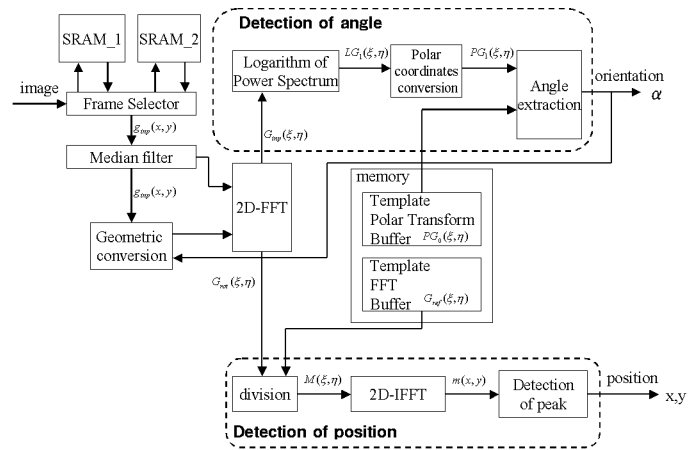


Fig.2: Circuit design overview

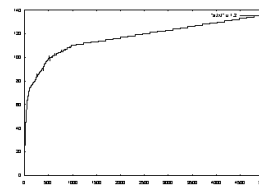


Fig.3: Simulation result of log module

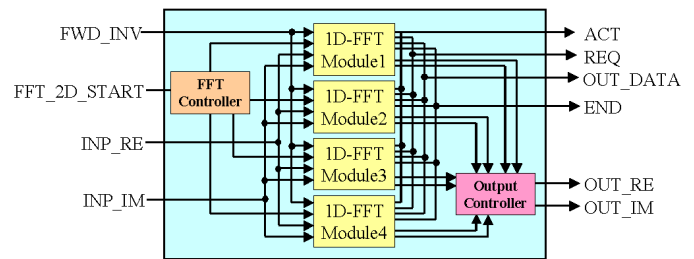


Fig.4: 2D-FFT module

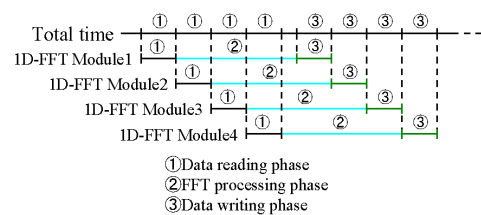
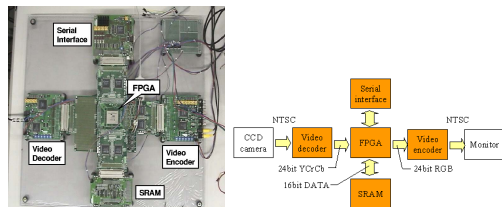


Fig.5: Timing chart

点高速フーリエ変換を実現する。この1次元高速フーリエ変換モジュールを用いて、2次元高速フーリエ変換モジュールを設計する。2次元高速フーリエ変換モジュールは、画像データに対して、縦方向・横方向とフーリエ変換を行う。そのため処理時間が多くかかってしまうので、1次元高速フーリエ変換モジュールを4つ使ったパイプライン処理で設計し、高速化を実現する。設計構成を Fig.4 に示す。パイプライン処理の際のタイミングチャートは Fig.5 のようになる。図中の ① がデータ読み込みフェーズ、② が高速フーリエ変換処理フェーズ、③ がデータ書き出しフェーズである。



(a) Appearance of vision board system (b) Signal flow

Fig.6: Vision system

3.5 回転角度検出モジュール

極座標変換した画像データから、1次元 Matched Filter を用いて回転角度を検出する。極座標変換した画像データの各 f ごとに θ に関する 1次元 Matched Filter を行い、検出される各 f ごとのピーク値に対応する角度を投票し、投票数が最大値となる角度を求める。ここでは、1次元高速フーリエ変換モジュールと除算モジュールを用いて設計する。

4 実装

本研究で用いるビジョンシステムボードと、ボード上の信号の流れを Fig.6 に示す。このボードは、写真科学製の CM59(Xilinx 製 VirtexE 2000 搭載) である。大規模 FPGA ボードを中心に、三菱マイコン機器ソフトウェア製の MU200-VDEC(ビデオデコーダ)、MU200-VENC(ビデオエンコーダ)、MU200-SRAM、MU200-EX40(FPGA 学習用ボード) を接続したものである。映像入力には CCD カメラからの NTSC ビデオ信号である。

実際に極座標変換モジュールを実装した結果を Fig.7 に、幾何変換モジュールを実装した結果を Fig.8 に示す。極座標変換する対象画像を Fig.7-(a)、Fig.7-(b) に、出力結果を Fig.7-(c)、Fig.7-(d) にそれぞれ示す。極座標変換の結果より、画面の中心を原点に 0 度 ~ 180 度 (画像右半分) に対して行われていることが確認できる。幾何変換については、1 フレームごとに 1 度づつ回転させている時の画像の様子を示している (Fig.8)。画像の中心を原点に回転している。これらのモジュールを実装させた場合の実装面積 (使用スライス数) は、極座標変換モジュールでは 276/19,200 全体の 1%、幾何変換モジュールでは 1,453/19,200 全体の 7% となった。

5 おわりに

今回、画像処理アルゴリズムにおける各モジュールの設計と検証を行った。また、FPGA ビジョンボードに極座標変換と幾何変換を実装し、それぞれのモジュールにおいてリアルタイムでの処理が確認できた。今後は設計・検証ができていない残りのモジュールを接続し、Matched Filter を基にした画像アルゴリズムを完全実装させ、リアルタイムビジョンシステムを構築する。

参考文献

[1] 座光寺正和, 増淵章洋, 坪井辰彦, 平井慎一: “片側ラドン変換を基にしたビジョンアルゴリズムの FPGA 実装”, 日本機学会ロボティクスメカトロニクス講演会 講演論文

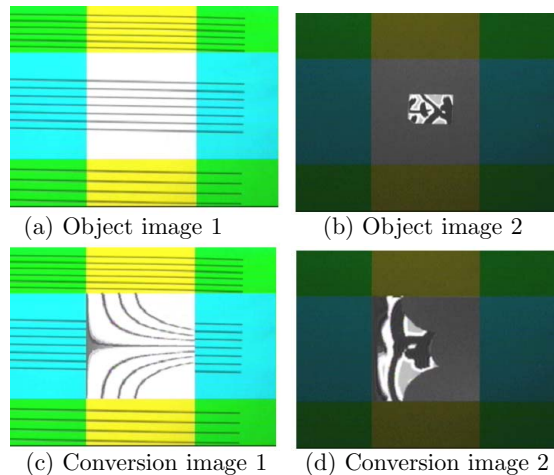


Fig.7: Polar coordinate conversion implementing result

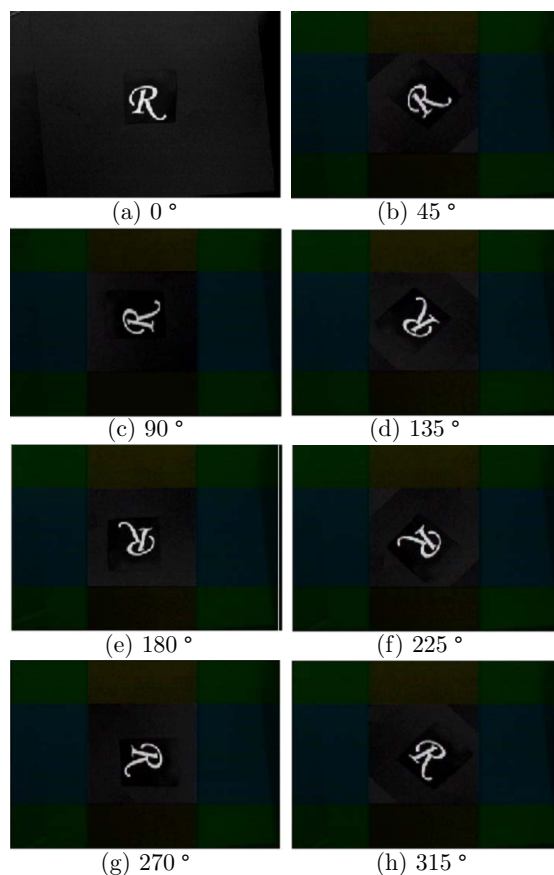


Fig.8: Geometry conversion implementing result

集 CD-ROM, 2002.

[2] Casasent, D., Psaltis, D.: “New Optical Transforms for Pattern and Recognition”, Proceedings of IEEE, Vol. 65, No. 1, pp.77-84, January, 1977.

[3] 平井慎一, 座光寺正和, 坪井辰彦: “FPGA ベースリアルタイムビジョン”, ロボティクスシンポジウム 予稿集 pp.99-105, 2003.