FPGA ベースリアルタイムビジョン

○平井 慎一, 座光寺 正和, 坪井 辰彦
立命館大学 ロボティクス学科

FPGA-based Realtime Vision

○ Shinichi Hirai, Masakazu Zakouji, and Tatsuhiko Tsuboi
Dept. of Robotics, Ritsumeikan Univ.

Abstract - *We will describe a realtime vision system based on FPGA's. First, we will show the developed FPGA-based vision system. Secondly, we will implement three vision algorithms; computation of the image gravity center, computation of a projection along half lines, and Hough transform. We will then demonstrate the realtime detection of the position and the orientation of a planar motion object using the FPGA-based realtime vision system.*
keywords: vision, realtime, FPGA, planar motion detection, Hough transform

1. はじめに

本論文では，FPGA (Field Programmable Gate Array) を基にリアルタイムビジョンシステムを構築し，FPGA ベースリアルタイムビジョンシステムの性能を，実験的に評価する．

現在のビジョンシステムは，大きく二つの手法に分類できる．ひとつは，ソフトウエアによる手法で，もうひとつは ASIC による手法である．ソフトウエアによる手法では，ビジョンアルゴリズムはプログラムの形で記述され，汎用の CPU 上で実行される．この手法では，様々なアルゴリズムを，低いコストで実装することができる．一方，アルゴリズムによっては，多くの計算時間を要し，リアルタイムで実行することができない．ASIC による手法では，ビジョンアルゴリズムは論理回路の形で記述され，アルゴリズムに特化して設計，製造された LSI 上で実行される．すでに，フーリエ変換 [1]，ハフ変換 [2]，正規化相関 [3, 4]，ステレオビジョンアルゴリズム [5] など，多くのアルゴリズムが ASIC 上に実装されている．また，論理回路とアナログセンサ回路から成る LSI も試作されている [6, 7]．ASIC による手法では，高速な演算が可能である反面，ASIC 上の回路設計と製造に，多大の労力とコストを要する．一般に，万単位以上のマーケットが見込めないと，ASIC を設計，製造するメリットはないと言われる．結局，ソフトウエアによる手法は，高いフレキシビリティを持つがリアルタイム性に劣り，ASIC による手法はリアルタイム性に優れるがフレキシビリティとコストパフォーマンスに劣る．

フレキシビリティとリアルタイム性とのこのようなジレンマを解決するために，本論文では FPGA ベースビジョンシステムを構築する．FPGA とは，ユーザが論理回路を書き込み，動作させることが可能な LSI である．ユーザは，ビジョンアルゴリズムに特化して設計された論理回路を FPGA 上に実装し，FPGA 上で回路を動作させることにより，ビジョンアルゴリズムを実行することができる．ビジョンアルゴリズムは論理回路の形で実行されるので，リアルタイム性は確保される．論理回路を再度設計し，FPGA に実装することができるので，フレキシビリティも確保される．

以上の考察から，本論文では，FPGA ベースリアルタイムビジョンシステムを構築する．特に，平面運動物体の位置と姿勢を検出するために，画像重心と放射状射影ならびにハフ変換を実装する．平面運動物体の位置と姿勢の検出は，指紋照合や印鑑照合における位置合わせ，LCD/PDP 組み立てにおけるマーカ合わせ等で必要とされている．本論文では，まず，構築した FPGA ベースビジョンシステムのハードウエアを紹介する．次に，FPGA ビジョンシステムに三つのアルゴリズム，画像重心の計算，放射状射影による姿勢の検出，ならびにハフ変換の計算を実装する．最後に，FPGA ベースビジョンシステムで，これらのアルゴリズムを実行した結果を述べる．

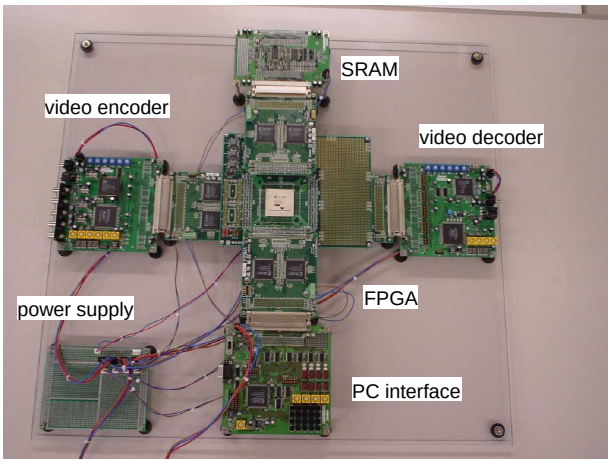


Fig.1: Prototype of FPGA-based vision system

2. FPGA ベースビジョンシステム

ハードウエア構成　本章では，FPGA を用いたビジョンシステムについて述べる．物体操作や人間識別など多くのアプリケーションにおいてビジョンシステムは，連続する一連の画像をリアルタイムで処理することが要求される．さらに，様々な使用環境に適合できるフレキシビリティを有することが求められる．多くのビジョンアルゴリズムは並列性が高く，並列処理を実現することにより計算時間を短縮することが期待できる．一方，FPGA 上に論理回路を並列に配置することにより，並列処理を容易に実現できる．したがって，FPGA ベースビジョンシステムに並列性の高いビジョンアルゴリズムを実装することにより，リアルタイム性とフレキシビリティの双方を達成することができる．

Fig. 1に FPGA ベースビジョンシステムのプロトタイプを示す．このプロトタイプは，FPGA ボード，ビデ

オデコーダボード，ビデオエンコーダボード，SRAM ボード，PC インタフェースから構成される．FPGA ボードは，エスケーエレクトロニクス社製 CM69 であり，FPGA としてザイリンクス社製 Vertex 2000-E が搭載されている．この FPGA 上には，200 万ゲート相当の論理回路を構築できる．ビデオデコーダボード，エンコーダボードは，三菱電機マイコン機器ソフトウエア社製 MU200-VDEC，MU200-VENC であり，ビデオエンコード，デコードチップとして Brooktree 社製 Bt812 と Bt856 が使われている．SRAM ボード MU200-SRAM は，1MB のメモリを提供する．NTSC 信号で送られる CCD カメラからの連続画像は，ビデオデコーダボードでデジタル信号に変換され，FPGA 上に実装される画像処理回路に送られる．画像処理回路は，必要ならば SRAM を記憶回路として使いながら，画像処理を進める．画像処理回路は PC 上で設計される．設計された回路は，PC インターフェース MU200-EX40 を通って，FPGA 上にダウンロードされる．

C 言語ベース回路設計　ビジョンアルゴリズムの回路設計においては，ハードウエア記述言語の代わりに，C/C++言語ベースの回路設計を導入する．従来より，多くのビジョンアルゴリズムが，C/C++言語を用いて開発されている．したがって，C/C++言語を用いた回路設計を導入することにより，アルゴリズムの開発と論理回路の設計を，シームレスに遂行することができる．また，ビジョンシステムの開発では，設計と評価を繰り返すことが多く，回路規模が大きいため論理合成に時間を要する．C/C++言語による回路設計では，C/C++言語の段階で設計を評価することができるため，論理合成の回数を減らし，開発時間を短縮することが期待できる．以上の理由から，C/C++言語を用いた論理回路設計を導入する．

本論文では，C/C++で記述したソースを HDL へ変換するソフトである SystemCompiler を使用する．SystemCompiler では，論理回路の記述に CycleC を用いる．CycleC は，C/C++のサブセットである．CycleC で設計した論理回路は，PC 上でコンパイル，実行することができるので，論理合成することなく論理回路を検証することができる．さらに，テストプログラムやテストパターンを PC 上で容易に構成できるため，効率的な回路の検証が可能である．以上の点より，SystemCompiler を導入することにより，設計した論理回路を短時間で検証することができる．CycleC による記述は，ハードウエア記述言語 VerilogHDL あるいは VHDL に変換される．ハードウエア記述言語に変換された設計を論理合成し，FPGA 上に実装する．以上のように，C/C++言語ベースの論理設計を導入することにより，ビジョンアルゴリズムを短期間で FPGA 上に実装することができる．Table 1に，CycleC によるセレクタの記述例を示す．表に示すように，C/C++言語のクラスとメソッドを用いて，論理回路を記述する．

3. ビジョンアルゴリズムの FPGA 実装

本章では，三つのビジョンアルゴリズムを，FPGA 上に実装する．すなわち，画像重心の計算，放射状射影による姿勢の検出 [8]，ならびにハフ変換の計算である．

画像重心の計算　まず，画像重心の計算を FPGA 上に実装する．計算アルゴリズムを簡単に説明する．グレースケール画像の幅を W，高さを H，格子点 (x, y) におけるピクセル値を $g(x, y)$ とする．画像重心の座標は，

$$\begin{bmatrix} c_x \\ c_y \end{bmatrix} = \begin{bmatrix} W_x/W \\ W_y/W \end{bmatrix} \tag{1}$$

Table 1: Description of selector in SystemCompiler

```
class Selecter {
public:
  uint1 reg;
  Selecter ( void ){ reg = 0;}
  void run ( uint1, uint1,
       uint1, uint1, uint1, uint1& );
};

void Selecter::run(uint1 clk, uint1 rst,
     uint1 a, uint1 b, uint1 s, uint1&y)
{
  uint1 temp = s ? a : b;
  if ( infer_clock(clk) ||
       infer_reset(!rst) ) {
    if (!rst) { reg = 0; }
    else { reg = temp; }
    y = reg;
  }
}
```

ただし

$$\begin{aligned} \begin{bmatrix} W_x \\ W_y \end{bmatrix} &= \sum_{x=0}^{W-1}\sum_{y=0}^{H-1} \begin{bmatrix} x \\ y \end{bmatrix} g(x, y), \\ W &= \sum_{x=0}^{W-1}\sum_{y=0}^{H-1} g(x, y) \end{aligned} \tag{2}$$

で与えられる．上式に示すように，画像重心の計算は，三つの総和と二つの除算から成る．総和 W_x, W_y, W の計算は，画像のキャプチャーに並行して実行することができる．すると，画像のキャプチャーが終った時点で，除算を実行することができる．さらに，二つの除算を並列に実行することができる．結果として，画像重心の計算は，画像キャプチャーの終了後，除算一回分の計算時間で終了する．Fig. 2に，画像重心を計算する論理回路を示す．論理回路の入力信号は，`START`，`X`，`Y`，`G` であり，出力信号は，`COG_X`，`COG_Y`，`COMPLETE` である．入力信号 `START` は，計算開始を指示する．信号 `X` と `Y` は画像の座標値を表し，信号 `G` は画素値 $g(x, y)$ を表す．出力信号 `COG_X` と `COG_Y` は，画像重心の座標値を表す．信号 `COMPLETE` は，計算完了を知らせる．モジュール `SUM` は，画像のキャプチャーと並行して，(2) 式で与えられる W_x，W_y，W を計算する．総和の計算が終了した後，信号 `div_start` が除算開始を指示する．

放射状射影　次に，放射状射影の計算を，FPGA 上に実装する．放射状射影に関して簡単に説明する．グレースケール画像上に座標系 $O - xy$ を固定する．画像上の点 (x, y) における画素値を $g(x, y)$ とする．画像重心から放射状に伸びる半直線に沿って，画素値を積分する．半直線が x 軸と成す角度を θ で表す．このとき，画素値の積分は，パラメータ θ の関数であり，次式で与え

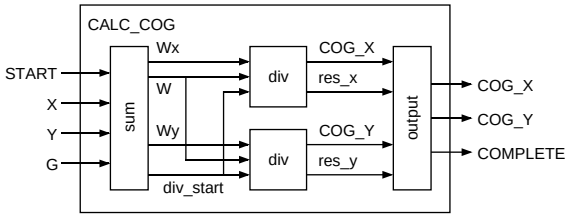
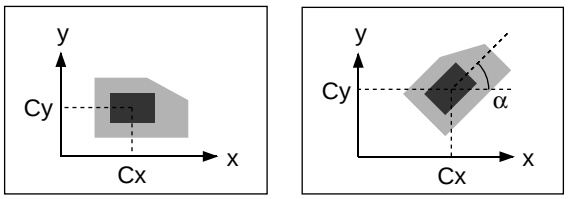


Fig.2: Circuit to compute gravity center



(a) template image (b) input image

Fig.3: Image with translation and rotation

Table 2: Voting in computation of projection

initialize array $R(\theta)$

for $(x, y) = (0, 0), (0, 1), \cdots, (W-1, H-1)$ do

$\Delta x = x - c_x$, $\Delta y = y - c_y$

compute $\theta = \arctan(\Delta y / \Delta x)$

increase $R(\theta)$ by pixel value $g(x, y)$

end

られる．

$$R(\theta) = \int_0^\infty g(c_x + \xi\cos\theta, c_y + \xi\sin\theta)d\xi. \quad (3)$$

ここで，c_x と c_y は，画像重心の座標値を表す．区間 $[0, 2\pi]$ で定められる積分値 $R(\theta)$ 全体を，放射状射影とよぶ．

参照画像を $g_{template}$，入力画像を g_{input} で表す．入力画像中の物体の姿勢を，Fig. 3に示すように α で表す．参照画像と入力画像における放射状射影を，$R_{template}$ と R_{input} で表す．二つの射影 $R_{template}$ と R_{input} は，次に示す関係を満たす．

$$R_{template}(\theta) = R_{input}(\theta + \alpha), \quad \forall\theta. \quad (4)$$

二つの射影間に，次式で与えられる誤差関数を導入する．

$$S(\tau) = \int_0^{2\pi} |R_{template}(\theta) - R_{input}(\theta + \tau)|\, d\theta. \quad (5)$$

この誤差関数は，$\tau = \alpha$ で最小値 0 を取る．したがって，上式で与えられる誤差関数を最小化することにより，姿勢角 α を算出することができる．

(3) 式で与えられる計算は，画像のキャプチャーが終了するまで開始できない．すなわち，この計算は，画像のキャプチャーと並行に計算することができず，リアルタイム処理の妨げになる．そこで，ハフ変換における投票の考えを導入し，画像のキャプチャーと並行に射影を計算できるように，アルゴリズムを構成する．

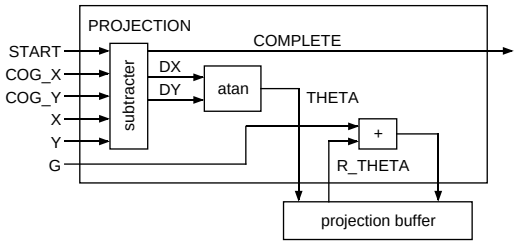


Fig.4: Circuit to compute projection

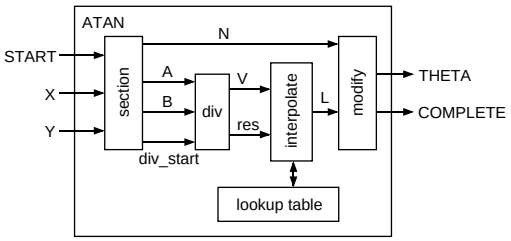


Fig.5: Circuit to compute function *atan*

(3) 式より，

$$x = c_x + \xi\cos\theta,$$
$$y = c_y + \xi\sin\theta.$$

パラメータ ξ を消去すると，

$$\tan\theta = \frac{y - c_x}{x - c_y}. \quad (6)$$

上式は，点 (x, y) における画素値が，角度 θ で定められる積分に寄与することを意味する．すなわち，(6) 式で計算された角度 θ に対応する積分 $R(\theta)$ に，画素値 $g(x, y)$ を加算すればよい．画像のキャプチャーにおいて，画素値は論理回路に順次与えられる．したがって，画像のキャプチャーと並行して，すべての画素値を対応する積分値に加算することができる．以上の投票を，Table 2に示す．

Fig. 4に，投票により放射状射影を計算する回路を示す．この回路は，六本の入力信号 `START`，`COG_X`，`COG_Y`，`X`，`Y`，`G` と一本の出力信号 `COMPLETE` を持つ．モジュール `subtracter` は，$\Delta x = x - c_x$ と $\Delta y = y - c_y$ を計算する．モジュール `atan` は，$\theta = \arctan(\Delta y / \Delta x)$ を計算する．射影 $R(\theta)$ の値は，バッファに保存される．信号 `THETA` を指定し，角度 θ における積分値をバッファから読み出す．読み出された積分値 `R_THETA` に，信号 `G` で表される画素値 $g(x, y)$ が加算され，再び積分値をバッファに書き込む．

モジュール `atan` を，Fig. 5に示す．あらかじめ，区間 $[0, 1]$ 内にいくつかの代表値 v を選び，$\theta = \arctan(v)$ を計算する．代表値における v と θ の組を，ルックアップテーブルに保存しておく．点 (x, y) が $0 \le y \le x$ を満たすとき，$\arctan(y/x)$ を計算する回路を構成しよう．まず，y/x を計算する．商は，区間 $[0, 1]$ に含まれる．したがって，関数値 $\arctan(y/x)$ は，ルックアップテーブル内の代表値を補間することで，計算することができる．モジュール `interpolate` は，この補間を実行する回路である．任意の点 (x, y) における関数値は，この補間を用いて計算することができる．たとえば，点 (x, y) が $0 \le -x \le y$ を満たす場合，関数値 $\arctan(y/x)$ の値は，$\pi - \arctan(-x/y)$ に等しい．商 $(-x/y)$ の値は区間 $[0, 1]$ に含まれるので，$\arctan(-x/y)$ の値はモジュール `interpolate` により計算することができる．そこで，x

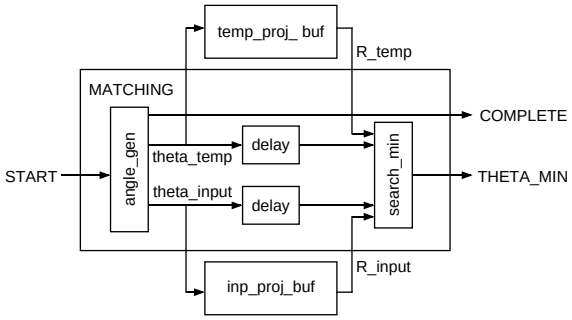


Fig.6: Circuit to compare projections

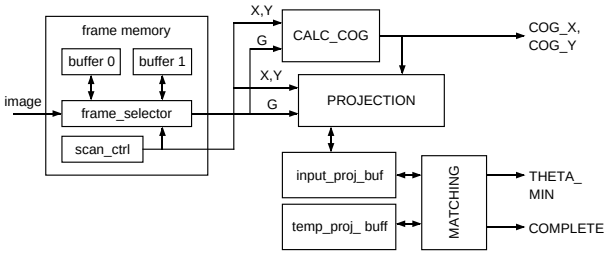


Fig.7: Circuit to detect planar motion

と y の符号，絶対値 $|x|$ と $|y|$ の大小関係により，x-y 平面を 8 個の領域に分割すると，どの領域に対しても，モジュール `interpolate` を用いて，関数値 $\arctan(y/x)$ の値を求めることができる．モジュール `section` は，点 (x, y) に対応する領域を求める．信号 `N` に領域番号，信号 `A` に $|x|$ と $|y|$ の小さい方，信号 `B` にもう一方を出力する．モジュール `modify` は，$\arctan(A/B)$ と N の値から，関数値 $\arctan(y/x)$ を構成する．

Fig. 6に，(5) 式で与えられる誤差関数を最小化する論理回路を示す．モジュール `angle_gen` は，二つの信号 `theta_temp`，`theta_input` を生成する．それぞれ，(5) 式中の角度 θ と $\theta+\tau$ に対応する．対応する放射状射影の値 $R_{template}(\theta)$ と $R_{input}(\theta+\alpha)$ は，バッファ `temp_proj_buf` とバッファ`inp_proj_buf` から読み込まれる．モジュール `search_min` は，射影値の絶対差分の総和を計算し，計算値が最小となる角度 α を求める．

物体と背景が明確に分離でき，画像内に複数の物体が存在しない場合，画像重心を計算することにより，平面運動物体の位置を求めることができる．さらに，参照画像と入力画像の放射状射影を比較することにより，平面運動物体の姿勢を算出することができる．Fig. 7に，重心計算と放射状射影の比較により，平面運動物体の位置と姿勢を計算する回路を示す．ビデオデコーダボードでは，ダブルバッファリングにより，画像を連続的にキャプチャーする．モジュール `frame_selector` は，二つの画像バッファを切替える．モジュール `scan_ctrl` は，画面全体を巡回する座標値 (x, y) を生成する．参照画像の射影は，あらかじめ計算され，バッファ`temp_proj_buf` に格納されている．信号 `COG_X` と `COG_Y` が平面運動物体の位置を，信号 `THETA_MIN` が姿勢を表す．

ハフ変換　ハフ変換は，ビジョンの基本的なアルゴリズムの一つであり，応用範囲が広い．たとえば，ハフ変換を用いることにより，平面運動物体の位置と姿勢を，照明変動やオクルージョンに対してロバストに検出できることが知られている [9]．ハフ変換は時間を要する処理であるが，並列性が高いため，FPGA ベースビジョンに実装することにより，高速な実行が期待できる．本節では，ハフ変換を FPGA ベースビジョンに実装する．

Table 3: Voting in computation of Hough transform

initialize array $H(\rho, \theta)$
for $(x, y) = (0, 0), (0, 1), \cdots, (W-1, H-1)$ do
　for $k = 0, 1, \cdots, K-1$ do
　　compute $\theta = k\Delta\theta$
　　compute $\rho = (-\sin\theta)x + (\cos\theta)y$
　　increase $H(\rho, \theta)$ by pixel value $g(x, y)$
end

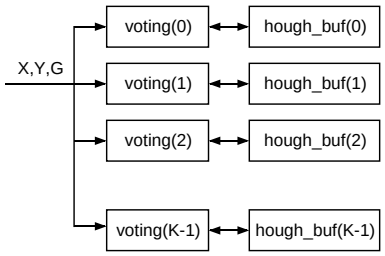


Fig.8: Circuit to compute Hough transform

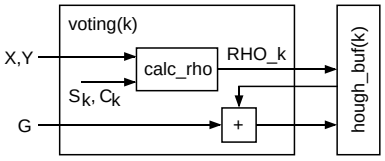


Fig.9: Circuit to perform voting

ハフ変換は，離散画像 $g(x, y)$ からハフ平面 $H(\rho, \theta)$ への変換である．まず，格子点 (x, y) に対して，次式を満たす ρ と θ の組を求める．

$$(-\sin\theta)x + (\cos\theta)y = \rho.$$

次に，上式を満たす ρ と θ の組に対して，$H(\rho, \theta)$ の値を画素値 $g(x, y)$ だけ増加させる．すべての画素値に対して以上の計算を行った結果が，ハフ変換である．上式は，点 (x, y) を通る直線群を表し，ρ は直線の原点からの符号付き距離を，θ は直線と x 軸が成す角を表す．以上の計算過程を，Table 3に示す．

Fig. 8に，ハフ変換を計算する回路の概略を示す．制御信号は省略する．区間 $[0, 2\pi]$ を K 分割し，$\Delta\theta = 2\pi/K$ とする．このとき，角度 θ の値は，$k\Delta\theta$ $(k = 0, 1, \cdots, K-1)$ である．角度 $k\Delta\theta$ に対する計算を，Fig. 9に示すモジュール `voting(`k`)` で行う．角度 $k\Delta\theta$ におけるハフ変換の結果は，バッファ`hough_buf(`k`)` に格納される．モジュール `voting(`k`)` の入力信号は画像の座標値 `X`，`Y`，ならびに画素値 `G` である．モジュールには，$c_k = \cos(k\Delta\theta)$ と $s_k = \sin(k\Delta\theta)$ の値が，あらかじめ記憶されている．モジュール `calc_rho` で $\rho_k = c_k y - s_k x$ の値を計算し，信号 `RHO_k` に出力する．距離 ρ_k の計算が完了後，バッファ`hough_buf(`k`)` を更新する．まず，信号 `RHO_k` を指定し，距離 ρ_k に対する積分値をバッファから読み出す．読み出された積分値に，信号 `G` で表される画素値 $g(x, y)$ を加算し，再び積分値をバッファに書き込む．以上の計算は，角度 $k\Delta\theta$ $(k = 0, 1, \cdots, K-1)$ に対して，並列に実行することができる．したがってハフ変換の計算は，モジュール `voting(`k`)` とバッファ

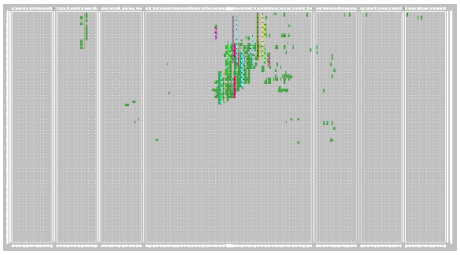
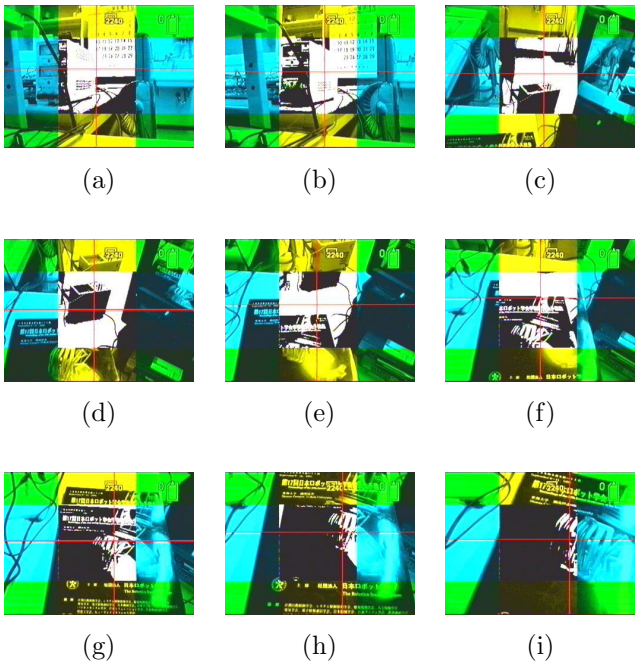


Fig.10: Implementation of logic circuit to compute gravity center



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Fig.11: Detection of image gravity center

hough_buf(k) を，角度の分割数 K 個並列に並べた，Fig. 8に示す回路で実現される．

4. 実験的評価

画像重心の計算　画像重心を計算する回路を，FPGA ベースビジョンシステムに実装した．画面中央部 256×256[pixel] の領域に対して画像重心を計算する．このとき FPGA は，2%のゲートを消費した．Fig. 10に，実装回路で消費された FPGA のゲートを示す．回路を 58MHz で駆動できることが，シミュレーションで判明した．ここでは，画像のキャプチャに合わせて，回路を 12MHz で駆動した．除算一回は 32 クロックで終了するので，画像重心の計算はキャプチャ後約 2.7[μsec] で完了する．Fig. 11に，画像重心の計算例を示す．水平線と垂直線が，計算された画像重心の座標を表す．図に示すように，画像重心をリアルタイムで計算することができる．

放射状射影　重心計算と放射状射影の比較により，物体の平面運動を検出する論理回路を，FPGA ベースビジョンに実装した．画面中央部 256×256[pixel] の領域に対して画像重心と放射状射影を計算し，放射状射影のマッチングを行う．角度は，64 段階で表される．このとき FPGA は，13%のゲートを消費した．Fig. 12に，実装回路で消費された FPGA のゲートを示す．回路を 24MHz で駆動できることが，シミュレーションで判明した．ここでは，画像のキャプチャに合わせて，回路を

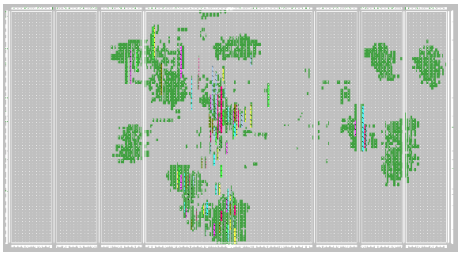


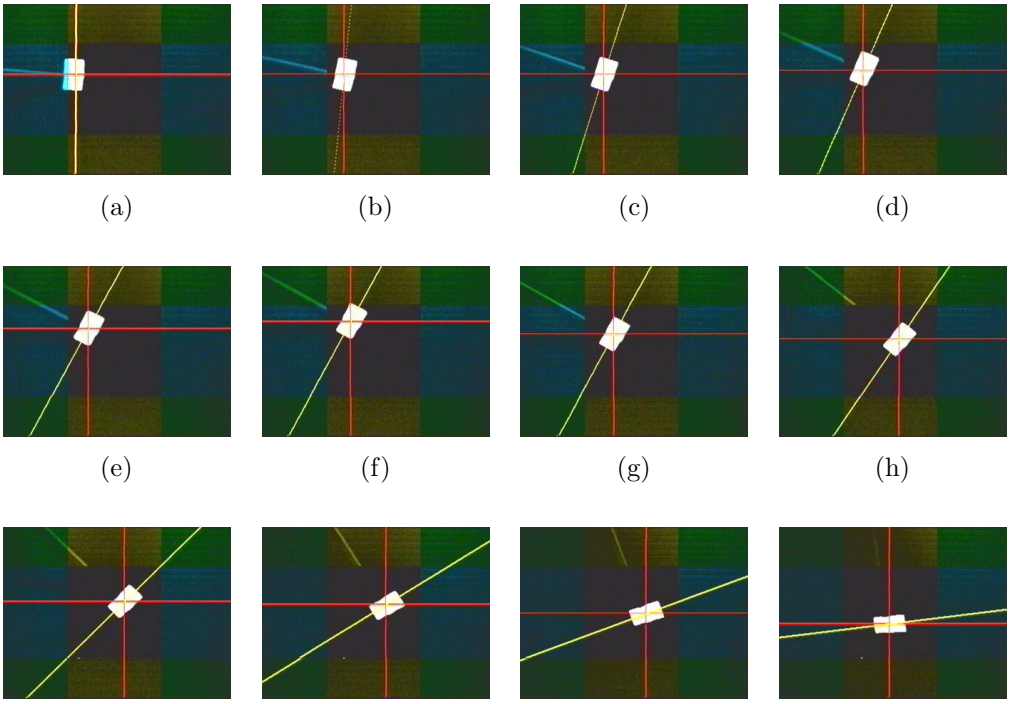
Fig.12: Implementation of logic circuit to detect planar motion object

12MHz で駆動した．一画素に対する投票には 1 クロック要するので，放射状射影の計算には，1×256×256 クロックが必要である．これは，5.46[msec] に相当する．ただし，画像のキャプチャと並行して実行されるので，放射状射影の計算は画像のキャプチャとともに完了する．一方，射影間のマッチングに 64×64 クロックを要しているため，位置と姿勢の検出は，キャプチャ後約 341[μsec] で完了する．Fig. 13に，平面運動物体の位置と姿勢の検出例を示す．水平線と垂直線が物体の位置を，斜めの直線が算出された姿勢を表す．図に示すように，物体の位置と姿勢をリアルタイムで計算することができる．

ハフ変換　ハフ変換を計算し，変換結果を出力する論理回路を FPGA 上に実装した．ハフ変換の結果は，FPGA 内の特別な領域であるブロック RAM に書き込む．回路設計においては，CycleC による回路記述に対してシミュレーションを実行することにより，回路記述を検証する．Fig. 14に，C 言語で記述したソフトウエアによりハフ変換を実行した結果と，ハフ変換の回路記述のシミュレーションによりハフ変換を実行した結果を示す．双方がほぼ一致しており，CycleC による回路記述が正しいことを示す．ハフ変換回路による変換結果の出力例を，Fig. 15 に示す．画面中央部 256×256[pixel] の領域に対してハフ変換を計算し，その結果を画面中央下部に表示する．角度 θ は，144 段階で表される．すなわち，角度の分解能は 2.5° である．距離 ρ は，−127[pixel] から 127[pixel] まで 128 段階で計算する．すなわち，距離の分解能は 1[pixel] である．このとき FPGA は，77%のゲートを使用した．Fig. 16 に，実装回路で消費された FPGA のゲートを示す．回路は，24MHz で駆動することができた．一画素に対する投票には 2 クロック要するので，ハフ変換の計算には，2×256×256 クロックが必要である．これは，5.46[msec] に相当する．ただし，画像のキャプチャと並行して実行されるので，ハフ変換の計算は画像のキャプチャとともに完了する．Fig. 17に，ハフ変換の計算例を示す．図に示すように，入力画像に対するハフ変換をリアルタイムで計算することができる．

5. おわりに

本論文では，FPGA ベースリアルタイムビジョンシステムを紹介するとともに，画像重心と放射状射影を用いて平面運動物体の位置と姿勢を算出するアルゴリズムを実装した結果を述べた．その結果，200 万ゲート相当の FPGA の 15%以下のゲート数で，アルゴリズムを実装できることがわかった．計算時間は約 5[msec] である．ただし，画像のキャプチャと並行して計算が実行されるため，キャプチャ後約 0.3[msec] でアルゴリズムが完了する．また，ハフ変換の計算を実装した結果，200 万ゲート相当の FPGA の 80%程度のゲート数を消費することが判明した．これは，バッファを FPGA 内に構築したためであり，バッファを SRAM 内に構築することによりゲート数を削減することが可能である．

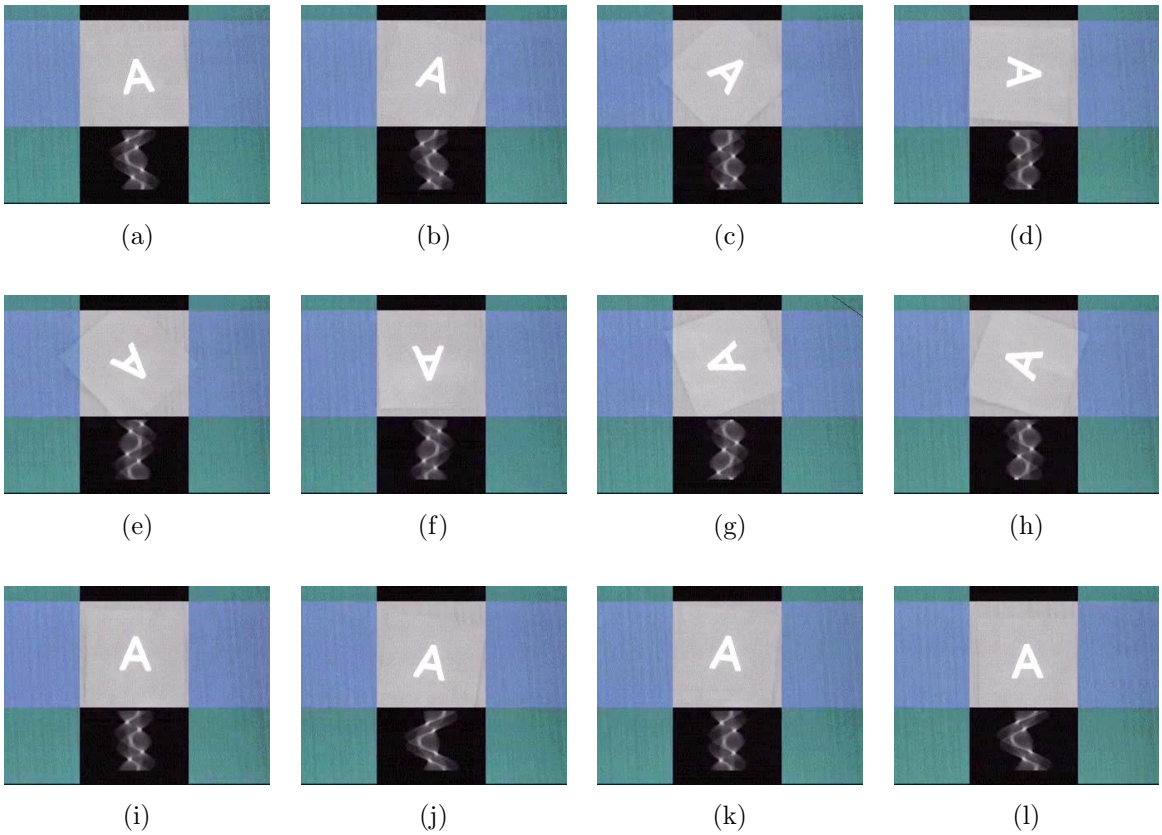


(a) (b) (c) (d)

(e) (f) (g) (h)

(i) (j) (k) (l)

Fig.13: Detection of position and orientation

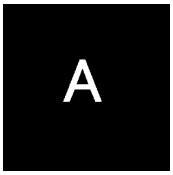


(a) (b) (c) (d)

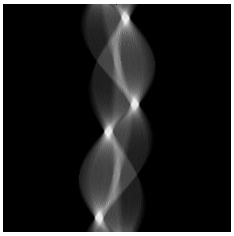
(e) (f) (g) (h)

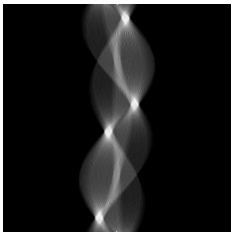
(i) (j) (k) (l)

Fig.17: Realtime computation of Hough transform



(a) input





(b) software (c) simulation

Fig.14: Hough transfroms by software and by simulation of logic circuit

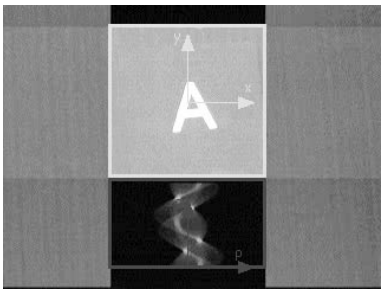


Fig.15: Image and its Hough transform

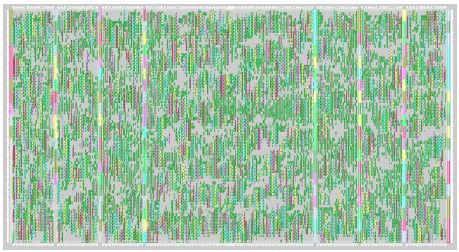


Fig.16: Implementation of logic circuit to compute Hough transform

計算時間は約 5[msec] である．ただし，画像のキャプチャとともに，ハフ変換の計算が完了する．以上のように，FPGA ベースリアルタイムビジョンシステムを用いることにより，計算時間の短縮のみならず，時間遅れの短縮を実現することができる．

本論文では，画像重心と放射状射影の計算ならびにハフ変換の計算を，FPGA に実装した．FPGA に実装することにより，並列処理が可能になるので，Sobel フィルタや Canny フィルタを始めとするフィルタリング処理，相関関数の計算や投票等を実装することは，処理速度の向上の点で有用であると考える．現在，バッファを含めてすべての回路を FPGA 内に実装している．バッファを外部の SRAM に実装することにより，計算量がさらに多いアルゴリズムを実装することができる．整合フィルタ [10] や位相限定相関法 [11]，ハフフーリエ変換法，一般化ハフ変換 [12] は，平面運動物体の位置と姿勢をロバストに検出でき，照明変動や対象物体の欠けなどに対応できることが知られている．しかしながら，これらのアルゴリズムは，一次元/二次元フーリエ変換，ハフ変換，極座標変換など，計算量の多い演算を必要とし，アルゴリズムをリアルタイムに実行することを困難にしている．これらのアルゴリズムを FPGA ベースビジョンに実装し，リアルタイムで実行することが今後の課題である．

【参考文献】

1) Thompson, C. D., *Fourier Transforms in VLSI*, IEEE Trans. on Computers, Vol.C-32, No.11, pp.1047–1057, 1983
2) 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, 1988
3) Inoue, H., Tachikawa T., and Inaba, M., *Robot Vision System with a Correlation Chip for Real-time Tracking, Optical Flow and Depth Map Generation*, Proc. IEEE Int. Conf. on Robotics and Automation, pp.1621–1626, Nice, May, 1992
4) 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
5) 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
6) 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
7) Ishii, I., Nakabo, Y., and Ishikawa, M., *Target Tracking Algorithm for 1ms Visual Feedback System using Massively Parallel Processing Vision*, Proc. 1996 IEEE Int. Conf. on Robotics and Automation, pp.2309–2314, Mineapolis, May, 2001
8) 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
9) 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
10) Turin, G. L., *An Introduction to Digital Matched Filters*, Proceedings of the IEEE, Vol. 64, No. 7, pp.1093–1112, 1977
11) Chen, Q., Defries, M, and Deconinck, F., *Symmetric Phase-Only Matched Filtering of Fourier-Mellin Transforms for Image Registration and Recognition*, IEEE Trans. PAMI, Vol.16, No.12, pp.1156–1168, 1994
12) Ballard, D., H., *Generalizing the Hough Transform to Detect Arbitrary Shapes*, Pattern Recognition, Vol. 13, No. 2, pp.111–122, 1981