RC-002

FPGA 上でのデュアルパイプラインを用いた BLOB 検出と前方車両検出への応用

FPGA-based BLOB Detection Using Dual-pipelining and Appling it to Front Vehicle Detection

野尻　直人†　　孟　林‡　　山崎　勝弘‡
Naoto Nojiri　　Lin Meng　Katsuhiro Yamazaki

1. まえがき

Binary Large OBject(BLOB)とは画像内に存在する特定のグレイスケールの値、または値の範囲を持ったピクセルのかたまりのことを指す。BLOB 検出は自立走行車の誘導、車載カメラ、道路標識認識、車両検出、監視システムなど様々なシステムで用いられる。ラベリングは BLOB 検出のための重要かつ基本的な処理であるが、従来の LUT(Look up table)を用いたラベル補正では、処理が逐次的であり、並列化は困難である。

本論文では、BLOB 検出の高速化を図るために、ラベリングの簡略化を行う。一般的にラベリングは仮ラベル生成、LUT の作成と更新、及びラベル補正で行われる。簡略化ラベリングでは、仮ラベル生成と LUT の作成のみを行い、ラベル補正は行わない。LUT 内の連結成分の情報のみを用いて、BLOB を検出する。

本研究では画像を上下 2 分割し、境界の行の仮ラベルを生成した後、上方向と下方向の仮ラベル生成と LUT 作成を並列に行う。その後、BLOB の検出、面積と重心の計算も並列に行う。

我々は、すでに簡略化ラベリングと BLOB 解析を、FPGA 上でデュアルパイプラインを用いて実現する方法を提案した[9,10]が、本論文はそれを用いて、前方車両検出へ応用する。デュアルパイプラインとは、対象画像を上下に 2 分割し、上部と下部それぞれに画像処理パイプラインを並列に実行することである。デュアルパイプラインを実現する上で、データ依存の関係から各画像処理モジュールのタイミングを考慮する。本論文での BLOB 検出は、ガウシアンフィルタと 2 値化、ラベリング、BLOB 解析から構成される。

近年、ITS(Intelligent Transport Systems:高度道路交通システム)では車載カメラを用いた前方車両検出が運転支援技術の重要な項目である。従来、主にミリ波やレーダ、ステレオカメラが用いられているが、これらは視野が狭いことやコスト面などに制約がある。我々は、デュアルパイプラインを用いた BLOB 検出を前方車両検出へ応用する。テールライトが赤色と法律で制定されているので、テールライトの赤色抽出と左右対称性を用いて前方車両を検出する。テールライト抽出と車両エリア検出をデュアルパイプラインで実行し、乗用車と大型車両を正しく検出できることを示す。

これらのシステムを FPGA 上で実現し、小量のハードウェアで実現できること、BLOB 検出が従来手法[1]と比べて 3.8 倍であること、FPGA による前方車両検出が CPU によるソフトウェア処理と比べて、十分高速であることを示す。

2 章では簡略化ラベリングによる BLOB の検出と解析の流れを述べる。3 章ではデュアルパイプラインを用いた BLOB 検出を説明し、4 章で BLOB 検出を用いた前方車両検出システムを説明する。5 章では実験の内容と結果を説明する。6 章でまとめと今後の課題を示す。

2. 簡略化ラベリングによる BLOB の検出と解析

2.1　デュアルパイプラインを用いた BLOB 検出の流れ

図 1 に本論文での BLOB 検出の流れを示す。BLOB 検出は BRAM からの画像データの読出し、ガウシアンフィルタと 2 値化、簡略化ラベリング、及び BLOB 解析の四つの大きな処理から構成される。これらの処理を 4 段パイプラインで並列に実行する。さらに、本研究では、画像を上部と下部に 2 分割して、デュアルパイプラインを用いて、並列に処理する。

画像データの読出しでは画像からピクセル単位で画像データ読出す。ガウシアンフィルタでは、注目画素とその周囲 8 近傍を用いて画像の平滑化を行う。2 値化では、ガウシアンフィルタ後の結果と閾値を比較して、閾値以上であれば、255 とし、そうでなければ 0 とする。

ラベリングは画像中の連結成分に同じラベルをつけ、異なる連結成分には異なるラベルをつける処理である。それにより、連結成分の個数などを得ることができる。

BLOB 解析では、LUT 内の連結情報を用いて BLOB の個数と各 BLOB の面積、重心を算出する。

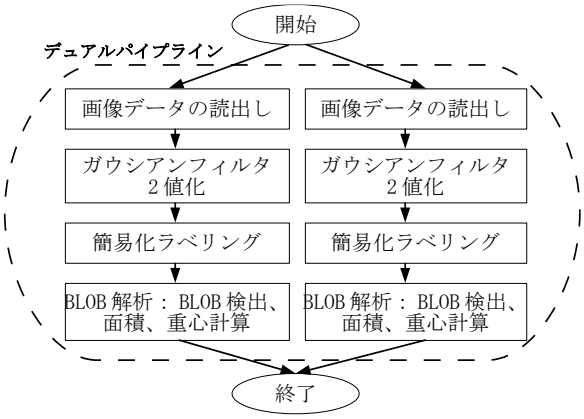


図 1.　BLOB 検出の流れ

† 立命館大学大学院理工学研究科, Graduate School of Science and Engineering , Ritsumeikan University
‡ 立命館大学理工学部,　College of Information Science and Engineering ,Ritsuemikan University

2.2　簡略化ラベリング

(1)　一般的なラベリング

ラベリングは仮ラベル生成、LUT の作成と更新、及びラベル補正から構成される。仮ラベル生成は注目画素の左上、上、右上、左の仮ラベルを参照し生成する。仮ラベル生成と同時に、LUT の作成も行う。ラベリングにおける LUT は、仮ラベルの連結成分を保存する。ラベル補正では、LUT の各行における仮ラベルの等価性を参照して画像中の連結成分を最小の仮ラベルに統一する。図 2(a)の元画像に対して、仮ラベル生成後の LUT は図 2(c)のようになり、仮ラベル補正後の結果は図 2(e)のようになる。

(2)　簡略化ラベリング

簡略化ラベリングは、仮ラベルの生成、LUT の作成と更新までを行う。LUT を参照して画像中の仮ラベルの書き換えは行わない。

仮ラベルの生成のマスクパターンと、画像中の仮ラベルは、一般的なラベリングと同一である。LUT 内の等価性も同じである。簡略化ラベリングでは、画像中の最終的なラベルは仮ラベルのままとなる。

2 値化画像図 2(a)に対して、仮ラベル生成を行うと、図 2(b)のように 1 つの BLOB 内に仮ラベル 1、2、3 が存在する。これらの連結成分を図 2(d)のようにテーブル内に保存し、その情報を用いて 1 つの BLOB として検出する。仮ラベル 5、6 についても同様である。すなわち簡略化ラベリングを用いた BLOB の検出は、テーブル（BLOB）の各行の仮ラベルの等価性のみを評価して行う。

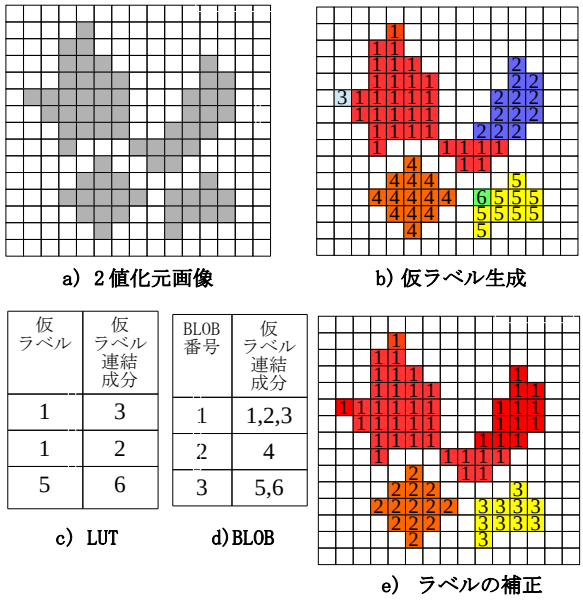
3.　デュアルパイプラインを用いた BLOB 検出

3.1 デュアルパイプラインシステムの構成

図 3 にデュアルパイプラインを用いた BLOB 検出システムを示す。対象画像は予め、BRAM に保存されているものとする。本システムでは、4 つの画像処理モジュール(FE, GAU, LAB, BLOB)、3 つの制御モジュール(PIX_Ctrl, GAU_Ctrl, LAB_Ctrl)、アドレス生成モジュール(Generate_ADDR)、及び 3 つの 3 ラインバッファ(PIX_Register, GAU_Register, LAB_Register)から構成される。4 つの画像処理を以下の 4 ステージのパイプラインで処理する。

- 第一ステージ（FE）：Block RAM から画像データを読出す。
- 第二ステージ（GAU）：ガウシアンフィルタと 2 値化を用いて、ノイズを除去する。
- 第三ステージ（LAB）：仮ラベルの生成と LUT を作成する。
- 第四ステージ（BLOB）：BLOB の個数と面積、重心を算出する。

BLOB で LUT 内の仮ラベル連結成分を検出し、BLOB の個数と各 BLOB の面積、重心を算出する。BLOB の解析結果を BLOB_RESULT に保存する。これらは、レジスタにより構成される。面積はピクセル数とし、重心(Gx,Gy)は以下の式を用いる[2][3]。



仮ラベル	仮ラベル連結成分
1	3
1	2
5	6

BLOB 番号	仮ラベル連結成分
1	1,2,3
2	4
3	5,6

図 2.　一般的なラベリングと簡略化ラベリング

$$\begin{cases} Gx = X\text{ 座標の合計/面積} \\ Gy = Y\text{ 座標の合計/面積} \end{cases}$$

3 ラインバッファは、各ステージにおいて、前のパイプラインステージの処理結果の保存、後のパイプラインステージの処理のデータの供給の役割をしている。そのため、3 ラインバッファは、読み出しと書き込みの機能を持つ 2-port BRAM により構成される。

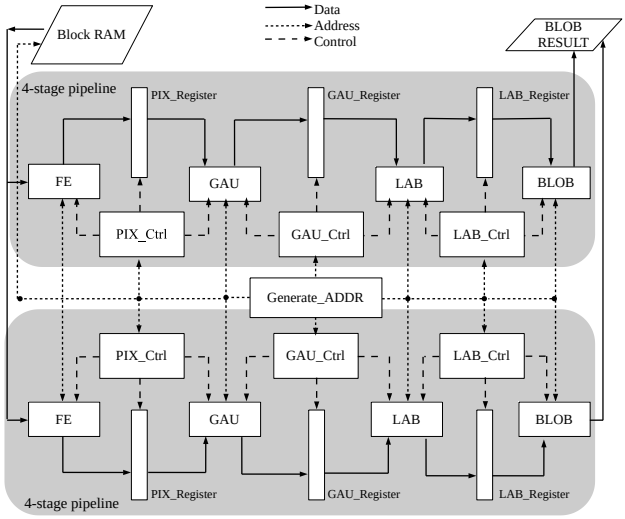


図 3. デュアルパイプラインを用いた BLOB 検出システム

本システムでは、1 枚の画像を上部と下部に 2 分割し、上のパイプラインが上部処理、下のパイプラインが下部処理を担当し、それぞれ FE から BLOB までの 4 段パイプラインを用いて、並列に処理する。制御モジュールは画像処理モジュールで得られた結果の書込みと、次の画像処理モジュールで必要なデータを読み出す。Generate_ADDR はアドレスを生成し、全体の制御を行う。FPGA の BRAM を十分に使用するために、1024*256 サイズの画像を対象にシステムを作成した。

3.2 タイミングチャート

デュアルパイプラインシステムを実現するために、画像を図 4 のように上部(A)と下部(B)に分割する。行番号を図の左側に示し、境界線を中心にして行番号 1 である A の最下行と B の最上行から開始してそれぞれ矢印方向に処理を進める。画像サイズについて、X 軸は M ピクセル、Y 軸は 2N ピクセルとする。

図 5 に各モジュールのタイミングチャートを示す。図 5(a)は全ての行のタイミングチャートで、数値は画像の行番号である。ガウシアンフィルタは注目画素の次のラインに依存するので、GAU の実行は A と B において FE から 1 ラインの遅延が必要である。ラベリングは注目画素の次のラインに依存しないので、LAB の開始は GAU と同一となる。境界線付近に BLOB が存在する可能性があるので、B1 の BLOB は A1 の BLOB より 1 ライン遅延させる必要がある。1 つの LUT 内に上部と下部で得られた全ての仮ラベル連結成分を保存する。これにより境界線付近に BLOB が存在する場合でも、LUT 内の仮ラベル連結成分を評価して、検出が可能となる。

図 5(b)は Y 軸(Y_Addr)が 4 のときの図 4A の部分のタイミングチャートである。すなわち、図 4 の各ラインの処理においても、ピクセル単位のパイプライン処理を行っている。各モジュールの Y 軸のアドレスは、図 5(b)の Y 軸アドレスと同じである。X 軸のアドレスについて、FE はアドレスを生成するために、1 クロックの遅延が必要である。GAU は 3*3 のマスクのデータを揃えるのに 2 クロックと、ラインバッファアクセスのために 1 クロックが必要であるため、３クロックの遅延が生じる。同様に、LAB も３クロックの遅延が必要である。さらに、BLOB は、ラインバッファにアクセスするために、１クロックの遅延が必要である。

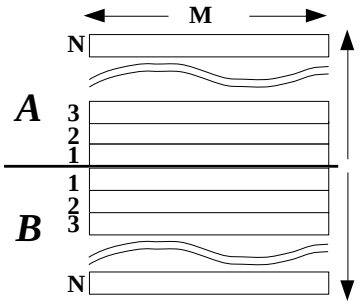


図 4.　画像の 2 分割

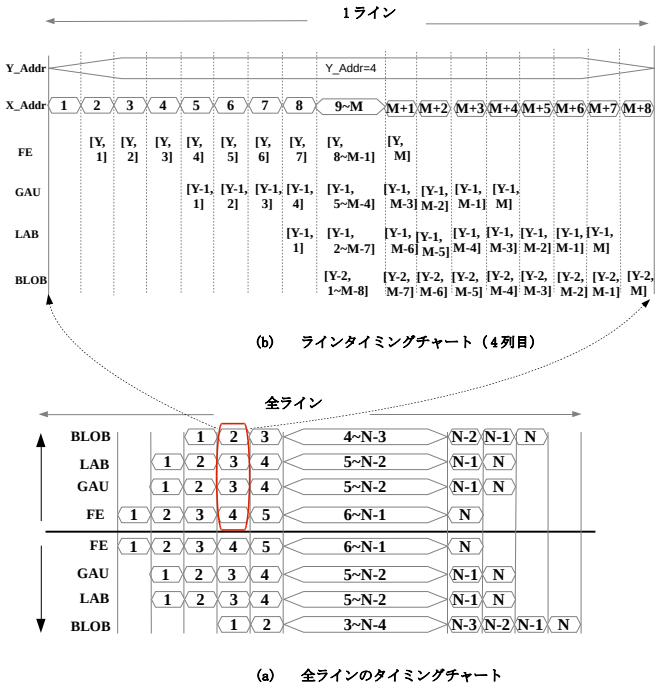


図 5.　タイミングチャート

4.　BLOB 検出を用いた前方車両検出

4.1 前方車両検出システム

本章では、デュアルパイプラインを用いてテールライトを着眼点とし、前方車両検出を説明する。日本の道路運送車両法の保安基準により、後面の 2 ヶ所に赤色の灯火をつけることが決められている。2 灯を両側に配置する場合には、車体中心より対称の位置とする必要がある。したがって、テールライトの高さ(Y 座標)と面積は、それぞれ同一になる。これらの左右対称性を用いて前方車両を検出する。

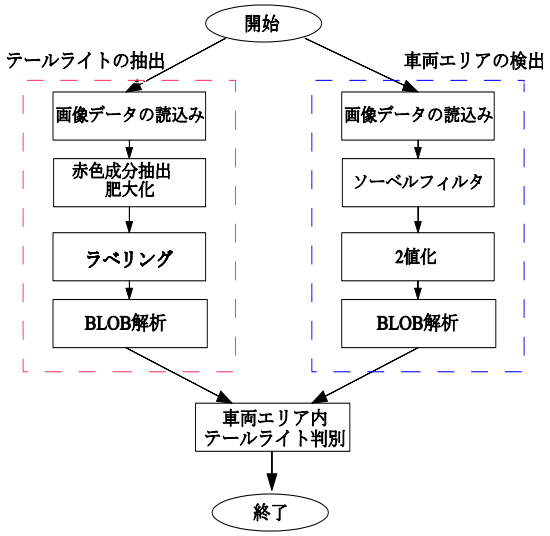


図 6.　前方車両検出の流れ

図 6 に本論文での前方車両検出の流れを示す。前方車両検出はテールライト抽出、車両エリア検出、及び車両エリア内テールライト判別の 3 つの処理から構成される。テールライト抽出は赤色成分の抽出と肥大化、ラベリング、BLOB 解析から構成される。車両エリア検出はソーベルフィルタ、2 値化と BLOB 解析から構成される。車両エリア内テールライト判別は、車両エリア内の座標内にテールライトの重心が存在するかを判別し、その場合、前方車両として検出する。

本論文では、テールライトの抽出と車両エリアの検出を、デュアルパイプラインを用いて並列に行う。最後に、車両エリア内でのテールライトの有無を判別する。

図 7 にデュアルパイプラインを用いた前方車両検出システムを示す。テールライト抽出について、FE は BRAM からピクセル単位でデータを読出し、RED は赤色成分の抽出と肥大化を行う。LAB は肥大化した赤色成分に対して仮ラベルと LUT を作成し、BLOB が各赤色成分の重心と面積を算出し、左右対称性評価を行う。これらのモジュールは (PIX_Ctrl, RED_Ctrl, LAB_Ctrl)の 3 つにより制御され、中間結果が 3 つの 3 ラインバッファ (PIX_Register, RED_Register, LAB_Register)に保存される。

車両エリア検出では、FE が画像データの読出しを行い、SBL がソーベルフィルタを用いて車両の輪郭を検出する。AREA は 2 値化を用いて車両の輪郭を抽出し、BLOB が車両の輪郭の X,Y 座標の最小値と最大値を算出する。これらのモジュールは (PIX_Ctrl, SBL_Ctrl, AREA_Ctrl)の 3 つにより制御され、中間結果が 3 つの 3 ラインバッファ (PIX_Register, SBL_Register, AREA_Register)に保存される。

最後に、Judge が車両エリア検出で算出した輪郭の X,Y 座標の最大値と最小値にテールライトの重心が存在するかを判別する。アドレス生成モジュール(Generate_ADDR)は、システムに必要な座標生成と全体の制御を行う。

4.2 前方車両検出システムの各処理

(1) テールライト抽出

赤色成分の抽出は赤色成分であれば“255”とし、それ以外は”0”と出力する。テールライト候補以外にも周囲の環境によっては赤色成分が抽出されるが、後処理で行うラベリングや BLOB 解析によって除去される。抽出した赤色成分だけでは、テールライトとしての認識が困難である。肥大化を行うことで、認識精度を向上させる。

肥大化は、図形成分の境界画素に接する背景成分中の画素の値を図形成分の画素の値に変換して 1 画素分膨らませることである。画像中の”255”を検出したときに、周囲 8 近傍を赤色成分“ 255”として肥大化する。

2.2 で提案した簡略化ラベリングを用いて、赤色成分に対してラベルを振り分ける。簡略化ラベリングは並列処理に有効であり、画像中の仮ラベルの書き換えを行わないので処理時間の短縮を図ることが可能である。

テールライト抽出における BLOB 解析では、各赤色成分の面積と重心を算出する。車両 1 台において、2 つのテールライトの面積と重心の Y 座標は概ね等しくなると考えられる。しかし、周囲の環境や影などの影響により数値が等しくなることは困難である。本論文では、面積と重心を算出した

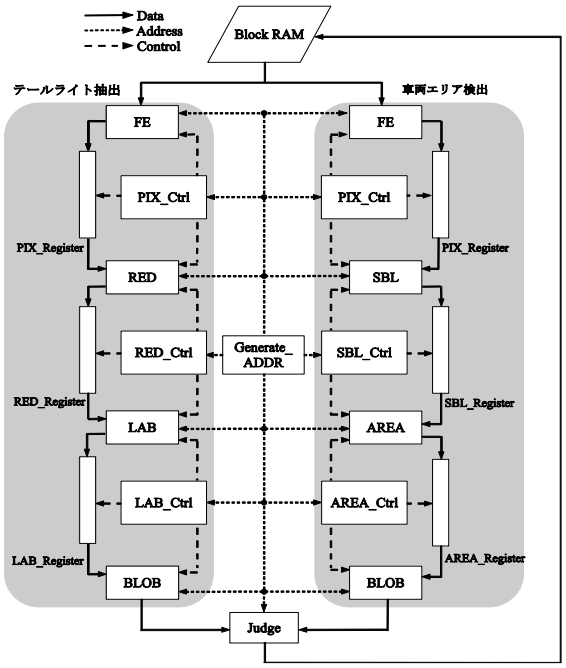


図 7. デュアルパイプラインを用いた前方車両検出システム

後に、それぞれの数値の前後 30%以内に収まる赤色成分を検出する。この範囲内で検出された赤色成分をテールライト候補として認識する。テールライト候補として認識された成分の重心座標を出力する。

(2) 車両エリア検出

ソーベルフィルタは画像内の主要なエッジを空間 1 次微分し強調する。画像の輝度値に対して、隣り合う画素の輝度差が大きいほど画像のエッジとなる。さらに、他のエッジフィルタと比べて、ノイズを抑え強いエッジを得ることが可能である。本論文では水平方向と垂直方向を用いて得られた結果を加算する。また車載カメラから撮影した画像では、前方車両が画像の中央に存在するので、ソーベルフィルタを画像中央部のみに適用する。2 値化では、閾値を設定する。閾値以下であればノイズとして除去し、車両の輪郭を抽出する。本論文では、2 値化を静的に行う。車両エリア検出での BLOB 解析では、車両エリアの輪郭の X,Y 座標の最小値と最大値を算出する。

(3) 車両エリア内テールライト判別

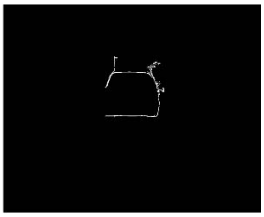
テールライトの重心(C_X, C_Y)と車両エリアの X,Y 座標の最大値と最小値(Min_X, Min_Y),(Max_X, Max_Y)を用いて、車両エリア内にテールライトの重心が存在するかどうかを判別する。判別式は以下の式を用いる。

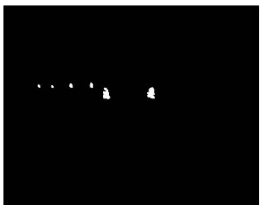
$$\begin{cases} Min\_X < C\_X < Max\_X \\ Min\_Y < C\_Y < Max\_Y \end{cases}$$


(a)　原画像


(b)　ソーベルフィルタ


(c)赤色抽出と肥大化


(d)　輪郭抽出


(e)左右対称性評価

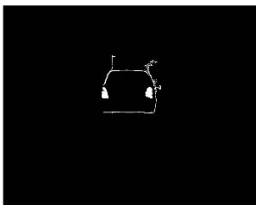
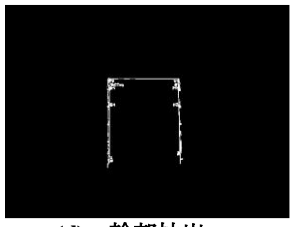

(f)車両エリア内判別

図 8.　前方乗用車の検出結果


(a)　原画像


(b)　ソーベルフィルタ


(c)赤色抽出と肥大化


(d)　輪郭抽出


(e)左右対称性評価

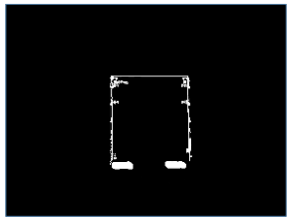

(f)車両エリア内判別

図 9.　前方大型車両の検出結果

4.3 車両エリア内テールライト判別

図 8 と図 9 には普通乗用車と大型車両の検出の一連の処理結果を示す。図 8,9 の(c)では、赤色抽出と肥大化により、2 種類の車両のテールライトが鮮明になっているが、図 8(c)には右上のビルの赤い看板、図 9(c)には大型車両のナンバープレートの反射材のノイズが残っている。提案手法では、左右対称性評価を行い、図 8(c)と図 9(c)からそれらのノイズを除去し、テールライトだけを取り出すことができた。

また、図 8,9 (d)では、図 8,9 (b)のソーベルフィルタの結果から両方の車両の輪郭を抽出できた。最後に、2 つの図の(f)から見ると、テールライトが車体の輪郭に含まれ、両方の前方車両が検出できた。

5.　FPGA 上での実験

5.1　実験内容

(1)　BLOB 検出

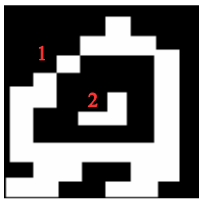
BLOB 画像のサイズは、100*100,1 画素 8bit とした。対象画像は文献[1]の画像と自作画像 2 枚である。デュアルパイプラインシステムの実行時間とハードウェアサイズを、先行研究[1]とシングルパイプラインシステムのそれらと比較する。

(2)　前方車両検出

前方車両の画像サイズは、128*128 とする。対象車両は、

BLOB	面積	重心(X,Y)
1	320	(30,26)
2	630	(42,52)
3	320	(84,66)
4	320	(52,88)

(a) 文献 1 の画像と解析結果



BLOB	面積	重心(X,Y)
1	3643	(58,64)
2	275	(48,52)

(b) 自作画像 1 と解析結果

BLOB	面積	重心 (X,Y)
1	2647	(44,59)

(c) 自作画像 2 と解析結果

図 10.　BLOB 解析結果

昼間での一般車両と大型車両とする。前方車両検出では実行時間の比較を行うために、同様の画像処理を CPU 上のソフトウェアで実行して実行時間を計測する。

(3) 実験条件

本論文で提案する 2 つのシステムは画像データを FPGA 内の BRAM に格納し、画像処理を行った。FPGA は Xilinx 社の Virtex5(XC5VFX-70T)評価ボード、デザインツールは ISE 14.5、シミュレーションツールは ISim、ハードウェア設計言語は Verilog 2001 である。実行時間は画像ファイルからデータを読み出し、全データに対して画像処理を行い、メモリに格納するまでの時間とした。比較対象であるソフトウェア CPU の実験環境は使用言語が C 言語、実装メモリが 8.00GB、ソフトウェア処理の動作環境が Ubuntu14.04 の Intel®Core™i7-4770s CPU@3.40GHz である。

5.2 実験結果

(1) BLOB 解析結果

BLOB 解析結果を図 10 に示す。解析内容は、BLOB の個数、面積と重心である。図 10 (a)は先行研究[1]から引用し、図 10 (b)(c)は自ら作成した画像である。本論文では、ラベリングを簡略して BLOB 検出を行った。図 10 (b)(c)のように複雑な画像に対しても、正確に解析を行うことができた。図 8, 9 (c)により、赤色成分が抽出することができ、前方の普通乗用車と大型車両を検出できることが分かった。

(2) ハードウェアサイズ

表 1 に BLOB 検出の各モジュール、デュアルパイプラインとシングルパイプラインのハードウェアサイズを示す。表 2 に前方車両検出の各モジュールとシステム全体のハードウェアサイズを示す。表 1 と表 2 中の%は Vertex 5 の使用率を示す。

デュアルパイプラインを用いた BLOB 検出に使用する Registers、LUTs、LUT-FF pairs の使用率は 1.6%から 13.6%であり、非常に小規模なハードウェアで BLOB 検出が実現できた。デュアルパイプラインでは LUTs、 Registers の使用率がシングルパイプラインの約 2 倍になった。

前方車両検出は、デュアルパイプラインを用いた BLOB 検出システムを応用して構築したので、使用するハードウェア量は概ね同じであり、Registers、LUTs、LUT-FF pairs の使用率は 1.7%から 20.8%までであった。

BRAM の使用率は、BLOB 検出と前方車両検出が、ハードウェア資源の 90%以上を使用している。全体的には両システムとも、FPGA 上で実装可能なハードウェアサイズと成った。

(3) 実行時間

表 3 に BLOB 検出と前方車両検出の実行時間を示す。BLOB 検出について、動作周波数は 99.18MHz であり、シングルパイプラインシステムと同じである。またピクセル当たりの実行時間は 10.08ns で、100*100 サイズでの全実行時間は、10.08ns*108*53 で 57.70μs となった。従って、デュアルパイプラインの実行時間は先行研究[1]と比較して 3.81 倍、シングルパイプラインシステムと比べて、1.94 倍の速度向上を得た。

データ依存が生じるため、FE はアドレスを生成するのに 1 クロックを必要とし、GAU は 3*3 のマスクのデータを揃えるのに 2 クロックを必要とし、BLOB は 3*3 のマスクのデータを揃えるのに 2 クロックを必要とする。また、3 つのラインバッファはアドレスを生成するために、それぞれに 1 クロックを必要とする。従って、100*100 画像サイズに対して、ライン毎に必要なクロック数は 100+1+2+2+3=108 となる。

表 1. BLOB 検出におけるハードウェアサイズ

	Reg	LUTs	LUT-FF pairs	Block RAM
FE	2	56	0	64
PIX_Ctrl	72	101	61	2
GAU	46	0	0	0
GAU_Ctrl	48	67	41	1
LAB	9	1209	9	0
LAB_Ctrl	72	103	61	2
BLOB	8	11	8	0
Generate_ADDR	287	271	209	0
デュアルパイプライン	696 (1.6%)	3284 (7.3%)	478 (13.6%)	136 (91.9%)
シングルパイプライン	453 (1.0%)	1945 (4.3%)	271 (12.7%)	132 (89.2%)
Kiran[1]	--	2784 (6.2%)	--	4 (2.7%)
Virtex 5	44800	44800	2127	148

表 2. 前方車両検出におけるハードウェアサイズ

	Reg	LUTs	LUT-FF pairs	Block RAM
FE	2	58	0	64
PIX_Ctrl	72	101	61	2
RED	1	0	0	0
RED_Ctrl	48	67	41	1
LAB	9	1209	9	0
LAB_Ctrl	72	103	61	2
BLOB(テールライト)	31	82	31	0
SBL	93	0	0	0
SBL_Ctrl	72	101	61	2
AREA	8	30	8	0
AREA_Ctrl	72	103	61	2
BLOB(車両エリア)	54	0	0	0
JUDGE	38	0	0	0
Generate_ADDR	315	271	209	0
システム全体	771 (1.7%)	2388 (5.3%)	543 (20.8%)	137 (92.6%)
Virtex 5	44800	44800	2616	148

表 3. BLOB 検出と前方車両検出の実行時間

	BLOB 検出			前方車両検出	
	dual	single	Kiran[1]	FPGA	CPU
動作周波数 (MHz)	99.18		100	127.93	--
実行時間/ ピクセル(ns)	10.08		--	12.14	--
全実行時間	57.70μs	112.13μs	220μs	0.22ms	8.73ms
速度向上(倍)	3.81	1.96	1	39.68	1

実行ライン数について、GAU が FE より 1 ラインの遅延が必要であり、BLOB が LAB より 2 ラインの遅延が必要である。デュアルパイプラインであるため、実行ライン数は 100/2+3 = 53 となる。

前方車両検出の動作周波数は、127.93MHz となった。ピクセル当たりの実行時間は 12.14ns である。128*128 サイズでの全実行時間について、データ依存が生じるために、シングルパイプラインを用いた BLOB 検出と同じような遅延が生じ、12.14ns *(128+8)*(128+3)=0.22ms となった。デュアルパイプラインを用いた前方車両検出は、ソフトウェア処理と比較して 39.68 倍の速度向上を得た。

5.3 考察

(1) ハードウェアサイズ

BLOB 検出と前方車両検出における Generate_ADDR は、アドレスの生成と全体の制御を行っているため、Registers 数と LUT-FF pairs 数が他のモジュールよりも多くなっている。

前方車両検出システムは BLOB 検出システムを応用して構築したので、ハードウェア量が BLOB 検出と概ね同じ使用率となった。FPGA 上での前方車両検出システムは、全体のハードウェア量の観点から、実装可能なハードウェア量となった。余ったハードウェア資源を用いて、さらに高度なアルゴリズムの実装にも使えると考えられる。

BRAM がハードウェア資源の約 90%以上を使用しているが、処理前後の画像を SRAM に格納することにより、BRAM の使用率を減らせると考えられる。

(2) 実行時間

デュアルパイプラインによる BLOB 検出において、境界線付近の処理の遅延が生じるが、それらの影響は少なく、シングルパイプラインシステムと比べて、1.94 倍の速度向上を得た。

デュアルパイプラインを用いた前方車両検出は、ソフトウェア処理と比較して 39.68 倍の速度向上を得た。FPGA 上で前方車両検出システムを構築することで、ソフトウェア処理より高速に検出することが可能であり、デュアルパイプラインが有効であることが分かった。

(3) マルチパイプラインによる高速化

本研究ではデュアルパイプラインシステムを構築したが、4 個や 8 個のマルチパイプラインを用いて実現することも可能である。例えば、4 個のパイプラインの場合、画像を 4 分割し、上半分と下半分に本手法を適用する。その後、上半分と下半分の境界での補正処理を行う。BRAM を除いて、Vertex 5 のハードウェア資源の使用率は 20%以下であるので、マルチパイプラインによる高速化が可能であると考えられる。

6. 関連研究

BLOB の関連研究について、Kiran らは BLOB 検出における重心と面積の計算を、FPGA 上で効率的に実現する手法を提案し、Virtex 5 上でハードウェア量と実行時間を評価した[1]。Bochem らは大容量のビデオストリーム画像における BLOB 検出に run-length-encoding 法を提案し、Bounding Box 法と Center-of-Mass 法を用いて提案手法を評価した[2]。Xiong らはソフトウェアプロセッサコア上で、BLOB 検出のためのシステムアーキテクチャを提案し、2*3 のマスクを用いた Dual Connected Component Labeling を提案した[3]。Kim らはフレーム単位でのパイプラインを用いて、ラベリング成分と追跡成分から成るシステムを提案した[4]。

我々は仮ラベル生成と LUT の作成を行い、ラベル補正を省略する簡略化ラベリングを提案した。仮ラベル生成と LUT 作成は並列に行えるので、本手法による BLOB 検出システムを FPGA 上にデュアルパイプラインで実現して、高速化できることを示した。

前方車両検出について、Bhaskar らは背景から前景を分離し、ガウス混合モデルとブロブ検出を用いて、91%以上の精度で検出を行った[5]。Li らは AOG(and-or graph)を用いた車両検出を提案し、渋滞での車両を表すための AOG の構築、AOG でのパラメーターの訓練、ボトムアップ推論の 3 つにより、様々な車両や時刻及び気象に有効な車の検出を行った[6]。Fang らは白線検出、車両検出、車間距離推定のシステムを提案し、昼間では車両の影と水平エッジを用いて、夜間ではテールライトの性質を用いて、車両検出を行った[7]。Sivaraman らは白線検出と車両検出を統合したシステムを提案し、評価した[8]。

[5~8]は CPU ソフトウェア上で実装しているが、我々は FPGA を用いて、前方車両の検出を行っている。テールライト抽出と車両エリア検出をデュアルパイプラインで並列に FPGA 上で実現している点に特徴がある。FPGA のハードウェア資源に余裕があるので、より複雑な画像に対処できると考えられる。

7. おわりに

本論文では、BLOB 検出の高速化を実現するために、仮ラベル生成と LUT 作成のみを行い、ラベル補正を省略する簡略化ラベリングを提案し、それを FPGA 上でデュアルパイプラインで実現して、先行研究と比べて 3.8 倍の速度向上を達成できることを示した。また、本手法を前方車両検出に応用し、乗用車と大型車両の検出が可能であり、CPU 上でのソフトウェア処理と比べ、約 40 倍の速度向上を達成できることを示した。今後の課題として、ノイズ除去における動的な閾値設定、マルチパイプラインによる更なる高速化、より複雑な前方車両の検出などが挙げられる。

参考文献

[1] D.Kiran, A.I.Rasheed and H.Ramasangu, “FPGA Implementation of BLOB Detection Algorithm for Object Detection in Visual Navigation”, the 2013 Int. Conf. on Circuits, Controls and Communications (CCUBE), pp.1-5, 2013.

[2] A.Bochem, K. B. Kent and R.Herpers, “FPGA based Real-Time Object Detection Approach with Validation of Precision Performance”, 2011 22nd IEEE Int. Symp. on Rapid System Prototyping (RSP), pp.9-15, 2011.

[3] J.Xiong, T.M.Nguyen, and Q.M.J.Wu, “FPGA Implementation of BLOB Recognition”, the 2011 Canadian Conf. on Computer and Robot Vision (CRV), pp.125-131, 2011.

[4] D.K.Kim, D.R.Lee, T.C.Pham, T.T.Nguyen, and J.W. Jeon, “Real-time Component Labeling and Boundary Tracing System Based on FPGA”, the 2007 IEEE Int. Conf. on Robotics and Biomimetics (ROBIO), pp.189-194, 2007.

[5] P. K. Bhaskar and S.P. Yong, “Image Processing Based Vehicle Detection and Tracking Method”, the 2014 Int. Conf. of Computer and Information Sciences (ICCOINS), pp.1-5, 2014.

[6] Y. Li, B. Li, B. Tian and Q. Yao, “Vehicle Detection Based on the AND– OR Graph for Congested Traffic Conditions”, IEEE Trans on Intelligent Transportation Systems, Vol.14, No. 2, pp.984-993, 2013.

[7] C. Y. Fang, J. H. Liang, C. S. Lo and S. W. Chen, “A Real-time Visual-based Front-mounted Vehicle Collision Warning System”, the 2013 IEEE Symp. on Computational Intelligence in Vehicles and Trans Systems (CIVTS), pp.1-8, 2013.

[8] S. Sivaraman, and M. M. Trivedi, “Integrated Lane and Vehicle Detection, Localization, and Tracking: A Synergistic Approach”, IEEE Trans on Intelligent Transportation Systems, Vol.14, No.2, pp.906-917, 2013.

[9] N.Nojiri, L.Meng and K.Yamazaki, “FPGA-based BLOB Detection Using Dual-pipelining”, the 2015 ACM/SIGDA Int. Symp. on Field-programmable gate arrays, 048, 2015.

[10] 野尻, 孟, 山崎, “FPGA 上でのデュアルパイプラインを用いた BLOB 検出”, 情報処理学会　第 77 回全国大会, 3K-02, 2015.