Vol.2017-DPS-170 No.10
Vol.2017-CSEC-76 No.10
2017/3/2

JointHOG 特徴による特定物体認識

高松秀行[†1]　築地立家[†2]

概要：本論文では，組み込みシステムにおいて周辺環境認識プログラムに用いられる JointHOG 特徴の計算処理を，GPGPU によって高速化する手法を提案する．高速化させることで，JointHOG 特徴の計算処理を組み込みシステムへの導入のハードルを下げる．出口大輔らは，JointHOG 特徴は精度が高いものの，計算コストが高くリアルタイム処理が困難であると述べている．そこで，高コストとなる計算処理を GPU に並列処理させる手法を提案する．GPU と CPU で使用するメモリ間でのデータのやり取りは大きなオーバーヘッドとなるために，データコピーの回数を削減する工夫により，本手法を実現した．本提案によって JointHOG 特徴の計算処理時間を削減できることを確認するために，組み込み開発キットである「Jetson TK1」を利用して四輪車を JointHOG 特徴で検出するための処理時間を計測した．結果として，JointHOG 特徴の計算処理時間を 90%削減することができた．

1. はじめに

近年，自動運転システムの開発は自動車の開発分野でも特に注目を集めており，2020 年の東京オリンピックで技術を披露しようと活発に研究が進められている[1].

コンピュータの処理性能も日々向上しており，GPU を搭載した組み込みボードなども登場している．そのため GPU で並列的に計算処理を行わせる GPGPU(General-purpose computer on graphics processing units)技術を用いることで，プログラムの工夫や効率次第で性能やコスト以上のパフォーマンスを得られるようになった．
これにより，コストを抑えながらも高いパフォーマンスが必要とされる組み込みシステムの分野では GPGPU 技術の研究が注目されている．

2. 研究の背景と目的

2.1 背景

GPU を搭載した組み込み開発キットである Jetson TK1 と GPGPU 技術の登場により，組み込み開発キットでもコストに対して高いパフォーマンスが得られるようになった．

そして近年，防犯意識の向上や事故を起こした際の証拠として認められたことから，ドライブレコーダーが広く普及していることで，RGB カメラを搭載した車両が増加した．

そこで，画像内から人や四輪車などの識別精度が高いものの，計算コストが高く，リアルタイム処理が困難とされる[2][3]Joint HOG 特徴[4]を用いて，組み込み開発キット上での周辺環境認識の高速化の研究を行う．

2.2 目的

本研究では，自動運転で用いることを想定して RGB カメラから周辺画像を取得し，画像内から人や四輪車など特定の物体の認識を行う．

画像内から特定の物体を認識する手法は様々な研究が行われている．その中でも DET(Detection Error Tradeoff)カーブによって，人や四輪車など運転をする上で認識する必要がある物体に対して，高い認識精度を示した Joint HOG 特徴量を選択した．

そして，NVIDIA 社製の GPU を搭載した組み込み開発キットである Jetson TK1 を用いて，GPGPU 技術の実装により計算処理の高速化を目的とした．

3. JointHOG 特徴量

局所特徴量を使用した物体検出の精度比較として藤吉研究室が DET カーブによる識別精度の比較を行っている[5]．図 1 が人物検出における DET カーブを示し，図 2 が車両による車両検出における DET カーブを示す．

DET カーブは縦軸が未検出率，横軸が誤検出率を表し，原点に近づく程高精度であることを示す．(1)式によって未検出率，(2)式によって誤検出率を算出する．

$$未検出率 = 1 - \frac{Pos画像の正解数}{Pos画像の総数} \quad (1)$$

$$誤検出率 = \frac{Neg画像の不正解数}{Neg画像の総数} \quad (2)$$

(1)，(2)式の Pos 画像は検出対象のみが映った画像を示し，Neg 画像は検出対象が映っていない画像を示す．

これらの結果から本研究では人や四輪車など運転をする上で認識する必要がある物体に対して，高い認識精度を示した特徴量として 2 つの HOG 特徴を RealAdaBoost によって組み合わせた Joint HOG 特徴を用いる．

†1 東京電機大学大学院情報学専攻
Informatics, Graduate School of Tokyo Denki University,
†2 東京電機大学情報システムデザイン学系
Division of Information System Design, Tokyo Denki University,

Vol.2017-DPS-170 No.10
Vol.2017-CSEC-76 No.10
2017/3/2

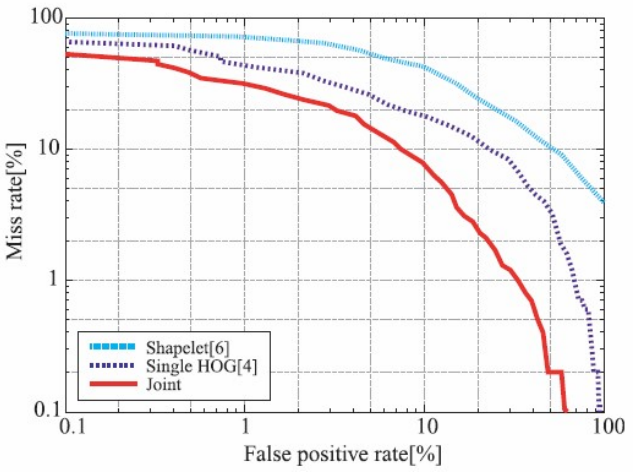

図 1　人物検出における DET カーブ

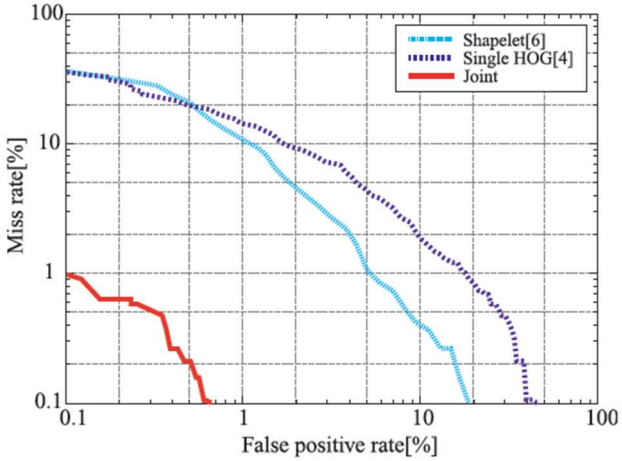

図 2　車両検出における DET カーブ

HOG 特徴[6]とは N.Dalal と B.Triggs らが提案したもので，図 3 のようにグレイスケール化した画像データからピクセル間の輝度値の勾配の方向と強度を算出する．複数のピクセルを 1 つのセルとして勾配ベクトルの正規化を行い，得られる勾配ベクトルをヒストグラムとしてまとめることで局所特徴量とする．

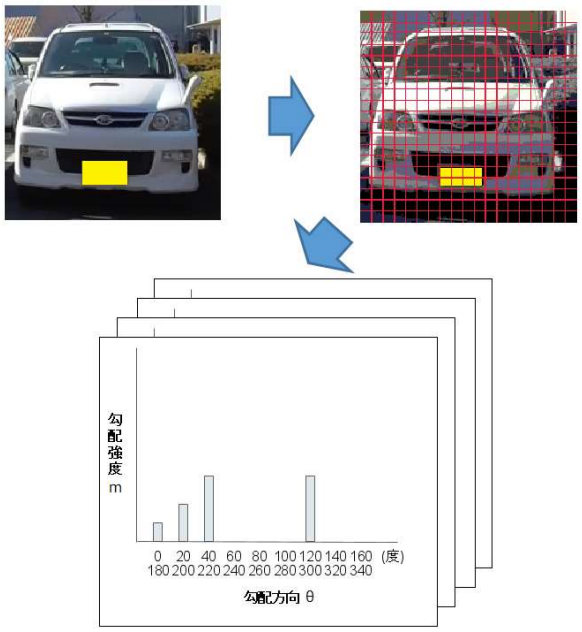

図 3　HOG 特徴量の算出手順

輝度値の勾配ベクトルは縦横の隣り合うピクセルとの輝度値の差から勾配強度と勾配方向を算出する．輝度値を L，勾配強度を m，勾配方向を θ とした時，(5)式によって対象のピクセルから縦横に隣り合うピクセルの輝度値の差を求め，(3)式によって勾配強度，(4)式によって勾配方向が求められる．

$$\mathrm{m}(x,y) = \sqrt{f_x(x,y)^2 + f_y(x,y)^2} \tag{3}$$

$$\theta(x,y) = \tan^{-1}\frac{f_x(x,y)}{f_y(x,y)} \tag{4}$$

$$\begin{cases} f_x(x,y) = L(x+1,y) - L(x-1,y) \\ f_y(x,y) = L(x,y+1) - L(x,y-1) \end{cases} \tag{5}$$

5×5ピクセルを 1 つのセルとして扱い，算出した勾配強度 m と勾配方向θを正規化して勾配方向ヒストグラムとする．

勾配ヒストグラムは図 4 のように勾配方向と勾配強度から作成される．勾配方向 θ は 0°~360°の値を算出するが特徴量として勾配の向きを考慮する必要がないため，0°~160°を 20°ごとに分割して，9 方向の勾配方向ヒストグラムとしている．

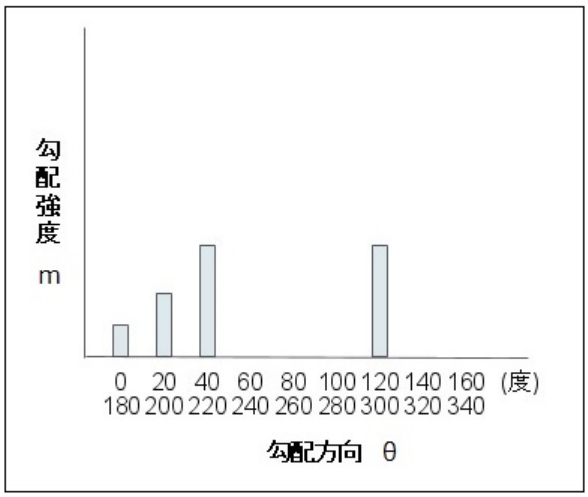

図 4　勾配ヒストグラム

Joint HOG 特徴は尾崎貴洋，山内悠嗣，藤吉弘亘らが提案した 2 箇所の Low-level な識別器である HOG 特徴を RealAdaBoost と呼ばれる機械学習アルゴリズムによって組み合わせることでより強い識別器を作成する手法である．

RealAdaBoost は AdaBoost の派生で，学習サンプルとして用意した識別対象が映った画像と映っていない画像を 2 箇所の HOG 特徴を用いて大量に識別し，それらの識別結果の成否から確率密度分布に応じて，2 箇所の HOG 特徴に対して 0~1 の実数値化を行う．その値に応じて識別に有効な識別器の組み合わせを採用することで，より強い識別器を構築する．

本研究では HOG 特徴を RealAdaBoost によって 2 段階学習させた Joint HOG 特徴をデータベースとして使用する．

4. GPGPU

GPGPU 技術は，主に画像処理に使用される GPU の演算資源を汎用計算の処理に使用する技術である．GPU は高速な画像処理を行うために，高速の VRAM と接続され，画像の描画に使用される複数のプロセッサを持っていることを特徴としている．GPU に搭載されているプロセッサは CPU に比べると構造が単純であり，限定的な機能となってしまうものの，GPU に搭載された複数のプロセッサを利用した単純な計算の並列処理を行うことが可能で，処理の高速化に利用されている．

本研究では自動車の運転中にリアルタイムでの周辺画像からの特定の物体認識を行うために処理の高速化を目的として GPGPU の技術を使用した．

GPUGPU プログラミング環境には NVIDIA 社が提供している GPU 向けの統合環境開発環境である CUDA を利用している．CUDA では主に C 言語に近い構文を利用してプログラミングできるため汎用的に利用できる利点もある．

しかし，CUDA を使用する上で C 言語にはない制約がある．

(1) CPU と GPU でアクセスできるメモリが別であり，GPU 側から CPU 側のメモリにはアクセスできず，CPU 側からも GPU 側のメモリの領域確保と解放，データの相互転送しか行えない．

(2) CPU から呼び出されるカーネル関数はすべて戻り値が void でなくてはならない．

(3) CPU と GPU のメモリ間でデータ転送が行えるもののオーバーヘッドが大きい．

まず，制約(1)によって GPU 側で演算を行う場合，CPU 側のメモリのポインタが使用できず，CPU 側から GPU 側のメモリ領域の確保を行い，演算に使用するデータの転送をする必要がある．制約(1)と(2)により GPU の演算結果もまた戻り値や CPU 側のメモリのポインタが使えないため，演算結果を GPU 側のメモリから CPU 側のメモリへデータの転送を行う必要がある．そして，制約(3)で述べたようにデータ転送自体のオーバーヘッドが大きいため，考えなしに処理が CPU と GPU を行ったり来たりした場合，データ転送によって CPU のみで計算した時よりもオーバーヘッドが大きくなってしまう．

5. 評価

5.1 実験環境

JointHOG 特徴を用いた物体検出の処理を，CUDA の実装の有無による処理時間で比較し，評価を行った．

本研究では自動運転に使用されることを想定しているため実験には NVIDIA 社で開発された組み込み開発キットである Jetson TK1 を使用している．スペックは表 1 に示す．

表 1　Jetson TK1 のスペック

CPU	ARM Cortex-A15
メモリ	2GB
GPU	NVIDIA Kepler
OS	Ubuntu 14.04
CUDA コア数	192

物体の認識にはサンプルの集めやすさを考慮して四輪車を対象とした．データベースの作成に使用するサンプル画像や実際に四輪車の検出の実験に使用する画像はあらかじめカメラで動画を撮影して利用した．

動画の撮影は東京電機大学鳩山キャンパスの駐車場にて，データベース作成用と四輪車の検出実験用の 2 回，解像度を 1080p である 1920×1080 ピクセル，フレームレートは 29.97fps で行う．

実験は四輪車の検出実験用の動画から四輪車の映った画像 100 枚を用意し，CUDA を実装しなかった場合と実装

Vol.2017-DPS-170 No.10
Vol.2017-CSEC-76 No.10
2017/3/2

した場合に分けて処理を行い，処理にかかる時間を計測する．

5.2 データベースの作成

四輪車の検出を行うために，学習サンプルとして図 5 のように四輪車を映したポジティブ画像と図 6 のような四輪車を映していないネガティブ画像のトリミングを行う．学習サンプルから JointHOG 特徴量を算出して，四輪車の検出に用いるデータベースを作成する．

図 5　ポジティブ画像

図 6　ネガティブ画像

ポジティブ画像を 2952 枚とネガティブ画像を 3035 枚用意してデータベースを作成したところ，図 7，図 8 のように画像内から四輪車を検出することに成功した．

図 7　四輪車の検出成功例 1

図 8　四輪車の検出成功例 2

赤の枠で囲まれた箇所が四輪車と認識した箇所である．このとき作成したデータベースを実験に使用する．

5.3 四輪車の検出手法

物体検出に用いる評価プログラムのフローチャートを図 9 に示す．

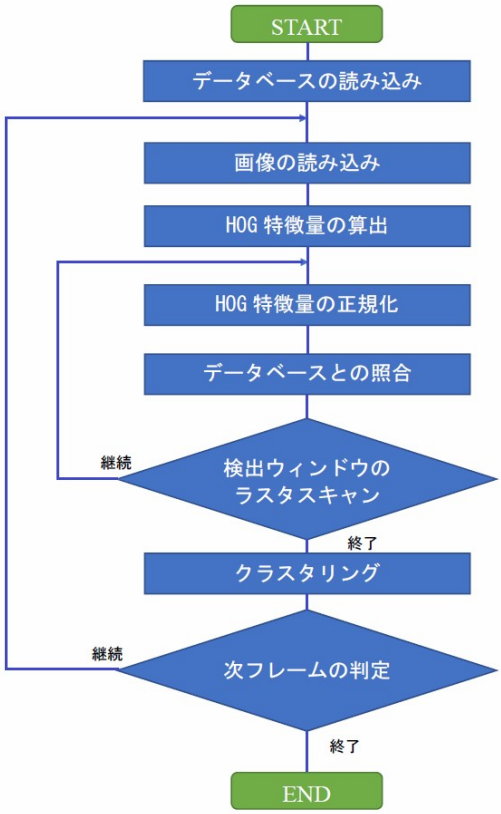

図 9　物体検出のフローチャート

四輪車の検出の実験に使用する画像は撮影した動画から 1 フレーム切り出し，解像度を 540×360 ピクセルに圧縮して使用する．

画像から四輪車の検出にはまず，あらかじめ作成したデータベースの読み込みを行う．次に，四輪車の検出を行う画像を読み込み，画像全体の HOG 特徴量の算出を行う．

Vol.2017-DPS-170 No.10
Vol.2017-CSEC-76 No.10
2017/3/2

画像内の左上端に 60×50 ピクセルの検出ウィンドウを作成し，右に 6 ピクセルずつずらして，右端に到達したら左端まで戻り，下に 6 ピクセルずらして再度右にずらしていく．検出ウィンドウは移動の度に検出ウィンドウ内の特徴量の正規化を行い，作成したデータベースと比較することで検出ウィンドウ内に四輪車が存在するかの判定を行う．これを繰り返して右下端まで到達したら検出ウィンドウを拡大して左上端から繰り返す．検出ウィンドウのサイズ変更は 4 回行い，初期値の 60×50 ピクセルを 1.0 倍とした時，1.0 倍，1.5 倍，2.0 倍，2.5 倍とサイズ変更する．検出ウィンドウのサイズ変更に対応して，検出ウィンドウの移動幅も初期値の 6 ピクセルから 2 ピクセルずつ増加して，6 ピクセル，8 ピクセル，10 ピクセル，12 ピクセルに移動幅を変更する．

画像内の検出ウィンドウの走査が完了したら，図 10 のように，検出ウィンドウ内が四輪車であると判定した検出ウィンドウが同一車両を多重検出した状態になる．

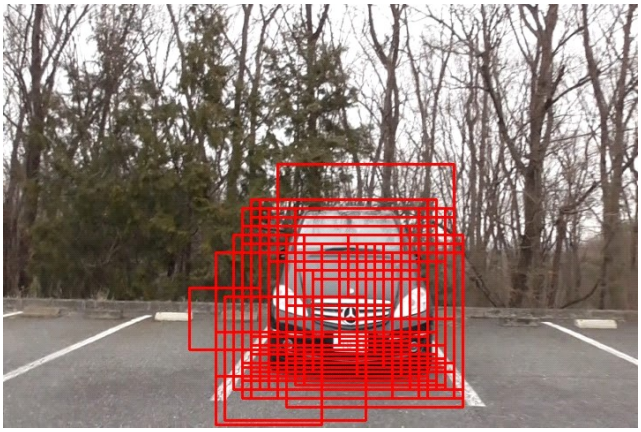

図 10　検出ウィンドウの走査後の判定結果

そこで，これらの検出ウィンドウを四輪車ごとに 1 つにまとめるために Mean Shift と呼ばれるクラスタリング手法を用いる．図 10 を Mean Shift によってクラスタリングした結果を図 11 に示す．

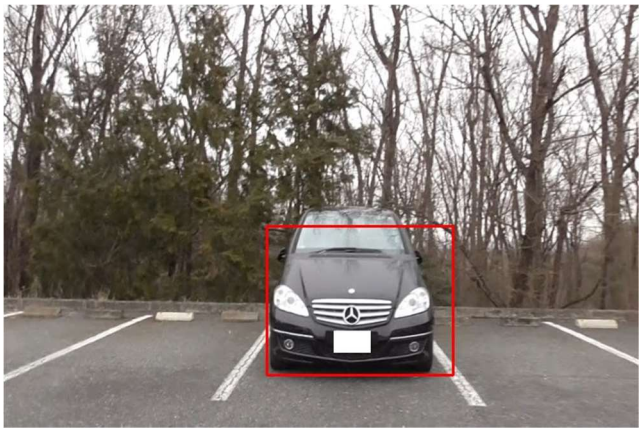

図 11　クラスタリング後の判定結果

すべての検出ウィンドウのクラスタリングを行い，現れたウィンドウを四輪車の検出箇所として記録する．

処理時間の測定は，1 フレームの処理にかかる時間を合計時間として，さらに処理を初期化時間，走査時間，クラスタリング時間の 3 つに分割して計測を行う．

分割の内訳は，処理する画像データの読み込みと画像内の HOG 特徴量の算出にかかる時間を初期化時間，画像内を検出ウィンドウがラスタスキャンを行い，データベースを基に画像内の四輪車を検出する時間を走査時間，検出ウィンドウのラスタスキャンによる検出結果をクラスタリングする時間をクラスタリング時間とする．1 フレームの合計時間はこれらの処理を含んだ画像 1 枚の検出にかかる処理時間を指す．

5.4 CUDA の実装

GPU のプロセッサは単純計算の並列処理は得意だが，if 構文などの条件分岐を苦手としており，処理に時間がかかる．そのため，並列計算に必要なスレッド数をあらかじめ計算できる処理が CUDA の実装に都合が良い．

そして，CPU と GPU のメモリが別であり，それぞれへのデータ転送には多大な時間を要するという問題があるので，CPU 側と GPU 側のデータ転送を極力減らす必要がある．

これらの特徴から単純に処理を並列化すれば高速化するということにはならず，データ転送における処理時間の増加を減らしながら，条件に合うように並列処理を行う方法を考える必要がある．

本提案では GPU のプロセッサそれぞれに検出ウィンドウを 1 窓担当させ，検出ウィンドウ内の特徴量の正規化とデータベースによる判定を並列的な処理を行う．検出ウィンドウ内の特徴量の正規化とデータベースによる判定をまとめて GPU に行わせることで，算出した特徴量を CPU 側へデータ転送する処理を省くことができ，計算後の CPU 側への転送するデータも，データベースによる判定結果だけのためオーバーヘッドを抑えることができる．

そして，検出ウィンドウによる画像内のラスタスキャンを GPU で並列処理する場合，画像内の HOG 特徴量は GPU 側でのみ使用することになる．そのため，HOG 特徴量の算出も GPU で並列処理を行うことが考えられる．

HOG 特徴量の算出処理は検出に使用する画像の解像度である 540×360 ピクセルのすべてのピクセルで勾配ベクトルを算出する処理であり，計算のループ回数が検出に使用する画像のピクセル数と固定であるため並列化に適している．さらに，演算結果も CPU 側へデータ転送せずにそのまま次の検出ウィンドウによるラスタスキャンで使用するため，データ転送の回数を抑えながら並列化による処理の短縮が見込める．

Vol.2017-DPS-170 No.10
Vol.2017-CSEC-76 No.10
2017/3/2

検出ウィンドウの判定後のクラスタリングも GPU で処理したいが四輪車だと判定される検出ウィンドウの数が不定であることから計算のループ回数がわからず，条件分岐の使用が必要となり並列処理による高速化が困難であるため，本提案ではクラスリングは CPU で処理している．

HOG 特徴量の算出と検出ウィンドウによるラスタスキャンは，処理としては続いているものの並列化するスレッド数が HOG 特徴量の算出が検出する画像のピクセル数，検出ウィンドウによるラスタスキャンは検出ウィンドウの移動回数と違うため，別の処理としてカーネル関数を作成する．

並列化する計算処理が決定したら，GPU で行う演算に必要なデータのメモリ領域を GPU 側のメモリに確保し，CPU 側のメモリから GPU 側のメモリへデータ転送しなくてはならない．

まず，GPU 側へ転送する必要があるデータは HOG 特徴量の算出に必要な検出に使用する画像データと検出ウィンドウの判定に使用するデータベースが必要となる．

次に，CPU 側へ転送する必要があるデータは検出ウィンドウによるラスタスキャンで得られた検出ウィンドウの判定結果である．

メモリ領域の確保と検出ウィンドウの判定に使用するデータベースの転送は 1 度行うだけでよいので，1 フレームごとの処理時間に影響を与えない．対して，検出に使用する画像データと検出ウィンドウの判定結果の転送は 1 フレームごとに転送が必要であり，1 フレームの処理時間に影響を与える

図 12 に CUDA 実装後の評価プログラムのフローチャートを示す

処理時間の測定は，CUDA の実装前と同様に 1 フレームの処理にかかる時間を合計時間として，さらに処理を初期化時間，走査時間，クラスタリング時間の 3 つに分割して計測を行う．

分割の内訳は，処理する画像データの読み込みと GPU 側のメモリへの転送，画像内の HOG 特徴量の算出にかかる時間を初期化時間，画像内を検出ウィンドウで並列的にラスタスキャンを行う時間と判定結果を CPU 側のメモリへ転送する時間を走査時間，検出ウィンドウのラスタスキャンによる検出結果をクラスタリングする時間をクラスタリング時間とする．1 フレームの合計時間はこれらの処理を含んだ画像 1 枚の検出にかかる処理時間を指す．

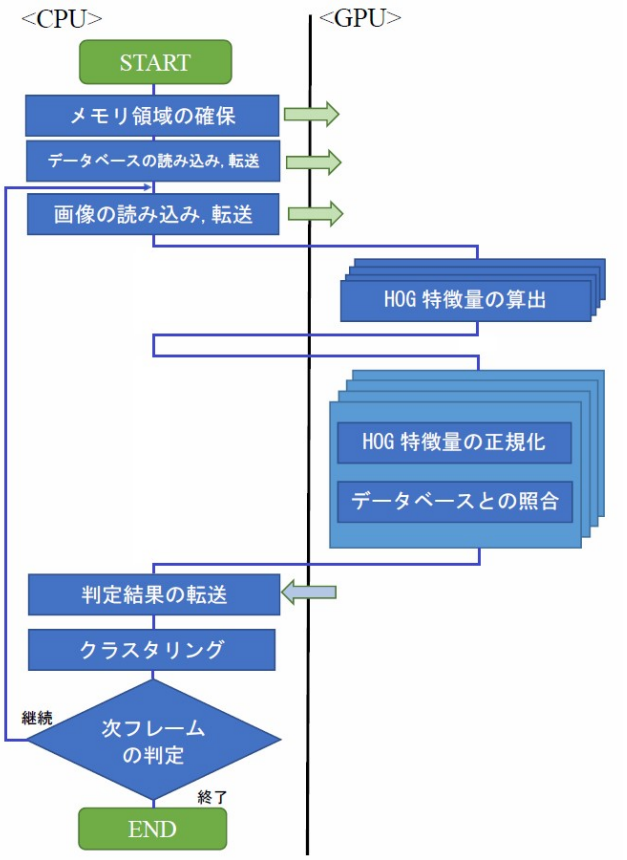

図 12　CUDA 実装後の物体検出のフローチャート

5.5 実験結果

実験結果は四輪車の映った画像 100 枚の初期化時間，走査時間，クラスタリング時間，合計時間を平均して表にまとめる．

まず，CUDA を実装していない場合での画像内の四輪車の検出処理の平均時間を表 2 に示す．

表 2　CUDA を実装していない場合での平均処理時間(s)

初期化時間	0.14
走査時間	1.99
クラスタリング時間	0.03
合計時間	2.17

次に，CUDA を実装した場合での画像内の四輪車の検出処理の平均時間を表 3 に示す．

表 3　CUDA を実装した場合での平均処理時間(s)

初期化時間	0.05
走査時間	0.08
クラスタリング時間	0.03
合計時間	0.16

CUDA の実装の有無で比較のために測定結果をまとめて

図 13 に示す

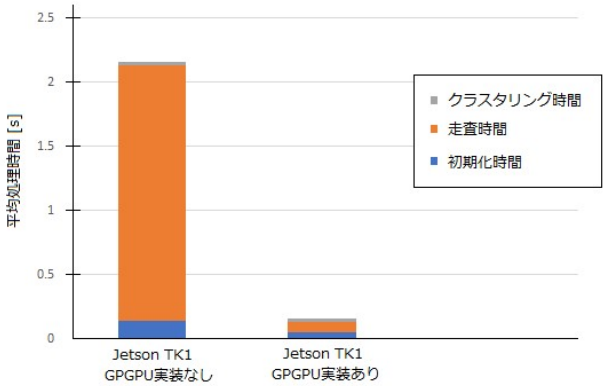

図 13　CUDA の実装の有無による平均処理時間の比較

5.6 まとめ

CUDA によって画像内の HOG 特徴量の算出部分を並列化することで初期化時間を短縮し，検出ウィンドウによる画像内のラスタスキャンを並列化したことで走査時間の短縮に成功した．

特に走査時間は 1 フレームの処理時間の大部分を占めているので，並列化の恩恵を受けて大幅な短縮に成功している．これを受けて，1 フレームの処理時間は 90%の削減に成功した．

5.7 GPU を搭載したコンピュータでの評価

今後，組み込み開発キットの更なる発展を見込み，処理性能の向上でどこまで高速化が行えるかを評価するために，「Jetson TK1」とは別に，GPU を搭載したコンピュータを用意して評価を行う．

GPU を搭載したコンピュータのスペックをそれぞれ表 4 に示す．

表 4　GPU 搭載コンピュータのスペック

CPU	Intel Core i7-6700
メモリ	32GB
GPU	NVIDIA GeForce GTX 1060 (6GB モデル)
OS	Windows10 (64bit)
CUDA コア数	1280

実験方法は「Jetson TK1」での評価実験と同様で，CUDA を実装した評価プログラムで，四輪車の映った画像 100 枚の処理を行い，平均処理時間を測定した．

5.8 実験結果

GPU を搭載したコンピュータを用いた，四輪車の映った画像 100 枚の平均検出処理時間の測定結果を表 5 に示す．

表 5　GPU 搭載コンピュータの平均処理時間(s)

初期化時間	0.000
走査時間	0.016
クラスタリング時間	0.000
合計時間	0.016

Jetson TK1 と処理時間を比較のために測定結果をまとめて図 14 に示す．

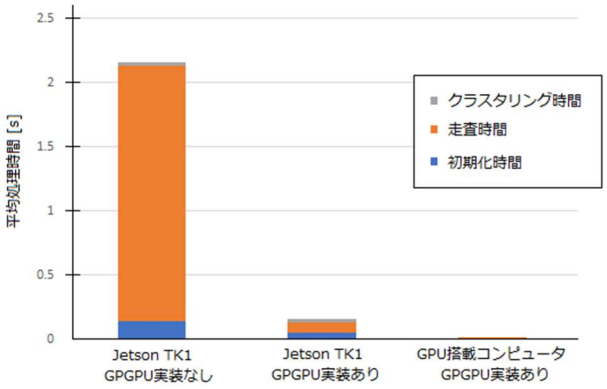

図 14　処理性能による平均処理時間の比較

5.9 まとめ

CPU と GPU の性能次第で 60fps に相当する処理速度まで到達し，スペック面での進歩でも，処理時間の短縮が行えることを示した．

6. おわりに

本稿では Joint HOG 特徴量を用いた検出を，GPGPU で高速化する手法を提案した．そして，CUDA を用いて計算処理を並列化することで，1 フレームにかかる処理時間を 90%削減することに成功した．

今後，組み込み開発キットの処理性能の向上でも処理の高速化が見込めることを示した．

しかし，クラスタリング時間が画像内の車の台数や車までの距離による検出判定の増加や誤認識の影響で処理時間が安定しないことが今後の課題となった．CUDA の実装部分でも述べたが，クラスタリング処理は検出ウィンドウの判定結果を基に処理のループ回数が決まるため，事前にループ回数は不定であり並列化が困難となっている．そのため，現状ではクラスタリング時間の高速化の手法として，検出精度とトレードオフの関係になってしまうが検出ウィンドウの移動幅を大きくとることで移動回数を減らし，検

出回数自体の調整により最適化を図っている．

参考文献

[1] “戦略的イノベーション創造プログラム(SIP) 自動走行システム 研究開発計画”. http://www8.cao.go.jp/cstp/gaiyo/sip/keikaku/6_jidousoukou.pdf, (参照 2017-01-10).
[2] 堀江忠裕, 松原一樹, 泉隆. 車両前方画像における影に着目した先方車両抽出 -EDH 特徴量による車両識別器の構築-. 日本大学理工学部 学術講演会論文集, 2012, p. 517-518.
[3] 出口大輔, 井出一郎, 村瀬洋. 画像認識と GPU. 日本ロボット学会誌, 2010, vol. 28, no. 3, p. 268-271.
[4] 尾崎貴洋, 山内悠嗣, 藤吉弘亘. Joint HOG 特徴を用いた 2 段階 AdaBoost による車両検出. 動的画像処理実利用化ワークショップ (DIA2008). 2008, p. 101-106.
[5] 山下隆義, 藤吉弘亘. 特定物体認識に有効な特徴量. 情報処理学会 研究報告. CVIM 165, 2008, p. 221-236.
[6] Dalal, N. and Triggs, B.. Histograms of Oriented Gradients for Human Detection. In Proc CVPR. 2005, p. 886-893.