2007－CVIM－157 (25)
2007／1／12

GPUを用いた全方位視覚センサの実時間動き推定

松井 良平† 長原 一†† 谷内田 正彦††

†,†† 大阪大学大学院基礎工学研究科 〒560-8531 大阪府豊中市待兼山町 1-3
E-mail: †matsui@yachi-lab.sys.es.osaka-u.ac.jp, ††{nagahara,yachida}@sys.es.osaka-u.ac.jp

あらまし 本稿では，全方位視覚センサの回転量の実時間推定の手法を提案する．本手法は，全方位画像を三軸直交の円筒面座標に投影する．その円筒画像の Integral Projection パターンをマッチングする事で，回転運動をロバストに推定する．一般的にはこのような大域的なマッチングによる手法は計算コストが高いため，実時間処理には向かない．本研究では，GPU(Graphic Processing Unit) の特徴を活かして提案手法の実装を行う事で，実時間処理を実現した．また，実験においてその処理の高速性や精度を確認した．

Real-Time Motion Estimation of Omnidirectional Sensor on GPU

Ryohei MATSUI†, Hajime NAGAHARA†† and Masahiko YACHIDA††

†,††Graduate School of Engineering Science, Osaka University Machikaneyama 1-3, Toyonaka, Osaka, 560-8531 Japan
E-mail: †matsui@yachi-lab.sys.es.osaka-u.ac.jp, ††{nagahara,yachida}@sys.es.osaka-u.ac.jp

abstract In this paper, we propose a real-time rotation estimation method for an omnidirectional sensor. We projected from an omnidirectional image to three cylindrical images along 3D rectangular coordinates and matching the integral projection pattern of the cylinder image to estimate the rotation angles around the coordinates. The method such as proposed that matches global pattern of image requires high computation cost even if the methods robustly estimate the motions under the local change or noise. We implemented the proposed method into Graphic Processing Unit(GPU) for realizing a real-time processing. We confirmed that the proposed method estimate the enough accurate rotation and worked on video rate.

1 はじめに

HyperOmni Vision[1] などの全方位視覚センサは，周囲 360 度の情報を一度に得ることができる．そのため，この実時間・広視野角の撮像を利用してロボットのナビゲーションやリアルタイムモニタリング，テレプレゼンスなどに利用されている．ロボットの自己位置の推定やテレプレゼンスにおける画像の揺れ補正には，センサ自体の傾き (回転) や並進などの動きを推定することが重要である．本報告では全方位視覚センサの動きを取得した画像から実時間で回転運動を推定する手法を提案する．一般的に回転運動の計測には，ジャイロセンサなどのモーションセンサが用いられる．画像の揺れ補正などの応用では，画像から動きが推定できれば，追加のモーションセンサが必要なくなり，システムが簡易になるという利点や，モーションセンサと画像との同期を考える必要がないと言った利点がある．

画像からの動き推定には大きく分けて，特徴ベースの手法 [2] と領域ベースの手法 [3, 4] がある．特徴ベースの手法は，オプティカルフローやトラッカなどを用いて点特徴のモーションベクトルを求め，そのベクトル群から運動を推定する手法で様々な運動推定モデルやアルゴリズムが提案されている [2]．点特徴を用いるため計算コストは低いが，ノイズや局所的変化の影響に弱い．また，モーションベクトルの導出時にフレーム間での特徴の対応付けの必要があり，大きな運動変化には対応できないという問題をもつ．同様のアプローチを全方位画像の射影関係を考慮して拡張することで，全方位視覚センサに対応させた手法も提案されている [5]．

一方，領域ベースの手法は画像そのものや，画像から直接的に得られるパターンをマッチングすることで，運動を推定するアプローチである．この手法は，一般

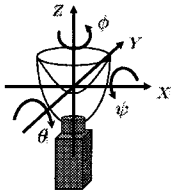

図1　全方位センサの運動

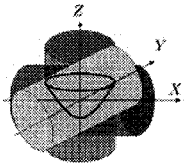

図2　3軸方向の円筒

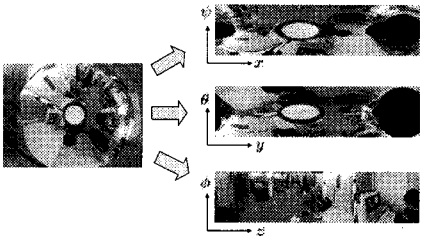

図3　パノラマ画像

に計算量が多いが大域的パターンを用いてマッチングを行うため，ノイズや局所的な動きの影響を受けにくい．また，[3, 4] では Integral projection という動きに垂直な方向への加算処理を行うことで，2次元のマッチング問題からから1次元のマッチング問題にすることで，領域ベースの手法ではあるが計算量の大幅な削減を実現している．このアプローチを全方位視覚センサに拡張した手法としては，全方位画像を球面座標上での相関を求める事で回転運動を推定する手法が提案されている [6].

本研究では，領域ベースである Integral projection を用いた手法を全方位画像に適用する．3次元の回転成分を分離するために，全方位画像を3軸直交系の軸周りの円筒状の面に射影する．この射影した画像に対して Integral projection を適用し，1次元のマッチングを行う．そのためノイズに対してロバストな回転量の推定ができる．さらに我々は，GPU で実装することにより実時間で処理を行う．Integral projection を用いた手法はアルゴリズムが単純で GPU での実装に向いており高速化が望める．

GPU は本来グラフィック処理を行うものであるが，近年急速に性能を向上させ，従来固定機能であったパイプラインの一部を現在ではプログラムで制御することが可能になってきた．また，GPU は高速な並列計算機でもある．さらに GPU はグラフィックスワークステーションや特殊な画像処理ハードウェアに比べ非常に安価なため，システムの構成も安価に行うことができる．そのため，本来グラフィックス専用のハードウェアである GPU を物理シミュレーションや画像処理などの，描画以外の汎用的な計算に用いる GPGPU (Genaral Purpose computation on GPUs) と呼ばれる研究が盛んに行われている [7]．例えば，[8] では GPU を用いて画像からオプティカルフローを計算し，その視覚化を行っている．[9] では顔検出アルゴリズムを GPU で実装することで処理の高速化を行っている．また，GPU では平面射影のような画像の幾何変換も高速に処理することができる [10]．本研究では，アルゴリズムのほぼ全てを GPU で実装し，実時間で処理を実現した．

2　動き推定の手法

ここでは，全方位視覚センサの運動モデルと全方位画像からパノラマ画像への変換，動き推定のアルゴリズムについて述べる．

2.1　全方位視覚センサの回転

全方位センサの回転は図1のようにミラーの焦点を原点とした，X, Y, Z 軸それぞれの方向の軸周りの回転で表すことができる．したがって推定するパラメータは X 軸周りの回転 ψ，Y 軸周りの回転 θ，Z 軸周りの回転 ϕ の計3つである．

双曲面ミラーを用いた全方位センサは単一視点を持つため，ミラーの焦点を中心とした天球として撮像される．そのため，ミラー焦点を中心とするオイラー角で表される回転運動はミラー焦点を中心とした3軸方向の円筒面上でのパターンのずれとして現れる．そこで，本研究では図2のようなミラーの焦点を中心としたそれぞれの軸方向の円筒面に全方位画像を投影する．そうすると図3のような画像が得られる．本研究では以降，この円筒面に射影した画像をパノラマ画像と呼ぶことにする．パノラマ画像へ変換することで各軸周りの回転は各パノラマ画像上で軸に垂直な方向へのパターンのずれとして現れる．このことを利用して回転量の推定を行う．例えば，全方位センサが Z 軸周りに回転したとすると図4のように Z 軸方向のパノラマ画像上で横方向のずれとして現れる．X, Y 軸方向に関しても同様で，このずれを求めることができれば全方位センサの各軸周りの回転量を推定することができる．

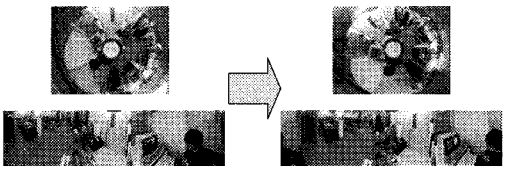

図4　Z 軸周りの回転

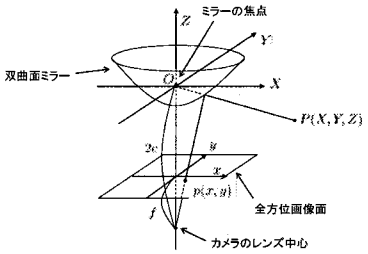

図5　全方位センサの幾何

2.2　パノラマ変換

全方位センサ HyperOmni Vision (図5) は一般のカメラと双曲面ミラーを組み合わせた構造で，単一視点を持つという特徴から，ミラーのパラメータを用いてパノラマ画像への変換が可能である．

双曲面ミラーを用いた全方位画像上の点 $p(x,y)$ と3次元空間中の点 $P(X,Y,Z)$ の間には

$$x = X \times f \times \frac{(b^2-c^2)}{(b^2+c^2)Z - 2bc\sqrt{X^2+Y^2+Z^2}}$$
$$y = Y \times f \times \frac{(b^2-c^2)}{(b^2+c^2)Z - 2bc\sqrt{X^2+Y^2+Z^2}} \quad (1)$$

という関係がある．ここで，b,c は双曲面のミラーパラメータ，f は全方位視覚センサのカメラの焦点距離である．

パノラマ画像へ変換は，例えば Z 軸方向の場合，図6のように，単位円の円筒面を考え，さらに，Z 軸方向の範囲を考慮し

$$X = \cos\phi,\ Y = \sin\phi,\quad 0 \le \phi < 2\pi,$$
$$Z_{min} \le Z \le Z_{max} \quad (2)$$

とする．そして，これらの式 (1)，(2) を連立することで全方位画像からパノラマ画像への変換を行うことができる．また，X,Y 軸方向のパノラマ画像に関しても，Z 軸方向と同様に変換することができる．

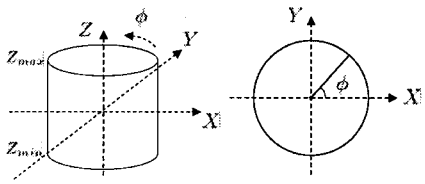

図6　円筒面の幾何

2.3　推定のアルゴリズム

本研究では前述のフレーム間のパノラマ画像のずれを Integral projection を利用して求めることにより，全方位センサの動き推定を行う．ここでは Z 軸方向の並進と Z 軸周りの回転の推定のみについて説明するが，X,Y 方向についても同様である．

Z 軸方向のパノラマ画像に対する垂直方向の Integral Projection は

$$\boldsymbol{p}_{\phi,t} = \sum_Z I_t(Z,\phi) \quad (3)$$

で計算される．ここで，$I_t(Z,\phi)$ は第 t フレーム目の位置 (Z,ϕ) の画素の輝度である．これを，$t-1$ フレーム目の projection

$$\boldsymbol{p}_{\phi,t-1} = \sum_Z I_{t-1}(Z,\phi) \quad (4)$$

と比較することにより，Z 軸周りの回転量 δ_ϕ と Z 軸方向の並進量 δ_Z を求めることができる．つまり現在のフレームの projection 結果と前のフレームの projection 結果をマッチングすることで動きを推定することができる．δ_ϕ は次式 (5),(6) を用いて求める．

$$D_v(d_\phi) = \sum_{i=0}^{N-1} \left|\boldsymbol{p}_{\phi,t}(i) - \boldsymbol{p}_{\phi,t-1}(i+d_\phi)\right| \quad (5)$$
$$\delta_\phi = \arg\min_{d_\phi} D_v(d_\phi) \quad (6)$$

ここで，$\boldsymbol{x}(i)$ はベクトル $\boldsymbol{x}$ の i 番目の要素を表し，d_ϕ はマッチングの際のずれ幅を表す．

このように，Integral Projection を利用すると2次元のマッチングから1次元のマッチング問題になり，計算コストを大幅に削減できる．また，Z 方向に加算することや大域的パターンを用いてマッチングを行うことから，ノイズや局所的変化に強いという利点を持つ．なお，X 軸と Y 軸方向のパノラマ画像は 360 度の情報がないため，パノラマ画像の一部に Integtal projection を適用することになる．

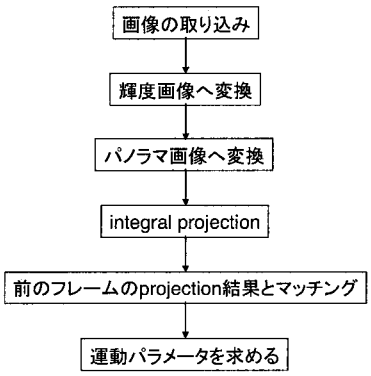

図7　推定手法の流れ

手法の流れをまとめると図7のようになる．まず，入力画像を X,Y,Z 軸方向のそれぞれのパノラマ画像へ変換する．作成した3つのパノラマ画像に対して，それぞれ垂直方向のIntegral projectionを計算する．つまり，1フレームにつき3つのProjectionを計算することになる．最後に，Projectionした結果を前のフレームのProjection結果とマッチングすることで全方位センサの運動パラメータを求める．

3　GPUへの実装

GPUはグラフィックス処理において，座標変換や色計算，レンダリング等を行う装置である．GPU上のプログラムは3次元空間における幾何演算を高速に実行する頂点シェーダと，2次元画像の処理を行うピクセルシェーダから構成される．これらを利用することで，様々な処理を高速に行うことができる．

頂点シェーダへの入力は頂点毎の座標やテクスチャ座標であり，ピクセルシェーダへの入力は頂点シェーダからの出力やテクスチャである．GPUで処理した結果（一般にはピクセルシェーダの出力）はテクスチャへ書き込むことで，次の処理に利用することができる．また，ピクセルシェーダは出力テクスチャのピクセルを単位として記述されることが特徴の1つである．ピクセルシェーダでは，出力テクスチャの1ピクセルを計算する際に複数のピクセル情報を組み合わせるgather処理は高速に実行できるが，1ピクセルの情報を出力テクスチャの複数のピクセルに反映させるscatter処理を行うことは困難である．さらに，シェーダでは全ピクセルに対して加算や積算のような同じ計算をするのは得意で高速な処理が可能である．しかし，if文のような制御文のある処理は少し速度が低下する．なお，GPUの詳しい特徴については参考資料[7]を参照されたい．

本節では，輝度画像への変換とパノラマ画像への変換，Integral projection，Projection結果のマッチングをGPUでどのように実現するかについて述べる．

なお，本研究ではGPUの機能を利用するためのAPIとしてDirectX Graphicsを使用し，各シェーダを記述する言語としてHLSL(Heigh-Level Shading Language)を用いる．また，演算精度を確保するために単精度浮動小数点テクスチャを用いる．

3.1　輝度画像への変換

Integral projectionでは輝度値を利用するため，入力画像を輝度画像へ変換する．そのために，まず入力画像をテクスチャへ保存しておく．輝度値はYCbCr色空間へのY値を利用する．入力画像のRGB色空間からYCbCr色空間のYへの変換は式(7)で与えられる．

$$Y = 0.2990R + 0.5870G + 0.1140B \qquad (7)$$

つまり，ベクトル (R,G,B) とベクトル $(0.2990, 0.5870, 0.1140)$ との内積を計算すればよい．GPUでこれを行うには，ピクセルシェーダでベクトルにテクスチャ（入力画像）を読み込み，$(0.2990, 0.5870, 0.1140)$ との内積を計算する．この計算結果を書き込み用のテクスチャへ出力することで輝度画像を生成する．ピクセルシェーダの擬似コードは以下のようになる．

```
float3 rgb2y={0.2990,0.5870,0.1140};
float3 in=tex2D(Input,TexUV);
return dp3(in,rgb2y);
```

ここでtex2Dは第1引数で与えられるsamplerオブジェクトである入力テクスチャから，float2型の引数で指定される座標におけるピクセル値をサンプリングする関数である．dp3は3要素のベクトル同士の内積を計算する関数である．シェーダ内ではベクトル演算が可能で3要素や4要素のベクトルの内積を高速に計算できる．

3.2　パノラマ画像への変換

GPUでパノラマ画像への変換を行う場合は頂点シェーダとピクセルシェーダの両方を利用する．まず，

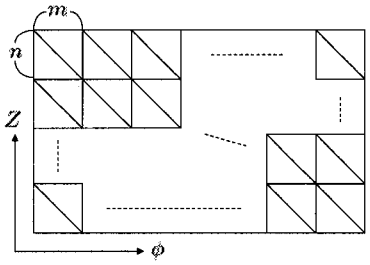

図8　パノラマ変換用メッシュ

図8のような，全体のサイズがパノラマ画像と一致する平面のメッシュを用意する．このメッシュは $m \times n$ のサイズの三角ポリゴンから構成されており，この頂点座標を頂点シェーダへの入力とする．頂点シェーダで式 (1),(2) を用いて対応するテクスチャ座標を計算する．このとき，テクスチャは全方位画像の輝度画像である．その後，頂点シェーダで計算したテクスチャ座標を用いて，ピクセルシェーダでテクスチャを読み込む（図9）．

頂点シェーダの擬似コードは以下のようになる．

```
float Z = -Zmax
  +(Pos.y/(PanoramaH-1.0))*(Zmin+Zmax);
float φ = (Pos.x/(PanoramaW-1.0)+0.5)
  *2*π;
float X' = cos(φ);
float Y' = sin(φ);
float eqn = f*(b2-c2)/
((b2+c2)*Z-2*b*c*sqrt(X*X+Y*Y+Z*Z))
TexUV0.x = (X*eqn+cX) / TexW; TexUV0.y
= (Y*eqn+cY) / TexH;
```

ここで，Pos.x と Pos.y はそれぞれ出力画像（パノラマ画像）の水平方向と垂直方向の座標値で，PanoramaW と PanoramaH はそれぞれパノラマ画像の幅と高さである．また，b, c はミラーパラメータで，$b2 = b^2, c2 = c^2$ であり，(cX, cY) は全方位画像の中心座標である．TexW と TexH はそれぞれ，テクスチャの幅と高さで，ここでは輝度画像，つまり入力画像の幅と高さである．TexUV0.x はテクスチャ座標の垂直方向成分，TexUV0.y はテクスチャ座標の水平方向成分である．テクスチャ座標をテクスチャの幅や高さで割っているのは，テクスチャ座標が [0,1] の範囲に正規化されているからである．頂点シェーダではテクスチャ座標 TexUV0 を出力し，ピクセルシェーダではこの座標値を用いてテクスチャ（輝度画像）をサンプリングするだけである．

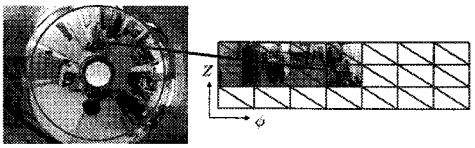

図9　テクスチャマッピング

なお，頂点シェーダで計算できるのは，頂点のテクスチャ座標だけで，頂点間のピクセルは頂点のデータから線形補間が行われる．式 (1),(2) から分かるように，パノラマ画像への変換は非線形であるためポリゴンの頂点にあたるピクセル以外は，近似的な変換になるが高速な変換が可能である．

3.3　Integral projection

Integral projection は式 (3) にあるように，画像の輝度値を垂直方向あるいは水平方向に加算するだけの処理である．この加算を実現する最も単純な方法は for ループを用いる方法である．最新の GPU はシェーダ内での for ループをサポートしており，実装できないことは無い．しかし，CPU ほど汎用性は無くループの最大回数は 256 回に制限されている．そのため，画像サイズが大きくなると対応できなくなる．そこで，1 ラインずつ加算していくことを考える．このとき，Z 軸方向のパノラマ画像に対する水力方向の projection の経過は図 10 のようになる．図 10 の上の図はパノラマ画像で，真ん中の図は上から半分まで projection が終了した図である．最終的に図 10 の下の図のようになり，この図の一番下の 1 ラインがパノラマ画像の projection 結果になる．

ピクセルシェーダのコードは以下のようになる．

```
float tmp1=tex2D(Input,TexUV0);
float tmp2=tex2D(Input,TexUV1);
return tmp1+tmp2;
```

tmp1 にこれまで projection した結果を読み込み，tmp2 に次に加算するピクセルを読み込む．そしてこの 2 つの値を加算して出力する．

この方法の場合，シェーダ内でループを使用しないため，画像サイズが大きくなっても対応することがで

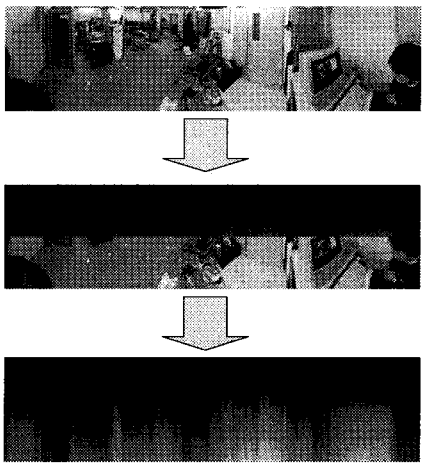

図 10　projection の経過

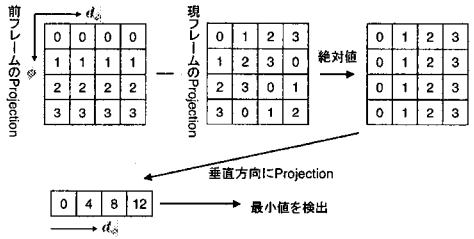

図 11　Projection 結果のマッチング

きる．

3.4　Projection 結果のマッチング

Projection 結果のマッチングの実装は図 11 のような考えで行う．前フレームの Projection 結果を列ベクトルとして並べたもの（図 11 左上）と，現フレームの Projection 結果を列ベクトルとして，1 列ごとに 1 要素ずつずらして並べたもの（図 11 上中央）を作成し，これら 2 つのものの差の絶対値をとる（図 11 右上）．さらにこれを垂直方向に加算する（図 11 左下）ことで式 (5) を計算する．現フレームの Projection 結果をこのように並べて考えることで，図 11 左下の結果から最小値を与える要素を検出することでマッチングを計算することができる．つまり図 11 左下の結果において，0 番目の要素が最小値なら前フレームと現フレームとのずれは 0 ピクセルで，1 番目の要素が最小値ならそのずれは 1 ピクセルというように計算することができる．GPU では，2 つの Projection の差の絶対値を計算し，その垂直方向への加算処理を行い，最後の最小値を与える要素の検出は CPU で行う．

2 つの Projection の差の絶対値を計算するピクセルシェーダの擬似コードは以下のようになる．

```
float prev=tex2D(PrevProjection,TexUV0);
float curr=tex2D(CurrProjection,TexUV1);
return abs(prev-curr);
```

ここで，PrevProjection は前フレームの Projection 結果を書き込んだテクスチャで CurrProjection は現フレームの Projection 結果を書き込んだテクスチャである．また，abs は絶対値を計算する関数でありこの関数の戻り値を書き込み用テクスチャへ出力する．GPU は全ピクセルに対して同じ計算をする処理が得意なため，Projection 結果全体をマッチングに利用しても高速な処理が可能である．この後の加算処理は Integral projection の計算と同じものを流用することができる．

最後に最小値を与える要素を検出するために，計算結果を GPU から CPU へ転送する必要がある．この処理は一般にコストが高く，GPU 処理による高速化のボトルネックになりやすい．そこで本研究では，6 パラメータの GPU での処理結果を 1 つのテクスチャに書き込むことで，GPU から CPU へのデータの転送処理を 1 度で済ませるようにした．

4　実験

提案手法を実装し実画像に適用して処理速度と Z 軸周りの回転成分の推定精度の評価を行った．本実験に使用したシステムの構成を表 1 に示す．入力画像の解像度は 640×480[pixels] で，入力のビデオレートは 30 fps である．パノラマ画像に変換するためのメッシュは，図 8 における m, n をともに 8[pixels] とした．

4.1　処理速度

提案手法の処理速度を，生成するパノラマ画像のサイズを変化させて測定した．パノラマ画像のサイズは表 2 に示す A～D の 4 つの組み合わせを用いた．ここで，X 軸方向と Y 軸方向のサイズが Z 軸方向と比較して小さいのは，2 節で述べたように 360 度の情報が得られないため，有効な部分のみを生成したからである．処理速度の測定結果を図 12 に示す．処理速度は 1 フレームの処理に要した時間を 500 フレーム計測した平均値である．入力画像のビデオレートは 30 fps であるため，パノラマ画像のサイズは C や D の組み合わ

表1　システム構成

OS	Windows XP SP2
CPU	Intel PentiumD 2.8GHz
RAM	2GB
GPU	GeForce 7800GTX × 2
VRAM	512MB
画像キャプチャボード	Photron FDM-PCI III
全方位センサ	TOSHIBA IK-M43H ＋双曲面ミラー

表2　パノラマ画像サイズの組み合わせ

組み合わせ	パノラマ画像のサイズ (pixels)		
	X 軸方向	Y 軸方向	Z 軸方向
A	112×256	112×256	512×256
B	168×256	168×256	768×256
C	224×256	224×256	1024×256
D	448×256	448×256	2048×256

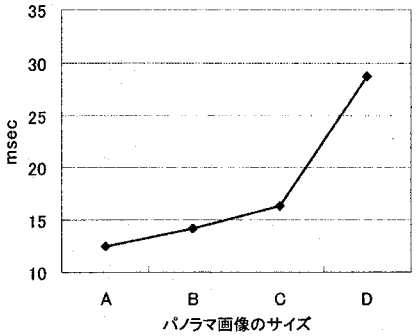

図12　処理速度

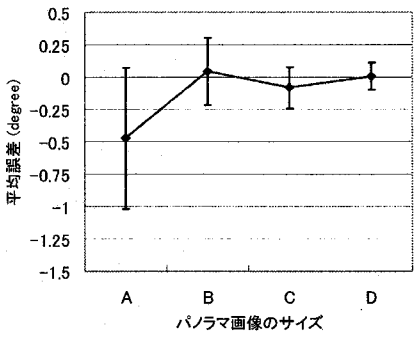

図13　画像サイズによる精度比較 (真値：3.0 degree)

せでも十分にビデオレートでの処理が可能である．ただし，この後さらに処理を追加するようなアプリケーションの場合はCの組み合わせ以下のサイズを用いるのが適当であると考えられる．

4.2　推定精度

上記の各パノラマ画像サイズの組み合わせA～Dに関して，Z 軸周りの回転のみが存在する環境で，その回転量の推定精度を検証した．回転ステージを利用し，Z 軸周りにフレーム間の回転角が3.0 degreeになるように，回転を与えて実験を行った．推定結果の平均誤差と標準偏差のグラフを図13に示す．グラフのプロットは点が各画像サイズの組み合わせでの平均誤差，縦の棒が標準偏差を表す．図13から分かるように，パノラマ画像のサイズが大きいほど推定精度が上がっている．これは，サイズを大きくするとパノラマ画像での角度分解能が高くなるからである．逆にAの組み合わせのようにサイズを小さくすると角度分解能が低くなり，推定精度が悪くなる．

実時間での動作が求められ，高精度な推定が必要な動画像の揺れ補正のようなアプリケーションにおいては，処理速度と推定精度の両方を考慮して，パノラマ画像のサイズはCの組み合わせが適当であると考える．さらに，Cの組み合わせにおいて Z 軸周りの回転のみの状況で，フレーム間の回転量の違いによる推定精度の変化を検証した．回転ステージを利用し，Z 軸周りにフレーム間の回転角に1.0, 10.0, 20.0, 30.0 degreeの4通りの角度を与えて実験を行った．推定結果の平均誤差と標準偏差のグラフを図14に示す．グラフのプロットは点が各画像サイズの組み合わせでの平均誤差，縦の棒が標準偏差を表す．Z 軸周りの回転のみの場合はフレーム間の回転量が変化しても推定値の平均誤差はほぼ変化しない．一方，推定値の標準誤差は回転量が大きくなるにしたがって増加しているのが分かる．

5　おわりに

本稿では，Integral projectionを用いた全方位視覚センサの回転量を推定する手法を提案した．全方位画像をミラーの焦点を中心とした3軸直交系の各軸方向の円筒状の画像（パノラマ画像）へ射影し，各パノラマ画像に対してIntegral projectionを適用した．これ

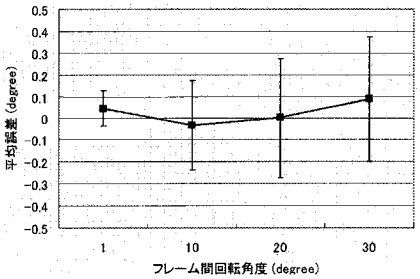

図 14　回転角度による精度比較 (画像サイズ：C)

により，画像の全領域を利用したマッチングを 1 次元で行うことができ，計算量を大幅に削減することができた．また，提案手法を GPU で実装し，実時間での処理を行うことができた．

実画像を用いた実験において Z 軸周りの回転に関する推定精度の評価を行い，フレーム間での回転量が微小な場合において，高精度で推定できることを確認した．今後，実アプリケーションに適用して提案手法の有用性を検討する．

参考文献

[1] 山澤一誠, 八木康史, 谷内田正彦: “移動ロボットのナビゲーションのための全方位視覚センサ”, 電子情報通信学会論文誌 D-II, Vol. J79-D-II, No. 5, pp. 698-707, May, 1996.

[2] T. Y. Tian, C. Tomasi, and D. J. Heeger, “Comparison of Approaches to Egomotion Computation”, Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 315-320, June, 1996

[3] A. J. Crawford, H. Denman, F. Kelly, F. Pitié and A. C. Kokaram: “Gradient Based Dominant Motion Estimation with Integral Projections for Real Time Video Stabilisation”, Proceedings of IEEE International Conference on Image Processing, Singapor, October 2004.

[4] J. H. Lee and J. B. Ra, “Block motion estimation based on selective integral projections”, IEEE Int. Conf. Image Processing, vol. I, pp. 689-693, 2002.

[5] J. Gluckman and S. K. Nayar, “Ego-Motion and Omnidirectional Cameras”, Proceedings of International Conference on Conputer Vision, pp999-1005, January, 1998.

[6] A. Makadia and K. Daniilidis, “Rotation Recovery from Spherical Image without Correspondences”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 28, No. 7, pp. 1170-1175, July, 2006.

[7] John D. Owens, et al, “A survey of general-purpose computation on graphics hardware”, Eurographics, State of the Art Reports, pp. 21-51, 2005.

[8] Robert Strzodka and Christoph Garbe, “Real-Time Motion Estimation and Visualization on Graphics Cards”, Proceedings IEEE Visualization, pp. 545-552, 2004.

[9] 鈴木裕一，山口泰，“GPU による畳み込み型顔検出器の高速化”, 画像の認識・理解シンポジウム (MIRU2006), No. OS3B-2, pp. 141-146, July, 2006.

[10] 松井良平, 長原一, 谷内田正彦: “GPU を用いた全方位画像処理の高速化”, 画像の認識・理解シンポジウム (MIRU2005), No. IS4-170, pp.1539-1546, July, 2005.