計測自動制御学会東北支部第295回研究集会(2015.6.26)
資料番号295-7

ステレオマッチングによる物体形状の取得

Acquisition of an Object Shape Using Stereo Matching

○吉田寛和*，釜谷博行*
○Hirokazu Yoshida*，Hiroyuki Kamaya*
*八戸工業高等専門学校
*National Institude of Technology, Hachinohe College

キーワード: ステレオカメラ(Stereo Camera)，三角測量(Triangulation)，
OpenCV(Open Source Computer Vision)

連絡先: 〒039-1192 八戸市田面木字上野平 16-1 八戸工業高等専門学校　産業システム工学科
釜谷博行，Tel.: 0178-27-7283，E-mail: kamaya-e@hachinohe-ct.ac.jp

1. はじめに

近年、3Dプリンタや車の自動ブレーキの普及により、三次元形状を取得する技術が注目されている。三次元形状を計測する手法として、２台のカメラにより構成されるステレオカメラが挙げられる。ステレオカメラに要求される性能として、画像の歪みが少ないこと、左右カメラの光軸のずれが小さいこと、正確に同期がとれていることなどがある。このため、ステレオカメラは高価になってしまう。

本研究では、一般の人が入手しやすい安価なステレオカメラを用いることで、三次元形状を計測するシステムについて検討する。

2. ステレオカメラの原理

人間は、物を見るとき２つの目で得られた視覚情報から三角測量の原理で物体の距離を判断する。ステレオカメラはこれと同様の原理で、物体の距離を計測することができる。ここでは、三角測量を用いた平行ステレオカメラの原理について説明する。Fig.1 にカメラと物体の関係図を示す。

レンズの中心を通り、レンズ面と直交するようにしたｚ軸を光軸と呼ぶ。光軸は画像面に垂直になるよう配置されるので、x-y 平面と画像平面が平行になる。このような座標系をカメラ座標系と呼ぶ。ステレオカメラでは、カメラを２台使用するのでカメラ座標も右と左の２つ存在する。２つのカメラ座標を平行に並べるものとすると、右カメラのレンズ中心を原点とした座標(P_{xr}, P_{yr}, P_{zr})は、左カメラのレンズ中心を原点とした座標

$(P_{x\ell}, P_{y\ell}, P_{z\ell})$とカメラ間隔 h を用いて

$$\begin{cases} P_{xr} = P_{x\ell} - h \\ P_{yr} = P_{y\ell} \\ P_{zr} = P_{z\ell} \end{cases} \quad (1)$$

と表すことができる。このとき、レンズを通して対象物 p をスクリーンに投影すると、実像となるので上下が逆転し、非常にわかりづらい。そこで、仮想的に画像面をレンズ中心の前に置くと、像が上下逆転せずに投影されるので、扱いやすくなる。実像と大きさを等しくするために、画像平面は焦点距離 f に置かれる。画像平面に投影された対象物 p の座標を左は(x_ℓ, y_ℓ)、右は(x_r, y_r)とすると、相似三角関係より

$$\begin{cases} x_\ell = z_\ell \frac{P_{x\ell}}{P_{z\ell}} = f\frac{P_{x\ell}}{P_{z\ell}} \\ y_\ell = z_\ell \frac{P_{y\ell}}{P_{z\ell}} = f\frac{P_{y\ell}}{P_{z\ell}} \end{cases} \quad (2)$$

$$\begin{cases} x_r = z_r \frac{P_{xr}}{P_{zr}} = f\frac{P_{xr}}{P_{zr}} \\ y_r = z_r \frac{P_{yr}}{P_{zr}} = f\frac{P_{yr}}{P_{zr}} \end{cases} \quad (3)$$

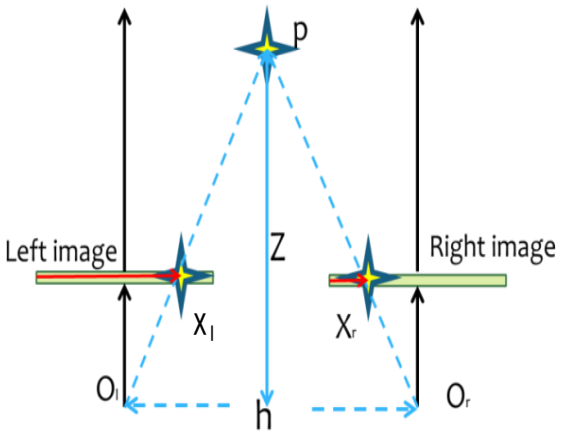
が得られる。ここで式(1)を式(3)に代入すると、

$$\begin{cases} x_r = f\frac{P_{x\ell}-h}{P_{z\ell}} = f\frac{P_{x\ell}}{P_{z\ell}} - f\frac{h}{P_{z\ell}} = x_\ell - f\frac{h}{P_{z\ell}} \\ y_r = f\frac{P_{y\ell}}{P_{z\ell}} = y_\ell \end{cases} \quad (4)$$

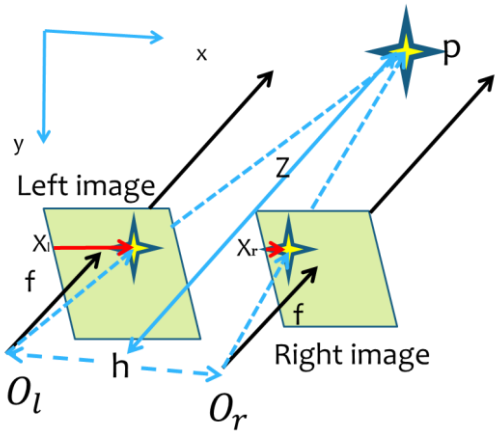
となり、式(4)より

$$P_{z_\ell} = \frac{fh}{x_\ell - x_r} \quad (5)$$

が得られ、対象物までの距離が求められる。分母 $x_\ell - x_r$ は視差を表している。視差とは、左右のカメラで見える同じ特徴点の画像平面の x 座標の差である。



(a)



(b)

Fig.1 The principle of stereo camera

3. 開発環境

3-1. システム構成

本研究を行うにあたって、使用したシムテム構成をFig.2に示す。本システムはパソコン1台とステレオカメラ1台から構成される。

使用したパソコンはSONY VAIO (Windows7(64bitOS), Intel® CORE™ i5-2430M, メモリ 4GB, クロック 2.4GHz) である。

Fig.2 System configuration

3-2.ハードウェア

ステレオカメラには、安価なOvrvision1[1]を使用した。このカメラの性能を以下に示す。

画素数：1280×480ピクセル
(640×480ピクセル×2)
最大フレームレート：60fps
インターフェース：USB2.0
カメラ間距離：50mm
大きさ：86×36
画角（対角）：135°

3-3.ソフトウェア

ソフトウェアの開発にはMicrosoft Visual C++ 2010 Expressを使用した。コンピュータービジョンライブラリには、OpenCV2.4.9を使用した。カメラキャリブレーションや色の識別、座標の取得にはOpenCVの関数を使用した[2]。ステレオカメラからの画像の取得には、Ovrvision SDK Version 2.15[1]を使用した。

4. 前処理

前処理としてキャリブレーションと画像の平行化について説明する。

4-1.キャリブレーション

ステレオマッチングを行うにあたって、Y軸上のずれやレンズによる特徴点の歪みがあると、マッチングに失敗する。また、式(5)より距離を計算するためには、カメラの焦点距離を求める必要がある。そこで、カメラキャリブレーションを行う。キャリブレーションとは、カメラの特性を表す内部パラメータと、外部の世界に基づいた座標系（ワールド座標系）とカメラ座標系の関係を表す外部パラメータを求める手法である。具体的なパラメータの内容をTable 1に示す[3]。

Table 1 Parameter

内部パラメータ	カメラ自体の特性 ・焦点距離　f ・レンズ歪み係数 k_1、k_2、p_1、p_2、k_3 ・画像中心（u_0, v_0）
外部パラメータ	カメラの位置と姿勢 ・回転　R ・平行移動　t

キャリブレーションには、ワールド座標の三次元空間中の物体を二次元画像にマッピングするモデルが必要である。本研究ではキャリブレーション用の物体として、Fig.3に示す６×９のチェスボード（白黒の正方形が交互に並んだパターン）を使用した。

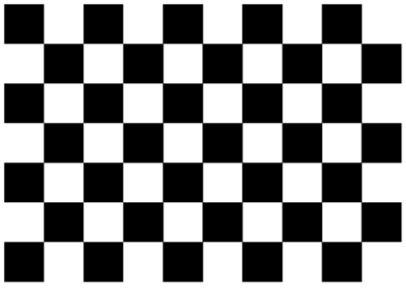


Fig.3 Chessboard

4-2. 画像補正

キャリブレーションより得られたパラメータを用いて、レンズによる画像の歪み補正と左右画像の平行化を行った。

Fig. 4に補正前の画像、Fig. 5に補正後の画像を示す。

Fig. 4 Before paralleled

Fig. 5 After paralleled

5. 実験内容

5-1. 準備

実験を行うにあたって、以下の準備を行った。

まず、OpenCVの関数を用いてステレオカメラのキャリブレーションを行い、左右のカメラのパラメータを取得した[4]。取得したパラメータより、レンズによる画像の歪み補正、左右画像の平行化を行った。キャリブレーションにより取得した内部パラメータの結果は、

左カメラのカメラ座標

$$\begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 309.8 & 0 & 319.9 \\ 0 & 313.4 & 276.7 \\ 0 & 0 & 1 \end{bmatrix}$$

右カメラのカメラ座標

$$\begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 312.0 & 0 & 309.6 \\ 0 & 312.2 & 259.1 \\ 0 & 0 & 1 \end{bmatrix}$$

平行化より得られた新しいカメラ座標

$$\begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 381.5 & 0 & 278.6 \\ 0 & 381.5 & 229.3 \\ 0 & 0 & 1 \end{bmatrix}$$

となった。

次に、対象物体をカメラの前に置き、カメラからの距離を変化させて画像を取得した。カメラからの距離は150㎜～450㎜を50㎜間隔で行った。対象物体にはあらかじめカラーマーカーを取り付けておき、画像上におけるマーカーの座標を取得することでカメラからマーカーまでの距離を計測する。対象物体（一辺100[㎜]の立方体）とマーカーをFig. 6に示す。

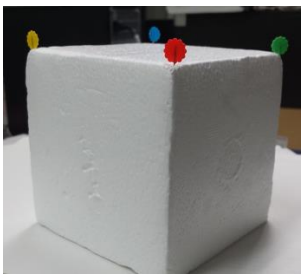


Fig. 6 Target object and marker

5-2. 人間による対応点の指定

初めに、左右画像の対応点を人間がそれぞれマウスで指定することで距離を計測する実験を行った。実験手順は以下の通りである。

1. 取得画像の読み込み

2. 画像の補正
3. 左画像における対応点の選択（マウスでのクリックにより指定）
4. 右画像における対応点の選択（マウスでのクリックにより指定）
5. 各対応点のx座標、y座標を表示
6. 手順5で得たx座標より式(5)を用いて距離を計算
7. 結果の表示

今回は、赤色と黄色のマーカーを使用し、対応点としてマーカーの左端を選択した。実験結果をFig. 7からFig. 10に示す。誤差率のFig. 8とFig. 10には、読み取り誤差を考慮して±１ピクセルのずれがあった場合についてエラーバーで示した。誤差率の平均は、赤色1. 6%、黄色1. 3%となった。

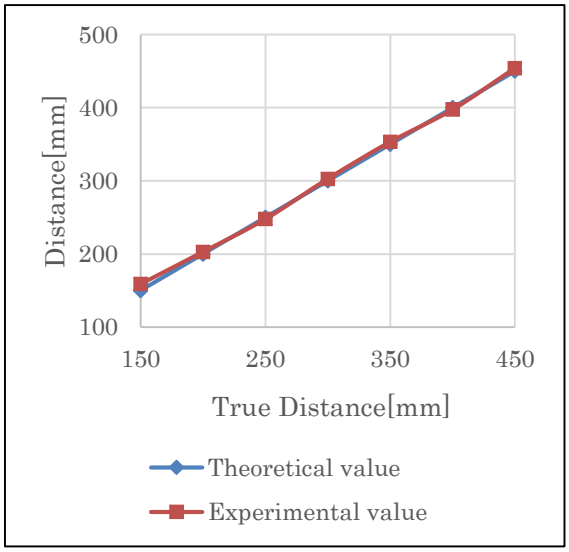


Fig. 7 Comparison result（red marker）

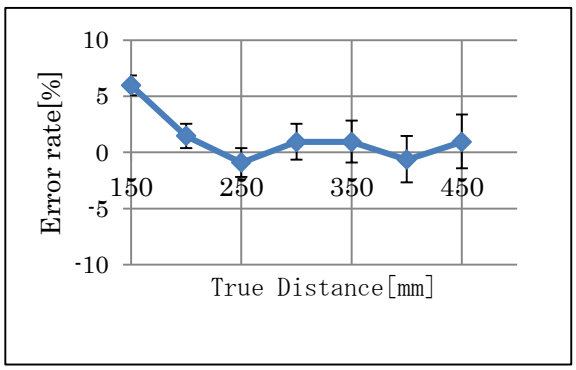


Fig. 8 Error rate(red marker)

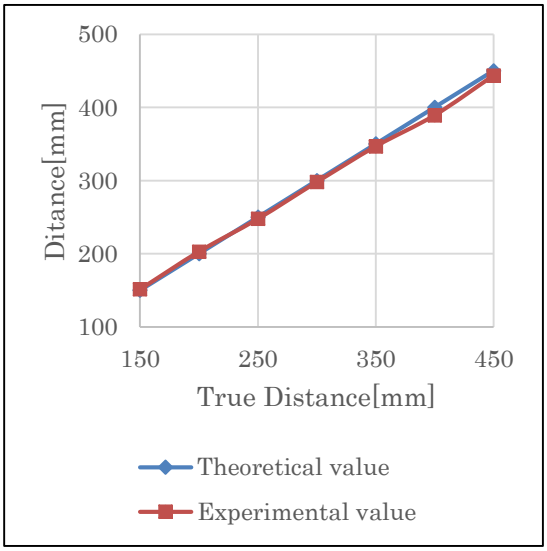


Fig. 9 Comparison result（yellow marker）

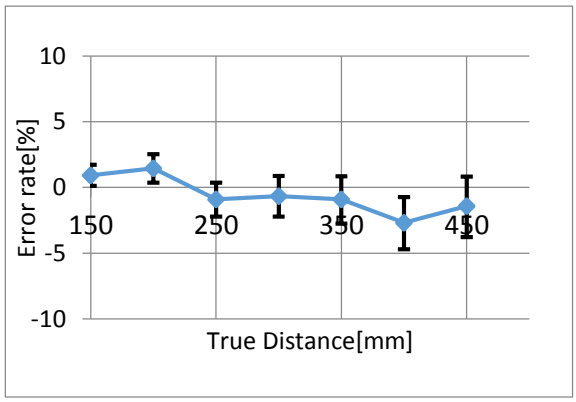


Fig. 10 Error rate（yellow marker）

5-3. 対応点探索の自動化

自動で対応点を探索し、対象物体までの距離を計測する実験を行った。

マーカーの色認識をすることで対応点を決定した。色認識の手法には色の減算処理を用いた。この処理は取得画像をRGBに分割し、次の式にあてはめることにより行った。左辺が色を減算した結果である。

$$\begin{cases} R' = R - G \\ B' = B - R \\ Y' = G - B \\ G' = G - R \end{cases} \quad (6)$$

(R : red, B:blue, G:green, Y:yellow)

この減算処理を行った画像を二値化することで色領域を抽出する。このときの閾値は100とした。

実験手順を以下に示す。

1. 取得画像の読み込み
2. 画像の補正
3. 色の減算処理
4. 画像の二値化（閾値：100）
5. 領域面積が最大の部分のみ選択
6. 左画像における対応点の探索
7. 右画像における対応点の探索
8. 各対応点のx座標、y座標を表示
9. 手順8で得たx座標より式(5)を用いて距離を計算
10. 結果の表示

実験結果をFig. 11、Fig. 12に示す。人間が対応点を指定する実験と同様、赤色マーカーと黄色マーカーを認識して対応点の探索を行った。赤色マーカーは正しく認識することができた。このときの誤差率の平均は2.2%となった。一方、黄色のマーカーは、減算処理より抽出された領域面積が小さすぎたため、正しく認識することができない場合もあった。

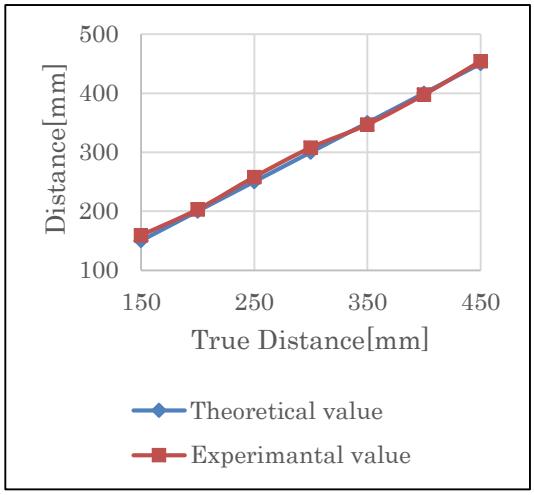


Fig. 11 Comparison result (red marker)

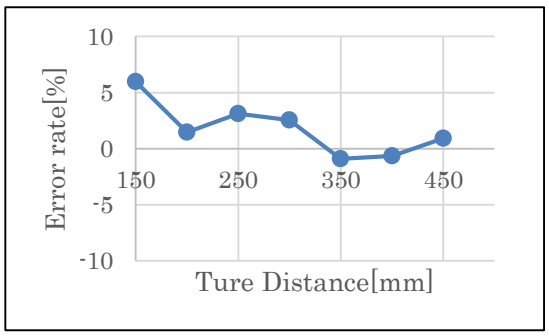


Fig. 12 Error rate (red marker)

6. リアルタイム形状取得

対応点探索の自動化により、リアルタイムでの対象物体の形状取得を行った。マーカーには、黄色、赤色、青色、緑色を用いた。それぞれの色の検出には式(6)を用いた。

この実験で得られたマーカー座標をTable 2に示す。

Table 2 Acquisition of an Object Shape

Color	X[mm]	Y[mm]	Z[mm]
Red	164.4	132.7	134.3
Yellow	75.7	134.9	140.3
Blue	155.6	180.6	238.4
Green	237.2	170.9	221.8

ユークリッド距離を計算したところ、Fig.13のようになり、精度があまり良くなかった。この原因として、取り付け時のマーカーの位置ずれ、緑色マーカーのｘ座標取得の失敗などが考えられる。なお、取得にかかった平均時間123.9［ms］となった。

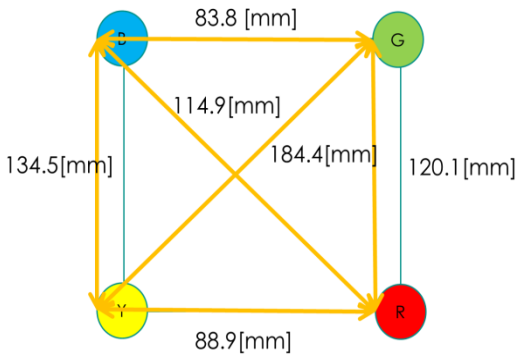


Fig.13 Measured object shape

7. まとめ

安価なステレオカメラを用いて、左右の画像でのマーカーの視差よりカメラから対象物体までの距離を求めた。人間により対応点を指定した場合は、誤差率が平均で1.6[%]以内となった。

対応点探索の自動化では、赤色マーカーについては、人間による対応点の指定と同程度の精度で取得することができた。しかし、黄色のマーカーについては、対象物までの距離が離れるにしたがってマーカーの領域面積が小さくなり、対応点を取得できない問題も生じた。

リアルタイムでの物体形状の取得では、マーカーの取り付け位置が悪いなどが原因で、あまり精度が良くなかった。

8. 今後の展望

今回、物体形状の取得を行うために4色のマーカーを用いて対応点を探索した。対応点として領域面積の左端を選択した。誤差率を小さくするためには、重心座標を計測すべきであると考えられる。また、複数色のマーカーを使用する方法では、複雑な形状を取得する際にマーカーの種類が増えてしまい、形状の取得が困難となることが予想される。そこで、マーカーの色を単色にし、同じY座標内で領域面積を比較することで対応点を探索する方法を検討している。リアルタイムでの形状取得では、座標の取得はできたが、処理速度が遅く、精度もあまり良くなかった。今後は、処理速度の高速化と精度の向上について検討していきたい。

参考文献

[1] しのびや.com, http://dev.ovrvision.com/doc_ja/ (2015年６月)

[2] OpenCV.jp, http://opencv.jp/ (2015年 6月)

[3] 出口光一郎、ロボットビジョンの基礎、コロナ社 (2000)

[4] Gray Bradski and Adrian Kaebler, 詳解 OpenCV -コンピュータビジョンライブラリを使った画像処理・認識 (2009)