機械情報工学科 演習
メディアインタフェース（2）
カメラと AR の基礎

担当：谷川智洋，鳴海拓志，中垣好之
TA: 木山亮，上田雅道

2011 年 11 月 25 日

1 演習の目的

本日の演習では，メディアインタフェースの代表的な実現手段として，カメラに基づくインタフェースについてさらに学ぶ．特に，カメラ画像から撮像対象の 3 次元情報を取得する方法（ピンホールカメラモデル，透視変換，座標変換など）を学び，最終的に，カメラによる 3 次元位置計測法について学ぶ．C 言語による実装を通じそれらへの理解を深める．

1.1 資料など

本日の演習の資料などは，

```
http://www.cyber.t.u-tokyo.ac.jp/~tani/class/mech_enshu/
```

においてある．

本日使用するソースファイルも同じ場所からダウンロードすること．前回の資料を参照したい場合も同じ場所を参照のこと．

1.2 出席・課題の確認について

出席の確認は，課題の確認によって行う．課題が終了したら，教員・TA を呼び，指示に従ってプログラムを実行して説明せよ．確認後，課題をメール提出せよ．演習時間終了後（16 時 15 分以降）は，課題が全て終了せずとも退室は自由である．

メールは，mireport@cyber.t.u-tokyo.ac.jp 宛て，件名は「メディアインタフェース演習 20111124 の課題メール：abcdefgh」とせよ．ただし abcdefgh は学籍番号（半角の数字列）に置き換えよ．適切な件名でないものは未提出扱いになる可能性があるので要注意．ただし，メールは週明け (11/28) までに提出のこと．

1.3 貸し出し物の返却について

課題が全て終了後，帰宅前に，貸し出している USB カメラ，Wii リモコン，Bluetooth アダプタを TA まだ返却せよ．不具合があったりした場合は，来年も使うために，不具合の内容を返却時に報告すること．

2　位置センサとしてのカメラ

近年画像を使ったインタフェースへ注目が集まっている．例えば，拡張現実感（Augumented Reality：AR）を代表とする現実空間と計算機空間を共棲させる技術などがある．AR ではカメラは 3 次元位置センサとして利用されており画像から 3 次元世界を理解する技術が重要となってくる．本資料では，AR などで使われている 3 次元位置センサとしてのカメラの処理の枠組みについて説明する．

２次元の画像そのものは本来，カメラを通じて３次元の世界を投影し得られたものなので，３次元の情報を２次元の画像面に投影するカメラモデルについて知っておく必要がある．よって本資料ではまずカメラモデルについて説明する．カメラモデルはいわば投影変換処理そのものであるが，複数のカメラの撮像結果を関係付けるための記述に必要な座標変換についても述べる．

3　カメラモデル

3.1　ピンホールカメラモデル：カメラの単純化

カメラとは，レンズ[*1]の集光性（光がレンズを通じて屈折する現象）を利用し，三次元空間中の点から発した光を集め，「画像面」に像として映す機器である（図 1）．画像面に相当する部分に撮像素子（CCD や CMOS）が並び，光の強度が各素子でサンプリングされる．レンズの中心を光学中心とよぶ．図 1 からも明らかなように，三次元空間中の一点と対応する画像中の結像点には幾何的関係がある．

しかしながら本来レンズの集光性は複雑なものであり，その幾何的関係の記述は簡単ではない．ステレオ視を簡単に実現するために，レンズ特性を極端に単純化し，対象と結像関係を理想化したモデルとして，ピンホールカメラを仮定してみよう（図 2）．ピンホールカメラとは，ティッシュ箱のような閉じた直方体の一面の一点をピンで穴を開け，その逆の面の箱の内側にスクリーンを置いたものである．ピンホールカメラモデルでは，ピンで開けた穴が光学中心となる．３次元空間中の一点から，様々な方向へ光が発するが，光学中心を通過した光は画像面の一点と交わる．交わった位置で像を結ぶ．図 2 のような実際のピンホールカメラでは，画像が逆転して生成されるので，図 3 のように画像面を配置しなおした方がわかりやすい．この配置では，光学中心から画像面を透かして対象を見たように結像していると仮定しているので，透視投影（perspective projection）と呼ばれる．２つのモデルは，画像面の像が逆さまになっている以外は同じである．

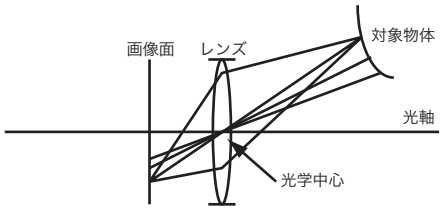

図 1　カメラの物理的なモデル

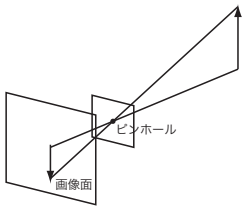

図 2　ピンホールカメラ

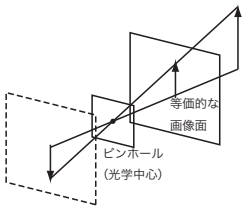

図 3　透視投影モデル

[*1] Wikipedia によれば，レンズマメ（和名：ヒラマメ）と初期の凸レンズの形状が類似していたためこのような名前になったということである．

3.2 透視変換の定式化

さて，ピンホールカメラを仮定し，三次元空間中の位置と結像についての幾何的関係がなんとなくわかったところで，この変換処理を数学的なモデルとして捉えてみよう．数学的，といっても「相似」関係を使って表現するだけである．

まず，光学中心に原点 $\boldsymbol{o}_c$ がある座標系を考える．光学中心を通る直線の一つを z 軸とする．これをカメラの光軸（optical axis）と呼ぶ．z 軸と直交する平面を画像平面とする．光学中心 $\boldsymbol{o}_c$ から画像面までの距離を（レンズの焦点距離とほぼ等しい）を f とする．つまり，画像面は $z = f$ である．z 軸と直交する x 軸，y 軸を，画像の座標系と平行もしくは直交するように設置する．つまり，画像の水平右向きの軸を x 軸，画像の鉛直下向きの軸を y 軸とする．このような座標系をカメラ座標系（camera coordinates）と呼ぶ（図 4）．

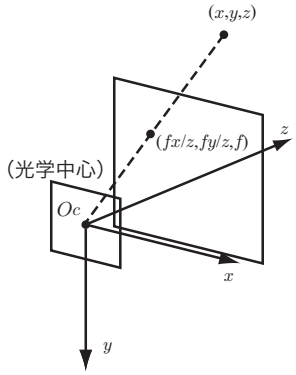

図 4 カメラの結像に関する座標系

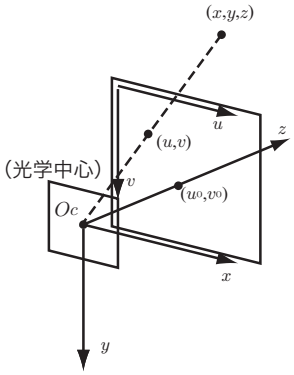

図 5 カメラと画像の座標系

この場合，この座標系で (x, y, z) にある空間の点に対応する像は，「相似」の関係を使うと $(fx/z, fy/z, f)$ にできる．このうち「画像」として有効な情報である画像面上の点 $(fx/z, fy/z)$ に着目すると，前述の透視変換は，

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \frac{f}{z} \begin{pmatrix} x \\ y \end{pmatrix} \tag{1}$$

になる．これにより，カメラに対する相対的な位置がわかれば，結像される位置もわかる．

3.3 カメラの内部パラメータ

3.3.1 透視変換の定式化

実際にはカメラ座標系での座標値 $(fx/z, fy/z, f)$ は取得できない．計算機上の画像 2 次元配列の座標値 (u, v) が取得される（ここでは，画像の水平方向右向きを u 軸，鉛直下向きを v 軸で表現し，画像の左上の点を原点とした（図 5））．理想的な透視変換処理と得られる画像情報とを結ぶためには，以下の 2 つの要素を考慮する必要がある．

- カメラの光軸すなわちカメラ座標の z 軸は，画像面のおおよそ中心を通るが，画像の座標系は左上を原点としているため，z 軸の通過位置は $(0, 0)$ ピクセルにはならない．

- 画像座標でのスケールとカメラ座標でのスケールが異なる．(fx/z) は物理的な長さを持ったものであるが，ピクセルは物理量がない．つまりピクセルと物理量の関係を知る必要がある．

以上を考慮し，透視変換を少し修正してみる．まず，前者の要素について次の仮定を置く．光軸は画像面上 (u_0, v_0) を通るとする．後者の要素については，ピクセル単位とカメラ座標系との単位の比（スケールファクタ）を k_u，k_v とおいてみる．すると，画像上の座標値とカメラの相対的な位置は次のようになる．

$$u = fk_u \frac{x}{z} + u_0 \tag{2}$$

$$v = fk_v \frac{y}{z} + v_0 \tag{3}$$

行列を使ってコンパクトに表現してみることを考える．その結果以下のようになる．

$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} fk_u & 0 & u_0 \\ 0 & fk_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{4}$$

$s = z$ はスケールファクタである．s を導入するメリットは，除算が省かれることである．３次元位置情報と２次元画像情報との関係を記述するだけなら，レンズの焦点距離 f，ピクセルのスケールファクタ k_u, k_v それぞれを調べるよりも，これらをまとめて表現した方が都合が良い．ここで $\alpha = fk_u$，$\beta = fk_v$ とし，

$$\boldsymbol{A} = \begin{pmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5}$$

とおくと，カメラに対する相対的な３次元位置 $\boldsymbol{x} = (x, y, z)^\mathsf{T} \in \mathbb{R}^3$ と二次元画像上の点 $\boldsymbol{m} = (u, v)^\mathsf{T} \in \mathbb{R}^2$ には，

$$s\tilde{\boldsymbol{m}} = \boldsymbol{A}\boldsymbol{x} \tag{6}$$

が成り立つ．ただし，$\tilde{\boldsymbol{m}}$ は $\boldsymbol{m}$ の斉次座標とよばれ，$\tilde{\boldsymbol{m}} = \left(\boldsymbol{m}^\mathsf{T}, 1\right)^\mathsf{T} \in \mathbb{R}^3$ である．カメラ固有のレンズ焦点距離，ピクセルのスケールファクタ，光学中心位置に依存した 4 個の独立したパラメータからなる行列 $\boldsymbol{A} \in \mathbb{R}^{3\times3}$ をカメラの内部パラメータ行列（camera intrinsic matrix）と呼ぶ．本資料では，特に断りがない限りベクトルは列ベクトルであり，d 次元のベクトル $\boldsymbol{x} \in \mathbb{R}^d$ は d 行 1 列の行列と等価であるとする．また，T で書かれる演算子は，転置をあらわす．具体的には $\boldsymbol{x}^\mathsf{T}$ は 1 行 d 列の行列となる．

なお，実際の画像処理の研究においては，カメラ座標系における x 軸と u 軸，y 軸と v 軸が平行でないことや，光軸（z 軸）と画像面が直交しないことなどを考慮した 5 個のパラメータからなる行列を考えることがある．また，ピンホールカメラと実際のレンズの幾何的特性の違いをレンズ歪みとしてモデル化することもある．より深く学びたいと思うものは，コンピュータビジョンの教科書（例えば [3] など）を読んでみると良い．

3.3.2 カメラの内部パラメータの物理的解釈

カメラの内部パラメータ行列からわかるカメラの物理的な解釈を述べる．まず，カメラの画角と画像サイズからおおよその内部パラメータ行列が計算できる．具体的には，配布した USB カメラ（Elecom UCAM-DLQ30H）のつけられたレンズの画角はおよそ 38 ° [*2]である（たとえば $z_0 = 1000$mm では水平方向に撮像可能な幅 Δx_d は 680mm）．ここで USB カメラの画像サイズが VGA であることを利用すると $\alpha\Delta x_d/z_0 = 640$

[*2] 40 ° 程度の画角のレンズを標準レンズとよぶ

が成り立つことがわかるので $\alpha =$?????? になる．画角 θ と CMOS イメージのサイズ D とスクリーン光学中心間の距離 f には，次の関係がある（図 7）．

$$\tan\left(\frac{\theta}{2}\right) = \frac{D}{2f} \tag{7}$$

ここで UCAM で使われている CMOS センサのチップのサイズが仮に $D = 3.6$mm（垂直方向は 2.7mm）とすると $f =$????????mm という結果を得る．また，$D = 3.6$mm に対して CMOS 素子が 640 個並んでいるので CMOS1 ピクセルあたりの水平方向の幅は 5.6μm である．k_u は α，f から $k_u =$????????? ピクセル/mm として求められる．CMOS （CCD）センサのスクリーンサイズがわかれば，レンズとスクリーンの距離が計算できる．

レンズの後方焦点距離（レンズに平行に入ってきた光が一点に集まる点の最後方のレンズ中心からの距離）f_0 と CMOS 撮像面とレンズの光学中心の距離 f は，レンズの前面にあるもの距離 z にある点にピントを合わせると（UCAM ではレンズの前面にあるつまみを回す），ガウス光学の式より

$$\frac{1}{z} + \frac{1}{f} = \frac{1}{f_0} \tag{8}$$

が成り立つため（図 6）後方焦点距離は

$$f_0 = \frac{zf}{z+f} = \frac{f}{1+f/z} \tag{9}$$

と書ける．撮影する物体の距離は最低でも 300mm 程度以上離れているため，$f/z \approx 0$ が成り立つ．よって $f_0 \approx f =$????????mm となる．撮像対象の物体の距離に応じてピントの合う撮像素子とレンズ間距離は変わり，そして画角もわずかにかわることになるが，f の値の変化は 0.3mm 程度といえる．UCAM ではつまみを回し f を変更することで，被写体のピントを調整している．一部の教科書ではレンズの光学中心と撮像面の距離を焦点と呼んでいるものもあるが，近似的に成り立つのであって厳密には異なるので注意．UCAM の場合 D，f_0 が固定であり f（つまり $\boldsymbol{A}$），画角 θ がごく僅かであるが可変ということになる．f_0 が固定であるレンズを単焦点レンズと呼ぶ．

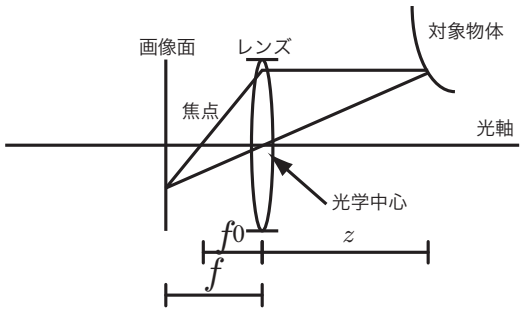

図 6　後方焦点距離と撮像面の関係

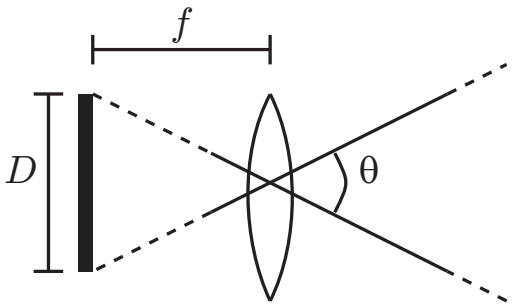

図 7　画角とレンズの光学中心と撮像面との関係

◇ **課題** 1 ◇

1. 上記の説明部分を良く読み，3.3.2 節「 カメラの内部パラメータの物理的解釈」について下線部 3 箇所（α, f, k_u）を埋めよ．
2. UCAM のレンズ近辺のつまみを回しピントの合う距離とレンズの位置について考察せよ．
3. カメラ内部パラメータとして $\boldsymbol{A}$ に加え「レンズ歪み」がよく使われる．「レンズ歪み」とは何か考えてみよ．

4　座標変換の基礎（復習事項）

カメラの透視変換の入力 $\boldsymbol{x}$ というのはカメラに対する相対的な位置である．ステレオ視とは複数のカメラの利用を前提としているため，カメラごとに座標系を定義する必要がある．よって，カメラ間の相対的な位置・姿勢を記述するフレームワーク，「座標変換」が必要である．この処理は運動学・動力学計算・コンピュータグラフィックスに必須の項目であるので，よく理解しておくこと．

ここでいう座標変換とは，座標軸を回転させること，原点を移動させること，それだけである．座標変換の中身の難しさよりも変数の表現方法で混乱を招くことが多いので，ここでは Craig 著のロボティクスの古典的な教科書 [1]（日本語訳もある）を参考に，曖昧さを避けるための記法を導入し説明する．３次元空間中の点 $\boldsymbol{x}$ を座標系 a で表現するとき ${}^a\boldsymbol{x}$ とし，別の座標系 b で表現するなら ${}^b\boldsymbol{x}$ とする．このとき 2 つの表現方法は次の関係を満たす．

$$ {}^a\boldsymbol{x} = {}^a_b\boldsymbol{R}{}^b\boldsymbol{x} + {}^a\boldsymbol{o}_b \tag{10} $$

ここで ${}^a_b\boldsymbol{R}$ は座標系 a に対する座標系 b の姿勢行列であり，座標系 a の原点に対する座標系 b の原点の相対的な位置を座標系 a で表現したものを ${}^a\boldsymbol{o}_b$ とする．姿勢行列の列成分に着目し

$$ {}^a_b\boldsymbol{R} = ({}^a\boldsymbol{r}_{b,x}, {}^a\boldsymbol{r}_{b,y}, {}^a\boldsymbol{r}_{b,z}) \tag{11} $$

として分解すると，それぞれのベクトル ${}^a\boldsymbol{r}_{b,x}$，${}^a\boldsymbol{r}_{b,y}$，${}^a\boldsymbol{r}_{b,z}$ は座標系 b の x, y, z 軸の単位ベクトルを座標系 a で表現したものとなっている．姿勢行列が直交行列になることは重要な性質である．直交行列とは，

$$ ({}^a_b\boldsymbol{R})^{-1} = ({}^a_b\boldsymbol{R})^{\mathsf{T}} \tag{12} $$

を満たすものであり，換言すれば

$$ {}^a_b\boldsymbol{R}({}^a_b\boldsymbol{R})^{\mathsf{T}} = \boldsymbol{I}, ({}^a_b\boldsymbol{R})^{\mathsf{T}}{}^a_b\boldsymbol{R} = \boldsymbol{I} \tag{13} $$

を満たすものである．これにより，

$$ ({}^a_b\boldsymbol{R})^{-1} = {}^b_a\boldsymbol{R} \tag{14} $$

となることもわかる．また，新たな座標系 c を導入し，c と b の関係が記述できる場合，

$$ {}^c_a\boldsymbol{R} = {}^c_b\boldsymbol{R}{}^b_a\boldsymbol{R} \tag{15} $$

が成り立つ．Craig の記法においては，座標変換に伴う行列の積算処理では，演算の左側の変数の座標系添え字の下側の変数と右側の変数の座標系添え字上側の変数が常に一致する．この場合，中間の座標系 b が積

算によって「消された」ことがポイントである．さらに，並進情報 ${}^a\boldsymbol{o}_b$ と姿勢行列をまとめて表現した行列 ${}^a_b\boldsymbol{T} \in \mathbb{R}^{4\times4}$ を

$$ {}^a_b\boldsymbol{T} = \begin{pmatrix} {}^a_b\boldsymbol{R} & {}^a\boldsymbol{o}_b \\ \boldsymbol{0} & 1 \end{pmatrix} \tag{16} $$

として定義すると，座標変換は次のような簡潔な線形演算として表現できる．

$$ {}^a\tilde{\boldsymbol{x}} = {}^a_b\boldsymbol{T}{}^b\tilde{\boldsymbol{x}} \tag{17} $$

ここで ${}^a\tilde{\boldsymbol{x}}$ は ${}^a\tilde{\boldsymbol{x}} = \left({}^a\boldsymbol{x}^\mathsf{T}, 1\right)^\mathsf{T}$ であり，${}^a\boldsymbol{x}$ の斉次ベクトルと呼ばれる．これを座標変換行列と呼ぶ．これは姿勢行列のときと同じような注目すべき性質がある．

$$ \left({}^a_b\boldsymbol{T}\right)^{-1} = {}^b_a\boldsymbol{T} \tag{18} $$

$$ {}^a_c\boldsymbol{T} = {}^a_b\boldsymbol{T}{}^b_c\boldsymbol{T} \tag{19} $$

この座標変換行列はコンピュータグラフィックスプログラミングにおいて，ある種の演算子として重要な役割を担う．

5　単眼カメラによる位置センシング

2 つ以上のカメラで得た画像間で適切な対応点が得られれば，また，カメラの内部外部パラメータが得られていれば，対象物体の 3 次元位置が計算できる．一方，ある基準となる地図上の点を計測し，カメラの位置を把握する，位置センシングについて考えてみる．

5.1　透視投影変換

座標変換の話題から，3 次元空間中の点の投影という元々の問題に戻ろう．カメラの内部パラメータの説明に用いた，カメラの光学中心を原点とする座標系は一般には利用しにくい．そもそも，この座標系の原点がカメラの光学中心であるため，測る術はない．よって，別の基準となる座標系を用いる必要がある．逆に，そのような座標系を基準として 3 次元空間を表現し，それをベースに画像上の座標値への変換を扱うこととすれば，複数のカメラがあったとしても統一的に 3 次元世界を表現できることになる．そのような基準となる座標系を便宜的に w とする．その場合，${}^w\boldsymbol{x}$ から $\boldsymbol{m}$ への変換は，座標変換の公式とカメラの内部パラメータ表現から次のようになる．

$$ s\tilde{\boldsymbol{m}} = \boldsymbol{A}\left({}^c_w\boldsymbol{R}{}^w\boldsymbol{x} + {}^c\boldsymbol{o}_w\right) \tag{20} $$

この式は，ステレオ視において非常に重要な役割を果たす．この式をさらに簡単に表現するために，

$$ {}^c_w\boldsymbol{H} = \boldsymbol{A}\begin{pmatrix} {}^c_w\boldsymbol{R} & {}^c\boldsymbol{o}_w \end{pmatrix} \in \mathbb{R}^{3\times4} \tag{21} $$

と新たに変数を導入すると

$$ s\tilde{\boldsymbol{m}} = {}^c_w\boldsymbol{H}{}^w\tilde{\boldsymbol{x}} \tag{22} $$

という関係が成り立つ．$\boldsymbol{H}$ は投影行列 (projection matrix) と呼ばれる．これは，カメラの内部パラメータ行列と位置・姿勢によって決まる．コンピュータビジョンでは，内部パラメータ行列と区別するため，位置・姿勢に関する情報をカメラの外部パラメータ (extrinsic parameter) とよぶ．

5.2　視線ベクトル・スケールファクタ

式 20 をよくみると，投影される 3 次元空間中の点 $\boldsymbol{x}$ は

$$ {}^{w}\boldsymbol{x} = s\,{}^{w}\boldsymbol{q} - {}^{w}\boldsymbol{w} \tag{23} $$

と表現できる．ただし，

$$ {}^{w}\boldsymbol{q} = \left(\boldsymbol{A}\,{}^{c}_{w}\boldsymbol{R}\right)^{-1}\tilde{\boldsymbol{m}},\ {}^{w}\boldsymbol{w} = {}^{c}_{w}\boldsymbol{R}^{-1}\,{}^{c}\boldsymbol{o}_{w} \tag{24} $$

としている．これは，スケールファクタ s を媒介変数とする直線の方程式とみなせる．ここで $-{}^{w}\boldsymbol{w}$ が，w 座標系におけるカメラの光学中心の位置であることに注意すると，この直線は「視線」ベクトルと呼べる．

5.3　投影行列の推定

3 次元空間中の一点 ${}^{w}\boldsymbol{x}_i$ と投影点 $\boldsymbol{m}_i$ のペアが最低 5 点以上あれば w 座標系におけるカメラの位置・姿勢は一意に求まることを示す．式 22 を参考に

$$ \boldsymbol{h}_1^{\mathsf{T}}\,{}^{w}\boldsymbol{x}_i + p_1 - u_i(\boldsymbol{h}_3^{\mathsf{T}}\,{}^{w}\boldsymbol{x}_i + p_3) = 0 \tag{25} $$

$$ \boldsymbol{h}_2^{\mathsf{T}}\,{}^{w}\boldsymbol{x}_i + p_2 - v_i(\boldsymbol{h}_3^{\mathsf{T}}\,{}^{w}\boldsymbol{x}_i + p_3) = 0 \tag{26} $$

のように分解すると

$$ \begin{pmatrix} {}^{w}\boldsymbol{x}_i^{\mathsf{T}} & 1 & \boldsymbol{0}^{\mathsf{T}} & 0 & -u_i\,{}^{w}\boldsymbol{x}_i^{\mathsf{T}} & -u_i \\ \boldsymbol{0}^{\mathsf{T}} & 0 & {}^{w}\boldsymbol{x}_i^{\mathsf{T}} & 1 & -v_i\,{}^{w}\boldsymbol{x}_i^{\mathsf{T}} & -v_i \end{pmatrix} \begin{pmatrix} \boldsymbol{h}_1 \\ p_1 \\ \boldsymbol{h}_2 \\ p_2 \\ \boldsymbol{h}_3 \\ p_3 \end{pmatrix} = \boldsymbol{0} \tag{27} $$

が成り立つ．これは 12 変数の未知数を持つ $\boldsymbol{H}$ を解くために，$\boldsymbol{x}_i, \boldsymbol{m}_i$ から 2 つの連立方程式を得たことを意味する．よって，6 点以上の点があれば，$\boldsymbol{H}$ が計算できる．この計算で求めた $\boldsymbol{H}$ はスケールに関して不定性を持つため（$\boldsymbol{H} = \boldsymbol{0}$ というのも解になってしまうという意味），例えば $|\boldsymbol{h}| = 1$ を制約条件として解く必要がある．実際に 7 点以上の点のペアが構成されるときは，ステレオによる 3 次元位置推定を求めた手順と同様，特異値分解法により算出する．

5.4　位置・姿勢推定

事前に内部パラメータ $\boldsymbol{A}$ が計算できていれば上の方程式で推定した解 $\hat{\boldsymbol{H}}$ から ${}^{c}_{w}\boldsymbol{R}$, ${}^{c}\boldsymbol{o}_{w}$ が計算できる．具体的には

$$ \begin{pmatrix} {}^{c}_{w}\boldsymbol{R} & {}^{c}\boldsymbol{o}_{w} \end{pmatrix} \propto \boldsymbol{A}^{-1}\hat{\boldsymbol{H}} \tag{28} $$

であり，$\det \boldsymbol{R} = 1$ であることから，$\hat{\boldsymbol{H}}$ に関するスケール不定性を解消できる．その後，ある基準座標系の原点に対するカメラの位置は ${}^{c}_{w}\boldsymbol{R}$, ${}^{c}\boldsymbol{o}_{w}$ から，

$$ {}^{w}\boldsymbol{o}_{c} = -{}^{w}_{c}\boldsymbol{R}\,{}^{c}\boldsymbol{o}_{w} \tag{29} $$

$$ {}^{w}_{c}\boldsymbol{R} = {}^{c}_{w}\boldsymbol{R}^{\mathsf{T}} \tag{30} $$

として計算できることになる．以上によりカメラを位置センサとして利用する手順が整った．

5.5 ホモグラフィ

ここで，全ての 3 次元空間中の点がある平面に属し，さらに，$z = 0$ であるとすると，

$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \boldsymbol{A} \begin{pmatrix} {}^{c}_{w}\boldsymbol{r}_1 & {}^{c}_{w}\boldsymbol{r}_2 & {}^{c}\boldsymbol{o}_w \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{31}$$

が成り立つ．ここで，

$$\tilde{\boldsymbol{H}} = \boldsymbol{A} \begin{pmatrix} {}^{c}_{w}\boldsymbol{r}_1 & {}^{c}_{w}\boldsymbol{r}_2 & {}^{c}\boldsymbol{o}_w \end{pmatrix} \tag{32}$$

という 3×3 の行列をホモグラフィ（Homography）と呼ぶ．ホモグラフィの推定には，最低 4 点の計測点があれば推定できるため，後述するカメラのキャリブレーションなどではホモグラフィを用いることが一般的である．ホモグラフィから投影行列が一意に復元できるためである．

6 OpenCV による線形代数演算

本節は OpenCV の API リファレンスマニュアルに従ってまとめたものである．あくまで参考程度に，プログラミングするときには別途リファレンスマニュアルをよく読むこと．API リファレンスマニュアルは，
/usr/local/share/opencv/doc/index.htm
http://opencv.jp/reference_manual
にある．線形代数の計算処理は，一般には，Fortran もしくは C 言語により実装された BLAS，LAPACK を用いることが多い．OpenCV の線形代数の関数はその一部で BLAS もしくは LAPACK を利用している．

6.1 行列の作成

OpenCV では，CvMat 型構造体を使って線形代数の計算を行うのが便利である．具体的には $\boldsymbol{X} \in \mathbb{R}^{5\times4}$ を示すデータを作成しようとした場合，

```
CvMat* X = cvCreateMat( 5, 4, CV_64F );
```

となる．3 つ目の引数は行列の各要素に入るデータの型を指定している．詳しくは API リファレンスマニュアルを読むこと．2 行 3 列目の要素に 4.5 という数字を代入する場合，

```
cvmSet( X, 1, 2, 4.5 );
```

と書く．一方で，4 行 3 列目の要素の値を知ろうとする場合，

```
int x_4_3 = cvmGet( X, 3, 2 );
```

と書く．インデックス演算子は C 言語の慣習に従う（1 行 1 列目は 0，0 と読み替える）ので注意が必要である．$\boldsymbol{X}$ が不要になったときには

```
cvReleaseMat( &X );
```

として，変数のメモリを開放する．

6.2 行列の乗算

前節で説明した CvMat 型構造体を用いて計算を行う．行列 $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{B} \in \mathbb{R}^{n \times k}$ の積算として，その結果を $\boldsymbol{C} \in \mathbb{R}^{m \times k}$ に書き込む計算，すなわち $\boldsymbol{C} = \boldsymbol{AB}$ を行うには，

```
CvMat* A;
CvMat* B;
CvMat* C;
//メモリの割り当てや行列データを設定
...
cvMatMul( A, B, C );
```

と書けばよい．

6.3 逆行列計算

OpenCV では，正方行列の逆行列は

```
CvMat* B;
CvMat* invB;
// setting some values and allocating memory...
...
cvInvert( B, invB, CV_LU );
```

として取得可能である．

6.4 擬似逆行列計算

一般に，式**??**で定義される擬似逆行列は，特異値分解（singular value decomposition）に基づき算出する．OpenCV では特異値分解に関する実装も用意されているが，直接計算を行ってくれる便利な関数がある．

具体的には $\boldsymbol{B} \in \mathbb{R}^{m \times n}$ に対する擬似逆行列 $\boldsymbol{B}^{\dagger} \in \mathbb{R}^{n \times m}$ は，

```
CvMat* B;
CvMat* pseudoInvB;
cvPseudoInv( B, pseudoInvB );
```

で算出可能である．なお，B は $\boldsymbol{B}$ に相当するものであり，pseudoInvB は $\boldsymbol{B}^{\dagger}$ に相当する．

なお，$\boldsymbol{Bx} = \boldsymbol{b}$ を満たす $\boldsymbol{x}$ を計したい場合，

```
cvSolve( B, b, x, CV_LU );
```

としても計算できる．いずれの引数も CvMat 型構造体へのポインタである．

改めてになるが，本演習で本資料を参考にする場合，必ず並行して API リファレンスマニュアルを読んでから実装すること．

7 カメラキャリブレーション

7.1 カメラキャリブレーションとは

■なぜキャリブレーションが必要か 本資料では，ステレオ視の原理の説明のため，カメラの内部パラメータ行列 $\boldsymbol{A}$ およびカメラと基準となる座標系の位置・姿勢の関係（${}^{c}_{w}\boldsymbol{R}, {}^{c}\boldsymbol{o}_{w}$）を既知と仮定している．しかしながら，実際には，このようなパラメータはカメラにより異なり，また，設置方法，座標系の設定方法によっても変わってしまう．よって，実際にステレオ視を行うためには，カメラのパラメータ情報を事前に獲得するプロセスが必要である．これをカメラのキャリブレーション（camera calibration）と呼ぶ．聞きなれない言葉かもしれないが，日本語ではカメラ較正（もしくは校正）と呼ばれている．カメラキャリブレーションはいわば，後に行うステレオ視のための下準備である．

■キャリブレーションがやること ステレオ視は画像の点 $\boldsymbol{m}, \boldsymbol{m}'$ とカメラパラメータ ${}^{c}_{w}\boldsymbol{H}, {}^{c'}_{w}\boldsymbol{H}$ から 3 次元位置 ${}^{w}\boldsymbol{x}$ を推定する問題であったが，${}^{w}\boldsymbol{x}$ と $\boldsymbol{m}$ の関係から，${}^{c}_{w}\boldsymbol{H}$ （さらには ${}^{c}_{w}\boldsymbol{H}$ から $\boldsymbol{A}$ と ${}^{c}_{w}\boldsymbol{R}$ と ${}^{c}\boldsymbol{o}_{w}$）を推定することがカメラキャリブレーションの役割である．つまり，カメラのパラメータ情報を獲得するキャリブレーションには，既知の 3 次元の位置情報と画像上の点の対応が与えられている必要がある．

ステレオ視においては，3 次元位置を与えず画像の 2 つのカメラ同士の対応点に基づきキャリブレーションをする方法もある．これを弱いキャリブレーション（weak calibration）と呼ぶ．基礎行列 $\boldsymbol{F}$ を求める問題がこれに相当する．

■キャリブレーションに王道なし カメラのキャリブレーションはその原理的には至ってシンプルなものが多いが，精度上の理由で確立された方法というのがないというのが実情である．それゆえに多くの方法が提案されている．それぞれメリットデメリットがある．手軽さをとるか，キャリブレーションの精度を追求するか，目的により最適なアルゴリズムは異なる．

■いろいろなキャリブレーション方法 直方体表面に印刷したパターンを撮影してキャリブレーションするもの，平面上のパターンを表示しキャリブレーションするもの，長さが既知の棒をカメラの前で自由に振り回しキャリブレーションを行うもの，など，様々な方法が提案されている．

7.2 Zhang のキャリブレーション

本資料では，平面ボードベースのキャリブレーションの一つである Zhang の方法 [2] を紹介する．この方法はキャリブレーションに必要な操作が手軽な一方，推定精度が高いといわれている．OpenCV でも実装されている，広く知られた計算法である．本演習でもステレオ視の下準備としてのキャリブレーションにこのアルゴリズムを用いる．Zhengyou Zhang 氏は Microsoft Research の Principal Researcher である．
http://research.microsoft.com/en-us/um/people/zhang/

7.2.1 Zhang の手法による内部・外部パラメータの推定法

以下，簡単ではあるが Zhang の方法による内部・外部パラメータの推定法を記す．ステレオ視では

$$s\tilde{\boldsymbol{m}} = {}^{c}_{w}\boldsymbol{H}\,{}^{w}\tilde{\boldsymbol{x}} \tag{33}$$

が成り立つが，${}^c_w\boldsymbol{H} = \begin{pmatrix} \tilde{\boldsymbol{h}}_1 & \tilde{\boldsymbol{h}}_2 & \tilde{\boldsymbol{h}}_3 & \tilde{\boldsymbol{h}}_4 \end{pmatrix}$ とおけば，${}^c_w\boldsymbol{H} = \boldsymbol{A}\,({}^c_w\boldsymbol{R}, {}^c\boldsymbol{o}_w)$ なので，

$$\boldsymbol{A}^{-1} \begin{pmatrix} \tilde{\boldsymbol{h}}_1 & \tilde{\boldsymbol{h}}_2 & \tilde{\boldsymbol{h}}_4 \end{pmatrix} \propto ({}^c\boldsymbol{r}_{w,x}, {}^c\boldsymbol{r}_{w,y}, {}^c\boldsymbol{o}_w) \tag{34}$$

であり，${}^c\boldsymbol{r}_{w,x}$ と ${}^c\boldsymbol{r}_{w,y}$ が直交の関係にあるため，

$$\tilde{\boldsymbol{h}}_1^\mathsf{T}\boldsymbol{A}^{-\mathsf{T}}\boldsymbol{A}^{-1}\tilde{\boldsymbol{h}}_1 - \tilde{\boldsymbol{h}}_2^\mathsf{T}\boldsymbol{A}^{-\mathsf{T}}\boldsymbol{A}^{-1}\tilde{\boldsymbol{h}}_2 = 0 \tag{35}$$

$$\tilde{\boldsymbol{h}}_1^\mathsf{T}\boldsymbol{A}^{-\mathsf{T}}\boldsymbol{A}^{-1}\tilde{\boldsymbol{h}}_2 = 0 \tag{36}$$

が成り立つ．$\boldsymbol{G} = \boldsymbol{A}^{-\mathsf{T}}\boldsymbol{A}^{-1}$ として，

$$\boldsymbol{g} = (\boldsymbol{G}_{1,1}, \boldsymbol{G}_{1,2}, \boldsymbol{G}_{2,2}, \boldsymbol{G}_{1,3}, \boldsymbol{G}_{2,3}, \boldsymbol{G}_{3,3})^\mathsf{T} \tag{37}$$

$$\boldsymbol{k}_{ij} = \left(\tilde{\boldsymbol{h}}_{i,1}\tilde{\boldsymbol{h}}_{j,1}, \tilde{\boldsymbol{h}}_{i,1}\tilde{\boldsymbol{h}}_{j,2} + \tilde{\boldsymbol{h}}_{i,2}\tilde{\boldsymbol{h}}_{j,1}, \tilde{\boldsymbol{h}}_{i,2}\tilde{\boldsymbol{h}}_{j,2}, \tilde{\boldsymbol{h}}_{i,3}\tilde{\boldsymbol{h}}_{j,1} + \tilde{\boldsymbol{h}}_{i,1}\tilde{\boldsymbol{h}}_{j,3}, \tilde{\boldsymbol{h}}_{i,3}\tilde{\boldsymbol{h}}_{j,2} + \tilde{\boldsymbol{h}}_{i,2}\tilde{\boldsymbol{h}}_{j,3}, \tilde{\boldsymbol{h}}_{i,3}\tilde{\boldsymbol{h}}_{j,3}\right)^\mathsf{T} \tag{38}$$

とおけば，

$$\begin{pmatrix} \boldsymbol{k}_{12}^\mathsf{T} \\ \boldsymbol{k}_{11}^\mathsf{T} - \boldsymbol{k}_{22}^\mathsf{T} \end{pmatrix} \boldsymbol{g} = 0 \tag{39}$$

と書ける．また，3次元位置と画像面との関係が明確な場合（かつ，$z = 0$ の場合）

$$\boldsymbol{\delta} = (h_{1,1}, h_{1,2}, h_{1,4}, h_{2,1}, h_{2,2}, h_{2,4}, h_{3,1}, h_{3,2}, h_{3,4}) \tag{40}$$

とすれば，

$$\begin{pmatrix} \tilde{\boldsymbol{x}}^\mathsf{T} & \boldsymbol{0}^\mathsf{T} & -u\tilde{\boldsymbol{x}}^\mathsf{T} \\ \tilde{\boldsymbol{x}}^\mathsf{T} & \boldsymbol{0}^\mathsf{T} & -v\tilde{\boldsymbol{x}}^\mathsf{T} \end{pmatrix} \boldsymbol{\delta} = 0 \tag{41}$$

が成り立つため，5点以上の対応関係があれば，特異値分解と呼ばれる（固有値分解と関連あり）計算により $\boldsymbol{\delta}$ が求められる．つまり，3次元位置と画像の対応点の組が取れれば，${}^c_w\boldsymbol{H}$ の一部が計算できる．内部パラメータ行列については，${}^c_w\boldsymbol{H}$ 一つにつき2本の方程式が得られるので，4枚以上の平面パターンを見せれば $\boldsymbol{g}$，すなわち $\boldsymbol{A}$ が計算できる．$\boldsymbol{A}$ が得られれば，${}^c_w\boldsymbol{R}$ と ${}^c\boldsymbol{o}_w$ が簡単に計算できる．

7.2.2 キャリブレーション手順

Zhang のキャリブレーションの手順を記す．

■チェスボードを用意する 図8のようなチェスボードを作図し，印刷する．

キャリブレーションプログラムではその前処理として，チェスボードの格子点を画像処理で抽出し，ボードのサイズ・格子点の相対情報などから抽出した点の 3 次元位置情報として利用する．よって，チェスボードの一マス分のサイズとマスの個数をプログラムに教える必要がある．演習向けに一マス分のサイズ 20mm のパターンを用意した．このチェスボードでは計 70 個の格子点を利用できる．また，図8のような座標系を設定する（長軸が x 軸，短軸が y 軸であり，z 軸が紙面上方向を向くものとする）．

■チェスボードの教示 70 個の格子点全てがカメラに写るようにし，適時撮像を繰り返す．

これにより画像上の点と3次元位置情報の対応が保存される．OpenCV では格子点の発見処理のために cvFindChessboardCorners という関数が用意されている．コーナーディテクタと呼ばれるテクニックが使われている．キャリブレーションには最低 4 枚の画像が必要である．

■パラメータの推定　複数回，画像と 3 次元位置の対応が取れたら，上記のキャリブレーション処理を行う．

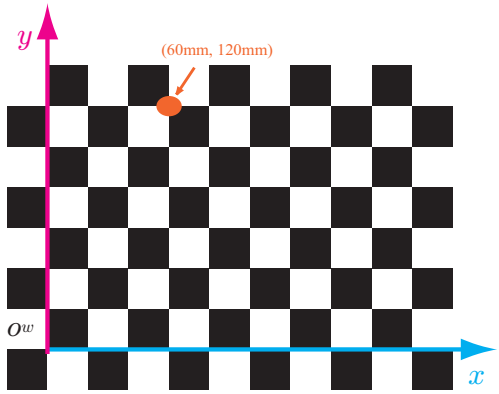

図 8　チェスボードと座標系

◇ 課題 2：カメラキャリブレーションと内部パラメータの取得 ◇

1. カメラキャリブレーションが必要になる状況を説明資料を参考に考えよ．また，カメラキャリブレーションの手順を理解せよ．
2. チェスボード紙と下敷きを TA より受け取れ．ペア一組当たり 1 枚ずつとする[a]．そして，チェスボード紙と下敷きを貼り合わせよ．
3. intCalib.c をコンパイル・リンクし intCalib というプログラムを作成せよ．その後，intCalib を実行せよ．
4. 実行中複数色の丸と直線が描画される状態になるよう（図 9）下敷きの位置姿勢を調整せよ．そして，図 9 の状態で Enter キーを押せ．画像対応点が保存される．その後，下敷きを少し動かし調整し，保存することを，あと 9 回繰り返せ（合計 10 回）．
5. プログラム終了後，intrinsic_param.txt というファイルを開いて，このファイルの中身を解釈せよ．基本的には $\boldsymbol{A}$ と「レンズ歪み」の各要素が記述されている．
6. intCalib.c の中身を理解せよ．（オプション）
7. **intCalib を実行した直後の intrinsic_param.txt を intrinsic_param_xxxxxx.txt として保存せよ**．xxxxx には学生証番号を入れること．

[a] チェスボード紙はソースにある PDF ファイルを印刷したものである．

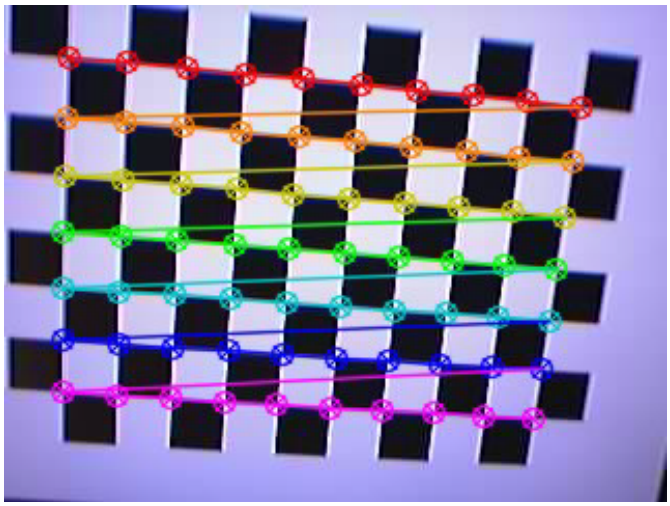

図 9　チェスボード画像を写したときのプログラムの出力画面

7.3　カメラ外部パラメータの取得

内部パラメータの取得手順は先の課題のようになるが，本日の演習では以降，カメラの内部パラメータは以下の値を用いることとする．

$$\boldsymbol{A} = \begin{pmatrix} 470 & 0 & 160 \\ 0 & 470 & 120 \\ 0 & 0 & 1 \end{pmatrix} \tag{42}$$

これは，以前の TA により内部パラメータキャリブレーションを数度行い得た値（の整数値の打ち切り）である．カメラによる個体差を考慮する場合，内部パラメータキャリブレーションは必須であるが今回はこの値を使う．この値は画像は 320 × 240 のサイズで取得することを前提としている．640 × 480 とする場合は全ての値を 2 倍に変更する必要がある．

◇ 課題 3：内部パラメータキャリブレーションの設定 ◇

1. 上記と intrinsic_param_xxxxx.txt[a]の値を参考に，intrinsic_param_ref.txt を完成させよ．[b]．
2. 先の課題で得られた値と書き換える値が違う場合，その理由を考えてみよ．

[a] xxxx は各自の学籍番号が入る
[b] ファイルを別名保存して，最初の 2 行を書き換えれば良い．

続いて，後続の課題のため，カメラの外部パラメータキャリブレーションを行ってみよう．intCalib というプログラムで取得したカメラの内部パラメータ行列 $\boldsymbol{A}$ を使い，自分が適当と思う位置・姿勢で投影したチェスボードの，カメラに対する位置・姿勢を取得する．すなわち，${}^{c}_{w}\boldsymbol{R}$ と ${}^{c}\boldsymbol{o}_{w}$ を求めてみよう．外部パラメータ

のキャリブレーションはホモグラフィ行列と内部パラメータ行列が得られるので，

$$\left(\begin{array}{cc} {}^{c}_{w}\boldsymbol{R} & {}^{c}\boldsymbol{o}_{w} \end{array} \right) = \boldsymbol{A}^{-1}{}^{c}_{w}\boldsymbol{H} \tag{43}$$

を計算している．

この処理の実装として extCalib.c を用意した[*3]．extCalib.c では，外部パラメータの獲得と同時に，そのタイミングで撮像した画像を board.jpg というファイルとして保存する．board.jpg というファイルを後続の課題で用いる．

◇ 課題 4：カメラの位置・姿勢のキャリブレーション ◇

1. extCalib.c をコンパイル・リンクし extCalib というプログラムを作成せよ．その後, extCalib を実行せよ．
2. 図 9 のように対応点を全て発見できるようになった状態で Enter を押してみよ．
3. ボードに対するカメラの位置・姿勢が計算され，extrinsic_param.txt に保存されるので，開け．
4. extrinsic_param.txt には ${}^{c}_{w}\boldsymbol{R}$，${}^{c}\boldsymbol{o}_{w}^{\mathsf{T}}$ が順に記録されている．妥当な結果が得られたか検証せよ．
5. 余裕があれば extCalib.c の中身を理解せよ．

8 座標変換・透視変換

◇ 課題 5：透視変換処理その１ ◇

1. projection_todo.c を projection.c としてコピーせよ．
2. project 関数の一部を説明資料の式 20 を参考に実装せよ．
3. projection.c をコンパイル・リンクし projection を作成し，実行せよ．
4. objectPoints[4],objectPoints[5] と xx, yy, xxx,yyy 等を適切に変更せよ．
5. 上記の結果，透視変換結果が見た目上わかりにくいと感じた場合，extCalib を再実行せよ．

このメカニズムを応用した処理を実現してみよう．ボード上の x, y 平面の座標値が (0mm, 0mm) から (180mm, 120mm) の矩形にフィットするようなサイズの絵があたかも貼りこまれているかのように，チェスボードを動かして見せたと同時にその絵も合わせて動くプログラムを書いてみよう．説明資料に書かれた，視線ベクトル

$${}^{w}\boldsymbol{x} = s{}^{w}\boldsymbol{q} - {}^{w}\boldsymbol{w} \tag{44}$$

[*3] 本来，内部パラメータ行列を取得する段階で，チェスボードの位置・姿勢も取得可能であるが，画像の透視変換を学ぶ例題をわかりやすくするために，外部パラメータ行列を別途求める手順をとった

を用いる（説明資料の式 35 の周辺を参考）．この視線ベクトルは座標系 w で記述されており，キャリブレーションボードの平面が w 座標系で $z = 0$ としているため，スケールファクタ s は

$$s = \frac{{}^{w}w_3}{{}^{w}q_3} \tag{45}$$

となるので，ボード上の点は，

$${}^{w}\boldsymbol{x} = \frac{{}^{w}w_3}{{}^{w}q_3}{}^{w}\boldsymbol{q} - {}^{w}\boldsymbol{w} \tag{46}$$

として計算できる．${}^{w}\boldsymbol{x}$ の x, y 座標が (0,0) から (180, 120) に収まっていれば，ボード上の点として描画計算を行う．

◇ **課題 6：透視変換処理その 2** ◇

1. inverseProjection_todo.c を inverseProjection.c としてコピーせよ．
2. computePositionOnPlane という関数で式 46 の処理が欠けている．それを補うよう実装せよ[a]．
3. inverseProjection.c をコンパイル・リンクし inverseProjection というプログラムを作成せよ．
4. inverseProjection というプログラムを実行せよ．
5. チェスボードをうまく映るようにしながら動かしてみよ．
6. 余裕があれば，ソースの中身をよくよみ理解を深めよ．

[a] 線形計算処理を用いるため，上述の説明資料を読むこと

9　単眼カメラの 3 次元位置推定

三次元空間中の点と画像の対応点から，カメラがチェスボードに対してどのような位置にあるか，計算し，最終的にその位置を表示するアプリケーションを作ってみよう．その手始めとして，これまでに学習した OpenGL や OpenCV を利用して，画像のキャプチャと CG 表示を行うサンプルアプリケーションを見てみよう．

◇ 課題 7：画像のキャプチャとカメラの 3 次元位置表示 ◇

1. cvglsample.c をコンパイル・リンクし cvglsample を作成せよ．
2. cvglsample を実行し，pthread, GLUT の復習をかねて，処理の中身を理解せよ．
3. cvglview.c をコンパイル・リンクし cvglview を作成せよ．
4. cvglview を実行し，直感的な結果を得るか確認せよ．

カメラを位置センサとして利用することが可能である．この技術の一つが，拡張現実感（Augmented Reality：AR）と呼ばれる分野であり，実空間と計算機空間を接続し，重ね合わせることが可能となる．まず，カメラから画像を取り込み，推定したカメラ位置に，カメラからキャプチャした映像を同時に表示することを試みる．

カメラからキャプチャしたテクスチャを，カメラ位置に置くと，カメラのファインダからのぞいた現実世界を見るような感じで，下記のように表示することが可能となる（図 10）．

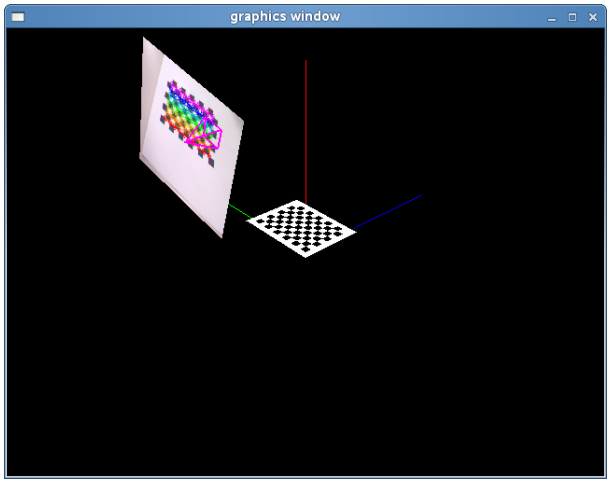

図 10　3 次元カメラ位置とキャプチャした画像

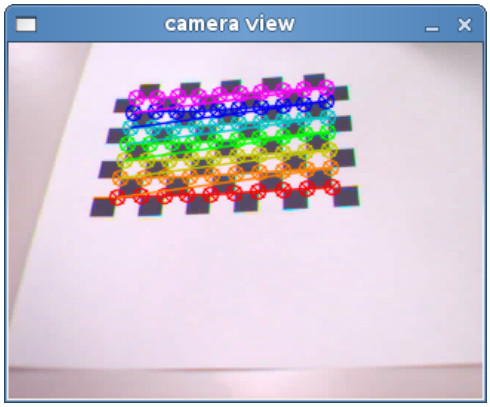

図 11　カメラからキャプチャした画像

◇ **課題** 8 ◇

1. 課題プログラムをダウンロードせよ．kadai8 のフォルダに入っている cvglcameraview.c をコンパイルし，実行せよ．
2. cvglcameraview.c を読んで，一通り何をやっているのか，把握せよ．基本的には OpenCV でカメラからのキャプチャをしている部分と，OpenGL でテクスチャマッピングをしている部分から成り立っている．まず，カメラキャプチャのスレッドを起動する部分を記述せよ（pthread を利用する）．
3. カメラ位置に，カメラからキャプチャしたテクスチャが貼られるように修正せよ．その際に，カメラ位置へテクスチャを移動する．また，回転はうまく行かないかもしれないが，少なくとも 1 軸は対応できるようにせよ．移動については，OpenCV におけるカメラの 3 次元位置推定結果を利用する．
4. 結果をスクリーンキャプチャせよ．

10　インタフェースとしてのカメラ・Wii リモコン

カメラの 3 次元位置をチェスボードとの相対関係で知ることができる．応用方法としての一つは上述したカメラの 3 次元位置を計算機内に描画することである．もう一つの方法としては，実世界でチェスボードの周囲でカメラを動かし，そのカメラの動きに合わせて，計算機の世界を変化させる．つまり，あたかもチェスボードに物が乗っているかのように動かすことで，視点を動かし，見回す映像を作ることができる．カメラを固定して，チェスボードを動かすことによっても同じような映像を作ることができる．今回は，OpenGL の世界の中の映像を見回すために，カメラをインタフェースとして利用する．また，同時に，Wii リモコンを利用し，Wii リモコンの動きに合わせて描画オブジェクトを変化させる．

座標マッピングの方法をうまく考えることによって，視点位置や描画画面の全体の変化をカメラで操作し，描画されているオブジェクトの個別の変化を Wii リモコンで操作することなど，直観的な操作が可能となる．インタフェースの特性を考慮に入れた設計などをすることが重要である．

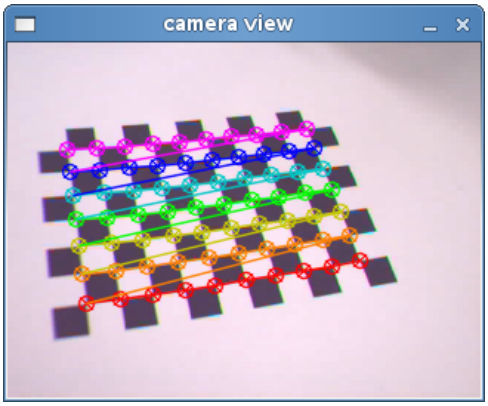

図 12　カメラからキャプチャした画像

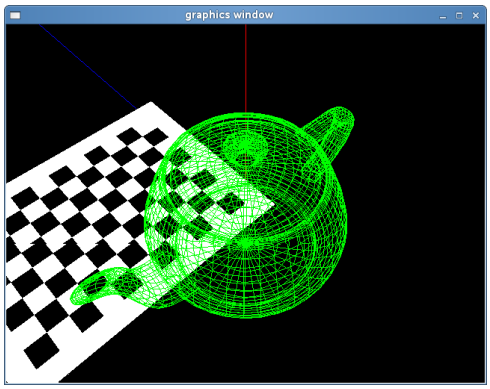

図 13　カメラの角度により，表示角度が変わる Teapot

◇ **課題** 9 ◇

1. kadai9 のフォルダに入っている cvglview_wii.c をコンパイルし，実行せよ．コンパイルには Makefile を用いる．実行方法は Bluetooth のアドレス 00:00:00:00:00:00 を引数にとり，実行する．Bluetooth のアドレスは貸し出した Wii リモコンの裏側に書かれている．

```
./cvglview_wii 00:00:00:00:00:00
```

2. Wii リモコン部分のソースが追加されているが，プログラム内でどう利用されているのか，ざっと読んで把握せよ．Wii リモコンのイベントのコールバックを実装せよ．その後，カメラ位置に合わせて映像が動くように gluLookAt を修正せよ．結果のイメージは，カメラ映像が図 12 であり，生成されているコンピュータグラフィックスが図 13 である．カメラの動きに応じて，Teapot の向きも変わることを確認せよ．
3. Wii リモコンのボタンや傾きによって，Teapot の移動や色の変化などインタラクションを加えよ．この変更結果はスクリーンキャプチャせよ．

11　インタフェースとしてのカメラ・Wii リモコンの応用

上記までの課題において，カメラの 3 次元位置をチェスボードとの相対関係を知り，操作可能なようになった．そこで，カメラからの映像をコンピュータグラフィックスの中に取り込み，その中にコンピュータグラフィックスを重ね合わせることにより，あたかもコンピュータグラフィックスが現実世界の上に乗っているように見せることが可能となる．カメラを動かすことにより位置を動かすことができ，Wii リモコンを動かすことにより他のパラメータをいじることができる．これらを発展させた技術が拡張現実感（AR）である．

図 14　カメラからキャプチャした現実世界

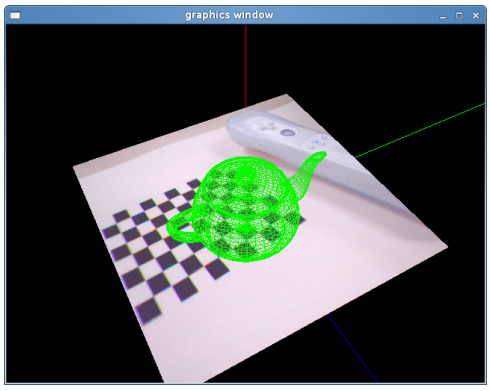

図 15　カメラからの現実世界の上に Teapot が乗っている状態

◇ **課題** 10 ◇

1. kadai10 のフォルダに入っている cvglarview.c をコンパイルし，実行せよ．コンパイルには Makefile を用いる．実行方法は Bluetooth のアドレス 00:00:00:00:00:00 を引数にとり，実行する．Bluetooth のアドレスは貸し出した Wii リモコンに書かれている．

```
./cvglarview 00:00:00:00:00:00
```

2. Wii リモコン部分，キャプチャ部分，表示部分など，プログラムの全体像を把握せよ．カメラ位置に合わせて映像が動くように gluLookAt を修正せよ．結果のイメージは，カメラ映像が図 14 であり，生成されているコンピュータグラフィックスが図 15 である．
3. 表示させる物体を teapot 以外の形に変化させよ．独自性が高いものが望ましい，その結果，現実世界の上に重なっているように見えるか試みてみよ．この変更結果はスクリーンキャプチャせよ．

◇ **課題メール** ◇

1. 自分の名前と学籍番号ををを書く．
2.「メディアインタフェース」の授業の感想と意見があれば意見をメールにて送付せよ．

参考文献

[1] John J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Longman Publishing Co., Inc., 2nd edition, 1989.

[2] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22, No. 11, pp. 1330–1334, November 2000.

[3] 出口光一郎. ロボットビジョンの基礎. コロナ社, 2000.

付録 A　Augmented Reality

現実環境と VR 環境をシームレスに融合させ，現実環境の情報の豊かさと VR 環境の柔軟性を兼ね備えた強化された環境を実時間に提供する，拡張現実感 (AR: AugmentedReality)，あるいは複合現実感 (MR: Mixed Reality) と呼ばれる技術が注目されている．これはヘッドマウントディスプレイ (HMD: Head Mounted Display) などを用いて実世界の画像と VR 物体を合成表示することにより，あたかも現実世界に仮想物体がそのまま出現したかのように見せる技術である．

CG 画像と実写画像の合成方法によって次の 2 つのタイプに分けられる．

- 光学シースルー (Optical See-Through) 方式
- ビデオシースルー (Video See-Through) 方式

それぞれの原理と特徴を述べると，前者は現実空間をハーフミラーで直接見て，VR 空間と重ね合わせる方式であり，実時間で安定して動作するという特徴がある．VR 物体による現実物体の遮蔽が困難であり，ハーフミラーを用いるために現実環境が若干暗く見え，VR 物体が半透明に見えるなどの短所がある．一方後者は，現実空間をビデオカメラで撮影して VR 空間と合成したあとにディスプレイに表示するものである．実写画像を一旦コンピュータに取り込んでから合成処理を施すため，シームレスな重ねあわせには有利である．しかし，処理の遅延により，船酔いや宇宙酔いのような VR 酔いを起こしやすい．

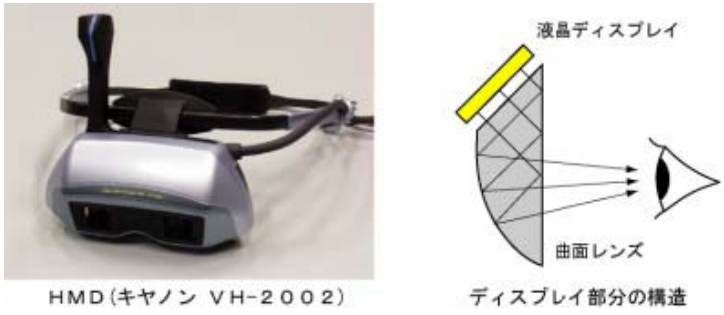

図 16　光学シースルー (Optical See-Through) 方式における HMD

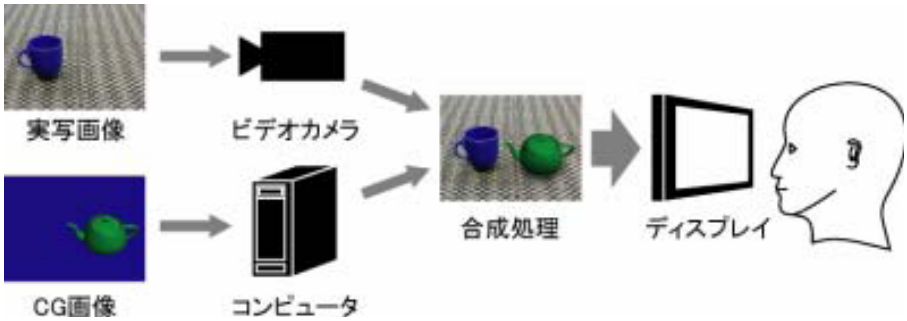

図 17　ビデオシースルー (Video See-Through) 方式

付録 B　ARToolKit

ARToolKit とは，これまで OpenCV や OpenGL で実現してきた拡張現実感（AR:Augmented Reality) の実現を容易にするためのライブラリ群である．ARToolKit はワシントン大学の HITL（Human Interface Technology Lab）により開発され，現在までに非常に多くの拡張現実感プロジェクトにおいて利用されている．ARToolKit Home Page (http://www.hitl.washington.edu/artoolkit/) よりダウンロードすることができる．

通常、AR アプリケーションを構築するには，画像解析の演習などで行った通り，カメラ等で現実世界を取り込んだ映像の解析，カメラや対象ビジュアルマーカの位置情報の解析，コンピュータグラフィックの生成及び合成といった非常に複雑な処理が必要になる．ARToolKit を利用することによりこれらのビジュアルマーカの検出アルゴリズムやカメラ位置の計算アルゴリズムグラフィックの生成等を行ってくれるため，開発者はビジュアルマーカ内の固有パターンの定義やグラフィックの動作等を定義することにより拡張現実感を実現することが出来る．

ARToolKit は現在 SGI IRIX, PC リナックス，Mac OS X，および PC Windows(95/98/NT/2000/XP) のオペレーティングシステムで動作する．ARToolKit の最新バージョンはマルチプラットフォームであり，ツールキットのそれぞれのバージョンの機能性は同じである．(図) ARToolkit のビデオキャプチャモジュールは，内部で OS に応じたインタフェースを使うようになっている．画像の描画が 3DCG における視点の設定 (グラフィック処理) には，OpenGL と GLUT を用いている．

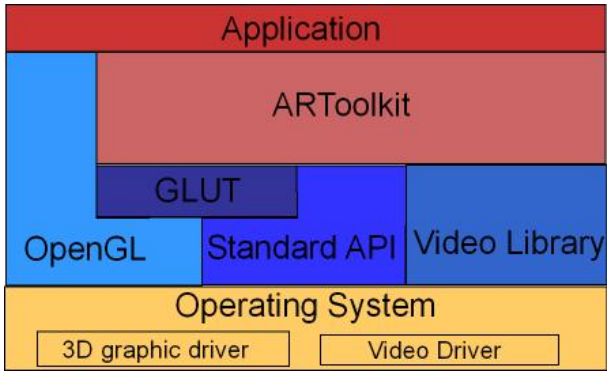

図 18　ARToolKit のアーキテクチャ

B.1　ARToolKit の概要

ARToolKit を使用するアプリケーションの開発は，アプリケーションを記述する部分と，アプリケーションで使用される実際のマーカー上の画像処理のトレーニング部に分かれる．

アプリケーションの主な構成は以下の方法を取る．

ステップ 1 と 6 はアプリケーションの初期化と閉鎖に実行され，ステップ 2〜5 はアプリケーションが終了するまでずっと繰り返される．これらのステップに加え，アプリケーションはマウス、キーボードまたは他のアプリケーションに特定のイベントの処理も行う．

初期設定	1. ビデオキャプチャを初期化. マーカーパターンとカメラパラメタの読み込み.
メインループ	2. ビデオ入力画像の取得.
	3. ビデオ入力画像にマーカーと認識されたパターンの検出.
	4. 検出されたパターンに比例してカメラパラメータを算出.
	5. 検出されたパターンに VR オブジェクトの描画.
終了	6. ビデオキャプチャの終了

表 1　ARToolKit の処理ステップ

B.2　ARToolKit のプログラム例

まずは，簡単な ARToolKit アプリケーションのソースコード (simpleTest(simple.c)) を通して，プログラムの書き方を説明する.

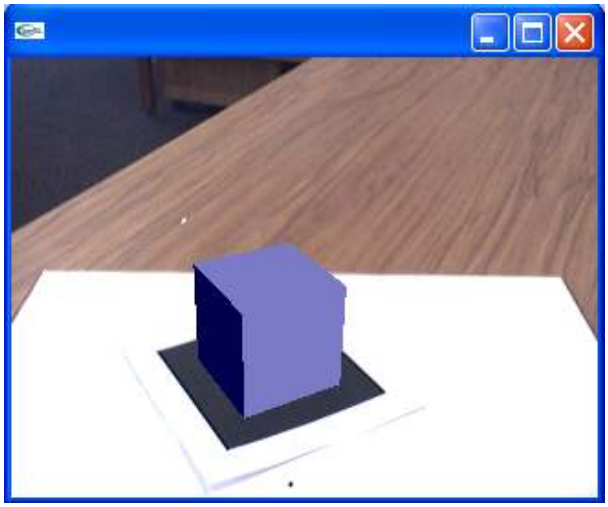

図 19　simpleTest の実行例

ARToolKit Step	Functions
1. Initialize the application	init
2. Grab a video input frame	arVideoGetImage (called in mainLoop)
3. Detect the markers	arDetectMarker (called in mainLoop)
4. Calculate camera transformation	arGetTransMat (called in mainLoop)
5. Draw the virtual objects	draw (called in mainLoop)
6. Close the video capture down	cleanup

表 2　ARToolKit の処理ステップと命令

プログラムの重要な部分は，main, init, mainLoop, draw, cleanup の 5 つになる.

main

main ルーチンは以下に示される通り．

```
main(int argc, char *argv[])
{
    init();
    arVideoCapStart();
    argMainLoop( NULL, keyEvent, mainLoop );
}
```

初期化ルーチンはマーカーとカメラパラメタを読み込み、ビデオキャプチャを始めるためのコード，およびウィンドウのセットアップを含んでいる．ステップ 1 に対応．次に、ビデオへの呼び出しで, arVideoCapStart 呼び，リアルタイムキャプチャが開始する．続いて，argMainLoop は，レンダリングを行う rendering loop を伴う main program loop を始動する．

init

初期化ルーチンは，main からコールされ，ビデオキャプチャの初期化，ARToolKit のアプリケーションパラメータの読み込みを行う．まず，video path を開き，画像のサイズを知る．

```
/* open the video path */
if( arVideoOpen( vconf ) < 0 ) exit(0);

/* find the size of the window */
if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);
```

ARToolKit アプリケーションパラメタの初期化．ARToolKit アプリケーションのための主要なパラメタは以下の通り．コマンドラインで指定できるファイル名か，デフォルトのパターンが，読まれる．カメラパラメータのデフォルト (Data/camera_para.dat).

- パターンテンプレート照合に使用されるパターンとこれらのパターンが相当する 3D オブジェクト．
- 使用されるビデオカメラのカメラ特性.

```
/* set the initial camera parameters */
if( arParamLoad(cparaname, 1, &wparam) < 0 ) {
    printf("Camera parameter load error !!\n");
    exit(0);
}
```

次に、パラメタは現在の画像サイズに応じて変更可能．サイズにより，同じカメラが使用されていても。カメラパラメタが変化する．

```
arParamChangeSize( &wparam, xsize, ysize, &cparam );
```

カメラパラメターの読み込み．カメラパラメータは，表示される．

```
arInitCparam( &cparam );
printf("*** Camera Parameter ***\n");
arParamDisp( &cparam );
```

パターンの定義を読み込む．(デフォルト：Data/patt.hiro)

```
if( (patt_id=arLoadPatt(patt_name)) < 0 ) {
    printf("pattern load error !!\n");
    exit(0);
}
```

patt_id は特定されたパターンの ID．

```
/* open the graphics window */
argInit( &cparam, 1.0, 0, 0, 0, 0 );
```

最後に，ウィンドウを開く．2 つめのパラメータは，ズームの比率．

mailLoop

ステップ 2-5 に対応する．まず，ビデオ画像は arVideoGetImage を使用することでキャプチャする．

```
/* grab a video frame */
if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
    arUtilSleep(2);
    return;
}
```

画面上にビデオ画像を表示．

```
argDrawMode2D();
argDispImage( dataPtr, 0,0 );
```

arDetectMarker はビデオ画像から正しいマーカパターンを持つ正方形を検出する．

```
if( arDetectMarker(dataPtr, thresh, &marker_info, &marker_num) < 0 ) {
    cleanup();
    exit(0);
});
```

発見されたマーカの数は，marker_num に含まれる．marker_info にはマーカの座標情報は，認識時の信頼度，ID が含まれる構造体のポインタが入る．

```
arVideoCapNext();
```

検出されたマーカのすべての信頼度を比較し，どの ID と相関が高いか調べる．

```
/* check for object visibility */
k = -1;

for( j = 0; j < marker_num; j++ ) {
    if( patt_id == marker_info[j].id ) {
        if( k == -1 ) k = j;
        else if( marker_info[k].cf < marker_info[j].cf ) k = j;
    }
}

if( k == -1 ) {
    argSwapBuffers();
    return;
}
```

arGetTransMat により，マーカーとカメラの間の Transformation を求める．

```
/* get the transformation between the marker and the real camera */
arGetTransMat(&marker_info[k], patt_center, patt_width, patt_trans);
```

マーカ i に対応したカメラ位置と方向が 3x4 マトリクス patt_trans に含まれる．

```
draw();
argSwapBuffers();
```

どのパターンも見つからなかった場合 (k==-1)，SwapBuffers のみ.

```
if( k == -1 ) {
    argSwapBuffers();
    return;
}
```

draw

The draw functions では，レンダリングの初期化, マトリックスのセットアップ, 物体のレンダリングに分かれる．ARToolKit に問い合わせを行い 3D レンダリングの初期化を行う.

```
argDrawMode3D();
argDraw3dCamera( 0, 0 );
glClearDepth( 1.0 );
glClear(GL_DEPTH_BUFFER_BIT);
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LEQUAL);
```

argConvGlpara により，ARToolKit で算出された 3x4 行列を，OpenGL 形式 (4x4 行列) に変換する．この値は，実際のカメラの位置と方向を表し，OpenGL のカメラ位置に設定することで，マーカー上にオブジェクトをレンダリングすることができる.

```
/* load the camera transformation matrix */
argConvGlpara(patt_trans, gl_para);
glMatrixMode(GL_MODELVIEW);
glLoadMatrixd( gl_para );
```

OpenGL の関数によりオブジェクトの描画．この場合，白色光の下の青い立方体を描画.

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
glLightfv(GL_LIGHT0, GL_AMBIENT, ambi);
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightZeroColor);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_flash);
glMaterialfv(GL_FRONT, GL_SHININESS, mat_flash_shiny);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
```

```
glMatrixMode(GL_MODELVIEW);
glTranslatef( 0.0, 0.0, 25.0 );
glutSolidCube(50.0);
```

OpenGL パラメータをデフォルトに戻す.

```
glDisable( GL_LIGHTING );
glDisable( GL_DEPTH_TEST );
```

cleanup

cleenup では，他のアプリケーションのため，ビデオプロセッシングを終了し video path を閉じる.

```
arVideoCapStop();
arVideoClose();
argCleanup();
```

◇ 課題参考課題 ◇

1. sampleTest.c をコンパイルし，カメラ画像が表示されること確認せよ．また，Hiro と書いてあるパターンにオブジェクトが重ねられることを確認せよ.
2. 表示されているオブジェクトを変更せよ.
3. 表示されているオブジェクトをアニメーションさせよ．上下移動や回転でよい.

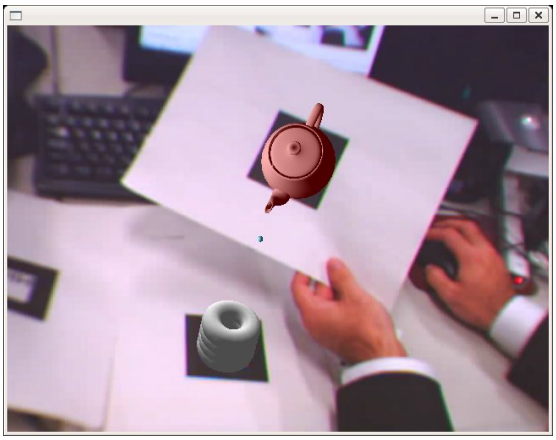

図 20　参考課題 実行時の画面

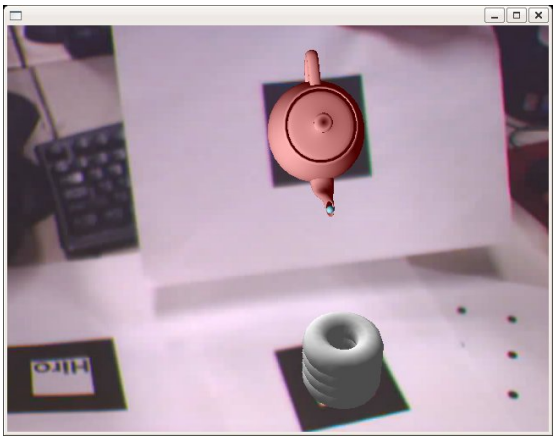

図 21　参考課題 実行時の画面