UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE – UFRN

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E COMPUTAÇÃO – PPGEEC

Disciplina: Visão Computacional – 2011.2

Prof. Luiz Marcos Garcia Gonçalves

Alunos: **André Tavares da Silva e Leonardo Enzo Brito da Silva**

NATAL-RN
OUTUBRO/2011

RELATÓRIO DO TRABALHO COMPUTACIONAL 2

MÉTODOS DE CALIBRAÇÃO DE CÂMERA

SUMÁRIO

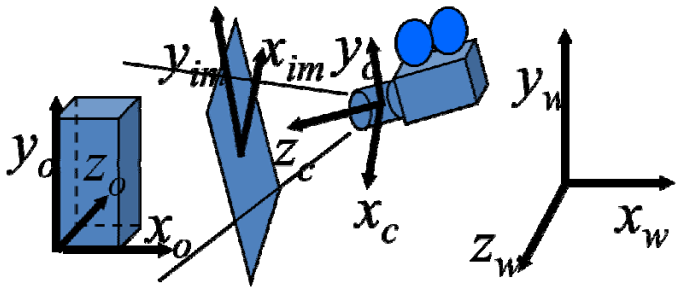
RESUMO

Este relatório tem por objetivo demonstrar os resultados obtidos através de experimentos computacionais de alguns métodos de calibração de câmeras, a saber, o método implementado pela biblioteca openCV, o método direto e pela matriz de projeção, ambos do Trucco, e por fim, o de Church e Ganapathy.

Todos os métodos de calibração descritos abaixo foram implementados na linguagem C++, fazendo-se uso também da biblioteca OpenCV e, somente para se plotar os gráficos, foi utilizado o software Matlab R2010a.

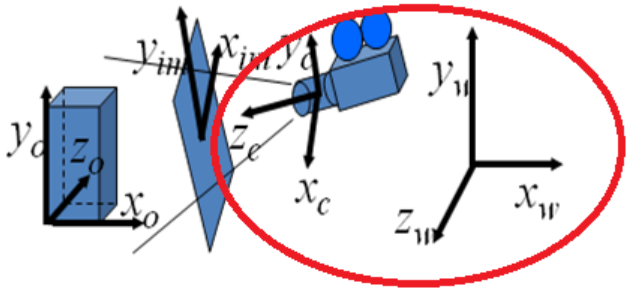
1 INTRODUÇÃO

Inicialmente é importante explicar o que é calibração e sua importância. Calibração é o processo pelo qual parâmetros internos ou intrínsecos e externos ou extrínsecos da câmera são obtidos ou estimados de forma que se possa localizar sistemas ou objetos em relação a um frame(ou sistema de coordenadas) de mundo a partir apenas de imagens de objetos da cena. Reconhecimento e reconstrução 3D com conhecimento da geometria real do objeto pode ser muito mais eficiente.



2 O PROBLEMA GERAL DE CALIBRAÇÃO

Os parâmetros extrínsecos são os que relacionam as coordenadas de mundo(xw,yw, zw) de pontos na cena com as coordenadas de câmera desses mesmos pontos, ou seja, coordenadas desses pontos no frame de câmera(xc,yc,zc). Essa relação é dada por uma rotação seguida de uma translação e matricialmente pode ser representada assim:



$$P_c = \mathbf{R}(P_w - \mathbf{T}) \Rightarrow \begin{bmatrix} xc \\ yc \\ zc \end{bmatrix} = \begin{bmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{bmatrix} * \left(\begin{bmatrix} xw \\ yw \\ zw \end{bmatrix} \pm \begin{bmatrix} T1 \\ T2 \\ T3 \end{bmatrix} \right)$$

Ou em coordenadas homogêneas:

$$\begin{bmatrix} xc \\ yc \\ zc \\ 1 \end{bmatrix} = \begin{bmatrix} r11 & r12 & r13 & \pm T1 \\ r21 & r22 & r23 & \pm T2 \\ r31 & r32 & r33 & \pm T3 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} xw \\ yw \\ zw \\ 1 \end{bmatrix} (1)$$

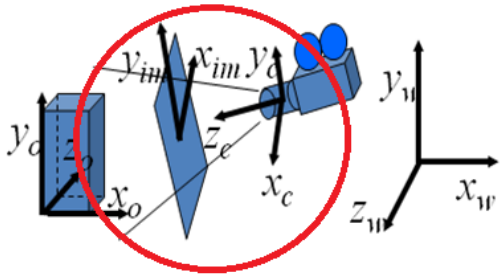
Resumindo, os parâmetros extrínsecos são os 9 Rij's ou outros parâmetros que os representem (por exemplo, 3 ângulos de Euler – Roll, Pitch, Yall, rotações ao redor dos eixos x, y e z, respectivamente) e os 3 de translação entre frame de mundo e frame de câmera T1, T2 e T3, sendo o sinal $\pm$ devido ao fato de alguns autores comsiderarem a translação ou positiva ou negativa.

Os parâmetros intrínsecos são os que relacionam as coordenadas de câmera(xc,yc, zc) com as coordenadas de imagem (de pixels) (X,Y,Z). Caracterizam as propriedades óticas, geométricas e digitais da câmera visualizadora. Para pin-hole, 3 conjuntos:

-projeção perspectiva (único parâmetro é *f*)

-transformação entre frames de câmera e píxel

-distorção geométrica introduzida pelo sistema ótico



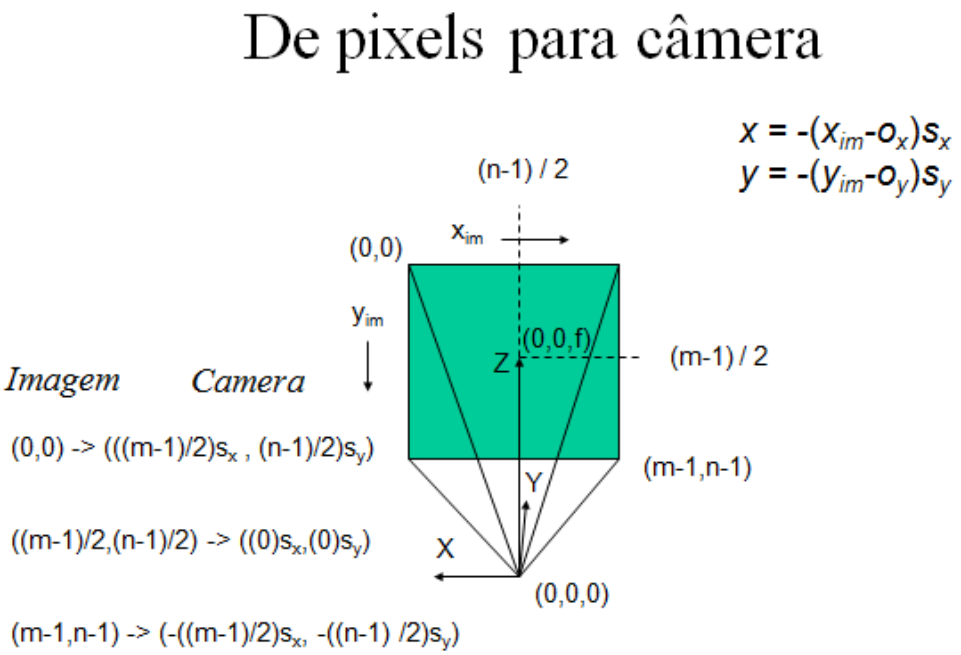
Inicialmente, os pontos que são representados no frame de câmera precisam ser projetados no que se chama plano projetivo de câmera, usando-se para isso a distância focal f e as coordenadas xc, yc e zc:

$$(xproj, yproj) = (f * \frac{xc}{zc}, f * \frac{yc}{zc}) (2)$$

Neglicenciando distorções e assumindo que o CCD é uma matriz retangular, a relação entre coordenadas projetadas (xproj, yproj) e de imagem ou pixel (X,Y) é dada por:

$$xproj = -(X \pm ox) * sx \; e \; yproj = -(Y \pm oy) * sy \; (3)$$

Sendo (o_x,o_y) as coordenadas em pixel do centro da imagem (ponto principal ou intercessão do eixo ótico da câmera com o plano projetivo) e sx e sy o tamanho efetivo do pixel (em milímetros) horizontal e verticalmente, respectivamente. O sinal $\pm$ é devido ao fato de que alguns autores consideram (+ ox, +oy) ao invés de (-ox, -oy), é apenas um modo diferente de se analizar o problema. Para o caso geral será considerado o par (-ox, -oy).



Se houver distorção radial, causada pelo fato de a lente não ser ideal (possuir espessura considerável), tem-se:

$$xproj\_dist = -(X - ox) * sx \; e \; yproj\_dist = -(Y - oy) * sy \quad (4)$$

E o fator de correção é:

$$xproj = xproj_{dist} * (1 + k1 * r^2 + k2 * r^4) \quad (5)$$

$$yproj = yproj_{dist} * (1 + k1 * r^2 + k2 * r^4)$$

Sendo $r^2 = xproj\_dist^2 + yproj\_dist^2$

Então, com correção de distorção:

$$xproj = -(X - ox) * sx * (1 + k1 * r^2 + k2 * r^4) \quad (6)$$

$$yproj = -(Y - oy) * sy * (1 + k1 * r^2 + k2 * r^4)$$

Por (2) e (6) conclui-se que os parâmetros intrínsecos são 7: f, ox, oy, sx, sy, k1 e k2.

Deseja-se agora relacionar diretamente coordenadas de pixel (X,Y) com coordenadas de mundo (xw,yw,zw):

Substituindo-se (2) em (1):

$$xproj = f * \frac{r11*xw+r12*yw+r13*zw-T1}{r31*xw+r32*yw+r33*zw-T3} \quad (7)$$

$$yproj = f * \frac{r21*xw+r22*yw+r23*zw-T2}{r31*xw+r32*yw+r33*zw-T3}$$

Substituindo-se (6) em (7):

$$X = -\frac{f}{sx} * \frac{r11*xw+r12*yw+r13*zw-T1}{r31*xw+r32*yw+r33*zw-T3} * (1 + k1 * r^2 + k2 * r^4) + ox \quad (8)$$

$$Y = -\frac{f}{sy} * \frac{r12*xw+r22*yw+r23*zw-T2}{r31*xw+r32*yw+r33*zw-T3} * (1 + k1 * r^2 + k2 * r^4) + oy$$

Negligenciando distorção radial, as equações acima (8) podem ser representadas de forma matricial:

$$Mint = \begin{bmatrix} \frac{-f}{sx} & 0 & ox \\ 0 & \frac{-f}{sy} & oy \\ 0 & 0 & 1 \end{bmatrix} \text{ e } Mext = \begin{bmatrix} r11 & r12 & r13 & -R1^t * T \\ r21 & r22 & r23 & -R2^t * T \\ r31 & r32 & r33 & -R3^t * T \end{bmatrix} (9)$$

Sendo $R1^t = [r11\ r12\ r13]$, $R2^t = [r21\ r22\ r23]$ e $R3^t = [r31\ r32\ r33]$

Então: $\begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = Mint * Mext * \begin{bmatrix} xw \\ yw \\ zw \\ 1 \end{bmatrix}$ (10)

$$M = Mint * Mext$$

Pode-se simplificar, considerando- se sx=sy=1, ox=oy=0 :

$$M = \begin{bmatrix} -f * r11 & -f * r12 & -f * r13 & f * R1^t * T \\ -f * r21 & -f * r22 & -f * r23 & f * R2^t * T \\ r31 & r32 & r33 & -R3^t * T \end{bmatrix} (11)$$

$$X = -f * \frac{r11*xw+r12*yw+r13*zw-T1}{r31*xw+r32*yw+r33*zw-T3} \quad (12)$$

$$Y = -f * \frac{r12*xw+r22*yw+r23*zw-T2}{r31*xw+r32*yw+r33*zw-T3}$$

Se a perspectiva for fraca, ou seja, se o objeto estiver muito distante na cena, ou seja, a distância entre o centróide de um conjunto de pontos no objeto e o centro de projeção ao longo do eixo ótico($(P^{'} - T)$ for muito maior que a distância entre tais pontos:

$$Mwp = \begin{bmatrix} -f*r11 & -f*r12 & -f*r13 & f*R1^t*T \\ -f*r21 & -f*r22 & -f*r23 & f*R2^t*T \\ 0 & 0 & 0 & R3^t*(P^{'}-T) \end{bmatrix} (13)$$

Pode-se, por convenção, se escrever as equações (11) e (12) de maneira diferente:

$$M = \begin{bmatrix} -f*r11 & -f*r12 & -f*r13 & f*R1^t*T \\ -f*r21 & -f*r22 & -f*r23 & f*R2^t*T \\ r31 & r32 & r33 & +R3^t*T \end{bmatrix} (14)$$

$$X = +f * \frac{r11*xw+r12*yw+r13*zw-T1}{r31*xw+r32*yw+r33*zw-T3} \quad (15)$$

$$Y = +f * \frac{r12*xw+r22*yw+r23*zw-T2}{r31*xw+r32*yw+r33*zw-T3}$$

Impõe-se ainda restrições de ortonormalidade na matriz R:

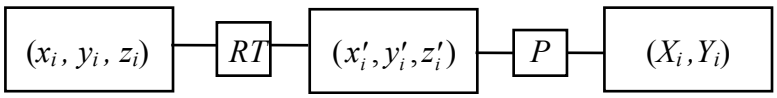
$$R = \begin{bmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{bmatrix}$$

$$\left.\begin{array}{l} r11^2 + r21^2 + r31^2 = 1 \\ r12^2 + r22^2 + r32^2 = 1 \\ r13^2 + r23^2 + r33^2 = 1 \\ R1 * R2 = r11*r12 + r21*r22 + r31*r32 = 0 \\ R1 * R3 = r11*r13 + r21*r23 + r31*r33 = 0 \\ R2 * R3 = r12*r13 + r22*r23 + r32*r33 = 0 \end{array}\right\} \quad (16)$$

Ou, se R1, R2 estiverem normalizados, R1X R2 = R3.

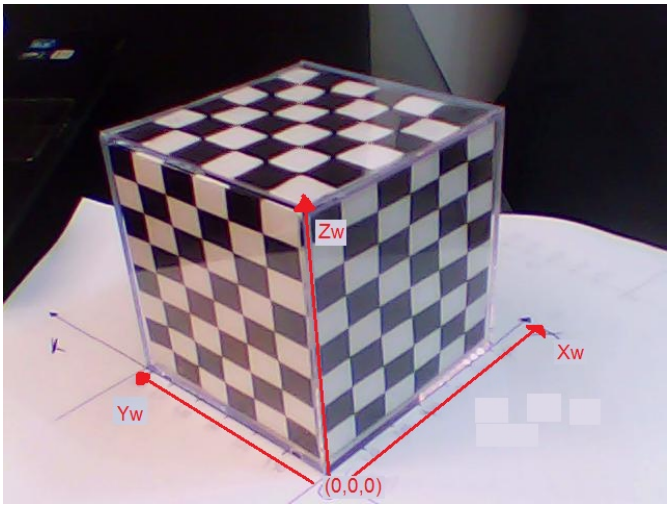
Calibrar a câmera é montar um sistema de equações com pontos de cena ou objeto em coordenadas de mundo conhecidas(Pw1, Pw2, …) e suas respectivas imagens (coordenadas de pixel) para se encontrar os 13 parâmetros de (15) ou os 19 de (8) se considerarmos distorção e não fizermos as simplificações.

Resume-se com o esquema abaixo o processo de calibração:

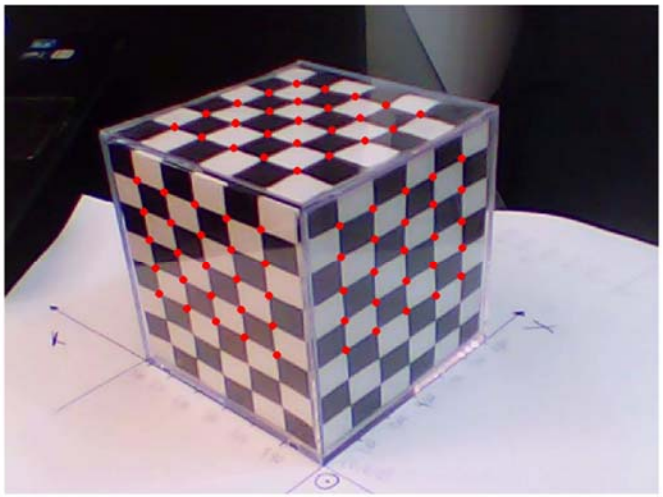


3 CASO ESPECÍFICO

O objetivo deste trabalho é calibrar a câmera webcam Clone VGA com taxa de amostragem de 30 fps por alguns métodos e se comparar os resultados de cada um. Para isso, utilizou-se o seguinte objeto de referência cujas coordenadas foram arbitradas no eixo de mundo (xw,yw,zw):



Objeto de calibração, cubo xadrez.



Pontos selecionados.

4 MÉTODO DA BIBLIOTECA OPENCV

4.1 Descrição do núcleo da função

A biblioteca openCV possui uma função chamada cvCalibrateCamera2 que usa o método do Tsai e dados (parâmetros) fornecidos para se calcular parâmetros extrínsecos e intrínsecos da câmera. Ela possui a seguinte sintaxe:

cvCalibrateCamera2(object_points2, image_points2, point_counts2, cvGetSize(image), intrinsic_matrix, distortion_coeffs, rotation_vector, translation_vector);

Descrição dos parâmetros:

=>object_points2 = ponteiro para a matriz de pontos conhecidos no mundo,ou seja, pontos de cena ou de objeto em coordenada de mundo, com tamanho n x 3, n linhas por 3 colunas, sendo as colunas correspondentes as coordenadas em xw, yw e zw, respectivamente.

=> image_points2= ponteiro para a matriz de pontos conhecidos na imagem,ou seja, em coordenadas de pixel, com tamanho n x 2, n linhas por 2 colunas, sendo as colunas correspondentes as coordenadas em X, Y de pixel, respectivamente.

=> point_counts2= ponteiro para a matriz 1 x 1 que representa a quantidade de planos usados no processo de calibração.

=> cvGetSize(image)= tamanho de cada imagem passada ao programa.

=> intrinsic_matrix = ponteiro para a matriz de parâmetros intrínsecos calculada pela função. É saída da função e possui o seguinte formato:

$$Mint = \begin{bmatrix} fx & 0 & cx = ox \\ 0 & fy & cy = oy \\ 0 & 0 & 1 \end{bmatrix}$$

=> distortion_coeffs = ponteiro para a o vetor de coeficientes de distorção da lente(radial, k1, k2 e k3) e do sensor (distorção tangencial p1 e p2, devido ao sensor não ser paralelo à lente, mas inclinado). É saída da função e possui o seguinte formato: $Dist = [k1 \quad k2 \; p1 \quad p2 \quad k3]$

=> rotation_vector = ponteiro para a matriz de vetores eixo de rotação cujo número de linhas depende da quantidade de planos ou imagens ou vistas usadas para calibração. Formato:

$$R = \begin{bmatrix} a11 & a12 & a13 \\ \dots & \dots & \dots \\ an1 & an2 & an3 \end{bmatrix}$$

Sendo cada linha o eixo de rotação do espaço tridimensional em coordenadas de câmera correspondente à rotação entre um plano imagem e o sistema de coordenadas de mundo. O módulo de cada linha é o ângulo de rotação no sentido anti-horário ao redor daquele eixo. Usando a transformada de Rodrigues (dada pela função cvRodrigues2()) pode-se obter de cada eixo a matriz 9 x 9 de rotação.

=> translation_vector = ponteiro para a matriz de vetores de translação cujo número de linhas depende da quantidade de planos ou imagens ou vistas usadas para calibração. Formato:

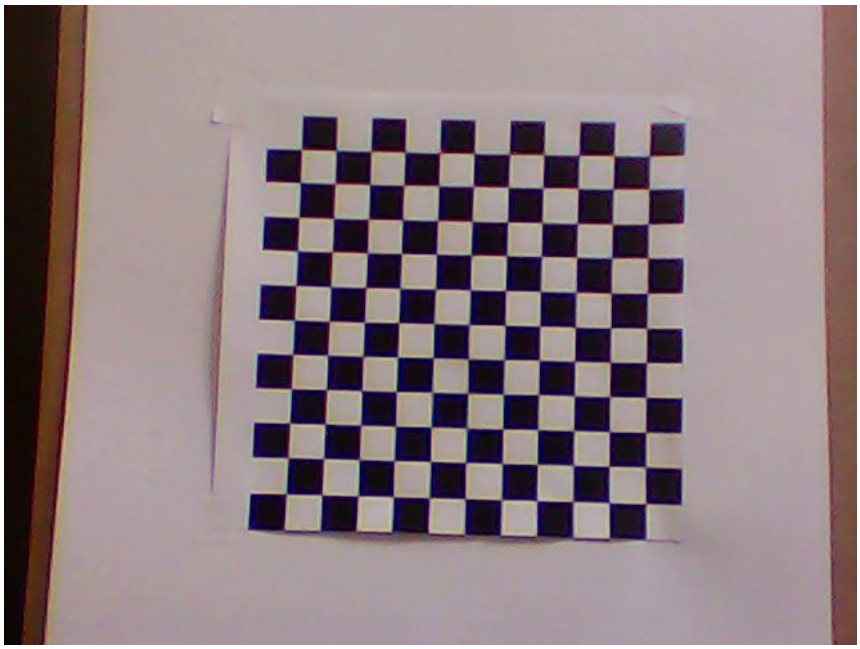
$$T = \begin{bmatrix} t11 & t12 & t13 \\ \dots & \dots & \dots \\ tn1 & tn2 & tn3 \end{bmatrix}$$

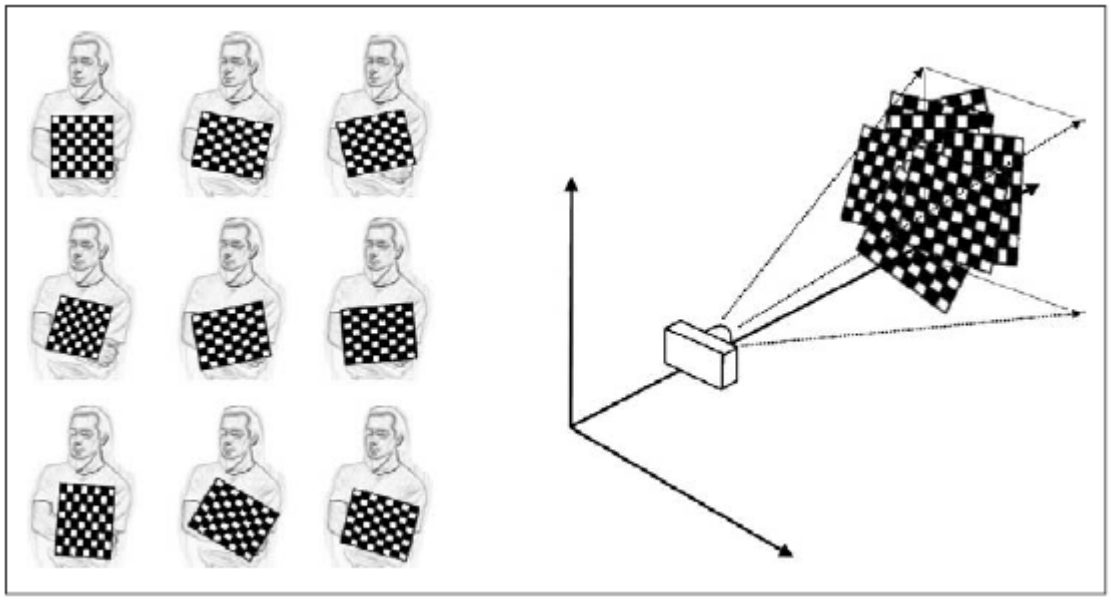
Sendo cada linha o vetor de translação do espaço tridimensional correspondente à translação entre um plano imagem e o sistema de coordenadas de mundo.

4.2 Como funciona o método

Para que a função acima funcione adequadamente é necessário que se use como entrada dela vários conjuntos de coordenadas de pontos (de objeto e respectivos de imagem ou pixel) coplanares, ou seja, cada conjunto de pontos apresentado deve pertencer ao mesmo plano. Isso é garantido quando se usa como objeto de calibração uma espécie de tabuleiro de xadrez, ou simplesmente xadrez, e se varia a sua posição com relação à câmera. Esssa restrição é fundamental, pois a função cvCalibrateCamera2 é capaz de detectar se os pontos fornecidos são coplanares ou não, e se não forem, não consegue determinar a matriz intrínseca e a pede como dado de entrada.

Para esse método foi utilizado o seguinte xadrez, cujos quadrados (144) medem 1 x 1 cm cada:

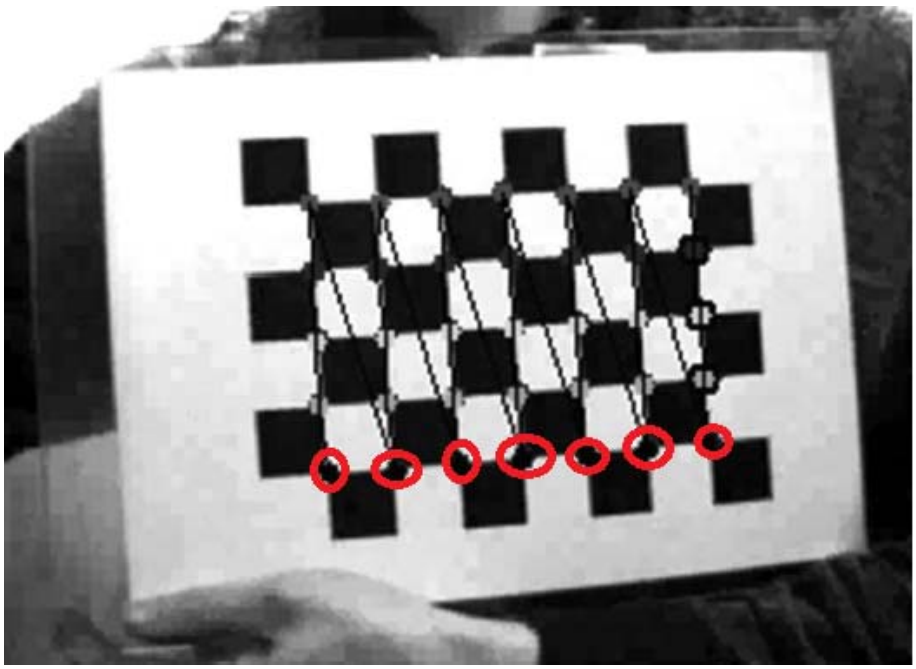




Método de se variar a pose (orientação mais translação) de um tabuleiro de xadrez para se obter conjuntos de pontos coplanares

Para cada posição do tabuleiro de xadrez acima foram armazenados as coordenadas em pixels e em coordenadas de mundo dos pontos das quinas de cada quadrado. Quinas são os encontros entre os cantos dos quadrados. Há uma função na biblioteca openCV que localiza as coordenadas das quinas de um tabuleiro de xadrez dada a quantidade de quinas existentes na imagem, o

tamanho do tabuleiro e a imagem (foto) do tabuleiro: cvFindChessboardCorners(image, board_sz, corners, &corner_count,CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS);



Destaque em algumas quinas encontradas em determinado tabuleiro.

Se a função acima detectar a quantidade de quinas informada, no caso do tabuleiro utilizado na calibração, 121 quinas (11 x 11 internas) , ela acha as coordenadas de cada uma delas em uma precisão maior (subpixel) e armazena tais coordenadas em 2 matrizes diferentes, uma de coordenadas de ponto de objeto e outra de coordenadas de pontos imagem (pixel). Após cada sucesso, a posição do tabuleiro deve ser mudada e o processo se reinicia, sendo os resultados acumulados nas matrizes supra-citadas. Caso a função não detecte todas as quinas, ela continua a procurar por elas.

A quantidade de poses de tabuleiro a serem utilizadas pode ser escolhida pelo usuário, mas pode-se mostrar que deve ser no mínimo duas poses.

Com esse algoritmo, já existem parâmetros de entrada suficientes da função cvCalibrateCamera2 e então se pode utilizá-la para se calcular os parâmetros extrínsecos e intrínsecos de câmera e os coeficientes de distorção.

Como os parâmetros intrínsecos só dependem da câmera, eles não variam de uma pose para outra do tabuleiro,ou seja, estão em uma matriz única, mas os extrínsecos variam. Além disso, o objeto de calibração é o cubo xadrez e não o tabuleiro. Então, foi usada a função de calibração do openCV apenas para se encontrar os parâmetros intrínsecos e outra função, a *cvFindExtrinsicCameraParams2(obj, imgg, intrinsic, distortion, rotation_vector,translation_vector);,* foi usada para se obter a matriz de parâmetros extrínsecos, tendo como entrada a matriz de parâmetros intrínsecos, as coordenadas objeto e imagem dos pontos do cubo xadrez, e os coeficientes de distorção encontrados.

4.3 Resultados obtidos

Foi criado um código em c++ com o nome *calibra_ocv.cpp* para se implementar o método descrito acima, o qual se encontra no anexo2 deste relatório. O número de tabuleiros (poses) utilizadas foi 10. O resultado final se encontra nas tabelas abaixo:

K1	K2	P1	P2	K3
-0.23104931	**13.11951447**	**-0.02363417**	**2.40606e-03**	**-98.4288101**

Coeficientes de distorção

$$Mint = \begin{bmatrix} fx & 0 & cx = ox \\ 0 & fy & cy = oy \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{988.66210938} & 0 & \mathbf{335.32232666} \\ 0 & \mathbf{981.93444824} & \mathbf{248.17175293} \\ 0 & 0 & 1 \end{bmatrix}$$

$$Rotations = \begin{bmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{bmatrix} = \begin{bmatrix} \mathbf{0.68270785} & \mathbf{-0.72843552} & \mathbf{-0.05737369} \\ \mathbf{-0.39022160} & \mathbf{-0.29708555} & \mathbf{-0.87147421} \\ \mathbf{0.61776787} & \mathbf{0.61735070} & \mathbf{-0.48707390} \end{bmatrix}$$

$$Translations = \begin{bmatrix} T1 \\ T2 \\ T3 \end{bmatrix} = \begin{bmatrix} \mathbf{-0.80445695} \\ \mathbf{6.36146545} \\ \mathbf{31.54652214} \end{bmatrix} = 32.19 * \begin{bmatrix} \mathbf{-0.0250} \\ \mathbf{0.1976} \\ \mathbf{0.9799} \end{bmatrix}$$

5 MÉTODO DE CHURCH E GANAPATHY

5.1 Formulação geral do problema

A equação geral do problema de calibração é relembrada a seguir:

$$Xp + ox = \frac{f}{sx} * \frac{r11*xw+r12*yw+r13*zw+T1}{r31*xw+r32*yw+r33*zw+T3} \quad (15)$$

$$Yp + oy = \frac{f}{sy} * \frac{r21*xw+r22*yw+r23*zw+T2}{r31*xw+r32*yw+r33*zw+T3}$$

O método considera $\frac{f}{sx} = \frac{f}{sy} = 1$

ox=(640-1)/2=319

oy=(480-1)/2=239

e considera pontos coplanares num plano genérico xw = 0.

Logo, tem-se:

$$Xp = \frac{r12*yw+r13*zw+T1}{r32*yw+r33*zw+T3} - ox \quad (15)$$

$$Yp = \frac{r22*yw+r23*zw+T2}{r32*yw+r33*zw+T3} - oy$$

$$Xp = \frac{r12*yw+r13*zw+T1-r32*yw*ox-r33*zw*ox-T3*ox}{r32*yw+r33*zw+T3}$$

$$Yp = \frac{r22*yw+r23*zw+T2-r32*yw*oy-r33*zw*oy-T3*oy}{r32*yw+r33*zw+T3}$$

$$r12 * yw + r13 * zw + T1 - r32 * yw * ox - r33 * zw * ox - T3 * ox - r32 * yw * Xp - r33 * zw * Xp - T3 * Xp = 0$$

$$r22 * yw + r23 * zw + T2 - r32 * yw * oy - r33 * zw * oy - T3 * oy - r32 * yw * Yp - r33 * zw * Yp - T3 * Yp = 0$$

As equações acima, sob forma matricial ficam:

A*X=0

$$A = \begin{bmatrix} yw1 & zw1 & 0 & 0 & -(yw1 * ox + yw1 * Xp1) & -(zw1 * ox + zw1 * Xp1) & 1 & 0 & -(Xp1 + ox) \\ 0 & 0 & yw1 & zw1 & -(yw1 * oy + yw1 * Yp1) & -(zw1 * oy + zw1 * Yp1) & 0 & 1 & -(Yp1 + oy) \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ ywn & zwn & 0 & 0 & -(ywn * ox + ywn * Xpn) & -(zwn * ox + zwn * Xpn) & 1 & 0 & -(Xpn + ox) \\ 0 & 0 & xwn & ywn & -(xwn * oy + xwn * Ypn) & -(ywn * oy + ywn * Ypn) & 0 & 1 & -(Ypn + oy) \end{bmatrix}$$

$$X = \begin{bmatrix} r12 \\ r13 \\ r22 \\ r23 \\ r32 \\ r33 \\ T1 \\ T2 \\ T3 \end{bmatrix} \qquad A * \begin{bmatrix} r12 \\ r13 \\ r22 \\ r23 \\ r32 \\ r33 \\ T1 \\ T2 \\ T3 \end{bmatrix} = \begin{bmatrix} 0 \\ \dots \\ 0 \end{bmatrix}$$

O posto de A é 9, mas se houver mais equações que variáveis, pode-se resolver esse sistema linear homogêneo por SVD(singular value decomposition).

O método SVD consiste em se decompor a matriz A em 3 outras:

$$A_{mxn} = U_{mxm} * W_{mxn} * V^T_{nxn}$$

Sendo $W_{mxn} = \begin{bmatrix} \sigma 1 & 0 & \dots & 0 \\ 0 & \sigma 2 & 0 & \dots \\ \dots & 0 & \dots & 0 \\ 0 & \dots & 0 & \sigma n \end{bmatrix}$

$\sigma i \geq 0$ *são os valores singulares de A*

O resultado do sistema é dado pela coluna de V correspondente ao valor singular de W nulo ou ao menor deles, ou seja, é a última coluna de V.

Com a solução desse sistema já se tem o vetor de translação T e as duas últimas colunas da matriz de rotação. Para se obedecer às restrições de ortonormalidade da matriz de rotação, se normaliza essas duas colunas(vetores coluna) e se determina a primeira pelo produto vetorial entre as duas últimas (entre os dois vetores coluna).

5.2 Resultados

Foi criado um código em C++ *calibra_church.cpp* que implementa todo o método acima. Há uma função da biblioteca openCV que resolve sistemas por SVD:

cvSVD(A,W,U,V).

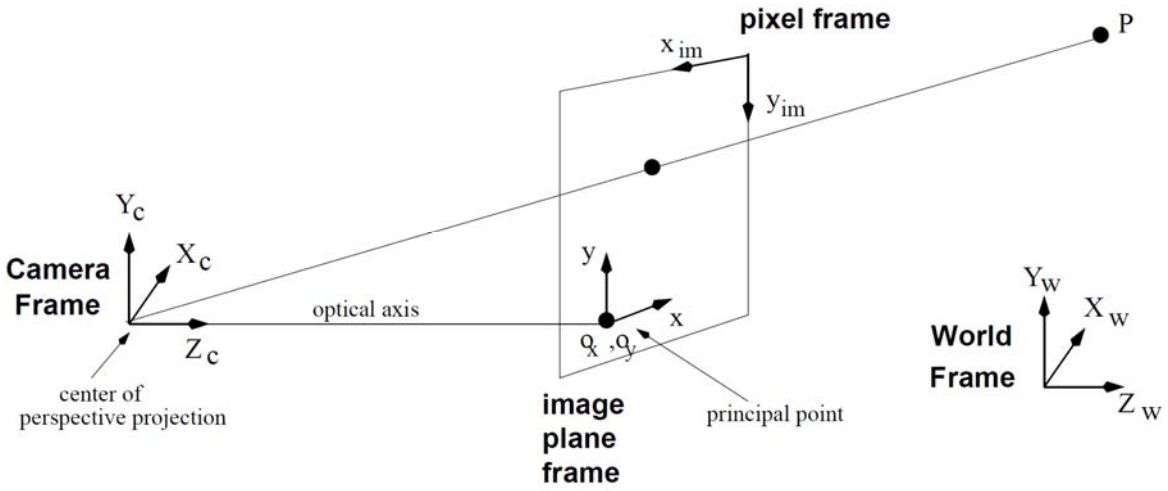
Os resultados obtidos se encontram nas tabelas abaixo:

$$Rotations = \begin{bmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{bmatrix} = \begin{bmatrix} 0{,}001752 & -0{,}999954 & -0{,}430110 \\ -0.001454 & -0{,}009407 & -0{,}902776 \\ 0{,}898689 & 0{,}001934 & -0{,}000622 \end{bmatrix}$$

$$Translations = \begin{bmatrix} T1 \\ T2 \\ T3 \end{bmatrix} = \begin{bmatrix} 0{,}678460 \\ 0{,}733072 \\ 0.001082 \end{bmatrix} = 0.9989 * \begin{bmatrix} 0.6792 \\ 0.7339 \\ 0.001083 \end{bmatrix}$$

6 MÉTODO DIRETO (TRUCCO)

Como já foi mencionado anteriormente, o objetivo da calibração consiste em estimar os parâmetros intrínsecos e extrínsecos da câmera dado um conjunto contendo uma ou mais imagens de um padrão de calibração. Uma nota importante a ser mencionada é que o ponto principal não é sempre o centro de "real" da imagem, como pode ser visto na figura a seguir.



O algoritmo de calibração consiste em duas etapas:

Etapa 1: assumindo que o_x e o_y são conhecidos, estimar todos os demais parâmetros.

Etapa 2: estimar o_x e o_y.

6.1 Etapa 1

6.1.1 Passo 1

Obter as coordenadas de mundo dos pontos escolhidos para a calibração.

6.1.2 Passo 2

Obter as coordenadas de imagem dos pontos escolhidos para a calibração.

6.1.3 Passo 3

Estabelecer a correspondência entre os pontos de imagem e de mundo e resolver o seguinte sistema de equações para os N pontos escolhidos.

Considere que:

$$P_c = RP_w + T$$

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = R \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + T$$

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$$

$$\begin{cases} X_c = r_{11}X_w + r_{12}Y_w + r_{13}Z_w + T_x \\ Y_c = r_{21}X_w + r_{22}Y_w + r_{23}Z_w + T_y \\ Z_c = r_{31}X_w + r_{32}Y_w + r_{33}Z_w + T_z \end{cases}$$

$$\begin{cases} x_{im} = -\dfrac{f}{s_x}\dfrac{X_c}{Z_c} + o_x \\ y_{im} = -\dfrac{f}{s_y}\dfrac{Y_c}{Z_c} + o_y \end{cases}$$

Os cinco parâmetros intrínsecos f, s_x, s_y, o_x e o_y não são independentes, então definimos os quatro seguintes parâmetros independentes:

$$f_x = \frac{f}{s_x} \quad , \qquad \alpha = \frac{s_y}{s_x} = \frac{f_x}{f_y} \quad , \qquad o_x \quad , \qquad o_y$$

Os parâmetros extrínsecos são a matriz **R** e o vetor **T**.

$$\begin{cases} x_{im} - o_x = -f_x \dfrac{r_{11}X_w + r_{12}Y_w + r_{13}Z_w + T_x}{r_{31}X_w + r_{32}Y_w + r_{33}Z_w + T_z} \\ y_{im} - o_y = -f_y \dfrac{r_{21}X_w + r_{22}Y_w + r_{23}Z_w + T_y}{r_{31}X_w + r_{32}Y_w + r_{33}Z_w + T_z} \end{cases}$$

$$\begin{cases} r_{31}X_w + r_{32}Y_w + r_{33}Z_w + T_z = -f_x \dfrac{r_{11}X_w + r_{12}Y_w + r_{13}Z_w + T_x}{x_{im} - o_x} \\ r_{31}X_w + r_{32}Y_w + r_{33}Z_w + T_z = -f_y \dfrac{r_{21}X_w + r_{22}Y_w + r_{23}Z_w + T_y}{y_{im} - o_y} \end{cases}$$

$$\begin{cases} x_{im} - o_x = x \\ y_{im} - o_y = y \end{cases}$$

$$-f_x \frac{r_{11}X_w + r_{12}Y_w + r_{13}Z_w + T_x}{x} = -f_y \frac{r_{21}X_w + r_{22}Y_w + r_{23}Z_w + T_y}{y}$$

$$yf_x(r_{11}X_w + r_{12}Y_w + r_{13}Z_w + T_x) = xf_y(r_{21}X_w + r_{22}Y_w + r_{23}Z_w + T_y)$$

$$y\frac{f_x}{f_y}(r_{11}X_w + r_{12}Y_w + r_{13}Z_w + T_x) = x(r_{21}X_w + r_{22}Y_w + r_{23}Z_w + T_y)$$

$$y\alpha(r_{11}X_w + r_{12}Y_w + r_{13}Z_w + T_x) = x(r_{21}X_w + r_{22}Y_w + r_{23}Z_w + T_y)$$

$$yX_w\alpha r_{11} + yY_w\alpha r_{12} + yZ_w\alpha r_{13} + y\alpha T_x = xX_w r_{21} + xY_w r_{22} + xZ_w r_{23} + xT_y$$

$$yX_w\alpha r_{11} + yY_w\alpha r_{12} + yZ_w\alpha r_{13} + y\alpha T_x = xX_w r_{21} + xY_w r_{22} + xZ_w r_{23} + xT_y$$

$$xX_w v_1 + xY_w v_2 + xZ_w v_3 + xv_4 - (yX_w v_5 + yY_w v_6 + yZ_w v_7 + yv_8) = 0$$

$$xX_w v_1 + xY_w v_2 + xZ_w v_3 + xv_4 - yX_w v_5 - yY_w v_6 - yZ_w v_7 - yv_8 = 0$$

$$Av = 0$$

$$A_{Nx8} = \begin{bmatrix} x_1X_w & x_1Y_w & x_1Z_w & x_1 & -y_1X_w & -y_1Y_w & -y_1Z_w & -y_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_NX_w & x_NY_w & x_NZ_w & x_N & -y_NX_w & -y_NY_w & -y_NZ_w & -y_N \end{bmatrix}$$

$$v_{8x1} = [v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5 \quad v_6 \quad v_7 \quad v_8]^T$$

$$\begin{cases} v_1 = r_{21} \\ v_2 = r_{22} \\ v_3 = r_{23} \\ v_4 = T_y \\ v_5 = \alpha r_{11} \\ v_6 = \alpha r_{12} \\ v_7 = \alpha r_{13} \\ v_8 = \alpha T_x \end{cases}$$

Calculando a decomposição em valores singulares de A (SVD), a solução do sistema estará na última coluna da matriz V.

$$A_{mxn} = U_{mxn}D_{nxn}V_{nxn}^T$$

6.1.4 Passo 4

Determinar $|\gamma|$ (fator de escala) e α (razão de aspecto).

$$v_{solução} = \bar{v}$$

$$\bar{v} = \gamma v$$

$$(\bar{v}_1, \bar{v}_2, \bar{v}_3, \bar{v}_4, \bar{v}_5, \bar{v}_6, \bar{v}_7, \bar{v}_8) = \gamma\left(r_{21}, r_{22}, r_{23}, T_y, \alpha r_{11}, \alpha r_{12}, \alpha r_{13}, \alpha T_x\right)$$

$$\sqrt{(\bar{v}_1)^2 + (\bar{v}_2)^2 + (\bar{v}_3)^2} = \sqrt{\gamma^2[(r_{21})^2 + (r_{22})^2 + (r_{23})^2]} = |\gamma|$$

$$({r_{21}}^2 + {r_{22}}^2 + {r_{23}}^2 = 1)$$

$$\sqrt{(\bar{v}_5)^2 + (\bar{v}_6)^2 + (\bar{v}_7)^2} = \sqrt{\gamma^2\alpha^2[(r_{11})^2 + (r_{12})^2 + (r_{13})^2]} = \alpha|\gamma|$$

$$({r_{11}}^2 + {r_{12}}^2 + {r_{13}}^2 = 1 \quad e \quad \alpha > 0)$$

6.1.5 Passo 5

Obter as duas primeiras linhas da matriz de rotação **R**, e os dois primeiros componentes do vetor de translação **T**.

$$\begin{cases} r_{21} = \bar{v}_1/|\gamma| \\ r_{22} = \bar{v}_2/|\gamma| \\ r_{23} = \bar{v}_3/|\gamma| \\ T_y = \bar{v}_4/|\gamma| \\ r_{11} = \bar{v}_5/(\alpha|\gamma|) \\ r_{12} = \bar{v}_6/(\alpha|\gamma|) \\ r_{13} = \bar{v}_7/(\alpha|\gamma|) \\ T_y = \bar{v}_8/(\alpha|\gamma|) \end{cases}$$

6.1.6 Passo 6

Obter a terceira linha da matriz de rotação **R** como o produto vetorial da estimativa das duas primeiras linhas obtida no passo anterior:

$$R_1 = \begin{bmatrix} r_{11} \\ r_{12} \\ r_{13} \end{bmatrix} \quad , \quad R_2 = \begin{bmatrix} r_{21} \\ r_{22} \\ r_{23} \end{bmatrix} \quad \Rightarrow \quad R_3 = R_1 \, x \, R_2 = \begin{bmatrix} r_{31} \\ r_{32} \\ r_{33} \end{bmatrix}$$

Os sinais dos elementos de R_3 não precisam ser alterados (eles permanecem fixos se os sinais de todos os elementos de R_1 e R_2 forem invertidos).

Forçar a restrição de ortogonalidade na estimativa de **R** através da decomposição em valores singulares (SVD): o cálculo de **R** não leva em conta explicitamente a restrição de ortogonalidade, portanto, não se pode esperar que a estimativa $\bar{R}$ de **R** seja ortogonal, isto é:

$$\bar{R}\bar{R}^T = I$$

Pode-se forçar a ortogonalidade em $\bar{R}$ usando seu SVD:

$$\bar{R} = UDV^T$$

Deve-se Substituir D por I (matriz identidade):

$$\bar{R}' = UIV^T \Rightarrow \bar{R}'\bar{R}'^T = I$$

6.1.7 Passo 7

Determinar o sinal do fator de escala γ.

Considere novamente o seguinte conjunto de equações:

$$\begin{cases} x = -\dfrac{f}{s_x}\dfrac{r_{11}X_w + r_{12}Y_w + r_{13}Z_w + T_x}{r_{31}X_w + r_{32}Y_w + r_{33}Z_w + T_z} = -\dfrac{f}{s_x}\dfrac{X_c}{Z_c} \\ y = -\dfrac{f}{s_y}\dfrac{r_{21}X_w + r_{22}Y_w + r_{23}Z_w + T_y}{r_{31}X_w + r_{32}Y_w + r_{33}Z_w + T_z} = -\dfrac{f}{s_y}\dfrac{Y_c}{Z_c} \end{cases}$$

Para cada ponto, se Zc >0, então x e $r_{11}X_w + r_{12}Y_w + r_{13}Z_w + T_x$ devem ter sinais opostos. Deve-se escolher um ponto para o qual x seja diferente de zero (é suficiente observar os sinais para apenas um dos pontos). Logo, se

$$x(r_{11}X_w + r_{12}Y_w + r_{13}Z_w + T_x) > 0$$

Os sinais das duas primeiras linhas de R e dos dois primeiros elementos de T devem ser invertidos.

6.1.8 Passo 8

Determinar T_z , f_x e f_y. Tanto T_z como f_x podem ser obtidos pelo método dos mínimos quadrados através da seguinte equação, escrita para N pontos:

$$x = -f_x\frac{r_{11}X_w + r_{12}Y_w + r_{13}Z_w + T_x}{r_{31}X_w + r_{32}Y_w + r_{33}Z_w + T_z}$$

$$x(r_{31}X_w + r_{32}Y_w + r_{33}Z_w + T_z) = -f_x(r_{11}X_w + r_{12}Y_w + r_{13}Z_w + T_x)$$

$$xT_z + (r_{11}X_w + r_{12}Y_w + r_{13}Z_w + T_x)f_x = -x(r_{31}X_w + r_{32}Y_w + r_{33}Z_w)$$

$$A\begin{bmatrix} T_z \\ f_x \end{bmatrix} = b$$

$$A_{Nx2} = \begin{bmatrix} x_1 & \left(r_{11}X_{w_1} + r_{12}Y_{w_1} + r_{13}Z_{w_1} + T_x\right) \\ \vdots & \vdots \\ x_N & \left(r_{11}X_{w_N} + r_{12}Y_{w_N} + r_{13}Z_{w_N} + T_x\right) \end{bmatrix}$$

$$b_{Nx1} = \begin{bmatrix} -x_1\left(r_{31}X_{w_1} + r_{32}Y_{w_1} + r_{33}Z_{w_1}\right) \\ \vdots \\ -x_N\left(r_{31}X_{w_N} + r_{32}Y_{w_N} + r_{33}Z_{w_N}\right) \end{bmatrix}$$

Usando SVD, a solução dos mínimos quadrados é:

$$\begin{bmatrix} T_z \\ f_x \end{bmatrix} = (A^T A)^{-1} A^T b = A^+ b$$

$$A = UDV^T \quad \Rightarrow \quad A^+ = VD^{-1}U^T$$

Para determinar f_y , temos que:

$$\alpha = \frac{s_y}{s_x} \ , \qquad f_x = \frac{f}{s_x} \ , \qquad f_y = \frac{f}{s_y} \quad \Rightarrow \quad f_y = \frac{f_x}{\alpha}$$

6.2 Etapa 2

6.2.1 Passo 1

Encontrar os três pontos de fuga p_1, p_2 e p_3, determinado pelos três conjunto de linhas paralelas.

6.2.2 Passo 2

Obter o ortocentro, O, do triângulo $p_1p_2p_3$, que corresponde ao centro da imagem em coordenadas de imagem.

6.3 Resultados

Os parâmetros intrínsecos e extrínsecos encontrados foram:

a) Supondo $\begin{cases} o_x = 0 \\ o_y = 0 \end{cases}$

$f_x = 867.875183$

$f_y = 886.334875$

$\alpha = 0.979173$

$\gamma = 0.062575$ (não é parâmetro intrínseco/extrínseco)

$$T = \begin{bmatrix} -9.2248 \\ -13.1089 \\ 25.2253 \end{bmatrix} = 29.8874 \begin{bmatrix} -0.3087 \\ -0.4386 \\ 0.8440 \end{bmatrix}$$

$$R = \begin{bmatrix} -0.8772 & 0.4410 & 0.1901 \\ 0.2504 & 0.0822 & 0.9647 \\ 0.4098 & 0.8937 & -0.1825 \end{bmatrix}$$

b) Supondo $\begin{cases} o_x = 319.5 \\ o_y = 239.5 \end{cases}$

$f_x = 451.540527$

$f_y = 485.314474$

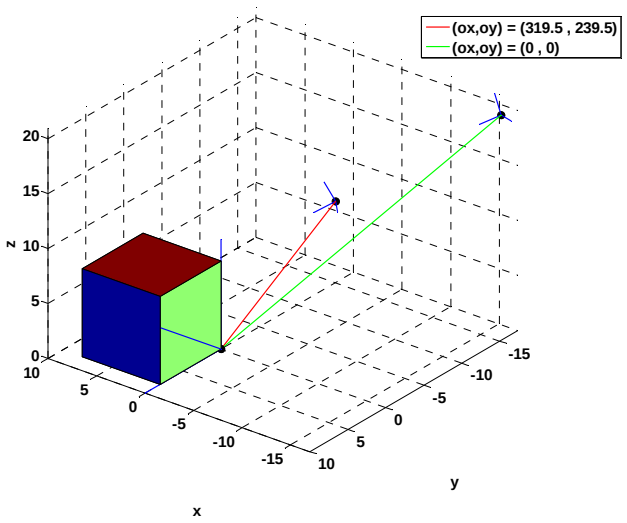
$\alpha = 0.930408$

$\gamma = 0.145841$ (não é parâmetro intrínseco/extrínseco)

$$T = \begin{bmatrix} 0.4770 \\ -6.7047 \\ 14.2621 \end{bmatrix} = 15.7666 \begin{bmatrix} 0.0303 \\ -0.4252 \\ 0.9046 \end{bmatrix}$$

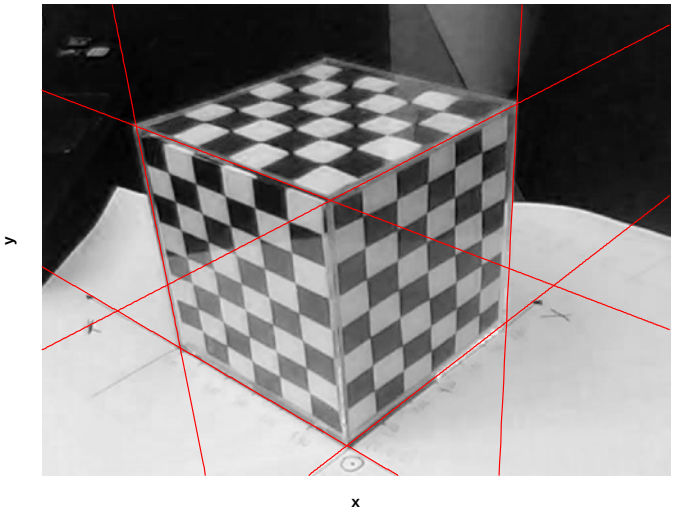
$$R = \begin{bmatrix} -0.6654 & 0.7459 & 0.0310 \\ 0.3735 & 0.2967 & 0.8789 \\ 0.6464 & 0.5964 & -0.4760 \end{bmatrix}$$

A figura a seguir ilustra a disposição do objeto e a câmera segundo os parâmetros extrínsecos calculados.

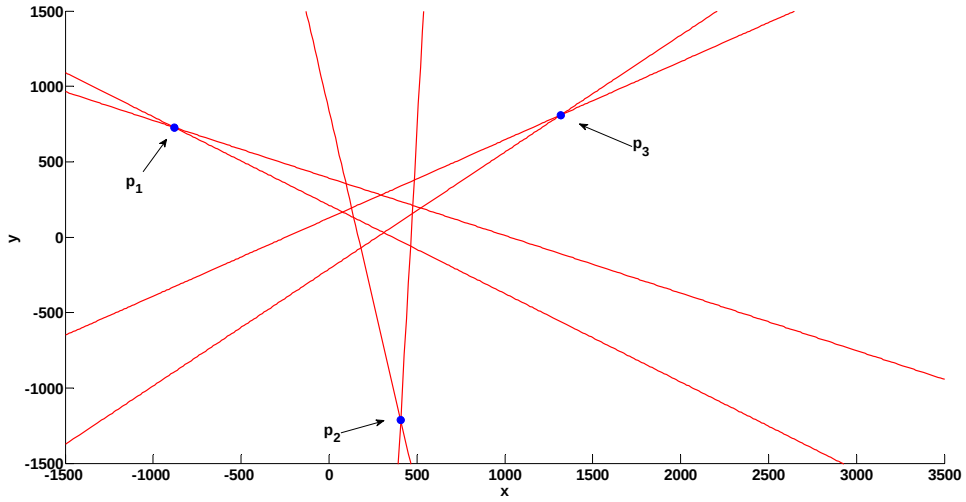


A fim de estimar o centro da imagem, foram determinados os pontos de fuga p1,p2 e p3, e a seguir o ortocentro do triângulo formado por eles (dado os três pontos, o

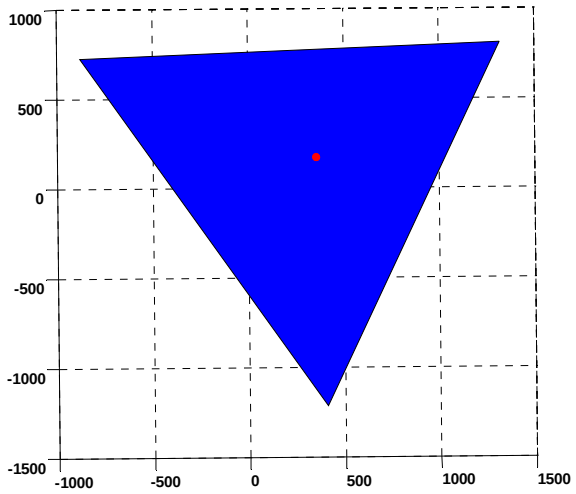
ortocentro foi determinado pelo código de Gustavo Morales, University of Carabobo, Venezuela). A figura a seguir ilustra as linhas paralelas.



A figura a seguir ilustra os pontos de fuga.



Dessa forma pode-se então calcular o ortocentro, conforme mostrado na figura abaixo.



Foi rodado 10 vezes, haja visto a imprecisão decorrente do fato de que os pixels (pontos) das retas paralelas são obtidos manualmente. A média resultou em:

$o_x = 327.5698$

$o_y = 192.4249$

7 MÉTODO DA MATRIZ DE PROJEÇÃO (TRUCCO)

Esse algoritmo de calibração consiste em duas etapas:

Etapa 1: estimar a matriz de projeção que liga as coordenadas de mundo às coordenadas de imagem.

Etapa 2: calcular os parâmetros intrínsecos e extrínsecos através dos elementos da matriz de projeção.

7.1 Etapa 1

7.1.1 Passo 1

Obter as coordenadas de mundo dos pontos escolhidos para a calibração.

7.1.2 Passo 2

Obter as coordenadas de imagem dos pontos escolhidos para a calibração.

7.1.3 Passo 3

Estabelecer a correspondência entre os pontos de imagem e de mundo e resolver o seguinte sistema de equações para os N pontos escolhidos.

Considere que:

$$P_c = RP_w + T$$

$$\begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} = M \begin{bmatrix} X_{w_i} \\ Y_{w_i} \\ Z_{w_i} \\ 1 \end{bmatrix} + T$$

$$\begin{cases} x_i = \dfrac{u_i}{w_i} = \dfrac{m_{11}X_w + m_{12}Y_w + m_{13}Z_w + m_{14}}{m_{31}X_w + m_{32}Y_w + m_{33}Z_w + m_{34}} \\ y_i = \dfrac{v_i}{w_i} = \dfrac{m_{21}X_w + m_{22}Y_w + m_{23}Z_w + m_{24}}{m_{31}X_w + m_{32}Y_w + m_{33}Z_w + m_{34}} \end{cases}$$

$$Am = 0$$

$$A_{Nx12} = \begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 & -x_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 & -y_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_N & Y_N & Z_N & 1 & 0 & 0 & 0 & 0 & -x_NX_N & -x_NY_N & -x_NZ_N & -x_N \\ 0 & 0 & 0 & 0 & X_N & Y_N & Z_N & 1 & -y_NX_N & -y_NY_N & -y_NZ_N & -y_N \end{bmatrix}$$

$$m_{12x1} = [m_{11} \quad m_{12} \quad m_{13} \quad m_{14} \quad m_{21} \quad m_{22} \quad +m_{23} \quad m_{24} \quad m_{31} \quad m_{32} \quad +m_{33} \quad m_{34}]^T$$

Calculando a decomposição em valores singulares de A (SVD), a solução do sistema estará na última coluna da matriz V.

$$A_{mxn} = U_{mxn}D_{nxn}V_{nxn}^T$$

7.2 Etapa 2

7.2.1 Passo 1

Os quatro parâmetros intrínsecos são f_x, f_y, o_x e o_y . Os parâmetros extrínsecos são a matriz **R** e o vetor **T**. Determinar $|\gamma|$ (fator de escala).

$$m_{solução} = \bar{m}$$

$$\bar{m} = \gamma m$$

$$\sqrt{(\bar{m}_{31})^2 + (\bar{m}_{32})^2 + (\bar{m}_{33})^2} = \sqrt{\gamma^2[(r_{31})^2 + (r_{32})^2 + (r_{33})^2]} = |\gamma|$$

$$({r_{31}}^2 + {r_{32}}^2 + {r_{33}}^2 = 1)$$

Deve-se dividir todos os elementos da matriz M por γ.

7.2.2 Passo 2

Determinar os parâmetros intrínsecos. Definindo os vetores:

$$q_1 = [m_{11} \quad m_{12} \quad m_{13}]^T$$
$$q_2 = [m_{21} \quad m_{22} \quad m_{23}]^T$$
$$q_3 = [m_{31} \quad m_{32} \quad m_{33}]^T$$
$$q_4 = [m_{14} \quad m_{24} \quad m_{34}]^T$$

$$o_x = q_1^T q_3$$
$$o_y = q_2^T q_3$$
$$f_x = \sqrt{q_1^T q_1 - o_x^2}$$
$$f_y = \sqrt{q_2^T q_2 - o_y^2}$$

7.2.3 Passo 3

Determinar os parâmetros extrínsecos ($\sigma = \pm 1$).

$$T_z = \sigma m_{34}$$
$$r_{3i} = \sigma m_{3i} \qquad i = 1,2,3$$
$$r_{1i} = \frac{\sigma(o_x m_{3i} - m_{1i})}{f_x} \qquad i = 1,2,3$$
$$r_{2i} = \frac{\sigma\left(o_y m_{3i} - m_{2i}\right)}{f_y} \qquad i = 1,2,3$$
$$T_x = \frac{\sigma(o_x T_z - m_{14})}{f_x}$$
$$T_y = \frac{\sigma\left(o_y T_z - m_{24}\right)}{f_y}$$

7.2.4 Passo 4

Determinar o sinal de σ. Sabe-se se a origem do frame de mundo está na frente (Tz>0) ou atrás (Tz<0) da câmera. Então facilmente pode-se obter o valor de σ.

7.3 Resultados

Os parâmetros intrínsecos e extrínsecos encontrados foram:

fx = 980.653809

fy = 1008.564575

ox = 316.568634

oy = 462.037476

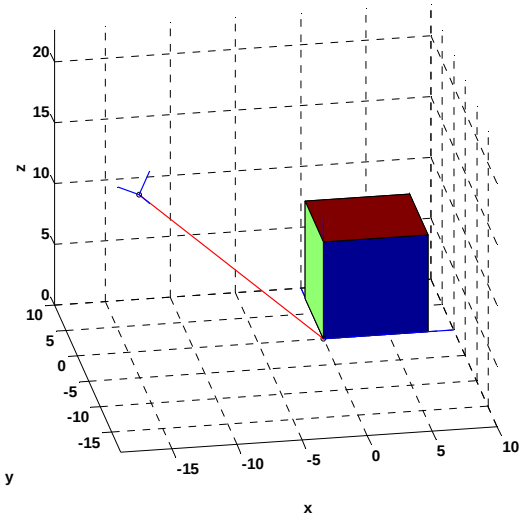
γ = 0.000055 (não é parâmetro intrínseco/extrínseco)

σ = 1.000000 (não é parâmetro intrínseco/extrínseco)

$$T = \begin{bmatrix} 0.202311 \\ 0.501168 \\ 33.347870 \end{bmatrix} = 33.3522 \begin{bmatrix} 0.0061 \\ 0.0150 \\ 0.9999 \end{bmatrix}$$

$$R = \begin{bmatrix} -0.689355 & 0.721431 & 0.065786 \\ 0.501561 & 0.409783 & 0.761915 \\ 0.522711 & 0.558226 & -0.644327 \end{bmatrix}$$

A figura a seguir ilustra a disposição do objeto e a câmera segundo os parâmetros extrínsecos calculados.



8 COMPARAÇÃO ENTRE OS MÉTODOS

A tabela a seguir resume os resultados encontrados para os métodos de calibração no escopo deste trabalho. Os métodos designados de Trucco 1 e 2 se referem ao método direto e ao método da matriz de projeção, respectivamente.

Calibração de Câmera		Parâmetros Intrínsecos				Parâmetros Extrínsecos	
		f_x	f_y	o_x	o_y	R	T
Métodos	**Tsai**	988.66	981.93	335.32	248.17	$\begin{bmatrix} 0.683 & -0.728 & -0.057 \\ -0.390 & -0.297 & -0.872 \\ 0.618 & 0.617 & -0.487 \end{bmatrix}$	$\begin{bmatrix} -0.805 \\ 6.362 \\ 31.547 \end{bmatrix}$
	Church	-	-	319.00	239.00	$\begin{bmatrix} 0{,}002 & -0{,}999 & -0{,}430 \\ -0.002 & -0{,}009 & -0{,}903 \\ 0{,}8987 & 0{,}002 & -0{,}001 \end{bmatrix}$	$\begin{bmatrix} 0{,}679 \\ 0{,}733 \\ 0.001 \end{bmatrix}$
	Trucco 1	867.87	886.33	327.56	192.42	$\begin{bmatrix} -0.877 & 0.441 & 0.190 \\ 0.250 & 0.082 & 0.964 \\ 0.409 & 0.893 & -0.182 \end{bmatrix}$	$\begin{bmatrix} -9.2248 \\ -13.1089 \\ 25.2253 \end{bmatrix}$
	Trucco 2	980.65	1008.56	316.56	462.03	$\begin{bmatrix} -0.689 & 0.721 & 0.066 \\ 0.501 & 0.409 & 0.761 \\ 0.521 & 0.558 & -0.644 \end{bmatrix}$	$\begin{bmatrix} 0.202 \\ 0.501 \\ 33.347 \end{bmatrix}$

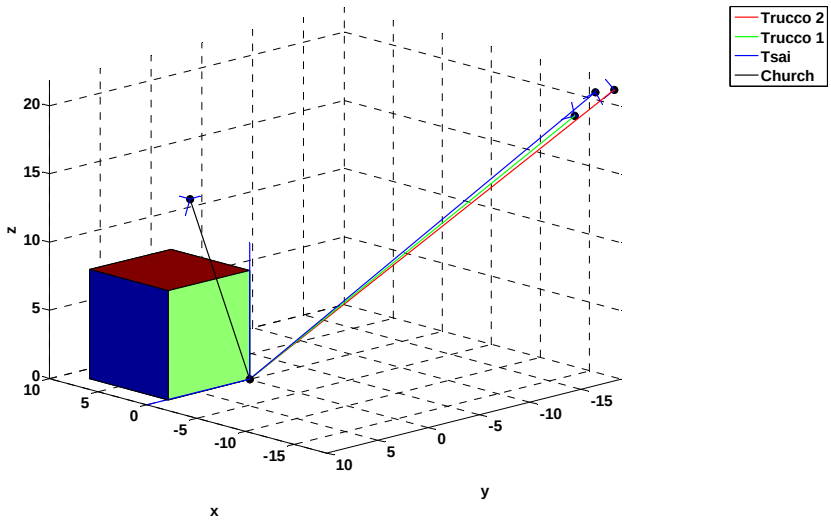
Pode-se observar que os parâmetros de calibração intrínsecos foram obtidos pelos métodos Tsai , Trucco 1 e Trucco2 e convergiram para valores próximos. Os parâmetros extrínsecos foram obtidos em todos os métodos acima, sendo que os obtidos pelos métodos Tsai, Trucco1 e Trucco2 convergiram para a mesma faixa de valores, enquanto que os obtidos pelo Church e Ganapathy não.

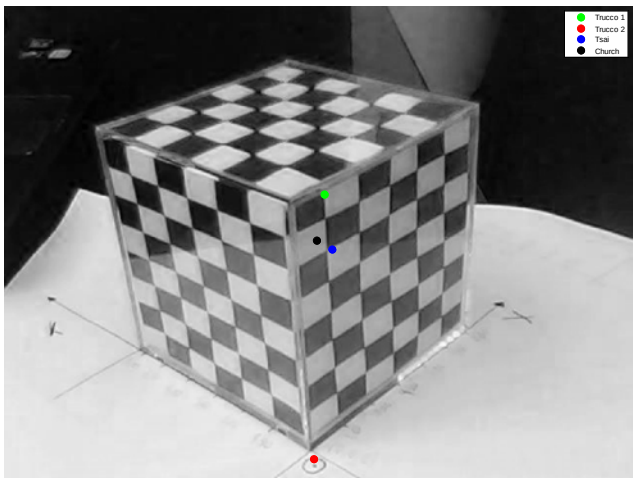
É impotante comentar que a precisão de calibração depende da precisão da correspondência entre os pontos do mundo e da imagem.

Embora os dois métodos Trucco 1 e Trucco 2 devessem produzir os mesmos resultados (teoricamente), foram obtidas soluções diferentes devido aos

diferentes erros de propagação. Além disso, o método da matriz de projeção é bem mais simples de ser implementado.

As figuras a seguir ilustram as informações da tabela. A primeira mostra a disposição do objeto e a câmera segundo os parâmetros extrínsecos calculados pelos diversos métodos, enquanto a segunda imagem mostra o centro da imagem calculado.





9 REFERÊNCIAS BIBLIOGRÁFICAS

[1] Laganière , Robert. **OpenCV 2 Computer Vision Application Programming Cookbook,** ed Packt Publishing. 2011.

[2] Bradski, Gary; Kaehler , Adrian. **Learning OpenCV,** ed O'Reilly. 2008.

[3] Gonçalves, Luiz Marcos Garcia. **Formação de Imagem - Aquisição – Programa de Pós Graduação em Engenharia Elétrica e de Computação – UFRN –** Notas de Aula [on line]. 2011. Disponível em: < www.dca.ufrn.br/~lmarcos/courses/visao > Acesso em: 20 set. 2011.

[4] Emanuele, Trucco; Verri, Alessandro. **Introductory Techniques for 3-D Computer Vision**, Prentice Hall, 1998.

[5] Bebis, George. **CS491E/791E: Computer Vision (Spring 2004) – University of Nevada, Reno –** Notas de Aula [on line]. 2004. Disponível em: < http://www.cse.unr.edu/~bebis/CS791E/ > Acesso em: 24 Out. 2011.

10 ANEXO 01: CÓDIGO MATLAB

10.1 Contents

10.2 Inicialização

```
clear all;
close all;
clc;
imagem = imread('cubo4.jpg');
```

10.3 Obter Pontos da Imagem (Matlab)

```
sw = input('carregar [0] ou obter pontos [1]?');
switch sw
    case 0
        load face1.mat;
        load face2.mat;
        load face3.mat;
        faces = [face1 ; face2 ; face3];
        save faces.mat faces
    case 1
        n = 25; %número de pontos
        imshow(imagem)
        [X,Y] = GINPUT(n);
        face1 = [X Y];
        [X,Y] = GINPUT(n);
        face2 = [X Y];
        [X,Y] = GINPUT(n);
        face3 = [X Y];
        faces = [face1 ; face2 ; face3];
        save faces.mat faces
end

figure
imshow(imagem)
hold on
plot(faces(:,1),faces(:,2),'r*')
```

10.4 Parâmetros extrínsecos (Método Direto - Trucco)

```
close all
clear all
clc

load R_trucco_11.txt
load T_trucco_11.txt
R = R_trucco_11';
T = T_trucco_11;
T = T';

figure;
% subplot(2,2,1)
plot3(0,0,0,'ko')
```

```
grid on
box off
hold all
k1 = 10;
line([0 0],[0 10],[0 0])
line([10 0],[0 0],[0 0])
line([0 0],[0 0],[0 10])

k = 2.0;
e1 = k*[1 0 0]';
e2 = k*[0 1 0]';
e3 = k*[0 0 1]';
Pw = R\([0 0 0]' - T);
plot3(Pw(1),Pw(2),Pw(3),'ko')
h = line([0 Pw(1)],[0 Pw(2)],[0 Pw(3)]);
set(h,'Color','r');
cam = R\(e1 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])
cam = R\(e2 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])
cam = R\(e3 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])

load R_trucco_10.txt
load T_trucco_10.txt
R = R_trucco_10';
T = T_trucco_10;
T = T';

k = 2.0;
e1 = k*[1 0 0]';
e2 = k*[0 1 0]';
e3 = k*[0 0 1]';
Pw = R\([0 0 0]' - T);
plot3(Pw(1),Pw(2),Pw(3),'ko')
h = line([0 Pw(1)],[0 Pw(2)],[0 Pw(3)]);
set(h,'Color','g');
cam = R\(e1 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])
cam = R\(e2 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])
cam = R\(e3 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])


xc=4; yc=4; zc=4;    % coordinated of the center
L=8;                 % cube size (length of an edge)
alpha=0.8;           % transparency (max=1=opaque)

X = [0 0 0 0 0 1; 1 0 1 1 1 1; 1 0 1 1 1 1; 0 0 0 0 0 1];
Y = [0 0 0 0 1 0; 0 1 0 0 1 1; 0 1 1 1 1 1; 0 0 1 1 1 0];
Z = [0 0 1 0 0 0; 0 0 1 0 0 0; 1 1 1 0 1 1; 1 1 1 0 1 1];

% C='blue';                                                 %
unicolor
C= [0.1 0.5 0.9 0.9 0.1 0.5];                               %
color/face
% C = [0.1 0.8 1.1 1.1 0.1 0.8 ; 0.2 0.9 1.2 1.2 0.2 0.8 ;
%      0.3 0.9 1.3 1.3 0.3 0.9 ; 0.4 0.9 1.4 1.4 0.4 0.9 ];   % color
scale/face
```

```
X = L*(X-0.5) + xc;
Y = L*(Y-0.5) + yc;
Z = L*(Z-0.5) + zc;

fill3(X,Y,Z,C,'FaceAlpha',alpha);    % draw cube
axis equal;

AZ=-20;          % azimuth
EL=25;           % elevation
view(AZ,EL);     % orientation of the axes
```

10.5 Parâmetros intrínsecos (Método Direto - Trucco)

```
% Estimativa dos Centros (pontos de fuga)

clear all;
close all;
clc;
imagem = imread('cubo4.jpg');
acumulador = [];
for j = 1:1:1
        n = 2; %número de pontos
        imshow(imagem(:,:,1))
        hold on
        for k = 1:1:6
        [X,Y] = GINPUT(n);
        A = [X(1,1) 1; X(2,1) 1];
        b = [Y(1,1) ; Y(2,1)];
        x = A\b;
        a1 = num2str(x(1));
        b1 = num2str(abs(x(2)));
        if (x(2)>0)
        reta = strcat(a1,'*x','+',b1);
        else
        reta = strcat(a1,'*x','-',b1);
        end

        ezplot(reta,[0 640 0 480]);

        clear A b x
        A = [X(1,1) 1; X(2,1) 1];
        b = [480-Y(1,1) ; 480-Y(2,1)];
        x = A\b;
        a1 = num2str(x(1));
        b1 = num2str(abs(x(2)));
        if (x(2)>0)
        reta = strcat(a1,'*x','+',b1);
        else
        reta = strcat(a1,'*x','-',b1);
        end
        mat_a(k) = x(1);
        mat_b(k) = x(2);
        switch k
            case 1
                reta1 = reta;
            case 2
                reta2 = reta;
            case 3
                reta3 = reta;
            case 4
                reta4 = reta;
            case 5
```

```
                reta5 = reta;
            case 6
                reta6 = reta;
        end
        end

j = 1;
for i=1:2:5
    mat_b2(j) = mat_b(i) - mat_b(i+1);
    mat_a2(j) = mat_a(i) - mat_a(i+1);
    j = j+1;
end

syms x
x1 = double(solve(mat_a2(1)*x + mat_b2(1),'x'));
y1 = mat_a(1)*x1 + mat_b(1);
x2 = double(solve(mat_a2(2)*x + mat_b2(2),'x'));
y2 = mat_a(3)*x2 + mat_b(3);
x3 = double(solve(mat_a2(3)*x + mat_b2(3),'x'));
y3 = mat_a(5)*x3 + mat_b(5);

figure;
[pos] = center([x1 y1 0] ,[x2 y2 0] ,[x3 y3 0],'orthocenter');

figure;
hold all
xmin = -1500;
xmax = 3500;
ymin = -1500;
ymax = 1500;
ezplot(reta1,[xmin xmax ymin ymax]);
ezplot(reta2,[xmin xmax ymin ymax]);
ezplot(reta3,[xmin xmax ymin ymax]);
ezplot(reta4,[xmin xmax ymin ymax]);
ezplot(reta5,[xmin xmax ymin ymax]);
ezplot(reta6,[xmin xmax ymin ymax]);
plot(x1,y1,'ko')
plot(x2,y2,'ko')
plot(x3,y3,'ko')
close all
acumulador = [acumulador pos];
end
mean(acumulador')
```

10.6 Parâmetros extrínsecos (Método da matriz de projeção - Trucco)

```
close all
clear all
clc

load R_trucco_2.txt
load T_trucco_2.txt
R = R_trucco_2;
T = T_trucco_2;
T = T';

figure;
% subplot(2,2,1)
plot3(0,0,0,'ko')
grid on
box off
hold all
```

```
k1 = 10;
line([0 0],[0 10],[0 0])
line([10 0],[0 0],[0 0])
line([0 0],[0 0],[0 10])

k = 2.0;
e1 = k*[1 0 0]';
e2 = k*[0 1 0]';
e3 = k*[0 0 1]';
Pw = R\([0 0 0]' - T);
plot3(Pw(1),Pw(2),Pw(3),'ko')
h = line([0 Pw(1)],[0 Pw(2)],[0 Pw(3)]);
set(h,'Color','r');
cam = R\(e1 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])
cam = R\(e2 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])
cam = R\(e3 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])

xc=4; yc=4; zc=4;    % coordinated of the center
L=8;                 % cube size (length of an edge)
alpha=0.8;           % transparency (max=1=opaque)

X = [0 0 0 0 0 1; 1 0 1 1 1 1; 1 0 1 1 1 1; 0 0 0 0 0 1];
Y = [0 0 0 0 1 0; 0 1 0 0 1 1; 0 1 1 1 1 1; 0 0 1 1 1 0];
Z = [0 0 1 0 0 0; 0 0 1 0 0 0; 1 1 1 0 1 1; 1 1 1 0 1 1];

% C='blue';                                                    %
unicolor
C= [0.1 0.5 0.9 0.9 0.1 0.5];                                  %
color/face
% C = [0.1 0.8 1.1 1.1 0.1 0.8 ; 0.2 0.9 1.2 1.2 0.2 0.8 ;
%      0.3 0.9 1.3 1.3 0.3 0.9 ; 0.4 0.9 1.4 1.4 0.4 0.9 ];   % color
scale/face

X = L*(X-0.5) + xc;
Y = L*(Y-0.5) + yc;
Z = L*(Z-0.5) + zc;

fill3(X,Y,Z,C,'FaceAlpha',alpha);    % draw cube
axis equal;

AZ=-20;         % azimuth
EL=25;          % elevation
view(AZ,EL);    % orientation of the axes
```

10.7 Todos os métodos

```
figure;

close all
clear all
clc

load R_trucco_2.txt
load T_trucco_2.txt
R = R_trucco_2;
T = T_trucco_2;
T = T';

plot3(0,0,0,'ko')
```

```
grid on
box off
hold all
line([0 0],[0 10],[0 0])
line([10 0],[0 0],[0 0])
line([0 0],[0 0],[0 10])

k = 1;
e1 = k*[1 0 0]';
e2 = k*[0 1 0]';
e3 = k*[0 0 1]';
Pw = R\([0 0 0]' - T);
plot3(Pw(1),Pw(2),Pw(3),'ko')
h = line([0 Pw(1)],[0 Pw(2)],[0 Pw(3)]);
set(h,'Color','r');
cam = R\(e1 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])
cam = R\(e2 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])
cam = R\(e3 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])

clear Pw R T cam

load R_trucco_10.txt
load T_trucco_10.txt
R = R_trucco_10';
T = T_trucco_10;
T = T';

Pw = R\([0 0 0]' - T);
plot3(Pw(1),Pw(2),Pw(3),'ko')
h = line([0 Pw(1)],[0 Pw(2)],[0 Pw(3)]);
set(h,'Color','g');
cam = R\(e1 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])
cam = R\(e2 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])
cam = R\(e3 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])


clear Pw R T cam
R = [0.68270785 -0.72843552 -0.05737369; -0.39022160 -0.29708555 -
0.87147421; 0.61776787 0.61735070 -0.48707390];
T = [-0.80445695 6.36146545 31.54652214]';

Pw = R\([0 0 0]' - T);
plot3(Pw(1),Pw(2),Pw(3),'ko')
h = line([0 Pw(1)],[0 Pw(2)],[0 Pw(3)]);
set(h,'Color','b');
cam = R\(e1 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])
cam = R\(e2 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])
cam = R\(e3 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])

clear Pw R T cam

load R.txt
```

```
load T.txt

Pw = R\([0 0 0]' - T);
plot3(Pw(1),Pw(2),Pw(3),'ko')
h = line([0 Pw(1)],[0 Pw(2)],[0 Pw(3)]);
set(h,'Color','k');
cam = R\(e1 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])
cam = R\(e2 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])
cam = R\(e3 - T);
line([Pw(1) cam(1)],[Pw(2) cam(2)],[Pw(3) cam(3)])


xc=4; yc=4; zc=4;    % coordinated of the center
L=8;                 % cube size (length of an edge)
alpha=0.8;           % transparency (max=1=opaque)

X = [0 0 0 0 0 1; 1 0 1 1 1 1; 1 0 1 1 1 1; 0 0 0 0 0 1];
Y = [0 0 0 0 1 0; 0 1 0 0 1 1; 0 1 1 1 1 1; 0 0 1 1 1 0];
Z = [0 0 1 0 0 0; 0 0 1 0 0 0; 1 1 1 0 1 1; 1 1 1 0 1 1];

% C='blue';                                                 %
unicolor
C= [0.1 0.5 0.9 0.9 0.1 0.5];                               %
color/face
% C = [0.1 0.8 1.1 1.1 0.1 0.8 ; 0.2 0.9 1.2 1.2 0.2 0.8 ;
%      0.3 0.9 1.3 1.3 0.3 0.9 ; 0.4 0.9 1.4 1.4 0.4 0.9 ];   % color
scale/face

X = L*(X-0.5) + xc;
Y = L*(Y-0.5) + yc;
Z = L*(Z-0.5) + zc;

fill3(X,Y,Z,C,'FaceAlpha',alpha);    % draw cube
axis equal;

AZ=-20;          % azimuth
EL=25;           % elevation
view(AZ,EL);     % orientation of the axes
```

10.8 Centros

```
clear all
close all
clc

imagem = imread('cubo4.jpg');

figure;
imshow(imagem(:,:,1));
hold on

% Método Trucco 1
ox = 327.5698;
oy = 192.4249;
plot(ox,oy,'go')

% Método Trucco 2
ox = 316.568634;
oy = 462.037476;
plot(ox,oy,'ro')
```

```
% Método Tsai
ox = 335.32232666;
oy = 248.17175293;
plot(ox,oy,'bo')

% Método Church
ox = 319.5;
oy = 239.5;
plot(ox,oy,'ko')
```

```
function [pos] = center(Pa,Pb,Pc,Type)
% CENTER calculates and shows the orthocenter, circumcenter,
barycenter and
% incenter of a triangle, given their vertex's coordinates Pa, Pb and
Pc
%
% Example: center([0 0.5 0], [1 0 0], [1 1 3], 'orthocenter')
%
% Made by: Ing. Gustavo Morales, University of Carabobo, Venezuela.
% 09/14/09
%
Pa = Pa(:); Pb = Pb(:); Pc = Pc(:); % Converting to column vectors (if
needed)
AB = Pb - Pa; AC = Pc - Pa; BC = Pc - Pb; % Side vectors
switch Type
    case 'incenter'%
        uab = AB./norm(AB); uac = AC./norm(AC); ubc = BC./norm(BC);
uba = -uab;
        L1 = uab + uac; L2 = uba + ubc; % directors
        P21 = Pb - Pa;
        P1 = Pa;
    case 'barycenter'
        L1 = (Pb + Pc)/2 -Pa; L2 = (Pa + Pc)/2 - Pb; % directors
        P21 = Pb - Pa;
        P1 = Pa;
    case 'circumcenter'
        N = cross(AC,AB);
        L1 = cross(AB,N); L2 = cross(BC,N); % directors
        P21 = (Pc - Pa)/2;
        P1 = (Pa + Pb)/2;
    case 'orthocenter'
        N = cross(AC,AB);
        L1 = cross(N,BC); L2 = cross(AC,N); % directors
        P21 = Pb - Pa;
        P1 = Pa;
    otherwise
        error('Unknown Center Type');
end
ML = [L1 -L2]; % Coefficient Matrix
lambda = ML\P21;  % Solving the linear system
pos = P1 + lambda(1)*L1; % Line Equation evaluated at lambda(1)
X = [Pa(1); Pb(1); Pc(1)]; Y = [Pa(2); Pb(2); Pc(2)];
if numel(Pa)== 3 % Tridimensional Case
    Z = [Pa(3); Pb(3); Pc(3)];
    patch(X,Y,Z,'b','FaceAlpha',0.5); hold on;
    plot3(pos(1),pos(2),pos(3),'o','Color','r'); view(3);
else             % Bidimensional case
    patch(X,Y,'b','FaceAlpha',0.5); hold on;
    plot(pos(1),pos(2),'o','Color','r');
end
 grid on; daspect([1 1 1]);
```

11 ANEXO 02: CÓDIGO C++

a) Método openCV – Tsai

```
//Trabalho2 de Visão computacional
//Grupo: Andre Tavares da Silva e Leonardo Enzo
//Calibracao de camera usando metodo da biblioteca openCV - Tsai
//Data termino 21/10/11

//Instrucoes (após compilar):
// 1)digite na linha de comando ./calibra_ocv num_de_quinas_hor num_de_quinas_vert num_de_vistas
// 2)Tecle enter
// 3)Tecle 'p' se quiser parar/continuar, ESC para sair
//Bibliotecas

#include <cv.h>
#include <highgui.h>
#include <stdio.h>
#include <stdlib.h>

//Inicializacoes

int n_boards = 0;
const int board_dt = 20; //Aguarda 20 frames por vista do xadrez(padrao)
int board_w;//qtd de ptos horizontais
int board_h;//qtd de ptos verticais

int main(int argc, char* argv[]) {

printf("Trabalho2 de Visão computacional\n");
printf("Grupo: Andre Tavares da Silva e Leonardo Enzo\n");
printf("Calibracao de camera usando metodo da biblioteca openCV - Tsai\n");
```

```
if(argc != 4){
printf("ERROR: Wrong number of input parameters\n");
return -1;
}

cvWaitKey(9000);//creditos iniciais....

//Plano usado para calibração: cada quadrado é exatamente 1 x 1 cm

IplImage* img = cvLoadImage("plano_calib.jpg" );
cvNamedWindow( "Imagem para calibracao dos parametros intrinsecos e distorcao", CV_WINDOW_AUTOSIZE );
cvShowImage( "Imagem para calibracao dos parametros intrinsecos e distorcao", img );
cvWaitKey(0);

board_w = atoi(argv[1]);
board_h = atoi(argv[2]);
n_boards = atoi(argv[3]);

int board_n = board_w * board_h;
CvSize board_sz = cvSize( board_w, board_h );

//Inicia captura pela camera

CvCapture* capture = cvCreateCameraCapture(0);
assert( capture );
cvNamedWindow( "Calibration" );

//Matrizes e vetores usados
```

```
CvMat* image_points= cvCreateMat(n_boards*board_n,2,CV_32FC1);//pontos imagem do plano
CvMat* object_points= cvCreateMat(n_boards*board_n,3,CV_32FC1);//pontos objeto do plano
CvMat* point_counts= cvCreateMat(n_boards,1,CV_32SC1);
CvMat* intrinsic_matrix = cvCreateMat(3,3,CV_32FC1);
CvMat* distortion_coeffs = cvCreateMat(5,1,CV_32FC1);
CvMat* rotation_vector = cvCreateMat(1,3,CV_32FC1);//vetor eixo de rotacao
CvMat* translation_vector = cvCreateMat(1,3,CV_32FC1);
CvPoint2D32f* corners = new CvPoint2D32f[ board_n ];
CvMat *obj = (CvMat*)cvLoad("pts_obj.xml");//pontos objeto do cubo
CvMat *imgg = (CvMat*)cvLoad("pts_img.xml");//pontos imagem do cubo
CvMat* R = cvCreateMat(3,3,CV_32FC1);//matriz de rotacao = transformacao do vetor de rot

int corner_count;
int successes = 0;
int step, frame = 0;
IplImage *image = cvQueryFrame( capture );
IplImage *gray_image = cvCreateImage(cvGetSize(image),8,1);//subpixel
// Captura vistas de xadrez até se obter n_boards (n_ quadros)
// capturas com sucesso (todos os corners(quinas) encontrados)
//
while(successes < n_boards) {
//Pula a cada quadro com sucesso board_dt frames para permitir ao usuario mover o xadrez
if(frame++ % board_dt == 0) {
//Acha quinas(corners) do xadrez
int found = cvFindChessboardCorners(
image, board_sz, corners, &corner_count,
```

```
CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS
);
//Pega precisao subpixel nessas quinas encontradas
cvCvtColor(image, gray_image, CV_BGR2GRAY);
cvFindCornerSubPix(gray_image, corners, corner_count,
cvSize(11,11),cvSize(-1,-1), cvTermCriteria(
CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 30, 0.1 ));
//Desenha as quinas na imagem
cvDrawChessboardCorners(image, board_sz, corners,
corner_count, found);
cvShowImage( "Calibration", image );

// Se captura um xadrez bom = todas as quinas encontradas, adiciona ao banco de
dados
if( corner_count == board_n ) {
step = successes*board_n;
for( int i=step, j=0; j<board_n; ++i,++j ) {
CV_MAT_ELEM(*image_points, float,i,0) = corners[j].x;
CV_MAT_ELEM(*image_points, float,i,1) = corners[j].y;
CV_MAT_ELEM(*object_points,float,i,0) = j/board_w;
CV_MAT_ELEM(*object_points,float,i,1) = j%board_w;
CV_MAT_ELEM(*object_points,float,i,2) = 0.0f;

}
CV_MAT_ELEM(*point_counts, int,successes,0) = board_n;
successes++;
}
} //Finaliza pulos board_dt entre capturas

//Lida com parar/continuar e ESC
int c = cvWaitKey(15);
```

```
if(c == 'p'){
c = 0;
while(c != 'p' && c != 27){
c = cvWaitKey(250);
}
}
if(c == 27)
return 0;
image = cvQueryFrame( capture ); //Pega próxima imagem
} //termina loop de coleta

//Organiza matrizes de acordo com quantos xadrezes foram encontrados

CvMat* object_points2 = cvCreateMat(successes*board_n,3,CV_32FC1);
CvMat* image_points2 = cvCreateMat(successes*board_n,2,CV_32FC1);
CvMat* point_counts2 = cvCreateMat(successes,1,CV_32SC1);

//Transfere os valores para as matrizes com tamanho correto

for(int i = 0; i<successes*board_n; ++i) {
CV_MAT_ELEM( *image_points2, float, i, 0) =CV_MAT_ELEM(
*image_points, float, i, 0);
CV_MAT_ELEM( *image_points2, float,i,1) =CV_MAT_ELEM(
*image_points, float, i, 1);
CV_MAT_ELEM(*object_points2, float, i, 0) =CV_MAT_ELEM(
*object_points, float, i, 0) ;
CV_MAT_ELEM( *object_points2, float, i, 1) =CV_MAT_ELEM(
*object_points, float, i, 1) ;
CV_MAT_ELEM( *object_points2, float, i, 2) =CV_MAT_ELEM(
*object_points, float, i, 2) ;
```

```
}
for(int i=0; i<successes; ++i){ //São todos o mesmo número
CV_MAT_ELEM( *point_counts2, int, i, 0) =CV_MAT_ELEM( *point_counts,
int, i, 0);
}

cvReleaseMat(&object_points);
cvReleaseMat(&image_points);
cvReleaseMat(&point_counts);
// Nesse ponto já se tem todos os pontos necessários

//Finalmente, CALIBRACAO DA CAMERA, PARAMETROS INTRINSECOS
E DISTORCAO!
cvCalibrateCamera2(
object_points2, image_points2,
point_counts2, cvGetSize( image ),
intrinsic_matrix, distortion_coeffs,
NULL,NULL);//NULL, NULL se referem aqui a rotacao e translacao,
respectivamente

// Salva os intrinsecos e distorcoes

cvSave("Intrinsics.xml",intrinsic_matrix);
cvSave("Distortion.xml",distortion_coeffs);

//Acha os extrinsecos: rotacao e translacao com base nos intrinsecos

IplImage* img2 = cvLoadImage("cubo4.jpg" );
cvNamedWindow( "Imagem para calibracao dos parametros extrinsecos",
CV_WINDOW_AUTOSIZE );
cvShowImage( "Imagem para calibracao dos parametros extrinsecos", img2 );
```

```
cvWaitKey(0);

//Carrega novamente os parametros intrinsecos e de distorcao, que agora
//serao usados como entrada de dados

CvMat *intrinsic = (CvMat*)cvLoad("Intrinsics.xml");
CvMat *distortion = (CvMat*)cvLoad("Distortion.xml");

//Funcao que acha os parametros extrinsecos: vetor eixo de rotacao e vetor translacao

cvFindExtrinsicCameraParams2(obj, imgg, intrinsic, distortion,
rotation_vector,translation_vector);

//rotation_vector eh o eixo de rotacao e seu modulo eh o angulo
//para achar a matriz R de rotacao usa-se a transf. de Rodrigues

cvRodrigues2(rotation_vector,R, NULL);
// Salva os parametros extrinsecos
cvSave("Rotations.xml",R);
cvSave("Translations.xml",translation_vector);
//Fecha janelas abertas
cvDestroyWindow( "Imagem para calibracao dos parametros extrinsecos");
cvDestroyWindow( "Imagem para calibracao dos parametros intrinsecos e distorcao");

//Fim!!
return 0;
}
```

b) Método Church e Ganapathy

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
// *                  CÓDIGO PARA CALIBRAÇÃO DE CÂMERA

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
// *   DISCIPLINA: EEC1515 -  VISÃO COMPUTACIONAL                    *
// *   VERSÃO : 1.5                                    *
// *   DATA : 23/10/2011                                  *
// *   CAMERA : WEBCAM CLONE                                    *
// *   RESOLUÇÃO: VGA

// *   ESCRITO POR: LEONARDO ENZO / ANDRÉ TAVARES
// *     TRABALHO2: Calibracao de camera usando metodo de Church and
Ganapathy
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
// *                       DESCRIÇÃO GERAL                       *
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//

//Instrucoes (após compilar):
// 1)digite na linha de comando ./calibra_church
// 2)Tecle enter


//Bibliotecas
```

```
#include <cv.h>
#include <highgui.h>
#include <stdio.h>
#include <stdlib.h>


int main() {

printf("Trabalho2 de Visão computacional\n");
printf("Grupo: Andre Tavares da Silva e Leonardo Enzo\n");
printf("Calibracao de camera usando metodo de Church and Ganapathy\n");

cvWaitKey(10);//creditos iniciais....

//Matrizes e vetores usados

cv::Mat img1=cv::imread("cubo4.jpg");
cv::Mat face1 = img1(cv::Rect(80, 120, 240, 340));
CvMat *obj = (CvMat*)cvLoad("pts_obj2.xml");//pontos objeto de 1 face do cubo
CvMat *imgg = (CvMat*)cvLoad("pts_img2.xml");//pontos imagem de 1 face do cubo
CvMat* A = cvCreateMat(25,9,CV_32FC1);
CvMat* W = cvCreateMat(25,9,CV_32FC1);
CvMat* U = cvCreateMat(25,25,CV_32FC1);
CvMat* V = cvCreateMat(9,9,CV_32FC1);

CvMat* P1 = cvCreateMat(9,1,CV_32FC1); //vetor de parametros
```

```
//centro do eixo optico em pixels

int ox;
int oy;
ox = (img1.size().width-1)/2.0;
oy = (img1.size().height-1)/2.0;

printf("ox,oy %d %d\n",ox,oy);

//Cubo usado na calibracao

cv::imshow("cubo4",img1);

//Face do cubo de onde foram retirados os pontos

cv::imshow("Face1",face1);
cvWaitKey(0);

//Construindo matrizes

//Matriz A

//Linhas pares: 0,2,4,6,...22

for( int j=0; j<13;++j) {
int i = 2*j;

//printf("teste %d\n",j);
//printf("teste %d\n",i);

CV_MAT_ELEM(*A, float,i,0) = CV_MAT_ELEM(*obj, float,i,1);
```

```
CV_MAT_ELEM(*A, float,i,1) = CV_MAT_ELEM(*obj, float,i,2);
CV_MAT_ELEM(*A, float,i,2) = 0.0f;
CV_MAT_ELEM(*A, float,i,3) = 0.0f;
CV_MAT_ELEM(*A, float,i,4) = -(CV_MAT_ELEM(*obj, float,i,1)*ox +
CV_MAT_ELEM(*obj, float,i,1)*CV_MAT_ELEM(*imgg, float,i,0));
CV_MAT_ELEM(*A, float,i,5) = -(CV_MAT_ELEM(*obj, float,i,2)*ox +
CV_MAT_ELEM(*obj, float,i,2)*CV_MAT_ELEM(*imgg, float,i,0));
CV_MAT_ELEM(*A, float,i,6) = 1.0f;
CV_MAT_ELEM(*A, float,i,7) = 0.0f;
CV_MAT_ELEM(*A, float,i,8) = -(CV_MAT_ELEM(*imgg, float,i,0)+ox);
}

//Linhas impares: 1,3,5,7,...23
for( int j=0; j<12;++j) {
int k = 2*j+1;

//printf("teste %d\n",j);
//printf("teste %d\n",k);

CV_MAT_ELEM(*A, float,k,0) = 0.0f;
CV_MAT_ELEM(*A, float,k,1) = 0.0f;
CV_MAT_ELEM(*A, float,k,2) = CV_MAT_ELEM(*obj, float,k,1);
CV_MAT_ELEM(*A, float,k,3) = CV_MAT_ELEM(*obj, float,k,2);
CV_MAT_ELEM(*A, float,k,4) = -(CV_MAT_ELEM(*obj, float,k,1)*oy +
CV_MAT_ELEM(*obj, float,k,1)*CV_MAT_ELEM(*imgg, float,k,1));
CV_MAT_ELEM(*A, float,k,5) = -(CV_MAT_ELEM(*obj, float,k,2)*oy +
CV_MAT_ELEM(*obj, float,k,2)*CV_MAT_ELEM(*imgg, float,k,1));
CV_MAT_ELEM(*A, float,k,6) = 0.0f;
CV_MAT_ELEM(*A, float,k,7) = 1.0f;
CV_MAT_ELEM(*A, float,k,8) = -(CV_MAT_ELEM(*imgg, float,k,1)+oy);
}
```

```
FILE *matriz_A;
        matriz_A = fopen("A.txt","w");
        for (int i=0; i<25 ; i++)
        {
        fprintf(matriz_A,"%f %f %f %f %f %f %f %f %f
\n",CV_MAT_ELEM(*A, float,i,0),CV_MAT_ELEM(*A,
float,i,1),CV_MAT_ELEM(*A, float,i,2),CV_MAT_ELEM(*A,
float,i,3),CV_MAT_ELEM(*A, float,i,4),CV_MAT_ELEM(*A,
float,i,5),CV_MAT_ELEM(*A, float,i,6),CV_MAT_ELEM(*A,
float,i,7),CV_MAT_ELEM(*A, float,i,8));
        }
        fclose(matriz_A);

//SVD

cvSVD(A,W,U,V,0);

//cvSave("A.xml",A);
//cvSave("W.xml",W);
//cvSave("U.xml",U);
//cvSave("V.xml",V);


// Montando vetor de parametros (temporario) = ultima coluna de V

for( int j=0; j<9;++j) {

//printf("teste %d\n",j);

CV_MAT_ELEM(*P1, float,j,0) = CV_MAT_ELEM(*V, float,j,8);
```

```
}
cvSave("P1.xml",P1);

float esc = 1.0;

//vetor de translacao

float t1,t2,t3;

t1 = esc*CV_MAT_ELEM(*P1, float,6,0);//T1 ou tx
t2= esc*CV_MAT_ELEM(*P1, float,7,0);//T2 ou ty
t3 = esc*CV_MAT_ELEM(*P1, float,8,0);//T3 ou tz

float T[]= {t1,t2,t3};

FILE *matriz_T;
        matriz_T = fopen("T.txt","w");
        for (int i=0; i<3 ; i++)
        {
        fprintf(matriz_T,"%f \n",T[i]);
        }
        fclose(matriz_T);

//Colunas da matriz de rotacao

float v13,v23,v33,v12,v22,v32;

v13 = CV_MAT_ELEM(*P1, float,1,0);//r13
v23 = CV_MAT_ELEM(*P1, float,3,0);//r23
v33 = CV_MAT_ELEM(*P1, float,5,0);//r33
```

```
v12 = CV_MAT_ELEM(*P1, float,0,0);//r12
v22 = CV_MAT_ELEM(*P1, float,2,0);//r22
v32 = CV_MAT_ELEM(*P1, float,4,0);//r32

//Normalizando vetores

float gama,alpha;
        gama = sqrt(pow(v13,2) + pow(v23,2) + pow(v33,2));
        alpha = sqrt(pow(v12,2) + pow(v22,2) + pow(v32,2));

v13 = v13/gama;//r11_normalizado
v23 = v23/gama;//r21_normalizado
v33 = v33/gama;//r31_normalizado

v12 = v12/alpha;//r12_normalizado
v22 = v22/alpha;//r22_normalizado
v32 = v32/alpha;//r32_normalizado


//produto vetorial

        float va[] = {v12, v22, v32};
        float vb[] = {v13, v23, v33};
        float vc[3];

        CvMat R1 = cvMat(3, 1, CV_32FC1, va);
        CvMat R2 = cvMat(3, 1, CV_32FC1, vb);
        CvMat R3 = cvMat(3, 1, CV_32FC1, vc);

        cvCrossProduct(&R1, &R2, &R3);    // cross product: R2 x R3 -> R1
```

```
FILE *matriz_R;
        matriz_R = fopen("R.txt","w");
        for (int i=0; i<3 ; i++)
        {
        fprintf(matriz_R,"%f %f %f \n",vc[i],va[i],vb[i]);
        }
        fclose(matriz_R);

return 0;
}
```

```
1  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
2  // *                    CÓDIGO PARA CALIBRAÇÃO DE CÂMERA                     *
3  // *                           MÉTODO  TRUCCO 1                              *
4  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
5  //
6  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
7  // *   DISCIPLINA: EEC1515 -  VISÃO COMPUTACIONAL                             *
8  // *   VERSÃO : 1.5                                                           *
9  // *   DATA : 14/10/2011                                                      *
10 // *   CAMERA : WEBCAM CLONE                                                  *
11 // *   RESOLUÇÃO: VGA                                                         *
12 // *   ESCRITO POR: LEONARDO ENZO / ANDRÉ TAVARES                             *
13 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
14
15 //bibliotecas
16 #include <iostream>
17 #include <stdafx.h>
18 #include <cv.h>
19 #include <highgui.h>
20 #include <math.h>
21 #include <stdio.h>
22 #include <conio.h>
23 #include <stdlib.h>
24 #include <string.h>
25
26 int main()
27 {
28
29     // Ler e mostrar a imagem utilizada para calibração
30     cv::Mat image001 = cv::imread("cubo4.jpg");
31     cv::imshow("Imagem para calibração", image001);
32
33     // Teste de validação da imagem lida
34     if (!image001.data) { printf("Falha ao abrir a imagem.\n"); }
35     else { printf("Imagem carregada com sucesso.\n"); }
36
37     // Carregando pontos
38     CvMat* image_points = (CvMat*)cvLoad("pts_img_75.xml");
39     CvMat* object_points = (CvMat*)cvLoad("pts_obj_75.xml");
40     std::cout  << "Pontos da imagem e do mundo carregados com sucesso." << std::endl;
41
42     int N = 75; //número de pontos imagem <-> mundo
43
44     CvMat* A  = cvCreateMat(N,8,CV_32FC1);
45     CvMat* U  = cvCreateMat(N,8,CV_32FC1);
46     CvMat* D  = cvCreateMat(8,8,CV_32FC1);
47     CvMat* V  = cvCreateMat(8,8,CV_32FC1);
48
49     float x,y,ox,oy,a_ij,Xw,Yw,Zw;
50     ox = (image001.size().width-1)/2.0;
51     oy = (image001.size().height-1)/2.0;
52     ox = 0;
53     oy = 0;
54     //ox = 316.568634;
55     //oy = 462.037476;
56     //ox = 351.2970;
57     //oy = 182.0324;
58
59     for (int i=0; i<N; i++)
60     {
61         for (int j=0; j<8; j++)
62         {
63         x = cvmGet(image_points,i,0) - ox;
64         y = cvmGet(image_points,i,1) - oy;
65         Xw = cvmGet(object_points,i,0);
66         Yw = cvmGet(object_points,i,1);
67         Zw = cvmGet(object_points,i,2);
68         switch (j)
69         {
70             case 0:
71                 a_ij = x*Xw;
```

```
72                  break;
73              case 1:
74                  a_ij = x*Yw;
75                  break;
76              case 2:
77                  a_ij = x*Zw;
78                  break;
79              case 3:
80                  a_ij = x;
81                  break;
82              case 4:
83                  a_ij = -y*Xw;
84                  break;
85              case 5:
86                  a_ij = -y*Yw;
87                  break;
88              case 6:
89                  a_ij = -y*Zw;
90                  break;
91              case 7:
92                  a_ij = -y;
93                  break;
94          }
95          cvmSet(A,i,j,a_ij); // carregando valores A(i,j)
96          }
97      }
98
99      std::cout  << "Matriz A carregada com sucesso." << std::endl;
100
101     FILE *matriz_A;
102     matriz_A = fopen("A.txt","w");
103     for (int i=0; i<75 ; i++)
104     {
105     fprintf(matriz_A,"%f %f %f %f %f %f %f %f \n",cvmGet(A,i,0),cvmGet(A,i,1),cvmGet(A,i,2),cvmGet(A↙
        ,i,3),cvmGet(A,i,4),cvmGet(A,i,5),cvmGet(A,i,6),cvmGet(A,i,7));
106     }
107     fclose(matriz_A);
108
109     cvSVD(A, D, U, V,0); // A = U D V^T
110     //flags
111     //CV_SVD_MODIFY_A Allows modification of matrix A
112     //CV_SVD_U_T Return U^T instead of U
113     //CV_SVD_V_T Return V^T instead of V
114
115     std::cout  << "SVD rodou com sucesso." << std::endl;
116
117     FILE *matriz_V;
118     matriz_V = fopen("V.txt","w");
119     for (int i=0; i<8 ; i++)
120     {
121     fprintf(matriz_V,"%f %f %f %f %f %f %f %f \n",cvmGet(V,i,0),cvmGet(V,i,1),cvmGet(V,i,2),cvmGet(V↙
        ,i,3),cvmGet(V,i,4),cvmGet(V,i,5),cvmGet(V,i,6),cvmGet(V,i,7));
122     }
123     fclose(matriz_V);
124
125     float v1,v2,v3,v4,v5,v6,v7,v8;
126     v1 = CV_MAT_ELEM(*V,float,0,7);
127     v2 = CV_MAT_ELEM(*V,float,1,7);
128     v3 = CV_MAT_ELEM(*V,float,2,7);
129     v4 = CV_MAT_ELEM(*V,float,3,7);
130     v5 = CV_MAT_ELEM(*V,float,4,7);
131     v6 = CV_MAT_ELEM(*V,float,5,7);
132     v7 = CV_MAT_ELEM(*V,float,6,7);
133     v8 = CV_MAT_ELEM(*V,float,7,7);
134
135     float gama,alpha;
136     gama = sqrt(pow(v1,2) + pow(v2,2) + pow(v3,2));
137     alpha = sqrt(pow(v5,2) + pow(v6,2) + pow(v7,2))/gama;
138
139     //verificar sinal do fator de escala gama
140     float teste;
```

```
141     x = cvmGet(image_points,0,0) - ox;
142     y = cvmGet(image_points,0,1) - oy;
143     Xw = cvmGet(object_points,0,0);
144     Yw = cvmGet(object_points,0,1);
145     Zw = cvmGet(object_points,0,2);
146     v1 = v1/gama;
147     v2 = v2/gama;
148     v3 = v3/gama;
149     v4 = v4/gama;
150     v5 = v5/(gama*alpha);
151     v6 = v6/(gama*alpha);
152     v7 = v7/(gama*alpha);
153     v8 = v8/(gama*alpha);
154     teste = x*(v5*Xw + v6*Yw + v7*Zw + v8);
155     if (teste>0)
156     {
157         v5 = -v5;
158         v6 = -v6;
159         v7 = -v7;
160         v8 = -v8;
161     }
162     teste = y*(v1*Xw + v2*Yw + v3*Zw + v4);
163     if (teste>0)
164     {
165         v1 = -v1;
166         v2 = -v2;
167         v3 = -v3;
168         v4 = -v4;
169     }
170
171     // Produto vetorial para achar R3
172
173     float vb[] = {v1, v2, v3};
174     float va[] = {v5, v6, v7};
175     float vc[3];
176
177     CvMat R1 = cvMat(3, 1, CV_32FC1, va);
178     CvMat R2 = cvMat(3, 1, CV_32FC1, vb);
179     CvMat R3 = cvMat(3, 1, CV_32FC1, vc);
180
181     cvCrossProduct(&R1, &R2, &R3);    // R1 x R2 -> R3
182
183     float r11,r12,r13,r21,r22,r23,r31,r32,r33;
184     r11 = va[0];
185     r12 = va[1];
186     r13 = va[2];
187
188     r21 = vb[0];
189     r22 = vb[1];
190     r23 = vb[2];
191
192     r31 = vc[0];
193     r32 = vc[1];
194     r33 = vc[2];
195
196     // Forçar ortogonalidade de R
197     CvMat* RR1  = cvCreateMat(3,3,CV_32FC1);
198     CvMat* UR   = cvCreateMat(3,3,CV_32FC1);
199     CvMat* DR   = cvCreateMat(3,3,CV_32FC1);
200     CvMat* VR   = cvCreateMat(3,3,CV_32FC1);
201     CvMat* RR2 = cvCreateMat(3,3,CV_32FC1);
202     CvMat* RR3 = cvCreateMat(3,3,CV_32FC1);
203     CvMat* IR   = cvCreateMat(3,3,CV_32FC1);
204
205     cvmSet(RR1,0,0,r11);
206     cvmSet(RR1,0,1,r12);
207     cvmSet(RR1,0,2,r13);
208     cvmSet(RR1,1,0,r21);
209     cvmSet(RR1,1,1,r22);
210     cvmSet(RR1,1,2,r23);
211     cvmSet(RR1,2,0,r31);
```

```
212     cvmSet(RR1,2,1,r32);
213     cvmSet(RR1,2,2,r33);
214
215     cvmSet(IR,0,0,1);
216     cvmSet(IR,0,1,0);
217     cvmSet(IR,0,2,0);
218     cvmSet(IR,1,0,0);
219     cvmSet(IR,1,1,1);
220     cvmSet(IR,1,2,0);
221     cvmSet(IR,2,0,0);
222     cvmSet(IR,2,1,0);
223     cvmSet(IR,2,2,1);
224
225     cvSVD(RR1, DR, UR, VR,CV_SVD_V_T); // A = U D V^T
226
227     cvMatMul(UR,IR,RR2); // UR*IR -> RR2
228     cvMatMul(RR2,VR,RR3); // RR2*VR -> RR3
229
230     va[0] = cvmGet(RR3,0,0);
231     va[1] = cvmGet(RR3,0,1);
232     va[2] = cvmGet(RR3,0,2);
233
234     vb[0] = cvmGet(RR3,1,0);
235     vb[1] = cvmGet(RR3,1,1);
236     vb[2] = cvmGet(RR3,1,2);
237
238     vc[0] = cvmGet(RR3,2,0);
239     vc[1] = cvmGet(RR3,2,1);
240     vc[2] = cvmGet(RR3,2,2);
241
242     r11 = va[0];
243     r12 = va[1];
244     r13 = va[2];
245
246     r21 = vb[0];
247     r22 = vb[1];
248     r23 = vb[2];
249
250     r31 = vc[0];
251     r32 = vc[1];
252     r33 = vc[2];
253
254     //Achar Tz e fx
255
256     CvMat* A2 = cvCreateMat(N,2,CV_32FC1);
257     CvMat* x2 = cvCreateMat(2,1,CV_32FC1);
258     CvMat* b2 = cvCreateMat(N,1,CV_32FC1);
259
260
261     for (int i=0; i<N; i++)
262     {
263         for (int j=0; j<2; j++)
264         {
265         x = cvmGet(image_points,i,0) - ox;
266         Xw = cvmGet(object_points,i,0);
267         Yw = cvmGet(object_points,i,1);
268         Zw = cvmGet(object_points,i,2);
269         switch (j)
270         {
271             case 0:
272                 a_ij = x;
273                 break;
274             case 1:
275                 a_ij = r11*Xw + r12*Yw + r13*Zw + v8;
276                 break;
277         }
278         cvmSet(A2,i,j,a_ij); // carregando valores A(i,j)
279         }
280     }
281
282     float b2_ij;
```

```
283     for (int i=0; i<N; i++)
284     {
285         x = cvmGet(image_points,i,0) - ox;
286         Xw = cvmGet(object_points,i,0);
287         Yw = cvmGet(object_points,i,1);
288         Zw = cvmGet(object_points,i,2);
289         b2_ij = -x*(r31*Xw + r32*Yw + r33*Zw);
290         cvmSet(b2,i,0,b2_ij); // carregando valores A(i,j)
291     }
292 
293     //CvMat* U2_T  = cvCreateMat(2,N,CV_32FC1);
294     //CvMat* D2  = cvCreateMat(2,2,CV_32FC1);
295     //CvMat* V2  = cvCreateMat(2,2,CV_32FC1);
296     //CvMat* A3 = cvCreateMat(2,2,CV_32FC1);
297     //CvMat* A4 = cvCreateMat(2,N,CV_32FC1);
298     //CvMat* D2_INV  = cvCreateMat(2,2,CV_32FC1);
299 
300     //cvSVD(A2, D2, U2_T, V2, CV_SVD_U_T);  // A2 = (U2_T^T)*D2*(V2^T)
301 
302     //cvInvert(D2,D2_INV);                  // inv(D2) -> D2_INV
303     //cvMatMul(V2,D2_INV,A3);               // V2*D2_INV -> A3
304     //cvMatMul(A3,U2_T,A4);                 // A3*U2_T -> A4
305     //cvMatMul(A4,b2,x2);                   // A4*b2 -> x2
306 
307     CvMat* A2T = cvCreateMat(2,N,CV_32FC1);
308     CvMat* A3 = cvCreateMat(2,2,CV_32FC1);
309     CvMat* A4 = cvCreateMat(2,2,CV_32FC1);
310     CvMat* A5 = cvCreateMat(2,N,CV_32FC1);
311     //CvMat *A2T, *A3, *A4, *A5;
312     cvTranspose(A2,A2T); // transpose(A2) -> A2T (cannot transpose onto self)
313     cvMatMul(A2T,A2,A3); // A2T*A2 -> A3
314     cvInvert(A3,A4); // inv(A3) -> A4
315     cvMatMul(A4,A2T,A5); // A4*A2T -> A5
316     cvMatMul(A5,b2,x2); // A5*b2 -> x2
317 
318     std::cout  << "Mínimos quadrados rodou com sucesso."<< std::endl;
319 
320     FILE *intrinsecos;
321     intrinsecos = fopen("intrinsecos_trucco_1.txt","w");
322     fprintf(intrinsecos,"fx = %f \n",cvmGet(x2,1,0));
323     fprintf(intrinsecos,"fy = %f \n",cvmGet(x2,1,0)/alpha);
324     fprintf(intrinsecos,"alpha = %f \n",alpha);
325     fprintf(intrinsecos,"gama = %f \n",gama);
326     fprintf(intrinsecos,"ox = %f \n",ox);
327     fprintf(intrinsecos,"oy = %f \n",oy);
328     fclose(intrinsecos);
329 
330     std::cout  << "Parâmetros intrinsecos exportados com sucesso."<< std::endl;
331 
332     FILE *matriz_R;
333     matriz_R = fopen("R_trucco_1.txt","w");
334     for (int i=0; i<3 ; i++)
335     {
336     fprintf(matriz_R,"%f %f %f \n",va[i],vb[i],vc[i]);
337     }
338     fclose(matriz_R);
339 
340     FILE *matriz_T;
341     matriz_T = fopen("T_trucco_1.txt","w");
342     fprintf(matriz_T,"%f %f %f \n",v8,v4,cvmGet(x2,0,0));
343     fclose(matriz_T);
344 
345     std::cout  << "Parâmetros extrinsecos exportados com sucesso."<< std::endl;
346 
347     //wait key for a user key
348     cv::waitKey(0);
349 
350     return 1;
351 }
```

```
1  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
2  // *                    CÓDIGO PARA CALIBRAÇÃO DE CÂMERA                     *
3  // *                           MÉTODO  TRUCCO 2                              *
4  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
5  //
6  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
7  // *   DISCIPLINA: EEC1515 -  VISÃO COMPUTACIONAL                             *
8  // *   VERSÃO : 1.6                                                           *
9  // *   DATA : 14/10/2011                                                      *
10 // *   CAMERA : WEBCAM CLONE                                                  *
11 // *   RESOLUÇÃO: VGA                                                         *
12 // *   ESCRITO POR: LEONARDO ENZO / ANDRÉ TAVARES                             *
13 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
14
15 //bibliotecas
16 #include <iostream>
17 #include <stdafx.h>
18 #include <cv.h>
19 #include <highgui.h>
20 #include <math.h>
21 #include <stdio.h>
22 #include <conio.h>
23 #include <stdlib.h>
24 #include <string.h>
25
26 int main()
27 {
28
29     // Ler imagem usada para calibração
30     cv::Mat image001 = cv::imread("cubo4.jpg");
31     cv::imshow("Imagem para calibração", image001);
32
33     // Teste de validação da imagem
34     if (!image001.data) { printf("Falha ao abrir a imagem.\n"); }
35     else { printf("Imagem carregada com sucesso.\n"); }
36
37     // Carregando pontos imagem <-> mundo
38     CvMat* image_points = (CvMat*)cvLoad("pts_img.xml");
39     CvMat* object_points = (CvMat*)cvLoad("pts_obj.xml");
40     std::cout  << "Pontos da imagem e do mundo carregados com sucesso." << std::endl;
41
42     int N = 50; //número de pontos imagem/mundo
43
44     CvMat* A  = cvCreateMat(N,12,CV_32FC1);
45     CvMat* U  = cvCreateMat(N,12,CV_32FC1);
46     CvMat* D  = cvCreateMat(12,12,CV_32FC1);
47     CvMat* V  = cvCreateMat(12,12,CV_32FC1);
48
49     float x,y,Xw,Yw,Zw,a_ij;
50     int resto;
51
52     for (int i=0; i<N; i++)
53     {
54         resto = i%2;
55         for (int j=0; j<12; j++)
56         {
57         x = cvmGet(image_points,i,0);
58         y = cvmGet(image_points,i,1);
59         Xw = cvmGet(object_points,i,0);
60         Yw = cvmGet(object_points,i,1);
61         Zw = cvmGet(object_points,i,2);
62         if (resto == 1)
63         {
64             switch (j)
65             {
66                 case 0:
67                     a_ij = Xw;
68                     break;
69                 case 1:
70                     a_ij = Yw;
71                     break;
```

```
72                  case 2:
73                      a_ij = Zw;
74                      break;
75                  case 3:
76                      a_ij = 1;
77                      break;
78                  case 4:
79                      a_ij = 0;
80                      break;
81                  case 5:
82                      a_ij = 0;
83                      break;
84                  case 6:
85                      a_ij = 0;
86                      break;
87                  case 7:
88                      a_ij = 0;
89                      break;
90                  case 8:
91                      a_ij = -x*Xw;
92                      break;
93                  case 9:
94                      a_ij = -x*Yw;
95                      break;
96                  case 10:
97                      a_ij = -x*Zw;
98                      break;
99                  case 11:
100                     a_ij = -x;
101                     break;
102             }
103         }
104         else
105         {
106             switch (j)
107             {
108                 case 0:
109                     a_ij = 0;
110                     break;
111                 case 1:
112                     a_ij = 0;
113                     break;
114                 case 2:
115                     a_ij = 0;
116                     break;
117                 case 3:
118                     a_ij = 0;
119                     break;
120                 case 4:
121                     a_ij = Xw;
122                     break;
123                 case 5:
124                     a_ij = Yw;
125                     break;
126                 case 6:
127                     a_ij = Zw;
128                     break;
129                 case 7:
130                     a_ij = 1;
131                     break;
132                 case 8:
133                     a_ij = -y*Xw;
134                     break;
135                 case 9:
136                     a_ij = -y*Yw;
137                     break;
138                 case 10:
139                     a_ij = -y*Zw;
140                     break;
141                 case 11:
142                     a_ij = -y;
```

```
143                     break;
144                 }
145             }
146             cvmSet(A,i,j,a_ij); // carregando valores A(i,j)
147             }
148     }
149
150     std::cout  << 1%2 << std::endl;
151
152     FILE *matriz_A;
153     matriz_A = fopen("A.txt","w");
154     for (int i=0; i<50 ; i++)
155     {
156     fprintf(matriz_A,"%f %f %f %f %f %f %f %f %f %f %f %f \n",cvmGet(A,i,0),cvmGet(A,i,1),cvmGet(A,i
        ,2),cvmGet(A,i,3),cvmGet(A,i,4),cvmGet(A,i,5),cvmGet(A,i,6),cvmGet(A,i,7),cvmGet(A,i,8),cvmGet(A
        ,i,9),cvmGet(A,i,10),cvmGet(A,i,11));
157     }
158     fclose(matriz_A);
159
160     std::cout  << "Matriz A carregada com sucesso." << std::endl;
161
162     cvSVD(A, D, U, V); // A = U D V^T
163     //flags
164     //CV_SVD_MODIFY_A Allows modification of matrix A
165     //CV_SVD_U_T Return U^T instead of U
166     //CV_SVD_V_T Return V^T instead of V
167
168     std::cout  << "SVD rodou com sucesso." << std::endl;
169
170
171     FILE *matriz_M;
172     matriz_M = fopen("M.txt","w");
173     for (int i=0; i<12 ; i++)
174     {
175     fprintf(matriz_M,"%f %f %f %f %f %f %f %f %f %f %f %f \n",cvmGet(V,i,0),cvmGet(V,i,1),cvmGet(V,i
        ,2),cvmGet(V,i,3),cvmGet(V,i,4),cvmGet(V,i,5),cvmGet(V,i,6),cvmGet(V,i,7),cvmGet(V,i,8),cvmGet(V
        ,i,9),cvmGet(V,i,10),cvmGet(V,i,11));
176     }
177     fclose(matriz_M);
178
179     float m11,m12,m13,m14,m21,m22,m23,m24,m31,m32,m33,m34;
180     m11 = CV_MAT_ELEM(*V,float,0,11);
181     m12 = CV_MAT_ELEM(*V,float,1,11);
182     m13 = CV_MAT_ELEM(*V,float,2,11);
183     m14 = CV_MAT_ELEM(*V,float,3,11);
184     m21 = CV_MAT_ELEM(*V,float,4,11);
185     m22 = CV_MAT_ELEM(*V,float,5,11);
186     m23 = CV_MAT_ELEM(*V,float,6,11);
187     m24 = CV_MAT_ELEM(*V,float,7,11);
188     m31 = CV_MAT_ELEM(*V,float,8,11);
189     m32 = CV_MAT_ELEM(*V,float,9,11);
190     m33 = CV_MAT_ELEM(*V,float,10,11);
191     m34 = CV_MAT_ELEM(*V,float,11,11);
192
193     float gama;
194     gama = sqrt(pow(m31,2) + pow(m32,2) + pow(m33,2));
195
196     // Normalização
197
198     m11 = m11/gama;
199     m12 = m12/gama;
200     m13 = m13/gama;
201     m14 = m14/gama;
202     m21 = m21/gama;
203     m22 = m22/gama;
204     m23 = m23/gama;
205     m24 = m24/gama;
206     m31 = m31/gama;
207     m32 = m32/gama;
208     m33 = m33/gama;
209     m34 = m34/gama;
```

```
210
211     float Tx,Ty,Tz,sigma;
212     Tz = m34;
213
214     // Verificar sinal do fator de escala gama
215     float teste;
216     teste = Tz;
217     if (teste < 0)
218     {
219         sigma = -1;
220     }
221     else
222     {
223         sigma = 1;
224     }
225
226     Tz = sigma*m34;
227
228     float r11,r12,r13,r21,r22,r23,r31,r32,r33;
229
230     r31 = sigma*m31;
231     r32 = sigma*m32;
232     r33 = sigma*m33;
233
234     // Produto escalar
235
236     float v1[] = {m11, m12, m13};
237     float v2[] = {m21, m22, m23};
238     float v3[] = {m31, m32, m33};
239     float v4[] = {m14, m24, m34};
240
241     CvMat q1 = cvMat(3, 1, CV_32FC1, v1);
242     CvMat q2 = cvMat(3, 1, CV_32FC1, v2);
243     CvMat q3 = cvMat(3, 1, CV_32FC1, v3);
244     CvMat q4 = cvMat(3, 1, CV_32FC1, v4);
245
246     float ox = cvDotProduct(&q1,&q3); // produto escalar:   q1 . q3 -> res
247     float oy = cvDotProduct(&q2,&q3); // produto escalar:  q1 . q3 -> res
248     float fx = sqrt(cvDotProduct(&q1,&q1) - pow(ox,2)); // produto escalar:   q1 . q3 -> res
249     float fy = sqrt(cvDotProduct(&q2,&q2) - pow(oy,2)); // produto escalar:   q1 . q3 -> res
250
251     r11 = sigma*(ox*m31-m11)/fx;
252     r12 = sigma*(ox*m32-m12)/fx;
253     r13 = sigma*(ox*m33-m13)/fx;
254
255     r21 = sigma*(oy*m31-m21)/fy;
256     r22 = sigma*(oy*m32-m22)/fy;
257     r23 = sigma*(oy*m33-m23)/fy;
258
259     Tx = sigma*(ox*Tz - m14)/fx;
260     Ty = sigma*(oy*Tz - m24)/fy;
261
262     // Forçar ortogonalidade de R
263     CvMat* RR1  = cvCreateMat(3,3,CV_32FC1);
264     CvMat* UR   = cvCreateMat(3,3,CV_32FC1);
265     CvMat* DR   = cvCreateMat(3,3,CV_32FC1);
266     CvMat* VR   = cvCreateMat(3,3,CV_32FC1);
267     CvMat* RR2 = cvCreateMat(3,3,CV_32FC1);
268     CvMat* RR3 = cvCreateMat(3,3,CV_32FC1);
269
270     cvmSet(RR1,0,0,r11);
271     cvmSet(RR1,0,1,r12);
272     cvmSet(RR1,0,2,r13);
273     cvmSet(RR1,1,0,r21);
274     cvmSet(RR1,1,1,r22);
275     cvmSet(RR1,1,2,r23);
276     cvmSet(RR1,2,0,r31);
277     cvmSet(RR1,2,1,r32);
278     cvmSet(RR1,2,2,r33);
279
280     cvSVD(RR1, DR, UR, VR,CV_SVD_V_T); // A = U D V^T
```

```
281
282     CvMat* IR  = cvCreateMat(3,3,CV_32FC1);
283     cvmSet(IR,0,0,1);
284     cvmSet(IR,0,1,0);
285     cvmSet(IR,0,2,0);
286     cvmSet(IR,1,0,0);
287     cvmSet(IR,1,1,1);
288     cvmSet(IR,1,2,0);
289     cvmSet(IR,2,0,0);
290     cvmSet(IR,2,1,0);
291     cvmSet(IR,2,2,1);
292
293     cvMatMul(UR,IR,RR2); // UR*IR -> RR2
294     cvMatMul(RR2,VR,RR3); // RR2*VR -> RR3
295
296     r11 = cvmGet(RR3,0,0);
297     r12 = cvmGet(RR3,0,1);
298     r13 = cvmGet(RR3,0,2);
299
300     r21 = cvmGet(RR3,1,0);
301     r22 = cvmGet(RR3,1,1);
302     r23 = cvmGet(RR3,1,2);
303
304     r31 = cvmGet(RR3,2,0);
305     r32 = cvmGet(RR3,2,1);
306     r33 = cvmGet(RR3,2,2);
307
308     ////////////////////////////////////////////////////////////////////
309     // Exportar resultados para arquivo .txt
310     ////////////////////////////////////////////////////////////////////
311
312     std::cout  << "Exportando parametros para arquivos .txt ..."<< std::endl;
313
314     FILE *intrinsecos;
315     intrinsecos = fopen("intrinsecos_trucco_2.txt","w");
316     fprintf(intrinsecos,"fx = %f \n", fx);
317     fprintf(intrinsecos,"fy = %f \n", fy);
318     fprintf(intrinsecos,"ox = %f \n",ox);
319     fprintf(intrinsecos,"oy = %f \n",oy);
320     fprintf(intrinsecos,"gama (não é parâmetro intrínseco) = %f \n", gama);
321     fprintf(intrinsecos,"sigma (não é parâmetro intrínseco) = %f \n", sigma);
322     fclose(intrinsecos);
323
324     std::cout  << "Parametros intrinsecos exportados com sucesso."<< std::endl;
325
326     FILE *matriz_R;
327     matriz_R = fopen("R_trucco_2.txt","w");
328     fprintf(matriz_R,"%f %f %f \n",r11,r12,r13);
329     fprintf(matriz_R,"%f %f %f \n",r21,r22,r23);
330     fprintf(matriz_R,"%f %f %f \n",r31,r32,r33);
331     fclose(matriz_R);
332
333     FILE *matriz_T;
334     matriz_T = fopen("T_trucco_2.txt","w");
335     fprintf(matriz_T,"%f %f %f \n",Tx,Ty,Tz);
336     fclose(matriz_T);
337
338     std::cout  << "Parametros extrinsecos exportados com sucesso."<< std::endl;
339
340     //wait key for a user key
341     cv::waitKey(0);
342
343     return 1;
344 }
```