# Trabalho 1 - Medições e Rastreamento em Vídeo Estéreo

Johannes Peter Schulte
johpetsc@gmail.com

Departamento de Ciência da Comptutação
Universidade de Brasília
Campus Darcy Ribeiro, Asa Norte
Brasília-DF, CEP 70910-900, Brazil,

**Abstract**

O trabalho que será apresentado, tem como objetivo apresentar e implementar os primeiros conceitos da disciplina de Visão Computacional. Por meio de vídeos e imagens de duas câmeras, serão encontrados os parâmetros intrínsecos e extrínsecos para que sejam feitas as respectivas calibrações e simular o comportamento de uma câmera estéreo. Por meio desses resultados, será possível identificar a posição de objetos no plano 3D através de suas coordenadas 2D e vice-versa.

## 1 Introdução

Visão Computacional é a ciência de se fazer um computador extrair e interpretar informações por meio de representações gráficas do nosso mundo. Para isso, é necessário que as fontes desses dados, que são alimentados ao computador, sejam o mais fidedignas possíveis à realidade, fazendo com que sejam necessárias utilização de técnicas para que tais resultados sejam alcançados.

Para o trabalho, serão utilizados arquivos de imagem e vídeos capturados por duas câmeras com aspectos diferentes uma da outra. Pela natureza física das câmeras atuais, onde a imagem capturada depende de deformações causadas por lentes ou detalhes de alinhamento dependentes de suas produções, é necessário que as câmeras sejam calibradas para que as imagens capturadas por elas sejam uma representação visual correta do espaço 3D capturado.

## 2 Calibração de Intrínsecos

A calibração de intrínsecos é a obtenção dos parâmetros intrínsecos que descrevem as propriedades geométricas de uma câmera. Seguindo o modelo de uma câmera pinhole, uma matriz de intrínsecos é dada por K, onde $f_x$ e $f_y$ são os parâmetros referentes à distância focal, que é a distância entre a *pinhole* e o plano de imagem(*image plane*). Numa câmera pinhole ideal, ambos $f_x$ e $f_y$ teriam o mesmo valor, porém, em câmeras digitais, essa diferença pode acontecer por uma série de fatores, mas os valores não devem divergir muito. [9]

$$K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

Os parâmetros $x_o$ e $x_y$ são o deslocamento do ponto principal(*principal point*), onde $x_0$ se refere ao deslocamento horizontal e $y_0$ ao vertical. O ponto principal é o ponto onde a linha principal, que passa pela pinhole, se encontra perpendicularmente com o plano de imagem. [9]

Por último, o parâmetro $s$ referente à inclinação do eixo que causa distorção na imagem. Esse parâmetro não se aplica em uma câmera pinhole ideal e raramente é diferente de 0 em câmeras digitais. [9]

## 2.1 Metodologia

Para a calibração dos parâmetros intrínsecos foram utilizadas as imagens com o padrão de tabuleiro de xadrez fornecidas para cada câmera com as ferramentas de calibração da biblioteca *OpenCV*, que serão identificadas por "cv." ao longo do texto.

O primeiro passo é encontrar as coordenadas, em pixel, para cada canto do tabuleiro com $cv.findChessboardCorners()$, onde são passadas a imagem e as dimensões do tabuleiro em linhas por colunas. A função retorna uma matriz com as coordenadas de cada canto do tabuleiro identificado na imagem, o resultado é visto com a função *cv.drawChessboardCorners*. Por meio dessa função foi possível encontrar imagens, na câmera 2, onde os cantos não foram identificados corretamente e estavam comprometendo o resultado final. [8]

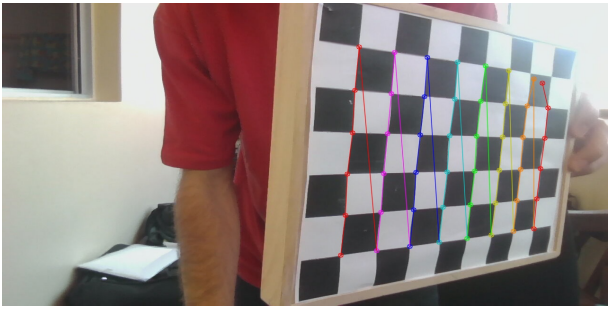

Figure 1: Padrão errado na câmera 2

Como é possível ver pela figura 1, o algoritmo não encontrava corretamente alguns cantos do tabuleiro, isso causava uma calibração incorreta e resultava em imagens distorcidas. O problema é causado pela qualidade inferior da câmera 2, o fato do tabuleiro ser uma impressão com algumas pontas dobradas e onduladas, junto com o fato de uma fonte de luz vinda da janela está ocultando parte da imagem. Mesmo que a câmera 2 já tenha um resultado inferior à câmera 1 por ter menos imagens com padrões de xadrez, a solução para o problema foi remover as imagens com problemas da pasta utilizada para a calibração.

Agora com todas as imagens tendo os cantos detectados, temos uma lista de matrizes contendo as coordenadas de cada canto para cada imagem. Cada um desses cantos é então associado à uma matriz de pontos em mundo real em mesmas proporções do tabuleiro(no caso, 8x6). Os pontos não utilizam medidas do mundo real, foram assumidos que os cantos estavam separados por uma distância unitária.

Por fim, utilizamos os pontos das imagens e o padrão de pontos para gerar a matriz de intrínsecos e o vetor de distorção de cada câmera utilizando $cv.calibrateCamera()$. [8]

## 2.2 Resultados e análise

Os resultados de calibração da câmera 1 provavelmente foram mais precisos por uma série de fatores: câmera de melhor qualidade, maior resolução das imagens, melhor iluminação, mais imagens para calibração e algumas imagens foram capturadas com um tabuleiro realmente reto. Os resultados foram:

$$K_1 = \begin{pmatrix} 898.8461771 & 0 & 631.9191529 \\ 0 & 899.98015849 & 357.08688478 \\ 0 & 0 & 1 \end{pmatrix}$$

Os valores de $f_x$ e $f_y$ se diferem em aproximadamente 0.1% e os pontos de deslocamento estão próximos da proporção da resolução(16/9 = 1,777..., 632/357 = 1,755...). Os valores não apresentam nenhuma discrepância ou valores estranhos, o que indica que foi uma calibração bem sucedida.

Para a câmera 2, mesmo com recursos inferiores, a calibração também gerou bons resultados:

$$K_2 = \begin{pmatrix} 955.37564478 & 0 & 626.37229852 \\ 0 & 955.70796633 & 345.49761106 \\ 0 & 0 & 1 \end{pmatrix}$$

As distâncias focais estão ainda mais próximas do que na câmera 1, com uma diferença menor que 0.1%. A proporção dos deslocamentos é um pouco mais distante, 626/345 = 1,814.., o que indica que o ponto principal está um pouco mais distante do centro do plano de imagem.

# 3 Calibração de Extrínsecos

Os parâmetros extrínsecos se referem às informações da câmera em relação ao mundo, como sua posição, ângulo e rotação. Para estimar as posições das câmeras, são calculadas as matrizes de rotação e o vetor de translação, que formam a matriz de extrínsecos, onde $r_{m,n}$ são os parâmetros da matriz de rotação e $t_n$ os parâmetros do vetor de translação, que são encontrados a partir de pontos em um plano no mundo real. [8]

$$[R|T] = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{pmatrix}$$

## 3.1 Metodologia

Para calcular a matriz de extrínsecos, foram utilizados as coordenadas do mundo real representadas pelas quatro peças posicionadas formando um plano no chão. O plano tem sua coordenada (0, 0, 0) no canto superior esquerdo, varia 2,6 metros no eixo $x$ e 1,4 no eixo $y$. As que os pontos representam nos vídeo foram inseridas direto no código, para que não seja necessário o clique de mouse para identificação.

Com os pontos de imagem e no plano 3D, a função *cv.solvePnP* foi utilizada, junto com os parâmetros intrínsecos do primeiro requisito para calcular o vetor de rotação de translação. A partir do vetor de rotação é possível obter a matriz de rotação com *cv.Rodrigues*. [9]

Na figura 2 são projetados os eixos do plano param fins ilustrativos utilizando os resultados. O ponto representado se encontra na posição (2.6, 0.7, 0) seguindo o padrão X, Y, Z e as linhas são projetadas 0.5m em cada direção.

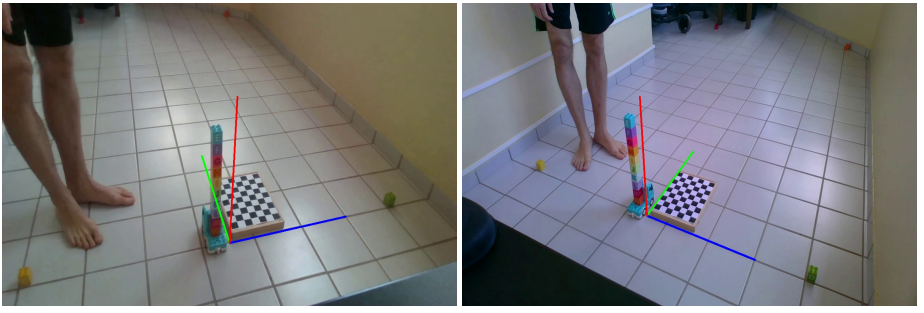

Figure 2: Projeção dos eixos 3D a partir da matriz de extrínsecos

## 3.2 Resultados e análise

$$[R|T]_1 = \begin{pmatrix} -0.43235451 & 0.90168808 & 0.00530871 & 0.31577748 \\ 0.49274284 & 0.24118995 & -0.83608128 & -1.20997097 \\ -0.75516493 & -0.35886768- & 0.54857991 & 3.68864699 \end{pmatrix}$$

Tendo os resultados da calibração de extrínsecos da câmera 1, podemos encontrar a posição da câmera em relação ao plano calculando a transposta da matriz de rotação com *transpose* da biblioteca *numpy*, a matriz resultante é multiplicada por $(-1)$ para obter a oposta e, por último, geramos o produto da matriz com o vetor de translação usando a função *dot*. O resultado para o eixo $X$ : [3.5182692], ou seja, tendo como referência o ponto (0, 0, 0) fornecido, a câmera 1 se encontra 3,5 metros de distância. Para o eixo $Y$ : [1.33083624], tendo como referência o mesmo ponto, a câmera 1 se encontra 1,33 metros à direita do mesmo. A altura da câmera resultante $Z$ : [1.01020719] indica que a câmera se encontra, aproximadamente, à 1 metro de altura do chão, que é a coordenada 0 no plano.

Para a matriz da segunda câmera, é possível reparar que alguns valores da matriz são parecidos com os da matriz anterior, e outros são parecidos porém com sinal invertido. Isso pode indicar que, por exemplo, as câmeras possuem posições no mundo parecidas uma com a outra, mas também estão rotacionadas para direções opostas, já que as duas capturam as mesmas posições porém de lados opostos da mesa.

$$[R|T]_2 = \begin{pmatrix} 0.29310942 & 0.95448981 & 0.05510061 & -1.45557307 \\ 0.52685471 & -0.11315986 & -0.84238884 & -1.00690123 \\ -0.79781638 & 0.27594212 & -0.53604567 & 3.15590542 \end{pmatrix}$$

Por mais que seja possível fazer uma análise pela matriz gerada, as posições exatas ainda são desconhecidas, por isso, seguindo o processo anterior, encontramos o valor de $X$ : [3.47496588] pelo qual é possível dizer que as duas câmeras formam uma linha paralela ao eixo Y do plano, já que sua posição no eixo X se difere em apenas 5 centímetros, o que pode indicar que a câmera dois está um pouco mais próxima do ponto ou alguma imprecisão nos resultados, porém numa margem de erro aceitável. Já no eixo $Y$ : [0.40454163] a câmera 2 se desloca 40 centímetros à direita do ponto e, aproximadamente, 1 metro à esquerda da câmera 1. Por último, o eixo $Z$ : [0.92371006] indica que a câmera 2 mais próxima do chão que a câmera 1, indicando que, por exemplo, o notebook no qual pertence é mais baixo, ou a tela está mais inclinada para a frente(o que explicaria estar um pouco à frente no eixo X), ou alguma imprecisão na calibração.

Os resultados são bem satisfatórios se as posições forem analisadas em relação ao ponto do plano mais próximo à cada câmera. Analisando pela imagem, a câmera 1 parece estar em linha com o eixo Y, para trás e acima do ponto D, e isso pode ser observado: 7 centímetros para à esquerda e, aproximadamente, 1 metro pra cima e para trás. Já a câmera 2, pela imagem, está entre o ponto C e o tabuleiro(0.7 no eixo Y) e igual a câmera 1 nos eixos X e Z. Sua posição se encontra à 40 centímetros para à direita e 1 metro para cima e para trás, o que bate com o que foi observado pela imagem.

# 4 Estimativa do Mapa de Profundidade

Se tratando de visão estéreo, uma imagem do mundo é vista através de duas câmeras posicionadas de forma a capturar objetos em comum. A partir disso podemos extrair informações de uma imagem 2D para informações do mundo observado. A informação de interesse no caso é a estimativa da profundidade, que pode ser conhecida através da disparidade entre os elementos em cada imagem. [1]

## 4.1 Metodologia

Para calcular a disparidade entre duas imagens é necessário que as mesmas estejam retificadas, ou seja, qualquer ponto em uma das imagens deve estar na mesma linha horizontal na outra imagem. O primeiro passo para a retificação foi fazer a calibração estéreo com $cv.stereoCalibrate()$ para gerar os valores de rotação e transação de uma câmera em relação à outra(ao invés de em relação ao mundo). Tendo esses valores, a retificação é feita com: $cv.stereoRectify()$ calcula a retificação; $cv.initUndistortRectifyMap()$ calcula o mapa de destorção e retificação; $cv.remap()$ realiza o remapeamento das imagens, resultando na imagem retificada. [5]

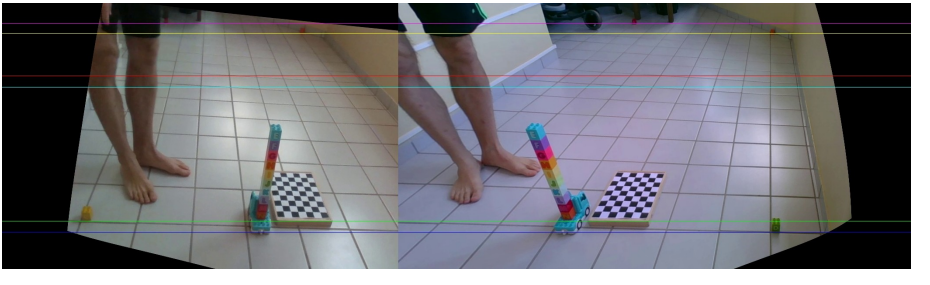

Figure 3: Imagem retificada

A Figura 3 mostra a imagem retificada, onde as linhas passam pelos pontos conhecidos no mundo real. As imagens aparecem cortadas por causa do parâmetro $alpha$ da função $cv.stereoRectify()$, que foi passado com o intuito de aprimorar a detecção da disparidade entre os pontos mais ao centro da imagem, sem alterar os resultados.

Com as imagens retificadas, os mapas são gerados com $cv.StereoSGBM()$ para a disparidade, e com isso $cv.reprojectImageTo3D$ gera um mapa onde, para cada pixel, temos um vetor de coordenadas 3D, a partir do qual extraímos apenas a coordenada de profundidade e geramos o mapa. [6]

## 4.2 Resultados e análise

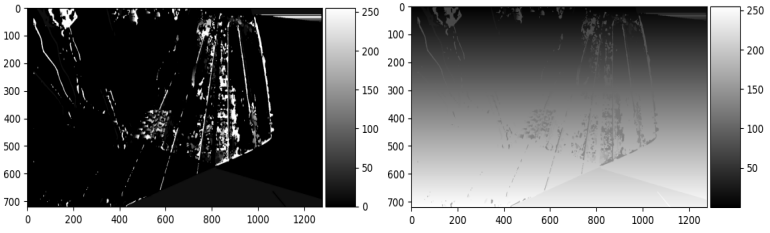

Figure 4: Mapa de disparidade à esquerda, profundidade à direita.

O mapa de disparidade é feito a partir da câmera 1, e por isso os objetos mais próximos à ela foram melhor detectados pelo algoritmo. Por mais que algumas linhas do chão tenham sido reconhecidas, o chão, por ter a mesma textura, está representado pelo valor 0, que equivale aos pixeis sem correspondência. O mapa de disparidade nesse caso não se assemelha aos mapas de uma câmera estéreo, onde os objetos mais próximos tem uma disparidade maior do que os do fundo, ao invés disso, os objetos nas extremidades possuem maior disparidade que os do centro. Isso acontece por causa do ângulo em que as câmeras se encontram, fazendo com que os pontos nas extremidades possuam coordenadas mais discrepantes do que as do centro.

Como o mapa de profundidade foi feito pelo de disparidade os dois representam os objetos igualmente. É possível perceber que, mesmo o chão não sendo reconhecido no mapa de disparidade, é possível ter uma certa estimativa da profundidade por causa dos parâmetros encontrados na calibração.

# 5 Rastreamento de pontos em 3D

Com todos as informações sobre as câmeras e a retificação feita, é possível rastrear objetos nas imagens, lendo suas coordenadas 2D nas imagens e transformando para suas coordenada 3D.

## 5.1 Metodologia

Para obter as posições 2D do objeto rastreado, foi utilizado a função *cv.TrackerMIL*, onde é selecionado uma área e o objeto é rastreado para cada frame, onde se retorna as coordenadas que são inseridas numa lista frame por frame para cada uma das câmeras. Com os pontos de cada vídeo armazenados, as listas são alimentadas à função $cv.triangulatePoints()$ junto às matrizes de projeção calculadas função $cv.stereoRectify()$, o resultado é uma lista com coordenadas 4D(X,Y,Z,W)para cada frame onde foi rastreado o objeto, e a coordenada 3D é gerada dividindo X, Y e Z por W. [2, 7]

## 5.2 Resultados e análise

Foram gerados gráficos para cada eixo rastreando o topo e a base do objeto, onde o topo passa alguns frames fora do vídeo. As coordenadas são em relação às câmeras e não ao plano usado na calibração.

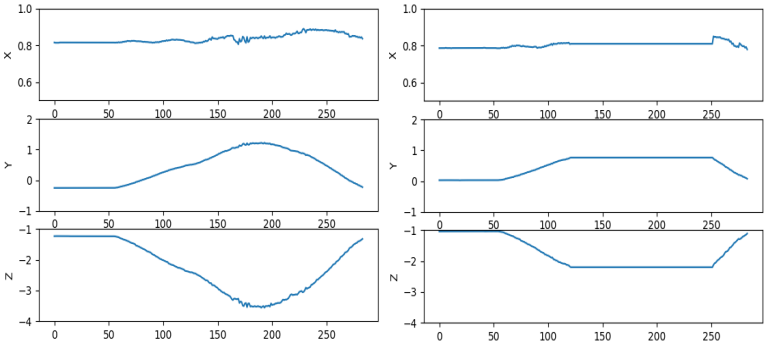


Figure 5: Base à direita, topo à esquerda

Analisando os gráficos, o eixo X, que é paralelo às câmeras, se mantém constante durante o trajeto, com uma variação de não mais que 10 centímetros, que é causada pela irregularidade do trajeto e algumas falhas da função de rastreamento.

O eixo Z, que é de profundidade e perpendicular às câmeras, também apresenta bons resultados: inicia o trajeto à 1 metro de distância das câmeras e vai até, aproximadamente, 3,6 metros, o que corresponde aos 2,6 metros de profundidade no eixo X do plano utilizado no requisito 2, assim como a distância do objeto às câmeras.

Para o eixo Y, que se refere à altura do objeto em relação à câmera, os resultados possuem uma margem de erro maior. É possível perceber que as coordenadas do eixo Y são inversamente relacionadas com o eixo Z. Isso quer dizer que o algoritmo perde noção de profundidade quanto mais o objeto se distancia. O motivo de isso acontecer, provavelmente, é porque a câmera foi calibrada com coordenadas que variam apenas nos eixos de distância e profundidade, e por isso o programa tem maiores dificuldades em rastrear a altura do objeto. Mesmo com essa falha, é possível perceber que a distância, em altura, do topo para a base, se mantém entre 30 e 25 centímetros ao longo do percurso do objeto, o que demonstra que o programa possui a capacidade de perceber alterações no eixo Y mesmo que não sejam tão precisas como os outros eixos.

# 6 Conclusão

O desafio do trabalho foi utilizar vídeos de duas câmeras, com aspectos técnicos completamente diferentes, posicionadas em posições e ângulos opostos e fazê-las se portarem como uma câmera estéreo. Os resultados alcançados foram satisfatórios, mesmo que, em alguns critérios, não perfeitos.

A maior deficiência que o resultado apresentou foi a falha do programa em perceber a altura dos objetos quando eles se afastavam das câmeras, mesmo os outros eixos apresentando resultados compatíveis coma a realidade.

Como a calibração foi feita utilizando quatro pontos posicionados no chão, que variavam apenas em dois eixos, é possível levantar a hipótese de que os objetos que variam em relação ao outro eixo não foram identificados de forma ideal, causando as falhas observadas no último requisito.

# References

[1] A. Zisserman S. Lazebnik adapted by Rob Fergus. Stereo reconstruction, 2020. https://cs.nyu.edu/~fergus/teaching/vision/.

[2] dos Santos. Revisão de conceitos em geometria de câmeras, 2012. Relatório de aluno do Anderson Rocha.

[3] Alexander Mordvintsev Abid K. Camera calibration, 2013. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.

[4] OpenCV. Pose estimation, 2020. https://docs.opencv.org/master/d7/d53/tutorial_py_pose.html.

[5] OpenCV. Camera calibration and 3d reconstruction, 2020. https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga617b1685d4059c6040827800e72ad2b6.

[6] OpenCV. Depth map from stereo images, 2020. https://docs.opencv.org/master/dd/d53/tutorial_py_depthmap.html.

[7] OpenCV. Triangulation, 2020. https://docs.opencv.org/3.4/d0/dbd/group__triangulation.html.

[8] Kyle Simek. Dissecting the camera matrix: The extrinsic matrix, 2012. http://ksimek.github.io/2012/08/22/extrinsic/.

[9] Kyle Simek. Dissecting the camera matrix: The intrinsic matrix, 2013. http://ksimek.github.io/2013/08/13/intrinsic/.