UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA

TRABALHO DA DISCIPLINA INF01030 - FUNDAMENTOS DE VISÃO COMPUTACIONAL

Implementação de algoritmo de calibração de câmera

Alunos: Leonardo Oliveira Wellausen
Matheus Fernandes Kovaleski
Orientadore: Prof. Dr. Cláudio Rosito Jung

Porto Alegre, Setembro de 2019

Sumário

Lista de Figuras

1 Introdução

O objetivo deste trabalho de implimentação é realizar a calibração de duas câmeras diferentes, lançando mão do método **DLT** e com base em duas imagens e dimensões conhecinhas dos objetos representados nestas imagens.

As ferramentas utilizadas para cumprir tal objetivo foram a linguagem de programação *Python 3* com o uso das seguintes bibliotecas não padrão: *opencv-python* para leitura e exibição de imagens; *numpy* para cálculo matricial, incluindo resolução de sistema linear pelo método **SVD**.

Para cada uma das questões presentes no enunciado do trabalho prático foi escrito um programa lançando mão das ferramentas citadas acima. Ambos programas são interativos e respondem a cliques do usuário executando as tarefas especificadas em seus respectivos enunciados; assim sendo, o funcionamento dos programas é como segue:

- *questao1.py* - Exibe a imagem maracana1.jpg e a cada clique com o botão esquerdo do mouse projeta um jogador (segmentos de reta de cor aleatória) de 1.8 metros de altura na posição clicada. Cliques com o botão direito projetam um Neymar aleatório (extraído de uma base de dados de Neymar) na posição clicada.

- *questao2.py* - Exibe a imagem maracana2.jpg e a cada clique com o botão esquerdo do mouse projeta uma linha de impedimento virtual passando por este ponto. Ou seja, uma linha que passa pelo ponto selecionado e é paralela (no mundo) à linha de fundo do campo de futebol.

2 Metodologia

Esta seção discutirá sobre a metodologia adotada para se atingir o resultado final. Partindo da escolha de pontos de calibração até implementação.

2.1 Pontos para a calibração

O método utilizado para a calibração de câmera neste exercício exige a resolução do sistema linear em 2.1. Este sistema é baseado em correspondências já existentes entre pontos no sistema de coordenadas de imagem e pontos no sistema de coordenadas real de mundo de objetos representados nesta imagem. Estes pontos de correspondência foram escolhidos de forma manual e, inicialmente, arbitrária.

Começamos com diversas tentativas de seleção de pontos de forma bastante espalhada pela imagem, com a justificativa de fornecer informação mais diversificada (em relação aos eixos canônicos) sobre o objeto (campo de futebol), notamos que esta tática não estava funcionando. A matriz de projeção obtida realmente fornecia resultados bons para os pontos já conhecidos (projetava um ponto $\mathbf{x_w}$ conhecido próximo ao ponto $\mathbf{x_i}$ conhecido) porém para outros pontos (também conhecidos, mas não inseridos no sistema) apresentava um erro enorme.

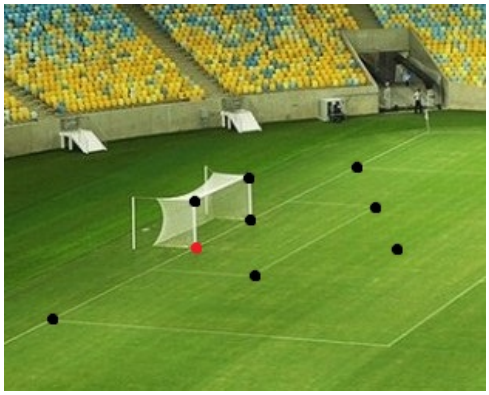
Após muitos resultados horríveis (como em 2.2) percebemos que pontos com maior simetria apresentavam resultados mais uniformes para a integridade do campo visível, mesmo que esses pontos fossem concentrados em uma região menor da imagem. Assim as seleções de finais de pontos, que apresentaram melhores resultados, são "simétricos"em relação aos eixos canônicos, e concentrados em uma região pequena do campo de futebol.

Após obtermos bons resultados com o mínimo necessário de pontos (6 e 4), começamos a adicionar pontos extras a fim de refinar esses resultados. Certas adições de pontos realmente levaram a resultados melhores! As seleções finais de pontos de correspôndencias são exibidas em 2.3a e 2.3b.

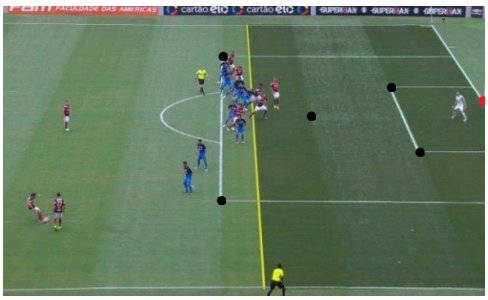
$$\begin{bmatrix} x_1 & y_1 & z_1 & 1 & 0 & 0 & 0 & 0 & -u_1x_1 & -u_1y_1 & -u_1z_1 & -u_1 \\ 0 & 0 & 0 & 0 & x_1 & y_1 & z_1 & 1 & -v_1x_1 & -v_1y_1 & -v_1z_1 & -v_1 \\ x_2 & y_2 & z_2 & 1 & 0 & 0 & 0 & 0 & -u_2x_2 & -u_2y_2 & -u_2z_2 & -u_2 \\ 0 & 0 & 0 & 0 & x_2 & y_2 & z_2 & 1 & -v_2x_2 & -v_2y_2 & -v_2z_2 & -v_2 \\ & & & & & & \vdots & & & & & \\ x_n & y_n & z_n & 1 & 0 & 0 & 0 & 0 & -u_nx_n & -u_ny_n & -u_nz_n & -u_n \\ 0 & 0 & 0 & 0 & x_n & y_n & z_n & 1 & -v_nx_n & -v_ny_n & -v_nz_n & -v_n \end{bmatrix} \begin{pmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

Figura 2.1: Sistema de equações lineares resolvido para a calibração da câmera

Figura 2.2: A diagonal pontilhada deveria ser paralela à linha lateral. Nota-se que o ponto da bandeirinha, fornecido ao sistema, está correto



(a) Maracana1.



(b) Maracana2.

Figura 2.3: Pontos de correspondência selecionados para as duas imagens. Origem dos sistemas de coordenadas de mundo nos pontos vermelhos.

2.2 Principais Funções

Aqui apresentamos de forma breve as principais funções presentes em cada um, ou ambos, dos programas fornecidos.

2.2.1 Função *projection(mat, arr)*

Presente em ambos programas, com diferenças de dimensionalidade apenas. Recebe uma matriz *mat*, representando a matriz **P** de projeção de uma certa câmera, e um ponto *arr* representando um ponto **3D** em coordenadas de mundo (coordenadas homogêneas). Efetua a multiplicação de *arr* por *mat*, seguida de divisão perspectiva. Retorna o ponto *arr* projetado em coordenadas **2D** de imagem.

2.2.2 Função *reprojection(mat, arr)*

Presente em ambos programas, com diferenças de dimensionalidade apenas. Recebe uma matriz *mat*, representando a inversa da matriz **P** de projeção de uma certa câmera, e um ponto *arr* representando um ponto **2D** em coordenadas de imagem. Efetua a multiplicação de *arr* por *mat*, seguida de divisão perspectiva. Retorna o ponto *arr* projetado em coordenadas **3D** de mundo, restrito aoplano $\mathbf{z = 0}$. Diferença meramente semântica da função anterior, porém também efetua um tratamento diferenciado dos vetores de entrada, visto que nossos pontos de imagem não estão, no geral, em coordenadas homogêneas.

2.2.3 Funções *draw_square_at draw_line* e *draw_ney*

Funções simples presentes nos dois programas. São as funções responsáveis por alterar a imagem original, desenhando segmentos de reta, pontos ou Neymar.

2.2.4 Função *mouse_callback* em *questao1.py*

Função responsável por tratar a interrupção do clique do mouse, provocada pelo usuário, em uma posição **x, y**. Este ponto é então projetado para coordenadas **3D** no plano do campo, então sua coordenada **z** é acrescida da altura de um jogador; este novo ponto é então projetado de volta para o plano de imagem, representando a cabeça do jogador. Após, um segmento de reta é desenhado na imagem original ligando o ponto inicial marcado pelo usuário com o ponto porjetado como sendo a cabeça do jogador.

2.2.5 Função *mouse_callback* em *questao2.py*

Função responsável por tratar a interrupção do clique do mouse, provocada pelo usuário, em uma posição **x, y**. Este ponto é então projetado para coordenadas **3D** no plano do campo, então são calculados dois novos pontos representando as intersecções de uma reta paralela à linha de fundo com as duas linhas laterias do campo, tendo esta reta a propriedade de passar, também, pelo ponto selecionado; estes pontos são então projetos para o plano da imagem. Após, é desenhado, na imagem original, um segmento de reta ligando os dois pontos de intersecção encontrados, gerando assim a linha de impedimento virtual.

2.2.6 "Função" *main()*

Nos dois programas há uma preparação antes de ficar em laço esperando interrupções de mouse. Este fluxo acontece em uma espécie de *main*, que na verdade não existe. Neste trecho de código são feitos os seguintes passos:

1. A imagem é carregada e é criada uma janela devidamente dimensionada.
2. São lidos os pontos de corresnpondência (*hard-coded*).
3. É criada a matriz **A** referente ao sistema linear em 2.1.
4. O sistema é resolvido utilizando o método **SVD** da biblioteca *numpy*.
5. O resultado do sistema é utiizado para a montagem da matriz **P** de projeção da câmera.
6. É calculada a matriz **inversa** de **P** possibilitando a projeção inversa de coordenadas de imagem para coordenadas de mundo (após devida redimensionalização de **P** no exercício 1, excluindo-se a terceira dimensão **Z**). O cálculo de matriz inversa também é feito lançando mão da biblioteca *numpy*.
7. A imagem é exibida e o programa fica aguardando cliques do usuário.

3 Resultados Obtidos

Após todo o trabalho de implementação e seleção de pontos de calibração, podemos executar nosso código e observar os resultados obtidos. Nesta seção executamos os programas, geramos alguns exemplos e analisamos o quanto esses resultados fazem sentido visualmente.

3.1 Questão 1

Aqui analisamos a figura 3.4. Prestamos atenção no ângulo que os segmentos de reta formam com o plano do campo (ignora-se os Neymar, pois é difícil analisá-los). Principalmente próximo às traves da goleira é possível notal que há paralelismo dos segmentos com as mesmas. Também é notável como a discretização da imagem (pouquíssimos *pixels*) dificulta um pouco a avaliação, pois os segmentos de retas possuem poucos *pixels* o que gera um certo *aliasing*.

No geral nota-se que quando há uma visível inclinação do plano do campo em relação à imagem (mais ao canto inferior esquerdo) os segmentos de reta também tendem a ter esta inclinação, gerando um resultado bastante coerente.

Figura 3.4: Resultado obtido para a questão 1.

3.2 Questão 2

Aqui analisamos a figura 3.5. O resultado visual para esta questão é ainda mais fácil de ser analisado, dado que há muitas linhas paralelas à linha de fundo para comparar à gerada pelo nosso programa, ao contrário da questão 1 onde não havia outros jogadores para comparar aos nossos.

Para averiguar a corretude das linhas geredas, foram selecionado pontos sobre linhas reais já existentes no campo; e nota-se que a linha vermelha gerada é sempre quase que exatamente igual às linhas já existentes. É notável também que as linhas paralelas ao eixo **Y** não são perfeitas, visto que os ponto de intersecção com as linhas lateirais se move; olhando para a lateral superior na imagem: próximo à bandeira a intersecão está além da linha lateral, à medida que no outro lado da imagem ela já está aquém da memsa linha lateral.

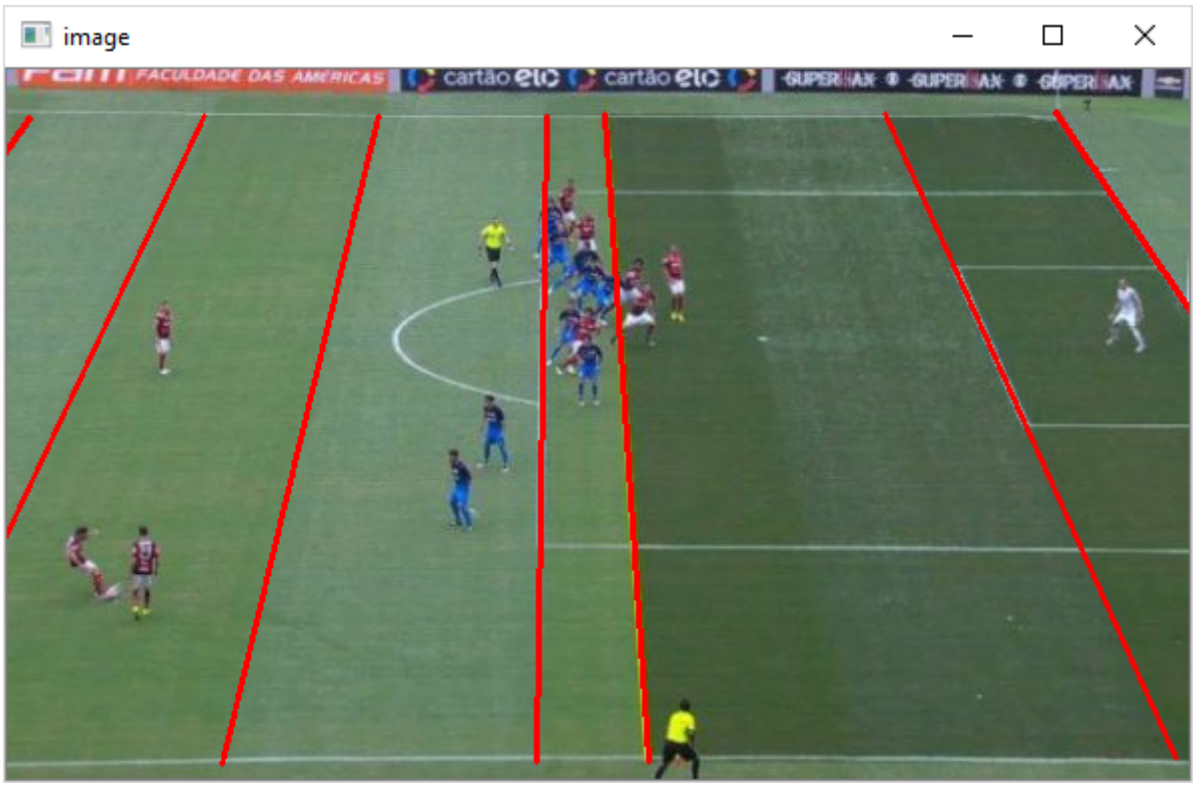


Figura 3.5: Resultado obtido para a questão 2.

4 Conclusão

Após implementada a técnica **DLT** para a calibração das duas câmeras e imagens fornecidas, notamos como é fácil e simples implementar uma técnica com resultados tão interessantes. Sentimos no couro como os algoritmos de *Visão Computacional* são sensíveis aos parâmetros de entrada; neste caso notamos como uma pequena diferença nos pontos de calibração escolhidos geravam resultados completamente distintos, inclusive pequenas alterações na posição dos *pixels* selecionados.

Por fim, ficamos extremamentes satisfeitos com os resultados obtidos, principalmente na questão 2 onde a corretude das linhas geradas é facilmente visível. Na questão 1, como verificar a corretude não era tão óbvio, adicionamos Neymar para os resultados ficarem mais interessantes.