Universidade Federal do Rio Grande do Sul
Instituto de Informática
INF 1030 - Fundamentos de Visão Computacional

→ **Alunos:**

Thiago Dal Moro 00185032
Andy Ruiz Garramones 00274705

→ **Objetivo do Trabalho**

O objetivo do trabalho de implementação 1 tem como resultado a implementação das projeções perspectivas através da calibração da câmera dado pelas equações (1) e (2), usando o método DLT (Direct Linear Transformation).

→ **Linguagem**

Utilizamos a linguagem PYTHON e a biblioteca principal de manipulação de imagens OPENCV.

→ **Contextualização**

A fins de entendimento, começamos o trabalho pela questão 2, que se trata de uma projeção perspectiva 2D - 2D e depois aprofundamos a ideia para a questão 1, que se trata da projeção 2D - 3D. Então recomendamos ler primeiro a explicação da questão 2 e logo depois a questão 1, tanto que por isso estão em ordem inversa apresentadas neste relatório.

→ **Medidas e mapeamento**
Foi usado a imagem do *pdf* da especificação do trabalho contendo as medidas do campo de futebol como base para selecionar os pontos do mundo. Assumimos que a largura do campo era de 68 metros e o resto apenas calculamos a partir da imagem com as medidas.

Para a questão 2 definimos que nosso ponto (0,0) do mundo era na linha lateral superior da imagem em linha reta à grande área. O eixo *y* aumentavam em direção à linha do gol.

Para a questão 1 definimos que nosso ponto (0,0) do mundo era no escanteio e aumentava em *y* em direção ao meio do campo.

→ **Questão 2**

Para começar a questão 2, precisávamos entender a relação dos pontos da imagem com os pontos do mundo, sua espécie de homografia.

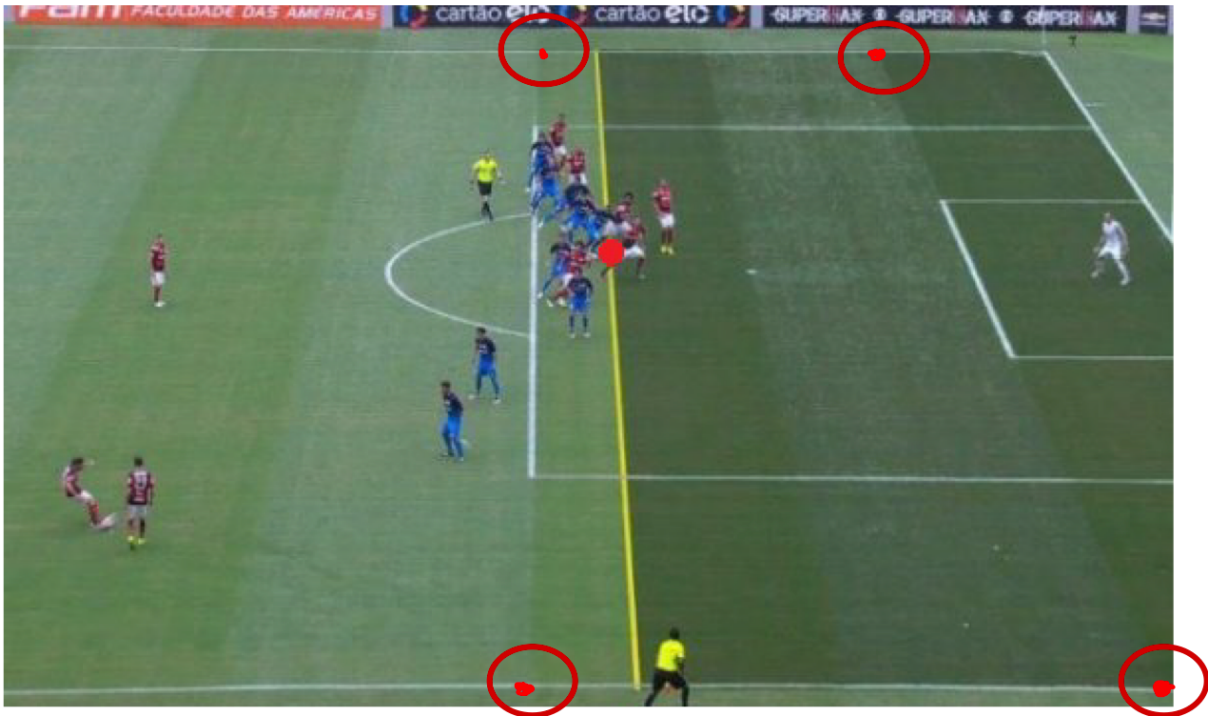
Na figura 2, passada na relatório, foi nos dado as dimensões do campo em metros.
Para tanto, precisávamos escolher pontos na imagem "Maracana2.jpg" que tivessem uma associação real com os pontos da coordenada de mundo das dimensões do campo.
Isso se dá pelo conceito da calibração da câmera, que então são incluídos no processo para determinar a matriz da câmera para a calibração da mesma.

Portanto, começamos por ler a imagem que nos interessava.

```
# Image read
img = cv2.imread('maracana2.jpg')
```

Posteriormente, escolhemos os pontos. (Foram feitos de forma manual).

```
# points
image_points = np.array([
        [268, 23],      #limite superior da área grande
        [264, 344],     #limite inferior da área grande
        [440, 24],      #limite superior área pequena
        [586, 346]])    #limite inferior área pequena
world_points = np.array([
        [0, 0],         #limite superior da área grande
        [68, 0],        #limite inferior da área grande
        [0,11],         #limite superior área pequena
        [68, 11]])      #limite inferior área pequena
```



Os pontos escolhidos foram esses pois, a partir desses pontos, conseguimos ter uma relação que nos era interessante para traçar as linha de impedimento em sequência.
A partir dos pontos escolhidos, temos a base para a implementação da DLT e encontrar nossa matriz de calibração.

Um ponto p = (xi, yi, zi) no mundo pode ser mapeado para um pixel q' = (ui, vi) na imagem através da transformação $q' = Pp$ sendo P a matriz de câmera 3x4.

A projeção em perspectiva de um ponto (xi, yi, zi) em coordenadas de mundo para um pixel (u, v) é dada por:

$$\begin{pmatrix} u_s \\ v_s \\ s \end{pmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix},$$

onde u = us/s e v = vs/s

Sabendo os valores correspondentes para p e q, o objetivo é descobrir os valores que formam a matriz de calibração da câmera. É feito um sistema de equações lineares a partir da relação da figura acima, cada par de correspondência gerando 2 equações lineares (uma para u e uma para v), obtendo 12 equações (número de valores presente em P).

$$\begin{bmatrix} x_1 & y_1 & z_1 & 1 & 0 & 0 & 0 & 0 & -u_1x_1 & -u_1y_1 & -u_1z_1 & -u_1 \\ 0 & 0 & 0 & 0 & x_1 & y_1 & z_1 & 1 & -v_1x_1 & -v_1y_1 & -v_1z_1 & -v_1 \\ x_2 & y_2 & z_2 & 1 & 0 & 0 & 0 & 0 & -u_2x_2 & -u_2y_2 & -u_2z_2 & -u_2 \\ 0 & 0 & 0 & 0 & x_2 & y_2 & z_2 & 1 & -v_2x_2 & -v_2y_2 & -v_2z_2 & -v_2 \\ & & & & & & \vdots & & & & & \\ x_n & y_n & z_n & 1 & 0 & 0 & 0 & 0 & -u_nx_n & -u_ny_n & -u_nz_n & -u_n \\ 0 & 0 & 0 & 0 & x_n & y_n & z_n & 1 & -v_nx_n & -v_ny_n & -v_nz_n & -v_n \end{bmatrix}$$

Acima está a matriz A gerada a partir das 12 equações para *n* pontos de correspondência. Para resolvê-la, ao invés de solucionar $A\boldsymbol{m} = 0$, minimizamos $\|A\boldsymbol{m}\|$ sujeto a ||m|| = 1 pois pode acontecer de nem todo sistema ser perfeito e conter alguns erros, por isso procuramos é resolver usando uma solução por mínimos quadrados.

Sendo assim, serão necessários no mínimo 6 pontos de correspondência para estimar todas as entradas na projeção em 3D.
Na transformação em 2D serão necessários 4 pontos de correspondência para achar as entradas em P pois se obtém apenas 8 equações para a projeção em 2D, já que se eliminam as coluna relacionadas a *Z* da matriz A)
Com isso, conseguimos criar a matriz de projeção a partir dos pontos escolhidos.

Execução:

A partir da matriz de projeção calculada escolhemos um ponto para ter como base o desenho da linha.

Por exemplo, escolhemos o ponto P = [80, 152].

Para o processo de ida (transformação de coordenadas de pixel para mundo), a equação é a descrita recentemente:

$$P^{-1}q = p$$

Onde P^{-1} é a matriz inversa da matriz de projeção calculada

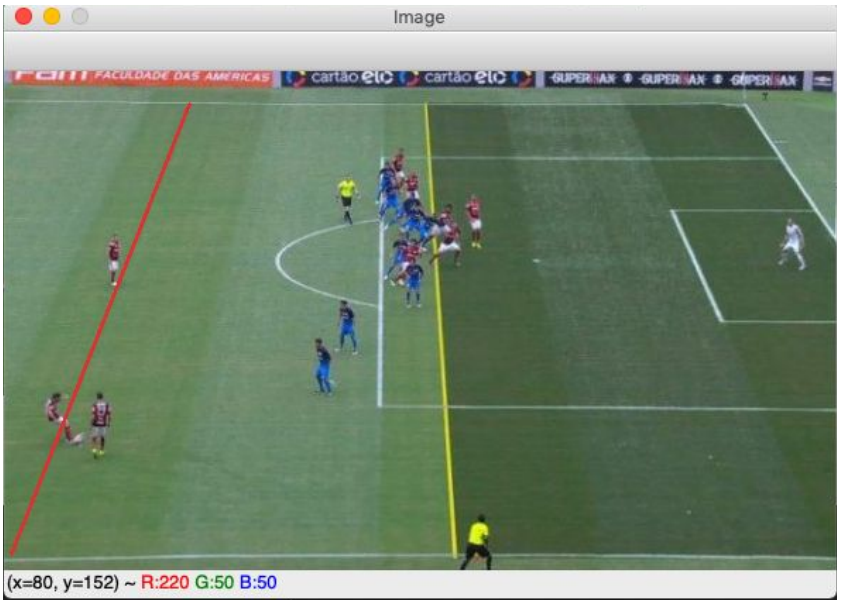
Após termos nosso ponto em coordenadas de mundo, mantemos o valor de *yi* constante e selecionamos os pontos (0, *yi*) e (68*, *yi*). Basicamente o que foi feito foi ir em linha reta, em paralelo à linha do gol, até a linha lateral superior e inferior dado o ponto no mundo escolhido. Com isso, teremos 2 pontos para traçar uma linha reta que passa pelo ponto selecionado e será paralelo à linha de fundo.
***** Foi considerado que o campo tem 68 metros de largura.

Para o processo de volta, (transformação de coordenadas de mundo para pixel), a equação é a descrita:

$$q = P\ p$$

Uma vez transformado os 2 pontos para coordenadas de pixel, basta traçar uma linha reta entre esses 2 pontos.

Resultado:



→ **Questão 1**

A questão 1 é uma extensão da questão 2. A diferença é que estamos falando de uma projeção perspectiva 3D. O que iremos considerar a mais será o eixo "Z".
Após selecionar os pontos e encontrar a matriz de projeção executamos o processo de ida usando a equação $P^{-1}q = p$

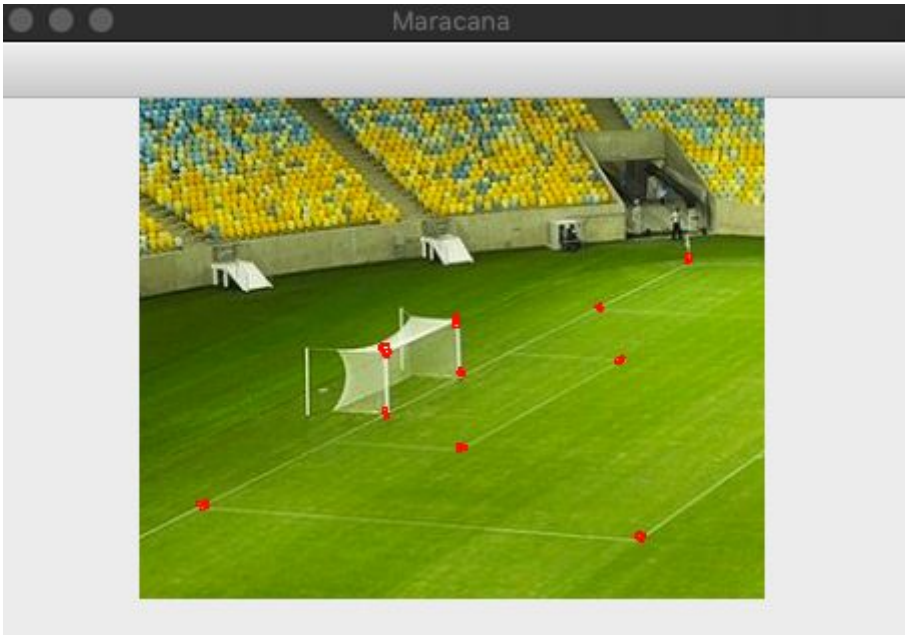
Agora, no momento que temos o ponto transformado de coordenadas de pixel para coordenadas de mundo, atribuímos sua altura como z = 0. Para encontrar o outro ponto, respectivo à cabeça do jogador que queremos desenhar, mantemos *xi* e *yi* constante e atribuímos a z 1.80 gerando o ponto (*xi, yi, 1.8*).

Agora só resta executar o processo de volta com esse novo ponto relativo à cabeça do jogador usando a equação $q = P\ p$

Por fim, teremos o ponto original que selecionamos na imagem e o ponto mapeado do mundo para a imagem que representa onde fica a cabeça do jogador. Então só desenhamos uma linha entre esses 2 pontos que representa um jogador.

Pontos escolhidos:
Pelo funcionamento da DLT, são necessários no mínimo 6 pontos de correspondência para se conseguir os 12 valores da matriz de projeção 3x4.
No entanto, um número maior de pontos de correspondência é recomendado, pois embora possa causar redundância é ela que nos possibilita maior precisão nos valores obtidos para P.

A seguir os 10 pontos escolhidos:




```
 # points
 image_points = np.array([
   [275,84],      # escanteio superior
   [227,107],     # borda exterior superior área pequena
   [159,141],     # trave superior ou esquerda
   [125,158],     # trave inferior ou direita
   [158,112],     # ângulo superior ou da esquerda da goleira
   [124,128],     # ângulo inferior ou direito da goleira
   [31,205],      # Canto inferior direito da grande área
   [250,222],     # Canto inferior direito da pequena área
   [241,132],     # borda externa superior ou esquerda da pequena área
   [160,177]      # borda externa inferior ou direita da pequena área
])


world_points = np.array([
   [0, 0, 0],     # escanteio superior
   [13.84, 0, 0],     # borda exterior superior área pequena
   [30.34, 0, 0],     # trave superior ou esquerda
   [37.66, 0, 0],     # trave inferior ou direita
   [30.34, 0, 2.44],  # ângulo superior ou da esquerda da goleira
   [37.66, 0, 2.44],  # ângulo inferior ou direito da goleira
   [54.16, 0, 0],     # Canto inferior direito da grande área
   [54.16, 16.5, 0],  # Canto inferior direito da pequena área
   [24.84, 5.5, 0],   # borda externa superior ou esquerda da pequena área
   [43.16, 5.5, 0]    # borda externa inferior ou direita da pequena área
 ])
```

Resultado:

→ **Aprendizados:**

1. Se torna muito mais simples a execução / implementação do trabalho quando se entende com detalhes a teoria por trás da aplicação.
2. Interessante em relação a ver que os conceitos vistos em aula são aplicados na realidade e possuem uma grande relevância.
3. Conceitos matemáticos e de processamento de imagens aplicados tornam-se muito mais fáceis de entender e interessantes para ficar explorando.

REFS:

https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.linalg.svd.html
https://en.wikipedia.org/wiki/Camera_matrix
https://en.wikipedia.org/wiki/Homography

Links para entender melhor os conceitos aplicados:
https://math.stackexchange.com/questions/494238/how-to-compute-homography-matrix-h-from-corresponding-points-2d-2d-planar-homog
https://www.youtube.com/watch?v=bRyUvt0ZnU0