Sistemas de Visión Artificial

Práctica 6. Geometría de la cámara

Nombre: Juan Casado Ballesteros

Fecha: 28 de noviembre 2019

Calibración

Utilizaremos las imágenes proporcionadas del tablero de calibración para calibrar la cámara que se utilizó para capturarlas.
Para calibrarla cargaremos dichas imágenes en la utilidad `cameraCalibrator` indicando dónde se encuentran las imágenes con las que deseamos calibrar y el tamaño real de los cuadrados del tablero.

```
file = 'checkerboard_pattern';
square_size = 21;
cameraCalibrator(file, square_size);
```

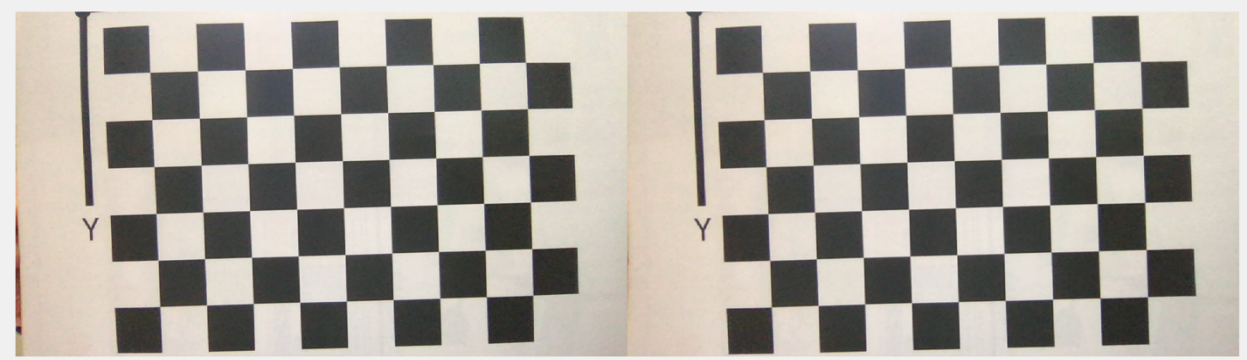
Cuando la calibración finalice podremos guardar los resultados en una variable del workspace o en un archivo .mat. Es recomendable usar un archivo .mat para no tener que repetir todo el proceso de la calibración en el futuro.
Podremos cargar dicho archivo .mat en una variable del workspace.

```
sesion = 'calibrationSession.mat';
cameraCalibrator(sesion);
```

Utilizamos el siguiente código para visualizar las diferencias entre la imagen original y esta procesada en base a la calibración realizada sobre la cámara.
Esto mismo puede hacerse desde la utilidad `cameraCalibrator`.

```
image_num = 1;
image_name = sprintf('checkerboard_pattern/image%d.png',image_num);
image = imread(image_name);
imageUndistorted = undistortImage(image,cameraParams);
imshowpair(image,imageUndistorted,'montage')
```

Puede verse que, en la imagen de la izquierda, la original los cuadrados no son completamente rectos mientras que en la de la derecha lo son mucho más.



Mostraremos ahora las matrices de corrección obtenidas intrínsecas de la cámara y extrínsecas de una de las imágenes.

```
cameraParams.IntrinsicMatrix
  953.5691      0            0
  2.2066        949.2276     0
  635.0427      355.1362     1.0000
```

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & sf_x & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix}$$

Esta matriz es propia de la cámara y común a todas las fotos capturadas con ella. En ella se realiza una operación de proyección desde un plano intermedio común a todas las imágenes y una traslación del centro de este plano al de las imágenes (esquina superior izquierda). Además, se realiza un cambio de unidades de las del mundo real a pixeles.
Por último, aunque previo al cambio de unidades realizamos también una corrección de las distorsiones radiales y tangenciales que nuestra lente pueda tener. Así como eliminamos el skew.
Para que se calcule el skew lo debemos indicar expresamente al calibrar.
La matriz obtenida está rotada respecto de la que ponemos como ejemplo explicativo en la foto.

```
image_num = 1;
cameraParams.RotationMatrices(:,:,image_num)
0.9995   -0.0300   -0.0069
0.0300    0.9995    0.0075
0.0067   -0.0077    0.9999
```

$$R = R_X(\alpha)\, R_Y(\beta)\, R_Z(\gamma)$$

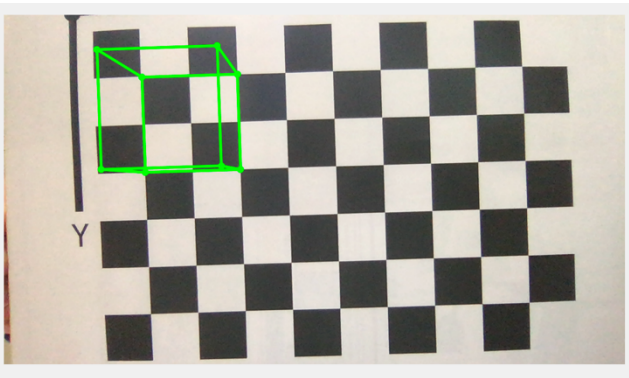
$$\begin{bmatrix} wX_c \\ wY_c \\ wZ_c \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Esta matriz se corresponde con las rotaciones que de se ben hacer para trasladar los puntos del plano real al plano intermedio paralelo al plano de imagen. Son la multiplicación de estas matrices sin la última fila ni la última columna.

```
cameraParams.TranslationVectors(image_num,:)
-74.4958   -48.4386   203.3346
```

$$\begin{bmatrix} wX_c \\ wY_c \\ wZ_c \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_X \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Traslaciones que realizar para llevar el origen del plano real al origen del plano mencionado anteriormente. El que está delante del plano de imagen.
Estas dos últimas matrices dependen de la imagen capturada y no solo de la cámara por ello cada una de estas matrices se corresponde con una imagen concreta.



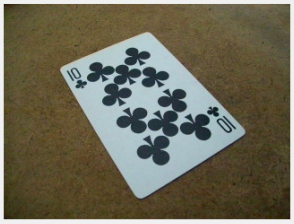
Por último, dibujaremos un cubo utilizando medidas del mundo real y transportando estas a la imagen usando las matrices de transformación entre ambos planos. El cubo debe se tener como arista una longitud de dos cuadrados y un vértice en el punto (0, 0).

```
imagePoints = worldToImage(cameraParams,...
    cameraParams.RotationMatrices(:,:,image_num),...
    cameraParams.TranslationVectors(image_num,:),...
    [0 0 0;
     1 0 0;
     1 1 0;
     0 1 0;
     0 0 -1;
     1 0 -1;
     1 1 -1;
     0 1 -1]*2*square_size);
image_name = sprintf('checkerboard_pattern/image%d.png',image_num);
 image = imread(image_name);
 figure;imshow(image);hold on;
 %Cuadrado superior
 plot([imagePoints(1, 1),imagePoints(2, 1)],...
      [imagePoints(1, 2),imagePoints(2, 2)] , 'g*-', 'LineWidth', 3);
 plot([imagePoints(2, 1),imagePoints(3, 1)],...
      [imagePoints(2, 2),imagePoints(3, 2)] , 'g*-', 'LineWidth', 3);
 plot([imagePoints(3, 1),imagePoints(4, 1)],
      [imagePoints(3, 2),imagePoints(4, 2)] , 'g*-', 'LineWidth', 3);
 plot([imagePoints(4, 1),imagePoints(1, 1)],
      [imagePoints(4, 2),imagePoints(1, 2)] , 'g*-', 'LineWidth', 3);
 %Cuadrado inferior
 plot([imagePoints(5, 1),imagePoints(6, 1)],
      [imagePoints(5, 2),imagePoints(6, 2)] , 'g*-', 'LineWidth', 3);
 plot([imagePoints(6, 1),imagePoints(7, 1)],
      [imagePoints(6, 2),imagePoints(7, 2)] , 'g*-', 'LineWidth', 3);
 plot([imagePoints(7, 1),imagePoints(8, 1)],
      [imagePoints(7, 2),imagePoints(8, 2)] , 'g*-', 'LineWidth', 3);
 plot([imagePoints(8, 1),imagePoints(5, 1)],
      [imagePoints(8, 2),imagePoints(5, 2)] , 'g*-', 'LineWidth', 3);
 %Aristas verticales
 plot([imagePoints(1, 1),imagePoints(5, 1)],
      [imagePoints(1, 2),imagePoints(5, 2)] , 'g*-', 'LineWidth', 3);
 plot([imagePoints(2, 1),imagePoints(6, 1)],
      [imagePoints(2, 2),imagePoints(6, 2)] , 'g*-', 'LineWidth', 3);
 plot([imagePoints(3, 1),imagePoints(7, 1)],
      [imagePoints(3, 2),imagePoints(7, 2)] , 'g*-', 'LineWidth', 3);
 plot([imagePoints(4, 1),imagePoints(8, 1)],
      [imagePoints(4, 2),imagePoints(8, 2)] , 'g*-', 'LineWidth', 3);
 hold off;
```

Homografía

Deseamos transformar por homografía la imagen que se nos proporciona en otra. Para poder hacerlo el objeto a transformar debe ser plano.
Obtenemos 4 puntos de referencia en un orden concreto para que nos sirvan de guía para hacer la transformación.

```
I = imread('card.jpg');
[x, y , P] = impixel(I);
srcPoints = [x y];
```



Creamos ahora un vector de 4 puntos en el mismo orden que antes los seleccionamos para a partir de ellos y de los anteriores obtener la matriz de transformación.
Usaremos dos sets de puntos distintos, uno para transformar la imagen dejando la carta en vertical y otro para dejarla en horizontal.

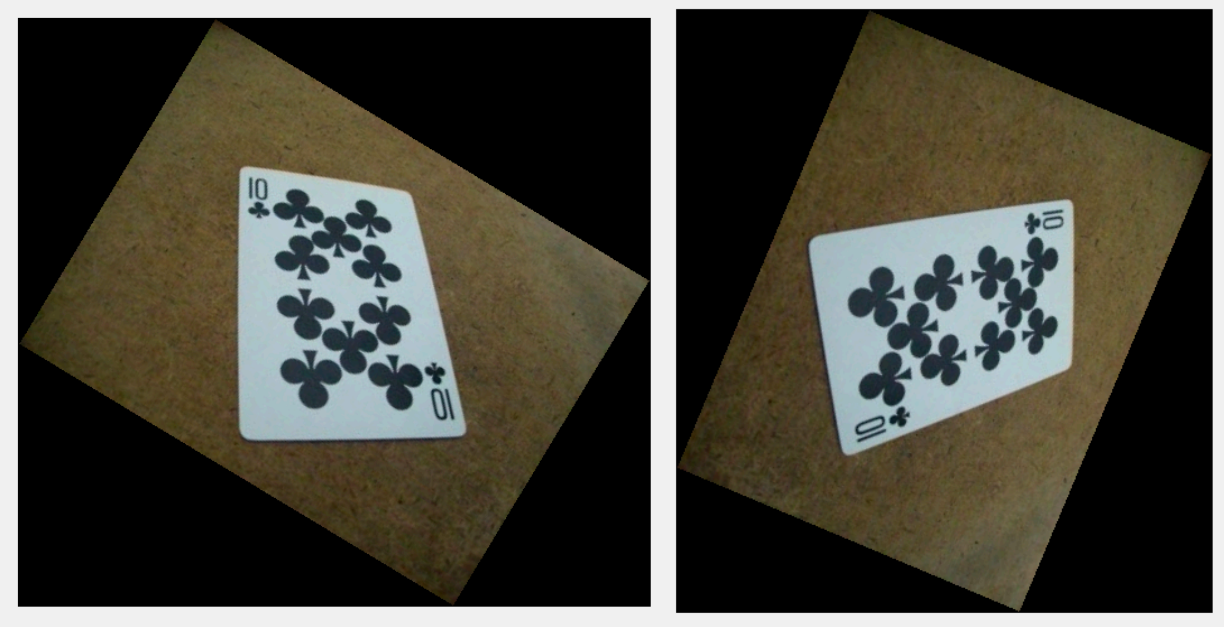
```
scale = 2;
card_w = 67*scale;
card_h = 100*scale;
hPoints = [0      0;
           0      card_w
           card_h card_w
           card_h 0];
vPoints = [0      0;
           card_h 0;
           card_h card_w
           0      card_w];
```

Obtenemos las matrices de transformación que nos permitirán cambiar la prespectiva de la imagen por homografía.

```
htransform = estimateGeometricTransform(srcPoints, hPoints, 'similarity');
vtransform = estimateGeometricTransform(srcPoints, vPoints, 'similarity');
```

Aplicamos las transformaciiones a la imagen y observamos los resultados

```
htransformedI = imwarp (I, htransform);
figure;imshow(htransformedI);
vtransformedI = imwarp (I, vtransform);
figure;imshow(vtransformedI);
```



Debemos advertir de que en este caso la calidad de los resultados obtenidos depende de la selección de puntos. Si esta es malintencionada la transformación no funcionará adecuadamente.