Informe de prácticas
2

Alejandro García Montoro
agarciamontoro@correo.ugr.es

21 de noviembre de 2015

1. Estimación de una homografía

El objetivo de este ejercicio es estimar la homografía que mejor aproxima dos conjuntos de puntos en correspondencia tomados de sendos planos proyectivos. La idea, entonces, es rectificar uno de ellos para llevarlo al plano del otro. Antes de ver todo el código, veamos matemáticamente qué tenemos que hacer.

Sean x e y dos puntos en coordenadas homogéneas que queremos poner en correspondencia a través de una homografía H. Se tiene que cumplir por tanto la ecuación

$$y = Hx \tag{1}$$

Como vemos en [Szelinski, p.37, eq 2.21] o en *Homography estimation*[1], al ser H una homografía, y normalizando el punto y, de (1) obtenemos las dos siguientes ecuaciones:

$$y_1 = \frac{H_{1,1}x_1 + H_{1,2}x_2 + H_{1,3}}{H_{3,1}x_1 + H_{3,2}x_2 + H_{3,3}} \tag{2}$$

$$y_2 = \frac{H_{2,1}x_1 + H_{2,2}x_2 + H_{2,3}}{H_{3,1}x_1 + H_{3,2}x_2 + H_{3,3}} \tag{3}$$

donde $H_{i,j}$ son los coeficientes de la matriz H; es decir, nuestras incógnitas. Además, hemos supuesto normalizado el vector x; es decir, $x_3 = 1$ —si no lo está, basta dividir entre x_3 todas las coordenadas—.

De (2) y (3), reorganizando las igualdades llegamos a las siguientes ecuaciones lineales:

$$ph = 0 \tag{4}$$

$$qh = 0 \tag{5}$$

[1] `http://cseweb.ucsd.edu/classes/wi07/cse252a/homography_estimation/homography_estimation.pdf`

donde

$$p = (-x_1 \quad -x_2 \quad -1 \quad 0 \quad 0 \quad 0 \quad y_1x_1 \quad y_1x_2 \quad y_1)$$
$$q = (0 \quad 0 \quad 0 \quad -x_1 \quad -x_2 \quad -1 \quad y_2x_1 \quad y_2x_2 \quad y_2)$$
$$h = (H_{1,1} \quad H_{1,2} \quad H_{1,3} \quad H_{2,1} \quad H_{2,2} \quad H_{2,3} \quad H_{3,1} \quad H_{3,2} \quad H_{3,3})^T$$

Las ecuaciones (4) y (5) las podemos escribir de forma matricial como

$$Ah = 0 \tag{6}$$

donde

$$A = \begin{pmatrix} p \\ q \end{pmatrix}$$

Pero nuestro objetivo es resolver (6) con un número N arbitrario de puntos —al menos cuatro para determinar una solución—, de manera que la matriz A se convertirá en

$$A = \begin{pmatrix} p_1 \\ q_1 \\ p_2 \\ q_2 \\ \vdots \\ p_N \\ q_N \end{pmatrix}$$

El problema de resolver sistemas de ecuaciones del tipo (6) está muy estudiado, y puede ser abordado a partir de la descomposición en valores singulares —SVD por sus siglas en inglés— de la matriz A:

$$A = USV^T = \sum_{i=0}^{9} \sigma_i u_i v_i^T$$

La mejor solución encontrada para h —exacta si $\sigma_9 = 0$ y aproximada si $\sigma_9 > 0$— es el último vector columna de la matriz V.

Tenemos por tanto un algoritmo claro para implementar en la función que queremos diseñar:

1. Definir dos conjuntos de puntos en correspondencia.
2. Para cada par de puntos, definir los coeficientes de p y q.
3. Construir la matriz A con todos los p y q calculados anteriormente.
4. Hacer la descomposición en valores singulares de la matriz A.
5. Reorganizar el último vector columna de V en los coeficientes de la matriz H.

Aclarado el algoritmo que vamos a seguir, veamos el código que lo implementa.

1.1. Selección de puntos

Se han seleccionado dos tipos de conjuntos de puntos en correspondencia entre las imágenes *Tablero1.jpg* y *Tablero2.jpg*.

Primero, se han determinado diez puntos de cada imagen de manera que estuvieran suficientemente repartidos por todo el tablero. La columna izquierda de la siguiente lista muestra los puntos de la imagen *Tablero2.jpg*; a la derecha, sus correspondencias en la imagen *Tablero1.jpg*:

```
1     Point2f(147, 13) --> Point2f(156, 47)
2     Point2f(504, 95) --> Point2f(532, 11)
3     Point2f(432, 444) --> Point2f(527, 466)
4     Point2f(75 , 388) --> Point2f(137, 422)
5     Point2f(227, 133) --> Point2f(238, 139)
6     Point2f(396, 169) --> Point2f(363, 133)
7     Point2f(362, 338) --> Point2f(413, 337)
8     Point2f(192, 308) --> Point2f(229, 327)
9     Point2f(286, 224) --> Point2f(308, 219)
10    Point2f(304, 251) --> Point2f(331, 245)
```

Además, se ha determinado otro conjunto de diez puntos en correspondencia, esta vez todos en las tres casillas de la esquina superior izquierda:

```
1     Point2f(148,14) --> Point2f(156,47)
2     Point2f(174,19) --> Point2f(177,45)
3     Point2f(198,24) --> Point2f(198,43)
4     Point2f(223,30) --> Point2f(221,40)
5     Point2f(141,40) --> Point2f(155,72)
6     Point2f(168,46) --> Point2f(176,70)
7     Point2f(191,50) --> Point2f(197,68)
8     Point2f(217,56) --> Point2f(219,66)
9     Point2f(136,63) --> Point2f(153,95)
10    Point2f(162,69) --> Point2f(174,93)
```

1.2. Estimación de la homografía

La función de estimación de la homorgafía toma un vector de pares de puntos e implementa el algoritmo visto en la introducción. Veamos el código:

```
1  Mat Image::findHomography(vector< pair<Point2f,Point2f> > matches){
2    // Build the equations system. See
         http://sl.ugr.es/homography_estimation
3    Mat mat_system, sing_values, l_sing_vectors, r_sing_vectors;
4
5    for (unsigned int i = 0; i < matches.size(); i++) {
6      Point2f first = matches[i].first;
7      Point2f second = matches[i].second;
8
```

```
9       float coeffs[2][9] = {
10          { -first.x, -first.y, -1., 0., 0., 0., second.x*first.x,
                second.x*first.y, second.x },
11          { 0., 0., 0., -first.x, -first.y, -1., second.y*first.x,
                second.y*first.y, second.y }
12      };
13
14      mat_system.push_back( Mat(2, 9, CV_32FC1, coeffs) );
15  }
16
17  // Solve the equations system using SVD decomposition
18  SVD::compute( mat_system, sing_values, l_sing_vectors,
        r_sing_vectors, 0 );
19
20  Mat last_row = r_sing_vectors.row(r_sing_vectors.rows-1);
21
22  cout << "sigma_9 for own findHomography: " <<
        sing_values.at<float>(8,0) << endl;
23
24  return last_row.reshape(1,3);
25 }
```