

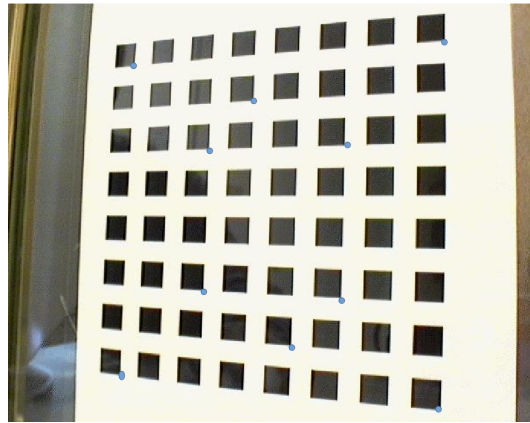
# VC: Informe práctica 2

Gustavo Rivas Gervilla

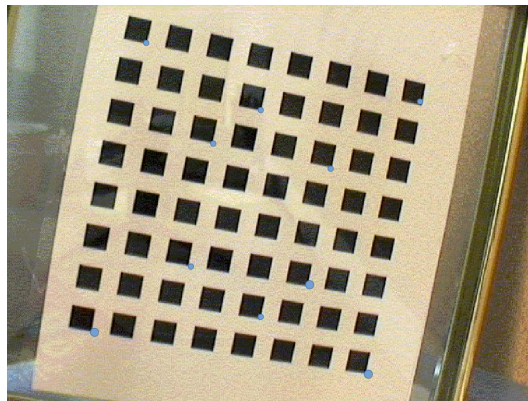


## 1 Estimación de la homografía

El código para la estimación de la homografía es muy sencillo si tenemos en cuenta lo que hemos visto en teoría junto con el artículo sobre mínimos cuadrados subido a la plataforma DECSAI, para ello lo primero que hemos de hacer es seleccionar 10 puntos en correspondencias en las dos imágenes de modo que estos puntos estén suficientemente distribuidos para que el cálculo de la homografía sea bueno, nosotros hemos seleccionado los siguientes puntos aproximadamente:



(a) Puntos elegidos en Tablero 1



(b) Puntos elegidos en Tablero 2

Figure 1: Puntos seleccionados en cada imagen en azul.

Una vez que tenemos estos puntos no tenemos más que introducirlos en nuestro código para ello hemos creado una clase muy simple llamada Punto que tiene simplemente dos floats que serán las coordenadas (podríamos usar la clase Point2f de OpenCV como haremos más adelante).

Ahora con estos puntos tenemos que formar la matriz que aparece en las transparencias que es la siguiente:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ & & & & & & & & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n & -y'_n \end{bmatrix}$$

Esto se hace con un bucle for muy básico y que por tanto no lo vamos a mostrar ya que es simplemente seguir la estructura que aquí se marca, creando en nuestro caso una matriz de 20x9, aunque al crear la matriz dónde almacenaremos los coeficientes lo hacemos inicializándola a cero para evitar tener que poner los ceros que tiene la matriz en el bucle, haciendo el código algo más legible (podemos ver el bucle en la función obtenerMatrizCoeficientes del main.cpp).

Ahora de acuerdo con el artículo sobre mínimos cuadrados mencionado no necesitamos más que calcular la descomposición SVD de la matriz y tomar la última columna de V, esto se hace con el siguiente código, usando las funciones implementadas en OpenCV (el código completo es el de la función obtenerMatrizTransformacion de main.cpp):

```
//Obtenemos la descomposicion SVD de la matriz de coeficientes ,
//la matriz que nos interesa es la vT:
SVD::compute(A, w, u, vt);

Mat H = Mat(3, 3, CV_32F);

//Construimos la matriz de transformacion con la ultima columna de v
//o lo que es lo mismo la ultima fila de vT:
for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
        H.at<float>(i, j) = vt.at<float>(8, i * 3 + j);
```

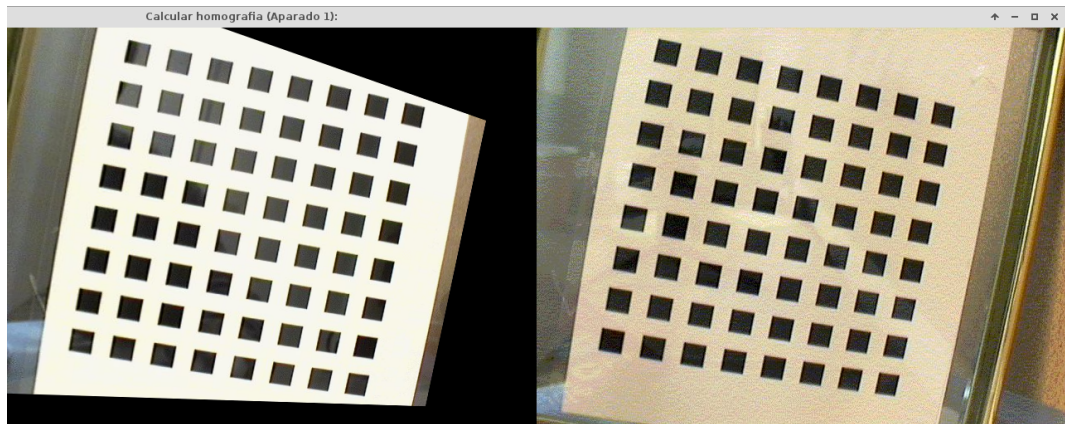
donde A es la matriz de coeficientes calculada previamente como hemos descrito arriba. Yo había pensado en resolver el sistema del que A es matriz de coeficientes ya que OpenCV tiene la función solve pero no podemos hacer esto ya que entre otras cosas no podemos codificar usando esta función la condición de que la solución encontrada tenga norma 1, con lo que podríamos obtener fácilmente la solución trivial algo que evidentemente no nos interesa.

Y ahora no tenemos más que aplicarle la homografía a Tablero1 (que es la que hemos considerado como imagen de partida) para obtener algo parecido a Tablero 2. Esto lo hacemos con el siguiente código:

```
Mat tablero1_transformada;

warpPerspective(tablero1, tablero1_transformada, H, Size(tablero2.cols, tablero2.rows));
```

Y aquí tenemos el resultado obtenido comparado con la imagen Tablero2:



Como podemos ver "la posición del tablero" es muy similar a la del Tablero2, ahora bien lo primero que apreciamos son los cortes negros producto de los puntos que caen fuera del rango de la imagen de destino por la homografía y en segundo lugar notamos lo que no podemos hacer simplemente con geometría y es el color de la imagen, es distinto, nosotros sólo estamos mapeando puntos de la imagen sobre otra no estamos cambiando el valor de dichos píxeles. Además de que por supuesto el resultado no es exacto.

Veamos ahora lo que obtenemos cuando los puntos en correspondencias seleccionados se toman en 3 cuadrados contiguos de una esquina, es esperable que el resultado sea peor puesto que la información que le damos no es tan buena ya que los puntos están más concentrados y por tanto el área de aplicación que "conocemos" de la homografía es más pequeña dando origen posiblemente a más ambigüedades. Aquí tenemos el resultado:

## 2 Búsqueda de KeyPoints con los detectores BRISK y ORB