PROYECTO: Calibración de una cámara
(para entregar por parejas en 7-10 días)

En este proyecto presentaremos los fundamentos del proceso de calibración de una cámara para obtener sus parámetros intrínsecos a partir de fotografías tomadas de un cierto patrón. Por simplicidad nosotros trabajaremos con una única fotografía, lo que limitará el número de parámetros de calibración que podremos determinar. Por ello inicialmente nos centraremos en hallar el valor de la focal f usada, suponiendo que el resto de los parámetros intrínsecos toman sus valores por defecto.

Planteamiento del problema:

Para saber donde se proyecta un punto 3D sobre una foto lo primero es cambiar a coordenadas de la cámara de acuerdo a la expresión:

$$\overline{X}_{cam} = R \cdot \left(\overline{X} - \overline{X}_0 \right)$$

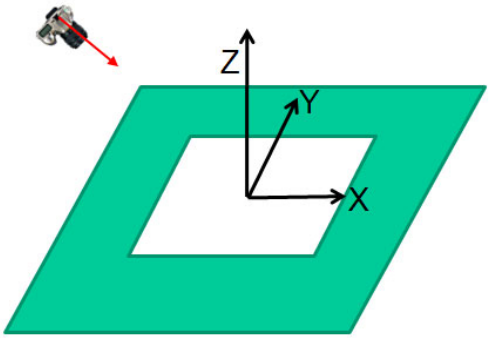
donde X es la posición del punto en las coordenadas de la escena, X0 es la posición de la cámara en los ejes de la escena y R es la matriz de rotación que representa el giro entre los ejes de la escena y los ejes de la cámara. Es habitual reescribir esta relación como:

$$\overline{X}_{cam} = R \cdot \overline{X} + \overline{t} \quad \text{con} \quad \overline{t} = -R \cdot \overline{X}_0$$

Expresada matricialmente con coordenadas homogéneas:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{cam} = \begin{pmatrix} & R & & \left| \overline{t} \right. \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

La calibración se basa en tomar una o más fotografías de un patrón en una superficie plana. Sin pérdida de generalidad podemos definir el plano XY de las coordenadas escena como el plano del papel (ver figura). En ese caso todos los puntos del patrón tendrán coordenada Z=0, y la expresión queda:



$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{cam} = \begin{pmatrix} & Q & \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \quad \text{con} \quad Q = \begin{pmatrix} \overline{r}_1 & \overline{r}_2 & \overline{t} \end{pmatrix}$$

donde $\underline{r_1}$ y $\underline{r_2}$ son las 2 primeras columnas de la matriz de giro R (la 3ª columna de R desaparece al estar siempre multiplicada por la coordenada Z=0).

Tras el paso a coordenadas cámara se aplica una segunda matriz K que incorpora los efectos de los parámetros intrínsecos de la cámara (focal f, posición u0, v0 del eje óptico, etc.). Finalmente hacemos la proyección final (dividiendo por la tercera componente) para obtener la posición en píxeles (u,v) sobre la foto:

$$\begin{pmatrix} U \\ V \\ W \end{pmatrix} = \begin{pmatrix} & K & \end{pmatrix} \begin{pmatrix} & Q & \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} = \begin{pmatrix} & H & \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \Rightarrow \begin{cases} u = U/W \\ v = V/W \end{cases}$$

Vemos que el problema se ha reducido a una transformación 2D entre puntos de un plano (el papel) con coordenadas X,Y (medidas en mm) proyectadas en otro plano (la foto) con coordenadas u,v (en píxeles). En particular es una transformación de tipo proyectiva, que se puede determinar si disponemos de las coordenadas de (al menos) 4 puntos en el plano de partida (X,Y en el papel) y en el de llegada (u,v en la foto). Determinar esta matriz H será nuestro primer objetivo de esta práctica.

1) Determinación de la matriz H

Cargad la imagen contenida en 'malla.jpg' y visualizarla. Es una foto de una hoja de papel con una malla impresa de 6 x 10 cuadraditos. La separación entre líneas (lado de los cuadrados) en el papel es la misma en el sentido horizontal (DX=20 mm) y vertical (DY=20 mm). En total hay 7x11 cruces entre líneas y nuestro objetivo es obtener las coordenadas (en píxeles) de todos esos cruces. Como un primer paso usaremos las 4 esquinas más exteriores pinchando sobre ellos para obtener sus coordenadas aproximadas.

El esqueleto de vuestro programa será el script lab2.m. Durante la práctica iremos añadiendo código al script y añadiendo/completando nuevas funciones para resolver los diferentes apartados. Si ejecutáis el script lab2.m veréis que carga la imagen de la malla y la visualiza. Luego entra en un bucle para pinchar con el ratón en 4 puntos: en cada paso se vuelcan las coordenadas elegidas y se pinta un círculo rojo en el punto seleccionado.

Para no tener que ser muy precisos al marcar los puntos escribiremos un código que detecte el punto de cruce exacto a partir de nuestra primera aproximación. Para refinar los puntos hay que completad la función: `function [xm,ym]=refinar(x,y)`

La función recibe una posición preliminar (x,y) y devuelve la posición final mejorada (xm,ym). El algoritmo trabajará sobre una imagen auxiliar (aux). Esta imagen (en blanco y negro y del mismo tamaño que la original) ha sido calculada en el script principal y está declarada de tipo global, para poder verse desde nuestra función. En cada píxel de aux se guarda el valor mínimo de la tripleta RGB del píxel original y luego se aplica un filtro promedio. Este tipo de filtros los veremos en otro tema, pero si visualizáis la imagen aux veréis que es esencialmente una versión en niveles de gris y suavizada de la imagen original.

El código para determinar de forma más exacta el cruce será idéntico al usado en la aplicación de tracking del proyecto anterior. Dentro de la función se define el radio de la zona a explorar (R=50) y se crean dos matrices dx, dy con las coordenadas X,Y de la zona referidas a su centro (desde -R hasta R en ambos ejes). El código a completar consiste en:

1. Redondear (round) las coordenadas (x,y) de entrada para obtener el píxel de la imagen (números enteros) alrededor del cual se va a buscar el cruce. De la imagen auxiliar aux extraer la subimagen S de +/-R píxeles centrada en ese píxel. **Recordad que las filas de una imagen en MATLAB corresponden a las coordenadas Y y las columnas a las X. No os olvidéis tampoco de convertir S a double para hacer las cuentas del siguiente punto.**

2. Determinar el valor mínimo m de S y calcular sus diferencias con respecto a ese valor mínimo d=abs(S-m). Luego haced $w=e^{-50d}$. Finalmente dividir w por la suma de todos sus valores para obtener una matriz de pesos.

3. Construir la matrix x+dx (con las coordenadas de los píxeles de la subimagen referidas a la imagen completa) y multiplicarla (punto a punto) por la matriz de pesos w. La suma de los valores de la nueva matriz será la estimación de la coordenada x del cruce (valor devuelto en xm, puede que no sea entero).

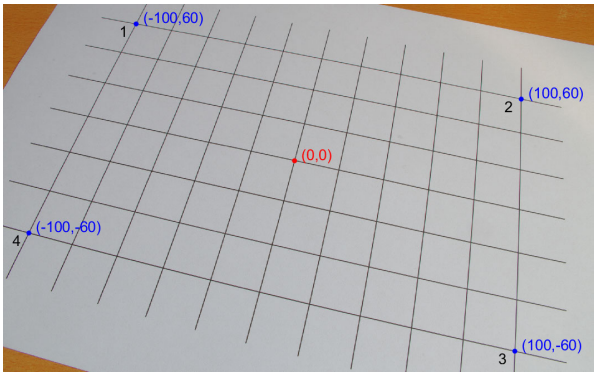
4. Repetir con las coordenadas y's y obtener la nueva coordenada ym mejorada.

Una vez completada la función, modificar el bucle de lab2.m llamando en cada paso a la función refinar() con los valores de (x,y) obtenidos con el ratón. Guardar los resultados "refinados" en u(k) y v(k) respectivamente. Modificar la llamada a fprintf para que vuelque las nuevas posiciones mejoradas junto con las originales y repetir el comando plot para pintar también la nueva posición (ahora en color verde). Si todo funciona deberías ver los nuevos puntos verdes correctamente centrados en los cruces. Obviamente si pincháis con el ratón a más de R píxeles del cruce el algoritmo no funcionará. Adjuntad el volcado con las posiciones iniciales y finales de las esquinas, así como un zoom de la 1ª esquina donde se aprecie el punto inicial (rojo) y su mejora (verde). Adjuntad el código de vuestra función refinar.m.

Una vez que tenéis guardadas las posiciones de las 4 esquinas en u,v, el comando `plot([u u(1)],[v v(1)],'b');` dibujará el rectángulo formado por las 4 esquinas. Superponerlo sobre la imagen. ¿Se superpone exactamente el rectángulo dibujado sobre la malla del papel? Adjuntad una captura de un zoom en una zona donde se aprecien diferencias. ¿A qué puede ser debido?

Conocidas las coordenadas (u,v) de las 4 esquinas en la imagen falta establecer sus coordenadas XY en el papel. Si el centro de la malla es el origen y sabiendo que cada cuadrado son 20 mm, las coordenadas X,Y de las esquinas (ver figura adjunta) serían:

X = -100, 100, 100, -100
Y = 60, 60, -60, -60



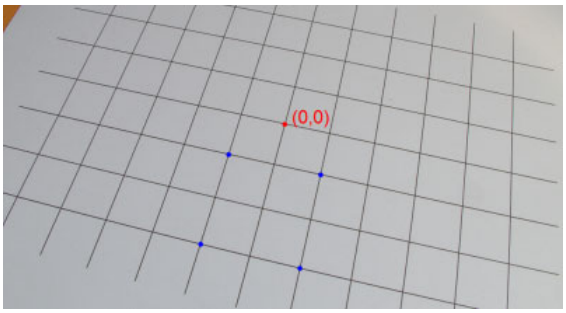
Guardar estas coordenadas en dos vectores fila (1x4) X e Y. El orden es importante ya que se asume que habéis marcado los puntos en el orden 1-2-3-4 indicado en la figura. Si habéis usado otro orden tendréis que alterar también el orden de las coordenadas en X e Y.

Conocidas las 4 coordenadas en papel (X,Y en mm) y en la imagen (u,v en píxeles) ya podemos hallar la matriz proyectiva H que transforma unas en otras. En un tema posterior veremos cómo se calculan estas transformaciones dados los puntos de origen (X,Y) y de destino (u,v). Como todavía no sabemos hacerlo, en esta práctica os daré yo la función a usar **`H=fc_get_H(X,Y,u,v)`**. La función recibe los vectores con las coordenadas de origen (X,Y) y destino (u,v) de los cuatro puntos y devuelve la matriz H que transforma unos en otros:

$$\begin{pmatrix} U \\ V \\ W \end{pmatrix} = \begin{pmatrix} & H & \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \Rightarrow \begin{cases} u = U/W \\ v = V/W \end{cases}$$

Calcular H y usar la función **`volcar_H(H)`** para mostrar los valores de la matriz así obtenida. Adjuntad el resultado. Guardar la matriz H con el comando **`save H1 H;`**
Así no tendréis que volver a marcar los puntos cada vez que ejecutéis el script. Haciendo **`load H1`** podéis recuperar la matriz H y usarla en el resto de los apartados.

Obviamente la transformación H obtenida dependerá de los 4 puntos escogidos. En este caso hemos usado las esquinas de la malla, pero podríamos haber usado otros puntos. Volver a ejecutar el código marcando ahora los 4 puntos indicados en azul en la figura adjunta. Recordad usar los correspondientes valores de las coordenadas X e Y para esos puntos. Adjuntad los valores usados para (X,Y) y la nueva matriz H obtenida (debe ser similar a la anterior). Guardar la nueva matriz obtenida con **`save H2 H;`**



Veis que para diferentes elecciones de los puntos usados se obtienen diferentes matrices H. Una buena estimación de H depende de si los puntos escogidos son representativos de lo que está pasando en toda la malla. Para lograr esto lo ideal sería estimar H a partir de TODOS los puntos de la malla, no solo cuatro. De esta forma H incorporará información de lo que pasa en todas las zonas de la foto. La función **`fc_get_H()`** es capaz de trabajar con cualquier número de puntos: si son más de 4 nos dará la matriz H que mejor **ajusta** todos los puntos.

Para capturar las coordenadas u,v de todos los puntos de la malla se podría ampliar el bucle usado de 4 a 77 pero sería bastante aburrido. Lo que haremos es predecir (usando p.e. la estimación de H guardada en H1) donde caerían los puntos de la malla en la imagen. La predicción no será exacta (salvo en los 4 puntos usados para estimar esa H), pero luego usaremos la función refinar() para mejorarla.

Continuad el script lab2, haciendo un **`load malla_XY`** que carga 2 vectores X, Y de tamaño (1 x 77) con las coordenadas X,Y de TODOS los puntos de la malla sobre el papel. Si hacéis **`plot(X,Y,'bo')`** veréis una malla regular desde -100 a 100 mm en la horizontal y desde -60 a 60 mm en la vertical a saltos de 20 mm (la separación entre líneas en el papel). Se trata de transformar estas coordenadas X,Y usando la matriz H para obtener sus correspondientes coordenadas (u,v) sobre la foto:

$\begin{pmatrix} U \\ V \\ W \end{pmatrix} == \begin{pmatrix} & H & \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$ y luego pasamos de (U,V,W) a (u,v) haciendo $\begin{cases} u = U/W \\ v = V/W \end{cases}$

Aunque podríamos hacer un bucle, es mejor aplicar la matriz H simultáneamente a todas las coordenadas haciendo:

```
UVW=H*[X; Y; ones(1,77)];
```

El resultado es una matriz 3x77 cuyas filas son las coordenadas (U, V, W) de los 77 puntos. Dividiendo (**punto a punto**) la 1ª fila (U) de UVW por la tercera (W) obtendremos las coordenadas up predichas sobre la foto de los 77 puntos. Si ahora lo repetimos con la 2ª fila (V/W) tendremos las coordenadas vp predichas.

Con un bucle barred los 77 puntos aplicando refinar() a las coordenadas de partida {up(k),vp(k)} para obtener el punto exacto del cruce. Guardar los resultados en dos vectores u y v. Visualizar la imagen original y superponer sobre ella los puntos predichos (up,vp) en rojo y los puntos corregidos (u,v) en verde. Adjuntad zoom de la imagen mostrando el resultado en uno de los cuadrados (4 puntos) de la malla donde se aprecien las diferencias entre los cruces reales y predichos.

Calcular la diferencia (en píxeles sobre la foto) entre las coordenadas de los puntos predichos (up,vp) y las de los verdaderos cruces (coordenadas u,v). Guardad estas diferencias en dos vectores du y dv. Usando estos vectores calculad la distancia entre los puntos predichos y reales como d=sqrt(du^2+dv^2). Volcad el valor medio de este vector d, lo que nos dará una idea del error medio de predicción de la matriz H.

Esta media no muestra cómo se distribuyen los errores en los distintos puntos de la malla. Usar la función suministrada fc_quiver(du,dv) que muestra estos errores de una forma más intuitiva. El error en cada punto se presenta como una flecha, mostrando (de forma exagerada) la magnitud y dirección de los desplazamientos. Con un 3er argumento (opcional) S podemos aumentar (S>1) o disminuir (S<1) la escala de las flechas. Adjuntad el gráfico obtenido con S=2. ¿Dónde es cero el error? ¿En qué puntos se observan los mayores errores?

Una vez que tenemos las coordenadas (u,v) de los 77 cruces podemos recalcular la matriz H con la información de todos los puntos (coordenadas X,Y de partida y u,v de llegada). Usando **`volcar_H()`** volcad la nueva matriz H. Guardarla con **`save H77 H`**.

Repetir el cálculo de los desplazamientos entre las coordenadas reales (u,v) y las predichas (up,vp) usando ahora la matriz de H77. ¿Cuál es ahora la media de las distancias entre coordenadas predichas y reales? Adjuntad la gráfica con los errores representados como flechas en este caso (usad la misma escala S=2 que en la anterior gráfica). ¿Por qué ahora el error no es nulo en ningún punto de la malla?

Salvo que se indique lo contrario en el resto de los apartados usaremos la matriz H (guardada en H77) obtenida con el ajuste de todos los puntos de la malla.

2) Estimación de la focal f y pose (posición/giro) de la cámara

Una vez que disponemos de la matriz H, nuestro problema es que dicha matriz es el producto de dos matrices K y Q. Por lo tanto en H está mezclada la información de los parámetros intrínsecos de la cámara (f, u0, v0 que están en la matriz K) con los extrínsecos (posición/orientación de la cámara dados por matriz R y vector t=-R·X0) que forman la matriz Q:

$$\begin{pmatrix} \\ H \\ \\ \end{pmatrix} = \begin{pmatrix} \\ K \\ \\ \end{pmatrix} \begin{pmatrix} \\ Q \\ \\ \end{pmatrix} = \begin{pmatrix} \\ K \\ \\ \end{pmatrix} \begin{pmatrix} \bar{r}_1 & \bar{r}_2 & \bar{t} \\ & & \end{pmatrix}$$

Si conociéramos la pose de la cámara tendríamos la matriz Q y podríamos despejar K, obteniendo los parámetros intrínsecos de la cámara. El problema es que medir de forma precisa la posición y rotación de la cámara con respecto a los ejes del papel es complicado. Afortunadamente si asumimos que u0 y v0 son conocidos es posible despejar la focal f como:

$$f = \sqrt{\frac{\left(\bar{h}_2^T \cdot B \cdot \bar{h}_2 - \bar{h}_1^T \cdot B \cdot \bar{h}_1\right)}{\left(H_{31}{}^2 - H_{32}{}^2\right)}} \quad \text{con} \quad B = \begin{pmatrix} 1 & 0 & -u_0 \\ 0 & 1 & -v_0 \\ -u_0 & -v_0 & u_0^2 + v_0^2 \end{pmatrix}$$

donde H_{31} y H_{32} son el 1er y 2º elemento de la 3ª fila de la matriz H y $\underline{h_1}$ y $\underline{h_2}$ las dos primeras columnas de la matriz H. Para construir B necesitamos conocer los valores de u0 y v0. Si asumimos que el eje óptico de la cámara está perfectamente alineado con el centro del sensor tenemos que u0=1/2 ancho del sensor y v0=1/2 alto del sensor (en píxeles). Estimar la focal f usando la fórmula anterior. La focal obtenida tiene las mismas unidades que u0 y v0 (píxeles). Para convertirla a milímetros hay que conocer el tamaño del sensor. Esta cámara usa un sensor de formato APS-C con un ancho de 23.7 mm y un alto de 15.7 mm. ¿Cuántos píxeles por mm hay en este sensor? ¿Cuál es la focal de la lente en mm?

Las cámaras digitales guardan mucha información sobre cómo se tomó la foto en el fichero jpeg usando un conjunto de etiquetas (EXIF, Exchangeable Image Format). Usad la aplicación Exif-tools o una web como https://www.verexif.com/ o similar para ver los datos contenidos en la foto "malla.jpg". ¿Qué focal registró la cámara al tomar la foto? Adjuntad una captura de pantalla con los datos mostrados.

Conocida la focal f y asumiendo que el eje óptico está perfectamente alineado con el centro del sensor (u0, v0 = 1/2 del tamaño del sensor, $\gamma = 0$) podemos construir la matriz K y despejar Q como:

$$H = K \cdot Q = \begin{pmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \bar{r}_1 & \bar{r}_2 & t \\ & & \end{pmatrix} \quad \Rightarrow \quad Q = K^{-1} \cdot H$$

Volcar la matriz Q obtenida. Esta matriz Q estará formada por la 1ª y 2ª columnas de la matriz de rotación R (vectores r_1 y r_2) y por el vector t (-R·X0).

Desgraciadamente hay un problema con la matriz H que no hemos mencionado. Estas matrices de proyección no son únicas: si en vez de H usamos por ejemplo 8·H los resultados serían iguales. Debido al factor 8, en vez de coordenadas (U,V,W) obtenemos (8U,8V,8W), pero como al final se divide por la 3ª componente el factor se cancela y las coordenadas 2D finales (que es lo que nos interesa) son las mismas (u=U/W, v=V/W). Este posible factor desconocido en H se trasladará a la matriz Q, lo que introduce una posible ambigüedad, ya que en vez de (r1, r2, t) sus columnas podrían ser ($\alpha \cdot r_1$, $\alpha \cdot r_2$, $\alpha \cdot t$) para un cierto factor α.

Afortunadamente las propiedades de R nos permiten estimar el factor α y resolver la ambigüedad. Las columnas de una matriz de rotación R deben tener norma unidad. Tras extraer r_1 y r_2 calcular sus normas n1 y n2 (que deben ser similares) y usarlas para normalizar ambos vectores: $r_1 = r_1/n_1$, $r_2 = r_2/n_2$. Luego, a partir de n1 y n2, estimamos α como α=sqrt(n1·n2) y lo usamos para corregir el valor de t: t=t/α.
Dar el valor de α y los vectores corregidos r1, r2 y t.

La 3ª columna de R no aparece en la matriz Q. Por suerte, una matriz de rotación es tan redundante que r3 puede obtenerse a partir de r1 y r2, como su producto vectorial, que en MATLAB se calcula r3=cross(r1,r2). A continuación normalizar r3, haciendo r3=r3/||r3|| para asegurar que su norma es también 1.

Todavía hay que arreglar un posible problema, ya que debido a errores al estimar H r1 y r2 posiblemente no sean perfectamente perpendiculares. El producto escalar entre dos vectores unitarios dot(a,b) corresponde al coseno del ángulo que forman. Haced el producto escalar entre r1 y r2 y aplicadle acosd() para obtener el ángulo entre ellos. ¿Valor del ángulo entre r1 y r2? (debería ser aproximadamente 90º).

Cread la matriz de rotación R=[r1 r2 r3] y volcad el producto $R^T \cdot R$. El resultado debería ser la matriz identidad, pero no lo será exactamente debido al problema indicado. La forma más sencilla de corregir R es hacer su descomposición en valores singulares (SVD) y forzar a que sus valores singulares sean todos 1's:

```
[U,S,V]=svd(R); R=U*eye(3)*V';
```

Adjuntad matriz R corregida (comprobad que ahora $R^T \cdot R=I$)

Completad la función **`function [R,t]=pose_from_HK(H,K)`** en la que juntamos todos los pasos de este apartado:

- Despejar Q a partir de H y K y extraer sus columnas.
- Normalizar r1 y r2, estimar el factor α y corregirlo en t. Calcular r3.
- Construir R y retocarla para que sea una verdadera matriz de rotación.

La función devuelve la matriz R y vector t obtenidos. Adjuntad código de función.

Conocidas R y t, podemos ahora calcular X0 (la posición de la cámara respecto a los ejes del papel). Despejad X0 de la expresión t=-R·X0 y adjuntad resultado obtenido. ¿Qué distancia había entre el "centro óptico" de la cámara y el punto central de la malla en el papel (origen de coordenadas) en el momento de tomar esta foto?

3) Mejora de la estimación de R, t y f usando optimización

El método anterior nos da una solución razonable del problema, pero dependemos de tener una buena estimación de la transformación H. Para no depender tanto de la matriz H inicial podemos aplicar una optimización a los resultados de R, $\underline{t}$ y f. El punto de partida serán las coordenadas [X,Y] (en papel) y sus correspondientes coordenadas [u v] (sobre la foto). Se trata de que al aplicar el cambio de ejes (dado por R, $\underline{t}$) y luego la proyección a píxeles (usando la focal f) deberíamos obtener valores muy próximos a las coordenadas (u,v) que hemos medido sobre la imagen.

Un algoritmo de optimización recibe una hipótesis inicial de los parámetros a hallar (en este caso serían R, $\underline{t}$ y f hallados a partir de la matriz H en el apartado anterior) y los modifica para intentar que la discrepancia (residuos) entre los valores de (u,v) obtenidos al aplicar el modelo y los medidos en la foto sea lo más pequeña posible. El algoritmo de optimización solo necesita que nosotros escribamos una función que calcule los residuos a minimizar: su tarea será ir cambiando (con una estrategia inteligente) los valores de los parámetros (en este caso R, t ,f) para conseguir que los residuos calculados por nuestra función se hagan cada vez más pequeños.

Un problema es que la matriz R que describe la rotación entre la cámara y la escena no es un buen parámetro a usar en una optimización. La matriz R es redundante (tiene 9 valores pero solo 3 son independientes). En la optimización esto supone un problema, ya que si el algoritmo modifica una de las componentes de R, la matriz resultante ya no será una matriz de rotación correcta. Por eso será preferible usar alguna otra de las formas que existen para representar una rotación. Usaremos el vector $\underline{\omega}$, cuyo módulo corresponde al giro aplicado (en grados) y su dirección al eje de giro. Al ser una representación mínima (3 parámetros), cualquier modificación del vector $\underline{\omega}$ corresponde a un giro válido.

Necesitamos ser capaces de pasar de un matriz de rotación R a su vector de giro $\underline{\omega}$ equivalente y viceversa. Para ello usaremos la función **`function out=R2w(in).`** Lo primero que hace esta función (que tenéis que completar) es comprobar el tamaño del argumento de entrada (in):

- Si es una matriz hacemos R=in y calculamos el vector w equivalente que es asignado al parámetro de salida out.
- Si es un vector hacemos w=in y construimos la correspondiente matriz R de rotación que será asignada al parámetro de salida out.

De esta forma la función puede usarse en ambos sentidos. Dentro de la función solo falta completar los detalles de la conversión entre R y $\underline{\omega}$ con las fórmulas que podéis encontrar en las transparencias. Adjuntad código de vuestra función R2w().

Adjuntad la matriz de rotación correspondiente a ω=[40; 0; -10] y el vector ω que equivale a la matriz R = **`0.3481  0.9332  0.0893`**
`0.6313 -0.3038  0.7135`
`0.6930 -0.1920 -0.6949`
¿A cuántos grados corresponde la rotación dada por la matriz R?

Una vez que disponemos de R2w() ya podemos centrarnos en la optimización. El algoritmo de optimización va a trabajar con un total de 7 parámetros: el vector t (3 componentes), el vector $\underline{\omega}$ (3 componentes, representando a R) y la focal f de la cámara (en píxeles). Estos parámetros los agruparemos en un vector columna P de tamaño 7x1: P=[t ; ω ; f]. Lo único que queda es escribir la función cuyo resultado debe minimizar el algoritmo de optimización:

```
function error=error_uv(P,X,Y,u,v)
```

La función recibe el vector P con los parámetros a optimizar, las coordenadas X,Y de los 77 puntos en el papel (en mm) y las correspondientes coordenadas u,v sobre la foto (en píxeles) que encontrasteis en el apartado 1. Dentro de la función debéis:

1. Reservar dos vectores uu y vv (mismo tamaño que u y v) donde guardaremos las coordenadas proyectadas.
2. Extraer los parámetros del vector P: t=P(1:3), ω=P(4:6) , f=P(7).
3. Obtener la matriz de rotación R a partir de ω. Extraer las 2 primeras columnas $\underline{r1}$, $\underline{r2}$ de la matriz R y juntarlas con $\underline{t}$ para construir la matriz Q.
4. Aplicar Q a las coordenadas homogéneas [X Y 1], para obtener coordenadas cámara $(XYZ)_{cam}$ y luego pasar a coordenadas normalizadas
$$\begin{cases} \tilde{x} = X_{cam} / Z_{cam} \\ \tilde{y} = Y_{cam} / Z_{cam} \end{cases}$$
5. Usando los valores u0,v0 por defecto (1/2 tamaño sensor) y la focal f pasar de coordenadas normalizadas a píxeles (uu,vv):
$$\begin{cases} uu = u_0 + f \cdot \tilde{x} \\ vv = v_0 + f \cdot \tilde{y} \end{cases}$$
6. Calculad (uu-u) y (vv-v) para obtener los errores entre los píxeles predichos por el modelo (uu,vv) y los medidos sobre la imagen (u,v). Juntad estos dos vectores en un único vector **error** y transponerlo. El resultado será un vector columna de tamaño 154 x 1, que es lo que devuelve la función.

Para los pasos 4 y 5 podéis usar un bucle convirtiendo las coordenadas X(k),Y(k) y rellenando el correspondiente valor de uu(k) y vv(k). Alternativamente se puede trabajar "en bloque" como se hizo en el apartado 2. Adjuntad código de error_uv()

Usando las estimaciones (f,R,t) del apartado anterior cread un vector P0=[t ; ω ; f] (tamaño 7x1), usando el vector de giro ω equivalente a R. Ejecutar la función error_uv() para dicho P0. ¿Cuál es la norma del vector de errores resultante?

Para optimizar esta estimación inicial (P0) usaremos una función de optimización de MATLAB, **lsqnonlin().** Su uso es el siguiente:

```
opts=optimset('Algorithm','levenberg-marquardt','Display','off');
f_min=@(P)error_uv(P,X,Y,u,v);
P=lsqnonlin(f_min,P0,[],[],opts);
```

La primera orden especifica el algoritmo a usar. En la segunda se crea un puntero a la función error_uv(), indicando que de todos sus parámetros, la optimización va a

actuar sobre el primero de ellos (P). Finalmente en la 3ª línea se llama a la función de optimización, pasándole la función a minimizar, junto con la hipótesis inicial P0 para los parámetros. El resultado es un vector P con los parámetros encontrados que minimizan los errores de la función error_uv(). Ejecutad la optimización y dad la focal f (en pixels y mm) obtenida, así como los vectores t y ω (con 2 decimales).

Ejecutar error_uv para los parámetros P optimizados devueltos por el algoritmo. Calcular la norma del vector de errores (ahora debería ser menor).

En este caso se ve que, aunque el error se reduce, la reducción es muy pequeña, lo que indica que la estimación inicial P0 era ya bastante buena. En este caso la fase de optimización puede que no fuese necesaria.

Vamos a ver ahora una situación (empezando con una matriz H menos acertada) donde aplicar optimización puede ser clave. Repetir el apartado 2) pero en vez usar la H calculada para todos los puntos (H77) usaremos la H calculada para la zona inferior de la malla (H2). Dar los resultados de la estimación de la focal f (en mm), vector de giro ω y vector t obtenidos con esta nueva matriz H. Aplicad la función error_uv() a esos parámetros y dad la norma del vector de errores obtenido.

Aplicad la optimización empezando en el nuevo conjunto de parámetros e indicad la norma de los errores resultantes. El resultado de la optimización será muy similar al anterior a pesar de que la hipótesis de partida era mucho peor.

Los errores devueltos por error_uv pueden visualizarse como antes como flechas en una malla. Para ello extraer du (errores en u) y dv (errores en v) del vector de errores y visualizarlos usando fc_quiver(du,dv). Haced este gráfico con los errores de antes y después de la optimización en el caso de usar la matriz H2. Adjuntad ambas gráficas (usando la misma escala S=2 en ambas). Observaréis que los errores tras la optimización son ahora mucho menores que los del punto de partida.

Introducir parámetros adicionales en la optimización

Cuando dibujamos el rectángulo uniendo las esquinas de la malla se apreciaba que las líneas exteriores (que son líneas rectas en el papel) aparecen curvadas en la foto. Este es un ejemplo de una distorsión causada por la lente, típica de un gran angular (distancia focal corta), como es el caso de la lente usada en esta foto.

Cambiar los parámetros de la matriz K no puede arreglar este problema. Una matriz es un operador lineal que nunca puede convertir una recta en una curva, como hace la lente usada. Este tipo de distorsiones aumentan al alejarnos del eje óptico, por lo se les suele hacer depender de las coordenadas normalizadas (que son una medida del ángulo respecto del eje óptico). Un modelo típico es:

$$r^2 = \tilde{x}^2 + \tilde{y}^2 \Rightarrow \begin{cases} \tilde{x} = \tilde{x} \cdot (1 + k_1 \cdot r^2) \\ \tilde{y} = \tilde{y} \cdot (1 + k_1 \cdot r^2) \end{cases}$$

Una vez calculadas las coordenadas normalizadas (x_n, y_n), se calcula r^2 (que mide lo lejos que cae un punto del eje óptico). Usando r^2 y un parámetro de distorsión k_1 se cambian las coordenadas normalizadas. Estas nuevas coordenadas normalizadas son las usadas para calcular la posición en píxeles sobre la imagen. Observad que si k1=0.0 no se introduce ningún cambio y por lo tanto no hay distorsión.

Es sencillo modificar el proceso de optimización para estimar este nuevo coeficiente de distorsión k1 junto con la focal f. Basta ampliar el vector de parámetros P para que incluya a k1 como una octava componente y cambiar el código de la función error_uv() para que extraiga dicho parámetro k1=P(8) y lo use para modificar las coordenadas normalizadas con la fórmula anterior. Adjuntad código añadido a la función error_uv.

Una vez modificada la función del error volver a correr el proceso de optimización incorporando el nuevo parámetro. En la hipótesis inicial P0, k1 puede inicializarse a 0, ya que normalmente será un valor muy pequeño. ¿Valores obtenidos para f y k1?

Rellenar la siguiente tabla con los valores de la norma del vector de errores:

	R, t y f deducidos de H	Optimización	Optimización con k1
||e|| = norm(e)			

Calcular los errores tras la optimización (usando y sin usar el parámetro k1 de distorsión) y visualizarlos usando fc_quiver (usad una escala S=4 ya que los errores serán ahora menores). Adjuntad ambas gráficas.

Como veis es bastante sencillo añadir parámetros adicionales a una optimización. Basta añadirlos al vector P y luego usarlos al calcular los errores en la función error_uv(). Por ejemplo, hasta ahora hemos asumido que el eje óptico de la cámara estaba perfectamente alineado con el centro del sensor, por lo que los valores de u0 y v0 correspondían a la mitad del tamaño del sensor. Si consideramos que esto no es así, podemos añadir ambos parámetros a P (para un total de 10 parámetros) y usarlos dentro de la función de error en vez de los valores asumidos hasta ahora.

Volver a correr la optimización con este nuevo par de parámetros adicionales (como hipótesis inicial usar los valores por defecto ¿Norma del error final?

Dad los valores finales obtenidos para t, w, f , k1, u0 y v0.
¿Cuánto se desplaza el eje óptico respecto a su posición ideal (en píxeles y en mm)?

Aunque vemos que el error se reduce bastante debemos ser cuidadosos con este enfoque. Está claro que al aumentar el número de parámetros que pueden variarse, la optimización siempre nos dará un error más bajo. Sin embargo, si seguimos añadiendo parámetros adicionales lo ideal sería usar más fotos para estar seguros de que las mejoras se repiten en varias fotografías. Así estaremos más seguros de que realmente se deben a que estamos detectando una propiedad de la cámara.

4) Estimación de los parámetros de la cámara de vuestro teléfono

Se trata de calibrar la cámara de vuestro móvil usando una fotografía de la misma malla que hemos estado usando en esta práctica.

Imprimid el fichero malla_calib.pdf (o usad el papel que os di en el LAB) y hacerle una foto con el móvil, similar a la foto con la que hemos trabajado en la práctica. Aseguraros de que está lo mejor enfocada posible y que entran las 4 esquinas de la malla. Procurad haced vuestra foto con buenas condiciones de iluminación para que el algoritmo de detección de los cruces funcione correctamente.

Correr el programa que habéis escrito durante esta práctica usando vuestra nueva foto. Tened en cuenta que el tamaño de vuestra foto será distinto. Si imprimís el pdf con el patrón es posible que tengáis que modificar en el programa las coordenadas X e Y, que correspondían a los puntos de la malla sobre el papel. Estas coordenadas pueden cambiar dependiendo del tamaño con el que salgan los cuadrados de la malla al imprimirse. Para recalcular dichas coordenadas podéis hacer X=(X/20)*DX, Y=(Y/20)*DY, donde DX,DY son las dimensiones de los cuadrados de vuestra malla impresa (en mm).

Adjuntad los siguientes resultados:

- Marca y modelo de vuestro teléfono
- Tamaño de los cuadros en vuestra malla de calibración (DX,DY).
- Tamaño del sensor en píxeles.
- Focal de la lente (mm) obtenida a través de la información EXIF de las fotos.
- Matriz H estimada para las 4 esquinas y para todos los 77 puntos de la malla.
- Gráfica de flechitas mostrando los errores de predicción usando la matriz H de las 4 esquinas y los errores usando la matriz H estimada con los 77 puntos.
- Valor de la focal f obtenida inicialmente (en píxeles) a partir de la matriz H estimada para los 77 puntos de la malla.
- Estimación del tamaño físico del sensor (ancho y alto en mm).
- Valores de f y k1 tras usar la optimización.
- Valor de la norma del vector de errores tras la optimización con solo la focal f y tras la optimización usando la focal y el coeficiente k1 de distorsión.
- Dad la posición de la cámara en el momento de tomar la foto referida a los ejes del papel posicionados en el centro de la malla.