Calibración de una cámara

(para entregar por parejas en unos 7 / 10 días)

En este proyecto veremos los fundamentos del proceso para obtener los parámetros intrínsecos de una cámara (calibración) a partir de una fotografía tomada de un cierto patrón. En la práctica, si queremos determinar con exactitud los parámetros intrínsecos (focal, posición e inclinación del eje óptico, distorsiones, etc.) es necesario tomar múltiples fotografías para tener más información. Por simplicidad en esta práctica nos limitaremos a trabajar con una única fotografía, lo que limitará el número de parámetros que podremos determinar. Nos centraremos en hallar el valor de la focal f, suponiendo que el resto de los parámetros toma sus valores por defecto, aunque en uno de los apartados consideraremos la distorsión de la lente.

Planteamiento del problema:

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

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

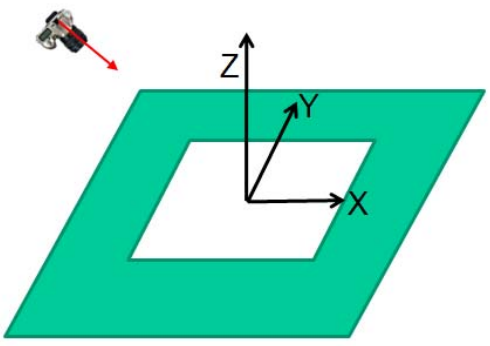
donde X es la posición del punto 3D en coordenadas de la escena, X0 es la posición de la cámara en dichas coordenadas y R es la matriz de rotación que representa el giro entre ejes escena y ejes cámara. En este problema será más conveniente reescribir la anterior relación como:

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

Expresado matricialmente con coordenadas homogéneas $\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{cam} = \left(\begin{array}{c|c} R & \bar{t} \end{array} \right) \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$

El proceso de calibración se basa en hacer una fotografía de un patrón situado en una superficie plana. Sin pérdida de generalidad podemos definir el plano XY de las coordenadas de la escena como el plano del papel (ver figura adjunta). En ese caso, todos los puntos en la foto tienen la misma coordenada Z=0, y la expresión anterior se simplifica como:

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



donde r1 y r2 son las 2 primeras columnas de la matriz de giro R (la 3ª columna de R desaparece al estar siempre multiplicada por Z=0).

Tras el paso a coordenadas cámara se aplica una segunda matriz K que incorpora los parámetros intrínsecos de la cámara (focal f, posición u0,v0 del eje óptico en la foto, etc.). Con la proyección final (dividiendo por la 3ª componente) obtenemos los píxeles (u,v) en la foto:

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

El problema se ha reducido a una transformación 2D entre puntos de un plano (el papel) con coordenadas X,Y (medidas en cm o mm) que se proyectan en otro plano (la foto) con coordenadas u,v (en píxeles). La transformación involucrada es de tipo proyectiva y se puede determinar si disponemos de al menos 4 puntos conocidos con sus coordenadas de partida (X,Y en el papel) y de llegada (u,v en la foto). La determinación de esta matriz H será nuestro primer ejercicio en esta práctica.

1) Determinación de coordenadas de las esquinas y estimación de H (15%)

Cargad la imagen contenida en el archivo 'malla1.jpg' y visualizarla. Es la foto de una superficie plana (hoja de papel) con una malla de 6 (vertical) x 10 (horizontal) cuadraditos. La separación entre líneas (lado de los cuadrados) es de DX=2.0 cm en el sentido horizontal y de DY=2.0 cm en el vertical. En total hay 7x11 cruces entre líneas. Nuestro primer objetivo es obtener las coordenadas (en píxeles) de todos esos cruces. Como las coordenadas de los puntos en la hoja de papel son conocidas, a partir de ellos podremos determinar la matriz H de transformación.

Como sería muy aburrido tener que pinchar de forma precisa en esos 77 puntos vamos a automatizar parcialmente el problema. El primer paso será seleccionar los 4 cruces de las esquinas, pinchando sobre ellos de forma aproximada y escribiendo un sencillo algoritmo que refine la búsqueda y encuentre el punto exacto del cruce.

El algoritmo será muy similar al que usamos en la práctica anterior para el tracking de los puntos. Si ejecutáis el script lab2.m veréis que carga la imagen de la malla y la visualiza. Luego entra en un bucle para pinchar en las 4 esquinas (empezando en la superior izquierda y siguiendo en el sentido de las agujas del reloj) y pinta un círculo en el punto seleccionado. Durante esta práctica iremos añadiendo código a este script y creando nuevas funciones para ir resolviendo los diferentes apartados.

En primer lugar, para no tener que ser muy precisos al pinchar los puntos vamos a escribir un código que nos permita detectar el punto de cruce exacto a partir de una primera aproximación (el punto donde pinchamos). Este procedimiento nos servirá luego para detectar automáticamente el resto de los puntos de la malla.

Para el algoritmo usaremos una imagen auxiliar aux que ya está calculada. Hemos pasado la imagen original a B/W usando rgb2gray() y luego le aplicamos un filtro promedio haciendo:

```
ff = fspecial('gaussian',15,5); aux=imfilter(aux,ff);
```

Este tipo de filtros los estudiaremos en un tema posterior, pero básicamente lo que hemos hecho es obtener una versión suavizada y en grises de la imagen original.
Visualizar la imagen aux y adjuntarla.

Para refinar los puntos seleccionados hay que completar la función:

```
function [x,y]=refinar(x,y)
```

La función recibe la posición preliminar (x,y) y debe devolver la posición final (x,y). El código a completar es muy similar al que usamos en la aplicación de tracking. Dentro de la función se define una variable RAD que marca el radio de la zona a explorar y dos matrices dx, dy con las coordenadas X,Y de esa zona referidas a su centro (desde -RAD hasta RAD en ambos ejes). También declaramos la variable aux como global, para poder acceder a ella dentro de la función. El código a completar consiste en:

1. Redondear las coordenadas de entrada x=round(x); y=round(y), para obtener el píxel de la imagen (que deben ser números enteros) alrededor del cual se va a buscar el cruce.

2. De la imagen aux extraer una subimagen s de +/-RAD píxeles centrada en (x,y) donde buscaremos el cruce. **Recordad que las filas de una imagen en MATLAB corresponden a coordenadas Y y las columnas a X**. **No os olvidéis tampoco de convertir s a double para hacer las cuentas.**

3. Hallamos el mínimo de s con m=min(s(:)) y calculamos $w = \exp\left(-(s-m)^2)\right)$. Luego dividimos w por la suma de todos sus valores, para obtener una matriz de pesos. Para sumar los valores de una matriz A sin tener que usar bucles se puede hacer sum(A(:)). La idea es que el punto buscado (al ser el cruce de dos líneas oscuras) estará en el centro de la zona con valores más bajos (el mínimo m). La matriz de pesos w tendrá sus valores más altos en esa zona.

4. Construir la matrix x+dx (las coordenadas de los píxeles de la subimagen) y multiplicarla (punto a punto) por la matriz de pesos w, sumando luego todos sus valores. El resultado será la estimación de la coordenada x del cruce. Guardar esa estimación en x, machacando el valor inicial de x. Repetir con las coordenadas y's, obteniendo la nueva coordenada y mejorada. La función devuelve dichas coordenadas (x,y) mejoradas.

Una vez completada la función, modificar el bucle de lab2.m llamando a refinar con los valores de (x,y) obtenidos con el ratón. Guardar los resultados devuelto en u(k) y v(k) respectivamente. Repetir el fprintf() para volcar las nuevas posiciones. Haced lo mismo con el plot, pintando también la posición refinada en color verde.

Si todo funciona correctamente deberías ver los nuevos puntos verdes centrados en los cruces. Obviamente, si la elección inicial está demasiado lejos (a más de RAD píxeles) del cruce correcto no se detectará adecuadamente. Como referencia, la posición corregida de la primera esquina debería salir: (1310.1, 651.4)

Adjuntad el volcado con las posiciones iniciales y finales, así como un zoom de una de las esquinas mostrando punto inicial (rojo) y su mejora (verde).

Adjuntad código de la función refinar.m una vez verificada.

Una vez que habéis guardado las posiciones de las 4 esquinas en u,v, el comando `plot([u;u(1)],[v;v(1)],'b')` dibuja un rectángulo uniendo las 4 esquinas halladas. Superponerlo sobre la imagen original. Fijaros en la base del rectángulo. ¿Caen los puntos de la malla dentro o fuera de la línea trazada? ¿A qué es debido?

Adjuntad una captura de un zoom de la zona donde se aprecie este detalle.

Conocidas las coordenadas (u,v) de los 4 esquinas sobre la imagen, falta establecer sus coordenadas XY sobre el papel. Tomando como origen el centro de la malla y sabiendo que cada cuadrado son 2cm, las correspondientes coordenadas X,Y de las esquinas sobre el papel son:

X	Y
-10	6
10	6
10	-6
-10	-6

Guardar las anteriores coordenadas en dos vectores columna (4x1) X e Y. Una vez que disponemos de las coordenadas originales (X,Y en el papel) y finales (u,v en la imagen) de las cuatro esquinas, podemos hallar la matriz H que transforma unas en otras. En un tema posterior estudiaremos cómo determinar estas transformaciones a partir de una serie de puntos de origen (X,Y) y de destino (u,v). Como todavía no hemos visto los detalles para esta práctica os daré yo la función a usar `fc_get_H(XY,uv)`. Los argumentos de entrada son las coordenadas origen XY de los 4 puntos (matriz 2x4) y sus coordenadas destino uv (matriz 2x4). La función nos devuelve una matriz H, verificando:

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

Como la función espera matrices 2x4 con las coordenadas de origen y destino debemos agrupar los vectores u,v y X,Y respectivamente: `H=fc_get_H([X Y],[u v])`

Tras calcular H usar el comando `fprintf('%6.1f %6.1f %7.1f\n',H');` para volcar la matriz obtenida. Adjuntad vuestro resultado.

Una vez calculada la estimación de H, podéis guardarla haciendo save H H. Luego comentar todo el código del bucle y el cálculo de H para no tener que volver a seleccionar las esquinas, etc. cada vez que hagáis una prueba en los ejercicios siguientes. Haciendo un load H, recuperareis la matriz H que es todo lo necesario para seguir haciendo los ejercicios.

2) Estimación de H con TODOS los puntos de la malla (10%)

Aunque con solo 4 puntos hemos sido capaces de determinar H podemos hallar una mejor estimación si usamos TODOS los puntos de la malla, no solo las 4 esquinas. De esta forma tendremos información de lo que pasa en el centro de la malla y no solo en la periferia.

Para capturar los 77 puntos de la malla podríamos ampliar el bucle anterior de 4 a 77 pero eso sería muy aburrido. Lo que haremos es predecir (usando la estimación que acabamos de hallar para H) donde deben caer los puntos de la malla sobre la imagen. El resultado no será exacto, debido a efectos como los que hemos visto antes (p.e. los puntos de un lado de la malla no caían en una línea recta sobre la imagen). La idea es usar la estimación como punto de partida y mejorarla con la función refinar().

Continuad el script lab2, haciendo un **`load malla_XY`**: son 2 vectores X,Y de tamaño 77x1 con las coordenadas X,Y (sobre el papel) de todos los puntos de la malla. Si hacéis un plot(X,Y,'bo') veréis una malla regular desde -10 a 10cm en la horizontal y desde -6 a 6 cm en la vertical a saltos de 2 cm (la separación entre líneas).

Se trata de transformar estas coordenadas X,Y usando la matriz H hallada antes:

$$\begin{pmatrix} U \\ V \\ W \end{pmatrix} == \begin{pmatrix} & H & \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$ para luego hacer $\begin{cases} u = U/W \\ v = V/W \end{cases}$ y obtener así u y v.

Aunque podríamos hacer un bucle, las coordenadas u,v pueden calcularse en bloque haciendo: **`C=[X Y X.^0]*H';`** El resultado C es una matriz 77 x 3 donde cada columna corresponde a U, V y W para todas las coordenadas. Si dividimos (punto a punto) la 1ª columna de C por la 3ª tendremos la lista de coordenadas u (U/W). Repitiendo con la 2ª columna (V/W) obtendremos las coordenadas {v}. Visualizar la imagen original y superponer sobre ella los puntos correspondientes a (u,v). Veréis que salvo las 4 esquinas iniciales (con las que se calculo la H usada) el resto de los puntos no cae exactamente sobre los cruces de la malla.

Haced un bucle barriendo las 77 posiciones {u(k),v(k)} y usar la función refinar() para capturar el punto exacto de la cruz (que no andará muy lejos). Guardar los resultados en los mismos vectores u(k) y v(k), machacando los datos de partida.

Superponer sobre la imagen anterior con los puntos (u,v) originales los nuevos puntos (u,v) corregidos en otro color. Adjuntad zoom imagen resultante donde se aprecie el resultado del refinamiento.

Una vez que disponemos de las nuevas coordenadas finales sobre la imagen (u,v), podemos volver recalcular la matriz H usando ahora los 77 puntos. Usando como antes el comando `fprintf('%6.1f %6.1f %7.1f\n',H');` volcad la nueva matriz H.
Adjuntad vuestro resultado.

3) Calibración: estimación de la focal f de la cámara (10%)

Conocida H nuestro problema es que dicha matriz es el producto de dos matrices, K y Q:

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

por lo que en ella están mezclados los parámetros de la cámara (matriz K) y los de la posición/orientación desde donde se tomó la foto (la matriz Q que engloba la información de la matriz R de rotación y el vector t de translación). Si conociésemos Q podríamos despejar K para hallar los parámetros de la cámara. Desgraciadamente es muy complicado medir con exactitud la posición de la cámara y su giro respecto a los ejes "del papel". Afortunadamente, si asumimos que salvo la distancia focal f el resto de los parámetros de K son conocidos (u0=1/2 ancho del sensor, v0=1/2 alto del sensor, γ=0, etc.), manipulando la expresión anterior (y usando la propiedad de que r1 y r2 son vectores ortonormales) podemos despejar f:

$$f = -\frac{\left(h_1^T \cdot B_0 \cdot h_2\right)}{\left(H_{31} \cdot H_{32}\right)} \quad \text{siendo} \quad B_{0'} = \begin{pmatrix} 1 & 0 & -u_0 \\ 0 & 1 & -v_0 \\ -u_0 & -v_0 & u_0^2 + v_0^2 \end{pmatrix}$$

y h1 y h2 las dos primeras columnas de la matriz H.

Estimar la focal f con la fórmula dada. Valor obtenido para f (píxeles).

La focal obtenida (al igual que u0,v0) está en unidades de píxeles. Para pasar a mm debéis saber que la cámara con la que se tomó esta imagen tiene un sensor de tipo APS-C con unas dimensiones de 23.7 mm de ancho por 15.7 mm de alto. Estimar la densidad de píxeles del sensor (pix/mm) ¿Cuál es la focal de la lente usada en mm?

Todo este trabajo nos lo podíamos haber ahorrado porque la cámaras digitales guardan mucha información de cómo se tomo la foto en un conjunto de etiquetas (EXIF, Exchangeable Image Format) dentro del fichero .jpg. Instalar la aplicación Exif-tools o usar un servicio web como https://www.verexif.com/ o similar para ver los datos de la foto "malla1.jpg". ¿Qué focal registró la cámara al tomar la foto?

Adjuntad una captura de pantalla con los datos mostrados.

4) Estimación de la posición / pose de la cámara (15%)

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, γ = 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 resultante.

Las columnas de Q son los vectores r1, r2 (1ª y 2ª columnas de la matriz R) y el vector t=-R·X0). Sin embargo hay un problema. No hemos comentado una pequeña pega que tienen las matrices de proyección como H. La matriz H hallada no es única, ya que si en vez de H usamos 8·H el resultado es equivalente. En vez de (U,V,W) obtendríamos (8U,8V,8W), pero como al final hay que dividir por la 3ª componente el factor 8 se cancela y las coordenadas 2D finales (u=U/W, v=V/W) son las mismas. Por lo tanto no sabemos si los resultados obtenidos (r1, r2, t) son correctos ya que igualmente podrían ser (λr1,λr2,λt) para cualquier valor de λ.
Afortunadamente las propiedades de R nos permiten estimar el factor λ y resolver la ambigüedad. Las columnas de una matriz de rotación R deben tener norma unidad. Tras extraer r1 y r2 calcular sus normas n1 y n2 (deberían ser similares) y calcular λ como λ=sqrt(n1·n2). Una vez estimado λ corregirlo, dividiendo r1, r2 y t por dicho valor. Dar valor obtenido para λ y los vectores corregidos r1, r2 y t.

Nos falta la 3ª columna (r3) de R, pero una matriz de rotación es tan redundante que r3 puede obtenerse de r1 y r2. La 3ª columna de R es el producto vectorial de r1 y r2, que en MATLAB se calcula como r3=cross(r1,r2). La matriz R=[r1 r2 r3] tiene todavía alguna pega, ya que los vectores r1 y r2 no son exactamente unitarios ni completamente perpendiculares. La forma más sencilla de hallar una buena matriz de rotación que sea lo más parecida a la R estimada es a través de su descomposición en valores singulares (SVD), forzando a que sus valores singulares sean todos 1's:

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

Adjuntad matriz R final. Comprobad que R es finalmente una matriz de rotación haciendo R'*R: para una matriz de rotación debe salir la identidad.

Escribid una función `function [R,t]=Rt_from_HK(H,K)` juntando todos los pasos del apartado. La función debe sacar Q de H y K, extraer sus columnas, calcular el factor λ y corregirlo en r1, r2 y t. Luego, calcular r3 y montar R (apilando r1, r2, r3), para finalmente "retocarla" para que sea una buena matriz de rotación, devolviendo la matriz R y vector t finalmente obtenidos. Adjuntad código de función.

Conocidas R y t, podemos calcular X0 (vector posición de la cámara en los ejes del papel) como X0=-R'*t (ya que para matrices de giro inv(R)=R'=transpuesta de R). Adjuntad vector X0 obtenido. ¿Qué distancia había entre la cámara y el centro de la malla en el papel (origen de coordenadas) al tomar la foto?

5) Mejora de las estimaciones usando optimización (20%)

A pesar de haber resuelto el problema, las estimaciones anteriores pueden tener errores (pensad en las apaños que hemos tenido que hacer en el apartado anterior modificando R para que fuese una buena matriz de rotación).

Para mejorar la estimación de R, $\underline{t}$ y f vamos aplicar un proceso de optimización. Disponemos de la colección de coordenadas [X Y] (en papel) y sus correspondientes coordenadas [u v] (en la imagen). Se supone que si aplicamos el cambio de ejes (dado por R, $\underline{t}$) y luego la proyección a píxeles (que depende de f) deberíamos obtener valores muy próximos a las coordenadas (u,v) medidas sobre la imagen.

Un algoritmo de optimización recibe una hipótesis inicial de los parámetros a hallar (en este caso R, $\underline{t}$, y f) y los va modificando intentando que la discrepancia entre los valores de (u,v) obtenidos y los medidos sea lo más pequeña posible. Para que pueda hacer esto es necesario que nosotros escribamos la función que calcula esas diferencias a minimizar. De esta forma la tarea del algoritmo de optimización es ir cambiando (de forma inteligente) los valores de los parámetros de forma que los residuos sean cada vez más pequeños.

El problema es la matriz R que hemos usado hasta ahora para describir la rotación entre escena/cámara, no es un buen parámetro en una optimización. La razón es que la matriz R tiene 3 x 3 = 9 valores, pero solo 3 de ellos son independientes. Si la optimización modifica un valor de R, el resultado ya no será una buena matriz de rotación. Es mucho más conveniente usar alguna otra de las representaciones que vimos para una rotación como, por ejemplo, el vector $\underline{\omega}$, cuyo módulo corresponde al giro aplicado (en rads) y su dirección indica el eje de dicho giro.

Necesitaremos sendas funciones para pasar de una matriz de rotación R a su vector de giro $\underline{\omega}$ equivalente y viceversa. Para pasar de R a ω completad la función **`function w=R2w(R)`** que recibe una matriz de giro R y devuelve el correspondiente vector de giro $\underline{\omega}$ como un vector columna 3x1 (w). Igualmente completad la función **`function R=w2R(w)`** que hace justo lo contrario. Usad para ello las relaciones entre R y $\underline{\omega}$ de las transparencias.

La siguiente de giro matriz R y vector w son equivalentes. Usadlas para verificar las funciones R2w() y w2R().

```
          0.6831   -0.5673   -0.4599          0.6000
R =       0.1829    0.7426   -0.6443    w = -0.7000
          0.7071    0.3560    0.6110          0.4500
```

Adjuntad el vector w obtenido para la matriz

```
R =  0.3481    0.9332    0.0893
     0.6313   -0.3038    0.7135
     0.6930   -0.1920   -0.6949
```

y la matriz de rotación correspondiente a w=[1 2 3]'

Adjuntad código de vuestras funciones R2w y w2R una vez verificadas.

Nuestro algoritmo de optimización trabajará con un total de 7 parámetros: las 3 componentes del vector $\underline{\omega}$ (equivalentes a R), las 3 componentes del vector t y la focal de la cámara (en píxeles). Estos parámetros los agruparemos en un vector columna P (de pose) de tamaño 7 x 1, P = [ω ; t ; f]

Lo único que nos queda es escribir la función cuyo resultado debe minimizar el algoritmo de optimización, con el template: `function e=error_uv(P,X,Y,u,v)`
Dentro de la función debéis:

1. Reservas 2 vectores uu y vv del mismo tamaño que u y v para guardar las coordenadas proyectadas.
2. Extraer los parámetros del vector P: ω=P(1:3), t=P(4:6) , f=P(7). Convertir ω en la correspondiente matriz de rotación R, extraer r1, r2 y juntarlos con t para construir la matriz Q.
3. Aplicar Q a coordenadas [X Y 1], obteniendo coordenadas cámara $(XYZ)_{cam}$.
4. Pasar a coordenadas normalizadas $\tilde{x}$ =X_{cam}/Z_{cam}, $\tilde{y}$ =Y_{cam}/Z_{cam}.
5. Usar la focal f para pasar de coordenadas normalizadas a píxeles (uu,vv), usando los valores u0,v0 por defecto (1/2 tamaño del sensor):
$$\begin{cases} uu = u_0 + f \cdot \tilde{x} \\ vv = v_0 + f \cdot \tilde{y} \end{cases}$$
6. Restar (uu-u),(vv-v) para obtener los errores entre los píxeles predichos (uu,vv) por el modelo y los que hemos medido sobre la imagen (u,v).
7. Poner los errores anteriores uno encima del otro, formando un solo vector e (columna, 154x1), que es lo que devuelve la función.

Para los pasos 3 a 5 podéis usar un bucle convirtiendo la coordenada X(k),Y(k) en cada paso y rellenando el correspondiente valor de uu(k) y vv(k). Alternativamente podéis trabajar como en el apartado 2, calculando en bloque las coordenadas XYZcam, ($\tilde{x}$,$\tilde{y}$) y (uu,vv). Adjuntad vuestro código de error_uv()

Usando las estimaciones de los apartados anteriores (f, R, t) crear un vector P0 (de tamaño 7x1): P0=[ω ; t ; f], donde ω es el vector de giro equivalente a la matriz R estimada. Ejecutar la función error_uv. Haced un gráfico con los errores resultantes. Adjuntad la gráfica. ¿Cuál es la norma de dicho vector de errores?

Para optimizar la estimación inicial (P0) usaremos la función **`lsqnonlin()`** (una de las funciones de optimización de MATLAB)**.** 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 las opciones del algoritmo a usar. En la segunda se crea un puntero a la función a minimizar (error_uv), indicándole que de sus parámetros, el primero de ellos (P) es sobre el que actúa la optimización. Finalmente en la 3ª línea se inicia el proceso de optimización, pasándole la función a minimizar, junto con una hipótesis inicial P0 para los parámetros. Al terminar, la función devuelve el vector P con los parámetros que mejor consiguen reducir el vector de errores de la

función error_uv(). Correr la optimización y dad la focal f (en mm) obtenida, así como los vectores w y t resultante.

Volver a ejecutar error_uv para los nuevos parámetros P devueltos por el algoritmo. Superponer sobre el gráfico anterior los nuevos errores. Adjuntad la gráfica y volver a calcular la norma del vector de errores (debería haberse reducido).

En este caso la reducción no es mucha, lo que indica que la estimación inicial era ya buena y que esta fase de optimización puede que sea innecesaria. Sin embargo, en otras ocasiones puede ser fundamental. Volver a correr todo el proceso completo para la imagen malla2.jpg, una fotografía de la misma malla tomada con otra focal.

¿Qué valor de f se obtiene con la 1ª estimación? ¿Y tras correr el proceso de optimización? Compararla con la focal guardada en el fichero jpg.

Adjuntad la gráfica de los vectores de error en este caso antes y después de la fase de optimización.

6) Estimación de parámetros de distorsión (10%)

En este apartado volvemos a trabajar con la imagen original (malla1.jpg). Cuando en el 1er apartado dibujamos el rectángulo uniendo las esquinas de la malla vimos que los puntos del borde de la malla (que están alineados en el papel) no lo estaban en la foto, mostrando una curvatura hacia afuera. Este es un claro ejemplo de una distorsión causada por la lente, típica de los grandes angulares (focales cortas).

Ninguno de los parámetros que pudiéramos modificar en la matriz K puede corregir este problema. Una matriz es un operador lineal que no puede convertir una recta en una curva, como está haciendo nuestra lente. Este tipo de distorsiones aumentan según nos alejamos del eje óptico, por lo que los modelos habituales las hacen depender de las coordenadas normalizadas (que son una medida del ángulo del objeto respecto del eje óptico). Un posible modelo es:

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

Una vez que tenemos calculadas las coordenadas normalizadas xn, yn, se calcula r (que es una medida del alejamiento del eje óptico en ambos ejes) y a partir de r y de una constante de distorsión k1 se modifican las coordenadas normalizadas. Son estas nuevas coordenadas normalizadas las que finalmente se usan para calcular la posición en píxeles sobre la imagen. Típicamente k1 es un valor muy pequeño (si k1=0, las coordenadas normalizadas no cambian y por lo tanto no hay distorsión).

Es muy sencillo volver a correr el proceso de optimización anterior para estimar este coeficiente k1 además de la focal f. Basta ampliar el vector de parámetros P para que incluya el valor de k1 como su octava componente y cambiar el código de la función error_uv() para que dicho parámetro k1=P(8), modificando la proyección de los puntos sobre la imagen usando la fórmula anterior.
Adjuntad código de la función error_uv modificada.

Una vez modificada la función del error volver a correr el proceso de optimización incorporando el nuevo parámetro. Para la hipótesis inicial P0, k1 puede hacerse 0, ya que normalmente será un valor pequeño.

¿Valores obtenidos para f y k1? ¿Cuánto bajaba la norma del error en (u,v) desde la estimación inicial cuando solo se usaba f como parámetro de calibración?
¿Y usando f y k1 como parámetros? ¿Qué nos indica esto?

Adjuntad una grafica superponiendo los errores en los tres casos (usando P0 inicial sin optimizar, tras la optimización en f, y tras la optimización en f y k1).

7) Estimación de los parámetros de la cámara de vuestro teléfono (20%)

Indicad la marca y modelo de vuestro teléfono y buscar la información que podáis hallar sobre su cámara:

- Tamaño del sensor en píxeles
- Tamaño físico del sensor (XX x YY mm)
- Focal de la lente (mm).

Para completar la información anterior vais a calibrar vuestro movil 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 movil, similar a la foto con la que hemos trabajado en la práctica. Aseguraros de que está lo mejor enfocada posible y que salen 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.

Al hacer la foto medid también de forma aproximada la distancia entre el movil y el centro de la malla. Adjuntad la foto. ¿Distancia movil- centro malla?

Correr el programa 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 vuestro propio patrón es posible que tengáis que modificar en vuestros programas las coordenadas X e Y, que correspondían a los puntos de la malla sobre el papel y pueden cambiar dependiendo del tamaño con el que salgan los cuadrados de la malla al imprimirse. Alternativamente podéis escalar el fichero al imprimirlo para asegurar que los cuadrados tienen 2cm de lado. Dad vuestros resultados:

- Tamaño de los cuadros en vuestra malla de calibración (DX,DY)
- H estimada para las 4 esquinas y para todos los puntos de la malla.
- Valor de f estimado inicialmente (en pix y mm) a partir de la matriz H.
- Resultado de la optimización usando f o (f+k1) si lo habéis implementado.
- Gráficos de errores en coordenadas (u,v) antes / después de la optimización.
- Valores de R y t obtenidos en la optimización.
- Calcular X0 y determinar la distancia cámara-malla para vuestra foto.