Proyecto proyección 3D / 2D (entrega por parejas)

Ejercicio 1: Vamos a escribir una función que implemente la proyección de datos 3D (XYZ) sobre los píxeles (uv) de la cámara, conocidos sus parámetros extrínsecos (seis datos = posición / pose de la cámara) e intrínsecos (u0, v0, focal, etc).

El template de la función a escribir será el siguiente:

```
function [uv Zc]=Tproj(X,datos3D)
global u0 v0 f
...
return
```

Los argumentos de entrada son:

- X: vector columna con los 6 parámetros extrínsecos: la posición (E0,N0,h0 en metros) de la cámara y los 3 ángulos (az, elev, inv en grados) que definen su orientación.
- datos3D: matriz 3 x N con las coordenadas de los N puntos 3D a proyectar. Cada columna corresponde a las coordenadas (E,N,h) de un punto.

Los parámetros intrínsecos de la cámara (focal, u0,v0) se definen como variables globales que se fijarán desde el programa principal. Lo hacemos así porque en estos ejercicios asumimos que son fijos y conocidos. Los argumentos de salida serán:

- uv una matriz 2 x N con los píxeles donde deben aparecer los puntos 3D sobre la foto. La 1ª fila es la coordenada horizontal y la 2ª la vertical.
- Zc, un vector 1 x N con las distancias del plano conteniendo los objetos 3D al plano de la cámara. Serán positivas si están delante de la cámara.

Los pasos a dar dentro de la rutina serán:

1) Pasar de la coordenadas 3D datos3D a coordenadas de la cámara, trasladando las coordenadas a la posición de la cámara y aplicando la rotación correspondiente. Usar las fórmulas indicadas en las transparencias (recordad pasar de º a radianes antes de calcular senos o cosenos en las rotaciones). El resultado será otra matriz 3xN con las nuevas coordenadas en los ejes de la cámara [Xc; Yc; Zc].

2) Obtener las coordenadas normalizadas $\tilde{x}$ e $\tilde{y}$ de una proyección ideal.

3) Pasar de coordenadas normalizadas a píxeles usando los parámetros intrínsecos de la cámara: la focal f y el centro de la imagen u0 y v0. Dichos parámetros (en píxeles) se fijan desde el programa principal y son accesibles como variables globales.

4) Extraer la componente Zc de las coordenadas cámara (2º argumento de salida).

Adjuntar código de vuestra rutina.

Ejercicio 2: Vamos a probar si vuestra función de proyección funciona. Para ello os daré una serie de datos 3D correspondientes a la zona del Circo de Gredos (Sistema Central, Ávila) junto con una foto del área:

- Haciendo **load datos_gredos** veréis 2 variables, una matriz **datos3D** (3x30) con coordenadas 3D (E,N,h) de 30 puntos de la zona y una lista (**nombres**) con los 30 nombres de dichos accidentes (montañas, collados, refugios, etc.)

- IMGP3047.jpg es una foto (3040x2022) de la zona tomada con una focal de 35mm. El sensor de mi cámara tiene una densidad de 129 píxeles/mm.

La foto fue tomada con los siguientes parámetros extrínsecos:

Posición: Coordenadas 306607 E, 4459599 N, 1766 m de altura.
Orientación: Azimuth = 190.9º, Elevación = 11.5º, Inclinación = 3.8º

Escribir un script donde:

1) Se definan las variables u0, v0 y f como globales: `global u0 v0 f`
y se les asignen (`u0 = …; v0 = …; f = …;`) los valores adecuados (a partir de los datos anteriores).

2) Se carguen los datos 3D y la imagen dada.

3) Usando la función de proyección hallar las posiciones en píxeles donde deberían aparecer los puntos en la foto (uv), así como su cooordenada Zc.

4) Determinar los puntos visibles haciendo: visibles = ( ) & ( ) & ( )...; donde dentro de cada paréntesis debéis poner las condiciones que aseguran visibilidad. ¿Cuántos puntos de los 30 serían visibles en la foto?

5) Quedaros sólo con las coordenadas y nombres de los puntos visibles. Para ello usaremos indexado lógico en MATLAB, haciendo nombres = nombres(visibles);
Hacer lo mismo para los valores de los píxeles uv.

6) Visualizar la imagen con image y usar fc_truesize para que salga con la correcta relación de aspecto. Superponer sobre ella (hold on/ hold off) los puntos seleccionados usando sus coordenadas en píxeles uv. Usar las siguientes opciones en el plot para pintar puntos gordos de color rojo:

```
plot(datos_u,datos_v,'ro','MarkerFaceColor','r','MarkerSize',3);
```

Adjuntar la imagen resultante. ¿Hay algún punto que debería verse pero está tapado por el terreno? Con la información disponible ¿podríamos detectar esa situación?

¿Cómo podríais calcular las distancias (en metros) desde la cámara a los puntos que habéis superpuesto sobre la foto? Hacer un volcado de los puntos visibles dando su nombre y la distancia a la cámara.

Una vez seguros de que vuestro código funciona bien, probad a repetir el ejercicio usando como nueva posición de la cámara: **E0=306533 N0=4459214 h0=1846** con la misma orientación (mismos ángulos). La nueva posición está unos 400-500 metros delante de la anterior (que es la correcta).
Adjuntar la imagen resultante (observar como los puntos ya no caen donde deben).

Si (manteniendo la posición incorrecta) quisierais mejorar el ajuste de los puntos podéis hacerlo "mintiendo" sobre la focal usada al tomar la fotografía. ¿En qué sentido creéis que debéis alterar la focal f? ¿Por qué?
Probad a modificar la focal en el sentido adecuado hasta conseguir un ajuste "decente" con los puntos. Podéis ir cambiando f un par de mm cada vez y volver a correr el programa hasta que los puntos se coloquen de nuevo más o menos donde deben. Adjuntar la imagen final y el valor de f usado.

Finalmente (volviendo a usar la posición y focal correctas) vamos a etiquetar los puntos visibles con su nombre sobre la imagen. Para ello usaremos la funcion:

text(x,y,txt)

que escribe la cadena de texto txt (p.e. txt='ABC') en las coordenadas dadas (x,y).

Los parámetros (x,y,txt) pueden ser un valor para cada uno (en cuyo caso se pone un único texto) o una colección (vectores) de N datos. En este último caso se escriben de forma simultánea las N etiquetas.

Adjuntar la imagen final. Para no solapar la etiqueta con el punto que ya teníais escribir la etiqueta 15 o 20 píxeles a la derecha de la posición del punto.
Añadiendo text(...,'Color','r') a la orden podéis indicar el color a usar en etiquetas.

Ejercicio 3: (determinación de posición/pose cámara con optimización)

El ejercicio anterior estaba "amañado", ya que claramente no he podido medir con tanta precisión mi posición ni los ángulos que definían la pose de la cámara en el momento de tomar la foto. Lo que realmente hice fue resolver el problema inverso: a partir de la info 3D de los puntos visibles y su info 2D sobre los píxeles en los que caen deduje la posición y pose más probable de la cámara que justificaban las observaciones.

Esta es vuestra tarea para este ejercicio: dada una colección de datos 3D y sus correspondientes píxeles 2D medidos sobre la imagen, determinar la posición/pose de la cámara que mejor proyecta unos en otros.

Datos de la imagen a usar:
- **IMGP3029.jpg**. (**3040x2022 píxeles)**
- Densidad de píxeles: **129 pix/mm.**
- Focal usada: **f = 23mm**.

Notad que la focal ha cambiado respecto a la otra foto !!

Sobre la imagen (ver figura adjunta) he marcado con círculos rojos los 4 puntos de control que vamos a usar. De izquierda a derecha son:

3er Hermanito, Casquerazo, Pico de los Huertos, Risco negro.

Dichos puntos corresponden a los índices **6, 10, 24 y 26** de la lista de datos 3D que teníamos de la zona. Extraer sus coordenadas 3D y guardarlas en una matriz 3 x 4.

En el ejercicio anterior suponíamos conocidos los parámetros de la cámara y aplicábamos la proyección obteniendo las coordenadas 2D (píxeles en imagen 2D).
En este ejercicio estamos interesados en el problema inverso: dados datos 3D + píxeles 2D determinar la P más probable tal que P(3D) ≅ 2D.

- Ej2) DATOS: coord 3D + P -> RESULTADOS: píxeles 2D en imagen.
- Ej3) DATOS: coord 3D + pix 2D -> RESULTADOS: parámetros de P.

Lo primero será medir sobre la imagen la posición en píxeles de los puntos de control. Cargar la imagen con imread, visualizarla con image y usar el zoom para ampliar la zona de cada punto de control. Usando el cursor de datos determinar las coordenadas (u,v) en píxeles (u horizontal, v vertical) de los 4 puntos de control. Dados que los puntos de control son montañas, tratar de escoger el píxel correspondiente al punto más alto.

Guardar las 4 coordenadas 2D medidas en una matriz 2 x 4.

Volcar vuestra matriz de datos 2D (2x4) con los píxeles (u,v) de los puntos de control (redondeados como enteros). Para asegurar que no habéis metido la pata superponer los valores anteriores sobre la imagen con un plot usando círculos rojos como hicimos en el ejercicio anterior. Adjuntar imagen resultante.

Para resolver el problema vamos a usar un **método de optimización.** A partir de una hipótesis inicial (posición + pose de cámara) el método irá iterando para mejorar dicha solución.

Un método de optimización necesita una función (función de coste) que le de una idea del error cometido para cada hipótesis. Usaremos una función que devuelva un vector con las discrepancias entre los píxeles que hemos medidos sobre la imagen y los calculados por nuestro modelo de proyección.

Usaréis el siguiente template para dicha función:

```
function dif = discrepancias(X,datos3D,datos2D)
  ...
return
```

Los parámetros de entrada son:

- X: vector columna con los 6 parámetros extrínsecos: la posición (E0,N0,h0 en metros) de la cámara y los 3 ángulos de orientación (az, elev, inv en grados).
- datos3D: matriz 3 x N con las coordenadas 3D de los N puntos de control a proyectar. Cada columna corresponde a la posición (E,N,h) de un punto.
- datos2D: matriz 2 x N con los píxeles 2D de los N puntos de control que acabáis de medir sobre la imagen.

La rutina es sencilla y sólo debe ocupar 3 o 4 líneas:

1) Llama a la función anterior Tproj con los datos 3D y la proyección X y obtiene las posiciones 2D predichos sobre la imagen.

2) Restamos dichos valores de los medidos (argumento de entrada datos2D) para obtener una matriz 2 x N con las diferencias entre píxeles.

3) Finalmente dichas discrepancias deben devolverse en el argumento de salida dif, un vector 2N x 1. Para construir el vector de salida hay varias posibilidades. Una de ellas es extraer la primera fila de las diferencias (Δu) y colocar a su lado la segunda fila (Δv). Al vector fila resultante lo trasponemos para obtener un vector columna.

Adjuntar código de vuestra función discrepancias.m

Como método de optimización usaremos la función suministrada opt_gauss_newton (esta función os la doy yo, no tenéis que codificarla vosotros):

```
function [X fval]=opt_gauss_newton(f,X0)
% f es un puntero a la función a minimizar.
% X0 = vector de partida = vector de n parametros
% El algoritmo trata de hallar X tal que minimize ||f(X)||
```

Parámetros de entrada:

- El parámetro f es un puntero a la función que deseamos minimizar. En este caso será la función discrepancias() que hemos escrito antes.

- X0 es un vector con las hipótesis iniciales de los parámetros. En este caso un vector 6x1 con la posición de cámara (3) + ángulos de orientación (3).

La función se llamaría como:

```
f_ptr=@(X)discrepancias(X,datos3D,datos2D);
[X fval]=opt_gauss_newton(f_ptr,X0);
```

de esta forma le indicamos que de los tres argumentos de la función discrepancias() el primero (X) es sobre el que actuará el algoritmo de minimización.

Para que podáis ir verificando los resultados parciales, la función suministrada sólo hace una iteración del proceso de optimización, devolviendo:

- X = nueva estimación del vector de parámetros de pos + giro cámara.

- fval = valores de la función a minimizar para el nuevo valor X. En este caso será un vector con las discrepancias en píxeles sobre la imagen entre nuestro modelo y la realidad.

Al hacer un solo paso, vosotros deberéis montar un bucle que vuelva a llamar a la función con el nuevo X y vaya monitorizando y enseñando los resultados.

Ya tenemos todo lo necesario para iniciar el proceso de optimización. Lo único que queda es escoger una hipótesis razonable como punto de partida para el vector de (6) parámetros de la cámara:

- **Azimuth:** la foto fue tomada mirando más o menos hacia el Sur. Recordad que el azimuth lo definíamos como el ángulo que nos apartamos de la dirección N.

- **Elevación e Inclinación** se inicializan a cero: supongo cámara nivelada.

- **Altura**: tomar la media de la altura de los puntos de control usados.

- **Posición Este**: como la foto fue tomada mirando al Sur podemos asumir que la coordenada E de la cámara será similar a la media de los puntos de control.

- **Posición Norte:** como la cámara miraba al Sur, eso quiere decir que estaba al norte de las montañas fotografiadas. Podemos hallar la media de las coordenadas N de los puntos de control y le sumaremos 3000 o 4000 metros para situarnos 3 o 4 km. al norte de los objetos que aparecen en la foto.

Dar vuestra estimación inicial para las coordenadas E,N,h y ángulos de giro de la cámara obtenida de acuerdo a las indicaciones anteriores.

Inicializar un vector X0 (6x1) con los valores anteriores (recordad usar grados para los ángulos).

Ahora es sólo cuestión de iterar el bucle de optimización. En cada paso:

1) Llamar a la función de optimización con la hipótesis actual X0. La función os devolverá el nuevo valor X mejorado y las discrepancias encontradas.

2) Volcar por pantalla los nuevos parámetros (precisión de metros en posición y de décimas de grado en ángulos).

3) Calcular dX = (X-X0) = variación de los parámetros.

Si todo va bien veréis disminuir las discrepancias en cada paso de la iteración.

Repetir el bucle hasta que dX sea suficientemente pequeño. Por ejemplo que el máximo (en valor absoluto) de sus componentes no supere 0.1.

Tras terminar la iteración presentar los resultados finales:

- Los parámetros definitivos de la cámara (posiciones en metros/ángulos en º).
- Las discrepancias no corregidas entre píxeles medidos y proyectados.
- La media del valor absoluto de dichos errores.

Finalmente mostrar la imagen y superponer sobre ella los píxeles medidos (en rojo) y los píxeles proyectados (en azul) con la mejor estimación para la posición y pose de la cámara.

Adjuntar la imagen completa y algún zoom que muestre los errores gráficamente en algunos de los puntos de control.

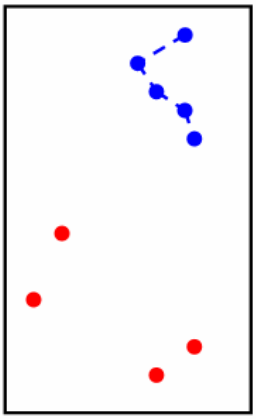
Posibles extras para mejorar la práctica

Al dar los valores de las discrepancias en los píxeles y su media, separar por errores en las coordenadas horizontales (u) y verticales (v).

Presentar varias imágenes como la pedida (foto + píxeles medidos + píxeles proyectados) para la primera hipótesis y un par de los resultados intermedios ilustrando para ilustrar los avances durante el proceso de optimización.

Para destacar la diferencia entre los píxeles medidos y los proyectados unirlos con rectas como en la foto adjunta. Las discrepancias serán máximas al principio y se irán reduciendo durante la optimización.

Guardar las posiciones de la cámara en las diferentes iteraciones y mostrar su evolución en un gráfico de superficie (E = eje X, N = eje Y), junto con los puntos de control. Algo así como la figura adjunta, donde en rojo mostramos los puntos de control y en azul las sucesivas iteraciones de la posición de la cámara.



Suponed que nos equivocamos al introducir la focal de la cámara y usamos f=35mm (la de la foto del ejercicio 1) en vez de la correcta f=23mm.

Volver a correr el código para la focal incorrecta f=35mm. ¿Converge el algoritmo de optimización? Adjuntar imagen final con los píxeles medidos + proyectados para esta focal. ¿De qué orden son ahora las discrepancias no explicadas?

La posición encontrada ahora con la nueva focal, ¿va a estar más cerca o más lejos que la correcta? ¿Por qué?

Si no conocemos la focal usada en la cámara podemos estar tentados de añadirla como una incógnita adicional y dejar que el algoritmo de optimización la encuentre junto con la posición y pose de la cámara. Aunque matemáticamente el problema se podría plantear, en la práctica no funcionaría muy bien. Con lo que hemos visto en la práctica ¿se os ocurre alguna idea de cuál podría ser el problema?