Práctica 2. Visión Estéreo

Visión Tridimensional 2020-21.
Practica 2. Abril 2021.

Este enunciado está en el archivo "PrácticaStereo2021_Alumnos.ipynb" o su versión "pdf" que puedes encontrar en el Aula Virtual.

Objetivos

Los objetivos de esta práctica son:

- reconstruir puntos de una escena a partir de una serie de correspondencias manuales entre dos imágenes calibradas;
- determinar la geometría epipolar de un par de cámaras a partir de sus matrices de proyección;
- hacer una reconstrucción densa de la escena.

Requerimientos

Para esta práctica es necesario disponer del siguiente software:

- Python 3.X
- Jupyter http://jupyter.org/.
- Las librerías científicas de Python: NumPy, SciPy, y Matplotlib.
- La librería OpenCV

El material necesario para la práctica se puede descargar del Aula Virtual en la carpeta `MaterialesPractica` del tema de visión estéreo. Esta carpeta contiene:

- Una serie de pares estéreo en el directorio images; el sufijo del fichero indica si corresponde a la cámara izquierda (_left) o a la derecha (_right). Bajo el directorio `rectif` se encuentran varios pares estéreo rectificados.
- Un conjunto de funciones auxiliares de `Python` en el módulo `misc.py`. La descripción de las funciones puede consultarse con el comando help o leyendo su código fuente.
- El archivo `cameras.npz` con las matrices de proyección del par de cámaras con el que se tomaron todas las imágenes con prefijo minoru.

Condiciones

- La fecha límite de entrega será el viernes **10 de mayo de 2021 a las 23:55** (en el Aula Virtual)
- La entrega consiste en dos archivos con el código, resultados y respuestas a los ejercicios:
  1. Un "notebook" de Jupyter con los resultados. Las respuestas a los ejercicios debes introducirlas en tantas celdas de código o texto como creas necesarias, insertadas inmediatamente después de un enuciado y antes del siguiente.
  2. Un documento "pdf" generado a partir del fuente de Jupyter, por ejemplo usando el comando `jupyter nbconvert --execute --to pdf notebook.ipynb`, o simplemente imprimiendo el "notebook" desde el navegador en la opción del menú "File->Print preview". Asegúrate de que el documento "pdf" contiene todos los resultados correctamente ejecutados.

1. Introducción

En los problemas de visión estéreo se supondrá la existencia de un par de cámaras calibradas cuyas matrices de proyección $\mathbf{P}_i$ vienen dadas por

$$\mathbf{P}_1 = \mathbf{K}_1 \cdot [\mathbf{I} \quad \mathbf{0}] \cdot \begin{bmatrix} \mathbf{R}_1 & \mathbf{t}_1 \\ \mathbf{0}^T & 1 \end{bmatrix},$$

$$\mathbf{P}_2 = \mathbf{K}_2 \cdot [\mathbf{I} \quad \mathbf{0}] \cdot \begin{bmatrix} \mathbf{R}_2 & \mathbf{t}_2 \\ \mathbf{0}^T & 1 \end{bmatrix}.$$

En esta práctica se usarán las matrices de proyección de dos cámaras para determinar la posición tridimensional de puntos de una escena. Esto es posible siempre que se conozcan las proyecciones de cada punto en ambas cámaras. Desafortunadamente, esta información no suele estar disponible y para obtenerla es preciso emplear el contenido de las imágenes (sus píxeles) en un proceso de búsqueda conocido como puesta en correspondencia. Conocer las matrices de proyección de las cámaras permite acotar el área de búsqueda gracias a las restricciones que proporciona la geometría epipolar.

In [1]:

```
import numpy as np
import cv2
import numpy.linalg as npla
import misc
```

1. Reconstrucción

Teniendo un conjunto de correspondencias entre dos imágenes, con matrices de calibración P_i conocidas, es posible llevar a cabo una reconstrucción tridimensional de dichos puntos. En el fichero `cameras.npz` se encuentran las matrices de proyección para las dos cámaras. Para cargar este fichero:

In [2]:

```
cameras = np.load("cameras.npz")
P1 = cameras["left"]
P2 = cameras["right"]

print("P1=\n", P1)
print("P2=\n", P2)
```

```
P1=
 [[-1.59319023e+02  4.10068927e+02 -8.61429776e+01  5.96021124e+04]
 [ 9.56736123e+01 -6.85256589e+00 -4.31511155e+02  2.98592912e+04]
 [-8.69896273e-01 -7.51069223e-02 -4.87482742e-01  5.44164509e+02]]
P2=
 [[-1.49296958e+02  4.20482251e+02 -8.03699899e+01  2.66669558e+04]
 [ 9.61686671e+01 -2.92284678e+00 -4.41950717e+02  3.12991880e+04]
 [-8.64354364e-01 -5.83462724e-02 -4.99486983e-01  5.42414607e+02]]
```

In [3]:

```
# uncomment to show results in a window
%matplotlib tk
# %matplotlib notebook
import matplotlib.pyplot as plt
```

Todas las imágenes con el prefijo minoru comparten este par de matrices de proyección.

Leemos las imágenes y marcamos al menos seis puntos correspondientes en cada una de ella.

In [4]:

```
img1 = cv2.imread("images/minoru_cube3_left.jpg")
img2 = cv2.imread("images/minoru_cube3_right.jpg")

# plt.figure()
# plt.imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))
# plt.figure()
# plt.imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))
```

In [5]:

```
# pt1, pt2 = misc.askpoints(img1,img2)

# Recomendación: Una vez marcados la primera vez con toda la precisión
# posible, generar dos arrays de numpy aqui, pt1 y pt2, con las
# coordenadas marcadas (para no tener que volver a marcarlas).
# Una vez colocadas esas variables ¡comentar el código que llama a
# miscaskpoints!

pt1 = np.array([[202.56676669, 105.1264066 , 207.05588897, 304.76031508,
        288.65228807, 202.56676669, 115.16091523],
       [  2.73396849,  23.5951838 ,  56.60343586,  25.17957989,
        151.40313578, 206.59293323, 150.0828057 ],
       [  1.        ,   1.        ,   1.        ,   1.        ,
          1.        ,   1.        ,   1.        ]])

pt2 = np.array([[117.22543136,   5.26144036,  82.89684921, 205.42348087,
        200.93435859,  96.89234809,  25.59452363],
       [  5.63869467,  27.82024006,  61.88475619,  28.34837209,
        158.26885221, 215.30711178, 154.57192798],
       [  1.        ,   1.        ,   1.        ,   1.        ,
          1.        ,   1.        ,   1.        ]])
```

In [6]:

```
plt.figure()
plt.imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))
plt.plot(pt1.T[:,0], pt1.T[:,1])
```

```
Out[6]: [<matplotlib.lines.Line2D at 0x27433076188>]
```

Ejercicio 1. Implementa la función `M = reconstruct(points1, points2, P1, P2)` que, dados una serie de N puntos 2D `points1` de la primera imagen y sus N homólogos `points2` de la segunda imagen (ambos en coordenadas homogéneas, 3 × N), y el par de matrices de proyección P1 y P2 de la primera y la segunda cámara respectivamente, calcule la reconstrucción tridimensional de cada punto. De ese modo, si `points1` y `points2` son 3 × N , la matriz resultante M debe ser 4 × N.

El tipo de reconstrucción debe ser algebraico, no geométrico.

In [7]:

```
def reconstruct(points1, points2, P1, P2):
# """Reconstruct a set of points projected on two images."""

# Transform homog to cartesian co-ordinates
    points1 = points1.T
    points2 = points2.T

    p11 = P1[0]
    p12 = P1[1]
    p13 = P1[2]

    p21 = P2[0]
    p22 = P2[1]
    p23 = P2[2]

    Ms = []

    for p1, p2 in zip(points1, points2):
        i1, j1 = p1[1], p1[0]
        i2, j2 = p2[1], p2[0]

        A = np.array([
            [(p11 - j1*p13)],
            [(p12 - i1*p13)],
            [(p21 - j2*p23)],
            [(p22 - i2*p23)]
        ])

        # reshape (4,1,4) -> (4,4)
        A = np.squeeze(A)

        A_ = A[:,0:3]
        b = -A[:,3].T

        # build coefficient matrix and compute reconstruction by least-squares.
        # Useful functions are npla.lstsq() and npla.pinv()
        M = npla.lstsq(A_, b, rcond=None)[0]

        Ms.append(M)

    # Added to make them homog
    ones = np.ones((1,len(points1)))

    M = np.array(Ms).T
    M = np.vstack((M, ones))

    return M
```

Reconstruye los puntos marcados y pinta su estructura 3D.

In [8]:

```
# reconstruct
mM = reconstruct(pt1, pt2, P1, P2)
# convert from homog to cartesian

# plot 3D
plt.figure()
misc.plot3D(mM[0,:],mM[1,:],mM[2,:])
```

```
Out[8]: <Axes3D:xlabel='X', ylabel='Y'>
```

Ejercicio 2. Elige un par estéreo de las imágenes del conjunto "building" de la práctica de calibración y realiza una reconstrucción de un conjunto de puntos de dicho edificio estableciendo las correspondencias a mano.

En este caso tenemos la cámara calibrada dado que las imágenes las hemos capturado con la misma cámara que en la práctica de calibración. Nos faltarían la posición relativa entre una cámara y la otra. Utilizar algunas funciones de OpenCV en el módulo de calibración de calib3d puede ser de gran ayuda.

In [9]:

```
# De la práctica de calibración
K_building = np.array([
    [3.20506009e+03, 0.00000000e+00, 1.97863570e+03],
    [0.00000000e+00, 3.20506009e+03, 1.45074623e+03],
    [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]
])
# No usar los parámetros de distorsión radial!!

img1_building = cv2.imread("building/build_001.jpg") # izda
img2_building = cv2.imread("building/build_002.jpg") # dcha

# Get points in both images
# plt.figure()
# plt.imshow(cv2.cvtColor(img1_building, cv2.COLOR_BGR2RGB))
# plt.figure()
# plt.imshow(cv2.cvtColor(img2_building, cv2.COLOR_BGR2RGB))
```

In [10]:

```
# pt1_building, pt2_building = misc.askpoints(img1_building,img2_building)

pt2_building = np.array([[1.78733256e+03, 9.35983721e+02, 8.18104651e+02, 1.81025349e-
        2.77948140e+03, 2.94647674e+03, 3.12984419e+03, 3.18878372e+03,
        1.80043023e+03, 1.79388140e+03, 6.41286047e+02,
        5.98718605e+02],
       [1.11638419e+02, 8.41833767e+02, 2.06646633e+03, 1.75212214e+03, 1.11638419e+0
        7.82894233e+02, 2.00097795e+03, 2.09266167e+03, 2.44302447e+03,
        2.40700586e+03, 1.83398260e+03, 2.17124772e+03,
        2.52488493e+03],
       [1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+00,
        1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+00,
        1.00000000e+00, 1.00000000e+00, 1.00000000e+00,
        1.00000000e+00, 1.00000000e+00]]).astype(np.float64)

pt1_building = np.array([[1.97337907e+03,1.13840233e+03, 1.01724884e+03, 1.99302558e+
        2.98190000e+03, 3.14889535e+03, 3.34208605e+03, 3.40430000e+03,
        1.98320233e+03, 1.97665349e+03, 8.46979070e+02,
        8.04411628e+02],
       [1.31284930e+02, 8.71303535e+02, 2.07628958e+03, 1.77176865e+03, 1.31284930e+0
        7.82894233e+02, 2.02389888e+03, 2.12213144e+03, 2.48231749e+03,
        2.42665237e+03, 1.85035470e+03, 2.18107098e+03,
        2.52815935e+03],
       [1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+00,
        1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+00,
        1.00000000e+00, 1.00000000e+00, 1.00000000e+00,
        1.00000000e+00, 1.00000000e+00]]).astype(np.float64)
```

In [11]:

```
# plt.figure()
# plt.imshow(cv2.cvtColor(img1_building, cv2.COLOR_BGR2RGB))
# plt.plot(pt1_building.T[:,0], pt1_building.T[:,1])
```

In [12]:

```
# plt.figure()
# plt.imshow(cv2.cvtColor(img2_building, cv2.COLOR_BGR2RGB))
# plt.plot(pt2_building.T[:,0], pt2_building.T[:,1])
```

In [13]:

```
# Encontrar P1_building y P2_building
# (matrices de proyección de las dos imágenes seleccionadas)

rhs =  np.column_stack( ( np.eye(3) ,np.zeros((3,1)) ) )
P1_building = K_building @ rhs

print("P1_building:\n", P1_building)


# ref: https://docs.opencv.org/4.5.1/d9/d0c/group__calib3d.html#ga13f7e34de8fa516a686
E, _ = cv2.findEssentialMat(pt1_building[:-1].T, pt2_building[:-1].T, K_building)

_, R2, t = cv2.decomposeEssentialMat(E)
rhs2 =  np.column_stack( ( R2 ,t ) )

P2_building = K_building @ rhs2

print("P2_building:\n", P2_building)


# # reconstruct
mM_building = reconstruct( pt1_building, pt2_building, P1_building , P2_building )

# plot 3D
plt.figure()
misc.plot3D(mM_building[0,:],mM_building[1,:],mM_building[2,:], azim=90, elev=-90)
```

```
P1_building:
 [[3.20506009e+03 0.00000000e+00 1.97863570e+03 0.00000000e+00]
 [0.00000000e+00 3.20506009e+03 1.45074623e+03 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00 0.00000000e+00]]
P2_building:
 [[ 3.30637548e+03  2.70778914e+01  1.80403919e+03  1.32391017e+03]
 [ 8.04470310e+01  3.22860332e+03  1.39525041e+03 -2.63974873e+03]
 [ 5.35275184e-02  1.65337598e-02  9.98429487e-01  2.43123227e-01]]
Out[13]: <Axes3D:xlabel='X', ylabel='Y'>
```

In [14]:

```
# El resto de la práctica lo podemos hacer con los datos de las
# dos imágenes de prefijo minoru o las dos seleccionadas del directorio
# building

usar_par_estereo_building = False

if usar_par_estereo_building:
    P1 = P1_building
    P2 = P2_building
    img1 = img1_building
    img2 = img2_building
    pt1 = pt1_building
    pt2 = pt2_building
    K = K_building
    mM = mM_building
```

Ejercicio 3. Reproyecta los resultados de la reconstrucción en las dos cámaras y dibuja las proyecciones sobre las imágenes originales. Pinta también en otro color los puntos seleccionados manualmente. Comprueba si las proyecciones coinciden con los puntos marcados a mano. Comenta los resultados. Para dibujar los puntos puedes usar la función plothom de la práctica anterior o la versión que se distribuye con esta práctica (misc.plothom).

In [15]:

```
# Proyecto los puntos en ambas cámaras
proy1 = P1 @ mM
proy2 = P2 @ mM

# Pinto con misc.plothom()
plt.figure()
misc.plothom(proy1,'r.')
plt.plot(pt1.T[:,0], pt1.T[:,1], 'bx')
plt.imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))
plt.show()

plt.figure()
misc.plothom(proy2,'r.')
plt.plot(pt2.T[:,0], pt2.T[:,1], 'bx')
plt.imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))
plt.show()
```

2. Geometría epipolar

La geometría epipolar deriva de las relaciones que aparecen en las proyecciones de una escena sobre un par de cámaras. La matriz fundamental $\mathbf{F}$, que depende exclusivamente de la configuración de las cámaras y no de la escena que éstas observan, es la representación algebráica de dicha geometría: a partir de ella se pueden calcular los epipolos y las líneas epipolares. La relación entre un par de cámaras $\mathbf{P}_1, \mathbf{P}_2$ y la matriz fundamental es de n -a- 1 (salvo factor de escala). Es decir, dadas dos cámaras calibradas, sólo tienen una matriz fundamental (excepto un factor de escala); dada una matriz fundamental existen infinitas configuraciones de cámaras posibles asociadas a ella.

2.1 Estimación de la matriz fundamental

Ejercicio 4. Implementa la función `F = projmat2f(P1, P2)` que, dadas dos matrices de proyección, calcule la matriz fundamental asociada a las mismas. $\mathbf{F}$ debe ser tal que, si m_1 de la imagen 1 y m_2 de la imagen 2 están en correspondencia, entonces $m_2^\top F m_1 = 0$.

In [16]:

```
def projmat2f(P1, P2):
    """ Calcula la matriz fundamental a partir de dos matrices de proyeccion"""

    K1, R1, t1, _, _, _, _= cv2.decomposeProjectionMatrix(P1)
    K2, R2, t2, _, _, _, _= cv2.decomposeProjectionMatrix(P2)

    # Correct ts
    t1 /= t1[-1]
    t2 /= t2[-1]

    t1 = t1[:-1]
    t2 = t2[:-1]

    t1 = -R1 @ t1
    t2 = -R2 @ t2

    e = -K2 @ R2 @ R1.T @ t1 + K2 @ t2 # epipolo generico

    if e[-1] != 0:
        e /= e[-1] # epipolo normalizado

    print("Epipolo:", e.T)

    F = misc.skew(e) @ P2 @ npla.pinv(P1)

    if F[2,2] != 0:
        return F / F[2,2]
    else:
        return F
```

In [17]:

```
# compute Fundamental matrix
F = projmat2f(P1, P2)
print("F =\n", F)
```

```
Epipolo: [[ 5.44542858e+03 -1.08576575e+04  1.00000000e+00]]
F =
 [[ 6.18328300e-08  3.18506753e-07  3.08058423e-03]
 [-2.81757724e-07  5.88352229e-09  1.63710279e-03]
 [-3.39593513e-03 -1.67052450e-03  1.00000000e+00]]
```

Ejercicio 5 ¿Cómo es la matriz fundamental de dos cámaras que comparten el mismo centro? (Por ejemplo, dos cámaras que se diferencian sólo por una rotación.)

Una matriz de zeros.

Comprobación para el caso particular: t1 = 0.

K1 = np.array([[4.22521094e+02, 0.0, 1.49785224e+02], [0.00000000e+00, 4.23213410e+02, 1.27642797e+02], [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]]) K2 = np.array([[4.29684742e+02, 0.0, 1.44655669e+02], [0.00000000e+00, 4.30801382e+02, 1.37795360e+02], [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]]) R1 = np.array([[-0.06868631, 0.99715456, -0.03106417], [0.48842877, 0.00646079, -0.87257985], [-0.86989627, -0.07510692, -0.48748274]]) R2 = np.array([[-0.05646744, 0.99822574, -0.01888912], [0.49970287, 0.01187786, -0.86611544], [-0.86435436, -0.05834627, -0.49948698]]) t1 = np.array([[0],[0],[0]]) t2 = t1 x = skew(np.squeeze(R2.T @ t2 - R1.T @ t1)) f1 = npla.inv(K2).T @ R2 @ x f2 = R1.T @ npla.inv(K1) F = f1 @ f2 F # array([[0., 0., 0.], # [0., 0., 0.], # [0., 0., 0.]])

2.2 Comprobación de F

En los siguientes dos ejercicios vamos a comprobar que la matriz F estimada a partir de P1 y P2 es correcta.

Ejercicio 6. Comprueba que F es la matriz fundamental asociada a las cámaras `P1` y `P2`. Para ello puedes utilizar el resultado 9.12, que aparece en la página 255 del libro Hartley, Zisserman. "Multipe View Geometry in Computer Vision." (sedond edition). Cambridge University Press, 2003:

"A non zero marix F is a fundamental matrix corresponding to a pair of camera matrices P and P' if and only if $P'^T F P$ is skew symmetric"

In [18]:

```
#  skew symmetric: -A=A.T
c = P2.T @ F @ P1

# redondeo para mitigar el error numerico
c = np.around(c, 5)

if (-c == c.T).all():
    print("F es matriz fundamental\n", np.around(F, 5))
else:
    print("F no es matriz fundamental\n", np.around(F, 5))
```

```
F es matriz fundamental
 [[ 0.       0.       0.00308]
 [-0.       0.       0.00164]
 [-0.0034  -0.00167  1.     ]]
```

También se puede comprobar geométricamente la bondad de una matriz F, si las epipolares con ella estimadas pasan por el homólogo de un punto dado en una de las imágenes.

Dada la matriz fundamental $\mathbf{F}$ entre las cámaras 1 y 2, se puede determinar, para un determinado punto m_1 en la imagen de la cámara 1, cuál es la recta epipolar l_2 donde se encontrará su homólogo en la cámara 2:

$$l_2 = \mathbf{F} m_1.$$

Las siguientes dos funciones sirven para comprobar esta propiedad. En primer lugar, se necesita una función que dibuje rectas expresadas en coordenadas homogéneas, es decir, la versión de plothom para rectas en lugar de puntos.

Ejercicio 7. Implementa la función `plothline(line)` que, dada una línea expresada en coordenadas homogéneas, la dibuje.

In [19]:

```
def plothline(line, axes = None):
    """Plot a line given its homogeneous coordinates.

    Parameters
    ----------
    line : array_like
        Homogeneous coordinates of the line.
    axes : AxesSubplot
        Axes where the line should be plotted. If not given,
        line will be plotted in the active axis.
    """
    if axes == None:
        axes = plt.gca()

    [x0, x1, y0, y1] = axes.axis()

    #     (x0, y0) ._____________________. (x1, y0)
    #              |                     |
    #              |                     |
    #              |                     |
    #              |                     |
    #              |                     |
    #              |                     |
    #     (x0, y1) .---------------------. (x1, y1)


    # POR HACER: Compute the intersection of the line with the image
    # borders.

    a, b, c = line

    yy0 = -(c+a*x0) / b
    yy1 = -(c+a*x1) / b

    plotline = axes.plot([x0, x1], [yy0, yy1], 'r-')

    axes.axis([x0, x1, y0, y1])
    return plotline
```

Ejercicio 8. Completa la función `plot_epipolar_lines(image1, image2, F)` que, dadas dos imágenes y la matriz fundamental que las relaciona, pide al usuario puntos en la imagen 1 y dibuje sus correspondientes epipolares en la imagen 2 usando `plothline`.

In [20]:

```
def plot_epipolar_lines(image1, image2, F):
    """Ask for points in one image and draw the epipolar lines for those points.

    Parameters
    ----------
    image1 : array_like
        First image.
    image2 : array_like
        Second image.
    F : array_like
        3x3 fundamental matrix from image1 to image2.
    """
    # Prepare the two images.
    fig = plt.gcf()
    fig.clf()
    ax1 = fig.add_subplot(1, 2, 1)
    ax1.imshow(cv2.cvtColor(image1, cv2.COLOR_BGR2RGB))
    ax1.axis('image')
    ax2 = fig.add_subplot(1, 2, 2)
    ax2.imshow(cv2.cvtColor(image2, cv2.COLOR_BGR2RGB))
    ax2.axis('image')
    plt.draw()

    ax1.set_xlabel("Choose points in left image (or right click to end)")
    point = plt.ginput(1, timeout=-1, show_clicks=False, mouse_pop=2, mouse_stop=3)
    while len(point) != 0:
        # point has the coordinates of the selected point in the first image.
        point = np.hstack([np.array(point[0]), 1])
        ax1.plot(point[0], point[1], '.r')

        # POR HACER: Determine the epipolar line.
        line = F @ point

        # Plot the epipolar line with plothline (the parameter 'axes' should be ax2).
        plothline(line, axes=ax2)
        plt.draw()

        # Ask for a new point.
        point = plt.ginput(1, timeout=-1, show_clicks=False, mouse_pop=2, mouse_stop=

    ax1.set_xlabel('')
    plt.draw()
```

Utiliza esta función con un par de imágenes llamándola de dos formas diferentes: seleccionando puntos en la imagen izquierda y dibujando las epipolares en la imagen derecha y viceversa. Comprueba en ambos casos que las epipolares siempre pasan por el punto de la segunda imagen correspondiente al seleccionado en la primera. Esto confirmara la corrección de la matriz F.

Añade dos figuras una que muestre la selección de puntos en la imagen izquierda y las rectas correspondientes en la imagen derecha, y otra que lo haga al revés. Indica para ambos casos qué matriz fundamental has usado al llamar a `plot_epipolar_lines`.

In [21]:

```
plot_epipolar_lines(img1, img2, F)
```

3. Rectificación de imágenes

La mayoría de algoritmos de puesta en correspondencia, incluyendo el que se va a implementar en esta práctica, requieren que las imágenes de entrada estén rectificadas.

Dos imágenes están rectificadas si sus correspondientes epipolares están alineadas horizontalmente. La rectificación de imágenes facilita enormemente los algoritmos de puesta en correspondencia, que pasan de ser problemas de búsqueda bidimensional a problemas de búsqueda unidimensional sobre filas de píxeles de las imágenes. En el material de la práctica se han incluido dos funciones que rectifican (mediante un método lineal) dos imágenes. La función `H1, H2 = misc.projmat2rectify(P1, P2, imsize)` devuelve, dadas las dos matrices de proyección y el tamaño de las imágenes en formato (filas,columnas), las homografías que rectifican, respectivamente, la imagen 1 y la imagen 2. La función `projmat2rectify` hace uso de `projmat2f`, por lo que es necesario que esta función esté disponible.

Ejercicio 9. Se tienen dos imágenes no rectificadas `im1` e `im2`, y su matriz fundamental asociada $\mathbf{F}$. Con el procedimiento explicado, se encuentran un par de homografías $\mathbf{H}_1$ y $\mathbf{H}_2$ que dan lugar a las imágenes rectificadas `O1` y `O2`. ¿Cuál es la matriz fundamental $\mathbf{F}'$ asociada a estas dos imágenes? ¿Por qué?

Nota: F ' depende exclusivamente de F , H1 y H2.

F' = H2.T @ F @ pinv(H1).T

Ejercicio 10. Rectifica el par de imágenes estéreo `img1` e `img2` implementando el algoritmo de `Fusiello, Trucco y Verri` visto en clase.

Para este ejercicio puede ser útil la función `cv2.decomposeProjectionMatrix`.

In [22]:

```
def projmat2rectify_fusiello(P1, P2):
    """
    Determine the transformation for the epipolar rectification.

    Given the projection matrices of an stereo pair and the size
    of the images from the cameras, this function returns a pair
    of linear transformations (i.e., homographies) which rectify
    the images so that the epipolar lines correspond to the scanlines.

    "A Compact Algorithm for Rectification of Stereo Pairs"
    Andrea Fusiello, Emanuele Trucco, Alessandro Verri.
    IJCV 2000

    Parameters
    ----------
    P1, P2 : ndarray
        Projection matrices of the cameras.
    imsize : tuple
        The size of the image (height, width)
    """

    # ref: https://www.researchgate.net/publication/2471375_Rectification_With_Uncons

    K1, R1, t1, _, _, _, _ = cv2.decomposeProjectionMatrix(P1)
    K2, R2, t2, _, _, _, _ = cv2.decomposeProjectionMatrix(P2)

    # Correct ts
    t1 /= t1[-1]
    t2 /= t2[-1]

    t1 = t1[:-1]
    t2 = t2[:-1]

    # Kn
    Kn = (K1 + K2) / 2
    Kn[0,1] = 0

    # Rn
    r1n = (t2-t1).T
    r2n = np.cross(R1[-1], r1n)
    r3n = np.cross(r1n, r2n)

    Rn = np.vstack((
        r1n / np.linalg.norm(r1n),
        r2n / np.linalg.norm(r2n),
        r3n / np.linalg.norm(r3n)
    ))

    # Homografics
    H1 = Kn @ Rn @ R1.T @ npla.inv(K1)
    H2 = Kn @ Rn @ R2.T @ npla.inv(K2)

    # New proyection matrix
    Pn1 = Kn @ np.column_stack((Rn, -Rn @ t1))
    Pn2 = Kn @ np.column_stack((Rn, -Rn @ t2))

    return H1.astype("float64"), H2.astype("float64"), Pn1, Pn2
```

In [23]:

```
H1, H2, Pn1, Pn2 = projmat2rectify_fusiello( P1, P2 )
print("H1=\n", H1)
print("H2=\n", H2)
O1 = cv2.warpPerspective(img1, H1, (img1.shape[1], img1.shape[0]))
O2 = cv2.warpPerspective(img2, H2, (img2.shape[1], img2.shape[0]))
```

```
H1=
 [[-3.17516863e-01  1.07454140e+00  2.05518462e+02]
 [-9.90328163e-01 -1.77385218e-01  3.62367404e+03]
 [-2.15887837e-05  7.30608184e-05  9.06455220e-01]]
H2=
 [[-3.37918354e-01  1.06823026e+00  2.55726017e+02]
 [-1.01208197e+00 -1.85511719e-01  3.49699737e+03]
 [-3.78751423e-05  6.80081045e-05  9.44655088e-01]]
```

In [24]:

```
plt.figure()
plt.imshow(cv2.cvtColor(O1, cv2.COLOR_BGR2RGB))
plt.title("O1")
plt.figure()
plt.imshow(cv2.cvtColor(O2, cv2.COLOR_BGR2RGB))
plt.title("O2")
```

```
Out[24]: Text(0.5, 1.0, 'O2')
```

Ejercicio 11. Calcula y muestra la matriz fundamental de las imágenes rectificadas. Justifica el resultado obtenido (mira la sección 9.3.1 del libro de Hartley y Zisserman, pág. 248 y 249).

In [25]:

```
# In page 249

# Fr= np.array([
#     [0,0,0],
#     [0,0,-1],
#     [0,1,0]
# ])

Fr = projmat2f( Pn1, Pn2 )

Fr=Fr/Fr[2,1]
print("Fr=\n", np.around(Fr, 5))
```

```
Epipolo: [[-1.15474459e+20  1.05945103e+03  1.00000000e+00]]
Fr=
 [[ 0.  0. -0.]
 [-0. -0. -1.]
 [ 0.  1. -0.]]
```

Ejercicio 12. Usa `plot_epipolar_lines` para dibujar varias líneas epiplares de las imágenes rectificadas. Muestra los resultados.

In [26]:

```
plot_epipolar_lines( O1, O2, Fr )
```

4. Búsqueda de correspondencias

La búsqueda de correspondencias consigue establecer automáticamente las correspondencias de puntos entre dos imágenes (lo que se ha hecho manualmente en el ejercicio 2) haciendo uso de las restricciones que proporciona la geometría epipolar.

4.1 Cálculo de las medidas de semejanza

Una vez rectificadas las dos imágenes de un par estéreo, se pueden buscar las correspondencias. Una matriz de disparidades $\mathbf{S}$ indica, para cada píxel de la imagen 1 rectificada, a cuántos píxeles de diferencia está su correspondencia en la imagen 2 rectificada. En la práctica, para simplificar el problema, podemos considerar que los elementos de $\mathbf{S}$ son enteros. Para el píxel en la posición (x, y) en la imagen 1, su correspondiente está en $(x + S[y, x], y)$ en la imagen 2. Si $S[y, x] < 0$, la correspondencia está hacia la izquierda; si $S[y, x] > 0$, la correspondencia está hacia la derecha; si $S[y, x] = 0$, las coordenadas de los dos puntos coinciden en ambas imágenes.

La búsqueda de correspondencias requiere ser capaz de determinar el parecido visual entre píxeles de dos imágenes. Si los píxeles m_1 y m_2 son visualmente parecidos, tienen más probabilidad de estar en correspondencia que otros que sean visualmente diferentes. Como la apariencia (el nivel de gris) de un único píxel es propensa al ruido y poco discriminativa, el elemento de puesta en correspondencia será una ventana centrada en el píxel. Dado un píxel m de una imagen, llamaremos vecindad del píxel de radio K al conjunto de píxeles de la imagen que se encuentren dentro de una ventana de tamaño $(2K + 1) \times (2K + 1)$ píxeles centrada en m . El número de píxeles de una vecindad de radio K es $N = (2K + 1)^2$. Dadas dos vecindades w_1 y w_2 de dos píxeles, el parecido visual entre ellas puede calcularse con la suma de *diferencias al cuadrado (SSD)* de cada una de sus componentes

$$d_{SSD}(\mathbf{v}, \mathbf{w}) = \sum_{i=1}^{N} (\mathbf{v}_i - \mathbf{w}_i)^2.$$

La distancia d_{SSD} es siempre positiva, es pequeña cuando dos ventanas son visualmente parecidas y grande en caso contrario.

4.2 Estimación de la disparidad sin regularizar

La implementación de cálculo de la disparidad para cada píxel pasa por obtener una matriz D con las distancias entre regiones a lo largo de la línea epipolar. El valor en D será un array de tamaño $M \times N \times L$, donde L es el número de disparidades a explorar, `L = len(disps)`; M y N son, respectivamente, el número de filas y de columnas de las imágenes de entrada. El elemento `D[y,x,l]` debe ser la distancia entre la ventana centrada en `im1[y,x]` y la ventana centrada en `im2[y,x + disps[l]]`.

`D[y,x,l]` debe ser muy grande para aquellos valores en los que `im2[y,x + disps[l]]` no esté definido, es decir, el índice `(y,x+disps[l])` se sale de la imagen 2.

La matriz D proporciona los costes unitarios D_i de una función de energía sin regularización de la forma

$$E(x) = \sum_i D_i(x_i),$$

donde $D_i(l)$ viene dado por $D[y, x, l]$, suponiendo que el píxel i tenga coordenadas (x, y). Las variables $x = (x_1, \ldots, x_{NM})$ indican las etiquetas de cada uno de los píxeles. En este caso, las etiquetas son los índices del array `disps`, que a su vez son las disparidades horizontales. Por eso, a partir de aquí se hablará indistintamente de etiquetas y disparidades. Sólo es necesario recordar que la etiqueta l está asociada a la disparidad `disps[l]`.

Minimizando la energía $x = \arg\min_x E(x)$, se obtiene un vector de etiquetas óptimo x^* que indica, para cada píxel, cuál es su disparidad horizontal entre las dos imágenes.

4.3 Estimación de la disparidad regulariazada: StereoSGBM

Una vez rectificadas las dos imágenes de un par estéreo, y dadas las medidas de semejanza entre píxeles a lo largo de la línea epipolar para un rango de disparidades dado, se puede modificar la función de energía $E(x)$ para añadir términos de regularización. La idea es que píxeles cercanos tengan una disparidad parecida e intentar evitar los problemas de las zonas de la imagen sin textura.

En OpenCV tenemos implementado el algoritmo presentado en el siguiente artículo:

`Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. IEEE Transactions on Pattern Analysis and Machine Intelligence, 30(2):328–341, 2008.`

Aunque la función de coste basada en Información Mútua del paper anterior se ha sustituido por la de Birchfield-Tomasi:

`Stan Birchfield and Carlo Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(4):401–406, 1998.`

La clase que lo implementa el algoritmo de cálculo de disparidades es `StereoSGBM`. Normalmente en un par estéreo se pretende reconstruir objetos que se encuentran más cerca, y a una distancia dada de la cámara. Esto nos dará lugar a una disparidad mínima a partir de la cual esos píxeles nos interesan (están más cerca que una distancia detertminada a la cámara). El parámetro `minDisparity` del algoritmo es crucial para lograr una estimación de disparidad adecuada. También es imporante el parámetro `numDisparities` para obtener una mayor o menor glanuralidad en las disparidades.

Ejercicio 13. Calcula las disparidades utilizando las imágenes rectificadas O1 y O2, mediante el algoritmo implementado en StereoSGBM.

In [27]:

```
# Para las imágenes del edificio de la URJC
if usar_par_estereo_building:

    ... POR HACER ...

    S = ... POR HACER ... # Disparidades para la imagen O1
```

```
  File "<ipython-input-27-af50bbbe5ea5>", line 4
    ... POR HACER ...
        ^
SyntaxError: invalid syntax
```

In [ ]:

```
# Para las imágenes del cubo
if not usar_par_estereo_building:

    ... POR HACER ...

    S = ... POR HACER ... # Disparidades para la imagen O1
```

In [ ]:

```
plt.figure()
plt.imshow(S,cmap='jet')
plt.colorbar()
plt.show()
```

Ejercicio 14. Implementa la función `plot_correspondences(image1, image2, S, H1,H2)` que, dado un par de imágenes sin rectificar, la matriz de disparidades entre las imágenes rectificadas y las homografías que llevan de las imágenes sin rectificar a las imágenes rectificadas, pida al usuario puntos en la primera imagen y dibuje sus correspondencias en la segunda.

Nota: Hay que tener en cuenta que algunos puntos sobre zonas sin textura tendrán la disparidad incorrecta. Prueba a seleccionar puntos sobre esquinas y otros puntos muy fáciles de emparejar (en ellos la disparidad será aproximadamente correcta).

In [ ]:

```
def plot_correspondences(image1, image2, S, H1, H2):
    """
    Ask for points in the first image and plot their correspondences in
    the second image.

    Parameters
    ----------
    image1, image2 : array_like
        The images (before rectification)
    S : array_like
        The matrix of disparities.
    H1, H2 : array_like
        The homographies which rectify both images.
    """
    # Prepare the two images.
    fig = plt.gcf()
    fig.clf()
    ax1 = fig.add_subplot(1, 2, 1)
    ax1.imshow(image1)
    ax1.axis('image')
    ax2 = fig.add_subplot(1, 2, 2)
    ax2.imshow(image2)
    ax2.axis('image')
    plt.draw()

    ax1.set_xlabel("Choose points in left image (or right click to end)")
    point = plt.ginput(1, timeout=-1, show_clicks=False, mouse_pop=2, mouse_stop=3)
    while len(point) != 0:
        # point has the coordinates of the selected point in the first image.
        point = np.c_[np.array(point), 1].T
        ax1.plot(point[0,:], point[1,:], '.r')

        # POR HACER: Determine the correspondence of 'point' in the second image.

        # POR HACER: Plot the correspondence with ax2.plot.

        ax2.plot( ... POR HACER ... ,'r.')

        plt.draw()
        # Ask for a new point.
        point = plt.ginput(1, timeout=-1, show_clicks=False, mouse_pop=2, mouse_stop=

    ax1.set_xlabel('')
    plt.draw()
```

In [ ]:

```
plot_correspondences( ... POR HACER ... )
```