PRÁCTICA 4:

TRANSFORMACIÓN ISOMÉTRICA AFÍN

Autores: Pablo Tore Piñana y Adrián Pérez Peinador (Grupo U1)

Introducción/motivación

El objetivo de esta práctica es, a partir de una figura en 3 dimensiones que hemos generado, realizar una animación en la que veamos la evolución de la figura dados el desplazamiento y la rotación que tiene que seguir la misma. El desplazamiento vendrá dado por el diámetro mayor de la figura, por lo que nuestro primer paso será poner los datos de tal forma que podamos obtener este diámetro de manera cómoda. Tanto en el apartado 1 como en el apartado 2 usaremos la envoltura convexa para hallar este diámetro.

Material usado

Para nuestro código hemos utilizado las librerías de skimage (para cargar la imagen del apartado 2), numpy (pues usamos los arrays de numpy en alguna ocasión), matplotlib (para crear la animación en ambos apartados y para plotear algunas figuras). Además usamos la librería math para hacer algún cálculo para sacar el centroide y diámetro de las figuras y scipy.spatial para poder calcular la envoltura convexa.

En nuestro código definimos varias funciones, que cumplen un papel distinto en el desarrollo de la práctica. Las funciones definidas son las siguientes.

- **animate**: Función que pasamos como parámetro a FuncAnimation para que dibuje la figura en el tiempo t. En ella se realizará la rotación y el desplazamiento a la figura en cuestión. Primero desplazaremos la figura al origen (desplazando el centroide al (0,0,0)), después realizaremos nuestra rotación en torno al origen y devolveremos la figura rotada a su posición inicial. Finalmente se desplazará con el desplazamiento dado en cada apartado.
- **init**: Función que pasamos como parámetro a FuncAnimation para especificar la función inicial.
- **Transf1D**: Función que realiza la transformación que se nos pide en el enunciado. Primero multiplicamos por la matriz de rotación y después desplazamos según el desplazamiento dado.

Una vez tenemos las matrices de coordenadas X,Y,Z de nuestra figura, las metemos en un vector K de tal manera que K[i] sean las coordenadas del punto i-ésimo de nuestra figura. De esta manera podemos calcular la envoltura convexa de la misma y, a partir de aquí, sacar el diámetro máximo. Esto lo haremos a través del atributo vértices de nuestra envoltura convexa. Simplemente calcularemos la distancia máxima entre cualesquiera dos vértices de la envoltura. Esta distancia será lo que tomaremos como diámetro.

Para calcular el centroide haremos la media entre todos los valores de nuestra figura coordenada a coordenada. En este primer apartado haremos esto solo con las coordenadas

x e y, ya que la rotación que haremos deja el eje z fijo y de esta manera nos ahorramos cálculos innecesarios.

Una vez tenemos el centroide y el diámetro podemos pasar a mover nuestra figura. Lo primero que hacemos es convertir las matrices X,Y,X de coordenadas en vectores (una fila detrás de otra). De esta manera nuestras coordenadas serán los vectores x0, y0, z0. Una vez hemos hecho esto, simplemente tenemos que llamar a la función FuncAnimation, pasándole nuestra función anímate como parámetro.

El apartado 2 es muy similar. Cargaremos la imagen que se nos da y cogeremos el subsistema generado por el segundo color cuando este es menor que 240. Una vez cambiados los vectores de coordenadas con este requisito, realizamos la envoltura convexa para calcular el diámetro y centroide de la misma forma que en el apartado anterior. Después crearemos la animación cambiando la función anímate por otra parecida, ya que esta vez tenemos una imagen en dos dimensiones y no una figura tridimensional. Además, el desplazamiento que tiene que seguir esta figura es distinta de la del apartado anterior.

Resultados

Para cada apartado hemos obtenido los resultados que se pedían, aparte de unos cuantos resultados intermedios que nos han facilitado la tarea.

Apartado 1:

Centroide: [-0.25 -0.25 0.]
Diámetro: 159.65
Animación generada con éxito. (Adjunta en la entrega)

Apartado 2:

Diámetro: 357.4143254837936
Centroide: [173.69 204.53 0.48]
Animación generada con éxito. (Adjunta en la entrega)

Conclusión

Como resultado de esta práctica hemos podido aprender, aunque ya habíamos hecho algo en la práctica anterior, a generar tanto una figura como una animación en Python. Además, hemos visto de manera práctica cómo hacer una rotación de esta misma figura en el espacio. En nuestro caso, sabíamos que había varias maneras de realizar la transformación afín que se nos pedía, pero decidimos inclinarnos por llevarnos la figura al origen y realizar la rotación desde ahí ya que conocíamos la matriz de la rotación que se nos pedía (con respecto al origen). Además, ya que en el ejercicio 1 las coordenadas venían dadas como matrices, hemos aprendido que cómo estén guardados los datos es de gran importancia a la hora de realizar este tipo de tareas ya que, con una representación concreta de los datos, esta tarea se nos puede facilitar en gran medida.

Código

A continuación se adjuntan capturas del código redactado para la práctica, comentado para aclarar aspectos que se consideran menos claros.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 10 18:58:33 2020

@author: Robert
"""

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from matplotlib import animation
from skimage import io

import math

from scipy.spatial import ConvexHull

# Apartado 1
print("Apartado i)")

fig = plt.figure()
ax = plt.axes(projection = '3d')
X, Y, Z = axes3d.get_test_data(0.05)
cset = ax.contour(X, Y, Z, 16, extend3d = True, cmap = plt.cm.get_cmap('plasma'))
ax.clabel(cset, fontsize = 9, inline = 1)
plt.show()

fig, ax = plt.subplots(subplot_kw = {"projection": "3d"})
ax.plot_surface(X, Y, Z, vmin = Z.min() * 2, cmap = plt.cm.plasma)
plt.show()

# Calculamos la envoltura convexa para hallar el diámetro
K = np.array([[X[i][j], Y[i][j], Z[i][j]] for j in range(len(X[0])) for i in range(len(X))])

hull = ConvexHull(K)
lista_indices = hull.vertices

vY = [Y[i][0] for i in range(len(Y))] #para tener los distintos valores de Y en un array

centroid = np.array([sum(X[0])/len(X[0]), sum(vY)/len(vY), 0])
diameter = max([math.dist(K[i], K[j]) for i in lista_indices for j in lista_indices])

print("Centroide:", centroid)
print("Diámetro:", diameter)

nframes = 20

n = len(X) # Número de filas
m = len(X[0]) # Número de columnas

# Transformamos las matrices X, Y y Z cada una en un único vector largo
x0 = np.array([X[i][j] for j in range(m) for i in range(n)])
y0 = np.array([Y[i][j] for j in range(m) for i in range(n)])
z0 = np.array([Z[i][j] for j in range(m) for i in range(n)])
```

```python
def transf1D(x, y, z, M = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]), v = np.array([0, 0, 0])):
    xt = x * 0
    yt = x * 0
    zt = x * 0
    for i in range(len(x)):
        q = np.array([x[i], y[i], z[i]])
        xt[i], yt[i], zt[i] = np.matmul(M, q) + v
    return xt, yt, zt

def animate(t):
    M = np.array([[np.cos(math.pi * 3 * t), -np.sin(math.pi * 3 * t), 0], [np.sin(math.pi * 3 * t), np.cos(math.pi * 3 * t), 0], [0, 0, 1]])
    v = np.array([0, 0, diameter * t])

    #print(t)
    ax = plt.axes(xlim=(-50, 50), ylim = (-50, 50), zlim = (-100, 300), projection = '3d')

    (xt, yt, zt) = transf1D(x0, y0, z0, v = centroid * (-1)) # Lo mandamos al origen restándole las coordenadas del centroide
    (xt, yt, zt) = transf1D(xt, yt, zt, M = M, v = centroid) # Lo rotamos con M (rot en torno al origen) y lo devolvemos a su posicion inicial
    (xt, yt, zt) = transf1D(xt, yt, zt, v = v) # Lo desplazamos v

    # Volvemos a representar X, Y y Z como matrices para poder representarlas
    X = np.array([[xt[m * i + j] for j in range(m)] for i in range(n)])
    Y = np.array([[yt[m * i + j] for j in range(m)] for i in range(n)])
    Z = np.array([[zt[m * i + j] for j in range(m)] for i in range(n)])
    cset = ax.contour(X, Y, Z, 16, extend3d = True, cmap = plt.cm.get_cmap('plasma'))
    return ax,

def init():
    return animate(0),

fig = plt.figure(figsize = (10, 10))
ani = animation.FuncAnimation(fig, animate, frames = np.linspace(0, 1, nframes), init_func = init, interval = 20)
ani.save("animation1.gif", fps = 10)

#Apartado 2

img = io.imread('arbol.png')
io.imshow(img)
#dimensions = color.guess_spatial_dimensions(img)
#print(dimensions)
#io.show()
#io.imsave('arbol2.png',img)

#https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html
fig = plt.figure(figsize=(5,5))

p = plt.contourf(img[:,:,0],cmap = plt.cm.get_cmap('viridis'), levels=np.arange(0,240,2))
p = plt.contourf(img[:,:,0],cmap = plt.cm.get_cmap('viridis'))


plt.axis('off')
#fig.colorbar(p)

xyz = img.shape

x = np.arange(0,xyz[0],1)
y = np.arange(0,xyz[1],1)
xx,yy = np.meshgrid(x, y)
xx = np.asarray(xx).reshape(-1)
```

```python
yy = np.asarray(yy).reshape(-1)
z  = img[:,:,1]
zz = np.asarray(z).reshape(-1)


"""
Consideraremos sólo los elementos con zz < 240

Por curiosidad, comparamos el resultado con contourf y scatter!
"""
#Variables de estado coordenadas
x0 = xx[zz<240]

y0 = yy[zz<240]
z0 = zz[zz<240]/256.
#Variable de estado: color
col = plt.get_cmap("viridis")(np.array(0.1+z0))

fig = plt.figure(figsize=(5,5))
ax = fig.add_subplot(1, 2, 1)
plt.contourf(x,y,z,cmap = plt.cm.get_cmap('viridis'), levels=np.arange(0,240,2))
ax = fig.add_subplot(1, 2, 2)
plt.scatter(x0,y0,c=col,s=0.1)
plt.show()


X = np.array([x0,y0,z0]).T
hull = ConvexHull(X)
#convex_hull_plot_2d(hull)

diameter = 0

for i in hull.vertices:
    for j in hull.vertices:
        if math.dist(X[i], X[j]) > diameter:
            diameter = math.dist(X[i], X[j])

print("Diámetro: " + str(diameter))
centroide = np.array([sum(x0)/len(x0), sum(y0)/len(y0), sum(z0)/len(z0)])

print("Centroide:" + str(centroide))
```

```
def animate(t):

    #print(t)
    theta = 3*3.141592
    M = np.array([[np.cos(theta*t),-np.sin(theta*t),0],[np.sin(theta*t),np.cos(theta*t),0],[0,0,1]])
    v=np.array([diameter,diameter,0])*t

    ax = plt.axes(xlim=(0,400), ylim=(0,400), projection='3d')
    #ax.view_init(60, 30)

    XYZ = transf1D(x0, y0, z0, np.array([[1,0,0], [0,1,0], [0,0,1]]) ,v=centroide*(-1))
    XYZ = transf1D(XYZ[0], XYZ[1], XYZ[2], M=M, v = centroide)
    XYZ = transf1D(XYZ[0], XYZ[1], XYZ[2],np.array([[1,0,0], [0,1,0], [0,0,1]]), v=v)


    col = plt.get_cmap("viridis")(np.array(0.1+z0))
    ax.scatter(XYZ[0],XYZ[1],c=col,s=0.1,animated=True)
    return ax,

def init():
    return animate(0),

fig = plt.figure(figsize=(6,6))
ani = animation.FuncAnimation(fig, animate, frames=np.arange(0,1,0.05), init_func=init,
                              interval=20)
#os.chdir()
ani.save("animation2.gif", fps = 10)
#os.getcwd()
```