

Práctica 7 - Geometría Computacional

Marcos Herrero Agustín

1. Introducción

El objetivo de esta práctica es aplicar una transformación isométrica formada por una rotación y una traslación sobre varios sistemas, con distinto número de variables de estado.

2. Datos y condiciones iniciales

Para la realización de la práctica se han utilizado los siguientes datos y condiciones iniciales:

- El plano de la rotación (xy) y el ángulo de la misma ($\theta = 3\pi$).
- El vector de traslación, $(d, d, 0)$, donde d es el diámetro del sistema. Recordamos que el diámetro es la mayor distancia entre dos elementos cualesquiera del sistema.
- Las restricciones a aplicar sobre el sistema del apartado *ii*): rojo < 240 .

3. Metodología

La transformación a aplicar es la misma en todos los apartados: una rotación sobre el plano xy de ángulo $\theta = 3\pi$ en torno al centroide C del sistema simultánea con una traslación de vector $(d, d, 0)$, donde d es el diámetro del sistema.

En cada apartado, pues, el primer paso después de construir el sistema S (cuyas filas representan elementos y cuyas columnas representan variables de estado) es calcular su centroide y su diámetro. El centroide se calcula simplemente como el promedio de todos los puntos del sistema. El diámetro, en cambio, puede ser más complejo, ya que calcular la distancia entre cada par de elementos es computacionalmente demasiado costoso. Una estrategia para reducir el coste del cálculo del diámetro es hallar primero la envolvente convexa del sistema (teniendo en cuenta solo las variables de estado espaciales, por supuesto) y calcular el diámetro como la máxima distancia entre los vértices de la envolvente. En general, el número de vértices de la envolvente convexa será mucho menor que el número de elementos del sistema.

Una vez calculados centroide C y diámetro d , la transformación consiste en aplicar sobre las componentes x e y de cada elemento del sistema $P_{x,y} \mapsto P'_{x,y} := C_{x,y} + M(P_{x,y} - C) + v$, siendo

$$M = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

y $v = (d, d)$. El resto de variables de estado deben permanecer invariantes. Por razones de eficiencia, en lugar de repetir el proceso para cada punto operamos matricialmente. En concreto, si S es la matriz cuyas filas son los elementos del sistema y cuyas columnas son las variables de estado y S_{xy} denota la matriz formada únicamente por las columnas x y y , la operación que haremos será $S \mapsto S'_{xy} := C_{x,y} + M(S_{xy} - C)^T + v$ donde la suma/resta de una matriz y un vector se hace fila a fila de la matriz.

Para realizar una animación de la evolución progresiva de la transformación generamos una malla de puntos equiespaciados en el intervalo $(0, 1)$ y, para cada t en esta malla, realizamos la transformación $P \mapsto P'_{x,y} := C_{x,y} + M_t(P_{xy} - C) + v_t$, siendo

$$M_t = \begin{pmatrix} \cos(\theta t) & -\sin(\theta t) \\ \sin(\theta t) & \cos(\theta t) \end{pmatrix}$$

y $v_t = (dt, dt)$.

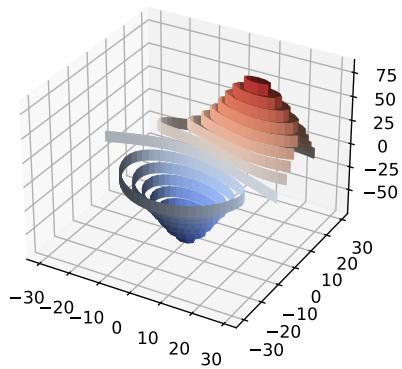


Figura 1

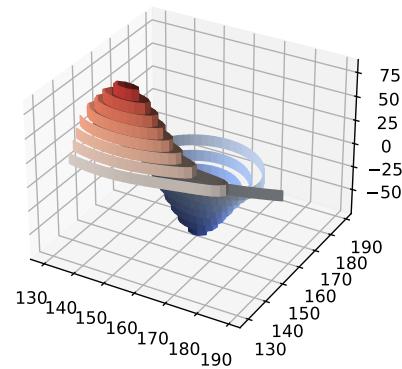


Figura 2

4. Resultados y discusión

4.1. Apartado *i*)

El sistema del apartado *i*) tiene 3 variables de estado X, Y, Z (el color que aparece en la figuras no aporta nuevas variables porque va ligado a la variable Z). En la figura 1 se muestra el sistema a transformar y en la figura 2 el mismo sistema una vez transformado.

Junto a esta memoria se adjunta un archivo “animacion.gif”, que muestra la aplicación progresiva de la transformación sobre el sistema.

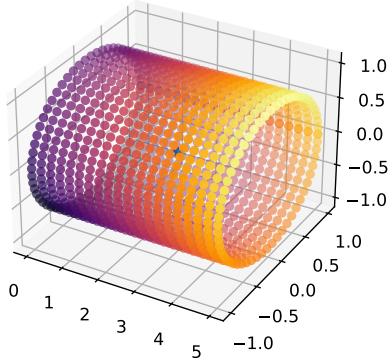


Figura 3

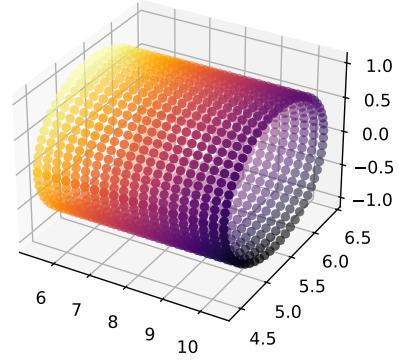


Figura 4

Además, he decidido realizar una parte adicional en este apartado, consistente en repetir el proceso sobre un sistema distinto, en el que el color sí represente dimensiones adicionales. Se ha elegido el cilindro que se muestra en la figura 3. Este sistema consta de 6 variables de estado: X, Y, Z, R, G, B . Las variables X, Y y Z son las espaciales, mientras que R, G y B codifican el color. En la figura 4 se muestra el resultado de aplicar la transformación. Además, junto a la memoria se adjunta un archivo “animacionExtra.gif” que muestra la aplicación progresiva de la transformación.

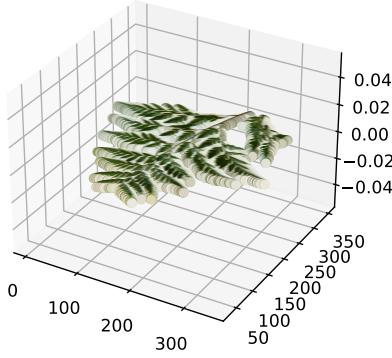


Figura 5

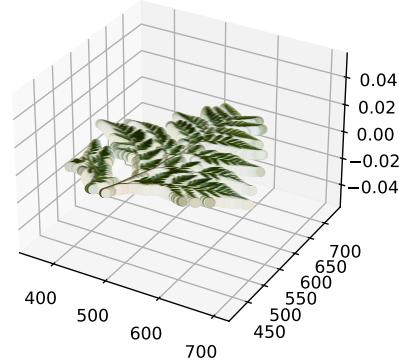


Figura 6

4.2. Apartado *ii)*

El sistema del apartado *ii*), al estar basada en una imagen bidimensional en color, tiene, originalmente, 6 variables de estado: X, Y (espaciales), R, G, B, A (color). Sin embargo, dado que nos piden aplicar la transformación en el espacio euclídeo tridimensional, hemos de añadir una tercera variable espacial Z inicializada a 0. Además, hemos de restringirnos al subsistema σ formado por los elementos con $R < 240$. El sistema resultante se muestra en la figura 5. Su centroide es el elemento cuyas variables de estado toman los valores:

$$(X, Y, Z, R, G, B, A) = (173.48, 204.16, 0, 112.11, 125.31, 88.89, 255)$$

que, si nos restringimos únicamente a las coordenadas espaciales, sería el punto $(173.48, 204.16, 0)$. Una vez aplicada la transformación, el resultado obtenido es el que se ve en la figura 6. Finalmente, la aplicación progresiva de la transformación puede verse en el archivo “animacion2.gif” adjunto.

5. Conclusiones

Esta práctica nos ha servido para entender cómo representar sistema con más de 3 variables de estado en el espacio tridimensional, haciendo uso del código RGB para codificar variables de estado en el color de los puntos representados. Asimismo, hemos observado un ejemplo de actuación de una transformación isométrica afín (en este caso, una combinación de una rotación y una traslación) del espacio euclídeo sobre estos sistemas.

Apéndice A Código utilizado

```
"""
Práctica 7 de Geometría Computacional
Autor: Marcos Herrero
"""

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

, ,
Funciones auxiliares
, ,

#Obtiene el centroide del sistema S
def calcularCentroide(S):
    return np.mean(S, axis=0)

#Obtiene el diametro del sistema 3D (hay que pasar solo las variables de estado espaciales)
def calcularDiam(Sesp):
    hull = ConvexHull(Sesp)
    diam = 0

    for i in range(len(hull.vertices)):
        for j in range(i+1, len(hull.vertices)):
            dist = np.linalg.norm(Sesp[hull.vertices[i], :]- Sesp[hull.vertices[j], :])
            if dist > diam:
                diam = dist

    return diam

#Transformación del sistema S cuyas filas representan los elementos del sistema
#y cuyas columnas representan sus variables de estado. La transformación realizada
#es  $P' = C + M(P-C) + v$ 
def transf(S,C,M,v):
    return (np.atleast_2d(C).T + np.matmul(M,(S-C).T)+ np.atleast_2d(v).T).T

#Realiza la rotación de theta grados en el plano XY (con centro en C) y traslación de vector
#Las variables X e Y han de ser las dos primeras columnas del sistema
def rotaTrasla(S,C,theta,d):
    M = np.array([[np.cos(theta), -np.sin(theta)],
                  [np.sin(theta), np.cos(theta)]])
    v = np.array([d,d])

    return np.concatenate([transf(S[:, :2], C[:, 2], M, v), S[:, 2:]], axis = 1)

#Animación utilizando contour
def fAnimationContour(t,S,theta,C,d,shape):
    St = rotaTrasla(S,C,theta*t,d*t)
    ax = plt.axes(projection='3d')
    ax.contour(St[:, 0].reshape(shape), St[:, 1].reshape(shape), St[:, 2].reshape(shape),
               16, extend3d=True, cmap = plt.cm.get_cmap('coolwarm'))

#Animación utilizando scatter
def fAnimationScatter(t,S,theta,C,d):
    St = rotaTrasla(S,C,theta*t,d*t)
    ax = plt.axes(projection='3d')
    color = list(zip(St[:, 3]/256,St[:, 4]/256,St[:, 5]/256))
    ax.scatter3D(St[:, 0], St[:, 1], St[:, 2], c = color, animated=True)
```

```

theta = 3*np.pi
"""

Apartado i): Partiendo de la figura dada, realizar una animación de una familia paramétrica
desde la identidad hasta rotación de ángulo 3pi + traslación con v = (d,d,0)
"""

print("Apartado i)")
print("===== ")

#Figura dada
X, Y, Z = axes3d.get_test_data(0.05)
shape = Z.shape

#Formamos el sistema
#El sistema tiene 3 variables de estado: las coordenadas espaciales X,Y,Z
#(el color es dependiente de Z)
S = np.column_stack([X.flatten(),Y.flatten(),Z.flatten()])
C = calcularCentroide(S) #centroide
d = calcularDiam(S) #diámetro

print("Centroide: {}".format(C))
print("Diámetro: {}".format(d))

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.contour(X, Y, Z, 16, extend3d=True,cmap = plt.cm.get_cmap('coolwarm'))
fig.savefig('fig1ini.pdf',format='pdf')
plt.show()

#Posición final
S1 = rotaTrasla(S, C, theta, d)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.contour(S1[:,0].reshape(shape),S1[:,1].reshape(shape),S1[:,2].reshape(shape),
           16,extend3d=True,cmap = plt.cm.get_cmap('coolwarm'))
fig.savefig('fig1fin.pdf',format='pdf')
plt.show()

#Animación
fig = plt.figure(figsize=(6,6))
ani = animation.FuncAnimation(fig, fAnimationContour,frames=np.arange(0,1.01,0.05),
                               fargs = (S,theta,C,d,shape), interval=20)
ani.save('animacion.gif',fps=5)
plt.show()

"""

Parte adicional: probamos la misma transformación con otra figura, en la que el color sí responde
a las dimensiones adicionales
"""

print('Parte adicional:')
print('===== ')
#Generamos un cilindro sobre el que aplicaremos la transformación

r = 1
phiini = np.linspace(0,2*np.pi,60)
xini = np.linspace(0,5,20)
phi, x = np.meshgrid(phiini,xini)

y = r*np.cos(phi)

```

```

z = r*np.sin(phi)

X = x.flatten()
Y = y.flatten()
Z = z.flatten()

rgb = plt.cm.get_cmap('inferno')(plt.Normalize()(X+Z))[:, :3]
R = rgb[:, 0]*256 # rango (0-255) para los colores
G = rgb[:, 1]*256
B = rgb[:, 2]*256

#Formamos el sistema. Tiene 6 variables de estado: las 3 espaciales y RGB
S = np.column_stack([X.flatten(), Y.flatten(), Z.flatten(), R.flatten(), G.flatten(), B.flatten()])
C = calcularCentroide(S) #centroide
d = calcularDiam(S[:, 0:3]) #diámetro (solo var espaciales)

print("Centroide: {}".format(C))
print("Diámetro: {}".format(d))

fig = plt.figure()
ax = plt.axes(projection='3d')
color = list(zip(S[:, 3]/256, S[:, 4]/256, S[:, 5]/256))
ax.scatter3D(X, Y, Z, c = color)
ax.plot3D(C[0], C[1], C[2], '*')
fig.savefig('fig2ini.pdf', format='pdf')
plt.show()

#Posición final
S1 = rotaTrasla(S, C, theta, d)

fig = plt.figure()
ax = plt.axes(projection='3d')
color = list(zip(S1[:, 3]/256, S1[:, 4]/256, S1[:, 5]/256))
ax.scatter3D(S1[:, 0], S1[:, 1], S1[:, 2], c = color)
fig.savefig('fig2fin.pdf', format='pdf')
plt.show()

#Animación

fig = plt.figure(figsize=(6,6))
ani = animation.FuncAnimation(fig, fAnimationScatter, frames=np.arange(0, 1.01, 0.05),
                             fargs = (S, theta, C, d), interval=20)
ani.save('animacionExtra.gif', fps=5)
plt.show()

, ,
Apartado ii)
, ,

print('Apartado ii')
print('=====')

#Cargamos la imagen
img = io.imread('arbol.png')

#Construimos el sistema. Tiene 7 variables de estado : las 3 coordenadas espaciales
# y 4 para construir el color en forma rgba
x = np.arange(0, img.shape[0])
y = np.arange(0, img.shape[1])
X, Y = np.meshgrid(x, y)
X = X.flatten()
Y = Y.flatten()
Z = np.zeros(X.size) #añadimos una dimensión para trabajar en el espacio tridimensional

```

```

R = img[:, :, 0].flatten()
G = img[:, :, 1].flatten()
B = img[:, :, 2].flatten()
A = img[:, :, 3].flatten()

S = np.column_stack([X, Y, Z, R, G, B, A])

#Representamos el estado inicial
fig = plt.figure()
color = list(zip(R/256, G/256, B/256, A/256))
plt.scatter(X, Y, c=color)
fig.savefig('ap2ini.pdf', format = 'pdf')
plt.show()

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter3D(S[:, 0], S[:, 1], S[:, 2], c = color)
plt.show()

#Nos quedamos con el subsistema sigma para el que el color rojo es menor que 240
sigma = S[np.where(S[:, 3] < 240)]

C = calcularCentroide(sigma) #centroide
d = calcularDiam(sigma[:, 0:2]) #diámetro

fig = plt.figure()
ax = plt.axes(projection = '3d')
color = list(zip(sigma[:, 3]/256, sigma[:, 4]/256, sigma[:, 5]/256, sigma[:, 6]/256))
ax.scatter3D(sigma[:, 0], sigma[:, 1], sigma[:, 2], c=color)
fig.savefig('sigmaini.pdf', format = 'pdf')
plt.show()

print("Centroide: {}".format(C))
print("Diámetro: {}".format(d))

#Posición final
sigma1 = rotaTrasla(sigma, C, theta, d)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter3D(sigma1[:, 0], sigma1[:, 1], sigma1[:, 2], c=color)
fig.savefig('sigmafin.pdf', format='pdf')
plt.show()

#Animación

fig = plt.figure(figsize=(6,6))
ani = animation.FuncAnimation(fig, fAnimationScatter, frames=np.arange(0, 1.01, 0.05),
                               fargs = (sigma, theta, C, d), interval=20)
ani.save('animacion2.gif', fps=5)

```

Apéndice B Resultado de la ejecución

Apartado i)

=====

Centroide: [-0.25 -0.25 -1.28110973]

Diámetro: 159.65273090569917

MovieWriter ffmpeg unavailable; using Pillow instead.

Parte adicional:

=====

Centroide: [2.5000000e+00 1.66666667e-02 -5.86989834e-17 1.69945493e+02
 7.60079671e+01 7.17529828e+01]

Diámetro: 5.38490161389728

MovieWriter ffmpeg unavailable; using Pillow instead.

Apartado ii)

=====

Centroide: [173.48190476 204.15631103 0. 112.10748644 125.31402049

88.89220012 255.]

Diámetro: 357.4143253984093

MovieWriter ffmpeg unavailable; using Pillow instead.