

Práctica 7 - GCOMP

Celia Rubio Madrigal

14 de mayo de 2022

Índice

1. Introducción	2
2. Material usado y metodología	2
2.1. Apartado <i>i</i>)	2
2.2. Apartado <i>ii</i>)	2
3. Resultados y conclusiones	2
3.1. Apartado <i>i</i>)	2
3.2. Apartado <i>ii</i>)	3
4. Código	4



1. Introducción

En esta práctica vamos a aplicar la misma transformación isométrica afín a dos sistemas distintos. Esta transformación va a ser la composición de una rotación y una translación.

Este tipo de transformaciones cumplen que mantienen invariantes la distancia entre todos los pares de puntos del sistema. Lo representaremos gráficamente para atestiguarlo.

2. Material usado y metodología

En esta práctica, consideraremos solamente la métrica euclídea.

La rotación será sobre el plano xy , y con un ángulo $\theta = 3\pi$, sobre el centroide de cada sistema.

Como buscamos una familia paramétrica en función de un parámetro temporal $t \in [0, 1]$, cada fotograma representará una rotación de ángulo $\theta \cdot t$. Es decir, aplicaremos la matriz $M(t)$ al vector $X - C(X)$, donde $C(X)$ es el centroide de X , que calculamos mediante `numpy.mean`, y:

$$M(t) = \begin{pmatrix} \cos(3\pi \cdot t) & -\sin(3\pi \cdot t) & 0 \\ \sin(3\pi \cdot t) & \cos(3\pi \cdot t) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Como el centroide queda invariante frente a la rotación, solo queda volver a sumárselo a las coordenadas que se obtienen de la transformación.

Además, aplicaremos una familia paramétrica de translaciones en función de t , cuyo vector es: $v(t) = (D(X), D(X), 0)$, con $D(X)$ el diámetro mayor del sistema. Lo calculamos aplicando el algoritmo de `ConvexHull` y comparando las distancias entre los vértices de la frontera, en vez de comparar todos los puntos entre sí directamente.

2.1. Apartado *i*)

En el primer apartado, aplicaremos la transformación afín $T(t) = M(t) \cdot X + v(t)$ a los puntos generados por la función `axes3d.get_test_data(0.05)`.

2.2. Apartado *ii*)

En el segundo apartado, aplicaremos la transformación afín $T(t)$ a una imagen digital. En concreto, tomaremos la imagen `arbol.png`, que tiene como variables de estado el plano xy y los colores `rgb`. Consideraremos el subsistema dado por los valores de la imagen cuya coordenada roja `r` sea menor que 240.

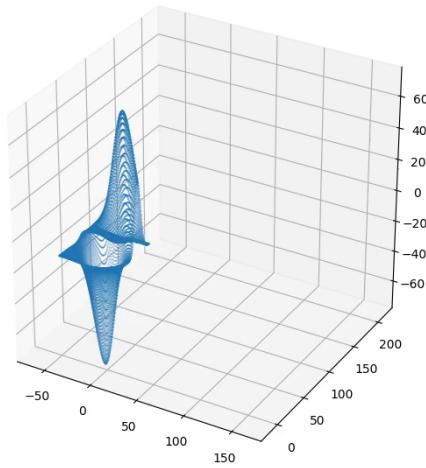
3. Resultados y conclusiones

3.1. Apartado *i*)

Para el primer sistema, su diámetro mayor es de 159.65 ± 0.01 , y su centroide se encuentra en las coordenadas $[-0.25, -0.25, -1.28] \pm 0.01$.

A continuación¹, y así como en los archivos adjuntos, se encuentra la animación de la sucesión de transformaciones paramétricas $T(t)$ sobre este sistema.

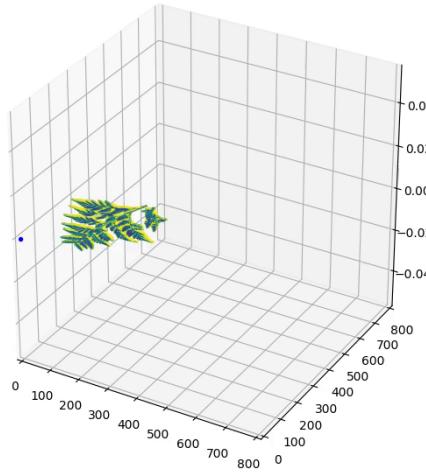
¹Para ver ambos gráficos en movimiento, se recomienda abrir este PDF en Adobe Reader u Okular.



3.2. Apartado *ii)*

Para el segundo sistema, dejaremos invariantes las coordenadas de color y solo transformaremos xy . El diámetro es de 357.41 ± 0.01 y su centroide se encuentra en $[173.48, 204.16] \pm 0.01$. Dibujaremos, además, la figura en el espacio tridimensional, poniendo su coordenada de altura a 0.

A continuación², y así como en los archivos adjuntos, se encuentra la animación de la sucesión de transformaciones paramétricas $T(t)$ sobre este sistema. Hemos coloreado de azul el punto origen, y de rojo el centroide original.



Mediante estos ejemplos, hemos calculado y averiguado cómo se transforman sistemas de tal manera que no se deformen las distancias entre sus puntos. Así, las animaciones obtenidas simulan objetos rígido moviéndose por el espacio.

²Para ver ambos gráficos en movimiento, se recomienda abrir este PDF en Adobe Reader u Okular.

4. Código

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

"""
Ejemplo para el apartado 1.

Modifica la figura 3D y/o cambia el color
https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html
"""

def get_diameter(syst):
    d = 0
    for i in range(len(syst)):
        for j in range(i + 1, len(syst)):
            d = max(d, distance.euclidean(syst[i], syst[j]))
    return d

def rotation_matrix(theta):
    return np.array([
        [np.cos(theta), -np.sin(theta), 0],
        [np.sin(theta), np.cos(theta), 0],
        [0, 0, 1],
    ])

X1, Y1, Z1 = axes3d.get_test_data(0.05)
xyz1 = np.array(list(zip(X1.flatten(), Y1.flatten(), Z1.flatten())))
hull1 = ConvexHull(xyz1)
diameter1 = get_diameter(np.array([xyz1[i] for i in hull1.vertices]))
print("Diametro i): ", diameter1)

cen1 = np.mean(xyz1, axis=0)
print("Centroide ii): ", cen1)

"""
Transformación para el primer apartado
```

```

"""
def transf1(x, y, z, M, v=np.array([0, 0, 0])):
    xt = M[0][0] * x + M[0][1] * y + M[0][2] * z + v[0]
    yt = M[1][0] * x + M[1][1] * y + M[1][2] * z + v[1]
    zt = M[2][0] * x + M[2][1] * y + M[2][2] * z + v[2]
    return xt, yt, zt

def animate1(t):
    ax = plt.axes(xlim=(-75, 175), ylim=(-25, 225), zlim=(-75, 75), projection="3d")
    XYZ1 = transf1(
        X1 - cen1[0],
        Y1 - cen1[1],
        Z1 - cen1[2],
        M=rotation_matrix(3 * np.pi * t),
        v=np.array([diameter1, diameter1, 0]) * t,
    )
    ax.scatter(
        XYZ1[0] + cen1[0], XYZ1[1] + cen1[1], XYZ1[2] + cen1[2], s=0.1, animated=True
    )
    fig.tight_layout()
    return (ax,)

def init1():
    return (animate1(0),)

fig = plt.figure(figsize=(6, 6))
fig.tight_layout()
ani = animation.FuncAnimation(
    fig, animate1, frames=np.arange(0, 1, 0.025), init_func=init1, interval=20
)
ani.save("p7a.mp4", fps=10)

"""

```

Transformación para el segundo apartado

*NOTA: Para el primer apartado es necesario adaptar la función o crear otra similar
pero teniendo en cuenta más dimensiones y hacerla MÁS EFICIENTE*

```

def transf1D(x, y, z, M, v=np.array([0, 0, 0])):
    xt = x * 0
    yt = y * 0
    zt = z * 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

```

"""

Segundo apartado casi regalado

Imagen del árbol

"""

```

img = io.imread("arbol.png")
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)
yy = np.asarray(yy).reshape(-1)
z = img[:, :, 0]
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] / 256.0
# Variable de estado: color
col = plt.get_cmap("viridis")(np.array(0.1 + z0))

```

```

xyz2 = np.array(list(zip(x0.flatten(), y0.flatten())))
hull2 = ConvexHull(xyz2)
diameter2 = get_diameter(np.array([xyz2[i] for i in hull2.vertices]))
print("Diametro ii): ", diameter2)

cen = np.mean(xyz2, axis=0)
print("Centroide ii): ", cen)

```

```

def animate2(t):
    ax = plt.axes(xlim=(0, 800), ylim=(0, 800), projection="3d")
    XYZ2 = transf1(
        x0 - cen[0],
        y0 - cen[1],
        z0,
        M=rotation_matrix(3 * np.pi * t),
        v=np.array([diameter2, diameter2, 0]) * t,
    )
    col = plt.get_cmap("viridis")(np.array(0.1 + XYZ2[2]))
    ax.scatter(XYZ2[0] + cen[0], XYZ2[1] + cen[1], c=col, s=0.1, animated=True)
    ax.scatter(cen[0], cen[1], c="r", s=8, animated=True)
    ax.scatter(0, 0, c="b", s=8, animated=True)
    fig.tight_layout()
    return (ax,)

def init2():
    return (animate2(0),)

fig = plt.figure(figsize=(6, 6))
fig.tight_layout()
ani = animation.FuncAnimation(
    fig, animate2, frames=np.arange(0, 1, 0.025), init_func=init2, interval=20
)
ani.save("p7b.mp4", fps=10)

```