Práctica 4: Transformación isométrica afín

Enrique Carro Garrido

I. Introducción (motivación/objetivo de la práctica)

En geometría diferencial, los sistemas dinámicos que generan información deben ser discretizados para que se puedan analizar computacionalmente. Además, la mayoría de procedimientos de la información requieren algún tipo de transformación continua.

En esta práctica nos centramos en el tratamiento digital de imágenes como un ejemplo de sistema que proporciona información, donde las transformaciones afines continuas juegan un papel muy importante.

En particular, se analizarán algunas transformaciones que no modifican el tamaño de la imagen, es decir, isométricas, como la rotación y la traslación.

II. Material usado (método y datos)

En ambos apartados se describe un sistema con N elementos, $S = \{a^j, (x_1^j, x_2^j, x_3^j)\}_{j=1}^N$. En el segundo apartado se trabaja realmente con un subsistema σ de S, pero podemos suponer sin pérdida de generalidad que σ es realmente S, para así poder explicar los métodos y datos usados en ambos apartados de forma conjunta.

El sistema S está contenido en el espacio afín $\mathbb{A} \equiv (\mathbb{R}^3, \mathbb{R}^3, +)$ donde se aplicará una transformación isométrica afín $f : \mathbb{A} \longrightarrow \mathbb{A}$ correspondiente a una rotación $R_\theta^{(xy)}$ aplicada en torno al centroide del sistema, y una traslación $\mathcal{T}_v$ con $v := (v_1, v_2, v_3)$ expresado en función del diámetro del sistema.

Para calcular el centroide, $C := (c_1, c_2, c_3)$, se suman todas las coordenadas y se divide el valor resultante por el número de elementos. Esto se reduce a la siguiente fórmula.

$$c_j = \frac{1}{N} \sum_{i=0}^{3} x_i, j = 1, 2, 3 \tag{1}$$

Para calcular el diámetro del sistema se tiene en cuenta el siguiente Teorema de topología,

Teorema II.1 *Dado $S \in \mathbb{R}^n$, y $C(S)$ su envoltura convexa. Entonces, $diam(S) = diam(C(S))$.*

Por lo tanto, calcular el diámetro de S es equivalente a calcular el de su envoltura convexa, que consiste simplemente en calcular la máxima distancia entre sus vértices. Por esta razón, se utiliza `ConvexHull` de la librería `scipy.spatial`.

Para el procesamiento de las transformaciones se han usado las librerías `numpy`, `math` y `scipy.spatial` para la parte de cálculos matemáticos y `matplotlib` para generar las animaciones. Para leer una imagen como un conjunto de valores RGB se ha empleado la librería `skimage`.

Los datos utilizados en la práctica son:

- "GCOM2023-Practica4_plantilla.py", de la que se obtienen ejemplos de animaciones de familias parametrizadas con respecto al tiempo, útiles para ambos apartados.
- "arbol.png", que se toma como ejemplo de sistema en el segundo apartado.

III. Resultados

En esta sección se comenta el procedimiento de obtención de los resultados, en particular de la obtención de la representación matricial de la familia paramétrica de transformaciones isométricas afines. Los resultados son los archivos ".gif"que vienen adjunto con este documento.

i) Genera una figura en 3 dimensiones (puedes utilizar la figura 1 de la plantilla) y realiza una animación de una familia paramétrica continua que reproduzca desde la identidad hasta la transformación simultánea de una rotación de $\theta = 3\pi$ y una translación con $v = (0, 0, d)$, donde d es el diámetro mayor de S.

Dado que rotaremos el sistema con respecto a su centroide, C, haremos un cambio de referencia afín de $\mathcal{R}_O := \{O; e_1, e_2, e_3\}$ a $\mathcal{R}_C := \{C; e_1, e_2, e_3\}$, con $\{e_1, e_2, e_3\}$ la base euclídea en $\mathbb{R}^3$.

Tras el cambio de referencia estamos en condiciones de representar matricialmente la aplicación $f : \mathbb{A} \longrightarrow \mathbb{A}$ como una rotación $R_{3\pi}^{(xy)}$ y una traslación, $\mathcal{T}_v$, con $v := (0, 0, d)^T$ y d el diámetro del sistema. Tomando coordenadas en $\mathcal{R}_C$, como la aplicación rota en torno al centro de la nueva referencia y simultáneamente traslada el sistema con v, la transformación resultante queda de la siguiente manera,

$$\begin{pmatrix} y_1' \\ y_2' \\ y_3' \end{pmatrix} = \begin{pmatrix} cos(3\pi) & -sen(3\pi) & 0 \\ sen(3\pi) & cos(3\pi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1' \\ x_2' \\ x_3' \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ d \end{pmatrix}$$

Por lo tanto, si parametrizamos la transformación con respecto al tiempo, obtenemos,

$$\begin{pmatrix} y_1' \\ y_2' \\ y_3' \end{pmatrix} = \begin{pmatrix} \cos(3\pi t) & -\operatorname{sen}(3\pi t) & 0 \\ \operatorname{sen}(3\pi t) & \cos(3\pi t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1' \\ x_2' \\ x_3' \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ dt \end{pmatrix}, t \in [0, 1]$$

Por último, nos queda expresar las coordenadas en la referencia afín inicial, $\mathcal{R}_O$, que solamente cambia el punto origen de $\mathcal{R}_C$. Por lo tanto, dado un punto $p \in \mathcal{A}$, cuyas coordenadas en $\mathcal{R}_O$ son $(x_1, x_2, x_3)^T$, la transformación afín parametrizada por $t \in [0, 1]$ sería,

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} \cos(3\pi t) & -\operatorname{sen}(3\pi t) & 0 \\ \operatorname{sen}(3\pi t) & \cos(3\pi t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \left(\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} - C \right) + \begin{pmatrix} 0 \\ 0 \\ dt \end{pmatrix} + C$$

Siendo $(y_1, y_2, y_3)^T$ las coordenadas de $f_t(p)$ en $\mathcal{R}_O$. El resultado de la animación está en el archivo "p4i.gif".

ii) Dado el sistema representado por la imagen digital "arbol.png", considera el subsistema σ dado por el segundo color(verde) cuando verde < 240. ¿Dónde se sitúa el centroide? Realiza la misma transformación que en el apartado anterior, con $\theta = 3\pi$ y $v = (d, d, 0)^T$, donde d es el diámetro mayor de σ.

Este apartado se resuelve del mismo modo que el anterior salvo por la definición del centroide, debido a que la tercera coordenada es el color de la imagen. Quedando C definido como (1) y siguiendo los mismos pasos que en el apartado anterior, teniendo en cuenta que la traslación se hace con el vector $v := (d, d, 0)^T$, la familia paramétrica queda definida de la siguiente manera:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} \cos(3\pi t) & -\operatorname{sen}(3\pi t) & 0 \\ \operatorname{sen}(3\pi t) & \cos(3\pi t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \left(\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} - C \right) + \begin{pmatrix} dt \\ dt \\ 0 \end{pmatrix} + C, t \in [0, 1]$$

El resultado de la animación está en el archivo "p4ii.gif".

IV. Conclusión

A partir de los resultados obtenidos se pueden concluir las siguientes afirmaciones:

- Para el tratamiento digital de imágenes, una herramienta apropiada en `python` es `contour` de la librería `matplotlib`. Los contornos son curvas continuas que representan los límites de un objeto en una imagen. En el campo del procesamiento de imágenes, los contornos se utilizan para detectar y analizar la forma y estructura de los objetos en las imágenes.
- Gracias a la geometría diferencial, el procesamiento digital de imágenes es una tarea sencilla y eficiente computacionalmente. En esta práctica hemos visto que rotar y trasladar un objeto en un plano o en el espacio consiste simplemente en operaciones sobre matrices.

V. Anexo con el código utilizado

```
"""
Enrique Carro Garrido

Práctica 4: Transformación isométrica afín
```

```
"""

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from matplotlib import animation
from skimage import io
from math import cos, sin, pi
from scipy.spatial import ConvexHull

# Transformación afín que se utilizará en ambos apartados. Sólo se necesita
# definir el centroide, el diámetro y el vector de la traslación
def f(t, v, c, X):
    # Cambiamos la referencia afín para que C sea el centro de la referencia
    X_ = X - c
    # Definimos la rotación
    R = np.array([[cos(3*pi*t), -sin(3*pi*t), 0],
                  [sin(3*pi*t),  cos(3*pi*t), 0],
                  [0,            0,           1]])
    # Aplicamos la transformación afín siendo C el centro de la referencia
    Y_ = np.matmul(X_, R.T) + v*t
    # Restauramos la referencia afín a la original
    Y = Y_ + c
    # Devolvemos el resultado
    return Y

# Cálculo del centroide y del diámetro del sistema que se utilizará en ambos
# apartados y para evitar que se repita el código.
def datos(X):
    # Definimos el centroide del sistema
    c = 1/N * np.sum(X, axis=0)

    # Definimos el diámetro del sistema (= diámetro de la envolvente convexa)
    hull = ConvexHull(X)
    d = 0
    for i in range(len(hull.vertices)):
        for j in range(i+1, len(hull.vertices)):
            p1 = X[hull.vertices[i]]
            p2 = X[hull.vertices[j]]
            distance = np.linalg.norm(p1-p2)
            if distance > d:
                d = distance
    return c, d

# i) Genera una figura en 3 dimensiones (puedes utilizar la figura 1 de la plantilla) y realiza
# una animación de una familia paramétrica continua que reproduzca desde la identidad hasta la
# transformación simultánea de una rotación de  = 3 y una translación con v = (0, 0, d), donde
# d es el diámetro mayor de S.

# Utilizamos la figura 1 de la plantilla, a partir de la cual obtenemos un vector de puntos
# en el espacio para simplificar los cálculos y definimos N, el número total de puntos
X1, X2, X3 = axes3d.get_test_data(0.05)
X = np.array([X1.reshape(-1), X2.reshape(-1), X3.reshape(-1)]).T
N = X.shape[0] # N := número de puntos = número de columnas de X.

# Recuperamos el shape de las coordenadas para reestablecerlo en la imagen, ya que
# contour sólo funciona con matrices.
shape = X1.shape

# Definimos el centroide y el vector de traslación
c, d = datos(X)
v = np.array([0,0,d])
```

```
# Creamos la figura sobre la que dibujaremos la animación
fig = plt.figure(figsize=(6,6))
ax = plt.axes(projection='3d')

# Creamos la animación
def animate(t):
    # Obtenemos la imagen de la transformación afín en el instante t
    Y = f(t, v, c, X)
    # Para meter los datos en Axes.contour se necesita que sean 2D
    Y1 = np.reshape(Y[:,0], shape)
    Y2 = np.reshape(Y[:,1], shape)
    Y3 = np.reshape(Y[:,2], shape)
    # Borramos lo que había antes en Axes
    ax.clear()
    # Definimos los límites
    ax.set_xlim(np.min(X[:,0]),np.max(f(t=1,
                                        v=v,
                                        c=c,
                                        X=X)[:,0]))
    ax.set_ylim(np.min(X[:,1]),np.max(f(t=1,
                                            v=v,
                                            c=c,
                                            X=X)[:,1]))
    ax.set_zlim(np.min(X[:,2]),np.max(f(t=1,
                                            v=v,
                                            c=c,
                                            X=X)[:,2]))
    # Ploteamos el resultado
    cset = ax.contour(Y1, Y2, Y3, 16, extend3d=True, cmap = plt.cm.get_cmap('viridis'))
    ax.clabel(cset, fontsize=9, inline=1)
    # Actualizamos el contador del tiempo
    ax.text(0.2,0.2,0.0, "t = {:,.3f}".format(t), transform=ax.transAxes, ha="right",
            bbox={'facecolor':'w', 'alpha':0.5, 'pad':5})
    return ax,

def init():
  return animate(0),

ani = animation.FuncAnimation(fig, animate, frames=np.arange(0,1.025,0.025), init_func=init,
                              interval=20)
ani.save("Practica4\\p4i.gif", fps = 10)

# ii) Dado el sistema representado por la imagen digital 'arbol.png', considera el
# subsistema  dado por el segundo color(verde) cuando verde < 240. ¿Dónde se sitúa
# el centroide? Realiza la misma transformación que en el apartado anterior, con
# =3 y v=(d,d,0), donde d es el diámetro mayor de .

# Obtenemos la imagen digital que representa el sistema
tree = io.imread('Practica4\\arbol.png')

# Obtenemos las coordenadas de la imagen
xyz = tree.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  = tree[:,:,1]
zz = np.asarray(z).reshape(-1)

# Variables de estado: coordenadas
x0 = xx[zz<240]
y0 = yy[zz<240]
```

```
z0 = zz[zz<240]/256.
X = np.array([x0, y0, z0]).T
N = X.shape[0]

# Variable de estado: color
col = plt.get_cmap("viridis")(np.array(0.1+z0))

# Obtenemos el centroide y el vector de traslación para la transformación afín
c, d = datos(X)
v = np.array([d,d,0])

# Definimos los límites del plot
fig, ax = plt.subplots(figsize=(6,6))
ax.set_xlim(np.min(X[:,0]),np.max(f(t=1,
                                    v=v,
                                    c=c,
                                    X=X)[:,0]))
ax.set_ylim(np.min(X[:,1]),np.max(f(t=1,
                                    v=v,
                                    c=c,
                                    X=X)[:,1]))

# Creamos los Artist de la animación
time = ax.text(0.1, 0.9, "",
          bbox={'facecolor':'w', 'alpha':0.5, 'pad':5},
          transform=ax.transAxes, ha="center")
scat = ax.scatter(x0, y0, c=col, s=0.1)

# Creamos la animación
def animate(t):
  # Ploteamos la imagen en el instante t
  scat.set_offsets(f(t=t,
                     v=v,
                     c=c,
                     X=X)[:,:2])
  # Actualizamos el contador del tiempo
  time.set_text("t = {:,.3f}".format(t))
  return scat, time,

ani = animation.FuncAnimation(fig, animate, frames=np.arange(0,1.025,0.025), interval=20)
ani.save("Practica4\\p4ii.gif", fps = 10)
```