1. Introducción

En esta práctica dados dos sistemas S_1 y S_2 aplicaremos a ambos una transformación correspondiente a una rotación $R_\theta^{(xy)}$ aplicada en torno al baricentro de cada sistema y una translación v.

2. Material empleado

Como la transformación es la misma en ambos apartados la estudiaremos de forma genérica. Como tenemos que recrear la transformación mediante una animación utilizaremos el parámetro $t \in [0, 1]$ para representar diferentes imágenes de la transformación. La rotación $R_\theta^{(xy)}$ tiene $\theta = 3\pi$ y es del plano xy por lo que la matriz que la representa es la siguiente:

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

con $t = 0$ es aplicar la identidad y con $t = 1$ es $R_{3\pi}^{(xy)}$. El resultado de la transformación se obtiene multiplicando M por la diferencia entre cada punto $p = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ y el punto sobre el que rotamos, en este caso el baricentro $b = \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix}$.

La translación que aplicaremos será el vector $v = \begin{pmatrix} dt \\ dt \\ 0 \end{pmatrix}$ donde d es el diámetro de cada uno de los sistemas. El resultado de aplicar esta segunda parte de la transformación se obtiene sumando v al vector obtenido previamente. Siendo la transformación la siguiente expresión matricial:

$$p' = M(p - b) + v = \begin{pmatrix} \cos 3\pi t & -\sin 3\pi t & 0 \\ \sin 3\pi t & \cos 3\pi t & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x - b_x \\ y - b_y \\ z - b_z \end{pmatrix} + \begin{pmatrix} dt \\ dt \\ 0 \end{pmatrix}$$

Para calcular el diámetro simplemente aplicamos la definición y calculamos todas las distancias entre dos puntos del sistema y nos quedamos con el máximo. Para calcular el centroide o baricentro de cada sistema, como todos los puntos ponderan igual, simplemente tenemos que calcular la media de todos los puntos.

A la hora de programar la transformación a cada una de las figuras, las únicas diferencias fueron la forma de los arrays de los puntos de la figura y a la hora de calcular el baricentro y el diámetro tener en cuenta que una figura está embebida en $\mathbb{R}^3$ y otra en $\mathbb{R}^2$.

3. Resultados

```
El centroide de la imagen, se situa en las coordenadas:( 173.482 204.156 )
```

Figura 1: Centroide de la imagen del segundo apartado

El resto de resultados corresponden a las animaciones “p7a.gif” y p7b.gif”.

4. Conclusión

Como conclusión simplemente recalcaremos que calcular el diámetro de una figura es computacionalmente muy costoso. La primera vez que ejecutamos el programa, como calculamos la raíz cuadrada de la distancia para cada par de puntos, el programa tardó más de tres horas en ejecutar el segundo apartado. Al cambiarlo y calcular únicamente la raíz cuadrada al final tarda considerablemente menos pero aún así tarda alrededor de media hora.

5. Código

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

@author: Jorge Sainero y Lucas de Torre con la plantilla de Robert
"""

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


"""
Ejemplo para el apartado 1.

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

######################      APARTADO 1       ############################
def animate3D(t):
    M = np.array([[np.cos(3*np.pi*t),-np.sin(3*np.pi*t),0],
                  [np.sin(3*np.pi*t),np.cos(3*np.pi*t),0],
                  [0,0,1]])

    v=np.array([d,d,0])*t

    ax = plt.axes(xlim=(0,200), ylim=(0,200), projection='3d')
    #ax.view_init(60, 30)

    x,y,z = transf3D(X-bx, Y-by, Z-bz, M=M, v=v)
    ax.contour(x, y, z, 16, extend3d=True,cmap = plt.cm.get_cmap('jet'))
    return ax,

def init2():
    return animate3D(0),

def transf3D(x,y,z,M, v=np.array([0,0,0])):
    xt = np.zeros(shape=(len(x),len(x)))
```

```
    yt = np.zeros(shape=(len(x),len(x)))
    zt = np.zeros(shape=(len(x),len(x)))
    for i in range(len(x)):
        for j in range(len(x)):
            q = np.array([x[i][j],y[i][j],z[i][j]])
            xt[i][j], yt[i][j], zt[i][j] = np.matmul(M, q) + v
    return xt, yt, zt


def diametro3D(x,y,z):
    d=0
    xaux=x.reshape(-1)
    yaux=y.reshape(-1)
    zaux=z.reshape(-1)
    for i in range(len(xaux)):
        for j in range(i+1,len(xaux)):
            aux=(xaux[i]-xaux[j])*(xaux[i]-xaux[j]) +
                (yaux[i]-yaux[j])*(yaux[i]-yaux[j]) +
                (zaux[i]-zaux[j])*(zaux[i]-zaux[j])
            if (aux>d):
                d=aux
    return np.sqrt(d)


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('jet'))
ax.clabel(cset, fontsize=9, inline=1)
plt.show()

d = diametro3D(X,Y,Z)
bx=np.mean(X)
by=np.mean(Y)
bz=np.mean(Z)

animate3D(np.arange(0.1, 1,0.1)[5])
plt.show()


fig = plt.figure(figsize=(6,6))
ani = animation.FuncAnimation(fig, animate3D, frames=np.arange(0,1,0.025),
                              init_func=init2, interval=20)
```

```
ani.save("p7a.gif", fps = 10)



######################      APARTADO 2       ############################

def animate2D(t):
    M = np.array([[np.cos(3*np.pi*t),-np.sin(3*np.pi*t),0],
                [np.sin(3*np.pi*t),np.cos(3*np.pi*t),0],
                [0,0,1]])

    v=np.array([d,d,0])*t

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

    XYZ = transf2D(x0-bx, y0-by, z0, M=M, v=v)
    col = plt.get_cmap("viridis")(np.array(0.1+XYZ[2]))
    ax.scatter(XYZ[0],XYZ[1],c=col,s=0.1,animated=True)
    return ax,

def init():
    return animate2D(0),

"""
Transformación para el segundo apartado

NOTA: Para el primer aparado es necesario adaptar la función o crear otra
similar pero teniendo en cuenta más dimensiones
"""

def transf2D(x,y,z,M, v=np.array([0,0,0])):
    xt = np.empty(len(x))
    yt = np.empty(len(x))
    zt = np.empty(len(x))
    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 diametro2D(x,y):
    d=0
    for i in range(len(x)):
        for j in range(i+1,len(x)):
            aux=(x[i]-x[j])*(x[i]-x[j])+(y[i]-y[j])*(y[i]-y[j])
            if (aux>d):
                d=aux
    return np.sqrt(d)

"""
Segundo apartado casi regalado

Imagen del árbol
"""


img = io.imread('arbol.png')
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))
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)
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.
#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()



d = diametro2D(x0,y0)
bx=np.mean(x0)
by=np.mean(y0)

print ("El centroide de la imagen, se situa en las coordenadas:
        (",round(bx,3),round(by,3),")")
animate2D(np.arange(0.1, 1,0.1)[5])
plt.show()


fig = plt.figure(figsize=(6,6))
ani = animation.FuncAnimation(fig, animate2D, frames=np.arange(0,1,0.025),
                              init_func=init, interval=20)

ani.save("p7b.gif", fps = 10)
```