

PRÁCTICA 4:

Transformación Isométrica Afín

Laura Cano Gómez (U2)

19 de abril de 2024

1. Introducción

El objetivo de esta práctica es transformar un sistema mediante una rotación y una traslación simultánea, visualizando el resultado en gráficas 3D.

En particular, este trabajo consiste en representar la transformación isométrica afín correspondiente a una rotación $R_\theta^{(xy)}$ y una translación $v = (v_1, v_2, \dots)$ aplicada en torno al centroide de un sistema $S = \{a^j, (x^j, y^j, \dots)\}_{j=1}^N$. En primer lugar, generamos una figura 3-dimensional cualquiera y le aplicamos una rotación $\theta = 3\pi$, desde la identidad, y una traslación $v = (0, 0, d)$, con d el diámetro mayor de la figura. En una segunda parte, trabajaremos con una imagen concreta, considerando el subsistema dado por el color azul restringido a un valor mayor o igual que 100. Situaremos el centroide y realizaremos una transformación similar a la anterior pero con $\theta = 6\pi$ y $v = (d, d, 0)$.

2. Material utilizado

Los recursos utilizados en este trabajo se han obtenido del campus virtual de la asignatura, en concreto, la imagen *hurricane-isabel.png*, que se estudia en el segundo apartado, y el fichero *GCOM2024-practica4_plantilla.py*, del que se ha extraído código e ideas para la implementación de esta práctica.

Por otro lado, se han utilizado las librerías de Python *numpy* para estructuras de datos, *matplotlib* para representar los resultados y crear el GIF, *mpl_toolkits.mplot3d* para la representación 3D, *skimage* para cargar una imagen en concreto con la que trabajar y *scipy.spatial* para calcular la componente convexa.

A continuación, se presentan las funciones incorporadas:

- **max_diameter1()** y **max_diameter2()**: obtienen el diámetro máximo del sistema dado (del apartado 1 y 2 respectivamente), es decir, la distancia máxima entre 2 puntos. Para ello, aplicamos *diameter_aux()* a cada punto de la componente convexa, obtenida con la ayuda de la librería *scipy.spatial* para hacer más eficiente el cálculo.
- **diameter_aux()**: función auxiliar para el cálculo del diámetro del sistema, que devuelve la distancia entre 2 puntos.
- **rotation()**: transforma las matrices de posición X e Y aplicando una rotación de ángulo θ , respecto al centroide de la figura. Para ello, trasladamos los puntos al origen restándoles el centroide, aplicamos la rotación θ equiespaciada, y volvemos a colocarlos sumándoles el centroide.
- **translation()**: transforma las matrices de posición X, Y y Z con una traslación v, sumando a cada componente la traslación correspondiente equiespaciada.
- **show_fig()**: representa gráficamente la figura escogida para transformar en el primer apartado.
- **get_coord()**: obtiene las coordenadas X, Y, Z de una imagen dada, con la restricción de que el color azul sea mayor o igual a 100.
- **get_centroid()**: obtiene las coordenadas x e y del centroide del sistema.
- **show_hurricane()**: representa gráficamente la imagen *hurricane-isabel.png* a través de las funciones *contourf* y *scatter* de la librería *matplotlib*, situando en cada una de ellas el centroide.
- **animate1()** y **animate2()**: generan una nueva imagen para cada frame del gif (para los apartados 1 y 2 respectivamente), aplicando la rotación y la traslación correspondiente a las posiciones X, Y, Z.

- **make_gif1()** y **make_gif2()**: crea el gif de la rotación y la traslación (para los apartados 1 y 2 respectivamente), llamando a la función *animate()* en cada frame, y gestionando la creación del *.gif* con el paquete *Animation* de la librería *Matplotlib*.
- **main()**: devuelve el resultado de los dos apartados pedidos en la práctica al ejecutar el *.py*, imprimiendo por pantalla sus resultados y guardando en la carpeta los *.gif* generados en cada apartado.

3. Resultados

3.1. Apartado i)

Hemos generado la figura tridimensional propuesta en la *plantilla*, *Figura 1*, y a continuación hemos calculado el diámetro máximo. Para ello, calculamos la componente convexa de la figura y comparamos las distancias entre cada par de puntos obtenidos. De esta forma, optimizamos el cálculo, ya que no debemos comparar cada punto, si no solo aquellos que pertenecen a la componente convexa.

Una vez obtenido este diámetro, $d = 14'14$, realizamos desde la identidad una rotación $\theta = 3\pi$ y una traslación $v = (0, 0, d)$, guardando el resultado en un nuevo archivo *gif_ej1.gif* y adjuntado en la memoria como *Anexo 1* (debe abrirse con Adobe Acrobat para poder visualizarse correctamente).

3.2. Apartado ii)

Hemos cargado la imagen *hurricane-isabel.png* y consideramos el subsistema dado por el color azul, restringiendo a un número mayor o igual que 100. Calculamos el diámetro del subsistema y obtenemos un valor $d = 522,36$. A continuación, calculamos las coordenadas del centroide, teniendo en cuenta que todos los puntos tienen el mismo peso, por tanto, este estará ubicado en la media de los puntos del subsistema, siendo las coordenadas $(291'18, 466'96)$. Para tener una idea más clara del conjunto, representamos el subsistema y su centroide con las funciones *contourf*, *Figura 2*, y *scatter*, *Figura 3*, de la librería *matplotlib*.

Finalmente, realizamos la transformación pedida, con $\theta = 6\pi$ y $v = (d, d, 0)$, y guardamos el gif obtenido como *gif_ej2.gif*, adjuntándose en esta memoria como *Anexo 1* (debe abrirse con Adobe Acrobat para poder visualizarse correctamente).

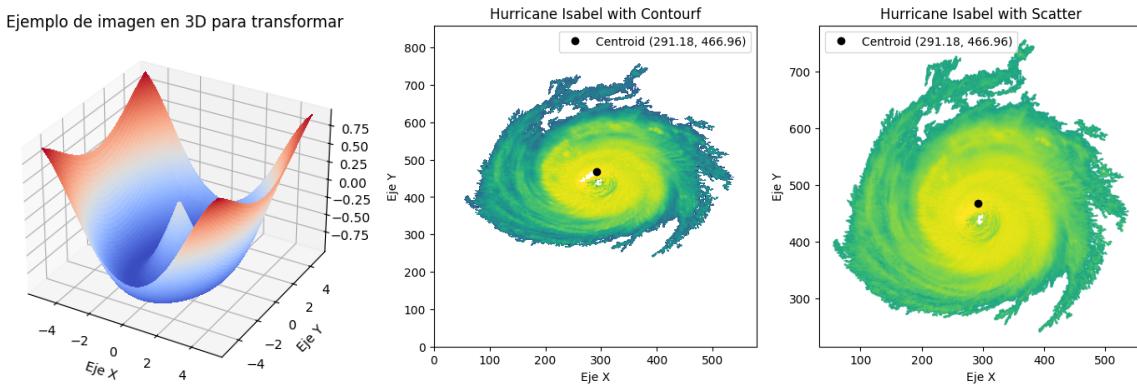


Figura 1: Imagen apartado i) Figura 2: Img. ii) con Contourf Figura 3: Img. ii) con Scatter

4. Conclusión

Calcular la dimensión de una figura de tamaño considerable puede ser muy costoso computacionalmente, sin embargo, sabemos que podemos obtener la componente convexa de la figura de forma eficiente, y que el diámetro máximo será la máxima distancia entre cada par de puntos de la componente convexa.

Por otro lado, en la *Figura 1* y *Figura 2* obtenidas en el segundo apartado, a pesar de restringir el subsistema al color azul mayor o igual que 100, obtenemos tonos de otros colores. Esto se debe a que en la representación utilizamos como *Colormap* la forma *viridis*, para ayudarnos a visualizar mejor los cambios.

Gracias a este sencillo ejemplo, hemos podido comprobar la utilidad de las funciones de transformación isométrica y el procesamiento de imágenes digitales en *Python*, que nos ofrece un amplio abanico posibilidades.

En función de los datos y el tipo de trabajo, se pueden elegir diferentes tipos de renderizado, que ayudarán visualmente a entender mejor el problema a tratar.

5. Anexo 1 - Gif obtenidos

6. Anexo 2 - Script/código utilizado

```
1 """
2 Practica 4. TRANSFORMACION ISOMETRICA AFIN
3
4 Alumna: Laura Cano Gomez (U2)
5 """
6
7
8 import numpy as np
9 import matplotlib.pyplot as plt
10 from matplotlib import animation, cm
11 from mpl_toolkits.mplot3d import axes3d
12 from skimage import io
13 from scipy.spatial import ConvexHull, convex_hull_plot_2d
14
15
16
17 #####
18 #####
```

```

19 # Enunciado
20 ######
21 '''
22 Dado un sistema con N elementos,
23 S = {aj , (xj , yj , . . . )} desde j = 1 hasta N, consideramos la
24 transformacion isometrica afin correspondiente a una rotacion R(xy) theta
25 aplicada en torno al centroide del sistema, y una translacion
26 v = (v1 , v2 , . . . ). Considera para ello la metrica euclidea.
27
28 i) Genera una figura en 3 dimensiones (puedes utilizar la figura 1 de la
29 plantilla) y realiza una animacion de una familia parametrica continua
30 que reproduzca desde la identidad hasta la transformacion simultanea de
31 una rotacion de theta = 3*pi y una translacion con v = (0 , 0 , d),
32 donde d es el diametro mayor de S. [1.0 punto]
33
34 ii) Dado el sistema representado por la imagen digital
35 'hurricane-isabel.png', considera el subsistema sigma dado por el tercer
36 color correspondiente al azul en [0 , 254] , pero restringiendo para
37 azul >= 100. ?Donde se situa el centroide? Realiza la misma transformacion
38 que en el apartado anterior, con theta = 6*pi y v = (d , d , 0) , donde d es
39 el diametro mayor de sigma. [1.5 puntos]
40
41 '''
42
43
44 #####
45 # Funciones comunes: Apartados 1 y 2.
46 #####
47
48 def diameter_aux(X , Y , Z):
49     '''
50         Calcula la distancia maxima entre 2 ptos de la figura (funcion auxiliar
51         de max_diameter()).
52
53     Returns a float object
54
55     Arguments:
56         X -> matriz que contiene las coordenadas de los puntos en el eje x
57         Y -> matriz que contiene las coordenadas de los puntos en el eje y
58         Z -> matriz que contiene los valores de la funcion en esos puntos
59     '''
60     N = len(X)
61     max_dist = 0
62
63     for i in range(N):
64         for j in range(i + 1 , N):
65
66             dist = ((X[i] - X[j]) ** 2) + ((Y[i] - Y[j]) ** 2) + ((Z[i] - Z[j]) ** 2)
67
68             if max_dist < dist:
69                 max_dist = dist
70
71     return np.sqrt(max_dist)
72
73
74 def rotation(X0 , Y0 , theta):           # theta es theta[t]
75     '''
76         Transforma las matrices de posicion X e Y con una rotacion de angulo
77         theta (equiespaciado en el intervalo en funcion de cada frame).
78
79     Returns
80         X -> new matrix position

```

```

81     Y -> new matrix position
82
83 Arguments:
84     X0      -> matrix
85     Y0      -> matrix
86     theta   -> float
87     ...
88 cx, cy = X0.mean(), Y0.mean()      # Cordenadas x e y del centroide,
89             respectivamente
90
91 X = np.dot(np.cos(theta), X0 - cx) + np.dot(-np.sin(theta), Y0 - cy) +
92         cx
93 Y = np.dot(np.sin(theta), X0 - cx) + np.dot(np.cos(theta), Y0 - cy) + cy
94
95 return X, Y
96
97
98 def translation(X0, Y0, Z0, t, v):      # t es v[t]
99     ...
100
101     Transforma las matrices de posicion X, Y y Z con una traslacion v
102     (equiespaciado en el intervalo en funcion de cada frame).
103
104 Returns
105     X -> new matrix position
106     Y -> new matrix position
107     Z -> new matrix position
108
109 Arguments:
110     X0 -> matrix
111     Y0 -> matrix
112     Z0 -> new matrix position
113     t  -> int (frame)
114     v  -> list
115     ...
116
117 at = []                      # Obtenemos la direccion de la traslacion, at =
118     axis_traslation. Ej: v = (0,0,d) -> at = [0,0,1]
119 for i in v:
120     if i == 0 :
121         at.append(0)
122     else:
123         at.append(1)
124
125
126
127 ##########
128 #APARTADO 1
129 #####
130
131 def max_diameter1(X, Y, Z):
132     ...
133     Calcula el diametro maximo entre 2 ptos de la figura con la ayuda de
134     ConvexHull para hacer optimo el calculo.
135
136 Returns a float object
137
138 Arguments:
139     X -> matriz que contiene las coordenadas de los puntos en el eje x
140     Y -> matriz que contiene las coordenadas de los puntos en el eje y

```

```

141     Z -> matriz que contiene los valores de la funcion en esos puntos
142     ...
143     x0, y0, z0 = X.reshape(-1), Y.reshape(-1), Z.reshape(-1)
144
145     H = np.array([x0,y0,z0]).T
146     hull = ConvexHull(H)
147
148     x, y, z = [], [], []
149     for i in hull.vertices:
150         x.append(x0[i])
151         y.append(y0[i])
152         z.append(z0[i])
153
154     d = diameter_aux(x, y, z)
155
156     return d
157
158
159 def show_fig(X, Y, Z):
160     """
161     Representa graficamente la figura escogida para transformar.
162
163     Returns
164         plot image
165
166     Arguments:
167         X -> matriz que contiene las coordenadas de los puntos en el eje x
168         Y -> matriz que contiene las coordenadas de los puntos en el eje y
169         Z -> matriz que contiene los valores de la funcion en esos puntos
170     """
171     fig = plt.figure(figsize=plt.figaspect(0.5))
172     ax = fig.add_subplot(1, 1, 1, projection='3d')
173
174     ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm,
175                     linewidth=0, antialiased=False)
176
177     ax.set_title('Ejemplo de imagen en 3D para transformar')
178     ax.set_xlabel('Eje X')
179     ax.set_ylabel('Eje Y')
180     plt.show()
181
182
183 def animate1(t, X0, Y0, Z0, thetas, vs, v, ax):
184     """
185     Genera la nueva imagen para cada frame del gif.
186
187     Returns
188         X -> new matrix position
189         Y -> new matrix position
190         Z -> new matrix position
191
192     Arguments:
193         t      -> int (frame)
194         X0    -> matrix
195         Y0    -> matrix
196         Z0    -> matrix
197         thetas -> array
198         vs    -> array
199         v     -> list
200         ax    -> graph axis
201
202     # Recalculamos la posicion tras rotar y trasladar
203     X, Y = rotation(X0, Y0, thetas[t])

```

```

203 X, Y, Z = translation(X, Y, Z0, vs[t], v)
204
205 # Ploteamos el frame
206 ax.clear()
207 ax.set_title("Rotacion y Traslacion ejemplo plantilla")
208 ax.set_xlabel('Eje X')
209 ax.set_ylabel('Eje Y')
210 ax.set_zlabel('Eje Z')
211 ax.set_xlim(-10, 10)
212 ax.set_ylim(-10, 10)
213 ax.set_zlim(-5, 15)
214
215 ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm,
216                  linewidth=0, antialiased=False)
217
218 def make_gif1(X, Y, Z, n_frames, thetas, vs, v):
219     """
220     Crea el gif de la rotacion y la traslacion llamando a animate() en
221     cada frame.
222
223     Returns
224         gif -> animation
225
226     Arguments:
227         X          -> matrix
228         Y          -> matrix
229         Z          -> matrix
230         n_frames   -> int
231         thetas     -> array
232         vs         -> array
233         v          -> list
234     """
235     fig = plt.figure(figsize=(6, 6))
236
237     ax = fig.add_subplot(1, 1, 1, projection='3d')
238
239     gif = animation.FuncAnimation(fig, animate1, frames= range(n_frames),
240                                   fargs=[X, Y, Z, thetas, vs, v, ax], interval= 10)
241
242     return gif
243
244
245
246 ##########
247 #APARTADO 2
248 ##########
249
250 def max_diameter2(X, Y):
251     """
252     Calcula el diametro maximo entre 2 ptos de la figura con la ayuda de
253     ConvexHull para hacer optimo el calculo.
254
255     Returns a float object
256
257     Arguments:
258         X -> matriz que contiene las coordenadas de los puntos en el eje x
259         Y -> matriz que contiene las coordenadas de los puntos en el eje y
260         Z -> matriz que contiene los valores de la funcion en esos puntos
261     """
262     x0, y0 = X.reshape(-1), Y.reshape(-1)
263

```

```

264     H = np.array([x0,y0]).T
265     hull = ConvexHull(H)
266
267     x, y = [], []
268     for i in hull.vertices:
269         x.append(x0[i])
270         y.append(y0[i])
271
272     z = np.dot(x, 0)      # Creamos un z con la misma dimension de x pero todo
273     # 0, para que no interfiera en el calculo del diametro
274
275     d = diameter_aux(x, y, z)
276
277     return d
278
279 def get_coord(img, a):
280     """
281     Obtiene las coordenadas de la imagen dada, con la restriccion de que
282     el color azul sea a >=100.
283
284     Returns
285         x -> list
286         y -> list
287         z -> list
288         x0 -> list with restriction a
289         y0 -> list with restriction a
290         z0 -> list with restriction a
291
292     Arguments:
293         img -> imagen
294         a -> int
295     """
296     xyz = img.shape           # Devuelve (580, 860, 4)
297
298     x = np.arange(0, xyz[0], 1)    # x = [0, 1, 2, ..., 579]
299     y = np.arange(0, xyz[1], 1)    # y = [0, 1, 2, ..., 859]
300
301     xx,yy = np.meshgrid(x, y)
302     xx = np.asarray(xx).reshape(-1)
303     yy = np.asarray(yy).reshape(-1)
304
305     z = img[:, :, 2]            # Seleccionamos la coordenada 2 que
306     # corresponde al color azul
307     z = np.transpose(z)
308     zz = np.asarray(z).reshape(-1)
309
310     # Restringimos el sistema a azules >= a=100
311     x0 = xx[zz >= a]
312     y0 = yy[zz >= a]
313     z0 = zz[zz >= a] / zz.max() # Hacemos / zz.max() para normalizar los
314     # datos
315
316
317     return x, y, z, x0, y0, z0
318
319 def get_centroid(X, Y):
320     """
321     Obtiene el centroide del sistema.
322
323     Returns
324         cx -> float
325         cy -> float

```

```

324
325     Arguments:
326         X -> matriz que contiene las coordenadas de los puntos en el eje x
327         Y -> matriz que contiene las coordenadas de los puntos en el eje y
328     ...
329     cx = np.mean(X)
330     cy = np.mean(Y)
331
332     return cx, cy
333
334
335 def show_hurricane(x, y, z, x0, y0, z0):
336     """
337     Representa graficamente la imagen hurricane-isabel.png a traves de
338     las funciones contourf y scatter de la libreria matplotlib, situando
339     en cada una de ellas el centroide.
340
341     Returns
342         plot 2 images
343
344     Arguments:
345         x -> list
346         y -> list
347         z -> list
348         x0 -> list with restriction a
349         y0 -> list with restriction a
350         z0 -> list with restriction a
351     ...
352     cx, cy = get_centroid(x0, y0)
353
354     # Hurricane Isabel with Contourf
355     fig = plt.figure(figsize=(5,5))
356     ax = fig.add_subplot(1, 1, 1)
357
358     plt.contourf(x, y, z, cmap= plt.cm.get_cmap('viridis'), levels=np.arange
359                   (100,255,2))
360     ax.plot([cx], [cy], 'ko', label= 'Centroid (291.18, 466.96)' )      #
361             Pintamos el centroide
362
363     ax.set_title('Hurricane Isabel with Contourf')
364     ax.set_xlabel('Eje X')
365     ax.set_ylabel('Eje Y')
366     ax.legend()
367     plt.show()
368
369     # Hurricane Isabel with Scatter
370     ax.clear()
371     fig = plt.figure(figsize=(5,5))
372     ax = fig.add_subplot(1, 1, 1)
373
374     plt.scatter(x0, y0, c= plt.get_cmap("viridis")(np.array(z0)), s= 0.1)
375     ax.plot([cx], [cy], 'ko', label= 'Centroid (291.18, 466.96)' )      #
376             Pintamos el centroide
377
378     ax.set_title('Hurricane Isabel with Scatter')
379     ax.set_xlabel('Eje X')
380     ax.set_ylabel('Eje Y')
381     ax.legend()
382     plt.show()
383
384
385 def animate2(t, X0, Y0, Z0, thetas, vs, v, ax):
386     ...

```

```

384     Genera la nueva imagen para cada frame del gif.
385
386     Returns
387         X -> new matrix position
388         Y -> new matrix position
389         Z -> new matrix position
390
391     Arguments:
392         t      -> int (frame)
393         X0    -> matrix
394         Y0    -> matrix
395         Z0    -> matrix
396         thetas -> array
397         vs     -> array
398         v      -> list
399         ax     -> graph axis
400     ''
401
402     # Recalculamos la posicion tras rotar y trasladar
403     X, Y = rotation(X0, Y0, thetas[t])
404     X, Y, Z = translation(X, Y, Z0, vs[t], v)
405
406     # Ploteamos el frame
407     ax.clear()
408     ax.set_title("Rotacion y Traslacion Hurricane Isabel")
409     ax.set_xlabel('Eje X')
410     ax.set_ylabel('Eje Y')
411     ax.set_zlabel('Eje Z')
412     ax.set_xlim(100, 1300)
413     ax.set_ylim(200, 900)
414     ax.set_zlim(-0.5, 0.5)
415
416     ax.scatter(X, Y, c= plt.get_cmap("viridis")(np.array(Z)), s= 0.1,
417                 animated=True)
418
419
420 def make_gif2(X, Y, Z, n_frames, thetas, vs, v):
421     ''
422
423     Crea el gif de la rotacion y la traslacion llamando a animate2() en
424     cada frame.
425
426     Returns
427         gif -> animation
428
429     Arguments:
430         X      -> matrix
431         Y      -> matrix
432         Z      -> matrix
433         n_frames -> int
434         thetas -> array
435         vs     -> array
436         v      -> list
437     ''
438
439     fig = plt.figure(figsize=(6,6))
440
441     ax = fig.add_subplot(1, 1, 1, projection='3d')
442
443     gif = animation.FuncAnimation(fig, animate2, frames= range(n_frames),
444                                   fargs=[X, Y, Z, thetas, vs, v, ax], interval= 10)
445
446     return gif
447
448
449

```

```

445 ##########
446 # RESULTADOS
447 #####
448 #####
449 def main():
450
451     print('APARTADO I)')
452     '''
453     Genera una figura en 3 dimensiones (puedes utilizar la figura 1 de la
454     plantilla) y realiza una animacion de una familia parametrica continua
455     que reproduzca desde la identidad hasta la transformacion simultanea de
456     una rotacion de theta = 3*pi y una translacion con v = (0, 0, d),
457     donde d es el diametro mayor de S.
458     '''
459     # Datos X, Y, Z del ejemplo de la plantilla
460     X = np.linspace(-5, 5, 100)                                # Malla de x
461     Y = np.linspace(-5, 5, 100)                                # Malla de y
462     X, Y = np.meshgrid(X, Y)                                    # Construye la malla 2
463     Z = np.sin(-np.sqrt(X**2/2 + Y**2/4))
464
465     d = max_diameter1(X, Y, Z)                                # Calculamos el diametro
466     (14.14)
467     #centroid = [X.mean(), Y.mean(), Z.mean()]      # Calculamos el centroide
468
469     show_fig(X, Y, Z)
470
471     theta = [0, 3 * np.pi]                                     # Datos
472     v = [0, 0, d]                                              # Datos
473
474     # Creamos el gif
475     n_frames = 50
476     thetas = np.linspace(theta[0], theta[1], n_frames)        # Malla de
477     theta's
478     vs = np.linspace(0, d, n_frames)                            # Malla de v's
479
480     gif = make_gif1(X, Y, Z, n_frames, thetas, vs, v)
481     gif.save("gif_ej1.gif", fps = 10)
482     plt.show()
483
484
485     print('APARTADO II)')
486     '''
487     Dado el sistema representado por la imagen digital
488     'hurricane-isabel.png', considera el subsistema sigma dado por el tercer
489     color correspondiente al azul en [0, 254], pero restringiendo para
490     azul >= 100. ?Donde se situa el centroide? Realiza la misma
491     transformacion
492     que en el apartado anterior, con theta = 6*pi y v = (d, d, 0), donde d
493     es
494     el diametro mayor de sigma.
495     '''
496     a = 100                                              # Datos
497     img = io.imread('hurricane-isabel.png')                # Datos
498
499     # Visualizacion del huracan y su centroide
500     x, y, z, x0, y0, z0 = get_coord(img, a)
501
502     cx, cy = get_centroid(x0, y0)
503     show_hurricane(x, y, z, x0, y0, z0)
504
505     print(f'\nEl centroide se situa en ({cx}, {cy}).')

```

```

503 # Creacion del gif
504 d = max_diameter2(x0, y0)                      # Calculamos el diametro
505     (522.36)
506
507 theta = [0, 6 * np.pi]                          # Datos
508 v = [d, d, 0]                                    # Datos
509
510 n_frames = 50
511 thetas = np.linspace(theta[0], theta[1], n_frames)      # Malla de
512     theta's
513 vs = np.linspace(0, d, n_frames)                  # Malla de v's
514
515 gif = make_gif2(x0, y0, z0, n_frames, thetas, vs, v)
516 gif.save("gif_ej2.gif", fps = 10)
517 plt.show()
518
519 if __name__ == '__main__':
    main()

```