1. Import Libraries

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

from mpl_toolkits.mplot3d.art3d import Poly3DCollection

- **numpy (np)** – Used for math operations, random numbers, arrays.
- **matplotlib.pyplot (plt)** – For plotting and visualization.
- **FuncAnimation** – Creates animations by repeatedly calling a function.
- **Poly3DCollection** – Draws 3D polygons (used here to draw spheres for obstacles).

---

2. Simulation Parameters

NUM_DRONES, NUM_OBSTACLES = 15, 30

SPACE_SIZE = 50

GOAL = np.array([45, 45, 20])

GOAL_RADIUS, CIRCLE_RADIUS = 3.0, 5.0

DT, MAX_SPEED = 0.1, 1.5

SENSING_RADIUS, OBSTACLE_SPEED = 8.0, 0.3

ALIGN_FRAME, ALIGN_DURATION = 100, 50

- **NUM_DRONES** – Total number of drones.
- **NUM_OBSTACLES** – Total moving obstacles.
- **SPACE_SIZE** – Size of the 3D space (0 to 50 in X/Y/Z).
- **GOAL** – Target location where drones must go.
- **GOAL_RADIUS** – Distance from goal at which drones switch to circling mode.
- **CIRCLE_RADIUS** – Radius of the circle drones form around the goal.
- **DT** – Time step for position updates.
- **MAX_SPEED** – Maximum speed for drones.
- **SENSING_RADIUS** – How far drones “see” obstacles for avoidance.
- **OBSTACLE_SPEED** – Speed of moving obstacles.
- **ALIGN_FRAME** – At which frame drones start forming a line.
- **ALIGN_DURATION** – How many frames to smoothly transition into line formation.

---

3. Initialize Drones

```
np.random.seed(42)

positions = np.array([5, 5, 0]) + (np.random.rand(NUM_DRONES, 3) - 0.5) * 4

velocities = np.zeros((NUM_DRONES, 3))

colors = plt.cm.rainbow(np.linspace(0, 1, NUM_DRONES))

trails = [[] for _ in range(NUM_DRONES)]
```

- **np.random.seed(42)** – Fixes randomness for consistent results.
- **positions** – Start near [5,5,0] with random offsets (small spread).
- **velocities** – Initially zero for all drones.
- **colors** – Assigns different rainbow colors to drones.
- **trails** – Stores previous positions (for drawing flight paths).

---

4. Initialize Obstacles

```
obstacle_positions = np.random.rand(NUM_OBSTACLES, 3) * SPACE_SIZE

obstacle_velocities = (np.random.rand(NUM_OBSTACLES, 3) - 0.5) * OBSTACLE_SPEED
```

- Random positions within the 3D space.
- Random velocities (positive or negative) scaled by OBSTACLE_SPEED.

---

5. Flags for Modes

```
circle_mode, line_mode = False, False

circle_angle, frame_count, line_progress = 0.0, 0, 0.0
```

- **circle_mode** – Whether drones are circling the goal.
- **line_mode** – Whether drones are aligning into a line.
- **circle_angle** – Used to rotate drones evenly around the circle.
- **frame_count** – Animation frame counter.
- **line_progress** – Smooth transition factor (0 to 1) for line formation.

---

6. Function to Create Spheres (Obstacles)

```
def create_sphere(center, r=1.5, seg=8):

    phi, theta = np.linspace(0, np.pi, seg), np.linspace(0, 2*np.pi, seg)

    faces = []
```

```
  for i in range(len(phi)-1):
    for j in range(len(theta)-1):
      p1=[center[0]+r*np.sin(phi[i])*np.cos(theta[j]),
        center[1]+r*np.sin(phi[i])*np.sin(theta[j]),
        center[2]+r*np.cos(phi[i])]
      p2=[center[0]+r*np.sin(phi[i+1])*np.cos(theta[j]),
        center[1]+r*np.sin(phi[i+1])*np.sin(theta[j]),
        center[2]+r*np.cos(phi[i+1])]
      p3=[center[0]+r*np.sin(phi[i+1])*np.cos(theta[j+1]),
        center[1]+r*np.sin(phi[i+1])*np.sin(theta[j+1]),
        center[2]+r*np.cos(phi[i+1])]
      p4=[center[0]+r*np.sin(phi[i])*np.cos(theta[j+1]),
        center[1]+r*np.sin(phi[i])*np.sin(theta[j+1]),
        center[2]+r*np.cos(phi[i])]
      faces.append([p1,p2,p3,p4])
  return faces
```

- Divides the sphere into small rectangular faces (latitude & longitude).
- Returns a list of faces to draw with Poly3DCollection.

---

7. Update Obstacles

```
def update_obstacles():
  global obstacle_positions, obstacle_velocities
  obstacle_positions += obstacle_velocities
  for i in range(NUM_OBSTACLES):
    for j in range(3):
      if obstacle_positions[i,j]<0 or obstacle_positions[i,j]>SPACE_SIZE:
        obstacle_velocities[i,j] *= -1
        obstacle_positions[i,j] = np.clip(obstacle_positions[i,j],0,SPACE_SIZE)
```

- Moves each obstacle.
- If it hits boundary (0 or SPACE_SIZE), reverse its velocity (bounce effect).
- np.clip ensures position stays inside bounds.

---

8. Detect Nearby Obstacles

```
def detect_local_obstacles(pos):
    dists = np.linalg.norm(obstacle_positions - pos, axis=1)
    return obstacle_positions[dists < SENSING_RADIUS]
```

- Finds obstacles closer than SENSING_RADIUS to a given position.
- Returns their positions for avoidance logic.

---

9. Reactive Obstacle Avoidance

```
def reactive_avoidance(drone_idx, direction):
    local_obs = detect_local_obstacles(positions[drone_idx])
    if local_obs.size == 0: return np.zeros(3)
    forward = direction / (np.linalg.norm(direction)+1e-6)
    directions = [[-forward[1],forward[0],0], [forward[1],-forward[0],0], [0,0,1], [0,0,-1]]
    best = np.zeros(3); min_count = float("inf")
    for d in directions:
        if len(detect_local_obstacles(positions[drone_idx]+np.array(d)*3)) < min_count:
            min_count, best = len(detect_local_obstacles(positions[drone_idx]+np.array(d)*3)), d
    return np.array(best)*2
```

- Checks four directions (left, right, up, down).
- Chooses the direction with **least obstacles nearby**.
- Returns an avoidance vector scaled by 2.

---

10. Update Drone Positions

```
def update_positions():
    global positions, frame_count, circle_mode, circle_angle, line_mode, line_progress
    update_obstacles()
```

- First updates obstacle positions.

10.1 Trigger Line Formation

```
if frame_count >= ALIGN_FRAME and not line_mode:
```

```
    line_mode = True
    start_line = positions.mean(axis=0)
    end_line = start_line + np.array([28, 28, 10])
    global line_targets
    line_targets = np.array([start_line+(end_line-start_line)*i/(NUM_DRONES-1) for i in range(NUM_DRONES)])
```

- After ALIGN_FRAME frames, drones start forming a line.
- Line endpoints: current center → offset [28,28,10].
- line_targets = evenly spaced positions along this line.

10.2 Smooth Transition to Line

```
if line_mode and line_progress < 1.0:
    line_progress = min(1.0, line_progress+1.0/ALIGN_DURATION)
    positions = positions*(1-line_progress)+line_targets*line_progress
```

- Gradually moves drones toward target line using interpolation.

10.3 Switch to Circle Mode

```
 if not circle_mode and np.linalg.norm(positions[0]-GOAL) < GOAL_RADIUS:
    circle_mode = True
```

- When **first drone reaches goal radius**, switch to circle mode.

10.4 Circle Motion

```
if circle_mode:
    circle_angle += 0.05
    for i in range(NUM_DRONES):
        angle = circle_angle + 2*np.pi*i/NUM_DRONES
        positions[i] = [GOAL[0]+CIRCLE_RADIUS*np.cos(angle),
                        GOAL[1]+CIRCLE_RADIUS*np.sin(angle), GOAL[2]]
        trails[i].append(positions[i].copy())
        if len(trails[i]) > 50: trails[i].pop(0)
```

- Drones evenly spaced in circle, rotating by circle_angle.

- Trails store last 50 positions (for drawing paths).

10.5 Normal Motion to Goal

```
else:
    for i in range(NUM_DRONES):
        direction = GOAL - positions[i]
        vel = direction/(np.linalg.norm(direction)+1e-6)*MAX_SPEED
        vel += reactive_avoidance(i, direction)
        if np.linalg.norm(vel) > MAX_SPEED: vel = vel/np.linalg.norm(vel)*MAX_SPEED
        positions[i] += vel*DT
        trails[i].append(positions[i].copy())
        if len(trails[i]) > 50: trails[i].pop(0)
```

- Moves drones toward goal with **MAX_SPEED**.
- Adds avoidance vector if obstacles detected.

10.6 Increment Frame Count

```
frame_count += 1
```

---

11. Visualization Setup

```
def run_simulation():
    fig = plt.figure(figsize=(10,10))
    ax = fig.add_subplot(111, projection='3d')
    ax.set(xlim=(0,SPACE_SIZE), ylim=(0,SPACE_SIZE), zlim=(0,SPACE_SIZE/2),
           xlabel="X", ylabel="Y", zlabel="Z")
    ax.scatter(*GOAL, c='green', s=200, marker='*')  # Goal point
```

- Creates 3D plot, sets axes limits and labels.
- Draws the goal as a green star.

11.1 Draw Drones and Trails

```
drone_scatter = ax.scatter(positions[:,0], positions[:,1], positions[:,2], c=colors, s=100)
    trail_lines = [ax.plot([],[],[], lw=2, c=colors[i])[0] for i in range(NUM_DRONES)]
```

labels = [ax.text(*positions[i], f"D{i+1}", color=colors[i], fontsize=9) for i in range(NUM_DRONES)]

- **drone_scatter** – shows drones.
- **trail_lines** – line objects for each drone.
- **labels** – text labels above each drone.

11.2 Draw Obstacles

```
obstacle_polys = []
for pos in obstacle_positions:
    poly = Poly3DCollection(create_sphere(pos), facecolors='blue', edgecolors='k', alpha=0.6)
    ax.add_collection3d(poly); obstacle_polys.append(poly)
```

- Creates blue spheres for each obstacle.

---

12. Animation Function

```
def animate(frame):
    update_positions()
```

- Called every frame by FuncAnimation.

12.1 Update Drone Positions

```
drone_scatter._offsets3d = (positions[:,0], positions[:,1], positions[:,2])
```

12.2 Update Trails

```
for i, line in enumerate(trail_lines):
        arr = np.array(trails[i])
        line.set_data(arr[:,0], arr[:,1]) if len(arr)>1 else line.set_data([],[])
        line.set_3d_properties(arr[:,2]) if len(arr)>1 else line.set_3d_properties([])
```

12.3 Update Labels

```
for i, lbl in enumerate(labels):
        lbl.set_position((positions[i,0], positions[i,1])); lbl.set_3d_properties(positions[i,2]+1)
```

12.4 Update Obstacles

for i, poly in enumerate(obstacle_polys):

poly.remove()

new_poly = Poly3DCollection(create_sphere(obstacle_positions[i]), facecolors='blue', edgecolors='k', alpha=0.6)

ax.add_collection3d(new_poly); obstacle_polys[i] = new_poly

12.5 Rotate Camera

ax.view_init(30, (frame*0.5)%360)

return [drone_scatter]+trail_lines+labels+obstacle_polys

---

13. Create Animation

ani = FuncAnimation(fig, animate, frames=800, interval=50, blit=False)

plt.show()

- **ani** must be assigned to avoid warning (keeps it alive until plt.show()).

---

14. Run Simulation

if __name__ == "__main__":

run_simulation()

- Ensures code runs only when executed directly (not imported).
