NOGOZONE TEST SIMULATION

import matplotlib.pyplot as plt

import numpy as np

from scipy.interpolate import CubicSpline

import math

from matplotlib.patches import Rectangle

# ——— Helpers —————————————————————————————————————————————————————

def generate\_rows(num\_rows=3, row\_length=10, spacing=2):

return [list(zip([i\*spacing]\*5, np.linspace(0, row\_length, 5)))

for i in range(num\_rows)]

def interpolate\_row(pts, res=0.2):

x, y = zip(\*pts)

t = np.linspace(0,1,len(x))

csx, csy = CubicSpline(t,x), CubicSpline(t,y)

t2 = np.linspace(0,1,int(1/res\*len(x)))

return list(zip(csx(t2), csy(t2)))

def dist(a,b): return math.hypot(b[0]-a[0], b[1]-a[1])

def head(a,b): return math.degrees(math.atan2(b[1]-a[1], b[0]-a[0]))%360

def diff\_h(c,t): return (t-c+540)%360 -180

def in\_no\_go(pos, b):

x0,y0,x1,y1 = b

return not (x0<=pos[0]<=x1 and y0<=pos[1]<=y1)

def nav\_cmd(pos, hdg, path, idx, b, thr=0.5):

if idx>=len(path): return "End", idx

target = path[min(idx,len(path)-1)]

nxt = [pos[0]+0.2\*math.cos(math.radians(hdg)),

pos[1]+0.2\*math.sin(math.radians(hdg))]

if in\_no\_go(nxt,b): return "No-go", idx

d = dist(pos,target)

if idx>=len(path)-1 and d<thr: return "End", len(path)

if d<thr: return "Advance", idx+1

desired = head(pos,target)

dh = diff\_h(hdg,desired)

if abs(dh)<5: return "Straight", idx

return ("Right", idx) if dh<0 else ("Left", idx)

# ——— Drawing —————————————————————————————————————————————————————

def draw(ax, bounds, paths, pos, hdg, phase,

current\_row=None, waypoints=None, transition=None,

violation=False, viol\_msg=""):

ax.clear()

# field boundary

ax.add\_patch(Rectangle((bounds[0],bounds[1]),

bounds[2]-bounds[0], bounds[3]-bounds[1],

linestyle='--', edgecolor='red', facecolor='none'))

# all rows

for i,p in enumerate(paths):

xs,ys = zip(\*p)

style,alpha = ('b-',0.7) if i==current\_row else ('k--',0.3)

ax.plot(xs,ys,style,alpha=alpha)

# waypoints

if waypoints:

wx,wy = zip(\*waypoints)

ax.scatter(wx,wy,c='blue',s=20,alpha=0.6)

# transition

if transition:

tx,ty = zip(\*transition)

ax.plot(tx,ty,'b:',alpha=0.6)

# rover

ax.plot(pos[0],pos[1],'ro')

ax.quiver(pos[0],pos[1],

math.cos(math.radians(hdg)),

math.sin(math.radians(hdg)),

scale=10, color='red')

# violation marker

if violation:

ax.plot(pos[0],pos[1],'rx',markersize=15,markeredgewidth=3)

ax.set\_title(viol\_msg, color='red', fontsize=14)

else:

titles = {

1: "Phase 1: Traverse Row 1",

2: "Phase 2: Spin & Row 1 Revisit Violation",

3: "Phase 3: Transition → Row 2",

4: "Phase 4: Traverse Row 2",

5: "Phase 5: Exit Violation from Row 2",

6: "Phase 6: Direct → Row 3",

7: "Phase 7: Traverse Row 3"

}

ax.set\_title(titles[phase])

ax.set\_xlim(-2,8); ax.set\_ylim(-2,12)

ax.set\_aspect('equal'); ax.grid(True)

plt.draw(); plt.pause(0.05)

# ——— Main Simulation —————————————————————————————————————————————————

def simulate\_all():

rows = generate\_rows()

paths = [interpolate\_row(r) for r in rows]

bounds = (-1,-1,7,11)

final\_wp3 = paths[2][-1]

fig, ax = plt.subplots(figsize=(8,6))

plt.ion()

pos = list(paths[0][0]); hdg=90; idx=0; phase=1

orient = {}

# buffers for transitions

transition = []; ti=0

while True:

# Phase 1: Traverse Row 1

if phase==1:

draw(ax,bounds,paths,pos,hdg,1,current\_row=0,waypoints=paths[0])

cmd,idx = nav\_cmd(pos,hdg,paths[0],idx,bounds)

action = ""

if cmd=="Straight":

pos[0]+=0.2\*math.cos(math.radians(hdg))

pos[1]+=0.2\*math.sin(math.radians(hdg))

action="Drive straight"

elif cmd=="Left":

hdg+=3; action="Align left"

elif cmd=="Right":

hdg-=3; action="Align right"

elif cmd=="Advance":

idx+=1; action="Advance to next waypoint"

elif cmd=="End":

pos=list(paths[0][-1]); action="End of Row 1 (snap to last WP)"

phase=2

dist\_goal=dist(pos,final\_wp3)

print(f"[P1] {action}, Pos=({pos[0]:.2f},{pos[1]:.2f}), Dist→final3={dist\_goal:.2f}")

continue

# Phase 2: Spin & revisit violation

if phase==2:

start=hdg

for i in range(18):

hdg=(start+10\*(i+1))%360

draw(ax,bounds,paths,pos,hdg,2,current\_row=0,waypoints=paths[0])

print(f"[P2] Spin 180°, Pos=({pos[0]:.2f},{pos[1]:.2f}), Dist→final3={dist(pos,final\_wp3):.2f}")

draw(ax,bounds,paths,pos,hdg,2,current\_row=0,waypoints=paths[0],

violation=True,viol\_msg="🚫 No-Go: Cannot Revisit Row 1")

print(f"[P2] Violation on revisit at Pos=({pos[0]:.2f},{pos[1]:.2f})")

plt.pause(1)

# prepare Row 2

phase=3

ep0,ep1 = rows[1][0], rows[1][-1]

d0,d1 = dist(pos,ep0), dist(pos,ep1)

choice=0 if d0<d1 else -1

orient[2] = 'normal' if choice==0 else 'rev'

next\_wp = rows[1][choice]

turn\_pt = (next\_wp[0], pos[1])

transition=[turn\_pt, next\_wp]; ti=0

continue

# Phase 3: Transition → Row 2

if phase==3:

tp = transition[ti]

draw(ax,bounds,paths,pos,hdg,3,transition=[pos,tp])

des=head(pos,tp); df=diff\_h(hdg,des)

if abs(df)>5:

hdg += 3 if df>0 else -3

action = f"Align {'left' if df>0 else 'right'}"

else:

pos[0]+=0.2\*math.cos(math.radians(hdg))

pos[1]+=0.2\*math.sin(math.radians(hdg))

action="Drive toward transition"

dist\_goal=dist(pos,final\_wp3)

print(f"[P3] {action}, Pos=({pos[0]:.2f},{pos[1]:.2f}), Dist→final3={dist\_goal:.2f}")

if dist(pos,tp)<0.5:

ti+=1

if ti>=len(transition):

pos=list(tp)

print(f"[P3] Snap to Row 2 start {pos}")

phase=4; idx=0

continue

# Phase 4: Traverse Row 2

if phase==4:

path2 = paths[1] if orient[2]=='normal' else paths[1][::-1]

draw(ax,bounds,paths,pos,hdg,4,current\_row=1,waypoints=path2)

cmd,idx = nav\_cmd(pos,hdg,path2,idx,bounds)

if cmd=="Straight":

pos[0]+=0.2\*math.cos(math.radians(hdg)); pos[1]+=0.2\*math.sin(math.radians(hdg))

action="Drive straight"

elif cmd=="Left":

hdg+=3; action="Align left"

elif cmd=="Right":

hdg-=3; action="Align right"

elif cmd=="Advance":

idx+=1; action="Advance to next WP"

elif cmd=="End":

pos=list(path2[-1]); action="End of Row 2 (snap)"

phase=5

dist\_goal=dist(pos,final\_wp3)

print(f"[P4] {action}, Pos=({pos[0]:.2f},{pos[1]:.2f}), Dist→final3={dist\_goal:.2f}")

continue

# Phase 5: Exit violation

if phase==5:

draw(ax,bounds,paths,pos,hdg,5,current\_row=1)

pos[0]+=0.05\*math.cos(math.radians(hdg)); pos[1]+=0.05\*math.sin(math.radians(hdg))

plt.pause(0.2)

action="Drive slow exit"

dist\_goal=dist(pos,final\_wp3)

print(f"[P5] {action}, Pos=({pos[0]:.2f},{pos[1]:.2f}), Dist→final3={dist\_goal:.2f}")

if in\_no\_go(pos,bounds):

draw(ax,bounds,paths,pos,hdg,5,current\_row=1,

violation=True,viol\_msg="🚫 Out of Field! Aborting")

print(f"[P5] Exit violation at Pos=({pos[0]:.2f},{pos[1]:.2f})")

plt.pause(1)

# prep Row 3 direct

phase=6

ep0,ep1 = rows[2][0], rows[2][-1]

d0,d1 = dist(pos,ep0), dist(pos,ep1)

choice=0 if d0<d1 else -1

orient[3] = 'normal' if choice==0 else 'rev'

next\_wp = rows[2][choice]

continue

# Phase 6: Direct → Row 3

if phase==6:

desired = head(pos,next\_wp); dh=diff\_h(hdg,desired)

if abs(dh)>5:

hdg += 3 if dh>0 else -3

action = f"Align {'left' if dh>0 else 'right'} to Row 3"

else:

pos[0]+=0.2\*math.cos(math.radians(hdg))

pos[1]+=0.2\*math.sin(math.radians(hdg))

action="Drive straight to Row 3 start"

draw(ax,bounds,paths,pos,hdg,6)

dist\_goal=dist(pos,final\_wp3)

print(f"[P6] {action}, Pos=({pos[0]:.2f},{pos[1]:.2f}), Dist→final3={dist\_goal:.2f}")

if dist(pos,next\_wp)<0.5:

pos=list(next\_wp)

print(f"[P6] Arrived at Row 3 start {pos}")

phase=7; idx=0

continue

# Phase 7: Traverse Row 3

if phase==7:

path3 = paths[2] if orient[3]=='normal' else paths[2][::-1]

draw(ax,bounds,paths,pos,hdg,7,current\_row=2,waypoints=path3)

cmd,idx = nav\_cmd(pos,hdg,path3,idx,bounds)

if cmd=="Straight":

pos[0]+=0.2\*math.cos(math.radians(hdg)); pos[1]+=0.2\*math.sin(math.radians(hdg))

action="Drive straight"

elif cmd=="Left":

hdg+=3; action="Align left"

elif cmd=="Right":

hdg-=3; action="Align right"

elif cmd=="Advance":

idx+=1; action="Advance to next WP"

elif cmd=="End":

pos=list(path3[-1]); action="End of Row 3 (snap)"

draw(ax,bounds,paths,pos,hdg,7,current\_row=2,waypoints=path3)

print(f"[P7] {action}, Pos=({pos[0]:.2f},{pos[1]:.2f}), Dist→final3=0.00")

print("✅ Row 3 done. Simulation complete!")

break

dist\_goal=dist(pos,final\_wp3)

print(f"[P7] {action}, Pos=({pos[0]:.2f},{pos[1]:.2f}), Dist→final3={dist\_goal:.2f}")

continue

plt.ioff()

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

simulate\_all()