|  |
| --- |
| from controller import Robot  import numpy as np  import random  import math  import pickle  import os  # Constants  TIME\_STEP = 64  EPSILON = 0.1  ALPHA = 0.1  GAMMA = 0.9  # Robot setup  robot = Robot()  ps = []  ps\_names = ['ps0', 'ps1', 'ps2', 'ps3', 'ps4', 'ps5', 'ps6', 'ps7']  for name in ps\_names:  sensor = robot.getDevice(name)  sensor.enable(TIME\_STEP)  ps.append(sensor)  left\_motor = robot.getDevice('left wheel motor')  right\_motor = robot.getDevice('right wheel motor')  left\_motor.setPosition(float('inf'))  right\_motor.setPosition(float('inf'))  left\_motor.setVelocity(0.0)  right\_motor.setVelocity(0.0)  # Actions: [Left Speed, Right Speed]  ACTIONS = {  0: [6.28, 6.28], # Forward  1: [3.0, 6.28], # Slight Left  2: [6.28, 3.0], # Slight Right  3: [-6.28, 6.28], # Spin left  4: [6.28, -6.28], # Spin right  }  # Q-table  Q = {}  reward\_log = []  # Discretize proximity sensor values  def get\_state():  return tuple(1 if ps[i].getValue() > 80.0 else 0 for i in range(8))  # Choose action via ε-greedy  def choose\_action(state):  if state not in Q:  Q[state] = [0.0] \* len(ACTIONS)  if random.random() < EPSILON:  return random.randint(0, len(ACTIONS) - 1)  return int(np.argmax(Q[state]))  # Reward function  def get\_reward(state):  if state[0] or state[7]: # front collision  return -10  elif sum(state) > 0: # side sensors detect  return -1  return 2 # clean path  # Optional: Save Q-table  def save\_q\_table():  with open("q\_table.pkl", "wb") as f:  pickle.dump(Q, f)  # Optional: Save rewards  def save\_rewards():  with open("reward\_log.txt", "w") as f:  for r in reward\_log:  f.write(f"{r}\n")  # Main Loop  step\_count = 0  total\_reward = 0  while robot.step(TIME\_STEP) != -1:  step\_count += 1  state = get\_state()  action = choose\_action(state)  # Execute action  left\_speed, right\_speed = ACTIONS[action]  left\_motor.setVelocity(left\_speed)  right\_motor.setVelocity(right\_speed)  # Small step to simulate movement  robot.step(TIME\_STEP)  new\_state = get\_state()  reward = get\_reward(new\_state)  total\_reward += reward  reward\_log.append(reward)  if new\_state not in Q:  Q[new\_state] = [0.0] \* len(ACTIONS)  # Q-learning update  Q[state][action] += ALPHA \* (reward + GAMMA \* max(Q[new\_state]) - Q[state][action])  # Save every 500 steps  if step\_count % 500 == 0:  print(f"Step: {step\_count} | Avg Reward (last 100): {np.mean(reward\_log[-100:]):.2f}")  save\_q\_table()  save\_rewards() |

A bar code graph with numbers

AI-generated content may be incorrect.

A graph with blue lines

AI-generated content may be incorrect.