import numpy as np

import heapq

import torch

from PIL import Image

import RPi.GPIO as GPIO

from time import sleep

from cryptography.fernet import Fernet

def heuristic(a, b):

return np.linalg.norm(np.array(a) - np.array(b))

def astar(grid, start, goal):

neighbors = [(0,1),(1,0),(-1,0),(0,-1)]

open\_set = []

heapq.heappush(open\_set, (0, start))

came\_from = {}

g\_score = {start: 0}

f\_score = {start: heuristic(start, goal)}

while open\_set:

current = heapq.heappop(open\_set)[1]

if current == goal:

path = []

while current in came\_from:

path.append(current)

current = came\_from[current]

path.append(start)

return path[::-1]

for dx, dy in neighbors:

neighbor = (current[0]+dx, current[1]+dy)

if (0 <= neighbor[0] < grid.shape[0] and 0 <= neighbor[1] < grid.shape[1] and grid[neighbor] == 0):

tentative\_g\_score = g\_score[current] + 1

if neighbor not in g\_score or tentative\_g\_score < g\_score[neighbor]:

came\_from[neighbor] = current

g\_score[neighbor] = tentative\_g\_score

f\_score[neighbor] = tentative\_g\_score + heuristic(neighbor, goal)

heapq.heappush(open\_set, (f\_score[neighbor], neighbor))  
import numpy as np

import heapq

import torch

from PIL import Image

import RPi.GPIO as GPIO

from time import sleep

from cryptography.fernet import Fernet

def heuristic(a, b):

return np.linalg.norm(np.array(a) - np.array(b))

def astar(grid, start, goal):

neighbors = [(0,1),(1,0),(-1,0),(0,-1)]

open\_set = []

heapq.heappush(open\_set, (0, start))

came\_from = {}

g\_score = {start: 0}

f\_score = {start: heuristic(start, goal)}

while open\_set:

current = heapq.heappop(open\_set)[1]

if current == goal:

path = []

while current in came\_from:

path.append(current)

current = came\_from[current]

path.append(start)

return path[::-1]

for dx, dy in neighbors:

neighbor = (current[0]+dx, current[1]+dy)

if (0 <= neighbor[0] < grid.shape[0] and 0 <= neighbor[1] < grid.shape[1] and grid[neighbor] == 0):

tentative\_g\_score = g\_score[current] + 1

if neighbor not in g\_score or tentative\_g\_score < g\_score[neighbor]:

came\_from[neighbor] = current

g\_score[neighbor] = tentative\_g\_score

f\_score[neighbor] = tentative\_g\_score + heuristic(neighbor, goal)

heapq.heappush(open\_set, (f\_score[neighbor], neighbor))  
pwm.ChangeDutyCycle(0)

pwm.stop()

GPIO.cleanup()

def secure\_data\_transmission(data):

key = Fernet.generate\_key()

cipher = Fernet(key)

encrypted = cipher.encrypt(data.encode())

decrypted = cipher.decrypt(encrypted).decode()

print(f"Original: {data}")

print(f"Encrypted: {encrypted}")

print(f"Decrypted: {decrypted}")

def main():

# 8. Path Planning

grid = np.zeros((10, 10))

grid[5, 2] = 1 # Obstacle

start = (0, 0)

goal = (9, 9)

path = astar(grid, start, goal)

print(f"Path from {start} to {goal}:\n{path}")

detect\_objects("test.jpg")

print("Moving robotic arm to 90 degrees...")

move\_servo(90)

secure\_data\_transmission("Robot location: (24.45, 85.67)")