#python code for object detection

import rospy

import actionlib

from geometry\_msgs.msg import PoseStamped

from move\_base\_msgs.msg import MoveBaseAction, MoveBaseGoal

from sensor\_msgs.msg import Image, LaserScan

from cv\_bridge import CvBridge

import cv2

import numpy as np

from queue import PriorityQueue

class Node:

def \_\_init\_\_(self, position, parent=None):

self.position = position

self.parent = parent

self.g = 0 # Cost from start to current node

self.h = 0 # Heuristic cost (estimated cost from current node to goal)

self.f = 0 # Total cost (g + h)

def \_\_lt\_\_(self, other):

return self.f < other.f

class PathPlanner:

def \_\_init\_\_(self):

rospy.init\_node('path\_planner')

self.move\_base\_client = actionlib.SimpleActionClient('move\_base', MoveBaseAction)

self.move\_base\_client.wait\_for\_server()

self.map\_resolution = 0.05 # Resolution of the occupancy grid map

self.map\_origin\_x = 0.0 # X-coordinate of the map origin

self.map\_origin\_y = 0.0 # Y-coordinate of the map origin

self.map = None

self.bridge = CvBridge()

self.object\_cascade = cv2.CascadeClassifier('object\_cascade.xml') # Path to object detection cascade file

self.object\_detected = False

def get\_map(self):

rospy.wait\_for\_service('static\_map')

try:

get\_map = rospy.ServiceProxy('static\_map', GetMap)

response = get\_map()

self.map = response.map

except rospy.ServiceException as e:

rospy.logerr("Service call failed: %s", e)

def occupancy\_grid\_to\_image(self):

width = self.map.info.width

height = self.map.info.height

data = np.array(self.map.data).reshape((height, width))

img = np.uint8(data)

img = cv2.cvtColor(img, cv2.COLOR\_GRAY2BGR)

img = cv2.transpose(img)

img = cv2.flip(img, 0)

return img

def heuristic\_cost(self, position, goal):

return np.linalg.norm(np.array(position) - np.array(goal))

def get\_neighbors(self, position):

neighbors = []

rows, cols = self.map.shape

r, c = position

if r > 0 and self.map[r - 1, c] == 0: # Up

neighbors.append((r - 1, c))

if r < rows - 1 and self.map[r + 1, c] == 0: # Down

neighbors.append((r + 1, c))

if c > 0 and self.map[r, c - 1] == 0: # Left

neighbors.append((r, c - 1))

if c < cols - 1 and self.map[r, c + 1] == 0: # Right

neighbors.append((r, c + 1))

return neighbors

def reconstruct\_path(self, current\_node):

path = []

while current\_node is not None:

path.append(current\_node.position)

current\_node = current\_node.parent

return path[::-1]

def astar(self, start, goal):

rows, cols = self.map.shape

open\_list = PriorityQueue()

start\_node = Node(start)

goal\_node = Node(goal)

open\_list.put((start\_node.f, start\_node))

visited = set()

visited.add(start)

while not open\_list.empty():

current\_node = open\_list.get()[1]

current\_position = current\_node.position

if current\_position == goal:

return self.reconstruct\_path(current\_node)

neighbors = self.get\_neighbors(current\_position)

for neighbor\_position in neighbors:

if neighbor\_position not in visited:

neighbor\_node = Node(neighbor\_position, current\_node)

neighbor\_node.g = current\_node.g + 1

neighbor\_node.h = self.heuristic\_cost(neighbor\_position, goal)

neighbor\_node.f = neighbor\_node.g + neighbor\_node.h

open\_list.put((neighbor\_node.f, neighbor\_node))

visited.add(neighbor\_position)

return None

def navigate\_to\_position(self, position):

goal = MoveBaseGoal()

goal.target\_pose.header.frame\_id = 'map'

goal.target\_pose.pose.position.x = position[0]

goal.target\_pose.pose.position.y = position[1]

goal.target\_pose.pose.orientation.w = 1.0

self.move\_base\_client.send\_goal(goal)

self.move\_base\_client.wait\_for\_result()

def scan\_callback(self, scan):

ranges = scan.ranges

min\_range = min(ranges)

if min\_range < 0.5:

self.object\_detected = True

def image\_callback(self, image):

cv\_image = self.bridge.imgmsg\_to\_cv2(image, desired\_encoding="bgr8")

gray\_image = cv2.cvtColor(cv\_image, cv2.COLOR\_BGR2GRAY)

objects = self.object\_cascade.detectMultiScale(gray\_image, scaleFactor=1.1, minNeighbors=5)

if len(objects) > 0:

self.object\_detected = True

def find\_object\_position(self):

rospy.loginfo("Searching for object...")

self.object\_detected = False

while not self.object\_detected:

rospy.sleep(0.1)

rospy.loginfo("Object detected!")

# You can modify this logic to determine the object's position based on the detected object

# For example, you can calculate the object's centroid from the detected bounding box

# For demonstration, we assume the object's position is at the center of the image

image\_width = 640

image\_height = 480

object\_position = (image\_width // 2, image\_height // 2)

return object\_position

def run(self):

rospy.loginfo("Finding object position...")

object\_position = self.find\_object\_position()

rospy.loginfo("Object position: {}".format(object\_position))

rospy.loginfo("Planning path...")

start\_position = (0, 0) # Replace with the actual starting position of the robot

path = self.astar(start\_position, object\_position)

if path is not None:

rospy.loginfo("Path found!")

rospy.loginfo("Following the path...")

for position in path:

self.navigate\_to\_position(position)

rospy.loginfo("Reached object position.")

rospy.loginfo("Picking up object...")

# Add your logic here to control the gripper arm and pick up the object

rospy.loginfo("Dropping off object...")

drop\_off\_position = (0, 0) # Replace with the actual drop-off position

self.navigate\_to\_position(drop\_off\_position)

rospy.loginfo("Object dropped off.")

else:

rospy.logerr("Failed to find a path to the object.")

if \_\_name\_\_ == '\_\_main\_\_':

planner = PathPlanner()

rospy.Subscriber('/scan', LaserScan, planner.scan\_callback)

rospy.Subscriber('/camera/image\_raw', Image, planner.image\_callback)

planner.run()