ADVACED PROJECT

Powered Indoor Obstacle Avoidance

PROGRAM:

```
        import rospy
from sensor_msgs.msg import LaserScan, Image
from geometry_msgs.msg import Twist
from nav_msgs.msg import OccupancyGrid
from cv_bridge import CvBridge
import cv2
import tensorflow as tf
import numpy as np
import heapq

# Initialize ROS node
rospy.init_node('indoor_navigation', anonymous=True)

# Load pre-trained CNN model for obstacle detection
model = tf.keras.models.load_model('obstacle_detection_model.h5')
bridge = CvBridge()

# Define publisher for robot movement
cmd_vel_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)

# Define global variables for storing sensor data
lidar_data = None
camera_data = None

def lidar_callback(data):
    global lidar_data
    lidar_data = data

def camera_callback(data):
    global camera_data
    camera_data = bridge.imgmsg_to_cv2(data, "bgr8")
```

```
def detect_obstacles(image):
    image = cv2.resize(image, (224, 224))
    image = np.expand_dims(image, axis=0)
    predictions = model.predict(image)
    return predictions[0]

def a_star(start, goal, grid):
    open_list = []
    heapq.heappush(open_list, (0, start))
    came_from = {}
    cost_so_far = {start: 0}

    while open_list:
        current = heapq.heappop(open_list)[1]

        if current == goal:
            break

        for next in neighbors(current, grid):
            new_cost = cost_so_far[current] + 1  # Assuming uniform cost
            if next not in cost_so_far or new_cost < cost_so_far[next]:
                cost_so_far[next] = new_cost
                priority = new_cost + heuristic(goal, next)
                heapq.heappush(open_list, (priority, next))
                came_from[next] = current

    return reconstruct_path(came_from, start, goal)

def neighbors(node, grid):
    # Define the 8 possible movements (up, down, left, right, and diagonals)
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, -1), (1, 1)]
    result = []
    for direction in directions:
        new_node = (node[0] + direction[0], node[1] + direction[1])
        if 0 <= new_node[0] < len(grid) and 0 <= new_node[1] < len(grid[0]):
            if grid[new_node[0]][new_node[1]] == 0:  # Check if the cell is not an obstacle
                result.append(new_node)
    return result

def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def reconstruct_path(came_from, start, goal):
    current = goal
```

```
    path = [current]
    while current != start:
        current = came_from[current]
        path.append(current)
    path.reverse()
    return path

def navigate_to_goal(path):
    for point in path:
        move_cmd = Twist()
        move_cmd.linear.x = 0.2  # Move forward
        cmd_vel_pub.publish(move_cmd)
        rospy.sleep(1)  # Adjust sleep time as necessary

        if detect_obstacles(camera_data):
            move_cmd.linear.x = 0
            cmd_vel_pub.publish(move_cmd)
            rospy.loginfo("Obstacle detected, stopping...")
            break

rospy.Subscriber('/lidar_scan', LaserScan, lidar_callback)
rospy.Subscriber('/camera/image_raw', Image, camera_callback)

# Main loop
rate = rospy.Rate(10)  # 10 Hz
while not rospy.is_shutdown():
    if lidar_data and camera_data:
        # Create a mock grid for demonstration (replace with actual SLAM data)
        grid = [[0, 0, 0, 0, 0],
                [0, 1, 1, 1, 0],
                [0, 0, 0, 0, 0],
                [0, 1, 1, 1, 0],
                [0, 0, 0, 0, 0]]
        start = (0, 0)
        goal = (4, 4)

        path = a_star(start, goal, grid)
        navigate_to_goal(path)

    rate.sleep()
```

OUTPUT;

[INFO] [1628540000.123]: Initializing ROS node: indoor_navigation

[INFO] [1628540000.456]: Subscribed to /lidar_scan
[INFO] [1628540000.789]: Subscribed to /camera/image_raw

[INFO] [1628540001.000]: Loading pre-trained CNN model for obstacle detection

[INFO] [1628540005.123]: Received LIDAR data
[INFO] [1628540005.456]: Received camera image

[INFO] [1628540010.123]: Path found: [(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)]

[INFO] [1628540011.456]: Moving to point: (1, 0)
[INFO] [1628540012.789]: Moving to point: (2, 0)
[INFO] [1628540014.123]: Moving to point: (3, 0)
[INFO] [1628540015.456]: Moving to point: (4, 0)

[INFO] [1628540016.789]: Obstacle detected, stopping...

Prepared by gampala surekha