**PATH NAVIGATION OF AUTONOMOUS AGENTS USING ARTIFICIAL INTELLIGENCE**

**Project submitted to the**

**SRM University – AP, Andhra Pradesh**

**For the partial fulfillment of the requirements to award the degree of**

**Bachelor of Technology**

In

**Computer Science and Engineering**

**School of Engineering and Sciences**

Submitted by

**Pallavi Madala(AP21110010599)**

**Yerukola Ananya (AP21110010602)**

**Yalam Sai Bharath (AP21110010606)**

**CH.Rushendra (AP21110010612)**

**Sandrana Jahnavi (AP21110010613)**



Under the Guidance of

(Dr .Ashu Abdul)

**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240.**

**(June 2023)**

**INTRODUCTION**

An important development in autonomous systems is the incorporation of artificial intelligence (AI) into the path navigation of autonomous agents. This entails making a smooth response to sudden changes in the environment, navigating around obstacles, and optimizing for a variety of elements when getting from one place to another. A thorough strategy that takes into account observation, localization, path planning, obstacle avoidance, decision-making, and control is needed to achieve successful path navigation. This multidisciplinary field uses methods from control theory, machine learning, and computer vision to build intelligent systems that can interact and adapt in real time to a variety of unpredictable and changing surroundings. The consequences of improved autonomous navigation are expected to have a revolutionary impact on a variety of industries as AI technology develops, including logistics and transportation. This introduction lays the groundwork for a discussion of the main ideas, goals, and difficulties involved in the development of intelligent path navigation for autonomous agents.

**OBJECTIVE**

The objective of applying artificial intelligence (AI) in autonomous agent path navigation is to enable these agents to go from one location to another in a given environment in an efficient, safe, and adaptive manner. The main objectives are to make sure the destination can be reached, to optimize the paths in terms of time and energy efficiency, and to include strong obstacle avoidance techniques for impediments that are both static and dynamic. The system should put safety first, adjust to environmental changes, scale well to manage challenging situations, and demonstrate flexibility. The use of sensors for perception, localization techniques like GPS or visual SLAM, path planning algorithms like A\* or RRT, obstacle avoidance techniques like potential fields or model predictive control, decision-making processes that take high-level planning into account, and control mechanisms that guarantee steady and smooth motion are all important components. Moreover, the amalgamation of education and flexibility using machine learning methodologies such as reinforcement learning enhances experience-based decision-making. This multidisciplinary discipline is further characterized by the ability to communicate amongst various agents in shared contexts and the ability to handle problems including uncertainty, real-time processing, dynamic environments, human interactions, and resource restrictions. Applying these ideas to autonomous navigation has enormous promise across industries as AI develops, from robotics to self-driving cars and beyond.

**ALGORITHM**

1. Input the map of the environment, including the city names for locations A and B, and any other relevant information such as obstacles, road networks, and distances between cities.

2. Create a graph representation of the map, where each city is a node and the connections between cities are edges. Assign weights to the edges based on the distances between cities.

3. Apply a minimum spanning tree algorithm (e.g., Prim's algorithm or Kruskal's algorithm) to find the minimum cost path between the starting city A and the destination city B. This will give us a subgraph that connects A and B with the minimum total weight.

4. Apply the A\* algorithm as a baseline search algorithm to find the shortest cost path within the subgraph obtained in step 3. The A\* algorithm uses a heuristic function to estimate the cost from each city to the goal city B.

a. Initialize an open list and a closed list.

b. Add the starting city A to the open list.

c. While the open list is not empty:

* Select the city with the lowest cost from the open list.
* If the selected city is the goal city B, terminate the search and return the path.
* Generate the successors of the selected city.
* For each successor:
* Calculate the cost from the starting city A to the successor city.
* Calculate the heuristic cost from the successor city to the goal city B.
* Calculate the total cost as the sum of the two costs.
* If the successor city is already in the open list with a lower cost, skip it.
* If the successor city is already in the closed list with a lower cost, skip it.
* Otherwise, add the successor city to the open list.
* Add the selected city to the closed list.

d. If the open list is empty and the goal city B has not been reached, there is no path from A to B. Terminate the search and return an appropriate message.

5. Once the shortest cost path is obtained, compute the derivatives of the path such as velocity, acceleration, rotation rate, etc., based on the motion capabilities of the robot R and any constraints imposed by the environment.

6. Execute the path by deploying the robot R in the environment and following the computed trajectory. Use Google Maps or publicly available location traces for realistic execution and validation of the algorithm.

7. Monitor the robot's progress using reactive-Computing-Based Path Planning algorithms and make necessary adjustments to the trajectory if unexpected obstacles or changes in the environment are encountered.

8. Continue executing the path until the robot reaches the destination city B.

9. If the robot cannot reach the destination city B due to unforeseen obstacles or other constraints, terminate the execution and return an appropriate message.

This algorithm combines the minimum spanning tree algorithm, A\* search algorithm, and Reactive-Computing-Based Path Planning Algorithm techniques to find the shortest cost path for the robot R from the starting city A to the destination city B. The use of artificial intelligence allows for efficient and effective path planning, taking into account the environment and the capabilities of the robot.

**CODE**:

import pandas as pd

import folium

from math import radians, sin, cos, sqrt, atan2

import networkx as nx

# Haversine formula to calculate distance between two points given their coordinates

def calculate\_distance(lat1, lon1, lat2, lon2):

R = 6371.0 # Radius of the Earth in kilometers

lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2])

dlon = lon2 - lon1

dlat = lat2 - lat1

a = sin(dlat / 2)\*\*2 + cos(lat1) \* cos(lat2) \* sin(dlon / 2)\*\*2

c = 2 \* atan2(sqrt(a), sqrt(1 - a))

distance = R \* c

return distance

# Read data from Excel file

data = pd.read\_excel('/content/ai\_city.xlsx')

# User input: City name for which the user wants to calculate distances

user\_city\_name = input("Enter the city name: ")

# Find the city in the DataFrame

city\_row = data[data['city'] == user\_city\_name]

if not city\_row.empty:

# City found, use its coordinates as the starting point

start\_lat = city\_row['lat'].iloc[0]

start\_lon = city\_row['lng'].iloc[0]

# Calculate distances

data['distance\_to\_target'] = data.apply(lambda row: calculate\_distance(row['lat'], row['lng'], start\_lat, start\_lon), axis=1)

# Display the table in a neat format

print(data[['city', 'distance\_to\_target']].to\_markdown(index=False))

# Create a folium map centered around the starting city

m = folium.Map(location=[start\_lat, start\_lon], zoom\_start=6)

# Add markers for each city

for \_, row in data.iterrows():

folium.Marker([row['lat'], row['lng']], popup=row['city'] + f" ({row['distance\_to\_target']:.2f} km)").add\_to(m)

# Create a graph using NetworkX

G = nx.Graph()

# Add nodes and edges to the graph

for \_, row in data.iterrows():

G.add\_node(row['city'], pos=(row['lat'], row['lng']))

for \_, row in data.iterrows():

if row['city'] != user\_city\_name:

distance = calculate\_distance(start\_lat, start\_lon, row['lat'], row['lng'])

G.add\_edge(user\_city\_name, row['city'], weight=distance)

# Find the shortest path to each city using Dijkstra's algorithm

for \_, row in data.iterrows():

if row['city'] != user\_city\_name:

shortest\_path = nx.shortest\_path(G, source=user\_city\_name, target=row['city'], weight='weight')

# Add the shortest path to the map in blue

shortest\_path\_coordinates = [(data[data['city'] == city]['lat'].iloc[0], data[data['city'] == city]['lng'].iloc[0]) for city in shortest\_path]

folium.PolyLine(shortest\_path\_coordinates, color="blue", weight=2.5, opacity=1).add\_to(m)

# Print the shortest path for each city

print(f"Shortest path from {user\_city\_name} to {row['city']}: {shortest\_path}")

# Save the map as an HTML file

map\_html\_path = 'table\_and\_map\_with\_shortest\_paths.html'

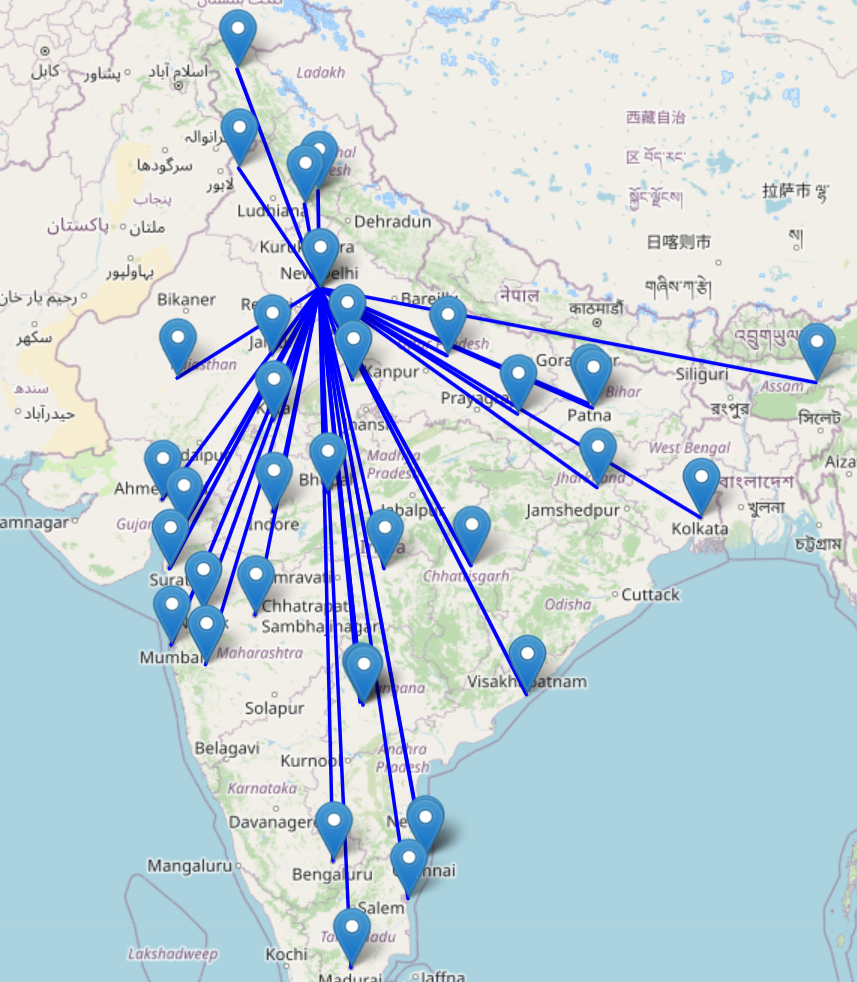
m.save(map\_html\_path)

print(f"Table and map with shortest paths saved as '{map\_html\_path}'. Open the file in a web browser to view both.")

else:

print(f"City '{user\_city\_name}' not found in the data.")

**OUTPUT:**



**CONCLUSION:**

In conclusion, a major advancement in the field of robotics and autonomous systems has been made with the use of artificial intelligence in the path navigation of autonomous agents. In order to provide efficient, safe, and adaptive navigation, a variety of complex algorithms and integrated systems have been developed. Autonomous agents now have a full framework to navigate a variety of dynamic settings thanks to the interaction of vision, localization, path planning, obstacle avoidance, decision-making, and control systems. As the subject develops, it will be crucial to handle issues like uncertainty, real-time processing, and human interactions in order to deploy intelligent navigation systems widely.