Title: Integrating Machine Learning into Path Planning for Autonomous Systems

Tahmid Zaman Tahi (G29401540), Mashrur Wasek (G33963852)

Abstract

This project explores the integration of classical and machine learning-based algorithms in path planning for autonomous systems. We focus on comparing A* (Al-Taharwa, Sheta, & Al-Weshah, 2008), Q-Learning (Le Duc Hanh & Vo Duy Cong, 2023), and Monte Carlo Tree Search (MCTS) in static (Eiffert, Kong, Pirmarzdashti, & Sukkarieh, 2020) and dynamic environments, highlighting their efficiency, adaptability, and computational requirements. This project also explores Deep Q-learning approach in dynamic environments for optimal path planning (Huang, Qin, Ling, & Lan, 2023). Key findings demonstrate varied strengths of these algorithms in different scenarios, with implications for their application in real-world autonomous navigation systems.

1. Introduction

The advent of autonomous systems necessitates advanced path planning techniques that are efficient, adaptable, and capable of handling complex environments. This project aims to compare and contrast the performance of three distinct algorithms: A*, Q-Learning, and MCTS. Each algorithm brings unique strengths to path planning—A* with its efficient, heuristic-based searching, Q-Learning with its adaptive learning capabilities, and MCTS with its probabilistic exploration. Deep Q-learning was explored for dynamic environment to handle the complexity that arises with a complex state-space. The objective is to assess their performance in static environments and then extend this comparison to dynamic settings, offering valuable insights into their applicability in real-world autonomous navigation systems.

2. Background and Related Work

Traditional Path Planning Methods

Path planning is crucial in robotics and autonomous systems, traditionally relying on algorithms like Dijkstra's and the A* for deterministic environments. However, these methods have limitations in dynamic or complex environments. Our goal is to find out the optimum algorithm for scenarios that mimic real life so that it can be used for practical purposes.

Previous Studies and Applications

Previous research often focuses on individual algorithmic performance, with limited comparative studies across diverse environments. This project draws inspiration from studies demonstrating the successful application of these algorithms in robotics, gaming, and autonomous vehicle navigation.

3. Methodology

We have run these algorithms in a static environment as a test case scenario. Building upon that, we have created a custom environment using Open AI’s gym library with static and dynamic elements to mimic real life scenarios and implemented the algorithms on the said environment to compare performance metrics.

Static Environment Implementation

Environment setup:

Grid-Based Model: A discretized grid represents the environment, with obstacles and a designated goal.
State Representation: For A* and MCTS, the state includes the robot's position and obstacle layout. For Q-Learning, it also includes historical data to inform decision-making.
Implementation of Algorithms
A*: Implemented with a standard heuristic function (Euclidean distance). The algorithm recalculates paths when encountering obstacles.
Q-Learning: Utilizes a reward-based system, learning optimal paths through trial and error. The agent's state includes its position and encountered obstacles, updating its policy based on a Q-table.
MCTS: Builds a search tree where nodes represent states and edges represent possible actions. The algorithm probabilistically explores various paths and strengthens the strategy based on the reward structure.

Dynamic Environment Implementation

Environment Setup:

Grid Configuration: The environment, termed AdvancedRobotEnv, is a square grid of configurable size. Each cell in the grid can be empty, contain a static obstacle, a dynamic obstacle, or the goal.
State Space: The state space includes the robot's position and the positions of all static and dynamic obstacles, represented as a vector.
Action Space: The agent can perform four actions: move up, down, left, or right within the grid.
Reward Mechanism: A reward system is designed to encourage the agent to reach the goal and penalize collisions with obstacles. The rewards are structured as follows:
A high positive reward for reaching the goal.
A negative reward for colliding with an obstacle.
A smaller penalty for each step taken, to encourage efficient pathfinding.
A small positive reward on reaching closer to the goal, based upon calculating diagonal distance to the goal.

DQN Agent

Neural Network Architecture

Function Approximator: A multi-layer perceptron (MLP) with several dense layers, each followed by a ReLU activation function. The output layer, with linear activation, maps to the estimated Q-values for each action.
Input and Output: The network inputs the environment state and outputs a Q-value for each possible action.

Learning Algorithm

Experience Replay: Implements a replay buffer to store and sample experience tuples (s,a,r,s'), enhancing data efficiency and breaking temporal correlations.
Epsilon-Greedy Exploration: Utilizes an epsilon-greedy policy for action selection, balancing exploration and exploitation. Epsilon is decayed over time according to a predefined schedule.
Q-Learning Update: Employs the Bellman equation for Q-learning, with iterative updates to minimize the loss between predicted Q-values and target Q-values.

Target Network

Stabilization Technique: Incorporates a target network, a periodically updated clone of the primary network, to compute stable target Q-values during training.

Checkpointing and Resumption

Model Persistence

Checkpointing Mechanism: Model states are periodically serialized and saved, capturing weights, optimizer state, and episode count.
Automated Checkpoint Selection: The script identifies and loads the most recent checkpoint, enabling training resumption post-interruption.

Visualization and Progress Tracking

Training Monitoring
Progress Bar: Integrates tqdm for real-time training progress visualization, displaying episodes and steps.
Performance Metrics: Tracks metrics like path length per episode, computation time, and cumulative reward. These metrics are plotted post-training using matplotlib for performance analysis.

Computational Considerations

Memory Management

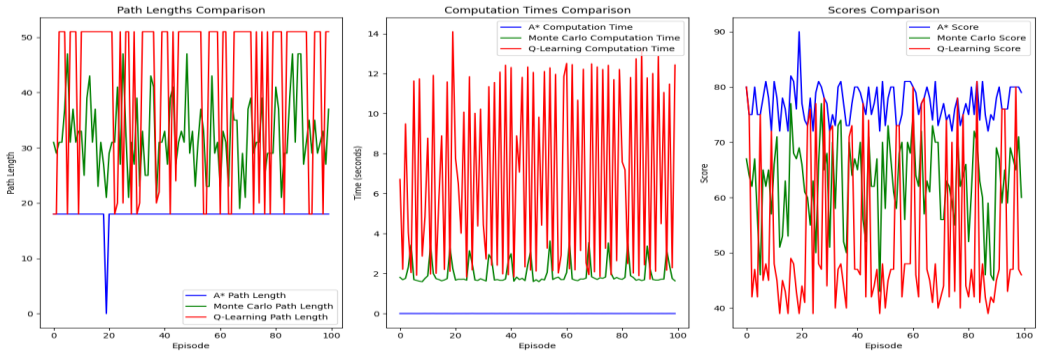
Buffer Size: The size of the replay buffer is a critical parameter, balancing memory consumption and learning efficiency.
Batch Processing: Training updates are performed on mini-batches, a compromise between stochastic and batch gradient descent, optimizing both memory usage and convergence speed.

4. Performance Metrics:

Key performance metrics such as path lengths, computation times, and episode scores are tracked and visualized using matplotlib to assess the agent's learning progress.
Efficiency: Measured by the time taken to find a path and the length of the path.
Adaptability: Assessed by the algorithm's ability to adjust to changes in the environment without significant performance degradation.

5. Results

Static Environment



A Search*: Exhibited optimal pathfinding with the shortest paths and highest scores.
Monte Carlo: Longer paths due to its stochastic nature.
Q-Learning: Demonstrated learning capabilities with varied path lengths, needs longer episode lengths

Dynamic Environment

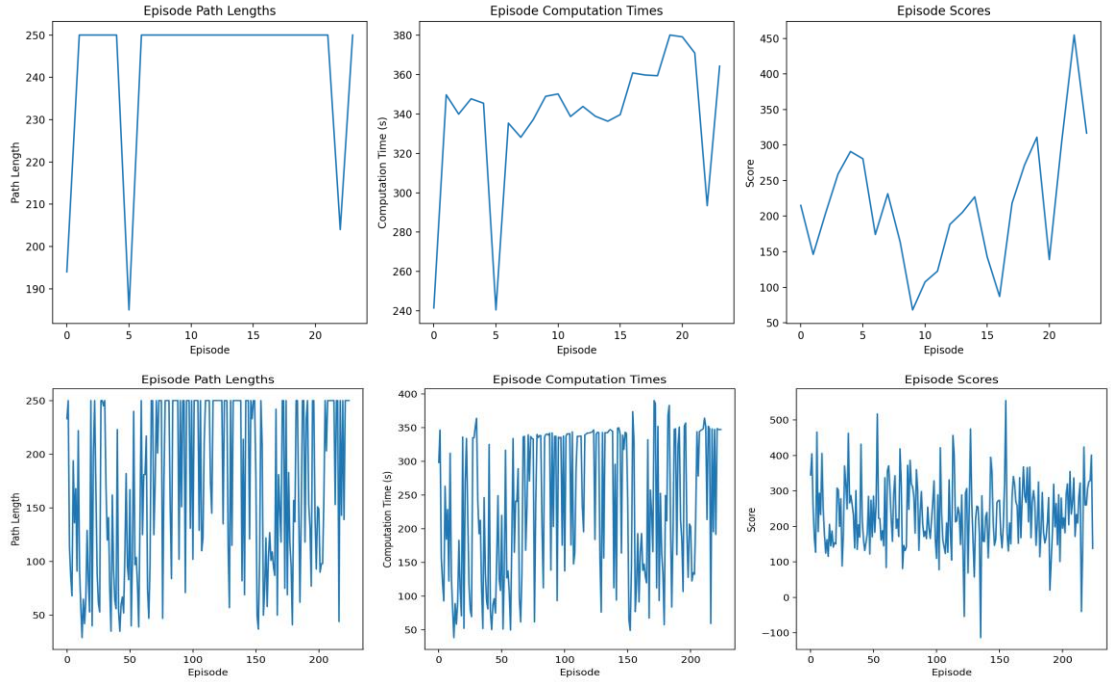


Fig: 1. Trained for 250 episodes, continued after last checkpoint of 225 2. 600 episodes (last checkpoint 375)

DQN Agent: Adapted to dynamic obstacles, showing improvement in pathfinding strategies over training episodes.

Computation Times
A Search*: Fastest in the static environment.
Monte Carlo and Q-Learning: Slower due to their respective computational complexities.
DQN: Required significant training time but offered real-time performance once trained.

Adaptability
A*, Monte Carlo, Q-Learning: Consistent in static environments but not applicable to dynamic scenarios.

DQN: Showed robustness in adapting to dynamic obstacles, learning efficient navigation strategies over time.

6. Discussion

Efficiency and Real-Time Application
A* Search: Ideal for static environments where real-time computation is crucial.
Monte Carlo: Simple but less efficient, suitable for less complex scenarios.
Q-Learning and DQN: While computationally intensive, these learning-based approaches excel in environments where adaptability is key.

Learning and Adaptability
Q-Learning: Effective in static environments but lacks the complexity needed for dynamic scenarios.
DQN: Demonstrates significant potential in dynamic environments, learning from interactions and adapting to changes.

Future Works
Since our ultimate goal is to implement this in a real-world scenario, there is still a lot of work to be desired. We could improve upon our current work in the following ways:

- Algorithm Enhancement: Further optimization of DQN, including tuning its architecture and hyperparameters for even greater efficiency and faster convergence.

- Complex Environmental Models: Testing the algorithms in more complex and unpredictable environments that better mimic real-world conditions, such as those with irregular terrain and varying obstacle dynamics.

- Real-World Implementation: Applying these algorithms to real-world scenarios, such as autonomous vehicle navigation, robotic path planning in cluttered spaces, and dynamic route finding in logistics.

- Hybrid Models: Investigating hybrid approaches that combine the strengths of different algorithms. For instance, integrating the heuristic efficiency of A* with the adaptive learning capabilities of DQN could yield a more robust pathfinding solution.

- Deep Learning Enhancements: Exploring advancements in deep learning, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to improve the model's ability to process spatial and temporal information in dynamic environments.

- Multi-Agent Systems: Expanding the study to include multi-agent scenarios where multiple entities interact and navigate simultaneously, posing additional challenges in pathfinding and coordination.

By pursuing these avenues, future research can significantly advance the field of pathfinding, leading to more sophisticated and versatile navigation solutions applicable in a wide range of disciplines.

Contributions

Nature of Work	Contributed by
Static environment setup	Mashrur Wasek
Dynamic environment setup	Tahmid Zaman Tahi
Algorithm implementations in static env	Mashrur Wasek
Deep Q-learning implementation	Tahmid Zaman Tahi
Initial experimentations and paper reading	Equal contributions
Final report write-up	Equal contributions

References

[1] Al-Taharwa, I., Sheta, A., & Al-Weshah, M. (2008). A Mobile Robot Path Planning Using Genetic Algorithm in Static Environment. Journal of Computer Science, 4(4), 341-344.

[2] Eiffert, S., Kong, H., Pirmarzdashti, N., & Sukkarieh, S. (2020). Path Planning in Dynamic Environments using Generative RNNs and Monte Carlo Tree Search.

[3] Le Duc Hanh, & Vo Duy Cong. (2023). Path Following and Avoiding Obstacle for Mobile Robot Under Dynamic Environments Using Reinforcement Learning. Journal of Robotics and Control (JRC).

[4] Huang, R., Qin, C., Ling, J., & Lan, X. (2023). Path planning of mobile robot in unknown dynamic continuous environment using reward-modified deep Q-network. Optimal Control Applications and Methods.

[5]