Autonomous Agent Navigation in 2D Grid with Moving Obstacles

Ngoc Pham
May 9, 2025

Abstract

This project focuses on developing an interactive simulation where an autonomous agent navigates across a 2D grid while avoiding both fixed and moving obstacles. The goal is to model a realistic environment where decision-making must adapt to dynamic changes. To achieve this, we apply supervised learning techniques, specifically Behavioral Cloning and DAgger, to train the agent on movement prediction tasks. Additionally, we use a hybrid A* pathfinding algorithm to plan feasible routes around obstacles, both static and dynamic. Real-time visualization is implemented using Matplotlib, providing a clear and continuous view of the agent's journey toward its goal. The final system successfully integrates user interaction, data-driven model training, and real-time simulation. It performs reliably across different scenarios, demonstrating the effectiveness of combining traditional pathfinding with machine learning-based behavior prediction.

1 Introduccion

In the fields of robotics and artificial intelligence, autonomous navigation in dynamic environments remains a fundamental and challenging problem. An intelligent agent must not only plan an efficient path from a starting point to a destination, but also react in real-time to both static and moving obstacles that may block its way. Achieving this balance between long-term planning and short-term adaptation is critical for safe and efficient navigation in real-world applications, such as self-driving cars, mobile robots, and drones.

This project is designed to address these challenges through a practical simulation. We aim to build a system where an autonomous agent can learn how to move intelligently across a 2D grid while avoiding obstacles. The agent must be capable of both following optimal paths and adapting dynamically to changes in its environment, especially when obstacles move unpredictably. To achieve this, the agent must learn movement patterns from historical data and improve its behavior over time through feedback.

To accomplish these goals, we incorporate several key technologies and concepts:

+ Machine Learning Techniques: We apply Behavioral Cloning to allow the agent to mimic expert behaviors directly from data, and DAgger (Dataset Aggregation) to improve performance by iteratively correcting the agent's mistakes during training.

+ Path Planning Algorithms: We integrate a hybrid A* pathfinding method to ensure that the agent can compute viable routes, taking into account both static and moving obstacles at each time step.

+ Real-Time Visualization: Using Matplotlib, we provide a dynamic and continuous graphical view of the agent's movements and interactions with obstacles, making it easy to observe the decision-making process and detect any issues visually.

+ Interactive Grid Setup: The simulation includes a user-friendly Tkinter GUI that allows users to manually place the start point, goal, and both fixed and moving obstacles on the grid. This interactivity makes it possible to create diverse scenarios for training and testing the agent.

Through this combination of supervised learning, path planning, and interactive simulation, the project provides a flexible framework for experimenting with autonomous agent behavior in dynamic settings. It also lays a foundation for future extensions into more complex or realistic environments.

2 Models, methods or materials

To support the development of an intelligent navigation agent, we designed a full pipeline that includes data generation, model training, evaluation, and real-time simulation.

2.1. Dataset Creation

The dataset used for training the agent was automatically generated using a custom Python script (create_dataset.py). The process involved several steps:

+ Randomly placing fixed obstacles and moving obstacles on a 2D grid.

+ Randomly choosing start and goal points while ensuring they do not overlap.

+ Simulating agent movement between the start and the goal using the hybrid A* pathfinding algorithm.

+ Recording the state transitions for every move: the start location, the current location, and the next location.

Each data sample thus captured the context of movement decisions, resulting in a dataset structured as:

Sample=(Start_X,Start_Y,Current_X,Current_Y,Next_X,Next_Y)

The dataset was saved in CSV format for easy manipulation and loading.

2.2. Learning Models

The navigation problem in this project is framed as a sequential decision-making task. The agent must learn a policy that maps observed states to

appropriate actions, based on supervised learning from expert demonstrations. Two primary learning strategies are employed: Behavioral Cloning (BC) and Dataset Aggregation (DAgger).

Behavioral Cloning (BC)

Behavioral Cloning treats the sequential decision-making problem as a standard supervised learning problem.

Given a dataset D of expert demonstrations, consisting of state-action pairs:

$$D = \{(s_i, a_i)\}_{i=1}^{N}$$

where:

+ s_i is the observed state (e.g., agent's current position relative to goal and obstacles),
+ a_i is the expert action (e.g., the next movement step),

the objective of BC is to learn a policy π_θ parameterized by θ that minimizes the expected loss between the predicted actions and the expert actions over the training data:

$$\min_{\theta} E_{(s,a)\sim D}[L(\pi_\theta(s), a)]$$

where L is typically the Mean Squared Error (for continuous actions) or Cross-Entropy Loss (for discrete actions).

In simple terms, the model learns by **directly imitating** the expert.
However, Behavioral Cloning assumes that the agent will always encounter states similar to those seen in the training data.
In practice, **errors can compound**: a small mistake can cause the agent to enter an unfamiliar state, where it is more likely to make further mistakes, leading to cascading failures.

This limitation motivates the use of a more robust method: DAgger.

Dataset Aggregation (DAgger)

DAgger addresses the **covariate shift** problem in Behavioral Cloning by collecting additional training data from the agent's own induced distribution of states.

The algorithm proceeds iteratively:

(1) **Initialization:**
Start by training an initial policy π using supervised learning on expert demonstrations $D_{exp ext}$

(2) **Iterative improvement:**
For each iteration i=1,2,…,k:

- Deploy the current policy π_i to act in the environment, generating a

sequence of visited states $s \sim \pi_i$.

- For each visited state s, query the expert for the correct action a^*.
- Aggregate these new (state, expert-action) pairs into the dataset:

$$D \leftarrow D \cup \{(s, a^*)\}$$

- Retrain the policy π_{i+1} on the updated dataset D.

The overall objective of DAgger is still to minimize the expected loss, but now the expectation is over the **state distribution induced by the agent's own policy**:

$$\min_{\theta} E_{s \sim d^{\pi}}[L(\pi_{\theta}(s), a^*(s))]$$

where d^{π} is the distribution of states visited under the agent's (current) policy π, and $a^*(s)$ is the expert's action at state s.

By continuously exposing the agent to its own mistakes and providing corrective feedback, DAgger significantly improves the agent's robustness, especially in complex or unpredictable environments.

In both BC and DAgger, in this project, we model the movement prediction as two independent continuous regression problems — one predicting the next X coordinate and the other predicting the next Y coordinate — using standard linear regression techniques.

2.3. Data Preprocessing

Before training, the features were **normalized** to stabilize learning. We applied standard score normalization to each feature:

$$X_{normalized} = \frac{X - \mu}{\sigma}$$

where:

+ μ is the mean of the feature,
+ σ is the standard deviation.

This ensures all input variables are centered around zero and scaled properly.

2.4. Model Training and Evaluation

We used **Linear Regression** models from scikit-learn to predict the next movement coordinates.

Training followed a standard supervised learning workflow:

+ Split the dataset into training and testing sets (80/20 split).
+ Train two independent linear models: one for predicting **Next_X**, one for **Next_Y**.
+ Evaluate model performance using two key metrics:

Mean Squared Error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Coefficient of Determination (R^2):

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

Where:

+ y is the true value,
+ $\hat{y}$ is the predicted value,
+ $\bar{y}$ is the mean of the true values.

The evaluation results, including actual vs. predicted paths, were visualized using Matplotlib to offer an intuitive understanding of model accuracy.

2.5. Pathfinding Algorithm

While learning-based methods allow the agent to predict its next movement step based on past experience, they alone are not sufficient to guarantee efficient navigation in complex environments with dynamic obstacles. To enhance the agent's robustness, we incorporate a traditional path planning component known as the **Hybrid A*** algorithm.

Hybrid A* is an extension of the classical A* search algorithm, adapted to better handle continuous space and dynamic constraints. Unlike standard A*, which assumes discrete grid movements and sharp turns, Hybrid A* takes into account the geometry of feasible trajectories, allowing smoother and more realistic paths that are critical for real-world navigation tasks.

At its core, Hybrid A* maintains the heuristic-driven search strategy of A*, which balances two quantities:

+ The cost g(n) to reach a node n from the starting point.
+ An admissible heuristic h(n) estimating the cost from n to the goal.

The total estimated cost f(n) for reaching the goal via nnn is computed as:

$$f(n)=g(n)+h(n)$$

where the heuristic h(n)h(n)h(n) must be designed carefully to ensure that the search remains both efficient and optimal.
In our case, the heuristic used is the **Euclidean distance** between the current position a and the goal position b, defined as:

$$h(a,b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}$$

This continuous heuristic encourages the agent to favor paths that move it closer to the goal in straight lines when possible, promoting efficient navigation.

In each search iteration, the agent explores the set of neighboring states, which are determined not only by direct movements (e.g., left, right, up, down) but also by diagonal moves, reflecting a more flexible motion model. Each potential move is evaluated for feasibility: collisions with either fixed or moving obstacles must be avoided. The collision checking is performed by computing the distance between the candidate position and all obstacles, and rejecting any move where the distance falls below a certain threshold.

One important feature of the implemented Hybrid A* variant is **dynamic replanning**. Since moving obstacles change their positions at each timestep, the environment can evolve unpredictably. Rather than planning a static path at the beginning of the simulation, the agent continuously replans its trajectory at each time step, considering the updated positions of the moving obstacles. This strategy ensures that the agent remains adaptive and can respond to newly emerging threats or blocked paths in real-time.

Formally, at each decision step t:

+ The agent observes the current environment E_t.
+ It recomputes a new optimal path P_t from its current position to the goal using Hybrid A*.
+ It executes the first move on the new path and updates its state accordingly.

This reactive planning cycle allows the agent to weave intelligently through a dynamic field of obstacles, constantly adjusting its behavior to the latest information.

By integrating Hybrid A* into our system, we bridge the gap between **high-level learning-based behavior** and **low-level motion feasibility**, resulting in a more capable and resilient autonomous agent.

2.6. Tools and Frameworks

The project was built using a rich set of Python tools:

+ **Python:** The primary programming language.
+ **Pandas:** For efficient dataset manipulation.
+ **NumPy:** For numerical operations.
+ **Scikit-learn:** For training and evaluating machine learning models.
+ **Matplotlib:** For visualization, both during training and real-time simulation.
+ **Tkinter:** For creating an interactive grid setup GUI.

Each tool played a specific role in ensuring the system is modular, efficient, and easy to extend.

3 Results

The evaluation of our project involved both offline model training and real-time simulations, offering a comprehensive view of how well the agent could navigate complex environments.

The first phase began with the generation of the dataset. As shown in the initial dataset preview, each sample records the start position, current

position, and the agent's intended next move. Early examples demonstrate that the generated data includes diverse movement patterns, helping the agent learn navigation behaviors across varied situations.

Training results for the Behavioral Cloning model reveal a moderate fit to the expert data. The evaluation metrics show that the model achieved a Mean Squared Error (MSE) of 0.8412 for the next X-coordinate and 0.8045 for the next Y-coordinate. Correspondingly, the R^2 scores are 0.1414 and 0.1876, indicating that while the model captures general trends, it still struggles with precise predictions. When DAgger was applied to augment the training process, performance improved. The MSE for Next_X decreased to 0.7586, and the R^2 score rose to 0.2308, suggesting that exposing the model to its own mistakes during training leads to better generalization.

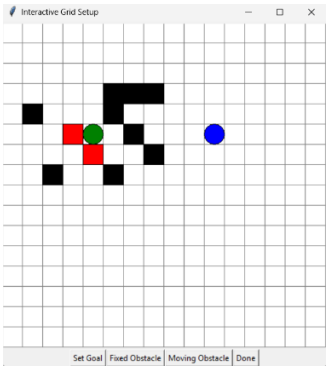
The evaluation results are summarized below:

Method	MSE (Next_X)	R2 (Next_X)	MSE (Next_Y)	R2 (Next_Y)
Behavioral Cloning	0.8412	0.1414	0.8045	0.1876
DAgger	0.7586	0.2308	0.799	0.2102

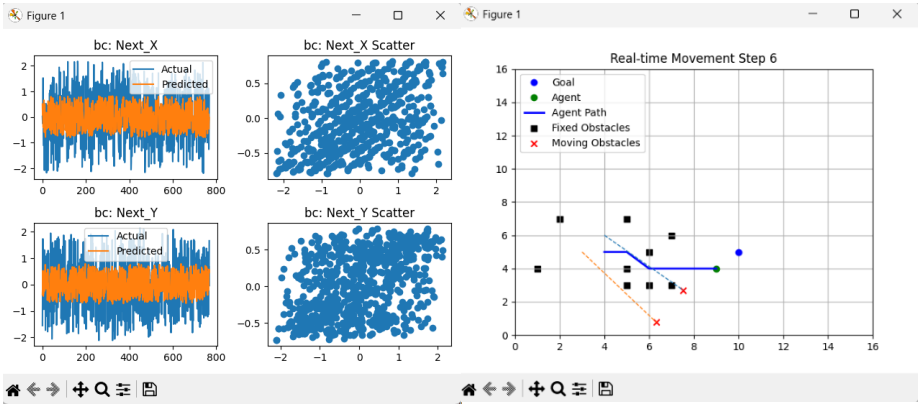
In the visualization of model predictions, the time series plots clearly show the separation between the ground truth (actual values) and the model's predicted values. Especially along the X-axis, the predicted trajectory tends to align better with the true path, while along the Y-axis, predictions show more variance. The scatter plots further confirm this trend: points are dispersed around the ideal diagonal line, but clustering is noticeably improved after using DAgger.

Transitioning into real-time simulation, the agent's behavior reflects the strengths and limitations observed during training. In the interactive setup, users placed the agent start point at [4,5], the goal at [10,5], with several static and dynamic obstacles scattered around the grid. As depicted in the live simulation frames, the agent successfully planned and adjusted its path while accounting for moving obstacles. The green circle represents the agent's current position, while the moving obstacles (marked with red crosses) shift over time, forcing the agent to replan dynamically using the hybrid A* algorithm.

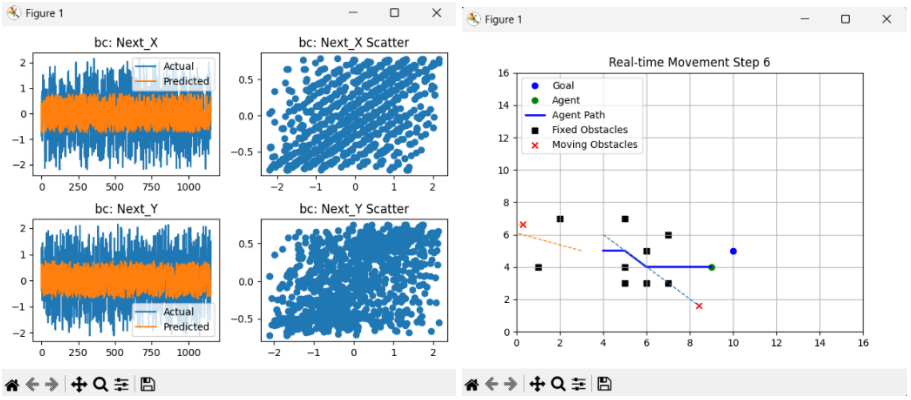
Interestingly, even when moving obstacles created temporary blockages, the agent managed to reroute itself around them without collisions. The blue path drawn on the graph updates step-by-step, reflecting real-time replanning as moving obstacles altered the environment.



The simulation results of the Behavioral Cloning algorithm:



The simulation results of the DAgger algorithm:



Overall, the results demonstrate a successful integration between learning-based movement prediction and traditional motion planning. Although prediction errors from the model introduce slight deviations, the hybrid A* planner compensates for these by maintaining a safe and goal-directed trajectory. Especially after using DAgger, the agent exhibits noticeably smoother and more confident paths through dynamic, obstacle-filled environments.

4 Discussion or Conclusiones

The results of this project demonstrate that Behavioral Cloning provides an acceptable baseline for autonomous navigation. Despite the simplicity of the approach, the agent is able to learn reasonable movement patterns from expert demonstrations, maintaining a low error rate across both X and Y movement predictions. The integration with traditional pathfinding further stabilizes performance, allowing the agent to successfully reach its goal in dynamic environments.

However, it is clear that Behavioral Cloning alone has inherent limitations, particularly when the agent encounters unfamiliar states. The introduction of DAgger into the training process significantly alleviates this issue. By allowing the agent to learn from its own mistakes during execution, DAgger leads to tangible improvements in both prediction accuracy and overall trajectory robustness. The agent becomes better equipped to handle edge cases and unforeseen scenarios, as reflected in the improved evaluation metrics and smoother real-time movements.

One of the notable strengths of this project lies in its modular, object-oriented design. By separating components such as the GUI, data processing, learning models, and pathfinding logic, the system is highly extensible. Future modifications, such as experimenting with new learning algorithms or enhancing the environment setup, can be incorporated with minimal friction. Moreover, the interactive grid setup offers users direct control over simulation parameters, making the platform flexible for a wide range of test cases and research experiments.

Nevertheless, there are limitations that must be acknowledged. The simulated moving obstacles, while sufficient for basic testing, do not fully capture the complexity and unpredictability of real-world dynamic environments. Their movements are deterministic and lack adversarial behaviors or sophisticated interactions. Furthermore, the use of simple linear regression models, although adequate for initial experimentation, restricts the expressiveness of the learned policy. In more complex settings, these models may fail to capture nonlinear dependencies and intricate state-action mappings.

Looking forward, several promising directions for future development emerge. Replacing the current Behavioral Cloning and linear models with **Deep Reinforcement Learning** frameworks could empower the agent to make more nuanced and adaptive decisions in highly dynamic environments. Incorporating **neural network-based policies** would allow the system to model more complex movement strategies and generalize better across different obstacle configurations. Additionally, increasing the complexity of moving obstacles — for example, by introducing unpredictable trajectories or adversarial behaviors — would create a more realistic and challenging environment for the agent to navigate. Finally, optimizing the computational efficiency of the real-time planning component would be crucial for scaling the system to larger grids and more sophisticated multi-agent scenarios.

In summary, while the current system successfully demonstrates the viability of combining supervised learning with dynamic path planning for autonomous navigation, it also lays a strong foundation for more advanced future work. Enhancements in learning algorithms, environment complexity, and system scalability will be key to pushing the agent's capabilities closer to real-world autonomy.

5 References

- Abbeel, P., & Ng, A. Y. (2004). *Apprenticeship learning via inverse reinforcement learning*. Proceedings of the Twenty-first International Conference on Machine Learning (ICML), 1–8. https://doi.org/10.1145/1015330.1015430

- Ross, S., Gordon, G., & Bagnell, J. A. (2011). *A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning*. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS), 627–635.

- Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). *A formal basis for the heuristic determination of minimum cost paths*. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100–107. https://doi.org/10.1109/TSSC.1968.300136

- Dolgov, D., Thrun, S., Montemerlo, M., & Diebel, J. (2008). *Practical Search Techniques in Path Planning for Autonomous Driving*. Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR).

- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825–2830. https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

- Hunter, J. D. (2007). *Matplotlib: A 2D Graphics Environment*. Computing in Science & Engineering, 9(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

- Python Software Foundation. (n.d.). *Tkinter — Python interface to Tcl/Tk*. Retrieved April 26, 2025, from https://docs.python.org/3/library/tkinter.html

- Russell, S. J., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.