Learning Based Navigation for Robots in Random 2D Worlds

Max Boulat, Tanya Neustice, and Dylan Scott-Dawkins

Department of Engineering, San Diego University

AAI-501-01: Introduction to Artificial Intelligence

Andrew Van Benschoten

August 11, 2025

Abstract

Autonomous navigation through complex environments is a fundamental challenge in robotics and artificial intelligence. This project explores whether a robot can learn to navigate in a randomly generated two-dimensional world with obstacles using supervised learning. A world generator creates 2D maps populated with single-tile obstacles and random linear walls; an agent equipped with eight radial sensors is tasked with navigating between two randomly selected points while avoiding obstacles. An A* path-planner produces optimal actions, which serve as the supervisory signal for training. We perform exploratory data analysis and evaluate multiple classification algorithms, including Random Forest, XGBoost, logistic regression, support vector machines, Naïve Bayes, K-nearest neighbors, and a neural network, on the resulting dataset. We find that a wide variety of machine learning models are capable of generalizing obstacle avoidance and goal-seeking behavior with the right features.

Keywords: Autonomous Navigation, Supervised Learning, Path Planning, Sensor-Based Perception, Classification Algorithms

Robots in Random 2D Worlds

AI models capable of navigating partially observable environments under uncertainty have significant potential across various fields, including self-driving cars, exploration, search and rescue missions, and military uses.

In this study, we investigate whether a robot can learn to navigate random two-dimensional environments with obstacles using supervised learning. Using A* as an expert, we generate trajectories, collect sensor readings, and take actions. We then train a variety of classification models to predict the next move from sensor input, compare their performance, and discuss the inherent limitations. Our objectives are to:

1. Define an artificial world and sensor model suitable for machine-learning experiments.
2. Generate a labelled dataset by following optimal paths computed by A*.
3. Analyze the dataset to understand feature distributions, correlations, and potential issues such as label contradictions.
4. Train and evaluate different machine-learning algorithms on the navigation problem.

Literature review

The difficulty of learning navigation policies from supervised data is well recognized in the literature. Pomerleau (1989) demonstrated the viability of training a neural network on camera and laser readings with the ALVINN system, a rudimentary self-driving truck capable of autonomous steering; while promising, it worked only on simple road scenes and suffered when the environment changed. Kaelbling, Littman and Cassandra (1998) formalized POMDPs (Partially Observable Markov Decision Processes), an extension of the Markov Decision Process method to the problem of optimizing route selection in the face of uncertainty. Later, Ross et al. (2011) introduced DAgger, an imitation-learning algorithm that addresses distribution shift by iteratively collecting expert feedback along the learner’s own trajectories.

Hausknecht and Stone (2015) proposed the Deep Recurrent Q-Network (DRQN) to handle partially observable environments by maintaining a hidden state over time. These architectures are especially suited for environments requiring memory of past observations. Such recurrent architectures could enable agents to accumulate information about the goal's direction across multiple steps. Tamar et al. (2017) introduced Value Iteration Networks (VIN), neural networks that embed a differentiable planning module and learn to perform approximate value iteration. VINs have been applied to grid-world navigation and could provide a more principled way to combine local observations with implicit planning. More recent work such as Neural Map (Parisotto & Salakhutdinov, 2017) incorporates external memory to build an internal map, which is critical when the task requires exploration and recall of previously visited locations.

Highlighting the limitations of traditional model performance metrics in pathfinding problems, Codevilla et al. (2019) show that behavior-cloned policies with high action agreement can still crash because they lack planning and fail to recover from mistakes. They proposed new performance criteria such as path efficiency, collision rates and goal success rather than solely action prediction accuracy.

Methodology

3.1 World Generation and Sensors

The world is a 20×20 bounded square grid. A world generator populates the grid with single-cell obstacles based on a probability (obstacle_prob = 0.1) and adds several horizontal or vertical walls of random lengths. The goal and start points are randomly selected in such a way that the Euclidean distance between them is at least eight cells and that none of them is on an obstacle. Each simulation generates a new world, offering a diverse collection of layouts. The agent can move in eight directions corresponding to the Moore neighborhood (N, NE, E, SE, S, SW, W, NW).

For navigation, the agent was initially provided with just local information from eight radial distance sensors. Each sensor reports the number of empty tiles from the agent's position in one of eight directions to the closest wall or barrier. Following our incremental approach to improving learning performance, we introduced additional features in a stepwise manner.

We initially added the Euclidean distance to the target, providing the agent with a global measure of its distance from the target. This component provided useful context not available from the local sensors alone and helped to further inform movement choices.

We also added goal direction as the computed angle in radians between the agent and the goal to add more spatial information. The final raw state representation, with these enhancements, was (sensor_0 to sensor_7), the (distance_to_goal), and the (goal_direction), and the target variable was the action taken by A*, an integer from 0 to 7 representing one of the eight movements.

3.2 Data Generation Using A* Search Algorithm

A* search algorithm uses a priority queue to explore nodes with the lowest estimated total cost (the cost so far plus a heuristic). We use the Euclidean distance to the goal as the heuristic. When constructing the dataset, we run A* on each randomly generated world to compute an optimal path from the start to the goal. For every step along the path, we record the timestamp, run identifier, current position, the eight sensor readings, the Euclidean distance to the goal, the goal direction, and the action taken. Table 1 summarizes the data schema.

Table 1

Description of Dataset Variables

Variable	Description
timestamp	Global index across all runs
run_id	Simulation run identifier
position_x, position_y	Agent's coordinates (not used as features)
sensor_0…sensor_7	Distances to nearest obstacle in eight directions
distance_to_goal	Euclidean distance to goal
goal_direction	The calculated angle between the current position and the goal, in radians
action	Optimal move (0–7) as determined by A*

To support the model training, Table 2 summarizes the dataset variation, including sample size, instances, and key features.

Table 2

Dataset Variations Generated for Model Training

Dataset Name	Sample Size & Instances	Key Features
action_sensor+dist	3000 runs (35,452 labelled instances)	No goal_direction
action_sensor+dist_dir	3000 runs (35,452 labelled instances)	goal_direction as a feature
action_sensor+dist_dir_10k	10000 runs (118211 labelled instances)	goal_direction as a feature
action_sensor+dist_dir_10k_3walls	10000 runs (118211 labelled instances)	goal_direction as a feature with 3 walls

3.3 Evaluation Metrics and Performance Criteria

To evaluate and compare the models, we primarily used accuracy scores. Accuracy scores measure the percentage of correct predictions made by the model on a test dataset. It is a good first-line estimator of capabilities, although it does have limitations when the classes are imbalanced. However, since our EDA shows that our classes are well-balanced, accuracy is an appropriate metric. In addition, we used a simulator to observe the trained models navigating new, randomly generated worlds in real time.

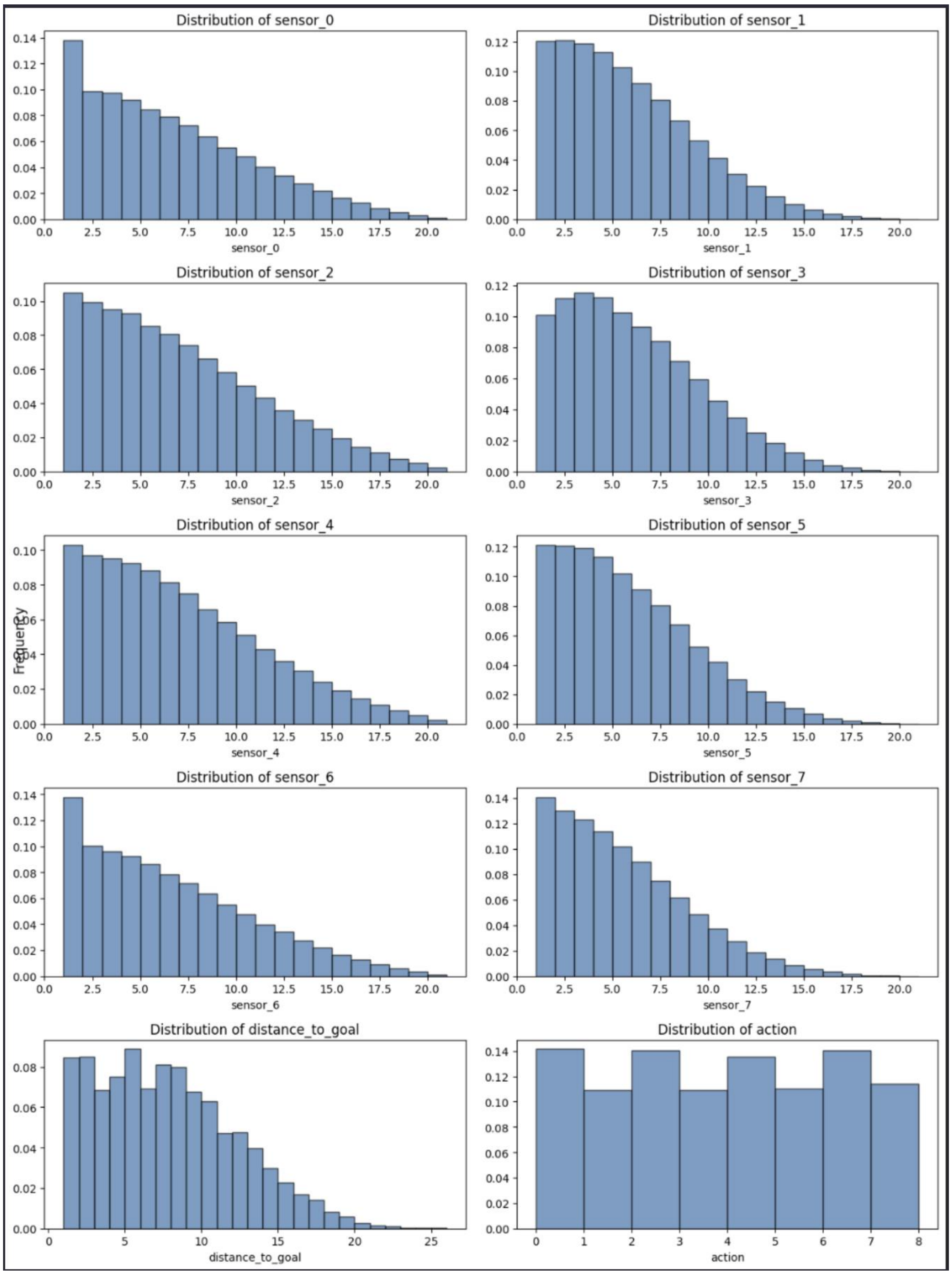
Exploratory Data Analysis

4.1 Distributions

We performed exploratory data analysis (EDA) on the dataset to understand feature distributions and potential issues.

Figure 1

Variable Distribution in the Dataset



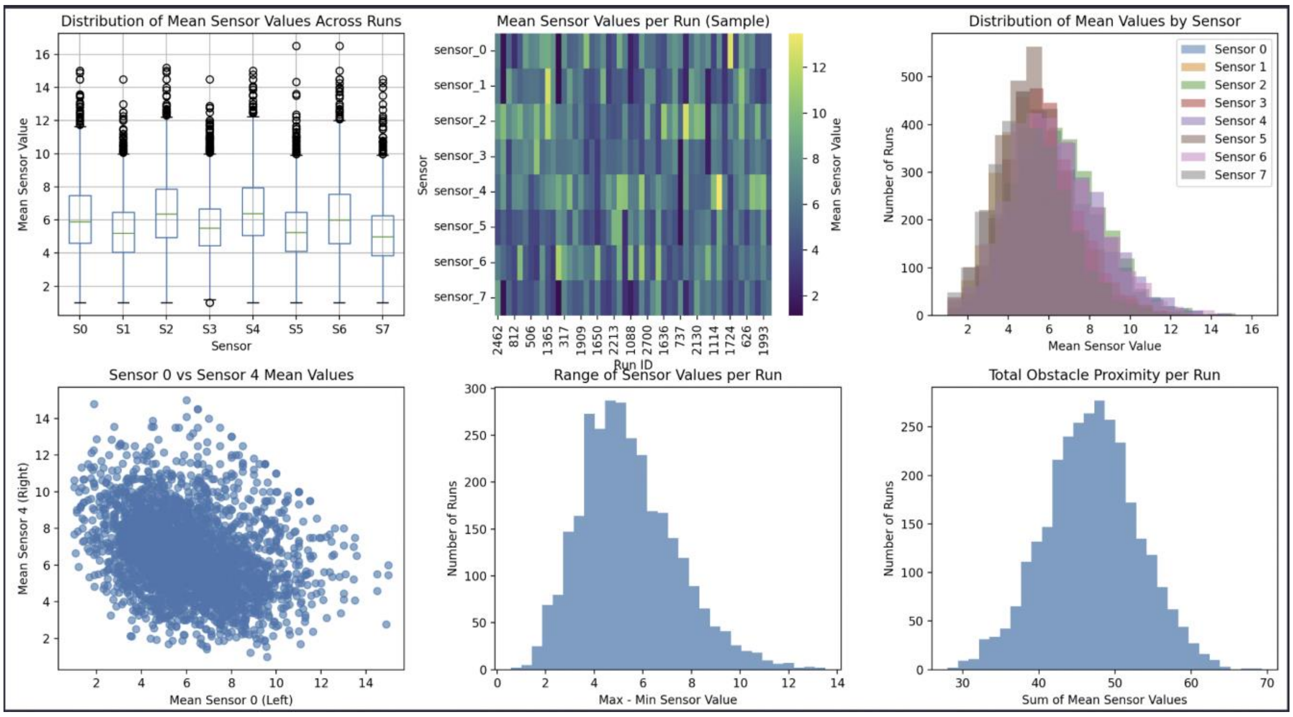
We observed that all sensor readings and distance to goal are right-skewed distributions, reflecting the prevalence of smaller distances and the diminishing frequency of larger distances in the readings. The distribution of the action variable is well balanced across all eight possible actions, with a slight convergence of the lateral/vertical actions over the diagonal ones. The balanced action distribution reduces class imbalance issues during training.

4.2 Run Level Variability

In Figure 2, we conducted a comprehensive run variability study to ensure that world configurations are evenly represented across runs.

Figure 2

Summary Statistics and Distributions of Mean Sensor Values Across Runs



The absence of distribution skew across sensors shows a good representation of world configurations across runs.

4.3 Shifting-Signals Problem and Information Asymmetry

An important observation from EDA is that identical sensor readings can correspond to different optimal actions depending on the global arrangement of obstacles and the relative position of the goal. For example, two states with the same local obstacles may require moving northeast in one case and southwest in another if the goal lies in different directions. A* can resolve this because it has global knowledge of the grid, but the machine-learning model sees only local distances and a scalar goal distance. This information asymmetry leads to shifting signals: identical inputs with opposite labels.

To quantify this phenomenon, we grouped training samples by sensor value patterns and counted the percentage of unique patterns that had conflicting action labels with and without goal direction. We found that 31.7% of the patterns had conflicts without goal direction, whereas only 8.8% had conflicts with goal direction.

To help visualize the problem further, we performed a full-fledged feature uniqueness analysis. Figure 3 shows that in the absence of goal direction information, the model relies on an arbitrary feature (sensor 6) as the discriminant feature and is unable to clearly discriminate between actions. Figure 4 shows that once the goal direction is added, the actions are clearly differentiated, and the goal direction becomes the dominating discriminant.

Figure 3

Without Goal Direction:

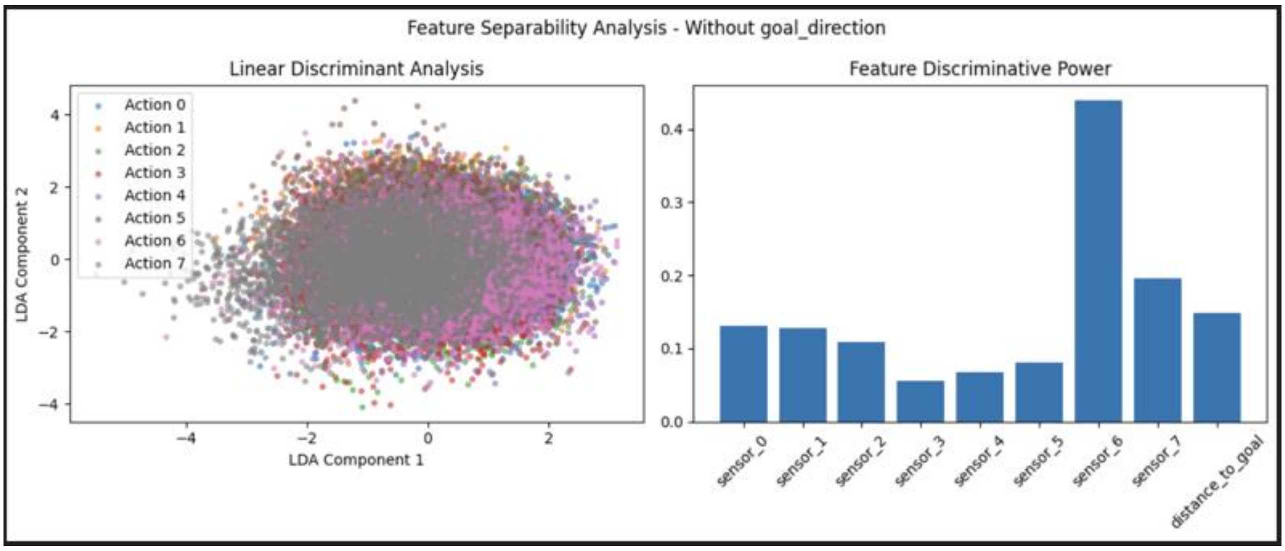
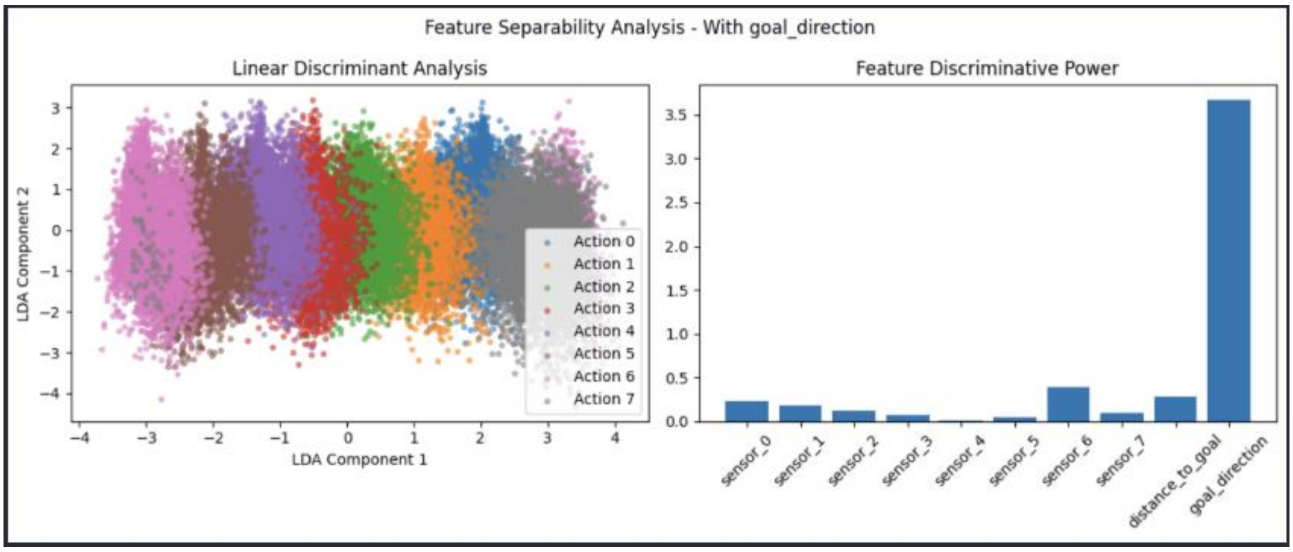


Figure 4

With Goal Direction:



Model Selection and Training

5.1 Model Comparison Overview

We compared a selection of classification models in scikit-learn and the XGBoost library. All models were trained to output the optimal action. We performed stratified 80/20 train-tests, so each action class was present in each subset approximately in proportion. The models and their training parameters were as follows:

1. **Random Forest.** Chosen due to its stability and ability to capture non-linear interactions between features without extensive preprocessing. Its ensemble nature provides it with a solid baseline for tabular data.
2. **XGBoost.** Included due to its success in structured data competitions. It can handle non-linear relationships and can produce calibrated class probabilities using the multi-class log-loss objective.
3. **Logistic Regression.** A simple and interpretable linear model is used as a baseline for multi-class classification. We used the multinomial variant with the LBFGS solver and boosted training (max_iter = 1000) to ensure convergence.
4. **Support Vector Machine (SVM).** With an RBF kernel, SVMs can generalize hard decision boundaries. SVMs are useful when the decision boundary between classes is smooth, yet

non-linear.

5. **Naïve Bayes.** Chosen for its performance and simplicity, although it makes strong independence assumptions. It can be used as a baseline against which to compare.
6. **K-nearest neighbors (KNN).** A non-parametric method with no distributional assumptions about the data. It is good at learning local structure but will struggle with noisy or high-dimensional data.
7. **Neural Network (MLP).** A multi-layer perceptron with multiple layers of hidden units using ReLU activations. This architecture has sufficient capacity to learn complex non-linear patterns in the sensor signals.

5.2 Choosing Appropriate Algorithm Structures

Selecting the right structure for the various AI-based models, e.g., the architecture for the neural network is a crucial part of our approach. Network structure like the number of layers, the units per layer, the activation functions, and regularization techniques plays a tremendous role in the ability of the model to learn from the input features and generalize to novel environments.

We experimented with architecture choice starting with simple fully connected (feed-forward) networks and then tuned based on performance. Shallow nets would under-fit the problem, especially after we included more nuanced features like (distance_to_goal). Increasingly deeper architecture provided the possibility of representing more complex relationships between inputs and optimal actions, but potentially at the expense of over-fitting.

Results

Table 3 shows each model's accuracy on the dataset versions for the held-out test set. The addition of goal direction to the distance features gives significant performance boosts to all the models. Comparing, for example, the (goal_dist_3k) dataset with the (goal_dist+goal_dir_3k) dataset, Random Forest accuracy goes from 0.379 to 0.891, while Naïve Bayes goes from 0.233 to 0.787.

Table 3

Model Accuracy Comparison Across Feature Sets

Model	Dist_3k	Dist+Dir_3k	Dist+Dir_10k	Dist+Dir+3W_10k
Random Forest	0.379	**0.891**	**0.889**	0.848
XGBoost	**0.436**	0.885	**0.889**	**0.849**
Support Vector Machine	0.343	0.844	0.859	0.812
Neural Network (MLP)	0.347	0.867	0.881	0.838
Logistic Regression	0.229	0.602	0.592	0.548
Naïve Bayes	0.233	0.787	0.781	0.679
K-Nearest Neighbors	0.279	0.588	0.626	0.622

Note. Dist = Goal Distance, Dir = Goal Direction, 3W = Three Walls. Bold values indicate the highest accuracy for each column.

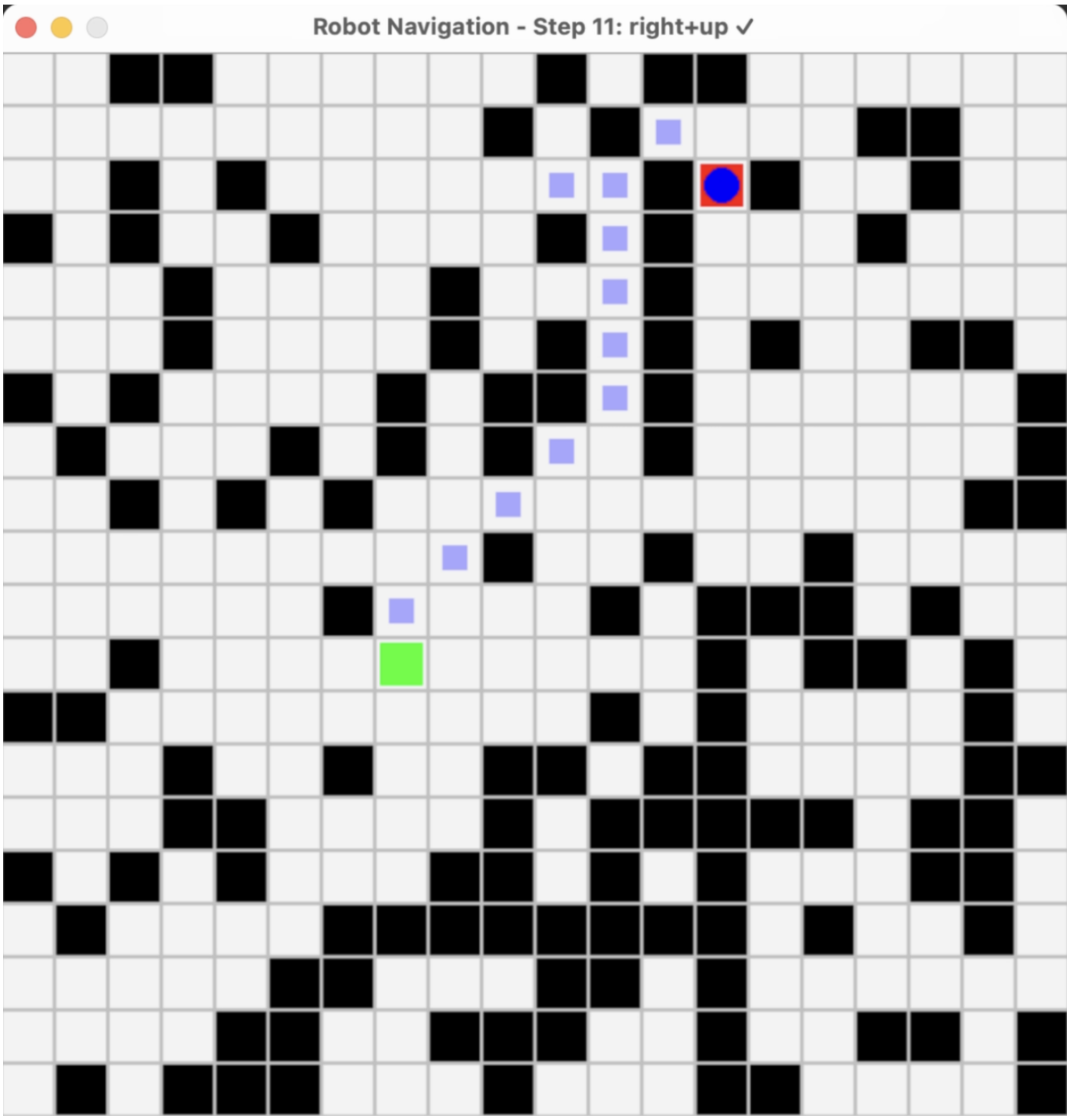
Among models compared, Random Forest and XGBoost are the most consistently top-performing models across dataset configurations, with a high performance in larger datasets, such as both achieving 0.889 accuracy on (dist+goal_dir_10k). The Neural Network (MLP) also performs strongly, with 0.881 and 0.838 on the 10k datasets. Support Vector Machine is also assisted by the extra directional features, but tends to fall behind ensemble methods, with the best accuracy of 0.859.

Logistic Regression and K-Nearest Neighbors lag consistently behind, especially as the datasets become more complex. Increasing the size of the dataset from 3k to 10k samples gives moderate accuracy gains to most models—for instance, the Neural Network goes from 0.867 to 0.881, and the SVM from 0.844 to 0.859. But introducing walls into the worlds (dist+goal_dir_3walls_10k) reduces accuracy for most models by some extent, due to greater complexity in navigating around walls, e.g., Random Forest accuracy drops from 0.889 to 0.848, and that of XGBoost from 0.889 to 0.849.

We were able to visualize the decision-making abilities of the various models by loading the model's prediction function into the simulator and presenting it with new, randomly generated worlds. The green square represents the starting point, the red square represents the goal, the solid blue circle is the agent, and the lighter blue squares represent the successive positions of the agent as it navigates.

Figure 5

Successful Robot Navigation

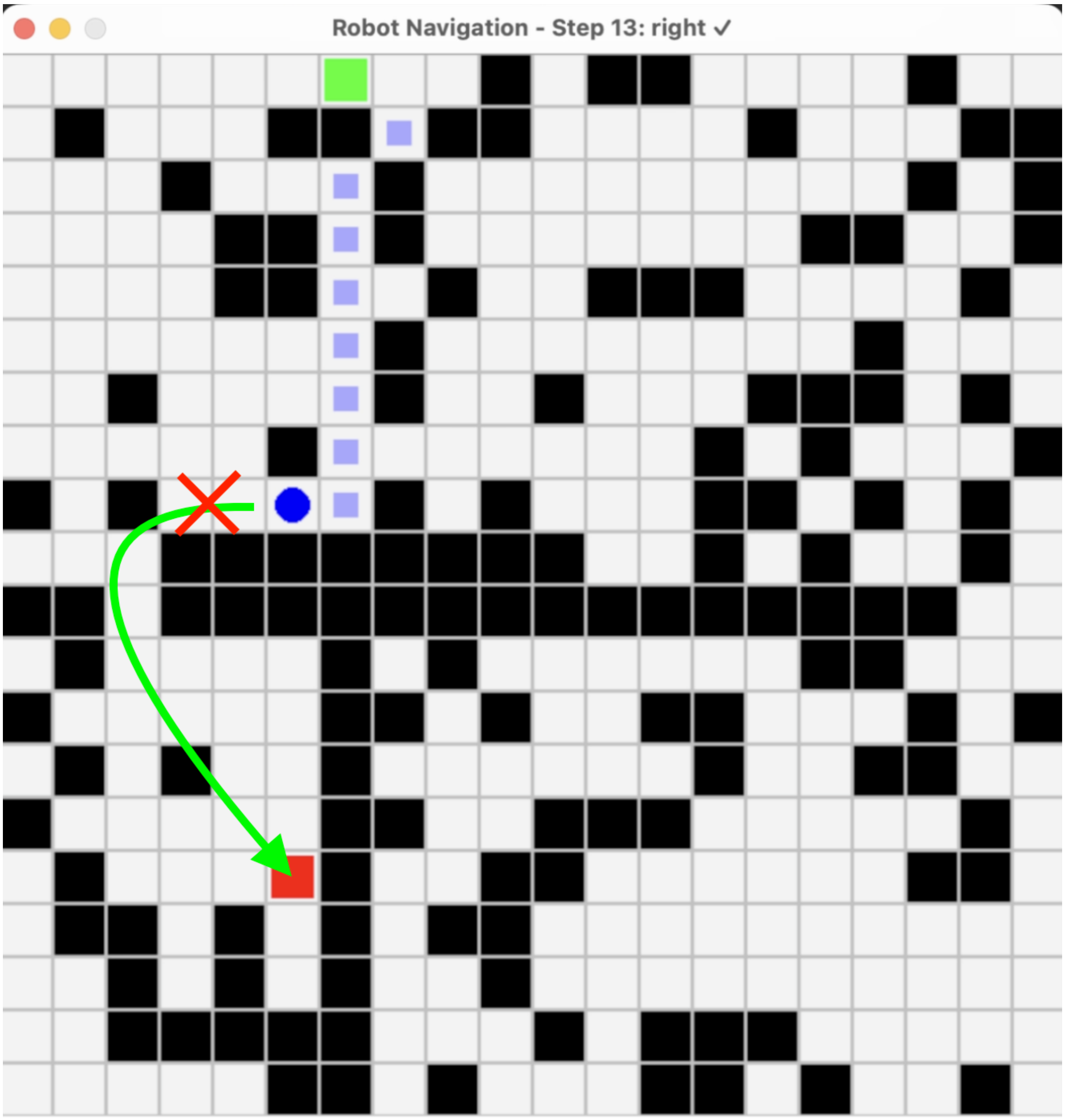


The models demonstrated generalization to novel maze configurations, effectively avoiding illegal moves (e.g., walking into walls) and progressing toward goals. This suggests that the models have learned underlying spatial heuristics rather than memorizing training trajectories.

It is interesting to note that even though the trained models were able to emulate A* behavior when the solution did not involve bypassing large obstacles, their ability to navigate around obstacles broke down when the path involved moving more than 3-4 tiles away from the goal direction.

Figure 6

Unsuccessful Robot Navigation



6.1 Selecting Neural Network Architecture

To observe how neural network architecture influences performance on our dataset, we experimented with several multi-layer perceptron (MLP) setups by varying both the number of layers (2 vs. 3) and the number of neurons per layer (16 to 256). We chose the more difficult dataset of`(dist+goal_dir_3walls_10k) to see if we could extract any further performance. As shown in Table 4, we observed that 2-layer MLPs perform better than deeper 3-layer ones, with the highest performance (0.846 accuracy) achieved by the [64, 32] setup. Wider architectures like [128, 64] and [256, 128] performed worse, possibly due to over-fitting or less generalization capacity for the dataset size.

Table 4

Comparison of MLP Architectures and Accuracy

Architecture	**Layer Sizes**	**Accuracy**
2-layer MLP	[16, 32]	0.839
2-layer MLP	[32, 16]	0.840
2-layer MLP	[64, 32]	**0.846**
2-layer MLP	[32, 64]	0.844
2-layer MLP	[128, 64]	0.835
2-layer MLP	[256, 128]	0.808
3-layer MLP	[64, 32, 16]	0.842
3-layer MLP	[16, 32, 64]	0.840
3-layer MLP	[128, 64, 32]	0.828
3-layer MLP	[256, 128, 64]	0.804
3-layer MLP	[128, 128, 128]	0.800
3-layer MLP	[256, 256, 256]	0.804

Note. Bold values indicate the highest accuracy for each column.

On the other hand, the 3-layer MLPs did not perform better than their 2-layer versions and, in most cases, had slightly worse accuracy. The best-performing 3-layer model, [64, 32, 16], had

0.842 accuracy, which is close but still worse than the optimal 2-layer score. Wider and deeper 3-layer models like [128, 128, 128] and [256, 256, 256] had even worse accuracies (0.800–0.804), showing that increased depth and width do not necessarily lead to increased performance and may also lead to optimization problems or overfitting.

In addition, funnel-shaped architectures (i.e., decreasing width between layers) performed slightly better than flat or expanding ones. For example, [64, 32, 16] performed better than [16, 32, 64] despite having the same total number of neurons. This suggests that it could be helpful to slowly reduce the representational capacity as the network approaches the output to force it to focus and generalize more.

Overall, these findings indicate that shallower networks with fewer layers and medium width are more suitable for this dataset. Depth and width must both be carefully tuned, especially when there is limited training data or minimal regularization.

Discussion

The results show that when the feature of goal direction is introduced a substantial performance gain is seen across all models. The models that saw the highest and most accurate results were the Random Forest and XGBoost models, with both achieving 0.889 on the (dist+goal_dir_10k) dataset. This is significant as the RF and XGBoost models are more interpretable than other models and this also shows their robustness to moderately elevated environmental complexity. The Neural Network model also achieved a high accuracy across varying configurations (comparable to the ensemble models) and world complexities.

These results highlight that richer spatial information or features improve the agent's ability to choose an action that closely follows the optimal actions. In addition, we do see an expected decrease in performance for the (dist+goal_dir_3walls_10k) experiment which introduced lengthy walls into the world indicating some deficiencies in bigger obstacle avoidance. There is an opportunity to explore more sophisticated models and representations, such as the application of learned spatial embeddings, memory-based networks, noise-robust sensor fusion, and, in parallel, the addition of additional sensors to expand the observability field and engineer more features to encode relevant spatial and temporal information. The addition of these features in the future could potentially allow models to generalize more abstractly about barrier walls and overcome the limited sensor information.

Future Work

Within the limitations we have imposed, our existing strategy has placed the accuracy of supervised AI models to near the limits for this problem. Future research must entail making the environment more complex and introducing more advanced learning paradigms that can handle planning, partial observability, and stochastic or noisy sensor readings. On the environmental front, this entails making the worlds more complex, with increasing obstacles, varied layouts, and harder navigation constraints, to more accurately mirror real-world environments. Increasing the number of sensors and ability to provide higher spatial and temporal information, e.g., relative goal angles, local occupancy patterns, or spatial embeddings induced thereof, might also reduce label ambiguity and perform better. On the modeling front, future work needs to investigate more spatial and temporal reasoning-friendly architectures.

Convolutional Neural Networks (CNNs) will be able to leverage local spatial patterns when sensor data is represented as a grid, but Recurrent Neural Networks (RNNs) or LSTMs will be able to accumulate observations over time so that they can maintain an implicit belief state to support improved decision-making in partially observable Markov decision processes (POMDPs). Graph Neural Networks (GNNs) offer a form of reasoning on topological structure, i.e., grid with obstacle connectivity. Apart from imitation learning, reinforcement learning algorithms, i.e., Q-learning, Deep Q-Networks (DQN), Proximal Policy Optimization (PPO), and even hierarchical planning systems like those used in AlphaStar (Mathieu et al., 2023), would allow agents to learn adaptive policies from reward signals end-to-end, enabling real obstacle avoidance and goal-directed planning rather than simply imitating A* outputs.

Collectively, said techniques comprise engineering, more informative sensors, temporal modeling, and reinforcement learning, and outline a direction to planners, adapters, and navigators that work well in increasingly more complex and noisy worlds.

Conclusion

This work investigated the impact that the feature design and model selection can have on learning navigation policies from local sensor measurements, in particular, how information asymmetry and partial observability can cause conflicting action labels for otherwise identical sensors. Through running extensive experiments, we demonstrated that including distance features with goal direction significantly enhances performance across all tested models, where ensemble methods such as Random Forest and XGBoost achieved the most robust and highest accuracies. Despite that, adding more challenging obstacle configurations revealed weaknesses in all models, most significantly under situations where local sensing cannot perceive the complete navigational context.

These findings point to two key findings: one, that carefully designed features can dramatically minimize label ambiguity; and two, that conventional supervised approaches remain inherently bound in environments requiring long-horizon planning or reasoning over hidden state. Breaking such constraints will require transcending fixed feature representations to more observant sensing, temporal abstraction, and adaptive decision-making methods such as reinforcement learning and recurrent models. By combining these techniques with the proposed feature engineering and denser sensor configurations, future-generation agents could offer more robust, generalized navigation capability against sensor noise, environmental complexity, and partial observability.

Acknowledgments

The authors acknowledge the use of AI (ChatGPT, Cursor) to assist in code experimentation, brainstorming, drafting and proof-reading.

References

Codevilla, F., Santana, E., López, A. M., & Gaidon, A. (2019). Exploring the limitations of behavior cloning for autonomous driving. *arXiv*. https://arxiv.org/abs/1904.08980

Hausknecht, M. J., & Stone, P. (2015). Deep recurrent Q-learning for partially observable MDPs. *arXiv*. http://arxiv.org/abs/1507.06527

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence, 101*(1-2), 99–134. https://doi.org/10.1016/S0004-3702(98)00023-X

Mathieu, M., Ozair, S., Srinivasan, S., Gulcehre, C., Zhang, S., Jiang, R., Le Paine, T., Powell, R., Żołna, K., Schrittwieser, J., Choi, D., Georgiev, P., Toyama, D., Huang, A., Ring, R., Babuschkin, I., Ewalds, T., Bordbar, M., Henderson, S., … Vinyals, O. (2023). AlphaStar unplugged: Large-scale offline reinforcement learning. *arXiv*. https://arxiv.org/abs/2308.03526

Parisotto, E., & Salakhutdinov, R. (2017). Neural map: Structured memory for deep reinforcement learning. *arXiv*. https://arxiv.org/abs/1702.08360

Pomerleau, D. A. (1989). ALVINN: An autonomous land vehicle in a neural network. In D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems* (Vol. 1). https://proceedings.neurips.cc/paper_files/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf

Ross, S., Gordon, G. J., & Bagnell, J. A. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. *arXiv*. https://arxiv.org/abs/1011.0686

Tamar, A., Wu, Y., Thomas, G., Levine, S., & Abbeel, P. (2017). Value iteration networks. *arXiv*. https://arxiv.org/abs/1602.02867

GitHub

https://github.com/MaximeBoulat/AAI501-TeamProject

Appendix

Team contributions

Maxime Boulat:

- Coding of simulator to visualize agent progress, coding of unified pipeline for training different models, conducting experiments with different data configurations.
- Research of relevant literature
- Wrote early draft of paper, combining research into the topic of shifting signals, methodology and model results + interpretation.
- Collaborated with teammates to rewrite the code in one place.
- Conducted EDA research on the subject of feature separability + variable distributions.
- Contributed to writing of final paper, wrote the designated slides in presentation (methodology + results).

Dylan Scott-Dawkins:

- Coding of initial robot_2.py code to run a_star and generate the training data set, also ran experiments around neural network architectures and q_learning experiments.
- Collaborated with teammates to rewrite the code in one place incorporating all teammates code.
- Researched and identified new goal_direction and goal_distance features to improve models performance across the board.
- Contributed to writing of final paper, wrote results, discussion and future work sections.
- Wrote slides for presentation for the neural network deep dive + evolution results and discussion and future work and conclusion.

Tanya Neustice:

- Coding experiments with different map configurations.
- Trained different models and results analysis.
- Conducted EDA.
- Made initial draft of paper.

- Made presentation initial draft.

- Prepare designated slides in presentation (Intro + EDA)

- Made final clean version of paper and ensured APA conformance.

- Made final version of presentation (cleanup and theming).