Generalizable Deep Q-Learning for Navigation in Unknown Obstacle Fields

Alvin James Bacani
Electrical and Systems Engineering
Washington University in St. Louis
St. Louis, Missouri
a.bacani@wustl.edu

Gabe Herman
Electrical and Systems Engineering
Washington University in St. Louis
St. Louis, Missouri
g.m.herman@wustl.edu

***Abstract*—In this project, we develop a Deep Q-Learning framework for robotic navigation in continuous 2D environments with uncertain and variable obstacle configurations. The robot is modeled as a stochastic system with state-dependent disturbances and a discrete action space composed of linear and angular velocity pairs. We train policies across multiple randomized environments to enable generalization to unseen test settings. The learned policy is evaluated on success rate and time-to-goal across several novel environments, demonstrating robust navigation behavior and effective obstacle avoidance despite the lack of prior knowledge of obstacle placement.**

I. Introduction

This project investigates deep reinforcement learning for goal-directed robotic navigation in continuous, partially unknown environments. The robot is modeled as a discrete-time stochastic system with state-dependent disturbances, where the state consists of its 2D position and orientation, and the control input comprises linear and angular velocities. The environment contains cylindrical obstacles, and the robot must reach a goal location from a fixed starting state while avoiding collisions.

Control is applied through a fixed, discrete action space, and the system dynamics are unknown—interaction occurs via a black-box simulator that returns the next state given a current state and action. Reinforcement learning is therefore used to learn a control policy based on experience, with Deep Q-Learning (DQL) as the primary algorithm.

The project is structured into three progressively challenging problems:

- **Problem 1:** The robot operates in a fixed environment with known obstacle locations. The goal is to learn a policy that consistently drives the robot from a fixed start to the goal while avoiding all collisions.
- **Problem 2:** The robot is trained across multiple environments, each with different obstacle placements and radii. The objective is to learn a policy that generalizes to novel, unseen environments at test time without retraining.
- **Problem 3:** Both the obstacle configurations and goal locations are randomized at test time. The policy must learn to navigate safely and efficiently toward varying goal regions, requiring generalization over both environment structure and task objective.

These three problem levels provide a structured evaluation of the robot's ability to learn and generalize robust navigation policies in the face of uncertainty and dynamic constraints.

II. Planning Algorithm

A. Deep Q-Network (DQN)

Deep Q-Networks (DQN) combine Q-learning with deep neural networks to approximate the action-value function over continuous state spaces. In this project, we use a DQN to learn a navigation policy that maps robot states to discretized velocity commands. The agent interacts with the environment by selecting actions that maximize its estimated Q-values, updating its network parameters via temporal-difference (TD) learning and experience replay.

The training curve in **Figure 1** shows the return per episode over the course of 500 training episodes. Initially, the return improves rapidly, indicating that the agent is successfully learning to avoid obstacles and approach the goal. However, after approximately 200 episodes, the performance exhibits significant variance and degradation. This instability is characteristic of DQN in stochastic environments, where the Q-function can overestimate returns or fail to converge if exploration, learning rate, or replay buffer parameters are not well tuned.

With this approach, the agent did successfully reach the goal, but even after extensive hyperparameter tuning, convergence success rates were limited to between 50% and 60%.

B. Double Deep Q-Network (DDQN)

To address the overestimation issues of DQN, we explored Double Deep Q-Learning (DDQN), which decouples action selection and evaluation by using two separate networks. While DQN tends to overestimate Q-values due to using the same network for both selecting and evaluating actions, DDQN mitigates this bias by using the target network to evaluate the next state's Q-value while the main network selects the action.

Figure 2 shows the training return per episode over 300 episodes. Compared to standard DQN, DDQN demonstrates significantly more stable learning. Early episodes show high variance in return as the agent explores, but the policy improves rapidly, with returns increasing and stabilizing around episode 100. From that point forward, the agent consistently

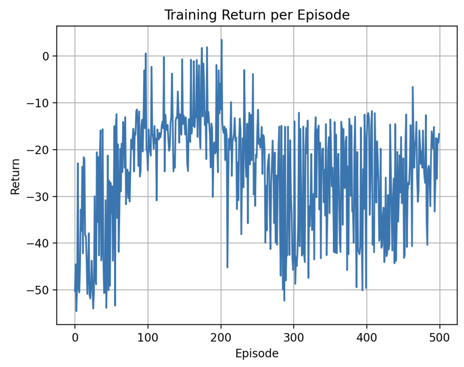

Fig. 1: Training return per episode using Deep Q-Network (DQN).

achieves near-optimal return with reduced noise, indicating convergence to a robust navigation policy.

These results confirm that DDQN provides improved stability and learning performance, particularly in noisy or highly variable environments, by reducing overoptimistic Q-value estimates.

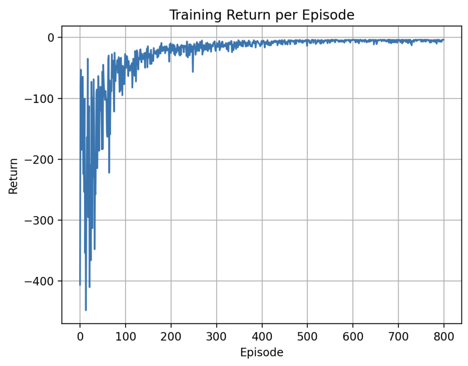

Fig. 2: Training return per episode using Double Deep Q-Network (DDQN).

III. Implementation

A. Network

The reinforcement learning agent was implemented using PyTorch. The Q-network used in both DQN and DDQN consists of a fully connected feedforward neural network. The input layer takes a 7-dimensional feature vector, and the output layer produces Q-values for each of the 23 discrete actions in the action space.

The network architecture includes two hidden layers with 128 and 64 units respectively. Each hidden layer uses the ReLU (Rectified Linear Unit) activation function to introduce non-linearity. The final output layer is linear, as it directly predicts Q-values without further transformation.

A range of different network structures were tested and the hidden layer sizes of 128 and 64 units were eventually settled on to balance representational capacity and computational efficiency. This architecture provides sufficient non-linear approximation power to learn the Q-function over a low-dimensional state space (x, y, θ), while remaining compact enough to avoid overfitting and enable stable learning. The progressive reduction in layer size encourages hierarchical feature extraction and supports generalization across environments with varying dynamics and obstacle configurations. This structure is commonly used in similar reinforcement learning tasks and performed effectively for this navigation problem without requiring extensive tuning.

The network is trained using the Adam optimizer with a learning rate of 1×10^{-3}. The loss function is the mean squared error (MSE) between the predicted Q-value and the target Q-value. A fixed target network is updated periodically to stabilize training, and a replay buffer is used to break correlation between samples and improve recollection.

This configuration was selected to balance performance and training stability, and was consistent across both DQN and DDQN implementations.

B. Reward Function and Feature Vector

The reward function is designed to balance safety, efficiency, and goal completion. It combines sparse terminal rewards with dense shaping components to guide learning:

- **Goal Reward:** The agent receives a reward of $+10$ for reaching the goal region, defined as being within 0.08 units of the goal location.
- **Collision Penalty:** A penalty of -10 is given if the agent collides with any obstacle, determined by checking whether the agent's distance to an obstacle center is less than or equal to the obstacle's radius.
- **Time Penalty:** A constant step penalty of -0.01 discourages prolonged episodes and incentivizes faster trajectories.
- **Distance to Goal Penalty:** A shaping term of $-0.5 \cdot$ dist_to_goal penalizes distance to the goal, encouraging the agent to reduce it.
- **Proximity Penalty to Obstacles:** If the agent is within 0.3 units of the nearest obstacle, it receives a penalty proportional to $0.3 -$ min_dist_to_obs. This provides a smooth gradient discouraging the agent from getting too close to obstacles.

The feature vector passed to the Q-network contains seven elements:

- x, y, ϕ: the agent's global position and heading,
- **Distance to Goal:** Euclidean distance from the agent to the goal,
- **Relative Angle to Goal:** Angular difference between the robot's orientation and the line to the goal,
- **Minimum Distance to Obstacle:** Closest distance from the agent to any obstacle surface.
- **Relative Angle to Nearest Obstacle:** Angular difference between the robot's orientation and the line to the nearest obstacle

This reward design encourages timely, goal-directed behavior while penalizing risky navigation near obstacles. Combined with geometric features, it helps the agent learn robust, collision-free trajectories in complex environments.

C. *Hyperparameters*

Several hyperparameters were chosen to balance learning stability, convergence speed, and generalization performance. Most values were selected based on empirical tuning and reinforcement learning best practices.

The discount factor $\gamma = 0.99$ encourages long-term planning, while the learning rate was set to 1×10^{-3} for stable convergence using the Adam optimizer. A relatively large batch size of 256 improves the stability of gradient estimates during training.

Exploration was handled via an ε-greedy strategy, starting at $\varepsilon = 1.0$ and decaying multiplicatively by 0.995 per episode down to a minimum of 0.05. Target networks were updated every 20 episodes to stabilize Q-learning. Training was conducted over 800 episodes with a maximum of 300 time steps per training episode.

Hyperparameter	**Value**
Discount factor γ	0.99
Learning rate (LR)	1×10^{-3}
Batch size	256
Exploration start $\varepsilon_{\text{start}}$	1.0
Exploration end ε_{end}	0.05
Exploration decay $\varepsilon_{\text{decay}}$	0.995
Target network update frequency	20 episodes
Replay buffer size	25,000
Number of episodes	800
Max steps per episode	300

TABLE I: Training Hyperparameters

These values were consistent across both DQN and DDQN experiments to ensure fair comparisons and isolate the effect of algorithmic changes.

IV. Extension to Unknown Environments

Problem 2 tasked us with learning a robot policy to navigate unknown environments, i.e. environments with possibly different obstacle configurations. Unlike Problem 1, where the agent could memorize obstacle locations, this setting required the learned policy to generalize across multiple randomized environments.

To address this, we trained the agent on a diverse set of obstacle configurations generated procedurally during training. Each episode involved a new environment sampled from a uniform distribution in the state space, exposing the agent to a wide range of layouts. This forced the policy to learn geometric and relative features-such as distances and angles to obstacles and goals-rather than memorizing fixed maps.

The same Deep Q-Learning framework was used, with the reward function and vector unchanged. Generalization was evaluated by testing the trained policy in previously unseen environments. Success was defined as reaching the goal within the time limit without collisions.

V. Results

Despite increased variability, the agent learned to navigate around novel arrangements effectively, achieving a consistent success rate across test environments. The structured reward shaping and rich feature representation contributed significantly to this generalization capability. The varying of obstacle configurations throughout training meant that convergence was slower, and return variance noticeably higher, as shown in Figure 3.

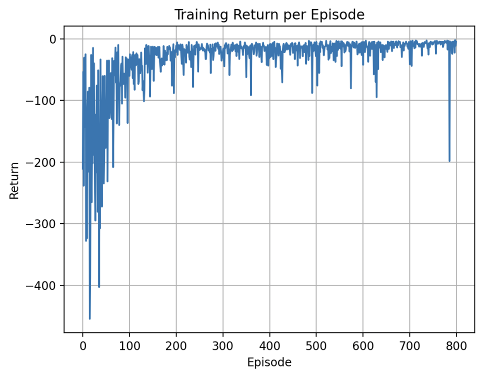

Fig. 3: Training returns under randomly varied environments

The new policy was tested on three predetermined testing environments, and then evaluated on a validation environment that was made available later in development. The results over 100 trials are shown in Table II.

Problem	**Success %**	**Avg. Steps to Goal**
Problem 1	100%	20.0
Problem 2 (Test Environments)	100%	26.0
Problem 2 (Validation Environment)	99%	27.2

TABLE II: Performance across different evaluation conditions.

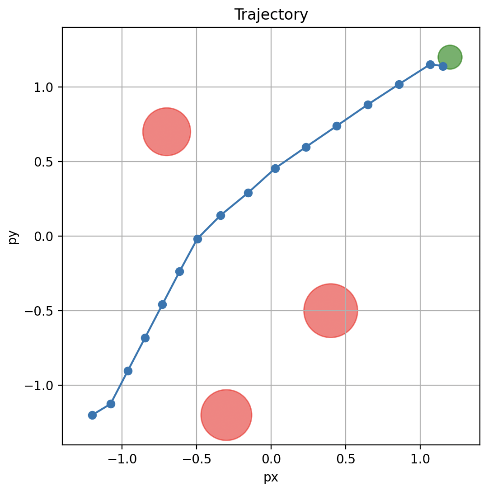

Fig. 4: Robot navigation in Test Environment 1

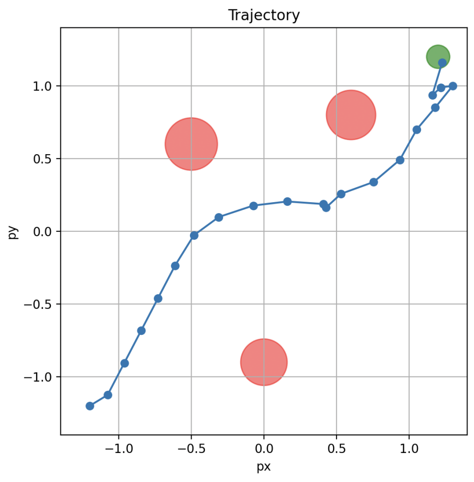

Fig. 5: Robot navigation in Test Environment 2

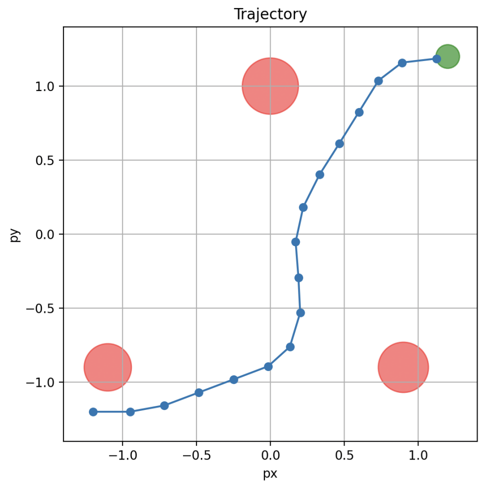

Fig. 6: Robot navigation in Test Environment 3

Figures 4 – 7 show example trials of the robot navigating each of the three test environments and the validation environment respectively. **Figure 6** shows that Test environment 3 in particular posed a challenge, due to the close proximity of one obstacle to the starting state, but consistent success was achieved after careful tuning of the learning rate, network size, batch size, and obstacle proximity penalty.

This extension demonstrated the robustness of deep reinforcement learning when trained with sufficient environmental diversity and emphasizes the importance of informative input features in generalizing memorized scenarios.

VI. Conclusion

The results of this project demonstrate that deep reinforcement learning can effectively produce robust navigation policies in environments with uncertain obstacle configurations. Through structured reward shaping and a feature-rich state representation, the agent was able to generalize its behavior beyond memorized settings.

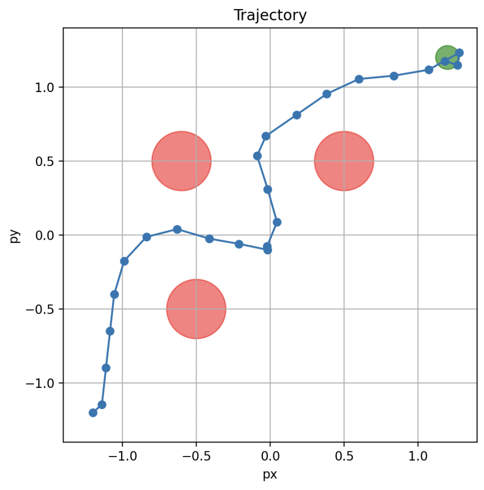

Fig. 7: Robot navigation in Validation Environment

In Problem 2, despite increased variability across training episodes, the learned policy maintained a 100% success rate across three unseen test environments and achieved 99% success in a validation environment. While convergence was slower and training returns showed higher variance compared to the fixed environment of Problem 1, the final policy remained stable and effective. The average steps to goal increased slightly—from 20 steps in Problem 1 to around 26–27 in Problem 2—indicating a modest trade-off for generalization.

The visualizations in **Figures 4 – 7** confirm the agent's ability to adapt to new obstacle layouts, including environments with narrow passages or early obstacle encounters. These outcomes validate the learning framework and highlight the role of training diversity and careful hyperparameter tuning in achieving reliable performance.

This work reinforces the potential of deep Q-learning-based approaches for motion planning in complex, partially unknown domains.

Work Distribution

Work has been equally divided between the two members.

- **Alvin** – 50%: Focused on fine-tuning the codebase and contributing to the final report.
- **Gabe** – 50%: Developed the initial scripts and co-authored the report.

References

[1] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations (ICLR)*, San Diego, CA, USA, May 2015.

[2] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 30, no. 1, 2016.