Near Optimal Path Learning from Rule-Based Algorithms

Jonathan Dumanski (jdumansk@stanford.edu), Mark Robinson (mrobin10@stanford.edu) , Manjari Talasila (mtalasil@stanford.edu)

Overview

<u>Background</u>: **Real-time path planning is computationally expensive** and **collision-prone s**ince rule-based algorithms like A* require paths to be recalculated in real-time and cannot predict dynamic obstacle motion.
<u>Solution</u>: We developed **three learning-based planners**: a reinforcement learning agent, a CNN cost-to-go predictor, and a hybrid self-learning model, **generating faster, collision-free paths.**
<u>Inputs</u>: **Randomly generated 2D grid** (32 - 128 units), the **goal position**, **static obstacle positions**, and **dynamic obstacle positions** at times t, t-1, and for the CNN only, t-2 as well.
<u>Outputs</u>: Agent **actions at each timestep** - one of eight possible directions: up, down, left, right, or diagonals.
<u>Results</u>: All three learning-based planners achieved **significantly higher success rates** than dynamic A*, with the RL agent **reducing computing time by 36x** on average, at a **small cost to path optimality** compared to the CNN and baseline dynamic A*.

Data

Data Generation:
- Generated training and test episodes each containing randomly generated grid dimensions and a deterministic seed that controlled the 2D grid environment for each episode.

2D Grid Environment:
- Initialized random start, goal and static obstacle positions at t=0.
- Controlled dynamic obstacle movement at each timestep according to probability distribution:
    - 0.7 to continue in the previous direction, 0.1 to adjacent moves, and 0.02 to the other five possible actions.

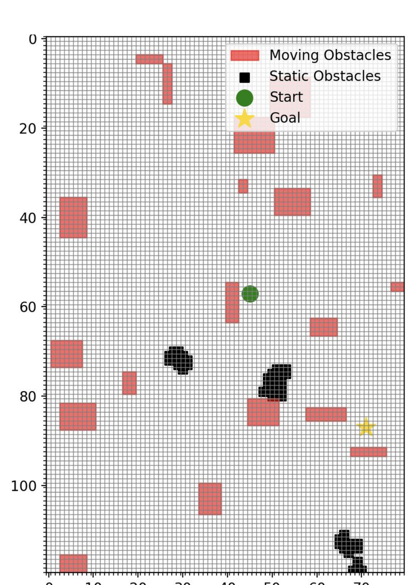

Figure 1. Random 80 x 120 grid at t=0

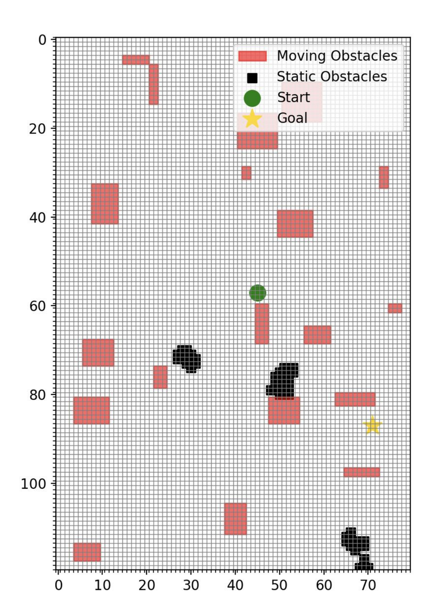

Figure 2. Random 80 x 120 grid at t=5

Features

Reinforcement Learning: Only derived features were used: a 7×7 local patch extracted from a 17×17 local field, where static obstacles are encoded as negative values, dynamic obstacles at t and t+1, inferred/predicted from velocity from t-1 to t, are encoded as negative Gaussians centered at each obstacle, and the goal as a positive Gaussian, providing spatial and temporal context suitable for learning collision-free paths.
CNN: Feature inputs consisted of the 64x64 patches of the current and past two occupancy grids (for inferring object dynamics), as well as a vector field encoding distance to the goal, centered about the agent's position.
Hybrid: 'Expert' paths were generated from dynamic A*, with moves valued according to directness and distance to goal. A 21x21 local window is used with 4 layers: the agent and goal position, and the current and previous obstacle positions.

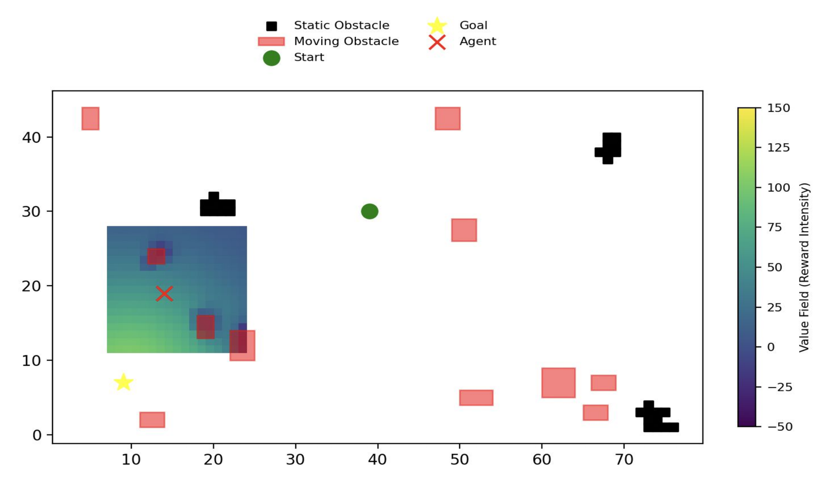

Figure 3. RL value field at a timestep during a training episode

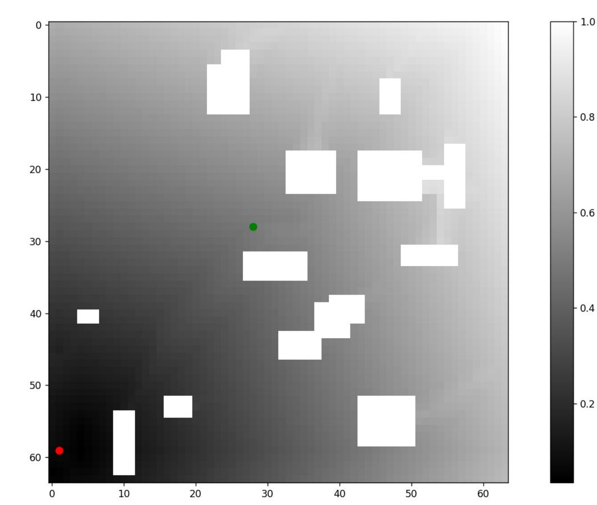
Figure 4. CNN cost-to-go map ground truth training example

Models

Reinforcement Learning:
- We used an Actor-Critic approach with separate policy and value networks, with two 128 neuron ReLU hidden layers each.
- The policy outputs logits converted via softmax to probabilities: $\pi(a_i \mid s) = \frac{e^{z_i}}{\sum_{j=1}^{8} e^{z_j}}$ and a reward is received based on collisions, steps taken, goal progress, and goal completion.
- The value network minimizes the MSE against the Bellman TD(0) target: $TD_{\text{target}} = r + \gamma V(s')$ while the policy network is optimized using the advantage: $A = TD_{\text{target}} - V(s)$, which encourages actions that exceed the expectation.

CNN:
- Used a UNet architecture with two encoder and decoder blocks to predict a normalized cost-to-go map from each node to the goal.
- Ground truth cost-to-go examples were computed using backward dynamic programming (goal-rooted Dijkstra's) on simulated rollouts, via the deterministic Bellman optimality equation: $J(x, y, t) = min_a [c(a) + J(x^a, y^a, t+1)]$
- Weighted MSE loss was used, with higher weights assigned to node regions with small gradients (near obstacles and the goal) to mitigate forming local minima: $MSE_w = \sum_i \sum_j w_{ij}(\hat{y}_{ij} - y_{ij})^2$

Hybrid:
- First training stage: CNN with three convolutional layers, ReLU activations, max-pooling and padding, with 'squeeze and excitation' block to prioritize key channels [2]. Flattened final layer in order to predict action values (MSE loss) for each possible action given 4-layer 21x21 input.
- Second training stage: Iterative self-learning: paths generated by the CNN are labelled and aggregated to the dataset with a replay buffer to prioritize new entry. Then the CNN is retrained, repeated for 30 cycles of 500 paths.

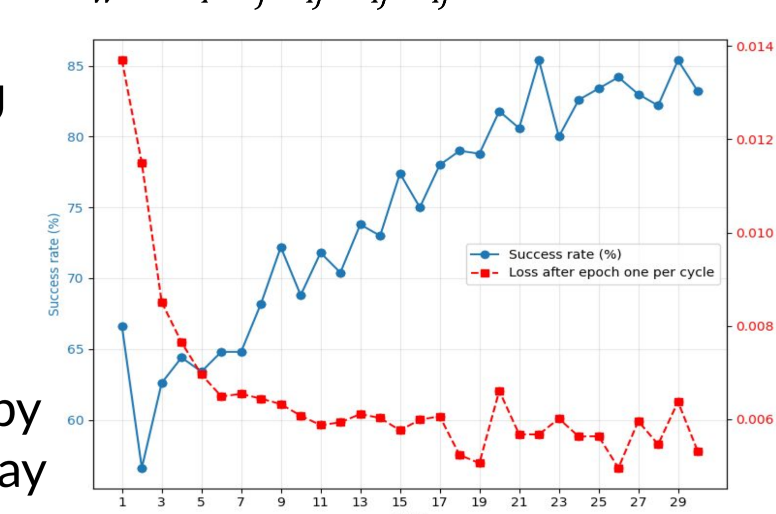

Figure 5. Success rate and training loss for self-learning cycles

Results

- **Training**: RL & CNN - 4000 episodes, Hybrid -16,000 episodes
- **Evaluation**: 10,000 pre-generated test episodes used for all models

	Success Rate (%) (Training Set)	Success Rate (%) (Test Set)	Path Length Ratio (Agent : A*)	Computation Time Ratio (Agent : A*)
Reinforcement Learning	91.56	91.04	1.195	0.028
CNN (UNet)	74.14	72.20	1.054	2.12
Hybrid	95.40	94.54	1.209	1.31
Dynamic A* (Baseline)	N/A	62.82	1.000	1.000

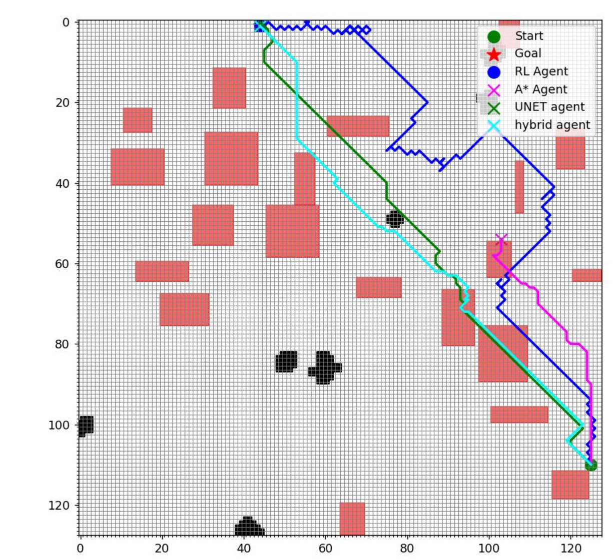

Figure 6. All four agent trajectories on a random 2D grid

Discussions & Future Research

<u>Discussions</u>:
- **All three learning-based approaches achieved higher success rates than the baseline**, with the **hybrid model achieving the highest success rate** (94.54%), followed by RL (91.04%), and then CNN (72.20%).
- The **CNN achieved the closest path length to A*** with a ratio of 1.054, followed by RL (1.195) and hybrid (1.209), likely due to sacrificing path length for higher success rates by encoding high penalties for collisions.
- **RL had the lowest computation time** compared to dynamic A* (0.028 ratio), while the hybrid and CNN exceeded the baseline time.

<u>Future Research</u>:
- As the RL agent outperformed dynamic A* in terms of both success rate and computation time, we would **continue refining the agent by tuning reward shaping**, specifically the step penalty, **to achieve a more optimal path length to the goal**.
- We would like to test our algorithm on **real-time navigation for physical robots** to evaluate how our model(s) perform **outside of simulation**.

References

[1] A. I. Panov, K. S. Yakovlev, and R. Suvorov, "Grid path planning with deep reinforcement learning: Preliminary results," Procedia Comput. Sci., vol. 123, pp. 347–353, 2018, doi: 10.1016/j.procs.2018.01.054.
[2] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Salt Lake City, UT, USA, Jun. 2018, pp. 7132–7141.
[3] R. Yonetani, T. Taniai, M. Barekatain, M. Nishimura, and A. Kanezaki, "Path planning using Neural A* search," arXiv preprint arXiv:2009.07476, Sep. 2020. [Online]. Available: https://arxiv.org/abs/2009.07476
[4] W. Li, Y. Liu, Y. Ma, K. Xu, J. Qiu, and Z. Gan, "A self-learning Monte Carlo tree search algorithm for robot path planning," Front. Neurorobot., vol. 17, p. 1039644, 2023.
[5] Y. Shen et al., "GundamQ: Multi-scale spatio-temporal representation learning for robust robot path planning," arXiv preprint arXiv:2509.10305, 2025. [Online]. Available: https://arxiv.org/abs/2509.10305
[6] Y. Zhang and P. Chen, "Path planning of a mobile robot for a dynamic indoor environment based on an SAC-LSTM algorithm," Sensors, vol. 23, no. 24, p. 9802, Dec. 2023, doi: 10.3390/s23249802.