Q-Learning and Model Predictive Control for Drone Wildfire Response

Chris Copans†
Department of Mechanical Engineering, Stanford University, Palo Alto, CA, 94305

Ankush Dhawan†
Department of Electrical Engineering, Stanford University, Palo Alto, CA, 94305

Mark Leone†
Department of Mechanical Engineering, Stanford University, Palo Alto, CA, 94305

† *Denotes equal contribution*

In this project, we explore the application of sequential decision-making algorithms for autonomous navigation for wildfire response drones. Using a simulated 2D grid environment, we model a drone navigating from its start position to a goal location while avoiding obstacles under limited sensing range constraints. The navigation problem was formulated as a Markov Decision Process (MDP), and Q-learning, SARSA, and Model Predictive Control (MPC) methods were implemented and compared. Experiments evaluated performance based on convergence speed, runtime, and path utility across various obstacle layouts in different grid worlds. Q-learning demonstrated robust baseline performance, while MPC provided real-time adaptability through limited-horizon planning. Hyperparameter optimization and reward shaping further enhanced algorithm efficiency. Results showed that integrating model-free and model-based methods, such as MPC with Q-learning, effectively balances computational efficiency, real-time planning, and decision-making flexibility, making it a promising approach for real-time drone navigation in limited visibility environments.

Our project GitHub page is available here: `https://github.com/ankushDhawan5812/aa228-project`

I. Introduction & Related Works

A. Introduction

Over the past two decades, catastrophic wildfires have become more common and ever larger [1]. At the same time, autonomous unmanned aerial systems (drones) are becoming cheaper and more capable. Small drones can be used for many purposes, but regardless of the specific application, their low acquisition and operating costs, and capacity for autonomy make them an attractive tool for managing wildfire risks. Our project was motivated by the confluence of these trends.

We investigate the ways in which sequential decision making algorithms can be applied to enable autonomous navigation of a fire fighting drone. While existing drone systems are extremely capable, the focus of this project is constrained to 2-dimensional navigation of a simulated quadrotor drone from its launchpad to a goal, with limited knowledge of its environment. The drone needs to navigate an obstructed space to a known destination, which simulates a wildfire.

Uncertainty is introduced by limiting the drone's sensing range. Although the drone will know it has reached the goal when it has reached as it traverses the environment, its observations change as its state trajectory evolves. The drone needs to decide on an appropriate path as it encounters obstacles along the way.

The ultimate objective of the project is to successfully find a path from the origin to the destination that avoids collisions with obstacles and minimizes the number of steps enroute. The final element of the project compares the efficiency and effectiveness of Markov Decision Process (MDP) solving methods and explores the effects of hyperparameter tuning, reward shaping, and other techniques to obtain better results.

B. Related Work

Many approaches have been successfully developed to navigate two-dimensional space. We chose to treat the problem as an MDP, with the drone aware of its true state throughout its journey. However, research outside that particular framework still proved valuable in shaping our work.

The Hex World scenario in *Algorithms for Decision Making* provides a foundation for our initial work. Hex World is an MDP example of navigating through a grid of tiles to reach a goal [2]. Model-free methods can be used to solve MDPs like Hex World, without the need for transition or reward models, through incremental value function estimations. The most basic of the model-free methods is the Q-Learning method, with SARSA and SARSA-Lambda methods building on Q-Learning to provide better performance in scenarios with sparse rewards. These ideas were the basis for our project, but we added uncertainty by limiting the drone's ability to identify obstacles beyond a certain distance.

In a scenario similar to ours, D. Asmar's Rock Sampling demonstration, shows a robot navigating a 2d-dimensional space using a sensor with a limited range, controlled by a parameter to enhance or degrade the sensor's efficiency [3]. His demonstration is framed as a Partially Observable MDP, but we were inspired by the idea of limiting the drone's sensing range and incorporated a constraint to our MDP-based approach where the drone knows its position but can only sense a limited number of steps ahead.

Conversations with M. Ho pointed us toward Model Predictive Control (MPC) for limited-horizon navigation. Ho described the process by which the MPC provides an action, leading to a new state, at which point another observation is made, an update is performed, and the process is repeated [4]. This method could include a limited horizon and is highly adaptable. Based on this work, we began investigating ways to combine Q-learning with MPC. Similarly, in the study of optimized industrial processes, Oh showed the potential of combining off-policy Q-learning with on-policy MPC [5]. The resulting two-stage prediction comes from a model and a model-free Q function [5]. Although this exact combination of methods was different from our intended path, it reinforced the idea of joining a limited-horizon MPC with the model-free Q-learning algorithm we used as a baseline.

II. Problem Setup

For this project, we model the agent as occupying one square space on a 10 by 10 grid, and invariant to rotation or orientation changes. The agent is bounded by the coordinates on the grid, and can take one action on any given time step. Each action can be a single step in any of the four cardinal directions, with no possibility of being in the same state as the previous time step. There is a 100% success rate of moving to the corresponding next state given your action (i.e. a deterministic transition model).

As depicted in Fig 1, the start state for the agent is at (0, 0), and the goal state is at (9, 9) (indicated by blue and green squares, respectively). For all grid worlds, the start and goal state remained the same. The agent receives a reward of 10 for reaching the goal state (highlighted in green). If the agent collides with a obstacle, it receives a reward penalty of -10 (indicated by red). All other states on the grid have no reward or penalty (in white). For the basic grid world, (depicted in Fig. 1a)), the obstacle is a central 2 by 2 square obstacle. To add complexity, in Fig. 1b only one path between the obstacles is possible. In Fig. 1c, the agent must choose between two possible directions around a staircase obstacle, with one leading to a dead end. In Fig. 1d, the square obstacle is moved to the top right of the grid, where two possible directions to approach the goal are still possible. In Fig. 1e, a single path through the obstacle is combined with a staircase that requires the agent to initially traverse away from the goal to eventually reach it. Finally in Fig. 1f, a maze-like obstacle is constructed where the agent must explore the long passages to find the goal, and potentially backtrack if it initially chooses the wrong passage.

In this project, we experiment with different path planning methods for the agent to reach the goal state on each of the different map worlds. The methods will be compared based on iterations to convergence, run time, and utility of the path.

III. Methods

A. Q-learning

Our first approach was to solve navigate using off-line Q-learning. This method computes a pre-compiled policy for the optimal action policy at each state in the grid space before the drone even takes off.

1. Assumptions

The location of the starting position is known. The drone will know an obstacle is encountered when it sees a negative reward in the horizon. The drone will know that it reaches the goal state when it , and the drone always knows its true position. All grid spaces are explored during trajectory sampling, and all actions that keep the drone within the boundaries of the map are valid for each grid space. Actions are deterministic, and the drone cannot remain in the same grid space.

2. Implementation

Implementation of the Q-learning method requires multiple phases. First, the selected map is imported with grid spaces represented as states with associated reward values. Then, a randomized 10-step trajectory sampling is performed to explore possible state, action, reward, next state combinations. This data is compiled as a matrix with columns representing state, action, reward, and next state (similar to the method used in Project 2 for the class).

The second phase is Q-learning. A Q-table of values at each state, action pair is initialized with zeros. Then, for each row of the sampled trajectories dataframe, equation (1) from [2] is used to update the action value function for each state, action pair. This is done iteratively, over a specified number of episodes, to ensure rewards propagate through the entire grid space.

$$Q(s,a) = Q(s,a) + \alpha\,(r + \gamma \max_{a} Q(s',a') - Q(s,a)) \tag{1}$$

The third phase is policy extraction. By looping through each state and extracting the action that results in the highest utility, using equation (2) from [2], the optimal policy for each grid space is identified. Using this policy, the drone will navigate along a path that maximizes value, which should avoid obstacles and reach the goal. The pseudo-code for the second and third phases of implementation is shown in Algorithm 1

$$\pi(s) = \arg\max_{a} Q(s,a) \tag{2}$$

Algorithm 1 Q-learning

```
Require: state_space_size, action_space_size, number_episodes, trajectories.csv, grid
  Q_Table ← zeros(state_space_size, action_state_size)
  for i ← 0 to number_episodes do                              ▷ Iterate for number_episodes
      for state ← 0 to number_trajectory_states do    ▷ Iterate over all states in trajectories.csv
          state, action, reward, next_state ← read_trajectory_csv(state)
          Q_Table ← Q_update(state, action, reward, next_state)       ▷ Update using equation (1)
      end for
  end for
  return Q_Table
  policy ← get_plan(Q_Table)                ▷ Extracts optimal policy for each state using equation (2)
  return policy
```

3. SARSA and Eligibility Traces

In addition to Q-learning, SARSA and SARSA with eligibility tracing (referred to as SARSA-λ), were also explored. SARSA builds on Q-learning by calculating the action value functions using the actual next action from a given state, instead of the max action. Hence, this is an on-policy method that updates as the agent follows the path. SARSA-λ then adds an exponential decay factor (λ), which returns larger rewards for states closer to the goal. These methods should allow faster convergence on a policy, especially in sparse reward environments like the maps in Fig 4. However, since the grid space is smal, the results were similar to our Q-learning baseline, and the team decided to pursue Model Predictive Control in lieu of continued work on SARSA to provide better online method results.

B. Model Predictive Control

Since drones will often need to plan while they are already flying in the air, especially in a time sensitive scenario like wildfire response, we explore model predictive control (MPC) approach. In this method, the drone explores a limited horizon in the full world space, plans a trajectory given that knowledge, executes a single action based on the plan, and repeats this until reaching the goal state or until the max number of iterations.

1. Assumptions

For this approach, we make the assumption that when the agent explores, it can explore only within a neighborhood, which is defined by one step in each of the four cardinal directions. In addition, we assume that the agent has full observability of the rewards in each of the states in the neighborhood. Finally, the agent will be able to stay in the same location only when it reaches the goal state (the algorithm terminates).

2. Implementation

For the implementation of MPC, the agent executes two phases: the plan, and the execute phase. First, the agent plans a trajectory given the current state and the rewards from the neighborhood states. This is done by performing a Q-learning update. Next, the agent executes the greedy action given by the `argmax` of the Q values in the Q matrix. This updates the state of the agent, which then plans and executes again until reaching the goal state, or terminating on the max number of iterations. Pseudo-code for the implementation is included in Algorithm 2.

Algorithm 2 Model Predictive Control

```
Require: start_state, max_steps, goal_state, backtrack_penalty, reward_shaping, grid
Ensure: states*, actions*
  state ← start_state
  append start_state to states
  for i ← 0 to max_steps - 1 do                                          ▷ Iterate for max_steps
    neighbors, actions ← get_neighborhood(state)
    rewards ← get_reward_from_neighbors(neighbors, grid, states, backtrack_penalty,
  reward_shaping)
    plan ← get_plan(state, neighbor_states, actions, rewards) ▷ Calls Q Learning on neighborhood
    state ← execute_action(state, plan)                ▷ Transitions agent to next state given action
    append state to states*
    append plan to actions*
    if state == goal_state then
      return states*, actions*                           ▷ Terminate early if goal state is reached
    end if
  end for
  return states*, actions*
```

IV. Analysis

A. Q-learning Baseline

Q-learning proved to be a successful method for navigating our maps. To further refine the performance of our model, a parametric sweep was performed for $0.1 \le \alpha \le 0.9$ and $0.1 \le \gamma \le 0.9$, to identify combinations that would speed convergence over three tuning maps (Fig. 1a-c), with a convergence tolerance of $1e^{-3}$. The results of the parametric sweep are shown in Fig. 5. As expected in a sparse reward space, low α values led to extended convergence times as future rewards along the path only minimally considered in the update. However, all combinations of $0.4 \le \alpha \le 0.9$ and $0.1 \le \gamma \le 0.6$ were able to converge on an optimal policy. When $\alpha = 0.4$ and $\gamma = 0.6$ were used for the three tuning maps, all resolved to an optimal path in one iteration, reaching the goal in only 18 steps. The resulting optimal policy rollouts are shown in Fig. 2.

However, when Q-learning was performed on the test maps (Fig. 1d-f) using the parameters found during tuning, map (f) failed to converge on the optimal path and the drone did not reach the goal. As shown in Fig. 3, the tuned Q-learning algorithm was not robust enough to transfer to new map spaces without additional tuning.

B. Model Predictive Control

Results for all MPC experiments were conducted with 100 simulations per MPC implementation, with each simulation comprising a maximum of a 200-step trajectory. Therefore, if the goal state was not reached within 200 steps, the simulation ended and the failed result of the MPC rollout was recorded. All results for the MPC ablation studies are detailed in Table 1.

			Tune Set		Test Set	
Ablation	*r*	*b*	% of sims reached goal	Mean steps to goal	% of sims reached goal	Mean steps to goal
Naive MPC	0	0	20.7	N/A	8	N/A
MPC with reward shaping	0.08	0	60.3	N/A	47.3	N/A
MPC with backtracking penality	0	0.3	95.0	N/A	100.0	69.0
Tuned MPC	0.08	0.3	100.0	38.0	100.0	37.7

Table 1 Results of ablation experiments. Values are averaged across each of the 3 maps for the tune and the test set. For each map, 100 simulations were run, with a maximum of 200 steps per simulation. *r* is the reward shaping parameter, and *b* is the backtracking penalty parameter.

1. Naive MPC

Our first implementation of model predictive control was Naive MPC, where the reward function for each neighboring state only depended on the reward of that state. Since we limited the observation horizon to 1 neighbor in each of the 4 cardinal directions of the current state, this did not allow for the propagation of a sparse reward signal from the goal state. Therefore, Naive MPC essentially explored the map randomly, avoiding obstacles but not receiving any positive reward signal unless within 1 state of the goal, only reaching the goal in 20.7% of simulations on the tune set.

2. MPC with backtracking penalty

To encourage exploration of previously unvisited states, we added a term to the MPC reward function that penalized visiting states already seen in that trajectory. The penalty was proportional to the number of times a state had already been visited and weighted by the backtracking penalty hyperparameter *b*. While the backtracking penalty enabled the agent to reach the goal more often than Naive MPC (finding the goal in 95% of simulations instead of only 20% on the tune set), the agent still unnecessarily explored much of the map before arriving at the goal due to a sparse reward signal.

3. Tuned MPC

In our problem formulation, we assume that the agent knows the position of the goal. We found that the sparse positive reward signal of the goal did not propagate back to most of the map through our prior MPC formulations. Therefore, we instilled the agent with knowledge of the goal position by adding a term to the MPC reward function that penalized the L1 distance to the goal of each state, weighted by reward shaping hyperparameter *r*. We report results for a "Tuned MPC" controller, which is the Naive MPC formulation with backtracking and reward shaping penalties added to the reward function. We tuned hyperparameters *b* and *r* by sweeping across reasonable ranges for each hyperparameter determined in initial experimentation, generating 100 simulations for each combination of *b* and *r*. The objective was to find the combination of hyperparameters that minimize the mean steps to goal across all 3 tune set maps, subject to the constraint that the goal must be reached in 100% of simulations, without the agent running into obstacles. This objective landscape is visualized in Fig. 6a. The tuned MPC controller reached the goal in 100% of simulations on the tune set, with a mean steps to goal across all 3 maps of 38.0.

4. Generalization of tuned MPC

We tested generalization of this controller and hyperparameter set on 3 other maps (the test set) and found that the tuned MPC controller reached the goal in 100% of simulations, with a mean steps to goal of 37.7 across the 3 test maps. Fig. 4 visualizes example trajectories of the agent across the train set (Fig. 4a-c) and test set (Fig. 4d-f). These results demonstrate that the MPC controller tuned on one set of maps can be applied to other sets of maps. We conducted a hyperparameter sweep on the test set to determine if the optimal hyperparameters on the tune set were also optimal in the test set, and found that a different combination of hyperparameters is optimal on the test set (Fig. 6).

V. Conclusion

Initially, Q-learning was chosen to serve as a baseline due to familiarity and ease of implementation, and in the end it proved to be quite effective as a solver. It was able to converge on the optimal solution for 5 out of 6 maps using the same parameters and within seconds. On more challenging maps, like the map in Fig. 1e, it was sensitive to minor changes to α and γ, and in the end it failed to reach the goal in the map in Fig. 1f.

The parameter sensitivity relates to the sparsity of the rewards in our grid space, and the relative rewards and penalties between the goal and the obstacles. The γ value was especially important, and hard to generalize across different maps. Maps with open areas did better with larger γ, as the goal reward was able to propagate more readily. However, when the same γ is used for an obstructed map, like the map in Fig. 1f, we hypothesize that when close to the goal, its large reward devalues the more immediate obstacle penalty, causing the drone to collide. So, while Q-learning was a suitable starting point, MPC proved to be much more robust.

MPC offered an online planning alternative to Q-learning, and aligned well as a solution for our initial problem formulation. While Naive MPC performed poorly, demonstrating the need for tuned MPC with reward shaping and backtracking penalties. Our MPC controller, tuned on a set of 3 maps, was able to readily generalize to a diverse set of 3 other maps without changing any hyperparameters. However, we found that when optimizing for minimal trajectory length, the tune set and test set of maps had different optimal hyperparameters, demonstrating that while the MPC controller can have 100% success at reaching the goal on an unseen set of maps, it is only optimal for the set of maps that it was tuned on.

When we held the hyperparameters found optimal on our tune maps constant, our tuned MPC controller performed better than the Q learning policy, which only reached the goal in 2 out of 3 test maps. To extend this MPC controller to perform near-optimally on an arbitrary set of maps, the hyperparameters should be tuned on a large and diverse set of maps. In real-world drone control, it would be necessary to consider state uncertainty and uncertainty in the transition model, which were not considered in this study. Additional consideration must be given to the "obstacles" in a real world drone control scenario, and how negative rewards would be determined for mid-air collisions, changes in altitude, and collisions with terrain.

VI. Team Contributions

This project was a true team effort with close collaboration on scoping, strategy, and implementation as we explored different approaches. This final report was written collaboratively by all team members.

- C.C. worked on Q-learning, SARSA, and SARSA-Lambda baselining, including parametrically optimizing the parameters of Q-learning, and testing Q learning on multiple maps.
- A.D developed the MPC function, including the naive and backtrack penalty ablations. A.D. was also responsible for developing the map generator and created visualization code for the policies.
- M.L. developed the trajectory sampler for Q-learning, implementing reward shaping in MPC, and running testing and parameteric optimization for MPC, including robustness testing and hyperparameter optimization, and created the visualizations for the Q-learning and MPC results.

Acknowledgments

We thank the entire AA228/CS238 teaching team for their instruction and support throughout the quarter. We would like to specifically thank Dylan Asmar and Michelle Ho for offering their insights in approach and methods during office hours, and Prof. Kochenderfer for his instruction throughout the quarter.

References

[1] Blakemore, E., “Large fires becoming even larger, more widespread,” *The Washington Post*, 2022, March 27. https://doi.org/https://www.washingtonpost.com/science/2022/03/27/wildfires-increasing-frequency-climate-change/.

[2] Kochenderfer, M., Wheeler, T., and Wray, K., *Algorithms for Decision Making*, The MIT Press, Cambridge, 2022.

[3] Asmar, D., “RockSample.jl [README],” *GitHub*, 2023.

[4] Ho, M., MPC followup from 228 OH [email correspondence], 2024.

[5] Oh, T., “Q-MPC: stable and efficient reinforcement learning using model predictive control,” *IFAC-PapersOnLine*, Vol. 56, No. 2, 2023, pp. 2727–2732. https://doi.org/10.1016/j.ifacol.2023.10.1369.

VII. Appendix I: Additional Figures

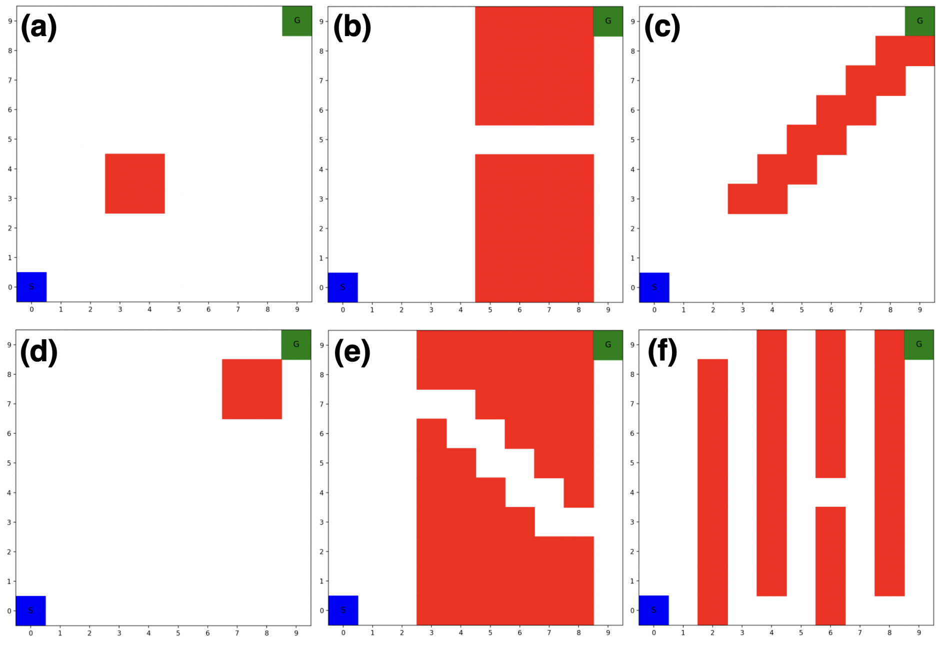

Fig. 1 Grid worlds for testing implementations

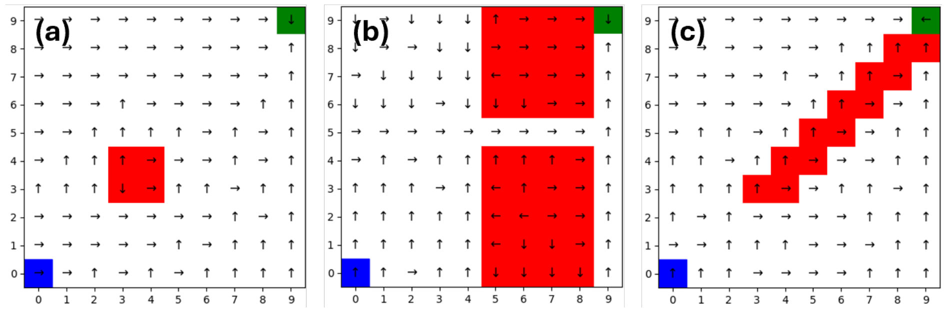

Fig. 2 Rollout of policies computed using Q-learning with $\alpha = 0.4$ and $\gamma = 0.6$

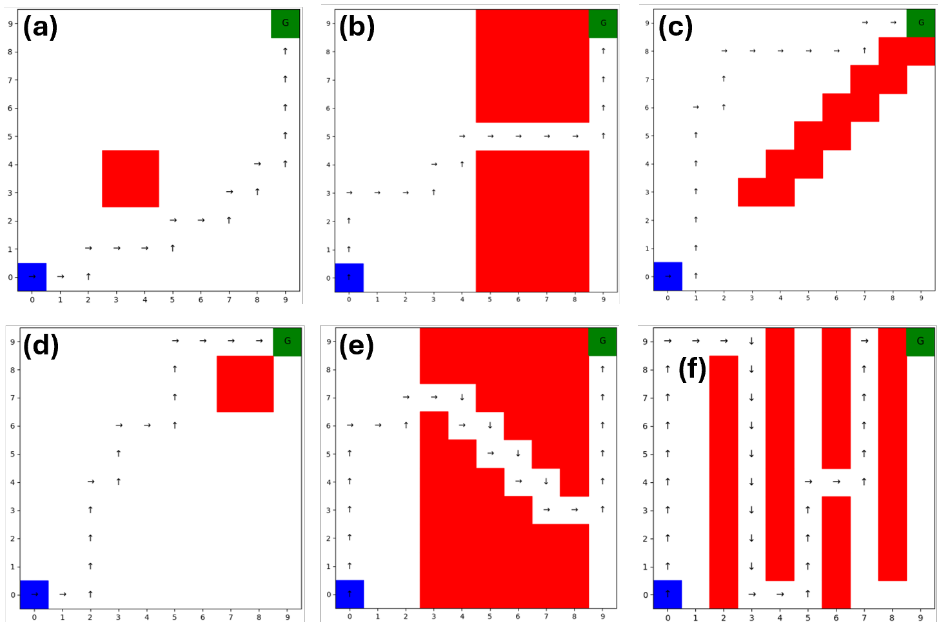

Fig. 3 Example rollouts of tuned Q-learning on all gridworld maps *(note failure to reach goal in map f)*.

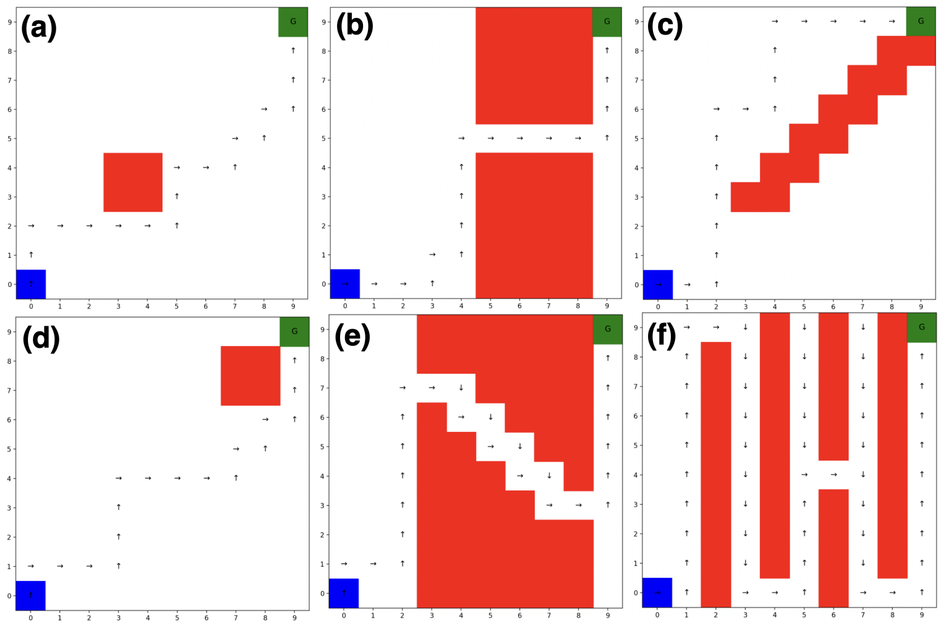

Fig. 4 Example rollouts of tuned MPC controller on all gridworld maps.

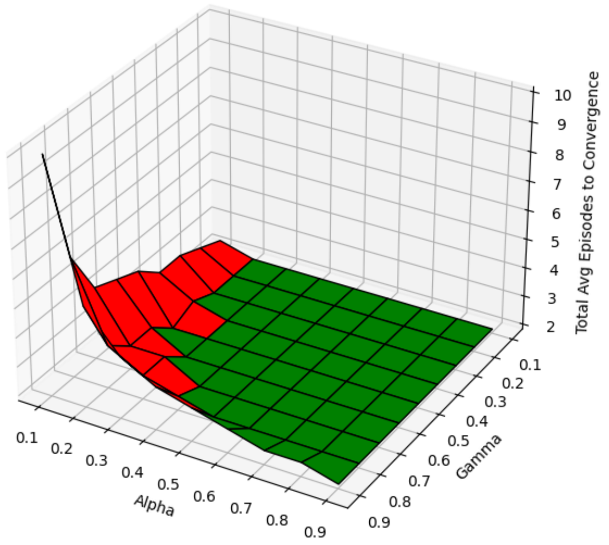

Fig. 5 Parametric Sweep of learning rate and discount factor in Q-learning

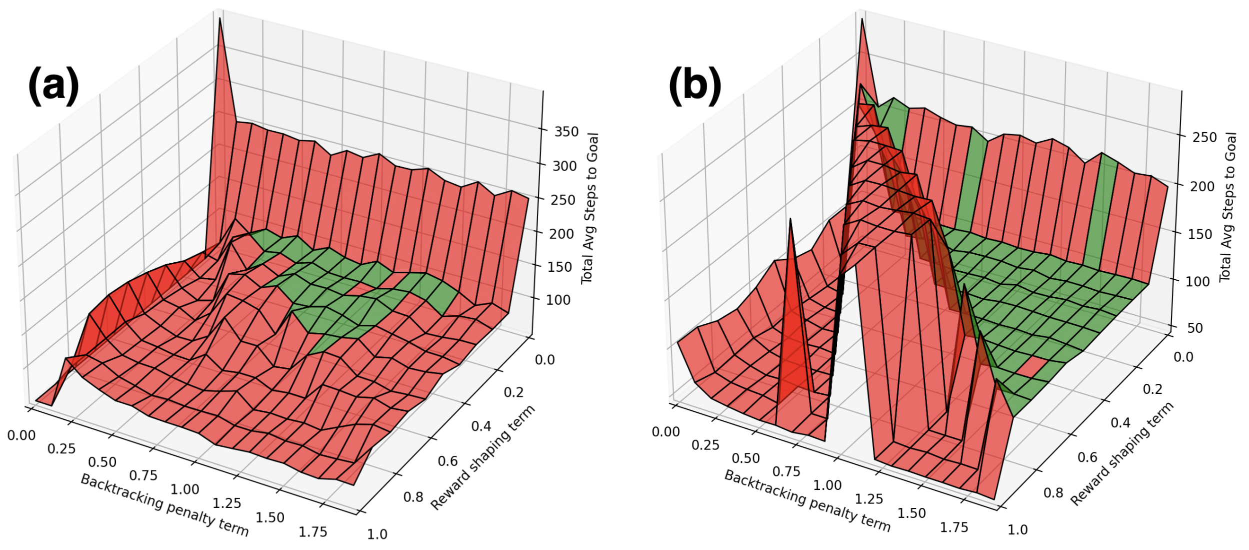

Fig. 6 Parametric Sweep of reward shaping term and backtracking penalty term in MPC for (a) maps 1-3, (b) maps 4-6. Red shading denotes parameter combinations that do not lead to 100% success at reaching the goal over 100 simulations, while green shading denotes parameter combinations that achieve 100% success.