High-Rate Autonomous Navigation Through Unknown GPS-Denied Environments

Parker Barrett
Ann and H.J. Smead Aerospace Engineering Sciences
University of Colorado
Boulder, USA
paba8584@colorado.edu

Abstract—**This paper details the formulation of a high-rate autonomous navigation problem in GPS-denied environments as a Markov Decision Process. Three solution methods, an offline value iteration solver, an online Monte Carlo Tree Search algorithm and a Deep Q Network reinforcement learning algorithm are studied. The results and relative strengths and weaknesses of each algorithm are compared both quantitatively and qualitatively.**

I. Introduction

There are many applications for high-rate autonomous navigation through GPS-denied environments, including urban air mobility platforms, military vehicles in contested environments, warehouse robotics and more. In these application spaces, an agent may be given a set of initial and terminal conditions but have no a-priori knowledge of the environment it finds itself in. A common platform for these kinds of environments are Unmanned Aerial Vehicles (UAVs), which have the advantage of being highly maneuverable and can carry a variety of sensor payloads.

In this paper, the author considers a UAV with known initial conditions and coordinates of a desired terminal state. The environments is a hypothetical warehouse with obstacles placed randomly throughout, to ensure the solution algorithms are generalizable. Three different types of algorithms, an offline solver, an online planner, and a reinforcement learning approach are implemented and their performance is compared.

II. Background and Related Work

There is a great body of literature available, both about the problem of high rate autonomous navigation in GPS-denied environments and about the solution approaches used in this project. This section surveys a handful of these relevant papers.

A. Survey on UAV Navigation in GPS denied environments [1]

This paper provides an overview of the problem of UAV navigation in GPS-denied environments and describes some of the core sensors and sensor fusion approaches used to solve the problem. While this project formulates the problem as a Markov Decision process in which full observability is assumed, this paper is relevant because it provides a strong background on some of the challenges an agent faces when navigating through GPS denied environments.

B. Value Iteration in Continuous Actions, States and Times [2]

This paper proposes a novel algorithm for applying value iteration to continuous action/state spaces on continuous time intervals. While much of the details are beyond the scope of this project, the paper included a primer on the traditional discrete value iteration algorithm and detailed important considerations for extending value iteration to more realistic applications and environments.

C. Mastering the game of Go with deep neural networks and tree search [3]

This paper is the famous AlphaGo paper written by Google DeepMind after successfully building the first artificial intelligence program capable of defeating professional Go players. The technical content of the paper is also highly relevant because it describes the algorithm which uses both Monte Carlo Tree Search and reinforcement learning techniques, both of which are leveraged in this project.

D. Goal-driven Navigation of an Unknown Environment with Monte Carlo Tree Search [4]

This paper is highly relevant to the current project, in which the authors define a mobile robot in a grid world with the goal of navigating from one location to another. The environment also contains random obstacles similar to this project and the paper is full of of lessons learned/tips and tricks for Monte Carlo Tree Search that can be used to improve the current algorithm.

E. Deep Reinforcement Learning Based Mobile Robot Navigation: A Review [6]

This paper provides a good overview of several deep reinforcement learning algorithms, including Deep Q Networks, Double Q Learning, Proximal Policy Optimization and more. This survey helped select a deep reinforcement learning algorithm to use in the challenge portion of this project.

III. Problem Formulation

The proposed autonomous navigation problem can be formulated as a Markov Decision Process. The state space is a 2-dimensional, 50x50 grid world. The action space is left, right, up or down, meaning that no diagonal actions are allowed in this model. Additionally, actions can only move one square at a time, "double moves" are not allowed. There is one positive reward in the environment, which will be in a known position within the grid world relative to the starting point. This is akin to providing an autonomous vehicle a way-point of some kind. The positive reward will be worth 300 points.

Obstacles will be placed randomly throughout the grid world. The reward at each of the states with obstacles will be -50 points. The outer ring of the grid world is assumed to be stationary walls which could also damage the vehicle if it is run into, therefore the border will also have a reward of -50 points, meaning the walls are classified in the same way as obstacles. All remaining squares have a reward of 0 points.

The Markov Decision Process has a discount factor of 0.95. The discount factor helps "discount" future rewards in the roll-out operations, balancing current vs future rewards. This value is largely problem specific, although it typically ranges from 0.9 to 0.99, and the selected value seemed to work well in the current environment.

Finally, perfect control will be assumed, so the transition probabilities are explicit (in contrast with a generative model), and will always be 1 for each state, action, next state tuple. The Markov Decision Process can be succinctly represented with equations/statements 1-5.

$$\mathcal{S} \in x = 1:50, y = 1:50 \tag{1}$$

$$\mathcal{A} \in \text{:right}, \text{:left}, \text{:up}, \text{:down} \tag{2}$$

$$\mathcal{R} \in +300 \text{ (final reward)}, -50 \text{ (obstacles)}, 0 \text{ (otherwise)} \tag{3}$$

$$\mathcal{T} \in 1 \text{ (for s, a, s' tuples)} \tag{4}$$

$$\gamma = 0.95 \tag{5}$$

Fig. 1 shows a single instance of the randomly generated, 50x50 grid world with obstacles. Here, the reward is placed in the center of the grid and is surrounded by the obstacles. The location of this reward can be defined by the user and is examined in depth later in this paper for each of the algorithms considered.

IV. Solution Approach

Three solution approaches of increasing complexity and difficulty were used to solve this problem. The simplest example, which also served as a baseline was an offline value iteration solver. This translates to the real world scenario in which the agent assumes full observability of the environment and is able to plan the optimal policy (which action to take in a given state) for the entire state space.

Next, the mid-level approach is an online planner, a time-constrained Monte Carlo Tree Search approach. The time-constraint is intended to simulate the high-rate dynamics of the agent in which there is a limited amount of time to select a new action before the state evolves and the current tree search is no longer valid.

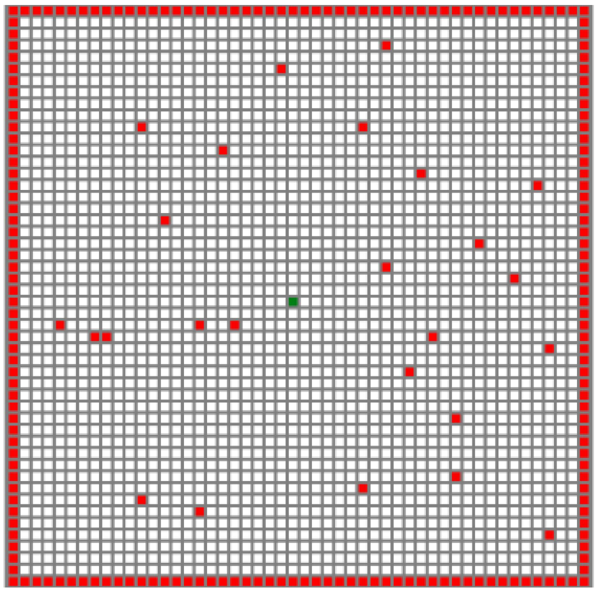
Fig. 1. Randomly generated grid world with obstacles.

Finally, the the third and most challenging approach is using a Deep-Q Network reinforcement learning algorithm. Unlike Q learning, a tabular RL algorithm, the Deep-Q Network approach uses a neural network to approximate the Q function. The specifics of each of these approaches are considered in the sections below.

A. Value Iteration

Value iteration is an iterative application of the Bellman Update, given in Eqn. 6, to recursively estimate the value function at each state in the grid world environment.

$$V'(s) = \max_a (R(s,a) + \gamma \sum_{s'} T(s'|s,a)V(s')) \tag{6}$$

Since the Bellman Update equation is a contraction mapping, it is guaranteed that value iteration will converge to the optimal value function.

In the algorithm implementation, V and V' are initialized to vectors of ones and zeros respectively, each the size of the state space, where each of entries represent the current best value estimate at each of the states in the grid world. The initial values are arbitrary since the algorithm is guaranteed to converge, it is just critical that they are different values since the convergence criteria is when the maximum element of the absolute value of the two vector is less than some tolerance, ϵ. In this implementation, we define ϵ = 1e-15.

B. Monte Carlo Tree Search

A common problem for online planning algorithms is computational complexity, especially when one considers the approach of creating and expanding a decision tree for every possible action in every possible state. These decision trees may work for very simple problems, but quickly grow exponentially and become intractable as the state and action spaces grow larger.

Monte Carlo Tree Search is an online planning algorithm which seeks to reduce computational complexity of a naive tree expansion while retaining many of the benefits. The approach of the algorithm is to first perform a search and expansion in the tree, followed by a roll-out simulation and backup to estimate the action value function, Q(s,a). After the roll-outs are complete, we select the action which maximizes Q(s,a).

In the implementation in this project an Upper-Confidence Bound (UCB) roll-out policy exploration heuristic is used for the roll-out, and is given in Eqn. 7. In the equation, c is the exploration parameter which is a scaling factor which can be tuned to increase or decrease the amount of exploration. N(s) is the total number of times a state in the tree has been visited and N(s,a) is the number of times an action has been taken from that state. When actions have been tried less frequently, the denominator grows large, making the exploration bonus increase, thereby increasing the probability of selecting that action.

$$a = argmax_a Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}} \tag{7}$$

One final note of importance with the implementation is that to simulate the high rate dynamics of the system, a planning time cap of 40ms was enforced. This means that the planner runs for 40ms and then must select an action. This step was repeated after each action to dynamically plan a path through the environment in real time. Additionally, in exploring the algorithm's performance, the author experimented with shifting this planning time limit to see the impact on performance. In a real world implementation, this would likely be a performance requirement driven by the rates of the vehicle dynamics, how dense the environment is with obstacles and safety considerations.

C. Deep Q Network

After looking into several deep neural network approaches to solving this problem, it was determined that using a Deep Q Network would be a strong candidate algorithm since it does a good job handling the three primary challenges of reinforcement learning: exploration vs exploitation, credit assignment and generalization. Created by Google's DeepMind in 2015, the idea of Deep Q Network is to use a neural network to learn and approximate the Q values, which can be used to estimate the optimal value function and actions from any given state.

A naive approach to implementing this algorithm would be to just combine Q learning with neural networks, however a number of issues arise which drive some of the decisions in the actual algorithm implementation. The main problems with the naive approach are highly correlated samples, size-1 batches and a moving target, meaning we attempt to estimate the network parameters as they are changing. To overcome the first two challenges, we leverage a data buffer, in which we collect many samples of data and process it all at once. To overcome the third challenge, we periodically freeze the target and train with the frozen network.

The algorithm begins by initializing two neural networks, the main and target networks. Using the main network, we collect a number of data samples by performing roll-out simulation with the a roll-out policy defined by the neural network approximated Q values. These (s, a, r s') tuples are stored in an experience buffer, which is periodically emptied. After a given number of roll-outs, we train our network on the accumulated data, periodically setting the target network equal to the main network to avoid moving the target after each training episode. Our loss function in training is given by Eqn. 8.

$$l(s,a,r,s') = (r + \gamma \max_{a'} Q_{\text{target}}(s',a') - Q_{\text{main}}(s,a))^2 \tag{8}$$

After training, we begin generating new data via roll-outs with our main network and repeat the process.

Since this is the challenge portion of the project, in addition to implementing this algorithm in the new grid world environment, an additional challenge of dynamic targets were introduced. The dynamic targets were allowed to move in the environment according to the same rules as the agent, however at a lower rate than the agent. The idea behind including dynamic targets is to simulate things like humans or other robots moving through the environment. In the real world, not all obstacles would be completely stationary, and the algorithms would need to handle the changes. A rate of 3 times slower than the agent was selected as a suitable dynamic obstacle rate to allow the agent enough time to plan and maneuver with room to spare.

V. Results

This section summarizes the results for the implementations of each of the previously described algorithms. It also details points of failure, leading to improvements and future work in the following section.

A. Value Iteration

The value iteration approach was successfully and efficiently implemented in the grid world. The value iteration function consistently converged under the tolerance of 1e-15 in less than 0.4 seconds, demonstrating a high confidence value estimate in a very small amount of time. Fig. 2 shows the results of the value iteration.

The algorithm works effectively regardless of where the random obstacles are placed, and the results are as expected, with the large reward placed in the center of the grid, therefore the values are propagated outwards towards the edges via the Bellman backup. Fig. 3 below shows the convergence criteria against the number of value iterations for a given run.

These results demonstrates that the algorithm quickly converges to a very low residual, and then takes many more

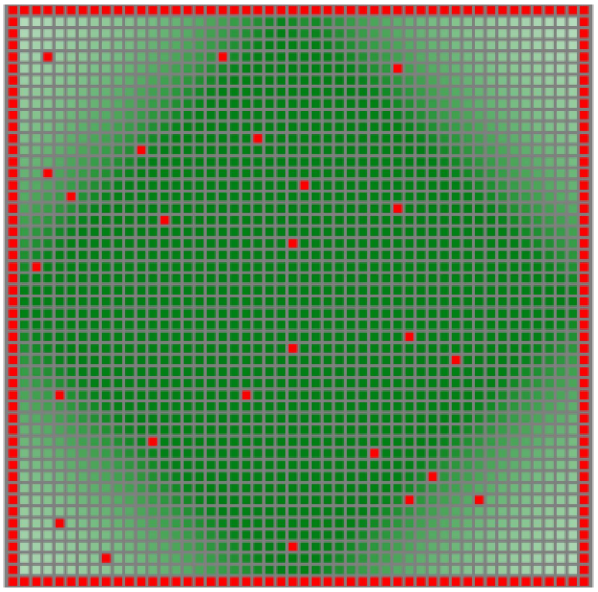

Fig. 2. Value iteration results

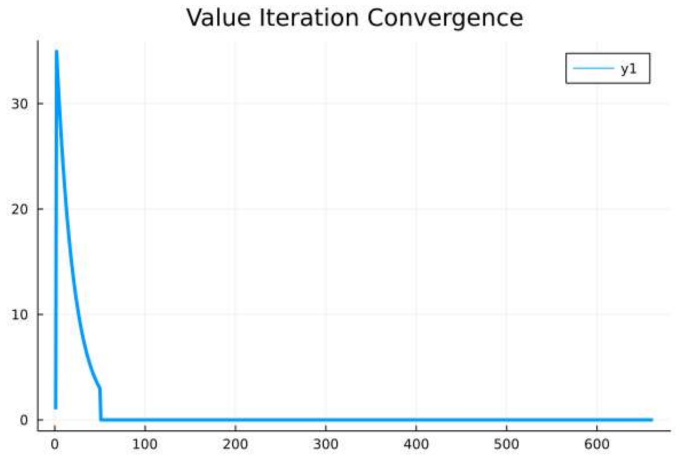

Fig. 3. Value iteration convergence per iteration

iterations to achieve the tolerance, which each have diminishing marginal returns. These results could be used along with performance/timing requirements to select an optimal convergence criteria for the specific application.

B. Monte Carlo Tree Search

The Monte Carlo Tree Search algorithm implementation was also successful and the algorithm demonstrated the ability to successfully navigate to the target. First, we can examine the case where the agent's initial position is in one of the corners, while the reward remains in the center of the grid world. Fig. 4 below shows how the algorithm is able to successfully navigate to the reward, avoiding obstacles.

Note that for this portion of the project, a new plotting environment was created from scratch with a few features to help illustrate the performance of the Monte Carlo Tree

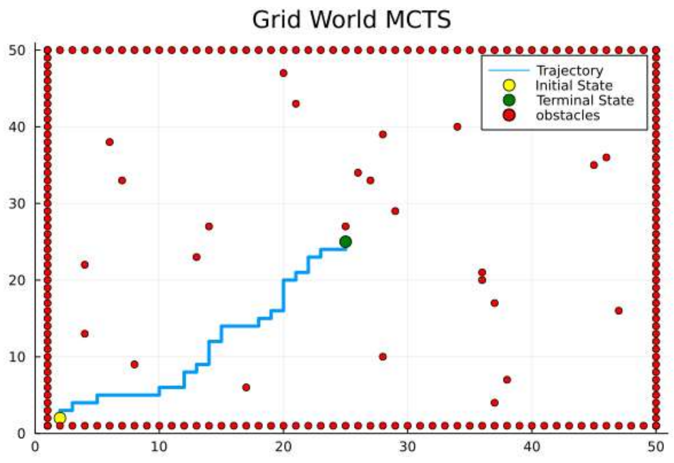

Fig. 4. Monte Carlo Tree Search Results

Search. The environment itself and rewards are identical to the remainder of the project.

While the entire planning algorithm is capped at 40ms, this represents a number of Monte Carlo simulations run and backed up to estimate the action value pairs. Each individual execution of the search, expand, roll-out and backup process consistently executed in less than 0.01 ms. This means the algorithm could perform approximately 400 roll-outs before any action selection was required, allowing it to fully explore the environment and find the reward.

A challenge in the implementation, was that occasionally the randomly generated obstacles would appear in a large cluster blocking the path near the reward. I found that increasing the search tree depth and relaxing the planning time limit to around 50-60ms was the best approach these situations.

C. Deep Q Network

The most challenging case, the deep Q network implementation was partially successful, in which the biggest challenge to overcome was generalization. In the first attempt at implementation, the obstacles were generated randomly at the beginning of the run and remained constant throughout training. The deep neural network was able to very successfully navigate to the target after training in this case, including under conditions that caused the Monte Carlo Tree Search to occasionally fail, such as when a large cluster of obstacles exist near the agent's initial condition or the reward. Fig. 5 shows the resulting path the agent in the environment.

Note that similar to the previous section, the new plotting environment was used. The environment itself and rewards are identical to the remainder of the project.

A few major challenges exist with the deep Q network approach. The first challenge arises when trying to make the implementation more generalizable. The neural network demonstrated the capability to adjust to randomized obstacles. The key to this success was the experience replay buffer, training the network on a batch of random data. This prevented the network from becoming over-confident due to "memorizing"

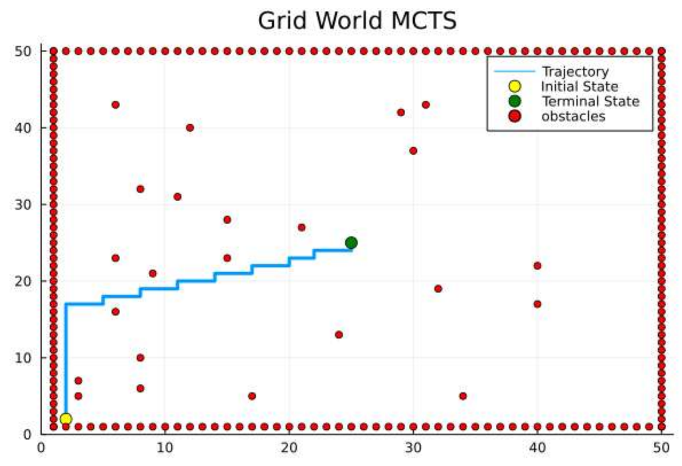

Fig. 5. Deep Q Network Results

the locations of obstacles. Despite these efforts new patterns of obstacles would occasionally still stump the network, causing issues and a lack of robustness/consistency.

The next major challenge is training time. When given a sufficient amount of time and data, the Q network is able to effectively navigate to the reward. Fig. 6 shows the learning curve for the algorithm, demonstrating the number of training episodes it takes to achieve the desired rewards. Note that 2000 training episodes make up each epoch, and the following model took well over an hour to train. While acceptable on a grid world problem, this would become a very real challenge on a continuous state space which is orders of magnitude larger.

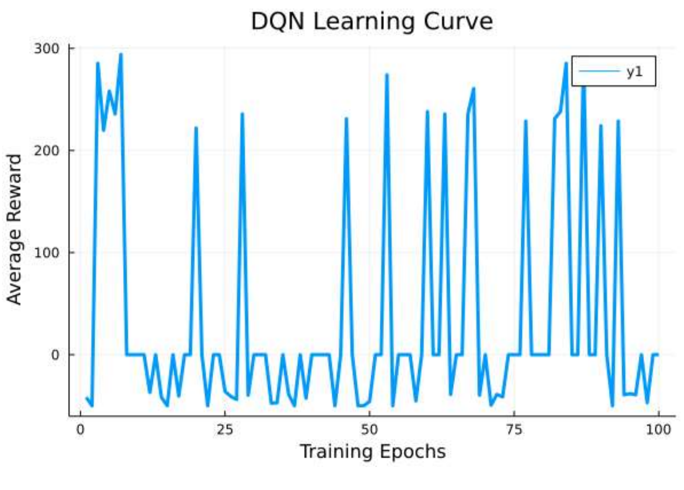

Fig. 6. Learning Curve for Deep Q Network Algorithm

Each training epoch in the figure consists of 2000 training episodes. The epsilon greedy exploration policy selects primarily random actions at the beginning, explaining the highly oscillating behavior. As the neural network learns and the exploration policy decays leading to the selection of the optimal actions, the average the negative rewards (obstacles) are not hit as often. The mean total discounted reward for a set of simulations after training was 48.75. The standard error of the mean for these simulations was 3.81.

One final approach taken to solve the problem was including dynamic obstacles which could move at rates of around 1/3 the speed of the agent. The neural network was unable to handle these cases and there was a limited amount of time available for testing. The next section discusses future work applicable to this shortcoming.

VI. Conclusion and Future Work

High rate autonomous navigation in GPS-denied environments is a very challenging problem in which many solution methods can be considered. In the grid world environment used throughout this paper, value iteration, Monte Carlo Tree Search and Deep Q Network approaches were compared.

The value iteration algorithm was able to routinely converge on to optimal value estimates for every state in the grid world within 0.4 seconds. The Monte Carlo Tree Search algorithm, capped with a planning time of 40 ms, was also able to expand its search tree and backup the Q value estimates in an efficient manner to successfully navigate the agent to the terminal state. Outliers existed in the Monte Carlo Tree Search approach when obstacles were clustered near the reward, however increasing the search tree depth and relaxing the planning time constraints helped overcome these challenges. Finally, with sufficient training and data sets, the Deep Q Network approach was capable of learning a set of network weights which approximated the the Q values of the environment. This network allowed the agent to successfully navigate to the terminal state with a mean return of 48.75.

Future work could be done on a handful of portions of this project. The first area of improvement would be the robustness of the Monte Carlo Search Algorithm. Finding and implementing methods for handling outlier cases in which the obstacles were placed in arrangements that made it difficult to find the reward in short planning times, would be the primary objective.

Another potential area of future work would be in the implementation of dynamic obstacles in the grid world environment. Generalization is one of the biggest challenges in reinforcement learning, and determining how to train the neural network to efficiently handle these cases would be an interesting problem.

One final area of potential future work is combining the Monte Carlo Tree Search with the neural networks, an approach that has historically proven effective in algorithms like AlphaGo. The successful combination of these techniques could lead to a highly efficient autonomous navigation algorithm, which could handle very unique environments with highly dynamic obstacles.

VII. Contributions and Release

Parker was fully responsible for writing the code for all aspects of the project, including algorithm development, collecting and cleaning data, generating visuals and writing the final report.

The author grants permission for this report to be posted publicly.

Acknowledgment

The author would like to thank the course instructor, Dr. Zachary Sunberg and the Teaching Assistant, Qi Heng Ho, for the learning opportunities, interesting lectures and rewarding assignments in the Decision Making Under Uncertainty Course.

References

[1] G. Balamurugan, J. Valarmathi and V. P. S. Naidu, "Survey on UAV navigation in GPS denied environments," 2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES), 2016, pp. 198-204, doi: 10.1109/SCOPES.2016.7955787.

[2] M. Lutter, S. Mannor, J. Peters, D. Fox and A. Garg, Proceedings of the 38th International Conference on Machine Learning, PMLR 139:7224-7234, 2021.

[3] Silver, David Huang, Aja Maddison, Christopher Guez, Arthur Sifre, Laurent Driessche, George Schrittwieser, Julian Antonoglou, Ioannis Panneershelvam, Veda Lanctot, Marc Dieleman, Sander Grewe, Dominik Nham, John Kalchbrenner, Nal Sutskever, Ilya Lillicrap, Timothy Leach, Madeleine Kavukcuoglu, Koray Graepel, Thore Hassabis, Demis. (2016). Mastering the game of Go with deep neural networks and tree search. Nature. 529. 484-489. 10.1038/nature16961.

[4] P. Contreras, M. Gee, and D. Knowles "Goal-driven Navigation of an Unknown Environment with Monte Carlo Tree Search," Unknown.

[5] K. Zhu, T. Zhang, "Deep Reinforcement Learning Based Mobile Robot Navigation: A Review." Tsinghua Science and Technology 26(05): 674-691.

[6] Z. Sunberg, "ASEN 5519-002 Course and Lecture Materials" University of Colorado Boulder, 2022.