University of Colorado Boulder

ASEN 5519 - Decision Making Under Uncertainty

End of Semester Project

Limited Sensor Range Grid-World Motion Planning Informed by Naive Global Information

Author:
Kyle Li
104649224

Author:
Joe Miceli
109283278

Author:
Peter Amorese
106114965

Professor:
Dr. Zachary Sunberg

May 3, 2022

College of Engineering & Applied Science
UNIVERSITY OF COLORADO BOULDER

Contents

I. Introduction

Navigating an environment as a mobile agent such as a car, rover, or drone requires knowledge of its environment. While the agent may be familiar with the broad characteristics of the environment there are often significant details such as poor road conditions, wildlife, or other undesirable characteristics that make originally planned routes undesirable. This is a common problem in both research and commercial applications [1]. To simulate such a problem, we define a car (the agent) that is equipped with a global sensor and a local sensor. The global sensor gives the layout of a grid-world environment to the agent, but is unaware of any hidden obstacles or costly paths. The local sensor alternatively has finite range but unveils the true state of the road and paths within range. In this problem approach there are two solvers: a local and global solver. Since the local sensor unveils the true state of the agent's nearby surroundings the local problem becomes a Markov Decision Process (MDP) problem. The local solver solves this MDP using the information from the local sensor, the dynamics of the car, and information from the global solver thereby producing the best action according to conditions within sensor range. The global solver takes in this information to inform its decision making to calculate the path across the global environment. In effect, the combination of both planners makes the algorithm behave similarly to an online planner.

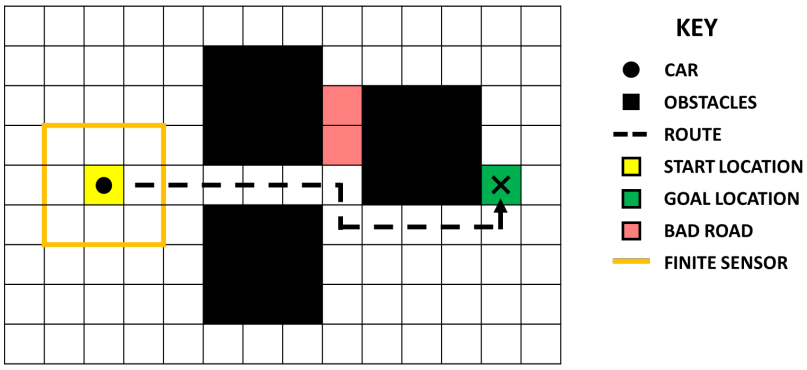

Fig. 1 This diagram shows the car traversing some grid world environment with a finite sensor range of 1.

II. Background and Related Work

The problem of navigating unknown environments while avoiding costly paths is a commonly studied topic in modern guidance and navigation problems [1]. There are many approaches to converting this real-world scenario into a problem approachable by algorithms. In the paper: Racetrack Navigation on OpenAIGym with Deep Reinforcement Learning [2], the environment was represented as a 96×96 RGB grid of randomly generated race track segments and the agent was taught to follow the track using double Q-learning. Another paper: Goal-driven Navigation of an Unknown Environment with Monte Carlo Tree Search utilized a simple grid world with randomly placed obstacles and goal location. This scenario's agent has the ability of traverse laterally or longitudinally and has limited knowledge of its surroundings globally, but can have perfect knowledge of the environment local to it [5]. The paper: Path Planning of Mobile Robot With Improved Ant Colony Algorithm and MDP to Produce Smooth Trajectory in Grid-Based Environment, used an MDP as a low level "filter" with an ant colony optimization motion planner. [4]. Our problem's local sensor can act as this low level "filter".

In our problem, each step is formulated as a local Markov Decision Process where the agent can only observe a subset of the entire state space of the environment. We assume the sensor provides perfect knowledge of states within range so maintaining beliefs of states is not necessary. Formulating the problem in this manner keeps it small and allowed us to use a value iteration solver rather than something more complicated like MCTS on histories, DESPOT, or POMCPOW [8] [6]. We use the idea of hierarchical iterative planning [3] to incorporate the local and global planners.

There are many potential approaches to solving MDPs and the approaches are broken down into model-based or model-free and on policy or off policy algorithms [10]. Model-based algorithms attempt to learn the transition function T and rewards R to find the optimal policy (π^*) from solving the MDP. A model-free algorithm attempts to find the optimal action value function (Q^*) or optimal policy without estimating T or R. On-policy algorithms use experience

generated from the algorithm's current policy whereas off-policy algorithms learn from experiences generated by both the current and previous policies [11]. We assume that both T and R are known in our problem so reinforcement learning is not required however, our agent can be thought of as "learning" about its environment as it moves.

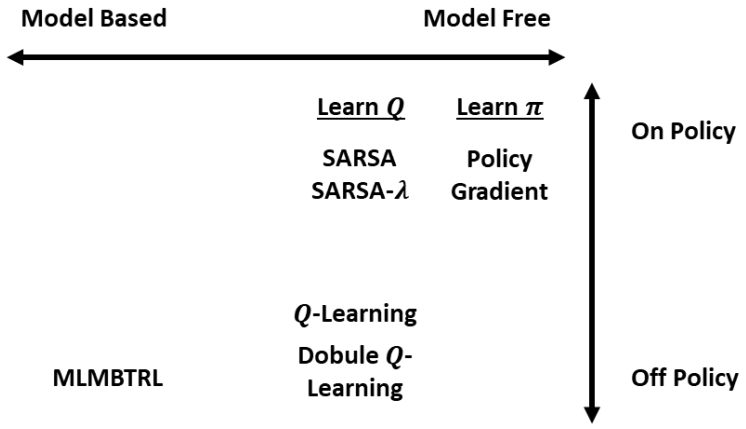

Fig. 2 Diagram showing various algorithms utilized to solve MDP problems. Note that MLMBTRL is the acronym for Maximum Likelihood Model Based Tabular Reinforcement Learning

III. Problem Formulation

Given an agent that has finite range access to an MDP, $\mathcal{S}$, as well as a naive global transition system, $\mathcal{G}$, guide the robot to goal while attempting to maximizing cumulative reward.

IV. Preliminaries

A. Local MDP

The underlying MDP can be formulated as a tuple $\mathcal{M} = (Z, A, R, T_M, \gamma)$ where:

- Z is a set of discretized grid world states,
- $A = \{up, down, left, right, stay\}$ is a set of actions,
- $R : Z \times A \times Z \mapsto \mathbb{R}$ is a reward function where $R(s,a) = \begin{cases} 0.0, & \text{if } s \in s_{goal} \\ -2000.0, & \text{if } s \in obstacle \\ -100.0, & \text{if } s \in badRoad \end{cases}$
- $T_M : Z \times A \times Z \mapsto (0,1)$ is a probabilistic transition function,
- $\gamma = 0.75$ is a discount factor

The MDP $\mathcal{M}$ describes the true underlying probabilistic dynamics of the vehicle. Due to the finite sensor range constraint for this problem, the system only has access to a local portion of $\mathcal{M}$. We can define the local MDP as a tuple $\mathcal{S} = (S, s_c, A, R, T, \gamma, \alpha)$ using the definition of $\mathcal{M}$ where:

- $S \subseteq Z$ is a limited set of states within the range of the sensor,
- s_c is the vehicle's current state,
- $T : S \times A \times S \mapsto (0,1)$ is a probabilistic transition function where $T(s,a,s') = T_M(s,a,s')$ iff $s, s' \in S$, otherwise $T(s,a,s') = 0$,
- $\alpha \in \mathbb{Z}_{>0}$ is a finite sensor range,

with all other items being equivalent.

The transition function $T(s,a)$ used a deterministic distribution for the next state, s' where s' was the state (within the state space) corresponding to the action taken from s (i.e. taking action "up" from state $(1,1)$ resulted in state $(1,2)$. If an action would produce a state outside the global state space, then the state did not change. To simulate randomness

in the dynamics, a random action was selected $\epsilon = 5$ percent of the time.

B. Global GPS Graph

To model the GPS, a deterministic simplified transition system was used to roughly model the dynamics in $\mathcal{M}$. The GPS graph can be defined as a tuple $\mathcal{G} = (Z, A, \delta, c, goal)$ where:

- Z is a set of states,
- A is the same set of actions,
- $\delta : Z \times A \mapsto Z$ is a deterministic transition function,
- $c : Z\times, A \mapsto \mathbb{R}_{\geq 0}$ is a transition cost function,
- $goal \subset Z$ is a set of goal states.

V. Solution Approach

Our solution combines value iteration and backwards reachability. Value iteration is an exact method in that it requires knowledge of T and R to solve the problem. It iteratively updates the Bellman backup equation

$$V_{k+1}(s) = \max_a (R(s,a) + \gamma \sum_{s'} T(s'|s,a) V_k(s')) \tag{1}$$

until convergence

A. Algorithm Overview

Our solution to the previously defined problem sought to combine information gathered from the local sensor with the global trajectory planner. To accomplish this, we developed a algorithm that combined an optimal backward reachability motion planner with a value iteration POMDP solver. Our algorithm consisted of the following pseudocode steps:

1) Initialize all transition weights to 1: $c(g,a) = 1 \; \forall z \in Z, a \in A$,
2) Run *Optimal Backwards Reachability* to determine the minimum cost to goal weighting function for each state: $w : Z \mapsto \mathbb{R}_{\geq 0}$.
3) Initialize a local "weighted" MDP $\mathcal{S}$ using the car's current state, a sensor range, and an augmented reward function $\mathcal{R}$.
4) Solve the local weighted MDP $\mathcal{S}$ using value iteration to return a value mapping $Q : S \times A \mapsto \mathbb{R}$.
5) Generate an action a_p by determining the $\text{argmax}_a(Q)$.
6) Initialize a local "unweighted" MDP $\mathcal{S}_u$ using the car's current state, and a sensor range.
7) Solve the local unweighted MDP $\mathcal{S}_u$ using value iteration to return a value mapping $Q_u : S \times A \mapsto \mathbb{R}$.
8) Take the action a_p produced by the policy of the weighted MDP $\mathcal{S}$.
9) For all $s \in G, a \in A$ update the transition cost $c(s,a) = -Q_u(s,a)$.
10) If s_c is not within $goal$, return to step 1.

B. Optimal Backwards Reachability

Optimal Backwards Reachability is a weighted graph search algorithm in which a set of target states are specified as input, and the algorithm produces a weighting for each node in the graph equal to the minimum cost to any target state from that node. Given a graph $G = (X, E, W)$ where X is a set of nodes, $E \subseteq \times X$ is a set of directed edges, and $W : E \mapsto \mathbb{R}_{\geq 0}$ is a positive edge weight function, a reverse graph G' can be defined as $G' = (X, E', W')$ where:

- X is the same set of nodes,
- $E' \subseteq X \times X$ is a set of reversed edges where $\forall e = (x_i, x_j) \in E, \exists e' = (x_j, x_i)$.
- $W' : E \mapsto \mathbb{R}_{\geq 0}$ is a reversed positive edge weight function where $\forall e = (x_i, x_j) W'((x_j, x_i)) = W((x_i, x_j))$.

The solution to Optimal Backwards Reachability on a graph G with target node set $goal \subset G$ is equivalent to the set of *tentative distances* obtained by running Dijkstra's algorithm for all reachable states on G' with initial states $goal$.

C. Augmented Reward Function

In order to inform the local MDP $\mathcal{S}$ using the minimum to cost goal solution w, an augmented reward function $\mathcal{R} : \mathbb{R} \times \mathbb{R}_{\geq 0}$ is defined. Using $\mathcal{R}$, the reward any state action pair (s, a) is $\mathcal{R}(R(s, a), w(s)) = R(s, a) - w(s))$. Variations of $\mathcal{R}$ were tested, namely $\mathcal{R}(R(s, a), w(s)) = R(s, a) - \beta * w(s)^2)$, however many successful results were produced without editing the coefficient or exponent of $w(s)$.

An important challenge of our solution was how to use state weights to formulate the local MDP while also using the solution of the local MDP to update state weights. This circular dependency required the implementation of two local MDPs - one that accounted for state weights in the reward function and one that did not. The first was used to calculate which action to take in the environment and the second was used to update state weights in the global planner, thereby removing the circular dependency.

VI. Results

Animated visualizations of several simulation results can be found on Google Drive [1]. All code implementation can be found on GitHub [2]. Note that all the animated figures found in this paper are found in the folder: "Paper Trajectories". In order to allow for statistical inferences to be made from simulations, we kept environments constant across simulation runs. Four environments can be seen below:

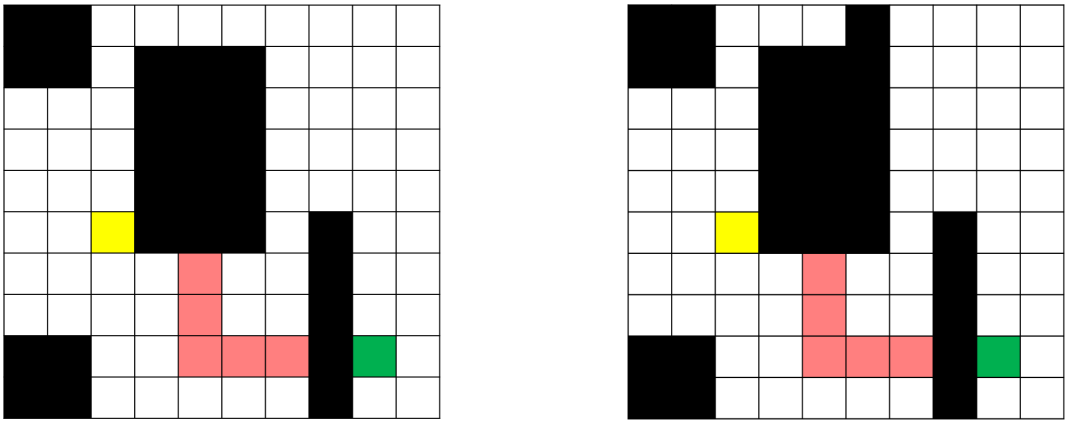

Fig. 3 Environment 1 is on the left and environment 2 is on the right. Note that environment 2 encourages exploration first prior to advancing across the bad road due to the added obstacle at $(6, 10)$**.**

[1] Shared Folder: https://drive.google.com/drive/folders/10-YyVtUeC4cxNw-YL6bYa1yNnvn75zpU?usp=sharing
[2] GitHub Repository: https://github.com/peteramorese/DMUFinalProject

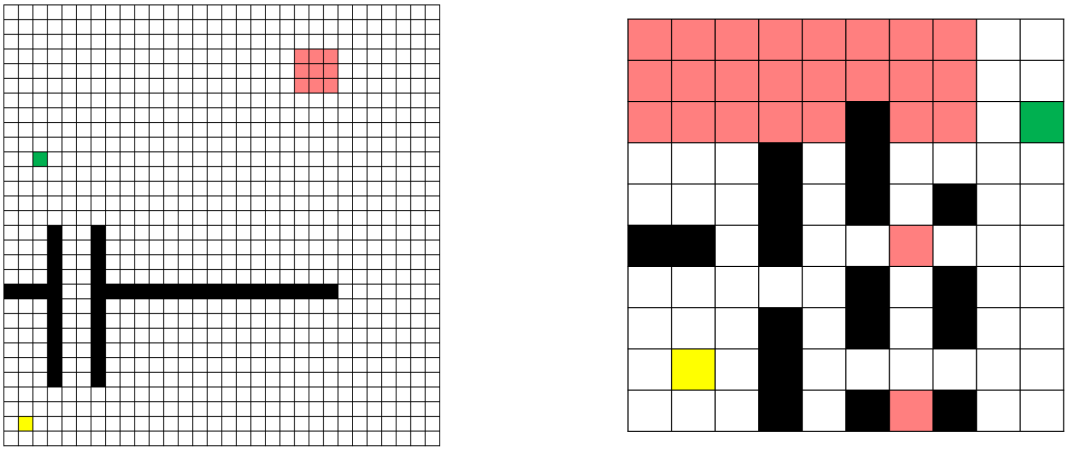

Fig. 4 Environment 3 is on the left and environment 4 is on the right. Note that environment 3 is a 30×30 grid world, rather than the usual 10. Environment 4 is a highly fragmented grid world with the most obstacles of all environments.

Note that the key from figure 1 applies here as well. The yellow square represents the start location, the green square is the goal location, pink squares are bad roads, and black squares are obstacles.

Using 50 simulations in each environment the sensor range was varied while keeping the some parameters constant. The probability of choosing a random action to encourage exploration is represented as ϵ and was set to 0.05, the reward for entering an obstacle was -2000, the reward for entering a bad road was -100, and the discount factor γ was set to 0.75. A high discount factor treats far away rewards with the same weight as rewards in the immediate future. Therefore, a lower discount factor weights immediate rewards higher than future rewards. Below are some trajectories associated with a car with a sensor range of 1.

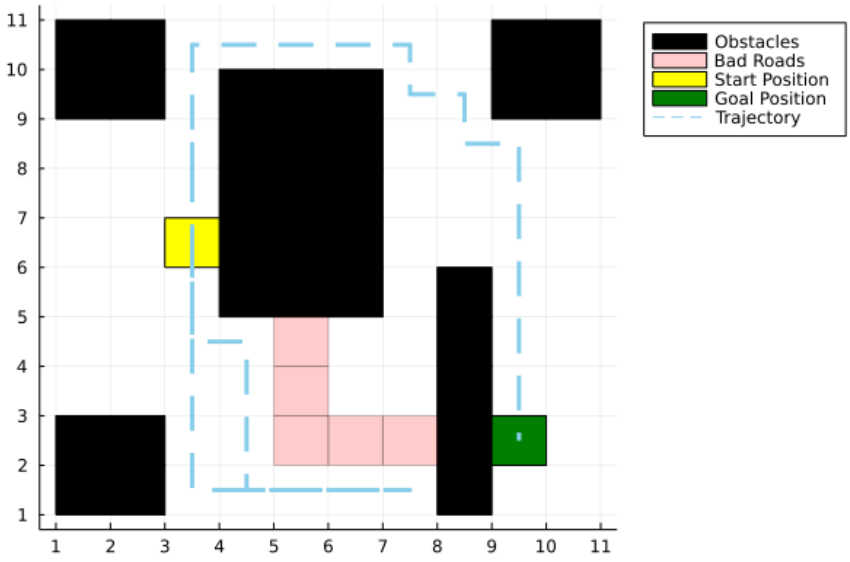

Fig. 5 Trajectory of car in environment 1. Notice the car first travels to $(7, 1)$ then turns around and explores to $(3, 10)$, ultimately finding a route free of bad roads.

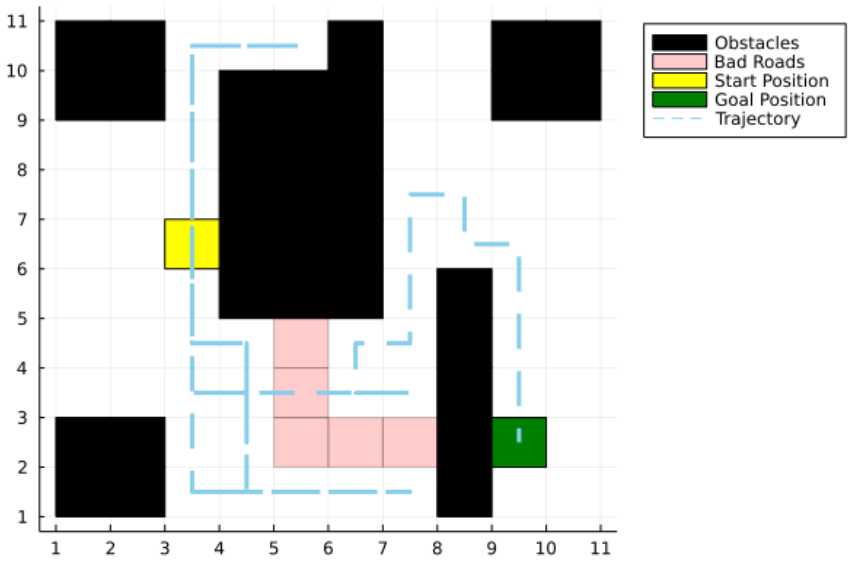

Fig. 6 Trajectory of car in environment 2. Unlike in environment 1 the car must travel over a bad road. The car initially travels to $(7, 1)$ but when it reaches $(5, 10)$ discovers the route is blocked and traverses over the bad road at $(5, 3)$.

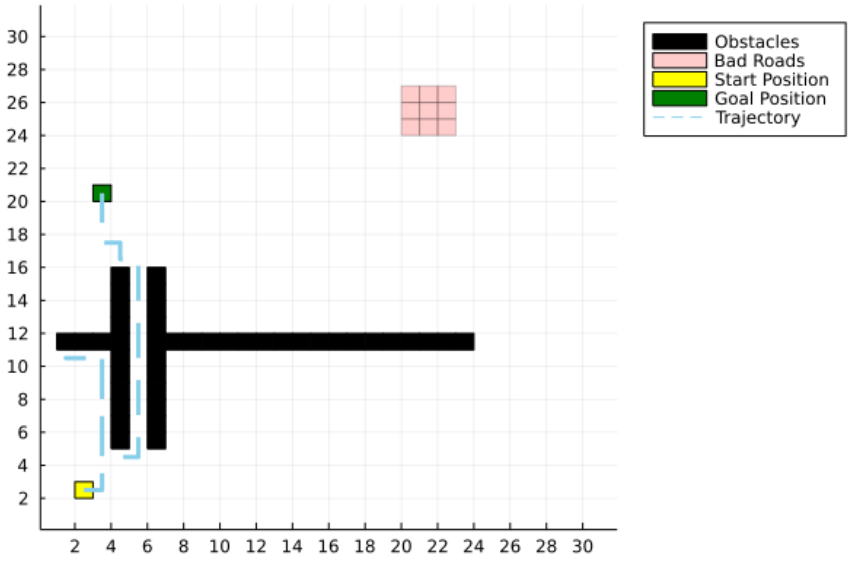

Fig. 7 Trajectory of car in environment 3. The car initially heads down immediately for the goal but finds the corridor after exhausting the most expedient route.

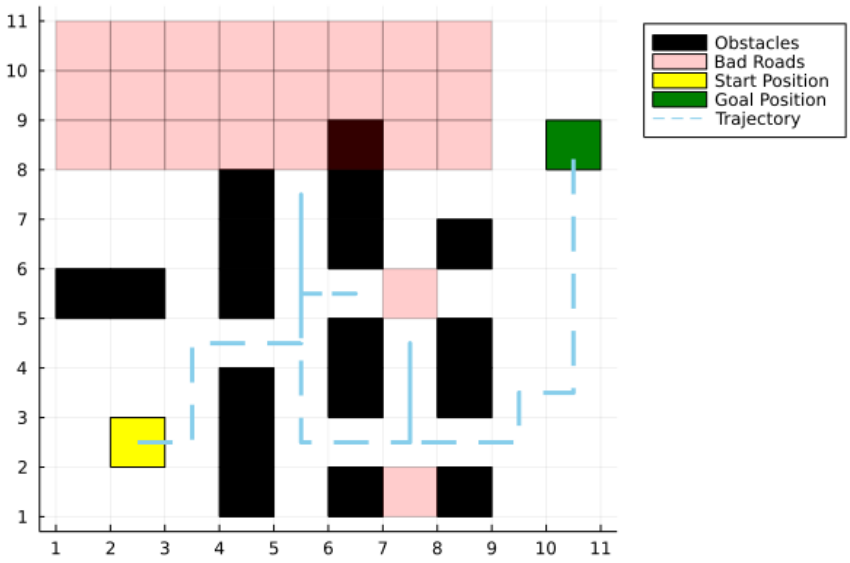

Fig. 8 Trajectory of car in environment 4. The car traverses across many dead ends prior to finding the one open route available to the goal, avoiding all bad roads and obstacles to take the longest route.

A table of the average simulated results from 50 trial runs can be found below:

	Sensor Range 1			Sensor Range 2		
Environment	Average Computation Time (s)	Average Trajectory Length	Average Total Reward	Average Computation Time (s)	Average Trajectory Length	Average Total Reward
1	1.4424	39.68	-867.702	5.809	38.58	-663.606
2	2.4366	73.38	-1786.954	7.0671	47.64	-1316.392
3	1.8192	41.58	-1483.884	7.1515	39.68	-1203.708
4	1.3855	31.64	-1490.882	3.6972	26.48	-604.416

Table 1 Average simulated results of two sensor ranges. Notice that performance improvement is not consistent across all environments. Some environment pathing cannot be improved with added sensor range

We note that as we increase the sensor range the computation time increases, but on average the average trajectory length decreases and the average reward increases. As mentioned above, not all environments can be equally improved with an increased sensor range since not all environments are equally made. In testing with the infinite sensor readings, the agent was able to reach the goal using the most expedient path however, the computation time is much higher. In the case of environment 3 attempted computation time exceeded an hour due to the MDP being too large and the simulation was abandoned. Animated trajectories can be found in the google drive under: "Infinite Sensor Range".

	Infinite Sensor Range		
Environment	Computation Time (s)	Trajectory Length	Total Reward
1	272.843	23	-2.1
2	213.363	15	-101.6
4	228.141	19	-1.7

Table 2 Infinite range sensor computation time and trajectory length results over 1 simulation.

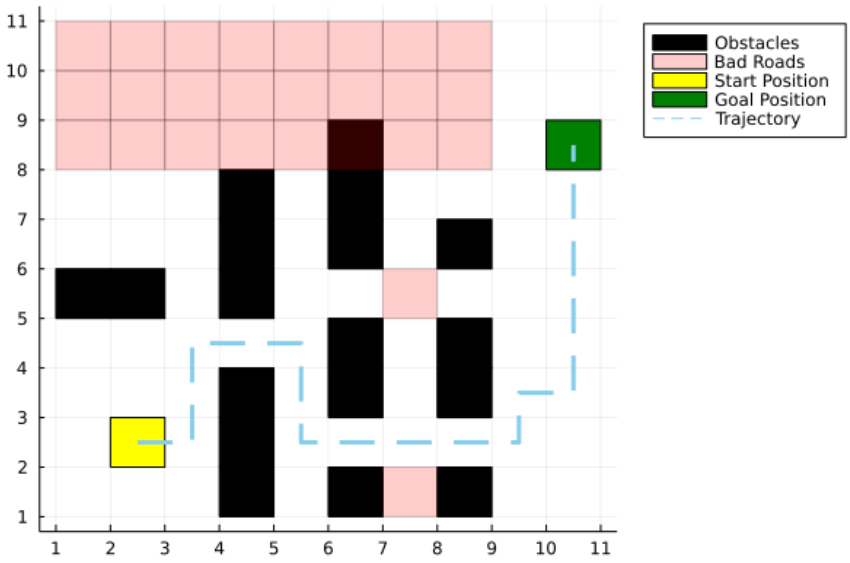

Fig. 9 An example of the agent operating with an infinite range local sensor. Note that the agent proceeds along the least costly path without any exploration.

VII. Conclusion

Navigation in unknown environments is a commonly studied topic and will certainly become more relevant as the world increasingly relies on autonomous systems. The work presented in this report demonstrated a potential method of navigation for a realistic scenario in which the agent was equipped with a global sensor (a GPS) and a local sensor (a LIDAR). Knowledge from the local sensor was used to form and solve a local MDP. The results of this solution were used to update the global planner in an iterative loop that drove the agent from start to goal while avoiding obstacles and costly paths. We made several interesting observations during this project. The first was the importance of the method used to map state costs from global planner to the reward function of the local planner. Often, it is difficult to identify reward functions in the real-world so we utilized the global planner to help define the reward. However, the behavior of the agent in the grid world was highly sensitive to how this mapping was conducted. The combination of the mapping function and the local MDP rewards impacted the how the agent weighed "exploration vs exploitation" and, in some cases, even impacted the agent's decision to collide with an obstacle or traverse a costly path.

We also observed that the agent's behavior was sensitive to our choice of discount factor, γ. We observed that lower discount factors generally led to more desirable results. This is likely due to the fact that the MDP is defined every step the agent takes so there is no need to significantly weight future local states.

In conclusion, we studied the impact of using a local MDP in combination with a global trajectory planner to produce a collision free path in unknown environments. That is, given no information about environment other than the desired destination, we successfully demonstrated that a mobile agent can navigate an unknown environment using value iteration.

VIII. Individual Contributions

Kyle Li contributed in problem adaptation, visualization code for generating the graphics, bench marking code, grid world generation, and was the principal author of the report. Joe Miceli assisted in formulating the problem and developed a large portion of the code used to define and solve the local MDP. Peter Amorese assisted in formulating the problem and also developed the code for the Optimal Backwards Reachability implementation and helped develop code for the planning algorithm and grid world generation.

IX. Relevance of Cited Materials

Reference [2] Both our projects focus on learning from a discretized grid world. We both utilize $Q(s,a)$ when determining its action and state weighting. The approach towards testing varied hyper-parameters in the paper was useful to our own project.

Reference [3], this paper utilizes the notion of hierarchical iterative planning, a core concept employed in our algorithm. The work in [3] looks at synergistically using a low-level motion planner to update edge weights on a high level discrete task planner. The paper proves probabilistic completeness of the iterative planning algorithm. Informing the high-level planner (GPS) by using information discovered in a low level planner (Sensor MDP) is loosely analogous to the synergistic planning framework.

Reference [4] uses an MDP to filter information from a global motion planner when determining actions to take in a grid world environment.

Reference [5], the concept of this paper is very similar to our project where it features a mobile agent with limited global knowledge of an environment but a local sensor that reveals the true states of its surroundings. This paper provided insight that one way to solve issues of exploration was increasing ϵ. Behavior of occasionally entering obstacles using MCTS or value iteration was also supported in both our projects due to the nature of randomly taking actions.

Reference [6] discusses the use of POMDP solvers to guide an agent safely through an environment. This paper discusses use of one offline planner, QMDP, and two online planners, POMCP and POMCPOW. While our report formulates the problem as an MDP, both papers discuss solving while the agent moves through the environment.

X. Release

The authors grant permission for this report to be posted publicly.

References

[1] Lidhoo , P., &; Singal, N. (2022, May 1). How Technology is Transforming the Auto Industry. Business Today. Retrieved May 1, 2022, from https://www.businesstoday.in/magazine/industry/story/how-technology-is-transforming-the-auto-industry-331116-2022-04-25

[2] Maniyar, Y., &; Manoj, N. (2019). Racetrack Navigation on OpenAIGym with Deep Reinforcement Learning. Stanford University AA228/CS238. Retrieved April 28, 2022, from https://web.stanford.edu/class/aa228/reports/2019/final9.pdf

[3] He, K., Lahijanian, M., Kavraki, L. E., &; Vardi, M. Y. (2015, May). Towards Manipulation Planning with Temporal Logic Specifications. Morteza Lahijanian . Retrieved May 2, 2022, from http://mortezalahijanian.com/papers/he2015.pdf

[4] Ali, H., Gong, D., Wang, M., &; Dai, X. (2020, July 9). Path Planning of Mobile Robot with Improved Ant Colony Algorithm and MDP to Produce Smooth Trajectory in Grid-based Environment. Frontiers. Retrieved May 3, 2022, from https://www.frontiersin.org/articles/10.3389/fnbot.2020.00044/full

[5] Contreras, P., Gee, M., &; Knowles, D. (2019). Goal-driven Navigation of an Unknown Environment with Monte Carlo Tree Search. Stanford University AA228/CS238. Retrieved April 28, 2022, from https://web.stanford.edu/class/aa228/reports/2019/final100.pdf

[6] Cash, B., &; Persson, J. (2018). Offline and Online POMDP Solution Approaches for Roomba Navigation. Stanford University AA228/CS238. Retrieved April 29, 2022, from https://web.stanford.edu/class/aa228/reports/2018/final183.pdf

[7] Sunberg, Z. (2022, February 2). CU DMU Materials: Online Methods. GitHub. Retrieved May 1, 2022, from https://github.com/zsunberg/CU-DMU-Materials/blob/master/lecture-notes/070-online-methods.pdf

[8] Sunberg, Z. (2022, March 29). CU DMU Materials: Online POMDP Methods. GitHub. Retrieved May 1, 2022, from https://github.com/zsunberg/CU-DMU-Materials/blob/master/lecture-notes/210-online-pomdp-methods.pdf

[9] Sunberg, Z. (2022, March 15). CU DMU Materials: POMDP Formulation Approximation. GitHub. Retrieved May 1, 2022, from https://github.com/zsunberg/CU-DMU-Materials/blob/master/lecture-notes/180-pomdp-formulation-approximations.pdf

[10] Sunberg, Z. (2022, March 1). CU DMU Materials: Advanced Exploration Actor Critic and Entropy Regularization. GitHub. Retrieved May 1, 2022, from https://github.com/zsunberg/CU-DMU-Materials/blob/master/lecture-notes/150-advanced-exploration-actor-critic-and-entropy-regularization.pdf

[11] Sunberg, Z. (2022, March 1). CU DMU Materials: Reinforcement Learning, Model-Based: Tabular RL. GitHub. Retrieved May 1, 2022, from https://github.com/zsunberg/CU-DMU-Materials/blob/master/lecture-notes/090-reinforcement-learning.pdf

XI. Acknowledgements

We would like to thank Dr. Zachary Sunberg for the wonderful and challenging year.