Goal-driven Navigation of an Unknown Environment with Monte Carlo Tree Search

Polo Contreras
05676100
jcontr83@stanford.edu

Mark Gee
06346053
markgee@stanford.edu

Derek Knowles
06341436
dcknowles@stanford.edu

***Abstract*— A mobile agent, such as a parts-carrying robot on a factory floor, a rover on an extraterrestrial surface or even a cleaning robot in an office building, often finds itself in an environment that it must navigate, moving to a designated target area in order to perform or complete a task. The agent is usually familiar with the general map characteristics of the environment in which it moves, but it may need to avoid small, local obstacles that have been placed in its path. In addition, the fact that the agent's next assigned goal location is unpredictable forces the agent to use online path planning to reach its goal. Our hero (the agent) must be able to use its sensors, assigned goal location, and set of possible actions to dodge the obstacles in the environment and reach the assigned goal location unimpeded. Here we show that Monte Carlo Tree Search (MCTS) can be implemented in order to reliably direct the agent to the assigned goal location.**

I. INTRODUCTION

With mobile robots in common use both in research and industry applications[1], an important challenge in most implementations is navigation in an unknown environment. Unknown and unpredictable environments are hallmarks of nearly any real setting — anything from factory floors to forested areas, planetary surfaces in general or even homes. Even in some areas, like manufacturing, where it could be possible to maintain a controlled environment, it may be far more practical to imbue a robot with the ability to navigate around obstacles; doing so will make the robot more robust, simplify and economize the maintenance of its work space, and allow it to work more effectively with humans. In this project we explore one possible solution to the problem in general, where a robot with limited sensing capabilities must chart a path through an unknown and changing environment in real time.

II. PROBLEM DESCRIPTION

In order to represent this, a virtual agent and environment are created. The virtual environment is based on a large two-dimensional grid world, where the agent can take one of four actions to move through this environment (up, down, left and right). Upon generation, the virtual environment contains a reward that represents the assigned goal location placed at a random location on the map, while various other random locations throughout the map contain obstacles. Entering the space associated with the reward (assigned goal location) is associated with a positive utility for the agent, while entering the space occupied by an obstacle incurs a penalty. The agent will therefore have to navigate through the map to reach the assigned goal location while avoiding obstacles, with the caveat that it is only able to detect obstacles within a limited range of itself (the type of sensors can vary with the specific application: for example, automotive applications readily use cameras in multiple directions, ultrasound, or LIDAR, among others[2]). Our problem setup is depicted in Figure 1.

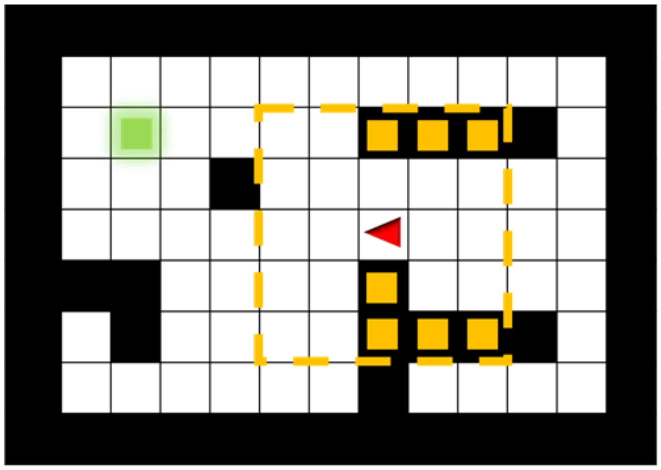
Fig. 1. Illustration of agent (red), goal (green), obstacles (black), sensor range (dotted line) and detected obstacles (yellow).

III. SOLUTION

Because the agent is in a largely unknown environment, the position of both the goal and obstacles are randomly generated, and the sensors of the robot are unable to detect the placement of obstacles outside of a specific range, the agent uses an online method to decide the best possible move at each step. In order to balance the tendency of the agent to find the shortest path possible towards the goal and the requirement for the agent to avoid obstacles as much as possible, the agent chooses its actions using the MCTS algorithm[3]. This algorithm uses an expanding state representation to determine the best next action (Figure 2). MCTS has been implemented in many artificial intelligence and game theory applications, including in problems with

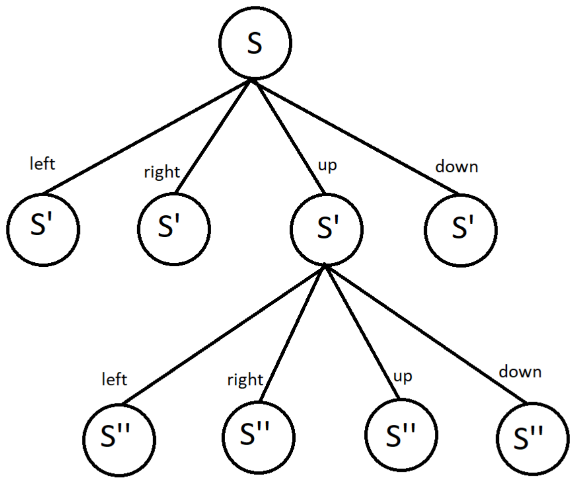

Fig. 2. Graphical representation of MCTS as used in this algorithm, with the agent considering the effects of its actions at future states (S', S") from the current state (S) with each available action. The default policy is used to analyze each terminal node.

large state and action spaces such as the game of Go[5]. We chose MCTS because it can handle large state and action spaces, and also has the benefit that its complexity does not grow exponentially with the problem decision horizon unlike other online methods. The default policy used by our modified search algorithm is to direct the agent in the shortest path towards the goal, while the updates generated at each point in the decision tree during the course of decision making are performed with the express intent to identify obstacles and avoid them. Since the discount from taking additional steps to reach the assigned goal location is small relative to the big reward associated with arriving at the goal and the big penalty of hitting an obstacle, the search will give priority to paths that circumvent obstacles in the environment. This simulates a situation where it is significantly worse to run into an obstacle than it is to incur the additional cost of fuel or power caused by a longer route.

Because the agent knows the position of itself and the goal within the map, the default policy of the agent is to move towards the goal in as direct a route as possible. This default policy is used at each leaf node in the course of MCTS. The agent then uses the MCTS to evaluate each potential future step on its way to the assigned goal location. In a situation where there are no obstacles between the agent and the goal, the discount associated with an additional action will cause the agent to select the action (out of the four available) which brings it closest to the goal. Through this default policy, the agent gradually seeks out the goal in the environment, prioritizing at each step the action that makes for the shortest path.

If the sensors of the agent detect an obstacle, MCTS will run specifically to select between the four actions that the agent has available. Because of the aforementioned penalty associated with passing through an obstacle, the agent will prioritize actions that result in avoiding an obstacle, moving in the general direction of the goal but simulating future actions in order to dodge obstacles as it moves. Notably, after each step, the agent maintains some of the previously computed values of the expected utility as it proceeds to the next step, avoiding the unnecessary computations associated with regenerating a completely new set of expected utilities after each move and updating the values instead.

It is worth taking into account that the algorithm, as described above, makes a very strong implicit assumption: space that is outside the range of its sensors is assumed to be free of obstacles. While this means that the agent will prioritize moving around obstacles instead of through them, this can create a potential problem situation where the agent can unknowingly be caught in a loop as it tries to maneuver towards the goal. For example, if the space between the goal and agent contains a specific arrangement of obstacles that causes the agent to move between multiple positions corresponding to local optima (always selecting the same action at each one of these positions), the agent may find itself constantly circling the goal or moving back and forth across the obstacles as it forgets the presence of any obstacle that it cannot see. This could be solved by making the agent remember the locations of obstacles it has encountered previously, but this would make it necessary for the agent to expand the decision tree as it explores its environment, vastly increasing processing power requirements. Instead, a degree of randomness is introduced into the actions of the agent in order to place more emphasis on exploration. Increasing the amount of randomness of the agent's actions increases the amount of exploration.

While the agent will regularly run a search in order to select the best action among its current options, there is a certain probability that the agent will instead ignore this and select an option at random. This allows the agent to explore its environment further and serves as a solution to a situation where the agent is caught in a loop. If there was a situation where the agent could move between several spaces in a fixed sequence while attempting to find the optimal path, it would eventually move in an unexplored direction and find an alternate route. Once the agent reaches the goal, the map can be reset to provide a new configuration of the goal and obstacles (the agent's internal representation of the map, represented by the utilities calculated in its search, will also be cleared). This serves as a representation of the agent needing to move through an environment that has changed, such as if each goal indicates the location of its next task and the time taken to carry out a given task is long enough that the configuration of obstacles will have changed.

While this generalized approach could be used in a variety of different situations, there are some limitations. In its current form, the algorithm makes its decisions based on the current layout of the goal and obstacles, and is not capable of reliably predicting changing conditions in the environment. For example, if some of the obstacles were to change position as the agent moved towards the goal but were not in range of the sensors of the agent as they moved, the agent would wholly ignore them and would not consider the possibility that they might impede a selected path. This might cause the agent to encounter an obstacle twice, for example. There are several potential solutions for this, which can vary somewhat with the nature of the specific problem being solved; with some knowledge of how obstacles are likely to move, for example, the agent can be given a method to predict the future positions of obstacles it detects.

It is also worth considering that this algorithm relies on the assumption that the agent is capable of reliably establishing its position within its working environment. This can be accomplished a variety of ways, such as with GPS on the surface of Earth or a similar, localized radiolocation system in a factory, but the specific implementation depends on the specific application. In a very large or a continuous area, this algorithm can still be used without modification by discretizing the area in real space (not necessarily in regular intervals, depending on the application) and using nearest-neighbor local approximation[4].

IV. IMPLEMENTATION

The described solution was implemented in Python with the PyQT framework[6] for the graphical representation. Readily manipulating the values required in order to quickly make decisions (a practical necessity in online decision making), Python also allows for efficient representation of the environment containing the agent, reward and obstacles, as well as the random generation of obstacles to represent the changing environment that the agent must navigate. Finally, implementing a graphical framework allows us to form a visual representation of the agent navigating the environment in real time, both for debugging purposes and as an intuitive demonstration of the agent's online decision making capability.

As a representation of the space in which the agent moves, the agent, obstacles and goal are contained in a three-dimensional representation where they are different-colored blocks placed on a surface grid. In this grid, the agent is represented by a red cylinder, the goal of the agent is represented by a green cylinder and two different colors are used for blocks representing obstacles: grey for obstacles that the agent cannot currently detect, and yellow for obstacles of which the agent is currently aware. Also included in the virtual space is a set of lines representing the current paths being considered by the agent in the course of its exploration, giving a real-time visualization of the combination of MCTS and the selected default policy in searching for and reaching the goal. The virtual environment

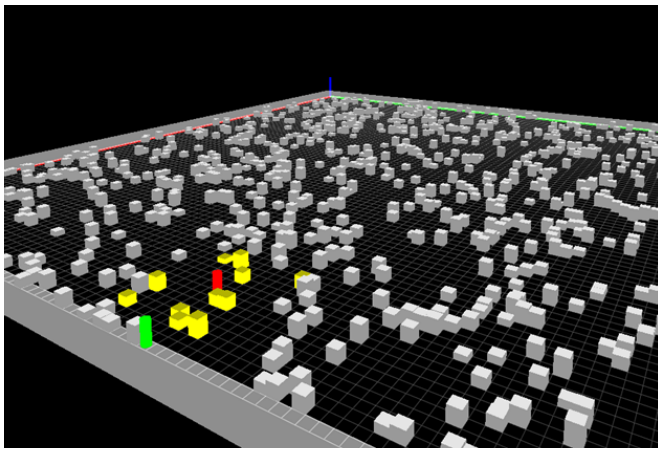

Fig. 3. Three dimensional virtual space, with agent (red), goal (green), unseen obstacles (grey) and detected obstacles (yellow).

thus serves as a combined representation of the physical space in which the agent finds itself and the agent's process of decision making as it navigates this environment.

The main feature of the user interface for the solution is the three-dimensional representation of the virtual space in which the agent resides and operates (Figure 3). The representation provides a live depiction of the agent maneuvering through real space in its attempt to complete tasks, with two important functions available to the user: the ability to change the perspective within virtual space, allowing the user to inspect the agent's maneuvering through space and around obstacles (useful if the agent is, for example, behind an obstacle from the user's perspective) and the ability to customize the parameters of the virtual space, such as the size of the space, the number of obstacles and the method of exploration (MCTS, direct or random policies). This allows the user to evaluate performance of the MCTS algorithm in spaces of varying relative size and compare it to the default or random policies.

In this specific implementation, the location of the goal and the layout of the obstacles are randomly generated, and the user does not set any of their locations. In some applications, however, this same algorithm could be used in a situation where a user manually set the location of the next goal of a mobile robot (for example, an extraterrestrial rover being monitored by telescope). Using this same algorithm, the agent would automatically move across the surface while dodging obstacles in its path until reaching the goal, then wait for the next position of the goal to be assigned.

Full source code of our implementation code can be found on GitHub [1].

[1] https://github.com/betaBison/robot100.

V. Results

In accordance with our predictions about the high effectiveness of the algorithm, the agent persistently seeks out the goal in repeated simulations while reliably avoiding obstacles, generally without regard as to the positions of the goal or obstacles (Figures 4 - 7). While some limitations are visible in its performance, these are inherent to the capabilities of the agent (the range of its sensors, for example) and correspond to the compromises that must be made when developing an agent capable of moving throughout real space. Being able to move through the simulated space while tracking its own internal representation of obstacles encountered, the agent thus showcases a generalized representation of one form of combined exploration and exploitation. The default policy of the agent hinges on exploitation, driving the agent directly towards the goal, while the decision process selected using the sensor input data combines elements of both exploration through sensor data and exploitation through the prioritization of steps that avoid obstacles.

When the agent is simulated to traverse the space in real time, the limitations of the agent become obvious. While the default policy has a tendency of moving the agent directly towards the goal, in certain layouts this can be a disadvantage; the agent will ignore obstacles until it moves close enough to detect them, which means the path taken is frequently longer than the optimal path for a specific map configuration. We found that this can be offset by giving the sensors a larger range, but the depth of the MCTS would need to be increased to suit this increased range, which thus increases the computational requirements of the agent. This balance should be set in accordance with the requirements of a specific application and detection range of its sensors.

Further, we found that MCTS performs poorly when there are a large amount of obstacles; this is expected because it may not explore enough possible paths to find a valid obstacle-free solution. To improve performance in such a situation, the depth of the search can be increased, or the amount of randomness in the agent's actions can be increased to increase the proportion of exploration. This was verified in our implementation by changing the appropriate parameters.

Overall, the agent serves well as a general representation of an automated decision making process capable of adapting to uncertainty in its environment, usable in a wide variety of applications in mobile robots.

VI. Conclusions

As an effort to develop a generalized algorithm useful in a variety of different contexts, this particular implementation showcases both the strengths and weaknesses of MCTS as well as those of online decision making in general. The agent in the simulated environment is capable of adapting to the conditions in its environment with minimal guidance

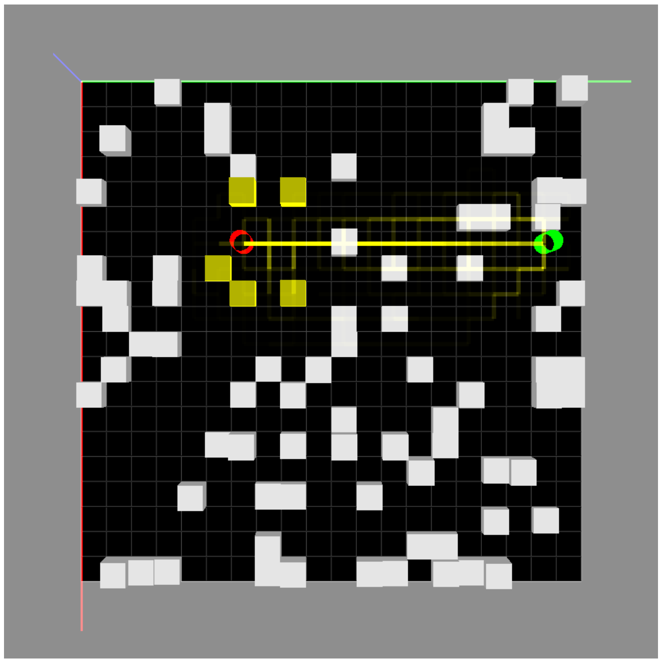

Fig. 4. No observed obstacles in path from agent (red) to goal (green). Yellow lines show paths explored in MCTS, with brightness correlated to expected utility.

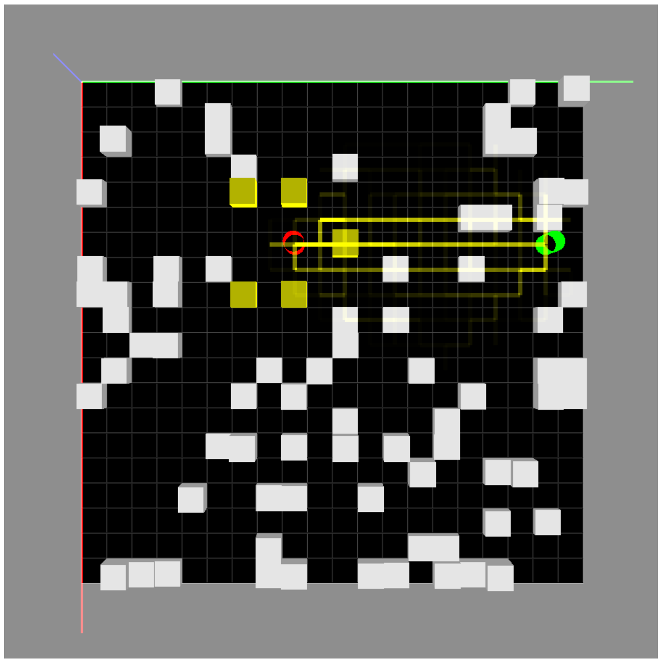

Fig. 5. Observes obstacles in path and reroutes.

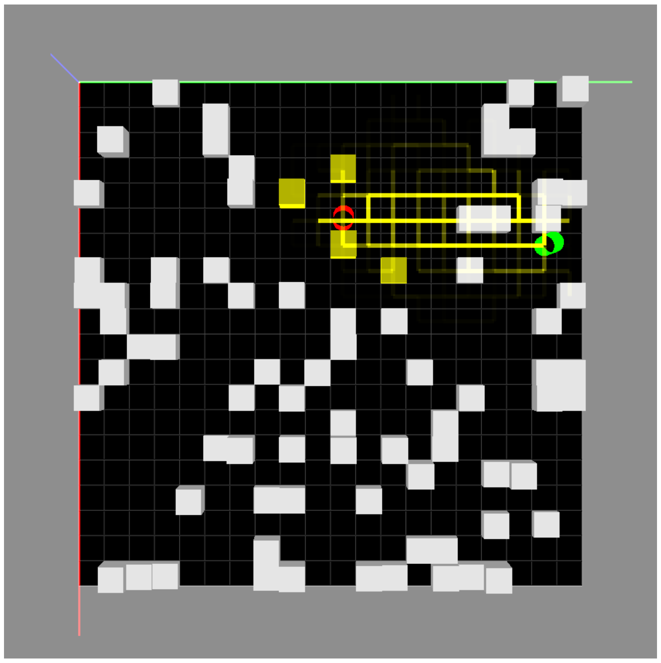
Fig. 6. Reroutes to avoid obstacles.

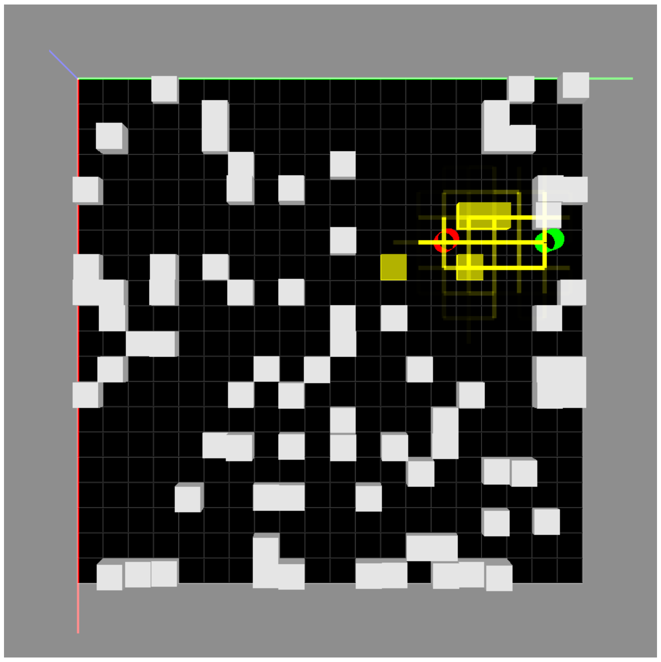
Fig. 7. Clear path to goal.

from humans. This is ideal in situations where the agent is stationed in a dynamic and unpredictable environment and constant directions from humans as to its movements are not feasible or desirable. However, the agent must reconsider its actions at every step, raising the threshold required for the computational power that the agent carries on board as well as potentially increasing the amount of time required between steps. Although the agent is able to use information learned from its environment in the selection of its next action, the capacity of the agent to make adjustments to changing conditions is limited, and it cannot use all of the information that has been processed previously once its circumstances have changed.

This particular implementation also showcases, albeit to a lesser degree, the viability of including expert knowledge not just in the description of the problem but also in the implementation of the solution. As the situation being simulated here is the displacement of a mobile agent across a surface, the knowledge that the terrain can be traversed, unless there is a detectable obstacle, allows the implementation of a default policy designed for this case. The default policy takes the agent in the direction of the goal and is an improvement over, for example, random movement, but still requires modulation by value calculations in order to select the best decision. Each decision, therefore, is made based on a combination of both the expert knowledge programmed in by the human designer and the knowledge that the agent acquires as it moves around its environment.

As a final note, although the capability of the agent to move in four directions across the grid world is rather abstract and might not be directly reproducible by an autonomous agent in real space, in such a situation this algorithm can serve as part of a hierarchical decision making process. This algorithm could still be used even if the agent were, for example, a wheeled vehicle with traditional Ackermann steering (such as a passenger car or truck), a vehicle with differential steering (such as a tractor or skid-steer loader) or even an air cushion vehicle; designating the location of the next step of the vehicle based on knowledge of the goal location and the sensor input, subordinate functions can then be called specifically to control the running gear in order to perform the next selected step. The algorithm can therefore be used with a variety of different control schemes, serving as a connection between the process that sets the goal location for the agent and the control scheme that directly affects the position of the agent in the environment.

VII. Individual Contributions

Polo Contreras contributed to the adaptation of the problem to a mathematical description (including the definitions of the goal and obstacles in terms of utility), the design of the algorithm to solve it (including the selection of a nonrandom default policy), and is the principal author of this report. Mark Gee contributed to the design of the problem, the algorithm to solve it and the implementation in code of the solution (including definition of the object classes and the default policies, optimization of the software stack, the decision processes and the graphical representation). Derek Knowles contributed to the representation of the problem, the design of the algorithm to solve it (including the selection of the search algorithm for selecting among possible paths) and the implementation in code of the solution (including the virtual, three-dimensional representation of the agent and environment).

REFERENCES

[1] MarketWatch, 2019, ”Autonomous Mobile Robots Industry: 2019 Market Research with Market Size Growth, Manufacturers, Segments and 2024 Forecasts Research - MarketWatch”. [ONLINE] Available at: https://www.marketwatch.com/press-release/autonomous-mobile-robots-industry-2019-market-research-with-market-size-growth-manufacturers-segments-and-2024-forecasts-research-2019-10-22. [Accessed 06 December 2019].

[2] Alpaydin, E., 2016, “Machine Learning: the new AI,” Where do We Go from Here? E. Alpaydin, The MIT Press, Cambridge, Massachusetts, pp. 149.

[3] Kochenderfer, M. J., 2015, “Decision Making Under Uncertainty: Theory and Application,” Sequential Problems. M. J. Kochenderfer, The MIT Press, Cambridge, Massachusetts, pp. 102-103.

[4] Kochenderfer, M. J., 2015, “Decision Making Under Uncertainty: Theory and Application,” Sequential Problems. M. J. Kochenderfer, The MIT Press, Cambridge, Massachusetts, pp. 94.

[5] Google Deepmind, 2016, ”Mastering the Game of Go with Deep Neural Networks and Tree Search”. [ONLINE] Available at: http://airesearch.com/wp-content/uploads/2016/01/deepmind-mastering-go.pdf. [Accessed 06 November 2019].

[6] PyQt - Python Wiki. 2019. PyQt - Python Wiki. [ONLINE] Available at: https://wiki.python.org/moin/PyQt. [Accessed 06 December 2019].