Extraterra Robotic Mining Automation

An exploration of motion planning concepts

Jay Dee M. Germer
University of Utah
College of Engineering
Electrical & Computer Engineering
u0673032@utah.edu

Braxton G. Smith
University of Utah
College of Engineering
Mechanical Engineering
braxton.smith@utah.edu

Justin D. Stucki
University of Utah
College of Engineering
Electrical & Computer Engineering
justin.stucki@utah.edu

***Abstract*—This document details a new automated algorithm for the Mars Rover used by the University of Utah Mars Rover Team. The team had previously used manual operation of the rover for annual competitions. The new algorithm uses sensors to examine the environment, and calculate height and slope for all locations on the map, as well as the location of obstacles and impassible locations. Dig sites are assigned beforehand and will be prioritized according to their location and proximity to other dig sites. Path planning is accomplished with Policy Iteration that uses varying degrees of stochasticity. The rover will traverse across Regolith-covered terrain and pick the best paths to travel while optimizing time and not getting trapped by previous dig sites. Dig samples are returned to a home depository, where the rover is stationed and begins its path planning.**

***Index Terms*—Mars rover, Regolith, motion planning, Policy Iteration, Value Iteration, A* Search, stochasticity, dig sites, slope map, obstacle avoidance.**

I. Introduction

The University of Utah has participated in the National Aeronautics and Space Administration's (NASA) Robotic Mining competition in recent years. A team of students control a rover that traverses around a small enclosed area which simulates Mars. The goal is to dig up as many samples of regolith as possible and return them to a depository within a given time limit. The dig sites are assigned beforehand, so the team knows where to send the rover. Points are awarded in the competition based on the number of dig sites visited, the amount of regolith returned to the depository, the amount of automation used in the rover, and several other categories.

Our focus is to automate the process of mining for regolith. This is a category that the Mars Rover team hasn't used in the competition before, mostly due to the complexity of the automation required. The things to consider for the automation is the speed of the rover, the slope of the terrain, and the location of obstacles. Most of the automation will use concepts covered in Motion Planning.

There is an abundant amount of uncertainty in the rover model. The rover turns using a method called "skid-steering", where it has wheels on one side of the rover move in the opposite direction from the wheels on the other side of the rover. This causes the rover to spin in place, but the center point of the rover doesn't stay in the same location as it is spinning.

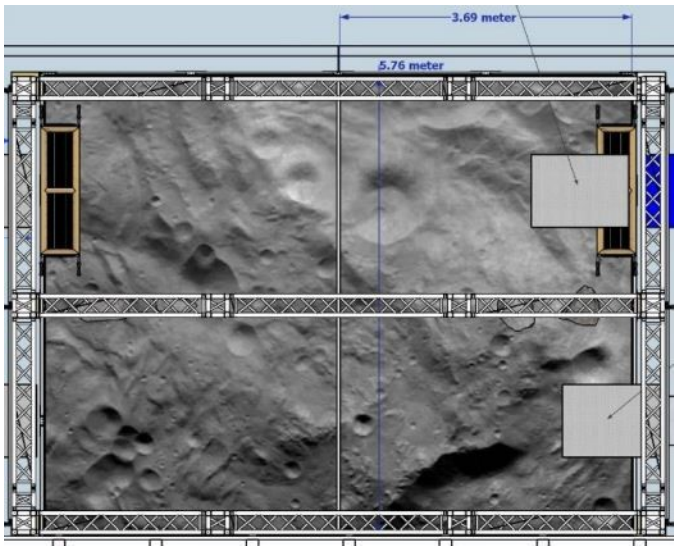


Fig. 1: Competition field as specified by NASA

More uncertainty is related to the terrain. Regolith is very loose composite material, and it can cause the rover to slide and go off-course. The change in slope on the terrain can affect the actual direction the rover moves. The rover will be pulled down by gravity as it goes up steeper terrain, which would affect the heading of the rover.

Implementing a stochastic model was deemed appropriate to correct the uncertainty in the rovers actions. We used a transition model that took into account the error when the rover unexpectedly changed its course. The probability of going in a given direction is inversely proportional to the slope of the terrain in that given direction. If the terrain is flat, the probability of going straight is very high. If the terrain is steep or uneven, the higher probability will be shifted to the lateral directions of the rover.

Since uncertainty would cause the rover to go off course, it would make sense to use Policy Iteration. Unlike other planning methods, Policy Iteration would allow the rover to have an action to follow for every location on the map. If the rover slides off course, it would know how to correct its path and reach its goal. This method of planning isn't always the fastest or the most optimal, but it prevents the robot from getting lost or trapped. This method of planning proved to be more robust than others, but with more cost due to handling exceptions.

Establishing rewards for given locations is the last thing that makes the policy work really well. The rover is drawn towards a positive reward located at the goal. The rover will avoid a select number of locations with negative rewards, such as obstacles and steep slopes. This better ensures the rover will go towards desirable locations that lead to the goal.

II. Related Work

There is literature that relates to robots mining or moving on extraterrestrial surfaces. Two articles in particular summarize the concepts that are covered in this design project. The first covers the autonomous control of mining vehicles using absolute navigation and reactive navigation, which are simple, robust, and low cost.

Absolute navigation involves the robot using localization to examine its current location and establish a real-world coordinate system. In order to travel from one location to another, the robot would follow markers, such as a dotted line, along a path. Policy Iteration best represents this idea, since there are ”arrows” or ”directions” marked across the environment that tell the robot exactly where to go. [5]

Reactive navigation is how the robot reacts to something in its immediate environment using sensors and actuators, and does not require a lot look-ahead planning for faster computations. This concept is incorporated into the transition model used for the rover. The height and slope are considered at every location on the map. The slope is examined in every direction, and allows the rover to determine if a path would become impassible. Probabilities are assigned for each direction based on the intended direction, and inversely assigned on the slopes of the intended direction and its adjacent directions. This is to prevent the rover from traversing along paths that have volatile terrains.

Another research article cited was in regards to mobile-robot navigation and getting complete coverage of an unknown environment. Initially this article said to send the robot around the entire map and plot out any obstacles. This would prove to be ineffective for the time limit in the rover competition, since it would take too long to drive over the whole terrain. However, it mentions using critical point that determine where the rover should, or should not, travel to. [3]

These critical points were initially places where the obstacles would begin and end. In this case, the critical points are similar to how slope changes on the map. If the slope is too high, then the location is avoided, or maneuvered around if the rover reaches it. These ”critical points” are seen by the rover’s decision-making process, which will consider traveling across them more, or less, than other directions. The critical points will be marked with rewards, where positive rewards are the dig sites and negative rewards are obstacles and steep slopes. This prevents illogical behavior of the rover. In fact, it is able to make some decisions similar to a human where it avoids doing anything too risky, such as going up a steep slippery hill. Even if the rover initially does this, it will quickly correct itself.

III. Methods

A. V-REP

The Virtual Robot Experimentation Platform (V-REP) by Coppelia Robotics has been our chosen simulation tool for this project for it’s built-in physics engines, 3D interaction, and compatibility with Python control scripts. Landscape is modelled in V-REP using the heightfield import tool interprets Red-Green-Blue (RGB) data from a color terrain image into highs and lows across the x-y plane. By configuring the heightfield to be responsive, it becomes a surface on which a vehicle can navigate. In addition to terrain, primitive shapes were added to indicate static dig sites and the designated dump site. These are not interactive but can be called upon during simulation to relay their locations. The final scenario also reflected less friction coefficients ($\mu_s = 0.5$ and $\mu_k = 0.08$) due to lower gravity ($g_m = -3.711 ^m/_{s^2}$) found on Mars. Finally, the Robotnik Summit-XL was brought in as an acceptable stand-in for the University of Utah’s skid-steering front-loader robot.

B. Grid Map

1) Discrete Map: The grid used in this project keeps track of the discretized terrain, dig locations, and dump site via queries to V-REP. Discretizing the terrain in Python is done by converting the RGB value of a pixel and converting it to it’s hexadecimal equivalent. After the whole image is imported this way, the entire array can be scaled to correspond to the V-REP scaling. Figure 4 shows an example of the various normalized terrain height layers read in from an RGB image. Because an image isn’t difficult to draw, various terrains can easily be generated with the intent to test certain situations.

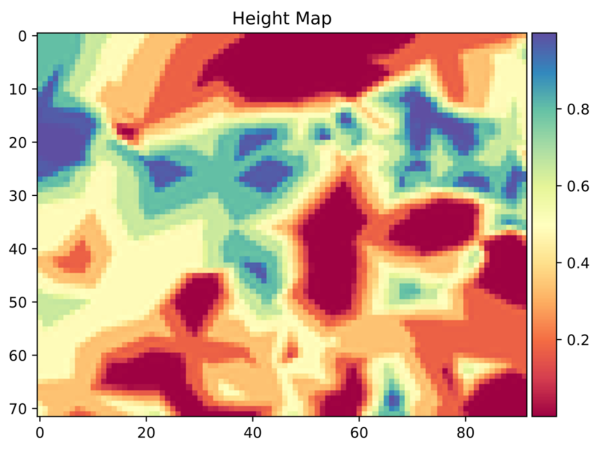
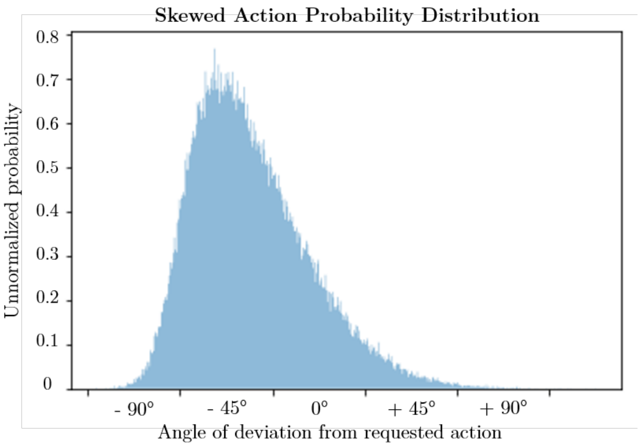


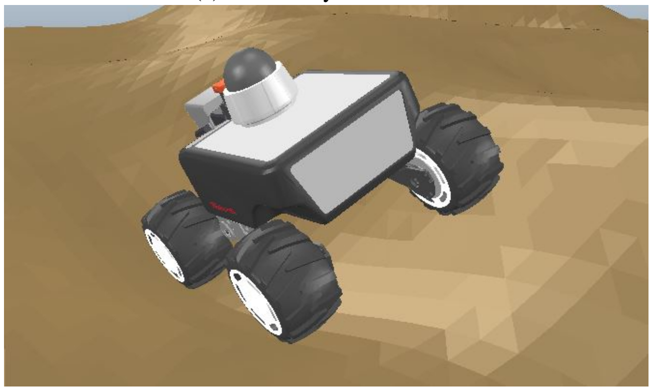
Fig. 2: Height map of competition field at 80mm resolution

2) Transition Model: Uncertainty is a large factor in this project. The skid-steering of the Summit-XL is inherently difficult to plan with because the dependence on friction causes uneven forces. The result is a stochastic lateral sliding that's difficult to predict compounded by sliding due to low friction in the regolith. With so many variables outside of the program's control, a Markov Decision Process (MPD) is the best plan of attack. Instead of letting uncertainty ruin planned outcomes, action choice are made to deal explicitly with uncertainty [4].

The first iteration of transition model was very simple and provided a uniform selection from a static Gaussian distribution across five directions of travel: forward (desired direction) and two directions deviating to either side. After some refinement, an improved transition model was generated using the cross product of the rover's heading vector and the terrain slope vector. The magnitude of the cross product served as the skew parameter in a skewed normal distribution. Finally the skew curve is divided into bins corresponding to the desired action and deviant actions, which are normalized and used as the action probabilities.



(a) Probability Distribution



(b) Rover Pose in Sloped Terrain

Fig. 3: Transition model reaction to uneven terrain

3) Terrain Slope: Slope is an integral part of our transition model. In order to calculate the slope, an 2nd-order[1] "rates" object is defined for every action direction in GridMap. This looks at the forward and "backwards" neighbors given an action and calculates difference in height, divided by the resolution or distance between states (see Equation 1)

$$\frac{h(s) - h(s')}{r} \tag{1}$$

for the "first" order rate. Subsequent orders use the rate values of the prior order rather than the height difference.

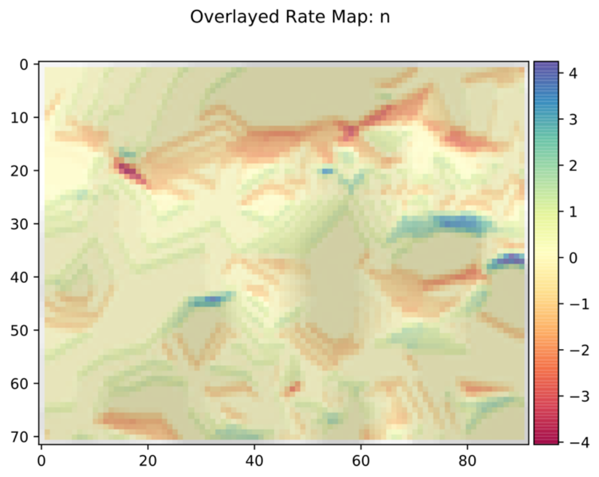


Fig. 4: Slope map of competition field at 80mm resolution

C. Dig Site Order

Fundamentally the dig site ordering problem is akin to the travelling salesman problem and as such is an NP-Hard problem. Our naive solution does not reduce the complexity and blows up as the number of dig sites increases. In addition, the branching factor given the environment is quite large and for our mining map it was unable to finish given an overnight run. Thus, we reduced the map complexity and ran the algorithm on randomly generated maps of a much smaller dimension.

The algorithm itself first generates a simplified environment by converting the heightmap into an occupancy grid of obstacles via a threshold of $\pi/7$. A bit of image manipulation via the library skimage is then used by first eroding small connections out and then diluting the result a bit to expand groups. OpenCV then takes the result and finds polygonal obstacle shapes. These shapes are fed in to an environment file that PolygonEnvironment from Project 2 can read for collision detection. Finally, the library networkx is used to create a grid of free-space, dig sites and the dump site.

To find the shortest total path, we look at every permutation of dig site order and generated a summed path by using A* [1]

[1]We only use the first order values but included n-order for completeness. In addition, more powerful transition models should take these higher orders into account but we did not get that far

from the rover's start position to the first dig site[2], from the first dig site to the dump location, dump location to the next dig site and back to the dump location. So on and so forth until every dig site has been checked. After checking every permutation we go with the dig site ordering that leads to the shortest total path length. Orderings that lead to isolation of other dig sites are automatically skipped out when A* fails.

D. Program Loop

1) Policy Iteration: Autonomous driving has a lot of stochastic factors that make navigation difficult. For example, imagine that the rover was instructed to follow a line to the destination, but a slope caused it to slide off the line or the skid-steering didn't grip immediately and the turn was insufficient. Over long distance, this would lead to divergence from the computed path, which is disastrous on such a high-stakes project with no way to intervene. Instead, having a valid plan at every possible state was a much safer approach.

The Policy Iteration (PI) algorithm computes the best action to take based on a calculated value for that state. This value describes how good it is to be there, and incorporates both the immediate value and the long term benefit of that state. From that state, the rover can take the indicated best action to move to the next highest value within reach. This is the concept of Dynamic Programming and is fundamental to the way PI works. PI starts with a reward/penalty distribution based on the desired dig site as a reward and risky slopes as penalties. A random preferred action is also assigned to every state (Equation 2 [2]). PI has two basic steps: Policy Evaluation and Policy Update.

$$\pi = \pi' \tag{2}$$

$$V_pi = R(s, \pi(s)) + \lambda \sum_{s' \in S} T(s, \pi(s), s') \cdot V_\pi(s') \tag{3}$$

$$\pi'(s) = arg\ max_a\,[V_pi] \tag{4}$$

The Policy Evaluation step iteratively calculates the value of all states based on the value of the adjoining states reached by taking the assigned action. Equation 3 [2] states that the value of the current state is equal to its initial starting value plus the discounted sum of values of neighboring states. This sum accumulates all the reachable values from the transition model's action distribution, multiplied by the likelihood/probability of moving to the corresponding state. This means that a state surrounded by high-value states will obtain a high-value for itself as well.

The discount factor, λ, is used to ensure convergence. Without it, the values would continually add upon themselves. The discount factor also grants a sort of resolution control with preference to computation time versus complete convergence. The desired outcome of this iterative process is to distribute the effect of rewards and penalties to throughout the state space until the change of values from one iteration to the next is less than some small tolerance. Using only the preferred action for each state, the values converge and the results are evaluated. Looking over every state, if a different action leads to a better value, the preferred action is updated for that state in the overall policy (see Equation 4 [2]). This is the Policy Update. Once the same policy goes one iteration without changing, the optimal policy has been obtained.

2) Rover Control: Maneuvering the rover is accomplished by switching between turning and progressing until the goal is reached. As mentioned before, Policy Iteration was the chosen planner because of its ability to provide a valid action no matter when the rover ends up. The gradient of values produced by PI also creates a sort of flow for the rover to follow. From any state, the rover looks up the best action corresponding to its current state and orients itself to that direction. Once the correct heading is obtained, the rover proceeds forward until it reaches a new state. Then it orients itself according to the policy of this state and proceeds. This turn-drive procedure repeats until the rovers state matches the dig-site.

NASA's competition limits excavation to 10 minutes. From the starting location, the rover computes a path to the first dig site. Upon reaching the site, regolith is collected. Without the ability to collect from more than one dig site, the rover then plans a path the competition collection bin. The rover makes trips between dig sites and the dump site until time runs out.

IV. Experiments

A. Height Maps

The first area of experimentation involved getting a useful heightmap in into the Python script. The first image attempted for this project contained a modest 369 x 288 pixels. This resulted in a surprising 106,000 states. The first series of Policy Iteration required over an hour to compute on a modern Intel Core i7. Attempting to simplify, re-sizing a high-resolution image combines a cluster of pixels into one average pixel. This loss of resolution increases the distance between states across the terrain but significantly reduces the convergence time throughout the planning algorithms.

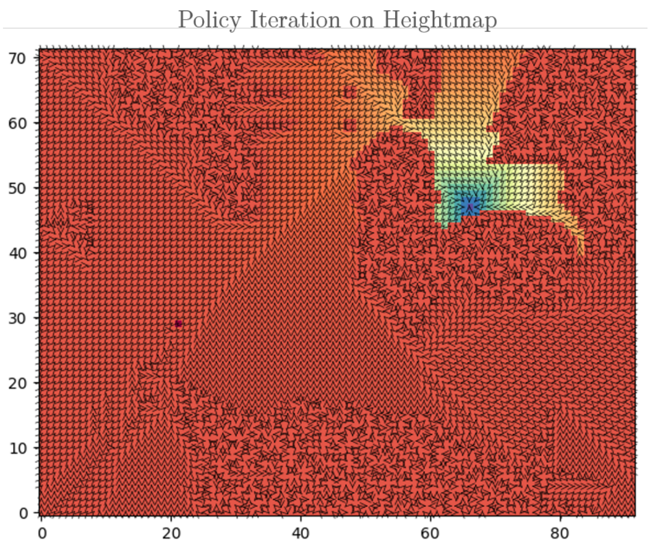
B. Obstacle Identification

The next improvement that this program experienced was the way that obstacles were defined. At first, obstacles were simply defined as terrain which passed a specified height threshold. This only works if the terrain is all flat with the exception of pillars jutting out of the ground. If the entire terrain were a gradual slope which gains height, everything would become an obstacle. Instead, the map gets evaluated to determine the slope from a state going in each of the eight directions. This would later be used to help the rover model its transitions for any state based on what direction it happens to be facing. This greatly improved the transition model, but forced an additional down-scaling of the heightmap to compensate for the eight direction slope evaluation.

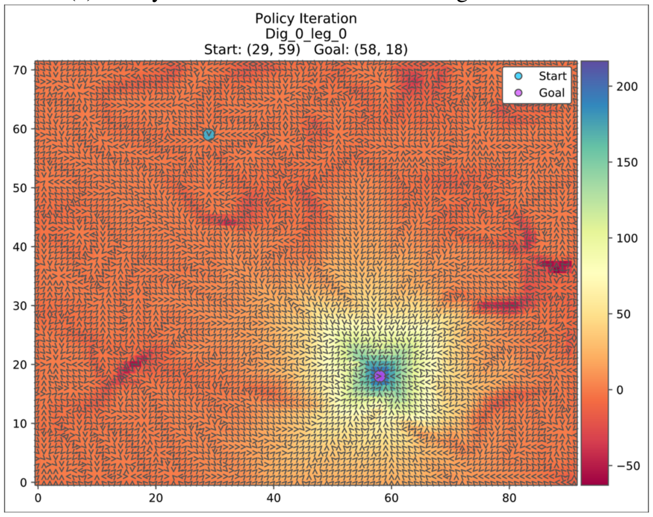
[2]Each dig site is removed from the graph after being "dug" to simulate the area being non-traversal

C. Resolution

High resolution heightmaps have potential to be really advantageous with a machine powerful enough to crunch the numbers in a reasonable amount of time. However, there is one side-effect of resolution that isn't associated with computation time: state progression. Measuring in an over 700mm, the rover is not a small vehicle. As mentioned in Section III-B2, trying to align itself with the policy heading is enough to move the rover into another state, which then prompts it to align it's heading with the policy of the new state. There's no problem if those two actions are the same, but if there not, the rover will endlessly turn back and forth. This behavior ultimately helped provoke the down-sizing of the heightmap resolution. In retrospect, it is over-ambitious to work with a 20mm resolution with a rover that's nearly a meter long.



(a) Policy Iteration based on terrain height threshold



(b) Policy Iteration based on terrain slope threshold

Fig. 5: Effect of obstacle detection method on Policy Iteration

D. V-REP Interface

At this point, some experimentation with V-REP was required. Thanks to the interfacing setup from Project 4: Trajectory Optimization using CasADi, a few simple modifications to the *execute_trajectory()* function allowed us to control the four wheel velocities instead of a 6-DOF arm. Adding a few more handles allowed the program to draw in the positions of the rover, as well as dump and dig sites. To get an idea of what effect values had on the velocity of each wheel, a ramp function was run on each wheel. This led to an increase spin out of control until the rover hurled itself off the edge of the scene. Turns out that a positive velocity turns each wheel counter-clockwise regardless of it's orientation, so wheels on opposite sides of the rover turn with velocities of opposite signs in order to move forward.

E. Obstacle Buffers

Once the rover was driving, the next oversight experienced by the team was the need for a buffer around obstacles. V-REP reports the rovers position from its geometric center by default. With the rover then moving based on instructions from the policy computation, it became evident that the rover must have had some self-destructive inclinations because it went right for the obstacles. At first, it seemed that the policy iteration might be flawed, but the actions of each state made sense when plotted. It was when we realized that the path took the rover near an obstacle that obstacle buffering came to mind. Measuring 475mm from center to outer corner, the rover needs a lot of room when maneuvering near obstacles. This was resolved by adding a 550mm buffer to every side of an identified obstacle.

V. Analysis

A. Use of VREP

V-REP turned out to be a really great interface to simulate terrain navigation from Python. We initially researched Gazebo and other direct ROS-interfacing platforms; however, operating system requirements and availability made this difficult to get running. V-REP offered all of the necessary application programming interface (API) to control the rover and get position and orientations. We also enjoyed have more intimate control over tiny details such as friction coefficients in the physics engine. Overall, it was a good platform choice, but it would have been convenient to interface V-REP with C++ to have better access to deeper environment variables during program execution.

B. Transition model

The initial transition model used was over-simplified such that the action probabilities did not accurately reflect the actual situation of the rover. By incorporating the rover's heading vector against the terrain slope vector, the new model adapts according to specific slopes instead of maintaining the same static probability distribution. This paints a better picture of what a model is likely to do at each state. Looking back, a lot of deliberation was required to come up with a model that accurately represented the scenario at all. This model still lacks the influence of the rover's velocity, which would focus the action probability in the desired direction as speeds increase even though the rover maintains very slow speeds. The model could be further improved by anticipating slippage; however, this would only be advantageous in extreme cases. Most of the time, the rover will travel along the flattest terrain.

C. *Grid Maps*

Using grid maps was the best option for displaying the height map and the rates map. Each location in the M x N array map was used to store the height of that location. On top of the M x N map array, there was a rates map that stored the slope of the of the terrain in all eight directions. This was a complex way to store the map information, but it was pertinent to have accurate readings for the rover to traverse the map.

D. *Dig Site Order*

The fact that the problem is NP-Hard greatly strangled our potential solutions. Initially we came up with an algorithm using bouncing walk checks between each dig site that was bogged down and almost never worked. This led us to the use of networkx which was truly fantastic. Still, the algorithm requires one to look at every possible permutation which blows up with n!. For instance, 6 dig sites required checking 720 combinations and took roughly ten minutes on a 30x30 grid. EThis issue is something we never address and still poses a major issue given the ten minute time limit the mining rover has in the competition. For now, an intelligent operator choosing the dig sites is a much faster, if less robust, solution.

The other issue with the implemented solution is that it is not run on the map used for PI due to the increased complexity of a 100+ by 100+ grid.[3]

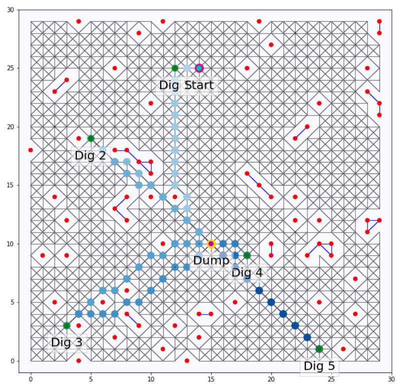
E. *Policy Iteration*

Policy Iteration was the best planning method to use for the rover because it provided an action for every location on the map. This was to account for the uncertainty, in case the rover got off the path. Fundamentally, it provided good directions to the goal and proved to be efficient with a reasonable resolution. The one drawback to policy iteration was that the rover had a hard time traveling along diagonal directions. When the actions converged along the optimal path in the policy, it would bounce from side to side while attempting to travel in that direction. It would rarely move in the desired direction, but it would always work really hard to stay on the line. And since the uncertainty from skid-turning threw off the course, it would have a hard time recovering its direct path to the goal.

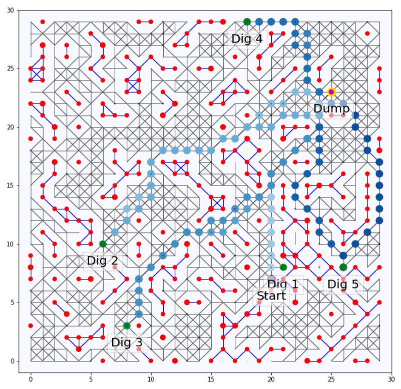
F. *Rover Control*

The rover faced some problems with only being to operate in eight directions. In some of the simulations, the rover would only need to travel in a diagonal direction to reach the goal. However, it would turn out of state and perform

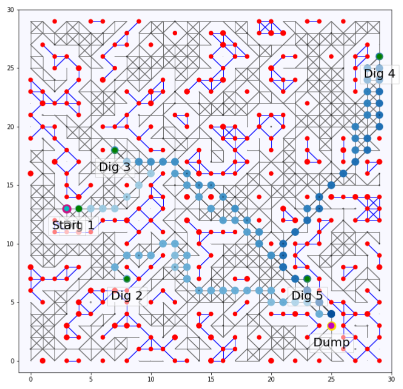
[3]We will be running this one last time tonight and will submit images of the results in the morning for completion.



(a) 10% Obstacle Coverage



(b) 25% Obstacle Coverage



(c) 33% Obstacle Coverage

Fig. 6: A* Search on a 30x30 grid with 5 dig sites and 8 direction movement

a "bounce-walk style" movement as it kept correcting itself to stay on path. The resolution had already been reduced enough, so reducing it more wouldn't be beneficial. Another heuristic would need to be developed so that the rover could be controlled with more than eight directions, and so it would not have to correct itself outside of those directions. It would be okay if the rover could follow close to the desired path, and only correct its path occasionally. The rover tended to waste a lot of time over-turning, or turning in the wrong direction, where it would turn all the way around to the right, instead of a little to the left.

VI. Discussion

The team that worked on this project was new to motion planning, and were able to see it work on a real-world scenario. There was the efforts that went into making a custom algorithm for the Mars rover to accomplish its task effectively. The biggest feat was making a policy that was complete and took the slope of every direction at every location into account.

The team had its share of shortcomings with the project. There was a quick realization that it is difficult working with uneven terrain. Despite what the terrain is shaped like, the rover doesn't always go where you want it to go. Detecting obstacles in a real-world environment can be difficult, because there is a wide array of types and it's not always clear what is defined as an obstacle. The method of detecting obstacles is highly dependent on what application it's being used for. And simplifying is the best way to lower the computation time and cost when computing the rover trajectory. In this case, lowering resolution was key to getting a reasonable computation time for the competition.

When handling this work, it would be recommended to get a better idea of what you want the rover to do in more circumstances. The team never got to see what actual terrain this would be used on, nor did they get to experiment with the rover that is used in the competitions. It would have brought better understanding to how the rover moves, although the speed and actions were simulated very closely.

The next steps of this project are to smooth the trajectory paths of the rover to make it faster and more efficient. The transition function would also need to be improved so that the probabilities would be spread appropriately across all actions for each location. The current function works well, but more research would be required to find the best way to distribute them. There are some moments where the rover made too many unnecessary turns, and it would be pertinent to figure out how to reduce them.

Another improvement we would like to implement is to use the A* plans from the Dig Site Planner as inputs to the utilized reward functions for each leg of the PI planner. The belief being that this will speed up convergence and incentivize the rover to follow the "dumbed-down" path which, in theory, should follow traversable ground.

References

[1] 2.2.2 Particular Forward Search Methods. *Planning Algorithms*, by Steven Michael. LaValle, 1st ed., Cambridge University Press, 2006.

[2] 10.2 Algorithms for Computing Feedback Plans. *Planning Algorithms*, by Steven Michael. LaValle, 1st ed., Cambridge University Press, 2006.

[3] Garcia, E., and P. Gonzalez De Santos. Mobile-Robot Navigation with Complete Coverage of Unstructured Environments. *Robotics and Autonomous Systems*, vol. 46, no. 4, 30 Apr. 2004, pp. 195204., doi:10.1016/j.robot.2004.02.005.

[4] Hermans, Tucker. Markov Decision Processes (MDPs). CS 6370. Motion Planning. Motion Planning, 19 Oct. 2018, Salt Lake City, Warnock Engineering.

[5] Roberts, J.m., et al. Autonomous Control of Underground Mining Vehicles Using Reactive Navigation. Proceedings 2000 ICRA. Millennium Conference. *IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, 24 Apr. 2000, doi:10.1109/robot.2000.845322.