ASEN 5519 - Final Project

Path Planning and Execution of MDP Directed RRT for Autonomous Systems

December 13, 2020

Jamison McGinley*, *University of Colorado - Boulder*

This final project for ASEN 5519 implements an iterative policy driven GoalBiasRRT with both geometric and kinematic constraints. The deterministic planners operate with the intent to publish control inputs to a Unity lunar rover simulation to facilitate autonomous navigation. It is found that sampling control inputs to handle kinematic constraints dramatically increase solution times, making the control inputs obsolete.

Nomenclature

N_a	=	Number of nodes in branch of MCTS tree corresponding to action a
u_ϕ	=	Steering angle [rad]
u_v	=	Linear velocity [m/s]
$A(s)$	=	Action space of state s
a	=	Action
c	=	Exploration Constant
$FaMSeC$	=	Factorized Machine Self-Confidence
$MCTS$	=	Monte Carlo Tree Search
MDP	=	Markov Decision Process
N	=	Number of nodes in MCTS tree
pdf	=	Probability Density Function
q	=	Configuration
ROS	=	Robot Operating System
RRT	=	Rapidly-Exploring Random Tree
s	=	State
VI	=	Value Iteration

*105207291

I. Introduction

Autonomous robots will be used in a variety of challenging environments to accomplish complex tasks in the place of humans. These tasks will be delegated and be supervised by human users, who (despite not necessarily being robotic experts) must be able to establish a basis for correctly using and depending on robotic autonomy for success. In operational contexts, it is well-known that the trust developed by a user towards an autonomous system could potentially lead to an inaccurate understanding of that system's capabilities [2]. Such misunderstanding can lead to improper tasking of the agent, and subsequent misuse, abuse, or disuse of autonomy. A potential means to address this is the *Factorized Machine Self-confidence (FaMSeC)* computational framework developed in refs. [1, 3, 4], which considers computation of several interrelated (and non-exhaustive) factors that enable autonomous decision-making agents to generate machine self-confidence assessments in the context of solving problems described by Markov decision processes (MDPs).

FaMSeC is composed of five metrics which can describe the machine self-confidence of an autonomous system. This is an active area of research, with many yet unanswered questions and potential research avenues. Current work into studying the impact of FaMSeC seeks to investigate the effects of *Outcome Assessment* and *Solver Quality* (two FaMSeC metrics) in regards to human decision making under uncertainty prior to task fulfillment. At this time, the basis for this research is in a simulated manner, abstract from the physical robotic problems that the research seeks to investigate. The purpose of this project is to construct a means to facilitate more realistic simulations of robotic navigational task fulfillment for use in the computation of machine self-confidence.

II. Problem Statement

The aforementioned FaMSeC metrics of interest, *Outcome Assessment* and *Solver Quality*, are both computed by simulating MDP roll-outs through a policy to construct a cumulative reward pdf. As such, it is critical that the motion planning of the autonomous system follow a policy. The ultimate goal of this project is to establish iterative online planning for a simulated lunar rover in the Unity game engine. The online planning needs to occur in two stages: 1) At a high level, the motion plan will follow a policy between large scale discrete grid cells, 2) At a lower level a GoalBiasRRT with a Kinematic model (first order car) for the rover will be used to execute the high level plan. After the RRT finds a path, the control inputs used will be published to the rover in Unity via ROS messages. This planning must occur iteratively between 1) and 2) due to the uncertain nature of the rover in the Unity environment. The low level RRT will attempt to guide the rover according to the high level plan, however rover's uncertainty may lead it to deviate from the the high level plan to a less desired grid cell (i.e. transition to an undesirable state s').

In order to build towards the goal of iterative online planning for a simulated lunar rover in the Unity game engine, the approach is broken down into four subproblems to solve.

II.A. Subproblem 1

Solve the MDP offline with Value Iteration to find a policy that will serve as a the high level planner, add GoalBiasRRT as a low level planner. The RRT does not yet take into account the Kinematic model of the rover.

II.B. Subproblem 2

Solve the MDP online with *Monte Carlo Tree Search (MCTS)* to find a policy that will serve as a the high level planner, add GoalBiasRRT as a low level planner. The RRT does not yet take into account the Kinematic model of the rover.

II.C. Subproblem 3

Augment subproblems 1) and 2) with a Kinematic model for the lunar rover.

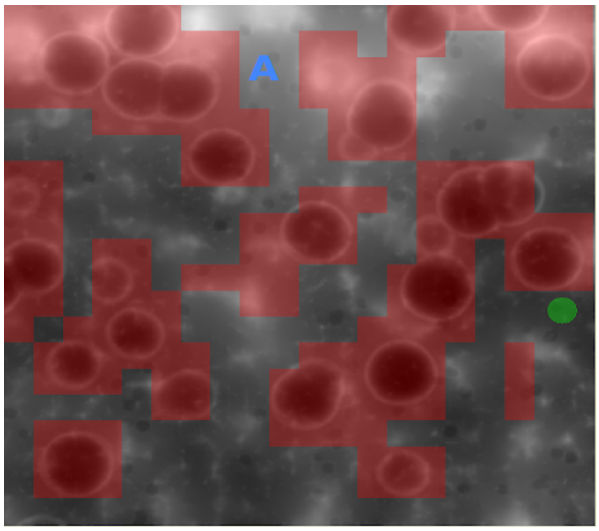
II.D. Subproblem 4

Establish meaningful communication with Unity simulation such that the control inputs used to guide the rover in each iterative low level plan are published to the physics engine. The rover will act on navigational commands and attempt the motion plan at each stage.

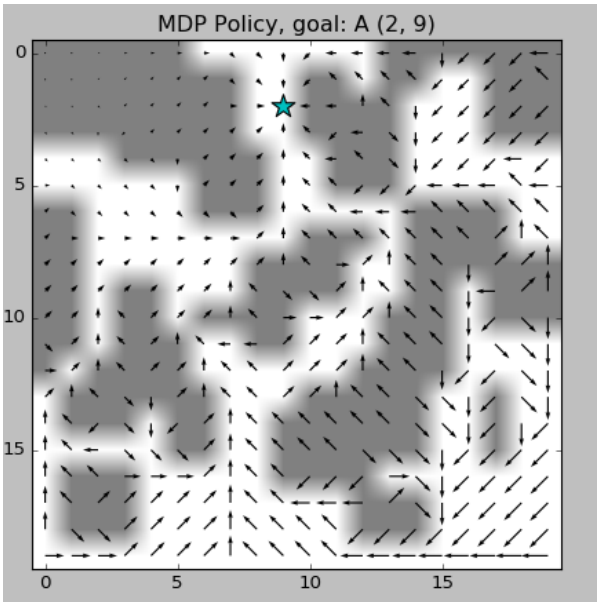
III. Methodology

The approach used to solve each subproblem builds on the assumptions used in the implementation of the problems. As such, while the focus of this project is the actual motion planning of the rover in the Unity environment, work that was done previous to this investigation needs to be addressed and explained.

The Unity terrain map for the lunar surface navigated here is a 1 $[km]^2$ map with 50 $[m]$ in elevation. Exporting into terrain map into a python environment it can be down-sampled to a 4097 $[unit]^2$ map to plan in. From here the space is discretized to a 20 x 20 grid to solve for with a policy for each goal. There are no explicit obstacles in this environment, so hazardous areas in the discretization are indicated by hand. The policies found through VI and MCTS operate on this discrete space.



(a) Navigation Problem for goal A. *Map showing free space and hazardous space in discretization over image of the lunar surface*



(b) Policy Solved with VI for goal A. *Map showing the directed action from the policy for each discrete space*

Figure 1: High level planning environment

III.A. Subproblem 1

For subproblem 1 the goal is to iteratively deploy a GoalBiasRRT in the down-sampled (4097 x 4097) workspace to facilitate the recommended action from the policy solved by VI. A key assumption being made in this and all following subproblems is that the configuration space can be considered the workspace due as the rover's size relative to the hazards is minimal and collisions don't immediately result in failure (in Unity). An abridged version of the approach is:

1. Find discrete cell (20 x 20), state s, that contains starting configuration, $q_0 = [x_0, y_0]$
2. Sample policy for recommended action, $a \in A(s)$
3. Determine desired state, s'
4. Assign intermediate goal configuration at the center of s'
5. Run GoalBiasRRT in workspace from starting configuration to intermediate goal configuration

6. $q_0 \leftarrow$ final configuration from RRT
7. Repeat

III.B. Subproblem 2

For subproblem 2 the goal is to iteratively deploy a GoalBiasRRT in the down-sampled (4097 x 4097) workspace to facilitate the recommended action from the policy solved by MCTS. The assumptions and procedure is essentially the same as subproblem 1, except rather than sampling the policy in state 2) the MCTS solver needs to be run for one transition. MCTS takes $a \in A(s)$, exploration constant c into $\max_{a \epsilon A(S)}(Q(s,a) + c * \sqrt{\frac{2*\ln(N)}{N_a^s}})$ to choose an action to simulate and extend towards. It iteratively updates the value of $Q(s,a)$ for each $a \in A(s)$ at deeper nodes and propagating value until ultimately an action is chosen. What is potentially different about the method used here is that the tree built by MCTS is not saved between iterations, which can have both positive and negative effects overall. So the approach is then:

1. Find discrete cell (20 x 20), state s, that contains starting configuration, $q_0 = [x_0, y_0]$
2. Run MCTS to find recommended action, $a \in A(s)$
3. Determine desired state, s'
4. Assign intermediate goal configuration at the center of s'
5. Run GoalBiasRRT in workspace from starting configuration to intermediate goal configuration
6. $q_0 \leftarrow$ final configuration from RRT
7. Repeat

III.C. Subproblem 3

The goal of subproblem 3 is to adapt the previous work to take into account a Kinematic model for the rover. For the rover the state is $\vec{x} = [x, y, \theta]^T$ with $\theta \in [0, 2\pi]$ for simplicity. The control inputs for the model are $u = [u_v, u_\phi]^T$ where u_v is linear velocity $\in [0, 4]$ and u_ϕ is steering angle $\in [\frac{-\pi}{4}, \frac{\pi}{4}]$. The control input constraints in the Unity environment are arbitrary, but these constraints were found to be reasonable in past research. The approach here is the same from the previous subproblems except GoalBiasRRT needs to be changed to handle this model:

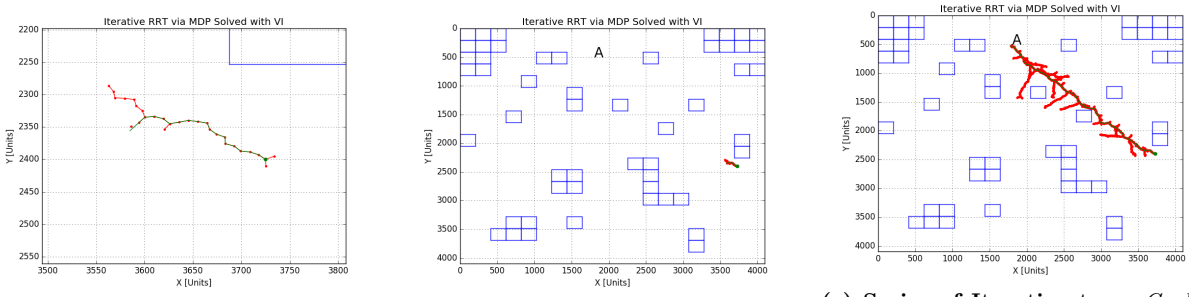
1. Find discrete cell (20 x 20), state s, that contains starting configuration, $q_0 = [x_0, y_0]$
2. Run MCTS to find recommended action, $a \in A(s)$
3. Determine desired state, s'
4. Assign intermediate goal configuration at the center of s'
5. Run GoalBiasRRT in workspace from starting configuration to intermediate goal configuration
   (a) **Accounting for kinematic constraints**
   (b) Sample controls
   (c) Numerically integrate using Runge-Kutta 4th order for 10 sec
   (d) Find closest point in resulting path to node to be added to tree
   (e) Repeat m times
   (f) Choose control law from m times that ended closest to node to be added
6. $q_0 \leftarrow$ final configuration from RRT
7. Repeat

III.D. Subproblem 4

Subproblem 4 is different from the previous three as it is essentially the implementation of subproblem 3. After the kinematic GoalBiasRRT successfully executes its first iteration, it will return a series of randomly sampled controls $u = [u_v, u_\phi]^T$ with the time for which these controls are active. The control will be published on latched to a ROS topic for the time they are active, while a subscriber in Unity will subscribe to this topic and act of the controls. The planner cannot continue planning while the rover is following the control inputs as it needs to start its next planning iteration from the final location of the rover from the current iteration.

IV. Results

IV.A. Subproblem 1



(a) Close View *GoalBiasRRT for a single state transition from VI*

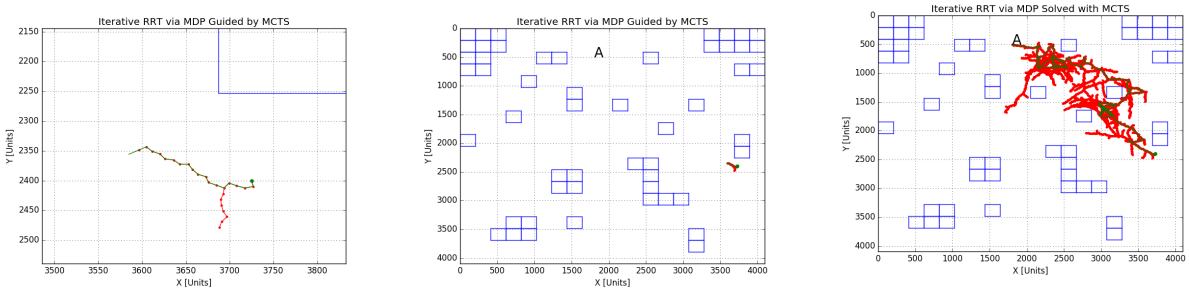
(b) Full View *GoalBiasRRT for a single state transition from VI*

(c) Series of Iterative trees *GoalBiasRRT for every state transition to reach A*

Figure 2: High level VI with geometric RRT low level planner [See Appendix for Full Scale Images]

Subproblem 1 gives very positive results. In Figure 2.a RRT and respective path can be seen, corresponding to a state transition as directed by the policy. The path this yields for a single direction is certainly direct, most likely due to the large scale of the overall configuration space. Figure 2.c shows all iterations in one plot, while it may appear to be a single RRT it is in fact many iterations worth following the high level planner. Comparing this to the policy for this goal in the Appendix, it can be seen that this high level directive was clearly followed.

IV.B. Subproblem 2



(a) Close View *GoalBiasRRT for a single state transition from MCTS*

(b) Full View *GoalBiasRRT for a single state transition from MCTS*

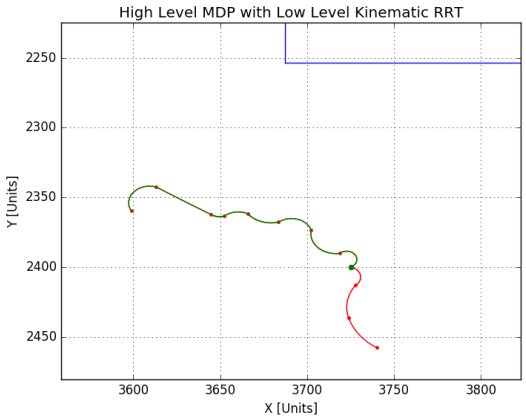
(c) Series of Iterative trees *GoalBiasRRT for every state transition to reach A*

Figure 3: High level MCTS with geometric RRT low level planner [See Appendix for Full Scale Images]

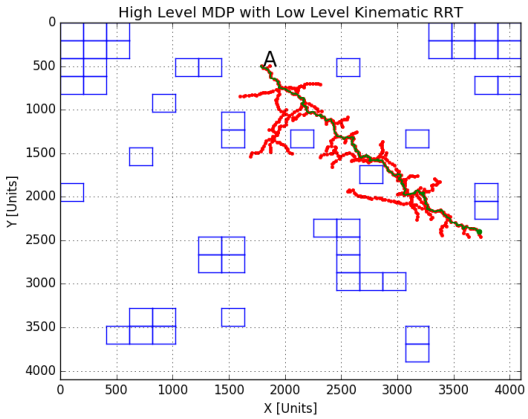
Subproblem 2 shows the same success as subproblem 1, the high level plus low level planners are certainly performing correctly. Figure 3.c indicates that policy from MCTS is sub-optimal, leading to many additional

states in the path. However this behavior is expected to some extent as MCTS is an online algorithm that builds a tree to determine a preferred action in real time. At its best MCTS can approach VI.

IV.C. Subproblem 3



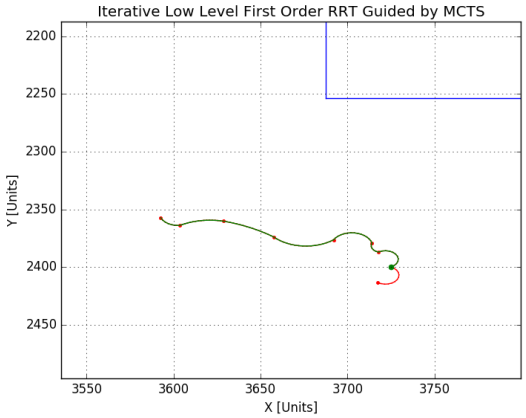
(a) Close View *GoalBiasRRT with Kinematic model for single state transition from VI*



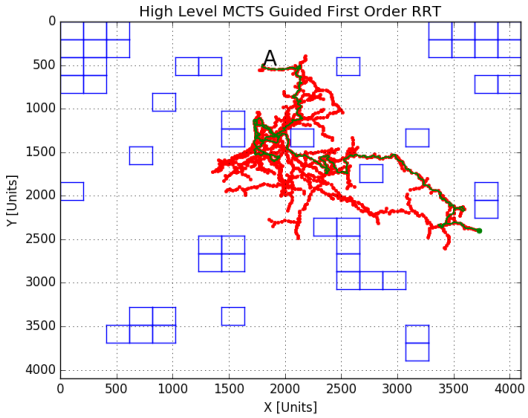
(b) Series of Iterative trees *GoalBiasRRT with Kinematic model for every state transition to reach A*

Figure 4: High level VI with Kinematic model RRT low level planner

With the use of the policy found with VI as a high level for this problem we once again see very good results. The RRT with kinematic constraints is finding control inputs and numerically integrating over time to create a feasible path to the next node. Additionally the connections between the nodes are smooth which is a good indication that the series of control inputs provided to the rover can be facilitated in a reasonable manner.



(a) Close View *GoalBiasRRT with Kinematic model for single state transition from MCTS*



(b) Series of Iterative trees *GoalBiasRRT with Kinematic model for every state transition to reach A*

Figure 5: High level MCTS with Kinematic model RRT low level planner

Using MCTS as the high level planner we see the same sub-optimal behavior from subproblem 2. While this high level plan is not direct, it is smooth so the rover can likely accomplish these iterative plans.

Overall while the results for subproblem 3 appear to operate as planned, there is an unexpected complication: Adding the kinematics constraints the to the GoalBiasRRT dramatically increased the runtime.

IV.D. Subproblem 4

As alluded to in the previous results section, the increased runtime posed a problem for actually facilitating control of the rover in the Unity simulation. The problem stems from the design of the iterative high level/low level planner as the RRTs need to be solved live. Based on the uncertain rover in Unity, there is no predicting the state of the rover after facilitating its set of commands for a given iteration. The solution was to build the next iteration from the state of the rover in simulation at the end of the last. This requires that the rover isnt stuck following it last published control law while waiting for a new one. Benchmarking was not a part of this project, but an arbitrary subproblem 3 with VI is shown below.

	RRT Iteration									
Solve Time [s]	15.138	13.603	8.509	31.843	18.405	33.658	21.098	15.174	32.751	19.594

Solve time is the time required to solve each RRT iteration to ultimately reach the goal A. With these being as long as they are, the rover will have to wait far too long to receive a new set of controls after accomplishing the set of controls from the previous iteration.

V. Discussion

Based on comparison between Figures 1 and 2, as well as 4 and 5, the extent to which VI outperforms MCTS is clear. As discussed briefly in the previous section, this is because MCTS is an online solver that is inherently sub-optimal whereas VI is optimal. Due to the length requirements, the only scenario explored was goal A with these specific hazards. However, from past experience MCTS has additional issues in regards to narrow passages. Furthermore the kinematics model used is also limited in these narrow corridor scenarios as the lunar rover has fixed minimum turning radius that could cause it to be stuck. However, research into FaMSeC with the lunar rover navigation task thankfully provides a sparse environment where these concerns are mitigated.

An overall concern with the current implementation of all algorithms is that both policies tend to take the rover near to obstacles. Given the stochastic Unity environment, this puts the rover at high risk of entering a hazardous area. While these areas aren't immediately dangerous, the rover is at increased risk of getting stuck here. It would be interesting to bias the low level RRT towards sampling in regions as far as possible from the hazards to add a factor of safety to the algorithm.

VI. Conclusion and Recommendations

Overall the approach developed in this project to implement an iterative policy driven GoalBiasRRT to facilitate autonomous motion was largely successful. Subproblems 1-3 indicate that this approach is definitely feasible in a deterministic scenario. Subproblem 4 was designed to show that the iterative aspect of the motion planning allowed for this approach to succeed in stochastic scenarios, however with the difficulties faced and a limited time to work the current approach would not work. There are likely many ways to get around the timing issue faced, the most obvious being to optimize the sampling of control laws and numerical integrating them. As it stands the GoalBiasRRT without kinematic constraints solved in under a second for each iteration. Optimizing the version used in subproblem 3 could likely approach these speeds and solve the issue.

References

[1] Matthew Aitken. Assured human-autonomy interaction through machine self-confidence. Master's thesis, University of Colorado Boulder, 2016.

[2] Mary T Dzindolet, Scott A Peterson, Regina A Pomranky, Linda G Pierce, and Hall P Beck. The role of trust in automation reliance. *International journal of human-computer studies*, 58(6):697–718, 2003.

[3] Brett Israelsen. *Algorithmic Assurances and Self-Assessment of Competency Boundaries in Autonomous Systems*. PhD thesis, University of Colorado at Boulder, Boulder, 2019.

[4] Brett Israelsen, Nisar Ahmed, Eric Frew, Dale Lawrence, and Brian Argrow. Machine self-confidence in autonomous systems via meta-analysis of decision processes. In *International Conference on Applied Human Factors and Ergonomics*, pages 213–223. Springer, 2019.

Appendix

VI.A. Code

All code (except .pkl file) can be found at:
https://github.com/jamc3951/amp_final

VI.B. Policy Solved with VI

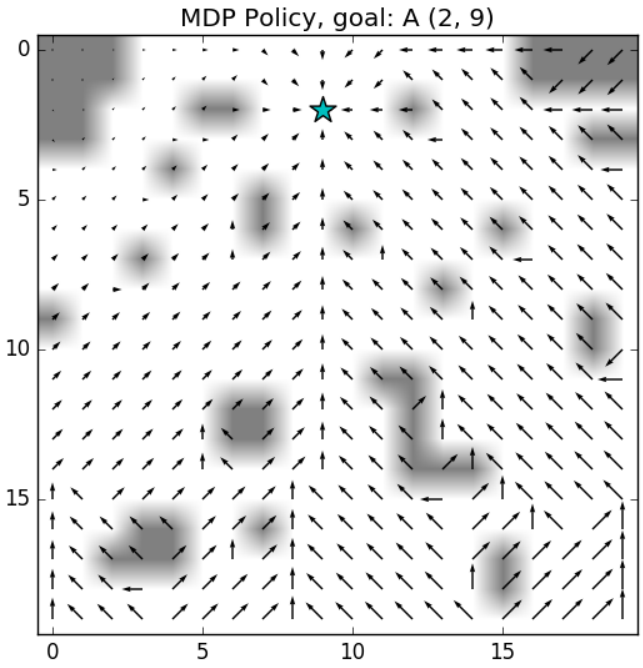
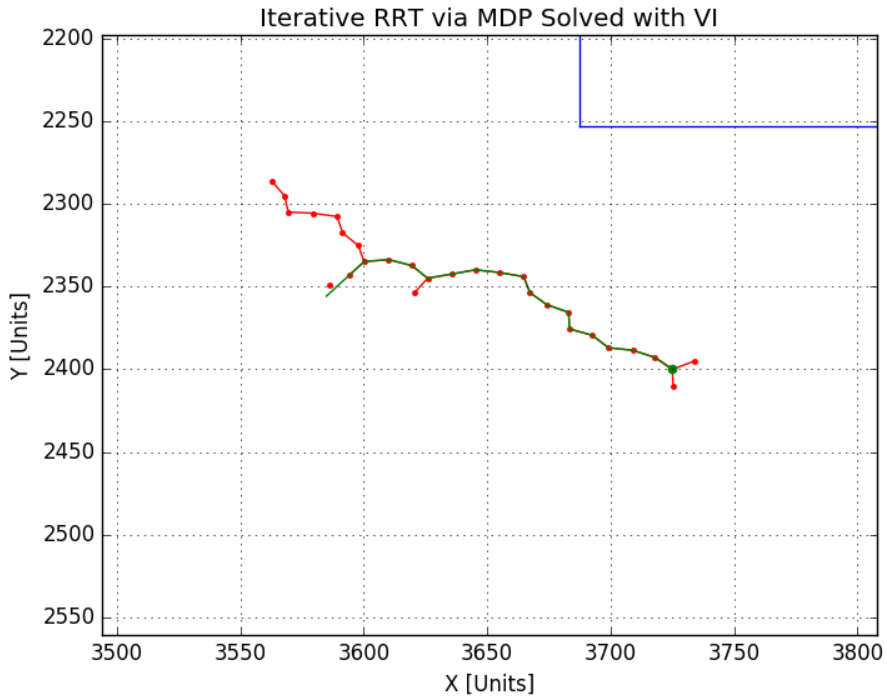
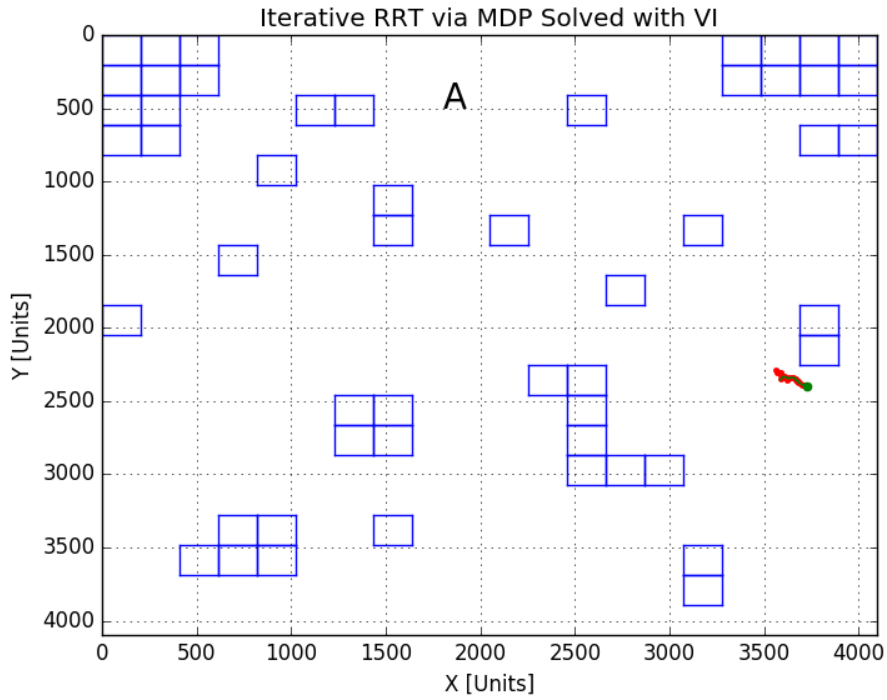


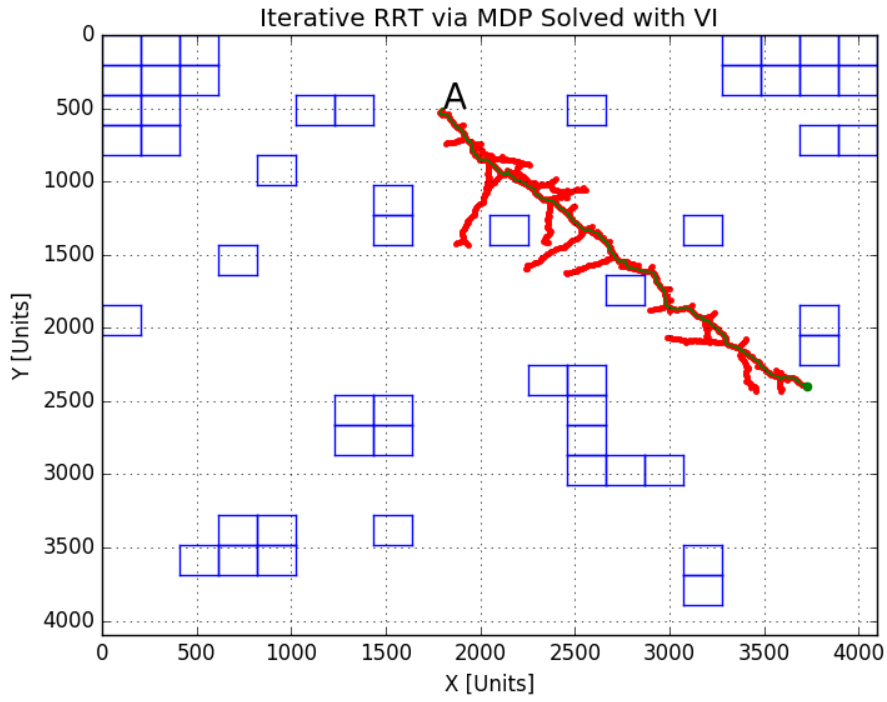
Figure 6: VI Policy leveraged for results in report

VI.C. Results

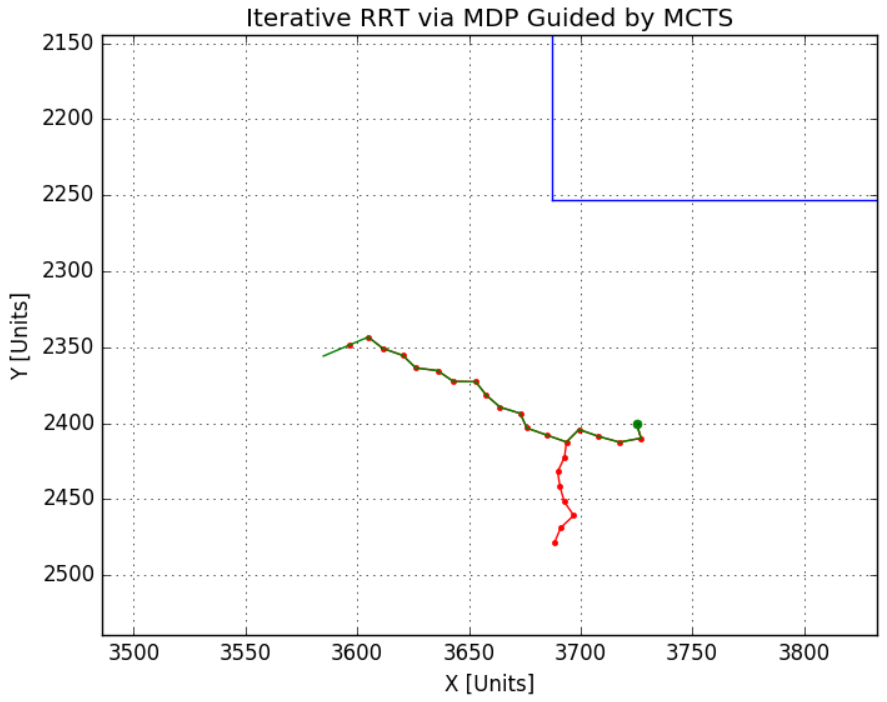
VI.C.1. Subproblem 1

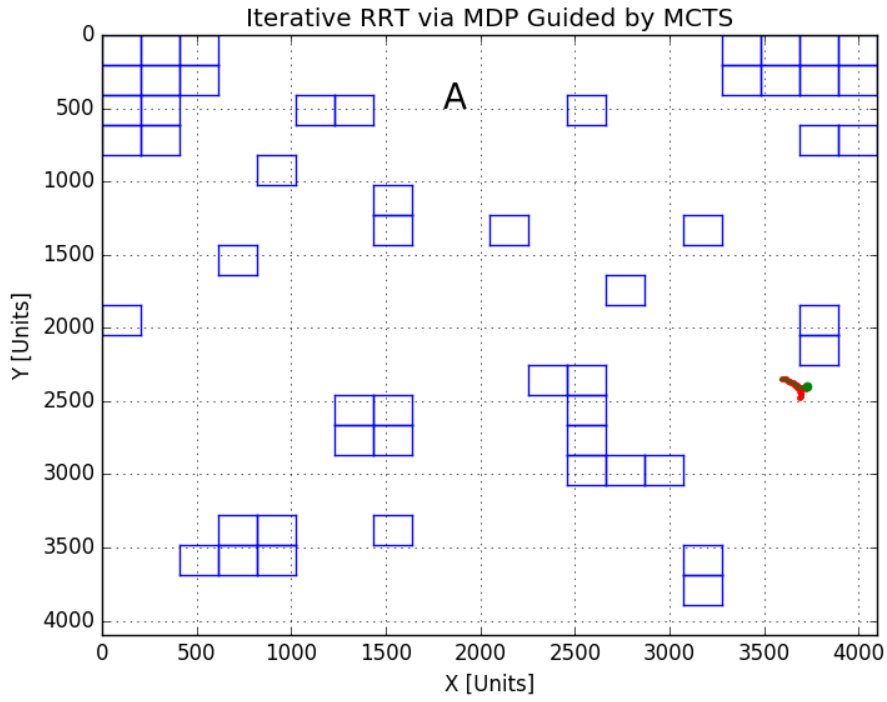


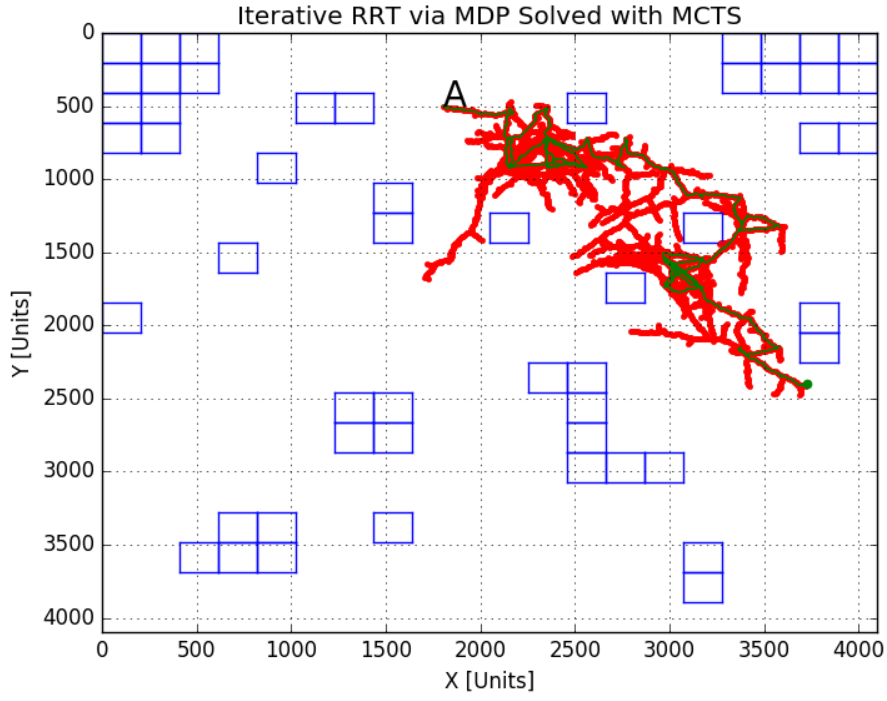




VI.C.2. Subproblem 2




Iterative RRT via MDP Guided by MCTS
A
Y [Units]
X [Units]
0
500
1000
1500
2000
2500
3000
3500
4000


Iterative RRT via MDP Solved with MCTS
A
Y [Units]
X [Units]
0
500
1000
1500
2000
2500
3000
3500
4000