Adaptable Mission Planning for Kino-Dynamic Systems

Tony Jimenez,* Larry Bush,†
and Brian Bairstow‡

May 11, 2005

Abstract

Autonomous systems can perform tasks that are dangerous, monotonous, or even impossible for humans. To approach the problem of planning for Unmanned Aerial Vehicles (UAVs) we present a hierarchical method that combines a high-level planner with a low-level planner. We pose the problem of high-level planning as a Selective Traveling Salesman Problem (STSP) and select the order in which to visit our science sites. We then use a kino-dynamic path planner to create a large number of intermediate waypoints. This is a complete system that combines high and low level planning to achieve a goal. This paper demonstrates the benefits gained by adaptable high-level plans versus static and greedy plans.

1 Introduction

1.1 Motivation

The human control of robots on Mars is difficult, partially due to large communication lag times. One solution to this problem is to use robots capable of autonomous activity. For unmanned aerial vehicles (UAVs), this capability is a necessity, since UAVs are constantly in motion and cannot stop and wait for instructions. Thus, UAVs will need to be able to handle low-level planning, for example the kinematics of getting from point A to point B.

However, the question remains of how to handle higher-level planning, specifically, choosing which science sites to visit and in what order. The mission plan can be created on Earth and sent to Mars, or created by the UAV on Mars. Any

*Draper Laboratory, Cambridge, MA.
†MIT Lincoln Laboratory, Lexington, MA.
‡Massachusetts Institute of Technology, Cambridge, MA.

plan sent from Earth would essentially be a static plan, since re-planning would be impossible because of the communication lag. This can be disadvantageous if the situation changes, for example if too much fuel is consumed or if new sensor readings show that a site is more interesting than previously thought. If the mission planning is performed on the UAV or otherwise in-situ, then the potential exists for response to changes in the environment. This can be implemented as continuous plan generation. Exploring the value that continuous planning provides will give an idea of the magnitude and insights into when it is useful.

1.2 Problem Statement

Our scenario involves a UAV on Mars traveling between science objectives. The UAV is given a set of interesting science sites and a limited amount of fuel. The problem is to guide the UAV to the science sites to gather as much science value as possible within the fuel constraints.

We address the design and analysis of an autonomous exploratory planner for a UAV. This problem involves merging an adaptable mission planner with a kino-dynamic path planner [1], and comparing the performance with that given by a static plan. The mission planner will adapt to new readings of the science sites from a UAV's long range sensors.

1.3 Previous Work

Our project is premised upon two main bodies of work, namely that done by B. Hasegawa[2] and T. Leaute[1].

1.3.1 Continuous Observation Planning for Autonomous Exploration

The thesis by Brad Hasegawa presents a new approach for solving a robotic navigation path-planning problem. The approach first formulates the problem as a selective traveling salesman problem (S-TSP), then converts it to an optimal constraint satisfaction problem and solves it using the Constraint Based A* algorithm. The solver, shown in the system architecture diagram in Figure 1, performs this key ability.

The solver is a continuous observation planner, which updates the plan when new observations affect the candidate set (possible places to visit). The objective of the robot is to map its environment. The robot chooses to navigate to observation locations, which will maximize information gain. Each observation may affect the utility and cost of unvisited observation locations (candidates), which necessitates re-planning. There is an implicit trade-off between the planning horizon and how often the candidates are updated. The planning horizon

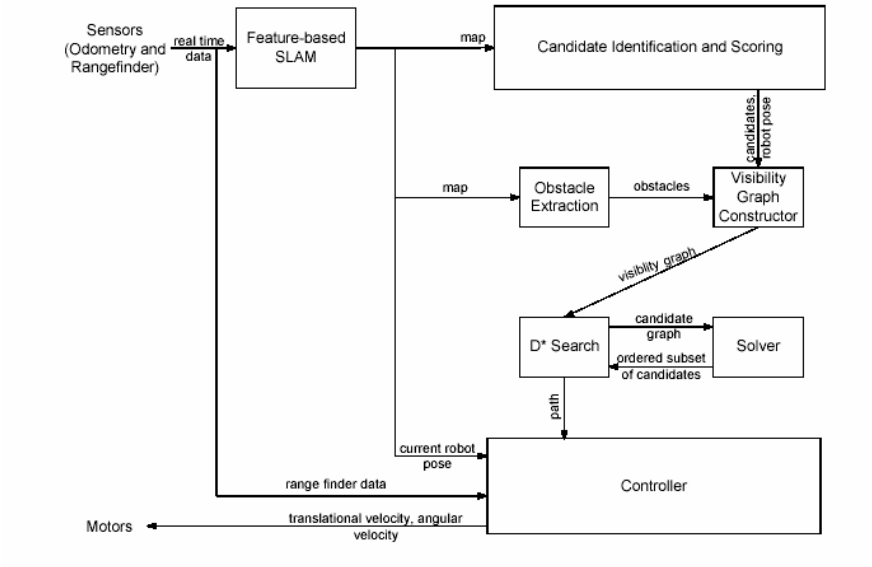

Figure 1: The above diagram is the system architecture for [2]. The navigation architecture starts with a partially complete map. Candidates and obstacles are extracted from the map, which are used to construct a visibility graph. The D* search is used to update the candidates. The candidates are passed to the solver, which creates a plan (ordered candidate subset).

should mirror the expected time period between re-planning. In other words, if we look ahead 5 tasks, we want to be able to execute those 5 tasks before we have to re-plan. If this does not occur, then our plan is optimized for a different planning period than it is executed for. This results in sub-optimal planning.

Ultimately, the system is making an exploration-exploitation trade-off, which can be generalized to other tasks. The tasks must involve observation and candidate list utility/cost updates. This method is likely to be effective when we have (at a minimum) a large-scale prior map of the exploration region.

The thesis [2] addresses a mapping application where the candidates frequently changed due to new observations. The finite-horizon technique is more effective when the candidates do not change frequently. Yet the mapping application actually favors observation candidates that increase its situational knowledge the most. For these reasons, the finite-horizon method is more effective when a high-level map is known. The attributes of continuous finite-horizon planning lend themselves to exploratory missions with a specific objective (i.e. a science exploration application) where a prior map is known. Refining the map will affect the cost estimate for the science tasks and the utilities of the science tasks may change as prior successes affect the probability of future successes. This necessitates continuous planning. However, the changes should be sufficiently infrequent, so that a finite-horizon is more effective than a purely greedy candidate selection strategy.

Key elements of the framework presented in [2] are shown in Table 1.3.1.

Table 1: Key attributes of the continuous observation-planning framework.

Exploration Problem:	Explore and construct a map of an environment
Exploration Method:	Feature based
Assumption:	The robot knows the large-scale environment structure
Path Cost:	Path length (physical distance)
Path Planner:	Visibility Path Planner : F(map, candidates, pose)
Map Type:	Feature based SLAM map
Pose:	Robot position and heading
Candidate:	An observation point bordering an unexplored area
Candidate Utility:	An estimate of the observable unexplored area
Candidate Dynamics:	How do candidates change as a robot explores

1.3.2 Coordinating Agile Systems Through the Model-based Execution of Temporal Plans

The work in [1] provides a novel model-based execution of a temporally flexible state plan (TFSP) for the purpose of UAV navigation. Its kino-dynamic controller is a continuous planning framework. However, the high level planner is not.

Our integration would enable this work to perform continuous high-level planning. The scope of our project includes enabling the simulation framework to accept TFSP updates. In particular, the observations would be considered when creating the temporally flexible state plan. Continuous high-level planning would allow the UAV to adapt its high-level goals to the observed environment. Specifically, it adapts the additional information that it learns about its environment (i.e. a more accurate map or the ramifications that one task has on the utility of future tasks). Therefore, this integration would provide a system which observes, learns and updates its higher level planning goals.

This work is further discussed in section 2.3.

2 Method

2.1 Problem Walkthrough

Our system takes a set of nodes, constructs a plan to visit a subset of those nodes, and simulates the flight path to test the validity of the plan. Plan construction starts in the adaptable mission planner. The adaptable mission planner takes a set of node locations and utilities and calculates the optimal path to get the most utility given a limited amount of fuel. This plan may not be feasible if the estimates of fuel cost between the nodes is too low.

The plan is then fed into the kino-dynamic path planner which creates waypoints to guide the UAV to the next node. These waypoints are built on a one-second time scale, so they represent a more finely discretized plan. These waypoints are followed using a simulation of the environment and an autopilot which actuates the aircraft. The simulation returns the actual fuel usage, which may cause changes in the plan. The block diagram of the complete system is shown in Figure 2. Each part of the system is explained in greater detail in the following sections.

2.2 Adaptable Mission Planner

The high-level mission planner has been designed to solve the finite-horizon path planning as a Selective Traveling Salesman Problem. This is also known as the

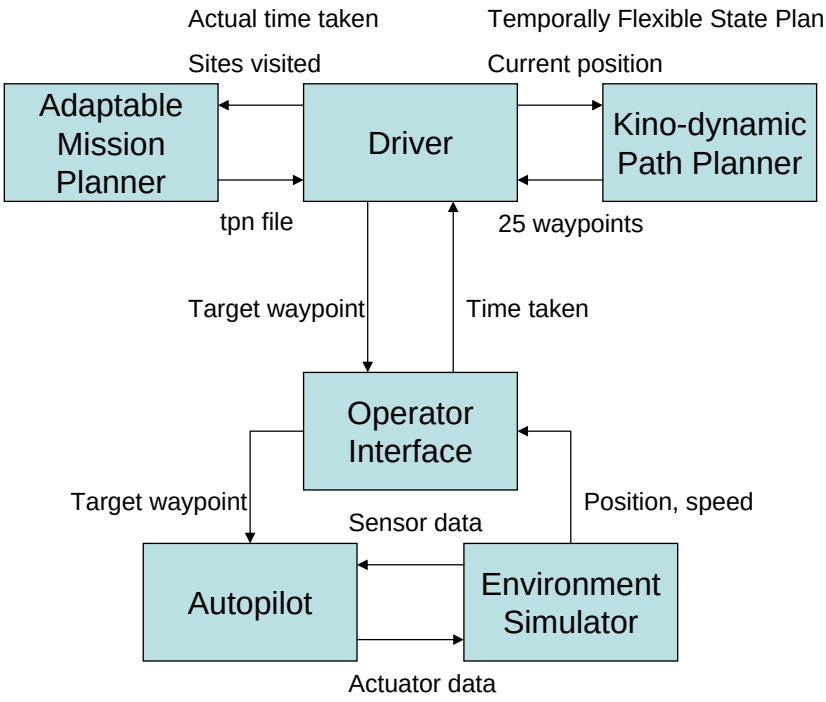

Figure 2: Block diagram of the complete system

Orienteering Problem.

The adaptable mission planner accepts a given set of utilities. This abstracts out the sensor model and provides a framework for future system engineering in analyzing the benefit of using an adaptable mission planner versus a static plan.

The mission planner also accepts a set of coordinates indicating the science sites. These are the targeted positions on the map that the mission planner is considering traveling to. The mission planner generates an traceable path of nodes that will maximize its utility with a given distance constraint.

The following steps outline the major parts of the adaptable mission planner:

1. Create Adjacency Matrix from coordinates of nodes
2. Order list of utilities to match ordering of nodes. Starting node must be at index 0.
3. Create traceable path through nodes by calling STSP Solver
4. Remove the next node to be traveled to from the list of considered nodes
5. Reduce the remaining distance by the distance to the next node
6. Repeat from step 1 with an updated set of utilities from the UAV's sensors

2.2.1 STSP Problem Formulation

The original traveling salesman problem involves a salesman trying to visit each city on a map while traveling as little distance as possible. Essentially the problem consists of a set of nodes N = $\{n_1, n_2, n_k\}$ and a set of arcs between nodes A = $\{(n_1, n_2), (n_1, n_3), (n_{k-1}, n_k)\}$. Each arc is associated with a cost C((n_i, n_j)). Thus the definition of the traveling salesman problem is to come up with an ordered set $\{n_a, n_b, \ldots\}$ which contains every member of N and minimizes the total cost $\sum$C = C((n_a, n_b)) + C((n_b, n_c)) + ...

The Selective Traveling Salesman Problem (STSP) approaches the problem from a different direction. It imposes a maximum cost constraint C_{max} such that the total cost $\sum$C $\leq C_{max}$. Furthermore, each node is assigned a utility U(n_i). The goal then is to come up with a subset of N $\subseteq \{n_a, n_b, n_c, \ldots\}$ which maximizes the total utility $\sum$U = U(n_a) + U(n_b) + U(n_c) + while meeting the constraint $\sum$C = C((n_a, n_b)) + C((n_b, n_c)) + . . . $\leq C_{max}$.

One aspect of the system that should be noted is that the STSP will visit each node only once, and does not need to return to the beginning node. The intention is for the UAV to crash into the surface of the planet instead of having to return to its starting position.

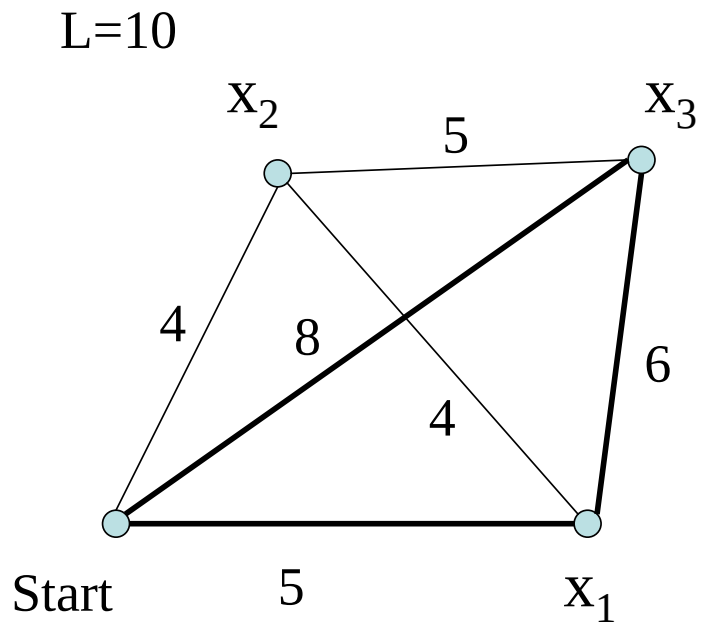

Figure 3: An example graph structure used to demonstrate an OCSP conversion

2.2.2 OCSP Problem Conversion

[2] demonstrates the capability of converting the STSP problem into an Optimal Constraint Satisfaction Problem. A CSP has a set of variables with a finite domain with a set of constraints over these variables. With an OCSP, we have a utility function that maps all assignments of the decision variables, which is a subset of the total set of variables. This function maps the assignments to real numbers. A solution to the OCSP will maximize this utility while satisfying all the constraints.

To convert an STSP to an OCSP, for each node, create a variable in the OCSP with a domain of $\{0, 1\}$. The utility of a variable with an assignment of 1 is the utility of the node from the STSP. The utility of the variable is zero otherwise. A variable with an assignment of 1 indicates that the node for that variable is included in the path that is being generated.

Figure 3 shows an example graph structure. The following represents the OCSP conversion for this structure:

- Decision Variables: x_1, x_2, x_3

- Domain: $\{0, 1\}$
- Attribute Utility Function:

  g1(x1=0)=0, g1(x1=1)=5

  g2(x2=0)=0, g2(x2=1)=9

  g3(x3=0)=0, g3(x3=1)=2
- Constraint: TSP(x_i=1) $\leq$ L
- Highlighted graph: x_1=1, x_2=0, x_3=1

To calculate the constraint for the OCSP, the cost of the TSP solution to the variables with an assignment of 1 should be less than that limit, L. A standard TSP solver, Concorde, is used in the current implementation. One will notice that the highlighted graph contains a Hamiltonian cycle through the variables with an assignment of 1. This is instead of a Hamiltonian path through these variables. However, to use Concorde to generate a Hamiltonian path instead of a Hamiltonian cycle, the graph structure needs to be converted into an asymmetric graph. This is done by setting the return cost to the start node to zero from all other nodes. This will cause the cost of all Hamiltonian cycles to then be equal to the equivalent path.

The current implementation of this OCSP/STSP solver then uses Constraint-based A* to solve the OCSP. This solution provides the Hamiltonian path through the STSP.

2.2.3 A Straightforward STSP Solver

While a powerful method has been shown to solve the STSP using a reformulation of the problem into an OCSP, our final design used a more straightforward method of solving the STSP. Our experiments did not require the more complex solver to handle the simulation.

We also found that in some sample problems, the OCSP/STSP solver as used in [2] gave similar performance characteristics as the straightforward STSP algorithm presented in this section. When the distance was restricted, the planner was not able to visit all the nodes. This aspect defines the difference between an STSP and a TSP problem. When not all the nodes were visited, the OCSP/STSP solver ended up supplying inconsistent answers. The straightforward STSP solver was then developed for this project to address these inconsistencies.

The straightforward STSP solver is a recursive function that explores every possible path within the constrained distance. It will return the path with the highest utility. In case of any ties of the utilities, it will return the path with

the least cost.

This algorithm has a complexity of O(N!). However, this upper bound is never realized in our simulations due to the distance constraint. Restricting the maximum depth of the search using the distance constraint results in a complexity of O((N-x)!) where N is the number of total nodes and x is the minimum number of nodes that cannot be traversed (due to the distance constraint). Likewise, N-x would be the maximum depth of the search. Consequently, this algorithm is sufficient for our experiments due to the number of nodes and the distance constraint used. This is a result of the hierarchical design of the system where the planner only deals with the high level nodes. In our problem, these nodes represent the science sites.

Future work can address the further development of the OCSP/STSP solver, but the goals for this project only required a working STSP solver, which in the future can easily be changed out of the overall system due to its object-oriented design.

2.3 Kino-Dynamic Path Planner Details

We used the Kino-dynamic path planner by Thomas Leaute [1]. We altered Leaute's algorithm to take in a new plan at each high-level planning node. Each of these nodes corresponds to a science site.

The Kino-dynamic path planner system consists of 2 main parts:

- Planner
  - Driver
  - Adaptable Mission planner
  - Kino-dynamic path planner
- Simulator
  - Operator interface
  - Autopilot
  - Environment simulator

The Planner software runs on a Linux platform. The simulator setup consists of three main components which run on separate platforms(Figure 4). The operator interface and the environment simulator run on two separate Windows PCs. The Piccolo UAV Autopilot is a specialized piece of programmable hardware. The Simulator also includes two connecting components, a CAN Bus and a Piccolo Ground Station. The simulation system is made by Cloud Cap Technology.

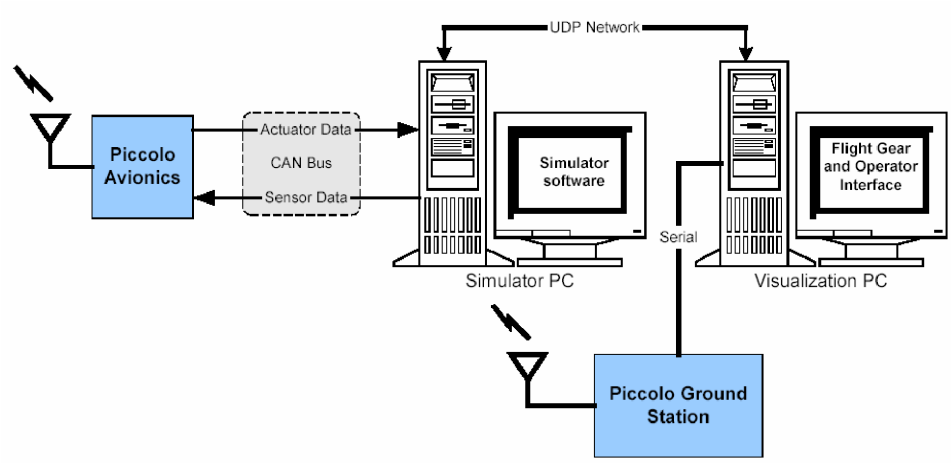

Figure 4: Piccolo hardware-in-the-loop simulator setup made by Cloud Cap Technology

The driver program sets up the map and parameters for the simulation. The map consists of the locations of the map corners and obstacles. In our system, we have 2 obstacles which represent mountains. The simulation uses the parameters and dynamics of a Cub aircraft, which is a light personal airplane with a simulated airspeed of 20 meters per second.

The data flowchart of the kino-dynamic path planner is shown in Figure 6. The driver communicates with the operator interface by sending it waypoints through a UDP network. The waypoints are created by the kino-dynamic path planner, which performs calculations to ensure that the way-points are feasible. The operator interface is a graphical interface which displays the map of the area, the set of waypoints from the kino-dynamic path planner, the current location of the aircraft, and the science sites.

The autopilot, which is an actual UAV autopilot, handles all of the control dynamics. The autopilot inputs the aircraft pose, velocity, and aircraft parameters from the environment simulator, and inputs the target waypoint from the operator interface. The autopilot then solves the problem of how to fly the airplane to the next waypoint, and sends actuator instructions to the environment simulator. The environment simulator calculates motion and sensor data given the actuator data and sends this data to the operator interface and to the autopilot as feedback. The operator interface then returns the time taken to the driver so the driver will know how long it took to reach the waypoint.

The driver program takes in the initial high-level plan in the form of a tpn file. This is an XML representation of a temporally flexible state plan. An ex-

emplar plan is shown in Figure 5. The plan has lower and upper bounds for the duration of each activity. In our problem, the activities consist of traveling to a science site to do a science experiment. A temporally flexible plan is designed to allow the kino-dynamic path planner to re-plan its path without changing the overall activity plan (visiting science sites in a given order). After reading in the plan, the driver initializes and runs the kino-dynamic path planner.

The kino-dynamic path planner initialization consists of initializing the start time, the CPLEX representation of the obstacles, the kino-dynamic path planning solver for a given plane, the dynamics of each plane, the CPLEX representation for each plane, and the constraints (CPLEX is a optimization software package for solving Linear Programming and Mixed Integer Linear Programming problems). It then creates the CPLEX representation of the high-level plan. The constraints are a pointer to a vector which is passed to the vehicles, the obstacles and the high-level plan. This vector puts all of the different types of constraints into a data structure that is compatible with the CPLEX solver.

During the run() phase of the kino-dynamic path planner, it generates a guided plan. This function creates a set of waypoints for the autopilot to follow in order to reach the next science objective. The kino-dynamic path planner plans 25 waypoints at a time to be followed over the next 25 seconds (at one waypoint per second). It then allows the plane to fly for 18 seconds before re-planning. A 25 second planning horizon is used, which ensures that it will not crash within that horizon. However, since the planner does not look-ahead any further than that, it may leave the plane in a difficult position at the end of that horizon. For example, it may end up on a collision course with an obstacle. While the plane is at the 18th step in that plan, it is sure to have 7 seconds of safe travel in front of it. Therefore, if it generates a new plan at that point, it is fairly likely that it can avoid any object that it is approaching.

As mentioned, when the plane reaches the end of its plan (the 18th waypoint) we generate a new set of waypoints. In our implementation, we changed the code so that it is able to accept a new plan at this point. Since the adaptable mission planner only plans for the high-level goals (the science sites) we first check to see if the plane has reached one of the science sites. If it has not, then the kino-dynamic path planner re-plans as usual. If it has reached a science site, then we pass information back to the adaptable mission planner which enables it to generate a new plan. The adaptable mission planner needs to know the actual time that it took to get to the new science site so that it can calculate the fuel usage. It also needs to know which science sites have been visited. Once it knows this, it can then generate a new plan based on where the plane currently is and how much fuel it has left to travel between science sites.

Once a new high-level plan has been created, it is then accepted by the kino-dynamic path planner. In order to integrate the new plan into the system, we must remove only the relevant constraints from the constraint set, and then

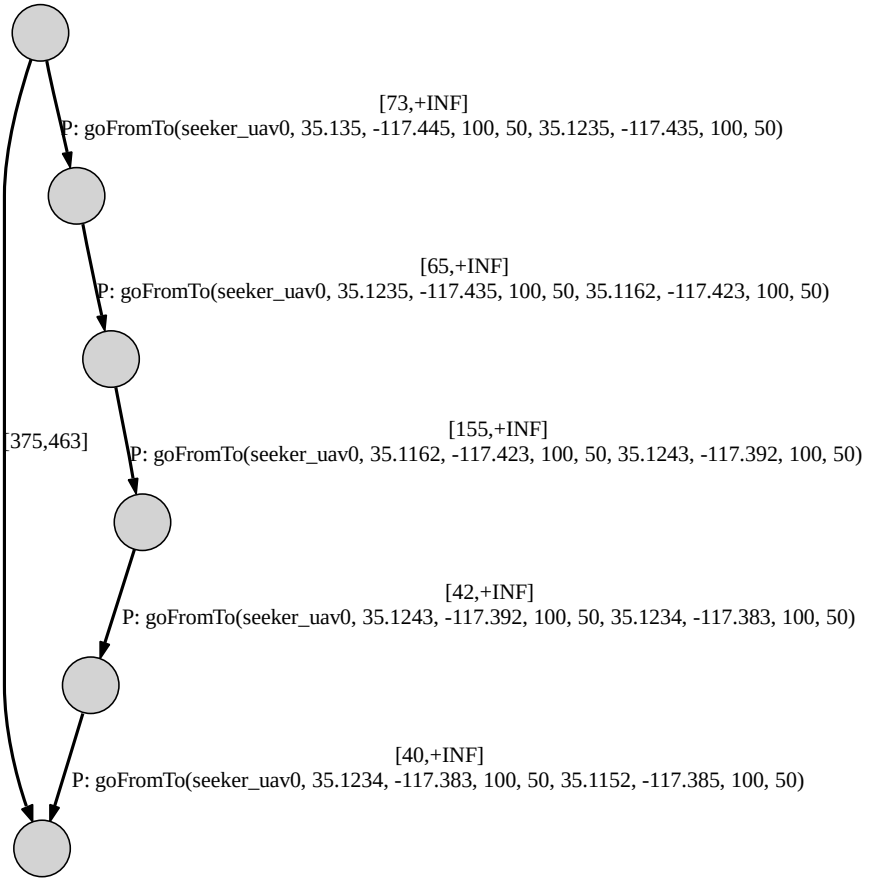

Figure 5: This figure shows an exemplar plan created by the adaptable mission planner. As an implementation detail, the adaptable mission planner creates the plan based on the estimated distance. However, certain activities' upper bounds must be set to infinity to allow the kino-dynamic path planner to quickly find a solution (a set of waypoints). The kino-dynamic path planner, however, respects the lower bounds and finds the shortest path which respects this lower bound. The kino-dynamic path planner also respects the overall mission constraints (upper and lower bounds).

re-introduce the new constraints based on the new high-level plan.

When this process has been completed, we then use that plan to create a new set of waypoints via the kino-dynamic path planner. These waypoints are fed into the auto-pilot. This process continues looping until the plan has been completed or a time constraint has been violated. Normally, the plan will finish as planned. However, though the high-level planner estimates the travel time to each science site, the plane may encounter obstacles which cause it to take longer than expected to get to a given science site. If the simulated travel time takes too much longer than the estimate, then the plane will run out of fuel and will not be able to complete the plan. In this case a constraint violation (too much total time taken) will halt the process.

2.4 Integration of Algorithms

The framework between the kino-dynamic path planner and the simulator was already in place, so what remained was to link the adaptable mission planner to the rest of the system. The input format for the kino-dynamic path planner is a tpn file, which uses an XML format. A piece of code was written to turn the vector output of the adaptable mission planner into a TFSP in the form of a tpn file for use by the kino-dynamic path planner. The kino-dynamic path planner was also modified to input new tpn files after reaching each node. This allowed for changes in the high-level plan.

3 Experiment Design

We have three different planning strategies to consider. We prepared a set of experiments in order to compare the performances of each strategy under different conditions.

The first plan type is the static finite-horizon plan. This strategy creates a plan to travel a certain distance (finite-horizon) at the beginning of a simulation, and then carries out the plan without making any changes (static). In our scenario, we have a UAV that flies until it runs out of fuel. The finite-horizon is made to match the amount of fuel, so that the UAV will complete the plan at the same time it runs out of fuel. This means that the finite-horizon plan actually covers the entire mission. In practice this plan uses our STSP solver once and feeds the result into the simulator, which then reports how much of the plan was successfully completed.

The second planning strategy uses an adaptable finite-horizon planner. This is similar to the static finite-horizon planner, in that it creates a plan that covers the lifetime of the UAV. It is different in that it re-plans after reaching each sci-

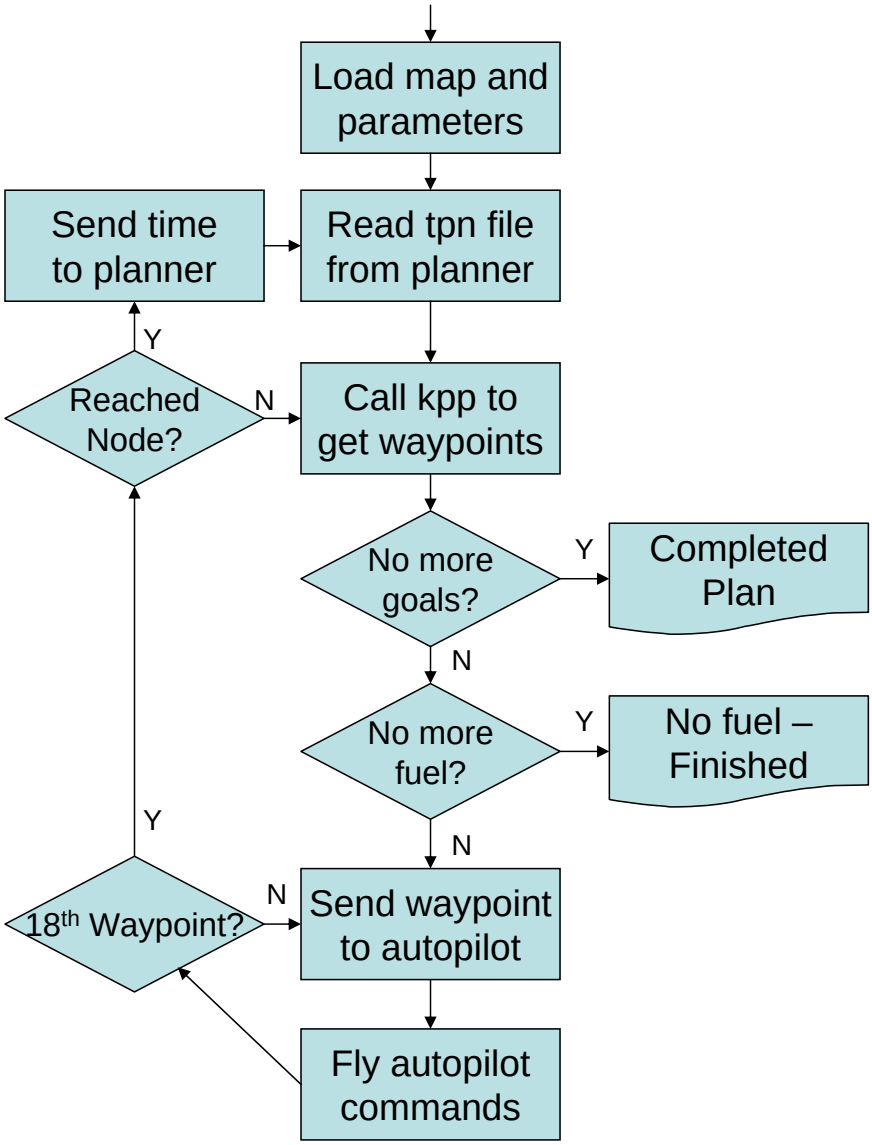

Figure 6: Flowchart of the Kino-dynamic System

ence objective, so it can react to changing assignments of utility to science sites. In practice this plan has the simulator call the STSP solver after each waypoint is reached. With our modifications, the simulator is then able to switch to a new temporally flexible state plan.

The third planning strategy is the greedy approach. This is essentially a one step receding-horizon planner. At each step it chooses the best node to travel to and then goes there. The best node is defined using a combination of travel cost and science utility. Our greedy planner tried to maximize utility/cost at each step. Different weightings would be possible for a greedy planner. For example, it could only consider distance and always fly to the closest point. It could also only consider cost and always fly to the point with highest utility. Different situations could be built where each of these designs performs better than the other, but we just chose an intermediate design that slightly prefers utility over cost.

Example 1: In Figure 7 we see a scenario with 3 nodes, A, B, and C, where the UAV starts at A. The distance between A and B is 2, the distance between A and C is 4, and the distance between B and C is 6. The utility of B is 3 and the utility of C is 6. If the UAV is allowed to fly a total distance of 4, then the utility maximizing greedy planner will be better. It will fly to C and get a utility of 6, while the cost minimizing greedy planner will fly to B and get a utility of 3. If the UAV is allowed to fly a total distance of 8, then the cost minimizing greedy planner will be better. It will fly to B, then C, and get a utility of 9, while the utility maximizing greedy planner will fly to C and get a utility of 6. It is not difficult to construct similar situations for different sets of greedy planners with different weightings between cost and distance.

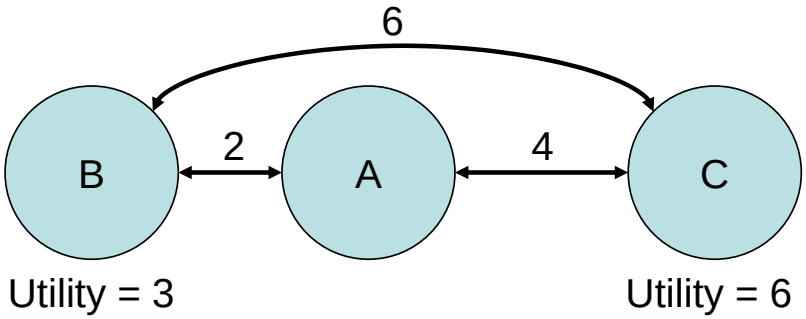

Figure 7: Simple example scenario

We chose our experiments to highlight the differences between the algorithms. In an unchanging environment, we would expect the adaptable finite-

horizon planner to make exactly the same plan as the static finite-horizon planner. This is because they would have the same knowledge at the beginning and create the same plan, and if the environment remained unchanged then the adaptable planner would have no reason to update its plan. In an unchanging environment we would also expect the finite-horizon planners to outperform the greedy planner, since they create an optimal lifetime plan.

When the assigned utilities start changing, this no longer applies. The finite-horizon planners are expected to suffer, since they no longer have perfect knowledge of the utilities. In particular, the static finite-horizon planner will be unable to react to changes in the environment, and the more severe the changes, the more random its performance will be. If the changes happen infrequently, we would expect the adaptable finite-horizon planner to perform fairly well, since it will be able to incorporate the changes into its plan. In the extreme case, however, the utilities will be completely random at each time step, and there will be no benefit from planning. In this case the greedy planner would perform well, since it would hit the high utility points as they appear.

Example 2: Imagine a scenario with 3 nodes, A, B, and C, where the UAV starts at A. The distance between A and B is 2, the distance between A and C is 4, and the distance between B and C is 6. The utility of B is 3 and the utility of C is 6. The UAV is allowed to fly a total distance of 8. The finite-horizon planner will travel to B, then C, to maximize its expected reward at 9. The greedy planner will travel to C and get a reward of 6, then run out of fuel. However, if after the first time step the situation changes so that C is only worth 1, then the finite-horizon planner will only receive a reward of 4, while the greedy planner will have already received its reward of 6. By incorporating changes it is very easy to construct situations where the greedy planner will perform better than the finite-horizon planner or vice-versa. It is also possible to construct situations where the static finite-horizon planner actually performs better than the adaptable finite-horizon planner. These situations basically involve a highly-changing environment where the static planner gets lucky while the adaptable planner gets unlucky.

In general, we would expect that the more frequent and significant the changes to utility, the less well the finite-horizon planners would perform. We would expect the greedy planner to not be especially affected by the changes, so its performance relative to the finite-horizon planners would improve. By running each strategy under different levels of change, we could see the crossover point of where greedy planning starts to outperform finite-horizon planning, and see where adaptable planning outperforms static planning. This is interesting because if different strategies perform better with different amounts of change, then it becomes useful to predict the amount of change. Using a model of expected change would allow for the adoption of the strategy that is most appropriate.

Thus we chose to test each strategy under different levels of change. We chose to implement change as a percent chance of happening to each node at each step. When change occurs, the utility of that node is swapped with another node at random. The reason we chose to swap utilities instead of assigning new utilities is that it keeps the magnitudes of the utilities the same. Basically it keeps the utilities in the range we want them without having to actively normalize them.

The percent change is a continuous and quantitative variable. This makes it possible to create relationships and plots between the change parameter and the performances of the planning strategies. The change parameter only affects frequency of change, and not magnitude. In order to look at magnitude, we would need to change the initial assignment of utilities. For example, a set of utilities with several high utilities and several low utilities would lead to large changes in utility when a high utility node gets swapped with a low utility node. Thus, we chose to examine a couple distributions of utility. This would allow us to look at the effects of magnitudes of change on the performance.

3.1 Experiment Parameters

The simulation was run with two sets of utilities. The first set approximates a Gaussian curve, so that there are several medium values and some high and some low values. There are eleven nodes, and the values 1, 5, 5, 10, 10, 10, 10, 10, 15, 15, and 20 were distributed among them. This fairly even distribution is expected to be similar to many real scenarios. The second set of utilities was created to give high magnitudes of change. The eleven nodes were assigned values of 5, 5, 5, 5, 5, 5, 5, 5, 100, 100, and 100. This distribution had three "jackpots" so each time a change occurred there was the potential that one node would drop a lot in value, while another would suddenly gain a lot of value.

The different planning strategies were run for different frequencies of change. The levels of change used were 0%, 5%, 10%, 15%, 100%, for a total of 21 different change frequencies. These levels of change represent the chance for each node to change its utility at each time step. Each of the three planning strategies was run 25 times for each change level, and the accrued utilities were averaged over the 25 runs.

In order to be fair to each of the three planning strategies, they were each run using the same random numbers. This means that the changes were calculated for each step of a run, and then all three strategies were run using that sequence of changes. This was done to keep one algorithm from getting lucky while the other algorithms got bad runs.

4 Results

4.1 Solution Walkthrough

In this section, we will show the results of one specific simulation run. This simulation run will demonstrate the benefit of the adaptable mission planning strategy. The main idea that will be demonstrated is the benefit of re-planning at each step in the TFSP (high level plan). A static planner constructs a TFSP at the beginning of the mission and follows that plan throughout the mission. It disregards the changing situation. However, an adaptable mission planner generates a new plan when necessary, which takes into account the amount of time that it actually took to get to the science site, as well as the knowledge gained from the visit. This knowledge is formulated as the updated set of utilities for the science sites.

As we stated earlier, our motivation is to create an autonomous UAV to conduct science experiments on Mars. To that end, we interfaced the adaptable mission planner with the Mars Plane simulator. The adaptable mission planner constructs a plan which is executed in the simulation environment. Figure 8 shows the operator interface of the simulation system. This interface displays the location of the UAV on a (representative) map of an area of Mars. The plane is currently hovering around the starting location, where the vehicle entered the Mars environment. The map depicts mountainous areas which the plane wants to fly *around*. It also depicts craters, rock outcroppings and cliffs. These are sites where the plane would like to fly *to* and conduct science experiments. For example, a science experiment may consist of taking and analyzing photographs of an area. The operator interface shows the progress of the UAV as it navigates through this environment.

The plan is constructed by the adaptable mission planner. The adaptable mission planner takes in a set of science sites (labeled B through J), the starting location (labeled A), a set of utilities for each science site and a maximum travel distance (5 nautical miles). The utilities reflect the science value of a given site. In this example, we start out with a set of utilities shown in Figure 9. The initial utilities reflect 6 low value sites (C, E, G, H, I, J) and 3 sites of high interest (B, D and F). The adaptable mission planner constructs the first plan which is reflected in Figure 9. This plan shows that the plane is able to hit all of the high value science sites. The red arrows indicate the path that the UAV will take.

The execution of the plan starts when the adaptable mission planner sends a TFSP to the kino-dynamic path planner. The kino-dynamic path planner takes that plan and constructs a set of waypoints (a low-level plan) which guide the UAV along the route designated by the TFSP. The set of waypoints are approximately one-second apart and respect the kino-dynamics of the plane. These waypoints are sent to the auto-pilot which essentially flies the airplane.

The plane flies to each waypoint in succession. To do this, the auto-pilot sends actuator instructions to the simulator which simulates both the UAV and the environment.

The plane initially proceeds from the Mars environment entry point (Site A) to the first science site (Site B). When the plane reaches a waypoint, it notifies the kino-dynamic path planner. The kino-dynamic path planner waits for the plane to reach the 18th node. When the 18th node is reached, it sends a new set of waypoints.

However, the kino-dynamic path planner must first check to see if a science site has been reached. If one has, it sends the status information to the adaptable mission planner which generates a new TFSP. Therefore, when the plane arrives at Site B, status information is sent back to the adaptable mission planner which uses this new information to generate a new plan. Specifically, it receives the actual travel time to the current site as well as the updated utilities. The travel time to a given node is estimated in the planner during plan creation. However, the environment simulator determines the actual time, which is a result of other factors such as the kino-dynamics, the initial pose or direction, as well as the unpredictable environment. When the plane arrives at Site B, it turns out that it took less travel time than expected. Therefore, the adaptable mission planner has more mission-time left than expected. Consequently, it creates a new plan (Figure 10) which is now able to travel to one extra science site.

The plane then proceeds to Site C, which is a crater. When the plane conducts the science experiment at Site C, it recognizes that this site is of very high science value. Consequently, the utility of all of the craters is increased. The adaptable mission planner then uses these updated utilities to generate a new plan (Figure 11) which is able to visit all of the remaining craters.

The plane then proceeds to Site G, which results in no change in the plan (Figure 12). However, when it reaches Site H, it returns a travel time which is lower than expected. Consequently, the adaptable mission planner is able to create a plan which visits one additional site, shown in Figure 13. The mission concludes when the plane travels to Sites I an J.

This walk-through demonstrates the benefit of the adaptable mission planner. In this example the plan is adapted to compensate for the actual travel time and the additional information learned when a site is visited. When necessary, the adaptable mission planner generated a new plan which took into account the actual travel time, as well as the knowledge gained from the visit. The result is a more effective planner.

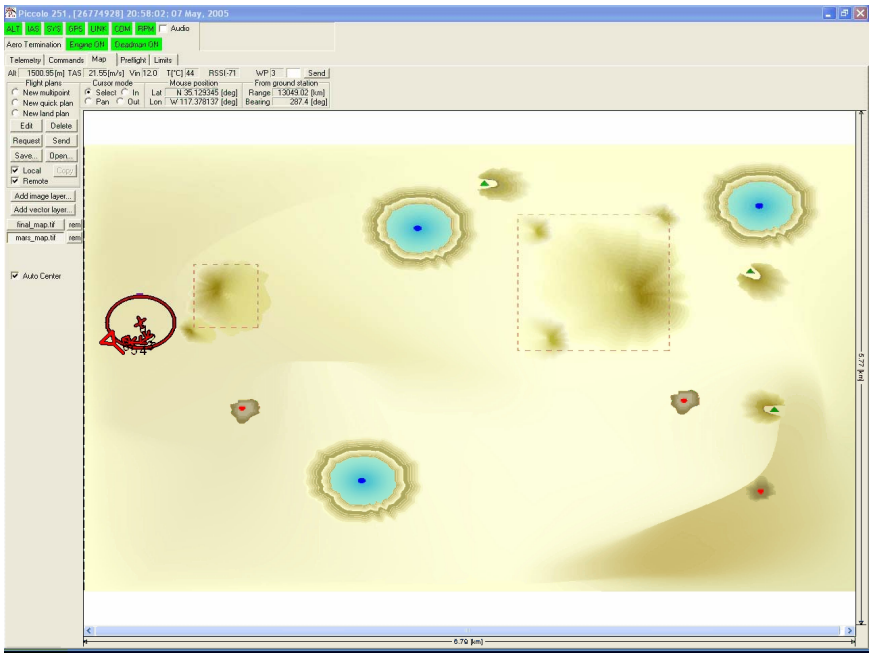

Figure 8: This is the operator interface that displays the UAV's position on the map with the science sites.

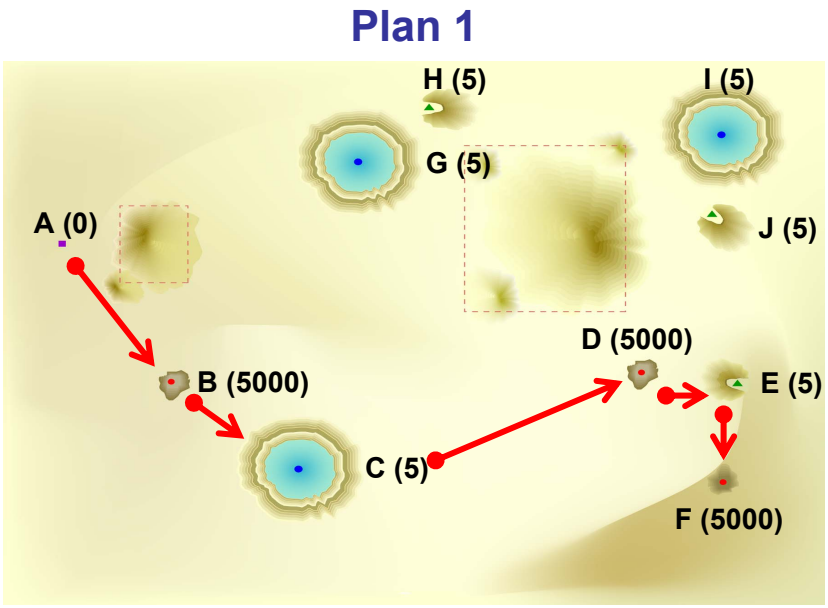

Figure 9: The above diagram shows plan 1 of the solution walkthrough.

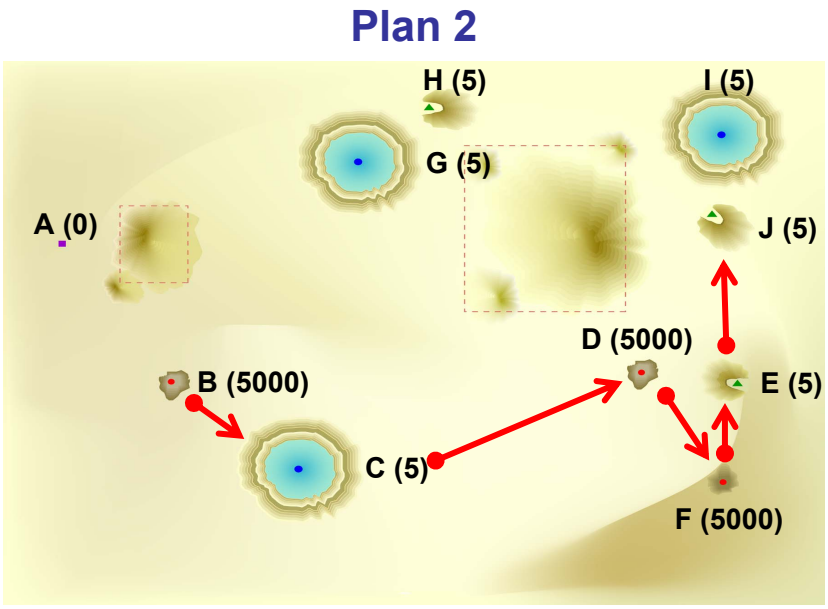

Figure 10: The above diagram shows plan 2 of the solution walkthrough.

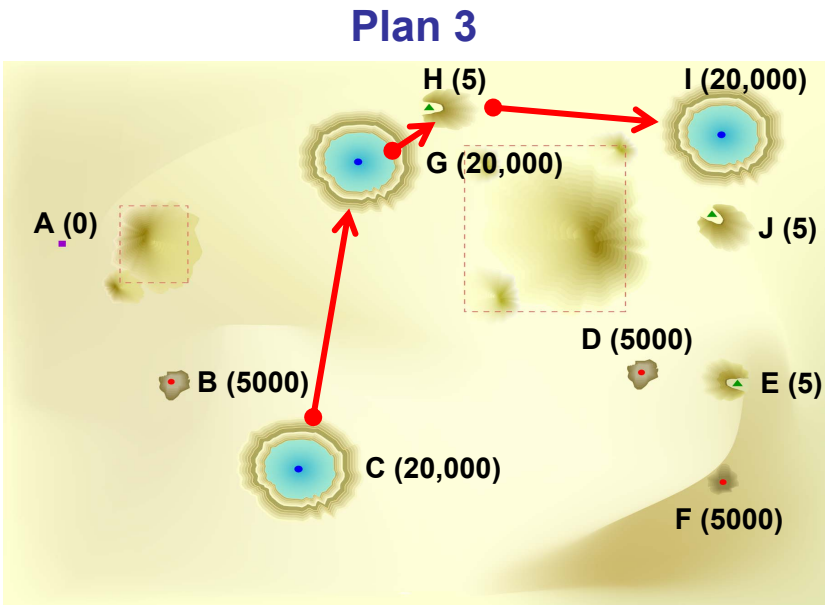

Figure 11: The above diagram shows plan 3 of the solution walkthrough.

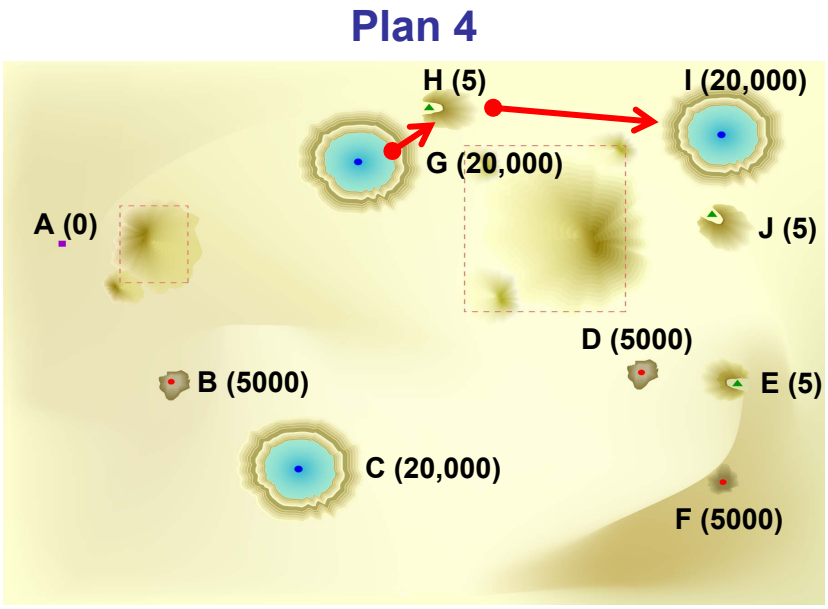

Figure 12: The above diagram shows plan 4 of the solution walkthrough.

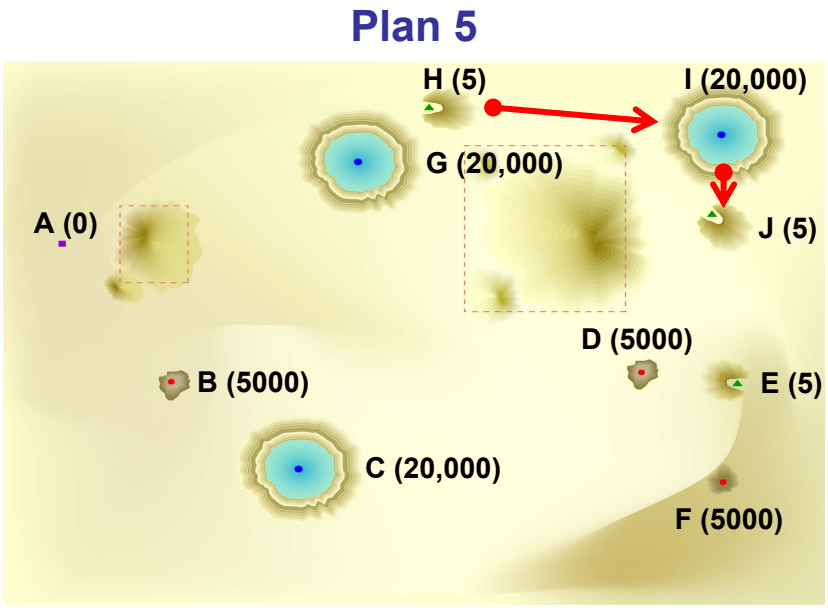

Figure 13: The above diagram shows plan 5 of the solution walkthrough.

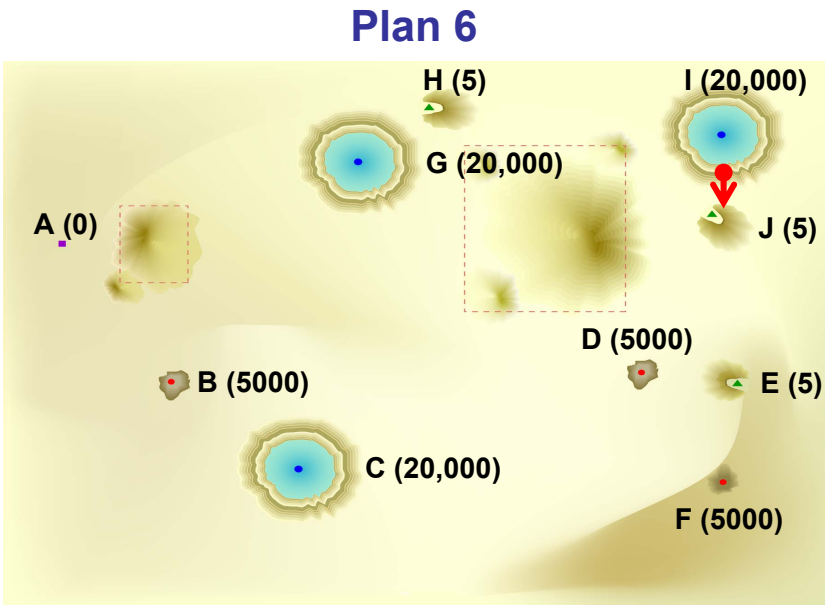

Figure 14: The above diagram shows plan 6 of the solution walkthrough.

4.2 Performance Metrics

When running each simulation, we end up with a total utility gained, which is simply a sum of the utilities gained as each point was visited. We measured (accrued) the utilities based on the current utility estimate for a given site. In other words, it is as if they were locked in when the previous node was visited. This means that if a point was visited when the estimated utility was 5, then no matter how it changed later it would still be accrued (to the planner's score) at a value of 5. The objective of this method is to measure how well a planning strategy works based upon what it knows. We are not attempting to measure the accuracy of the utility estimator. Different strategies make different trade-offs. For example, a finite horizon planner often forgoes a near-term reward in exchange for a higher reward later. A static planner chooses to ignore new information in favor of following its initial plan to fruition. All of these strategies have merit, however, our experiments show that the adaptive finite horizon strategy is most effective over a variety of conditions.

The main metric for comparing the performances of the different planning strategies is the total utility. This is intended to be representative of the total science value, which is the main goal of a mission to Mars.

4.3 Analysis

Figure 15 shows the results for the Gaussian distribution of utilities. The rate of change is along the x-axis and the average total utility gained is on the y-axis. The blue line is the static finite-horizon plan, the green line is the adaptable finite-horizon plan and the red line is the greedy plan (adaptable one-step receding-horizon plan).

For a zero rate of change the static planner performs as well as the adaptable planner. This is because they are doing the same thing. The adaptable planner takes into consideration new utilities. However, since the utilities don't change, this has no effect on the performance. Both of the finite-horizon plans outperform the greedy plan (for a zero rate of change). This is because the finite-horizon planners are optimizing over the entire mission. They trade off near-term rewards in exchange for a higher overall gain, whereas the greedy planner is simply doing what is best right not.

As the rate of change increases the performance of both finite-horizon planners decreases. The performance of the static planner drops most quickly. This demonstrates the deficiencies of the static planning strategy. However, the adaptive finite-horizon planner performs the best during a moderate rate of change. This highlights the strength of that strategy. After about 50% percent change, the static planner performs worse than the greedy planner and the adaptable finite-horizon planner performs about the same as the greedy planner. This is

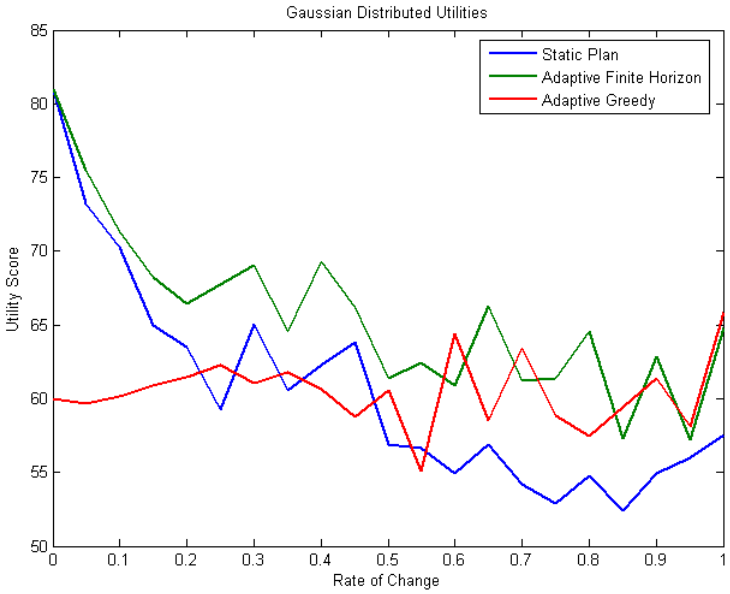

Figure 15: Total Utilities for a Gaussian Distributed Scenario

where the greedy planner's performance is the strongest. However, the adaptable finite horizon planner still performs approximately as well as the greedy planner.

These results are very interesting. As expected, the static planner outperforms the greedy planner under stable conditions and the greedy planner outperforms the static planner under unstable conditions. However, the most interesting thing that these results show is that the adaptable planning strategy is good in all situations. It is as good as the static planner under stable conditions, it is better than either planner under moderately changing conditions, and it is approximately equal in performance to the greedy planner under very unstable conditions.

You may notice that the results get very noisy for higher levels of change, despite being averaged over 25 runs. This is because when the system is chaotic, the utilities gained are very dependent on chance, especially for the static planner. As a final note, the two finite-horizon planners have curves that have some of the same bumps at the same places. This behavior is due to the fact that they were run using the same set of random conditions as described above.

Figure 16 shows the results for the distribution of utilities with 3 jackpots. If we look at only the curves for the finite-horizon planners we see much the

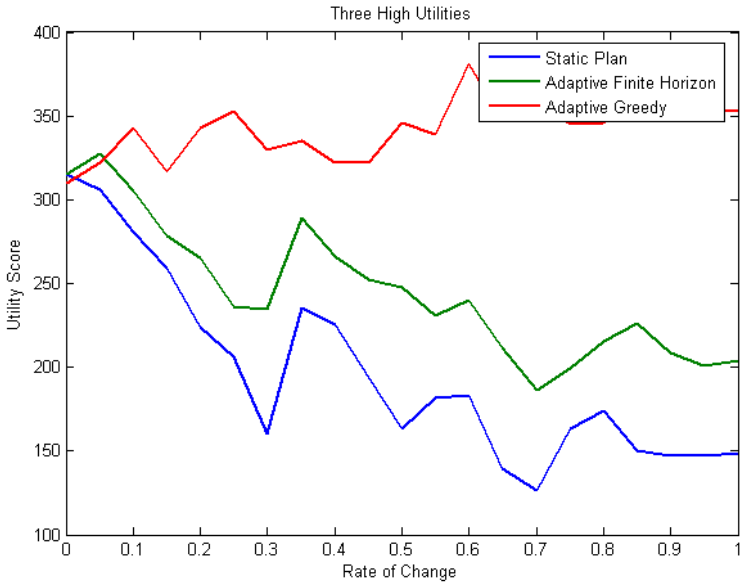

Figure 16: Total Utilities for a Scenario with three high utilities

same behavior as before, showing that they behave the same for high magnitudes of change as for low magnitudes of change. The adaptable planner gains a higher utility than the static planner with higher levels of change, and both finite-horizon planners fall and then stay fairly constant. The two finite-horizon planners again have approximately the same shape as each other as they were run using the same random sets of changes.

The greedy planner performs almost as well as the finite-horizon planners for no change, and then immediately does better as change is added. This is because this problem setup is very well suited for the greedy planner. The greedy planner can always go through and pick up the top three points. The remaining rewards are minor in comparison. When change is added, the greedy planner actually performs better, because it can potentially pick up the jackpots multiple times as they move around. Thus the greedy planner seems to perform well for high magnitudes of change.

This result is interesting as it shows how influential the utility distributions are on the experiment results. This corresponds to the results found in Hasegawa's work[2], in that when the problem setup contains a handful of extremely high utility values relative to the rest of the graph in a changing world, the greedy planner will outperform finite-horizon planners.

Essentially, this utility distribution is especially well suited to the greedy

planner. The Gaussian distributed utility problem, however, is not well suited for the greedy planner because the planner must optimize over a greater number of nodes in order to achieve good performance. It remains to be shown which utility model (and planning horizon) is reflective of the real world.

5 Conclusions

5.1 Impact on community

The major contribution that this work provides is a novel application of the above algorithms. We present a complete system that extends the kino-dynamic path planner to handle changes to its high level goals.

We also present a framework for evaluating high-level mission planners with different utility belief models. This framework can be used in the system engineering of a UAV in evaluating the benefits of different forms of planning.

5.2 Future Work

Improvements can be made to the STSP solver that is used in the adaptable mission planner. The current implementation of the OCSP/STSP solver returns inconsistent answers when plans are generated that do not cover all of the nodes. Once that has been completed, further improvements to the speed of the OCSP/STSP solver can be done by using Conflict-direct A* in place of the Constraint-based A*[3] algorithm used to solve the OCSP.

For other future work, the adaptable finite-horizon and greedy strategies could be blended. When change occurs infrequently we would expect the adaptable finite-horizon plan to perform best, but it could still benefit from a little greediness. This greediness could be implemented using discount factors. Essentially a fully greedy planner uses a discount factor of 0 (full discounting), while our finite-horizon planner uses a discount factor of 1 (no discounting). Intermediate values would allow for varying levels of greediness, and we could expect different levels of change in the environment to have different optimal discount values in the planning algorithm.

An extension of this work would be the autonomous creation of new nodes or science sites, and the decision making process necessary to decide when a new node is justified. Also, the work can be extended to allow the continuous re-planning as the UAV flew between nodes.

We explored how the different strategies performed under different levels of change in utility. Another parameter that could be changed is the estimates of

the distances between points. One rationale for changing these distances would be to have the initial estimates be incorrect, and become more refined over time. Another rationale would be if the targets actually did move, for situations such as fire-fighting UAVs on Earth. Thus an interesting topic to examine would be the level of benefit from re-planning as the estimates of the costs change.

5.3 Summary

One of the advantages to fully autonomous robotics is the ability to do in-situ planning. These plans can be adaptive and react to changes in the environment. We have demonstrated that an adaptable planner performs better than a static planner when changes in utility are modeled. We have also demonstrated that the adaptable planning strategy performs well at all levels of change assuming that utility distributions are Gaussian.

In order to perform adaptable planning we have used a hierarchical method that combines a high-level planner with a low-level planner. The high-level planner that we used was an adaptable mission planner for science site selection, while the low-level planner was a kino-dynamic path planner for vehicle actuation. This complete planner was successfully run on a simulator, and represents a novel application of these algorithms.

References

[1] T. Leaute and B. Williams. *Coordinating Agile Systems Through The Model-based Execution of Temporal Plans*. Massachusetts Institute of Technology, 2004, accepted to the International Workshop on Planning under Uncertainty for Autonomous Systems.

[2] B. Hasegawa. *Continuous Observation Planning for Autonomous Exploration*. Masters of Engineering Thesis, Massachusetts Institute of Technology, 2004.

[3] B. Williams and R. Ragno. *Conflict-directed A* and Its Role in Model-based Embedded Systems*. To appear in the Special Issue on Theory and Applications of Satisfiability Testing, accepted in Journal of Discrete Applied Math, January 2003.