AIAA Infotech@Aerospace 2007 Conference and Exhibit
7 - 10 May 2007, Rohnert Park, California

AIAA 2007-2972

Human-Robot Team Task Scheduling for Planetary Surface Missions

Maxime F. Ransan[1]
Laboratoire D'Analyse et d'Architecture des Systemes

Ella M. Atkins[2]
University of Michigan

Future manned space exploration missions will require collaborative activities with human astronaut and robotic agents. Proposed mission planning and execution architectures rely on reactive task planning, focusing on real-time information sharing between ground personnel, astronauts, and rovers. We present a planning/scheduling strategy that autonomously optimizes initial task plans/schedules over the human-robot team based on specified mission goals, but that also scales computationally to enable reactive replanning based on modifications to mission goals, revised resource utilization profiles, anomalies/emergencies, and astronaut directives. Collaborative exploration task models and search control algorithms are presented and evaluated in terms of computational complexity versus solution optimality. The implemented software was evaluated over teams of simulated and real rovers collaborating with a human companion. These tests demonstrate that the planning tool can reliably trade execution time for solution optimality when required and that the planner/scheduler is able to capably respond to detected anomalies.

I. Introduction

Astronauts and robots must collaborate effectively to safely and efficiently explore harsh extraterrestrial environments. The team goal is to efficiently complete a set of science or construction/maintenance activities at potentially disparate locations. Each agent (rover or astronaut) has a limited set of competences that constrains the individual agent's abilities but that enables the aggregate team to perform a wide range of tasks. For example, every astronaut may be able to pick up a rock but only "geologist" astronauts are able to identify which rocks are of scientific interest. Typical rovers have a suite of sensors and more sample storage capacity than an astronaut but would be more limited in their ability to select targets (e.g., rocks) of interest and rapidly collect desirable samples.

Multiple competences can be required to perform even a simple task. For instance, a rock sampling task might require not only a geologist to identify and pick up the rock, but also a rover to store the sample and return it to the base station. The planning tool must decide which agents can perform the task and when they will execute the task together. Scenarios ranging from sampling to habitat construction may require a specific task ordering (precedence). For example, before collecting a rock sample in an unknown area, it would be advantageous to first take high resolution pictures of the surroundings, select site(s) of interest, and then organize astronaut/rover teams with appropriate competences to explore sites of interest. Although rovers may encounter unexpected environmental conditions and improbable internal states, a robust robotic system will follow the specified plan to the best of its abilities. The very creativity and adaptability that make an astronaut a valuable explorer challenge a robot-astronaut planner. Although trained astronauts will do their best to accomplish assigned tasks, they are more likely than rovers to deviate from a planned task schedule due to the imprecise models used to estimate speed, traversal paths, and resource use (e.g., oxygen). Astronaut safety must be a top priority, even at the expense of goal achievement. Future rovers and astronauts will likely carry backup life support equipment for emergency situations (e.g., astronaut depleting his/her oxygen or unable to move due to injury). As team manager, either human or AI planner/executive should therefore be able to detect the emergency situation and respond to it in a quick and appropriate manner.

Another scenario where astronauts might behave unexpectedly is opportunistic exploration, situations in which they choose to deviate from the current plan to explore a site/perform a task they perceive as higher-priority. Since astronauts are indisputably the superior "creative" intellect given current technology, a planning system must support such deviations. Because the astronaut chooses not to accomplish the original planned task set, the plan is no longer valid. Both emergency and voluntary plan deviations require rapid detection and replanning/plan repair capabilities, primarily for safety in the former case and for efficiency, especially given cooperative tasks, in the latter case.

---

[1] PhD Student., Laboratoire D'Analyse et d'Architecture des Systemes (LAAS-CNRS), 7 avenue du Colonel Roche 31000, Toulouse, France, Student Member.
[2] Associate Professor, Aerospace Engineering Dept., University of Michigan, Ann Arbor, MI 48109, Associate Fellow.

Unexpected cooperation is yet another scenario that could arise. An astronaut could require assistance from a rover to complete a task, or a rover could lose a capability (e.g., broken sensor) and require assistance from an astronaut or rover to continue its task or return to base. For example, a "geologist" astronaut may opportunistically encounter a rock outcropping of potential interest but requires specific sensors or tools only possessed by a rover teammate to analyze, extract, or stow the geological sample. Although such situations must be handled expediently to prevent inefficiency for the team and frustration for the astronaut(s), safety is again the top priority.

Due to power and life support constraints as well as the inherent dangers of Extravehicular Activity (EVA), maximizing exploration productivity over the mission is critical. In planning terms, given a set of tasks or goals, it is important to identify and successfully execute a plan that maximizes the number of tasks accomplished while maintaining safety and minimizing overall execution time. Given traversal requirements, this problem can be related to the Multi Traveling Salesman Problem[1] but has significantly more constraints. In our situation the salesmen are not allowed to go everywhere, and "cities" are ordered, which makes traditional heuristics impossible to apply. Even if capability constraints reduce the search space, the problem of finding the optimal task allocation remains NP-hard, implying exponential time to compute the solution. Obviously, for large-scale, long-duration missions, exponential computational time renders full plan/schedule optimization an unrealistic requirement.

Reliable and efficient contingency response is also an important problem to address. Unexpected events that occur during EVA must be detected and reactively managed. A challenge is then to provide a planning tool that is able to compute, within a given time constraint, a valid plan that takes into consideration the specifics of the "unexpected" emergency situation. A "coordination executive" must also be guaranteed to detect at least safety-critical emergencies and to efficiently relay updated response plans to the team. Acting as a mission control or base station, a planning system must be able to plan actions that coordinate activities among all members of the astronaut-rover team. Planning parallel activities can also require extra computation and dedicated data structures, especially when multiple agents must cooperatively execute one or more tasks.

This paper describes a centralized planner/scheduler tool to manage a hybrid rover-astronaut team, focusing on real-time contingency response. Collaborative exploration scenarios studied include observation, sampling, and rescue tasks. A search strategy with branching factor control trades planning time with search space completeness. The planner reacts quickly to anomalies when required but also can build optimal plans that minimize deployment time while maximizing science return translated to goals/tasks. A six-wheeled rover was constructed with stereo vision-based navigation, path planning for target/waypoint tracking and obstacle avoidance, and onboard control, enabling navigation in unmapped environments. A human astronaut interacts with the real rover and a group of simulated (virtual) rovers. Rather than work offline with the scheduler through a collaborative scheduling tool, the scheduler allocates a default set of tasks to the astronaut but then provides the capability for the astronaut to redirect their activities in real-time based on perceived opportunities and hazards as well as his/her evolving preferences.

Below, following a review of planning/scheduling technologies, astronaut and rover models are presented in the context of state features, tasks, constraints, and costs. A search algorithm with branching factor control allows adjustment of the optimality-time tradeoff for to support offline and online (reactive) scheduling. The planner is situated in a real-time plan monitor and dispatching system, connecting multiple physical and simulated rovers as well as astronauts. Results from a series of simulation and hardware-based tests are presented to evaluate performance, particularly for real-time reaction to a suite of astronaut and hazard-driven anomaly scenarios.

II. Planning/Scheduling Background

A planning process identifies a set of actions that transforms the world from an initial state to a desired goal state or states. Due to the large number of possible states, this problem is usually NP-hard, requiring time exponential in the number of actions and features to compute a solution. Typical algorithms reduce the search space thus improve worst and average-case execution times. Classical planning techniques deal with restricted-state transition problems approximated as deterministic (each action results in a unique new state), static (if no actions are applied the world remains the same), finite (the system has a finite number of states), and fully-observable (the state of the world is known). Classical planning is also referred as STRIPS planning, one of the earliest planning systems.[2]

The most straightforward planning technique is *forward search* where states are expanded until the desired state is found. Forward search includes algorithms like A*, best-first-search, breadth-first-search, and uniform cost search. *Backward search* is the inverse of forward search: the search begins from the goal state and regresses to the initial state. STRIPS is a reduced version of backward search that identifies a satisfying solution relatively efficiently but that may not always find an existing solution due to an incomplete search strategy. Partial Order Planning (POP) was thus introduced,[3] maintaining computational efficiency similar to STRIPS but enabling solution to problems such as the Sussman Anomaly through least-commitment action ordering. UCPOP was perhaps the

most common POP planner and is still used, particularly for education purposes.[3] Plan-space planning (PSP) was produced through further evolution. In a PSP procedure, the initial plan is composed of two actions a_0 and a_∞. The procedure refines this initial plan by solving open goals and threats.

Because classical planning techniques do not scale to large problems, strategies such as Simple Temporal Logic and Hierarchical Task Networks were developed. Their main advancement was the exploitation of domain expertise within the planner, utilizing domain-specific rules to control and constrain the search space. Recent *planning-graph techniques* develop a solution through a procedure that bridges state space planning and plan space planning. The solution is specified as a sequence of action sets that incorporate precedence and exclusion constraints. *Graphplan* techniques were introduced in and are considered a significant representational advance.[4]

The development of control rules in Simple Temporal Logic (STL) has improved the performance of forward search algorithms. STL is a logical formalism that enables the search algorithm to prune nodes. STL was first introduced by Bacchus and Kabanza;[5] this implementation remains among the most efficient. The Hierarchical Task Network (HTN) is a practical technique with broad application.[6] The input of an HTN planning problem includes methods that can be viewed as domain dependent "recipes". The planning process consists of recursively decomposing non-primitive tasks until primitive tasks are reached.

As planning is the problem of finding a valid set of actions to achieve a goal, scheduling focuses on time and resource allocation for this set of actions. Historically, these two problems have often been treated separately; with the set of actions to perform tabulated for most scheduling algorithms. Such separation is optional, however. Scheduling can be seen as an optimization problem where different criteria are minimized depending on the application. Cost criteria minimized by schedulers include makespan (maximum ending time of the schedule), the total number of late actions, the peak resource usage, and total quantity of resources required. General multi-task, multi-processor scheduling problems are NP-hard, but approximation techniques have been implemented to scale large problems. Anderson et al provide a survey of scheduling methods.[7] Integrated approaches to planning and scheduling are also available, such as a method based on chronicles, the sets of timelines for every state variable.[8]

Classical planning assumes that every action produces a unique state but there are numerous real-world situations where this is not the case. When dealing with hardware, failures or off-nominal events can occur, implying that several states can potentially be reached with non-negligible probability when executing an action. This class of problem is called planning under uncertainty. The two solutions approaches are to avoid the complexity by detecting the unexpected state and computing a new plan from this state, and/or to employ a derivative of a Markov Decision Process (MDP) or Model Checker to build policies that apply over the most likely states.[6]

Multi-agent planning has received significant attention since many systems are composed of multiple entities, such as the astronaut/rover teams in this paper. The goal of general multi-agent planners is the global achievement of a common goal (set). Both centralized and decentralized approaches have been studied. In a centralized planner, each agent receives directives from a central entity and feeds back status and state information. This architecture enables a global view of the entire system and effective coordination through common knowledge. However centralized systems are only possible when all the agents can communicate and overhead may be incurred from the substantial information sharing requirements. Additionally, redundancy must be incorporated to enable robust, fault-tolerant operation. Distributed systems are more difficult to robustly design and maintain since no global view of the world is available to any agent. The main advantage is that goal achievement does not depend on a unique entity, ensuring greater overall safety in terms of redundancy. Although many multi-agent algorithms exist, a number of market-oriented techniques have been successful at task allocation across multiple agents.[9] Based on their current status each agent sends a bid or creates an auction, allowing agents to efficiently balance their task sets.

Space exploration researchers have begun to practically apply planning/scheduling algorithms. The Remote Agent Experiment demonstrated that planning/scheduling was possible onboard a spacecraft.[10] More recently task planning/scheduling systems such as ASPEN, incorporating an iterative repair algorithm to facilitate dynamic response, have been successfully operated long term and have gained the trust of scientists as viable and productive alternatives to manual schedule generation and oversight.[11] Activity planning and execution for a rover team has previously been investigated with systems such as CLEaR (Closed-Loop Execution and Recovery).[12] CLEaR relies on ASPEN, which incorporates an iterative repair algorithm to facilitate dynamic response. This system has been successfully demonstrated with multiple rovers but has not yet been extended to reason about direct collaboration with humans. Other researchers have specifically focused on the interaction between humans and robots, but typically rely on the humans for planning/scheduling rather than autonomous planning/scheduling tools.[13]

III. Planning System and Knowledge Representation

Analogous to a mission control or base station directing and monitoring a human/robot EVA team, a centralized planning and execution system has been implemented, as illustrated in Figure 1. The planner directs the activities of each mobile "agent" based on its goals and feedback from the agents (e.g., locations, available energy, capabilities). The planner also responds to directives issued by the astronaut team members. Such directives range from "I need help" to "I'm doing another task" to "Tell a rover to execute a new task" (e.g., defined by the astronaut). With the centralized planning structure, communication between astronauts and rovers is limited to data associated with the accomplishment of common tasks, with all other messages relayed through the planning/execution system.

Our application is composed of three interacting components: a team planner, a coordination executive, and the mobile agents deployed in the field (or simulated to facilitate experiments with many rovers). The centralized planning module takes the current state of the world and computes a *plan*, a scheduled set of high-level tasks or actions, for all mobile agents so that the maximum number of high-priority tasks can be accomplished by the team in a minimum amount of time. The implemented design-to-time algorithm can be configured to react quickly and possibly suboptimally or to consume the additional time needed for plan optimization. The execution module controls the flow of information between the central planner and the distributed set of rovers and astronauts, dispatching actions in real-time in accordance with the nominal policy and any dynamic updates. The execution agent also gathers and processes available state information from the rovers/astronauts to enable detection of off-nominal events. The executive is responsible for managing the reaction to off-nominal occurrences, either by adjusting the policy (e.g., task execution timings) or by notifying the planner that a new/modified plan must be created. For instance, when an astronaut requires help, the execution module will receive the information, detect the event, notify the rovers, and call for the planner to compute an emergency plan as quickly as possible. Each rover and astronaut is either a physical mobile entity or is simulated by a unique process that communicates with the execution module with the same protocol as a real agent would utilize. Rovers are assumed to understand high-level directives (task descriptions) and either to correctly execute them or return an annotated error message indicating the task was not successfully accomplished. The software is configured such that the simple simulated agents can be replaced with "real" robotic and astronaut agents.

Based on anticipated planetary surface missions, a set of state features and operators are defined to characterize the task-level goals, agents, and associated resource usage requirements and availability. Three state variable types were defined and uniquely indexed (i): tasks ($task_i$), rovers ($rover_i$), and astronauts ($astronaut_i$).

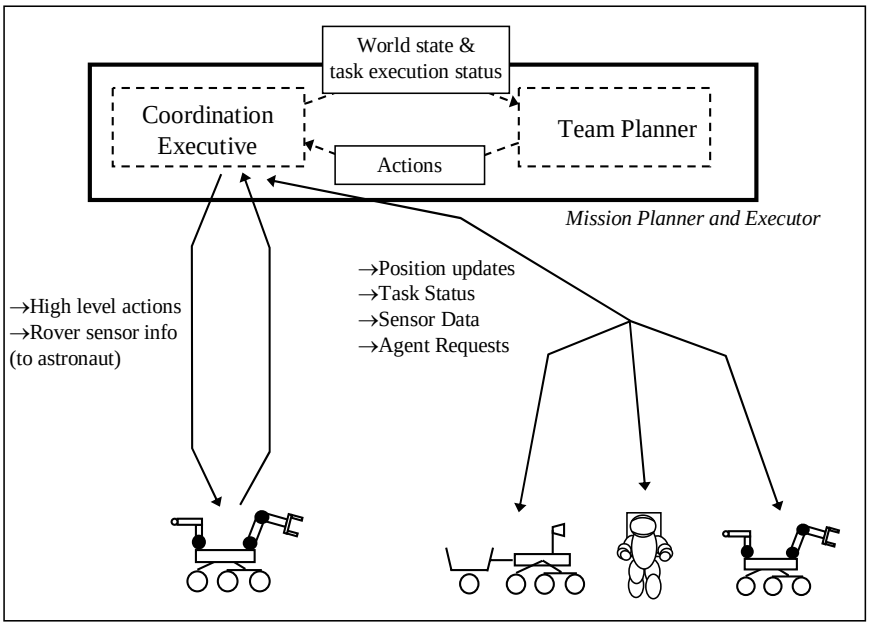


Figure 1: Planning & Coordination of a Multi-agent Rover-Astronaut Team.

Each task ($task_i$) is defined by the attributes {location, type, required competences, preconditions, energy, and duration}. Locations are discrete instantiated symbols defined by x and y coordinates (e.g., *loc1 is situated at (45.2 , 17.2))*. The tasks represent goals that the rovers and astronauts must complete and are defined as a certain *type*, implying a required set of competences needed for the task to be accomplished. Preconditions (previous actions) are provided for situations in which execution of a certain task requires the agent to have already executed another set of tasks. In the current implementation, each task has a constant (approximate) energy cost, representing the level of effort (resource expenditure) expected to accomplish this task. For the rover, this cost would represent power and fuel required, while for the astronaut, it would

represent physical exertion level and associated oxygen use. Constant expected duration (execution time) values are also specified for each task.

Similar to the task state variable, each rover has an *(x, y)* location and a set of competences. For this research, rovers are presumed to be fully able to navigate between task locations and to nominally be successful with the tasks they are equipped to handle. Rovers can also help an astronaut in an emergency (e.g., by providing oxygen or power when the astronaut runs low) or when the astronaut requires a specific tool onboard the rover. Rovers have a limited amount of onboard energy, a value diminished by each traversal and task execution.

Astronauts are capable of accomplishing many of the same tasks the rovers can complete, thus they are included as "mobile agent" resources within the planner. However, they also may elect not to execute the planned activities assigned to them. One of the reasons to have humans in the field is to capitalize on their superior sensing and reasoning abilities. While astronauts are trained to accept instructions, they also will not prefer to be strictly "pawns" of a computerized planning agent. In practice, the astronaut might decide to select a different task based on his preference or observations. In this work, we assume the astronaut will inform the planner of such deviations. We hypothesize in this research that the astronaut will prefer to provide input to the scheduling process in real-time rather than through an offline negotiation process.

In our world model, rovers and astronauts, more generally labeled "agents", can perform four different actions that change world state: (1) Navigate between locations (`Goto (agent_i, from_loc, to_loc)`), (2) Perform a task `(Execute (agent_j, task_k),` (3) Rescue an astronaut (`Rescue (agent_l, astronaut_m)),` or (4) Cooperatively perform a task `(Cooperate({rover1,astronaut2}, task4)).` A coherent list of preconditions must be true in the current state for an action to be applicable, including conditions ranging from "task and rover must be at the same location before rover can execute the task" to safety and competency checks. For instance, a rover assigned to an astronaut rescue will not be redirected until the rescue is complete.

A. Planner/Scheduler

We have implemented a recursive, time-controlled best first search algorithm to schedule the set of actions. A plan is typically a list of ordered actions with no indication of whether they can be executed in parallel or not. In order to deal with this issue we added two variables to each action: *start* and *end* time. Start time is the time when the action becomes applicable and end time is when the task is expected to end, deviation from which is dynamically fed back to the executor upon task completion. Each time a new action is added to the plan, the previous set of planned actions is evaluated to find the appropriate start time based on agent and task dependencies. For each agent to execute an action, the agent dependency function finds the time at which previously-scheduled actions will be completed. The latest end time over all collaborating agents determines the start time of the new action. Task dependency reconciles the preconditions that must be satisfied.

This work presumes energy is a constraint rather than a quantity to be optimized. Currently, the cost function *g(n)* is set to the largest end time over all mobile agent timelines for the set of actions from initial state to node *n*. The expected time required for task execution is tabulated or computed as a function of expected traversal distance. The planner's goal is to complete the maximum number of tasks in a minimum amount of time. Since limited time can be spent doing EVA, plan optimality in terms of mission time is a major concern, suggesting offline optimization will be highly useful to plan exploration activities. However, since optimal planning may require substantial (exponential) time, a more timely (real-time) search tool must also be available for use when needed. Our anytime implementation spawns different instances of best first search and stores results in a partial plan format. When a time limit is reached, the search process is interrupted and the best partial plan is returned. To find different plans in each best first search instantiation, we randomly select the node to expand rather than select the next entry in the cost-ordered open list. This strategy is effective despite the loss of cost ordering because all tasks have equal value in our current implementation, and best-first search prioritizes high-depth solutions due to its design. This method is analogous to optimization techniques that elect to explore different regions of the input space to avoid local minima.

The planning time limit also enables the planner to be tuned appropriately prior to execution, effecting a design-to-time strategy through branching factor control in concert with the anytime algorithm described above. Managing computation time is a critical issue, particularly when the plan must be adapted in real-time. Controlling the width of the search graph also enables control of optimality versus planning time. The selection among candidate actions for each node is made with the distance-based traversal cost function. For instance, when the branching factor is two, only the two best (closest) applicable tasks for that agent will be selected. This timeliness feature compromises optimality; however, since fewer partial plans are explored during the search the computation of the plan is much faster. This trade off is inevitable and considered acceptable for our application given the obvious need to rapidly replan, particularly when an astronaut requests emergency assistance. Branching factor selection is tightly coupled

to the time allowed to compute the plan. If a long period of time is allocated for the calculation of the plan then it is more advantageous to have a large branching factor. When developing a rescue plan, a small branching factor is better able to limit the search space with our closest-distance heuristic.

B. Coordination Executive

Once the centralized planner develops or updates a team activity plan, the *Coordination Executive* must communicate this plan to the mobile agents (human and robotic) and supervise/monitor its execution throughout. Coordination Executive design goals require the following capabilities: (1) Dispatch commands to each agent at the appropriate time, (2) Provide real-time monitoring capability both to dispatch tasks correctly and to identify discrepancies between actual and predicted task execution times and states, and (3) Direct the decision-making processes required to handle unexpected or anomalous events, requiring invocation of replanning and redirection of agents when a response has been planned.

The *Coordination Executive* is a multi-threaded C++ program as shown in Figure 2. The *Command thread* splits into as many sub-threads as there are agents in the field. Each sends the appropriate command when it becomes applicable. When an unexpected event is detected related threads are temporarily halted then redirected with a new activity list once available. The *update Agent i* threads are the communication threads that collect data from the mobile agents and update the Executive's knowledge of agent state (position, orientation, energy, etc). The *Monitoring thread* periodically checks the status of every agent and reports nominal task execution status including completion times as well as detecting unexpected events. This thread alerts every other thread when an anomaly occurs and a new plan needs to be computed. The *Planner / [Plan] Display thread* is in charge of first computing a new plan using the planning tool described earlier and then of displaying execution information at the screen. In the Figure 2 implementation, the Planner is an "agent" the Coordination Executive can invoke as needed. Upon startup, the Executive first calls the Planner with the specified set of tasks to accomplish. Once the planner returns the initial plan/schedule, the Executor begins directing all mobile agents in accordance with this plan, calling the Planner again as required to prepare contingency responses. The planner never directs or invokes the Coordination Executive except through the Executive's interpretation of the planned task schedules.

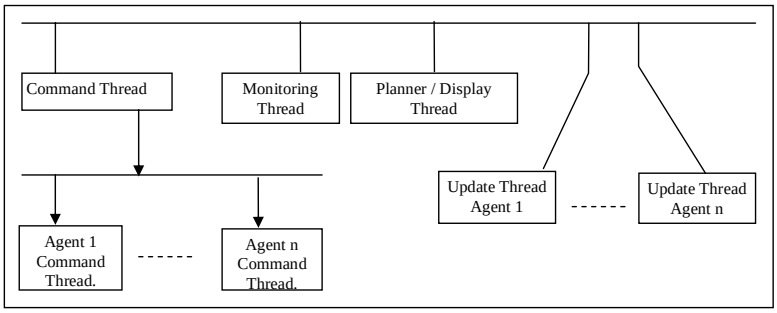


Figure 2: Coordination Executive.

IV. Results

Initial simulation-based tests were used to validate the scheduling algorithm and software architecture while assessing complexity and optimality of results. Next, simulations were used to evaluate the planner's ability to trade computational complexity with optimality. Tests with the real rover and a single astronaut agent were also conducted in indoor and outdoor environments. This paper focuses on the planner/scheduler system and its use for exploration missions; details of the six-wheel rover hardware and software, particularly the machine vision navigation and path planning algorithms, are available elsewhere.[14,15]

A. Multi-Agent Simulations

Each rover was simulated with a single process executed as navigation, task execution, message processing (RX), and update generation (TX) threads. Navigation includes nominal motions plus perturbations (noise) to provide more realistic deviations from the expected execution profile.

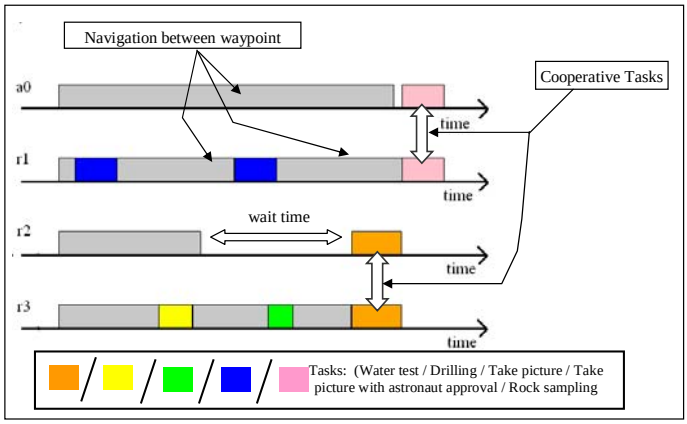


Figure 3: Base Scenario –Schedule and Expected Resource Use.

Each astronaut was simulated in an analogous fashion, except that the software was augmented to enable generation of requests for assistance and self-initiated deviations from nominal plans. As a baseline scenario, 3 rovers and 1 astronaut were assigned 6 tasks and were given 10 locations (4 initial agent locations plus one per task). Figure 3 shows the temporal representation of the plan with tasks inserted along a timeline based on their expected execution times. For this example, the planner's time limit was set to 10 seconds during which 2684 partial plans were stored using a branching factor of 4. The planner successfully finds a plan that accomplishes every task. All simulations were completed on a Dell Laptop with a 1Ghz Pentium 3 processor and 512 Mb of RAM under the Fedora Core 3 Linux distribution. Figure 4 illustrates plan execution by the simulated agents. We can see that the execution times are slightly different than the ones expected; this is due to the fact that in simulation task duration is randomly generated with a Gaussian centered on the expected value.

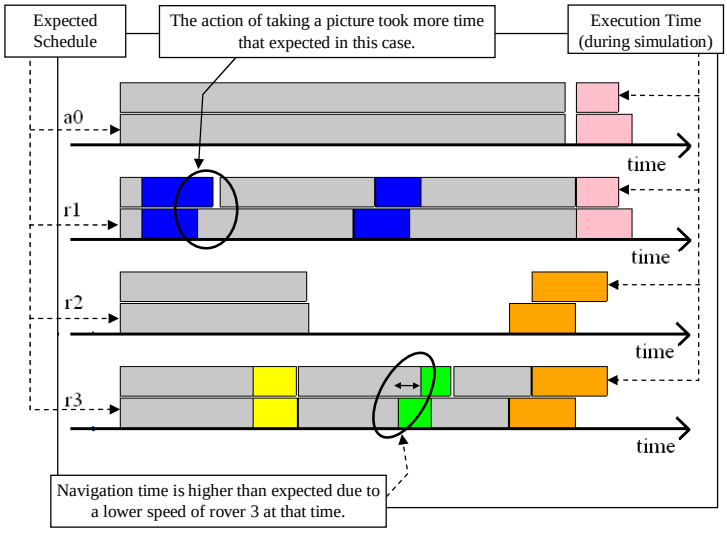


Figure 4: Planned and Actual Task Execution Times.

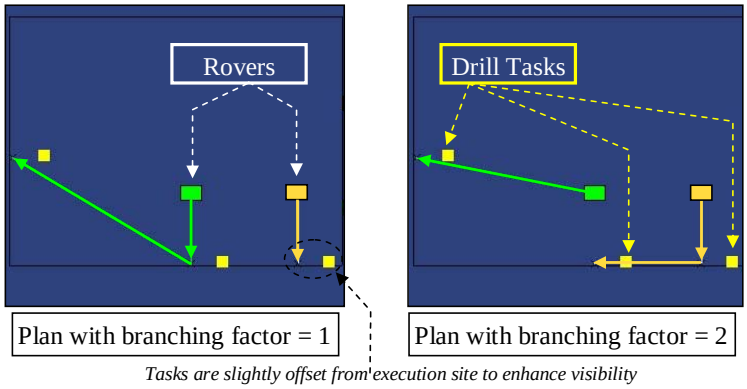
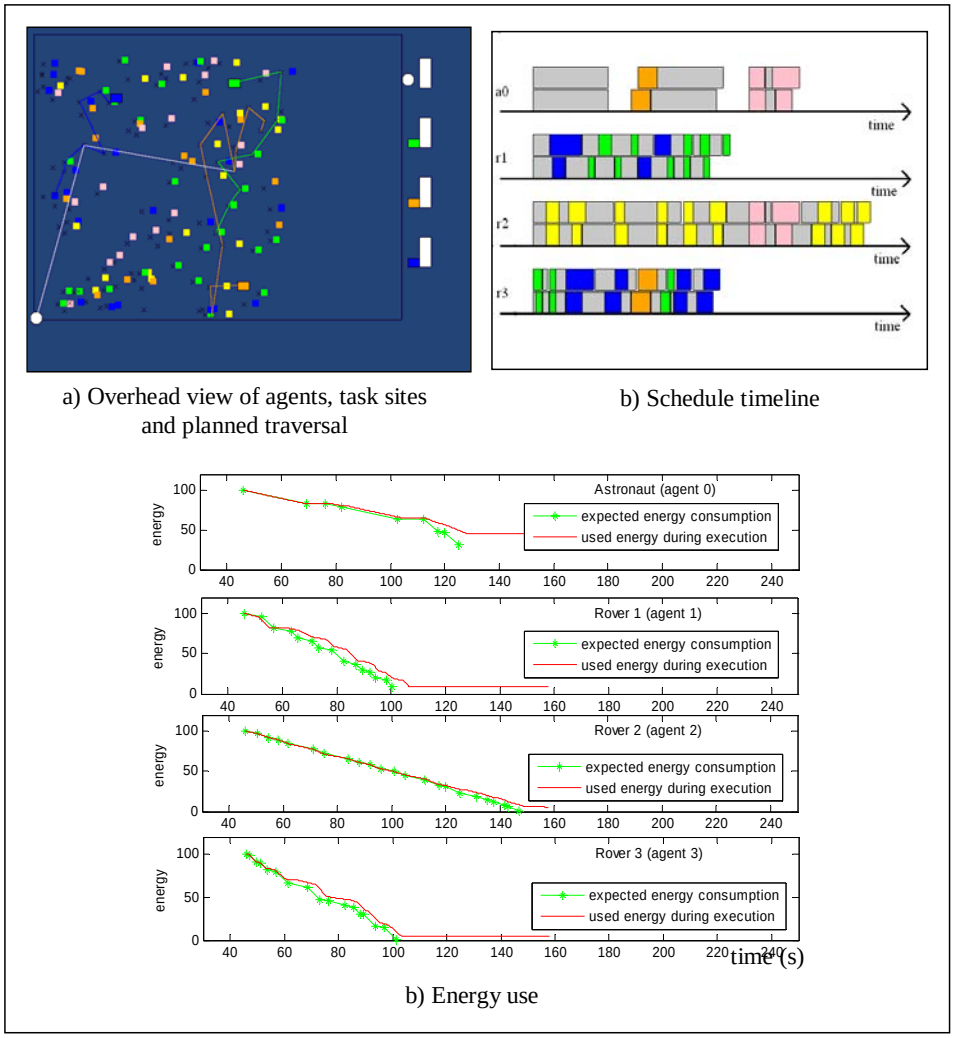


Figure 5: Plan variance with branching factor (top view).

A second test scenario demonstrates the importance of branching factor selection as well as the behavior of the planner. Two robotic agents are capable of independently performing the 3 drilling tasks to be performed at 3 different locations. Two branching factors were tested, with results illustrated in Figure 5. First the branching factor was set to one which implies the planner will only perform a classic best-first search without looking at other possible plans. During a second test the branching factor was changed to two, enlarging the search space. Two of the three tasks are close to the two rovers while the remaining one is far away. With a branching factor of 1, the planner selects the locations that are closest since the branches at a single level are ranked by past rather than future cost. Accordingly the two rovers are sent to the two closest tasks while the remaining task is scheduled at the next level and executed by the closest rover given their positions after the first task has completed. The best plan, however, takes advantage of the fact that two of the three tasks are very close and can be executed by the same agent. This optimized solution is found when the branching factor is set to two, enabling the search algorithm to find the alternate solution that sends one of the agents directly to the most distant task. Overall, the difference in execution time was 8.6 out of a total 29.9 seconds for the "slower" branching factor one solution. This trend was repeatedly observed as task complexity and problem size scaled, as illustrated by the 100-task example shown in Figure 6 which requires substantial time to fully optimize, but as illustrated 30 seconds is sufficient to generate a viable plan.

In addition to temporal constraints, an important capability of the planning system is to deal with unexpected events resulting from emergencies, opportunities, or poorly modeled task execution timings or effects. In this work, focus is placed on emergency response, which is the most critical situation to handle during rover-astronaut exploration activities. Consider the emergency situation where the astronaut calls for help. Although more generally specifics of the emergency would be required to ensure appropriate response, in this work the response to such an event is to redirect an agent with life support equipment to the astronaut, with preference to redirect the closest rover (the first-level response automatically selected by the planner given branching factor of 1). Figure 7 illustrates such a response with 24 tasks allocated among 3 rovers and one astronaut. All mobile agents possessed life support equipment. The planner was allowed 30 seconds for the original plan computation and 0.5 seconds to plan the emergency response, sufficient to generate the plans shown in Figure 7.



a) Overhead view of agents, task sites and planned traversal

b) Schedule timeline

b) Energy use

Figure 6: 100-task scenario: Partial plan and schedule after 30 planning seconds.

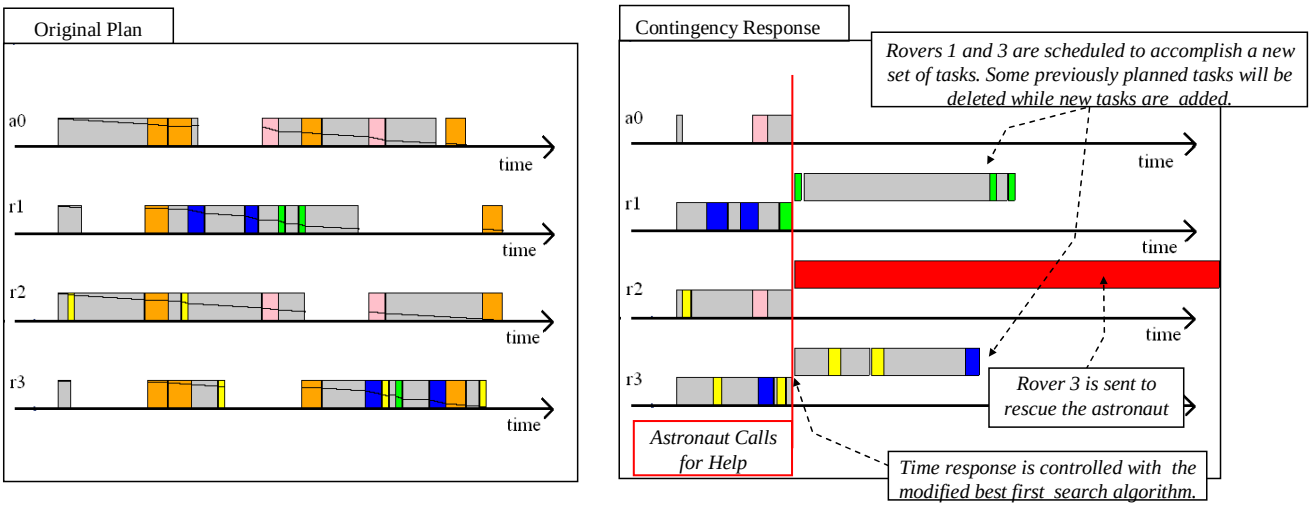


Figure 7: Contingency response to astronaut request for emergency assistance.

B. Computational Complexity vs. Optimality

To examine the trade off between optimality and computational time we further characterized planner performance over the example with 24 tasks. Not all of these tasks could be executed due to energy constraints, but depending on the ordering of the task execution, plans with 16 to 20 accomplished tasks could be found. Figure 8 shows the results obtained for different planner computation times. Each point in the graph is an average of five experiments and the standard deviation was taken into account when fitting the curve. The inverse of the standard deviation was used to computes weights in the error term:

$$error = \sum_{i=1}^{n} \frac{1}{\sigma_i^2} \left(y_i - f(x_i) \right)^2 \tag{1}$$

where y_i is the value of the ith experimental point, $f(x_i)$ is the fitted value of the ith experimental point, σ_i is the standard deviation of the ith experimental point. To compare the planner results of the different runs of the planner for the same problem we created a post-process plan scoring function. This function is a quantitative measure of plan quality, first ranking plans in order of number of tasks completed and secondarily ranking by execution time for plan sets that complete the same number of tasks:

$$score(plan) = \#\_of\_tasks + \frac{(execution\_time)}{upper\_bound - lower\_bound} \tag{2}$$

where *upper_bound* = the maximum amount of time the execution of *i* tasks could require and *lower_bound* = the minimum amount of time the execution of *i* tasks could require. This grading system ensures a plan with a greater number of tasks will always have a larger score, while secondarily incorporating task execution times. This function was only used to compare different solutions, not to guide the search process itself.

Figure 8 shows the tradeoff between optimality and computation time for this 24-task example. As expected, solution quality improves with planning time. For instance, ~10 seconds are required to plan 20 actions. The graph approaches an asymptote of 21 tasks which suggests the maximum score possible with a branching factor of four is not the "perfect" score of 24. This situation exists for two reasons: (1) Not all nodes are exhaustively searched in the graph with branching factor four, so the asymptote is not reached even with the higher time limits examined, and (2) Given resource constraints, no solution may exist in which all 24 tasks can be scheduled. For the 24-task scenario, planner statistics were collected with branching factors 2, 4, and 6.

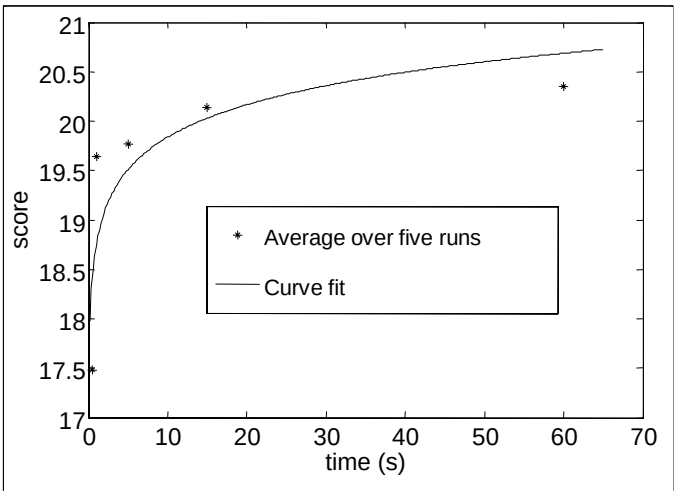


Figure 8: Optimality vs. Computation Time with Branching Factor=4.

Figure 9 shows curve fits computed from experimental results with branching factors 2, 4, and 6. These curves show two interesting dependencies of plan quality on branching factor. As expected, we observe that solution quality increases with branching factor. The second property is observed by focusing on the beginning of the graph where planning time is significantly restricted. In this case,

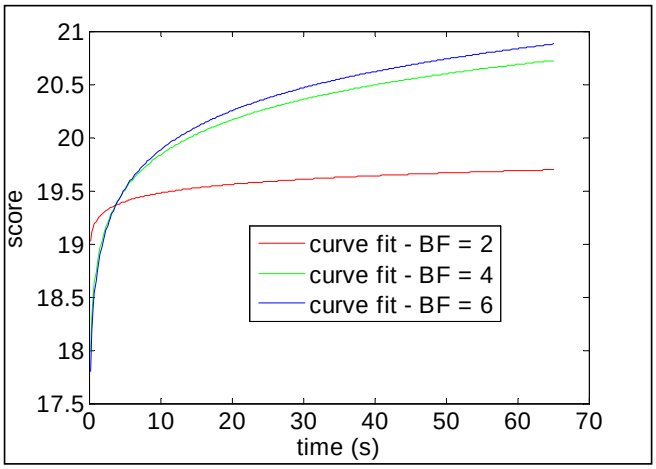


Figure 9: Influence of Branching Factor (BF) on Plan Quality.

small branching factors enable better planning results, illustrating that our distance-based heuristic used to define the reduced search-space is useful. With larger branching factor, the planner is more likely to explore costly nodes from a distance perspective, even though more nodes are eventually explored with sufficient time (thereby ultimately yielding an equal or higher-quality solution). Since the search is severely limited due to the time constraint, the solution will be somewhat random. Conversely, when the branching factor is small the few explored plans use nodes close to the shortest-distance heuristic. This test indicates the combined heuristic-branching factor control strategy is useful for fast-response emergencies as well as situations enabling substantial but constrained deliberation times.

B. Deployed Platform Tests

The first test scenario was designed to verify the planning system's integration with a real rover and a real human. The system was composed of those 2 agents, and two "Take a Picture with Approval" tasks were specified. Only the rover was equipped with a camera, therefore the planning problem was trivialized to the rover sequentially completing the two tasks but requiring approval from the astronaut (thus the astronaut had to focus attention on the "take picture" task at least briefly). At the discretion of the astronaut, a call for help may also be issued during plan execution to test both the planner and the rover reactions. The "Take a Picture with Approval" task consists of a sequence of three sub-actions for the rover: 1) tilt the camera pair to the horizon, 2) take a picture, and 3) tilt the camera back to its default configuration. During the execution, the coordination executive and task planner reacted as expected, effectively managing task execution and the flow of information. It also provided a correct response to the emergency call from the astronaut. Figure 10 shows the execution time line while Figure 11 shows the path followed by the rover during execution. We can clearly see that the rover is traveling more slowly than expected and that it does not follow the planned path. This is due to the fact that traversal time is computed using a straight line approximation between waypoints, without consideration of kinematic constraints. The closer the waypoints, the less accurate this approximation is, as the rover is taking a more circuitous path to account for required heading changes. As seen previously, dead reckoning results in increased position error over time, and orientation error also grows without magnetometer data (indoors). We can clearly see in Figure 11 that the rover, when returning to its initial position to rescue the astronaut, is approximately 1 meter away from where it believes it has traversed. During execution the rover performs a 180 degree change of orientation resulting in a significant heading error.[3] As error in heading has been observed to have significant influence in position error, this result was expected.

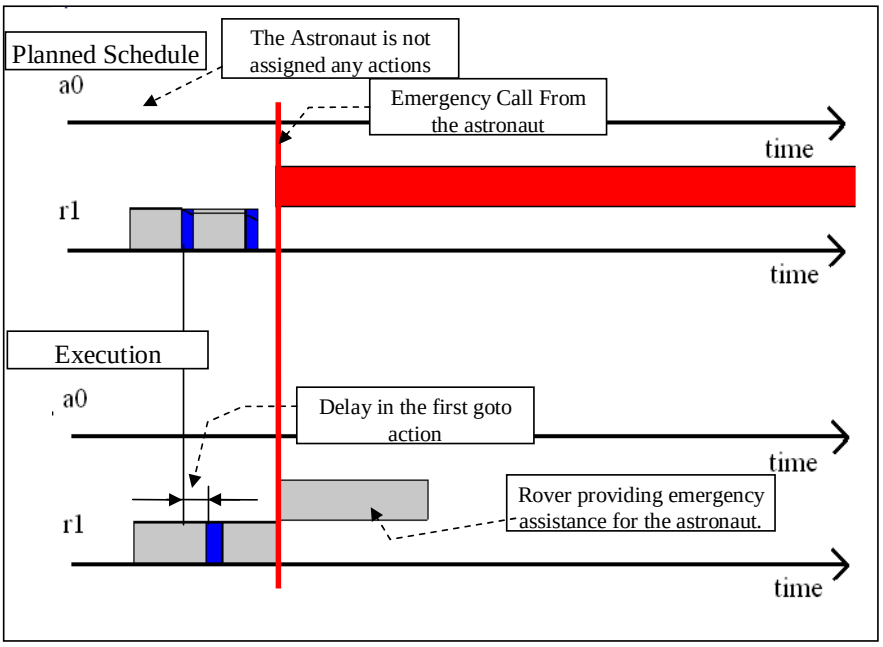


Figure 10: Plan and Execution Schedule.

[3] The magnetometer did not operate correctly in the lab thus heading error was more significant for indoor results.

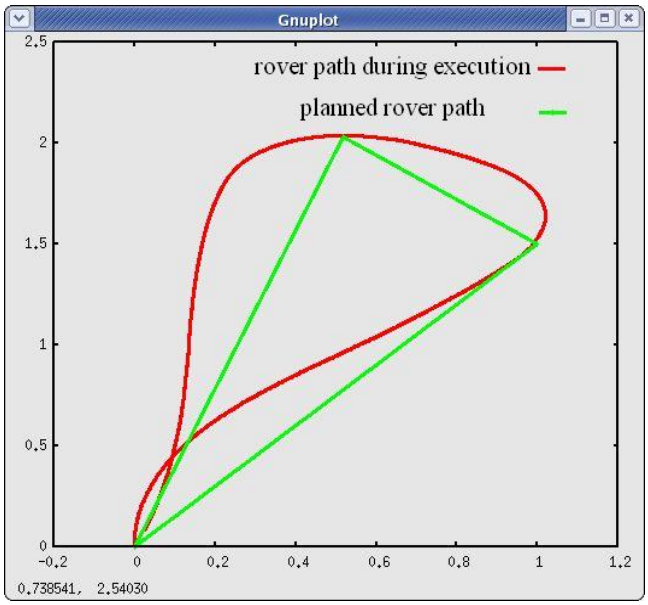


Figure 11: Rover path during execution compared to the planned path.

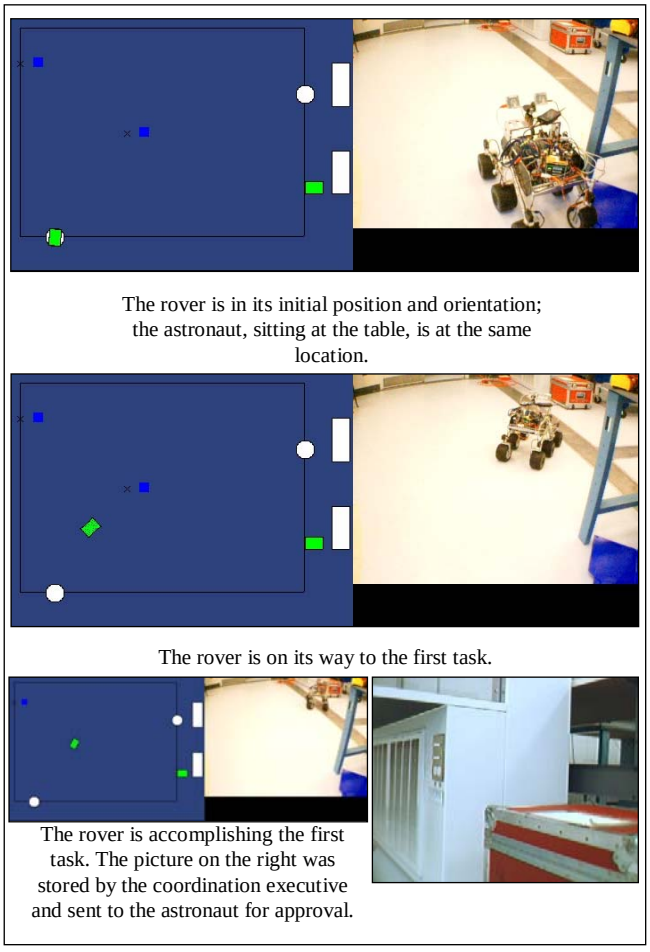


Figure 12: Execution snapshots from nominal plan execution.

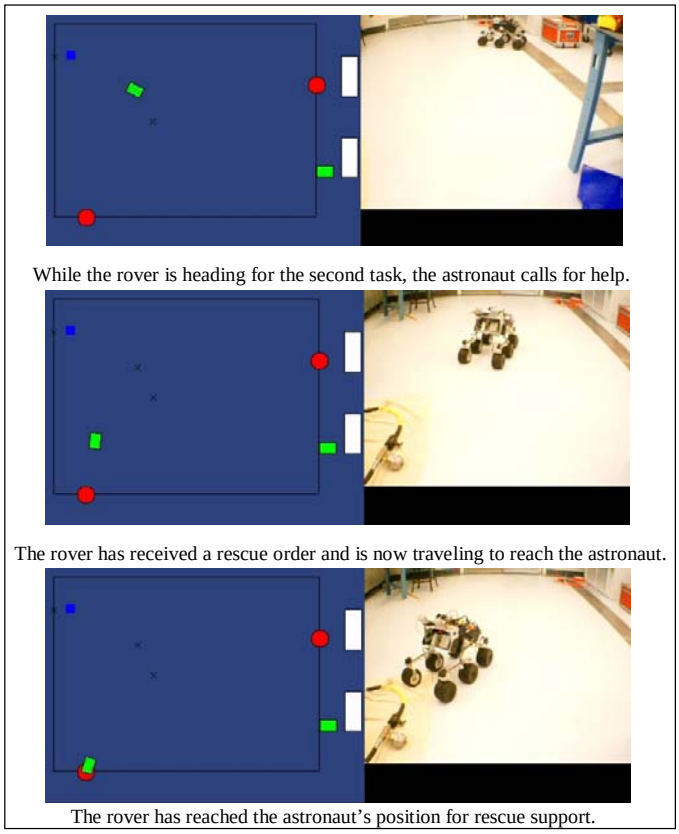


Figure 13: Execution snapshots for the emergency situation.

Figures 12-13 show information displayed by the planner/executive and the view from an external camera for the task execution and rescue operation. The two images are synchronized. The planner/executive process displays the position of the agents on the field based on the information it receives from them. Although the rover's position estimate may not be precise, the planner has no way to know the true position thus it displays position estimates as its best estimate of reality. This error is illustrated by the last image of the sequence. The coordination executive believes the planner has situated the rover in the same place as the astronaut, but by looking at the picture we can clearly see that the rover did not exactly return to its original position.

An outdoor test with rover and astronaut was also conducted. Figure 14 presents the internal map of the rover at different times during the traverse. We can clearly see the different obstacles in the last snapshot of the internal map. In the first image we see that the rover detects the four "obstacles" (markers) in front of it and chooses a path accordingly. A few seconds later, the rover perceives the second obstacle that appears to be intercepting the previously-planned trajectory. Consequently the rover re-computes a trajectory taking into account the terrain update. The magnetometer enabled more precise heading computation outdoor, but due to the terrain difference (mud, grass) more slippage was encountered. Nevertheless position computation was more accurate in the outdoor environment.

A final test was conducted to demonstrate the practical utility of this architecture as well as define its limitations. The test included the lab rover which is capable of not only taking pictures but also storing rock samples. The astronaut is able to identify interesting rocks and grab them. Two additional simulated rovers are included during the test. Figure 15 shows a picture displaying the experimental setup as well as the coordinate system. Seven tasks were required in this problem, including three "`Take Picture`", two "`Drill Rock`", one "`Water Test`", and one "`Rock Sampling`". The latter task is important because the competences of each agent were defined so that this task had to be performed collaboratively by the real astronaut and the real rover. Figure 16 shows the mission planner display including the computed plan. The planner was allowed 12 seconds for the plan computation. As the number of task was small, the branching factor was set to 8 so that no node was pruned during the search. Note the collaboration between the real astronaut and the real rover, with optimal (no-delay) execution requiring that they synchronously arrive at the same location to perform the rock sampling task.

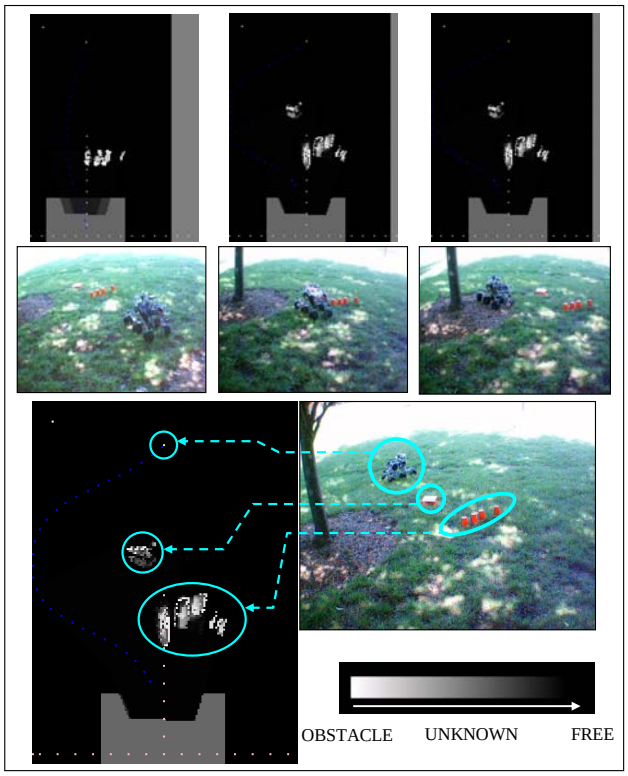


Figure 14: Outdoor test: External view & internal terrain map.

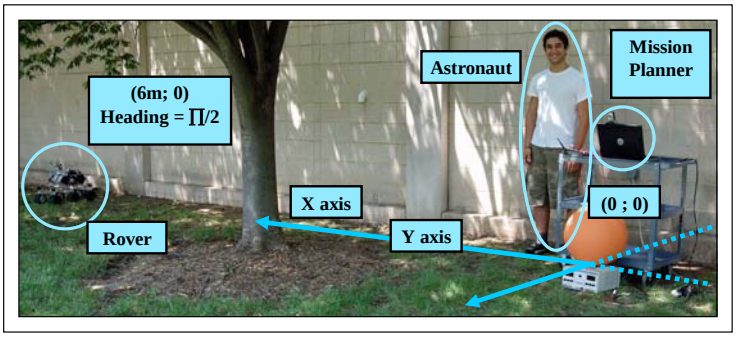


Figure 15: Outdoor Experimental Configuration.

As shown in Figure 16, the environment was designed such that simulated and real agents do not interfere during the plan so that the astronaut need not be aware of the virtual rover positions. Theoretically, nothing would prevent the planner from scheduling collaboration actions between a real and a simulated agent, although physical interaction would not be effective. For instance if the astronaut required a "`water sensor`" during execution of a task, the mission planner would have decided to send the simulated rover with this tool to the location of the astronaut. The astronaut would then be alerted (through the GUI) that the rover was present, but of course no actual sensor would be delivered to the real astronaut by the virtual rover.

This final test was successful. The rover with the help of the "astronaut geologist" is able to sample rocks and accurately return to base. The Coordination Executive succeeds in monitoring the activity of both the simulated and the real rover. Figure 17 illustrates the milestones achieved during plan execution. The primary difficulty encountered is that rover traversals require far more time than they are allotted in the schedule, requiring the astronaut to wait and requiring more total execution time than had been anticipated. In future work, the straight-line path approximation will be converted at least in the final plan to a more realistic path based on the best-available terrain and obstacle models.

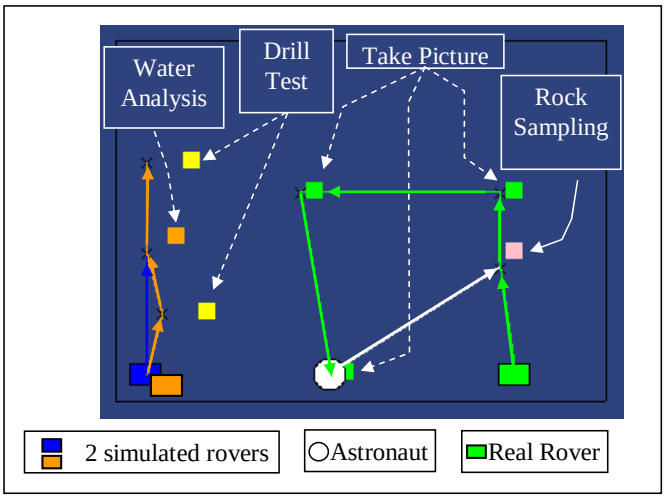


Figure 16: Collaborative Astronaut-Rover Plan.

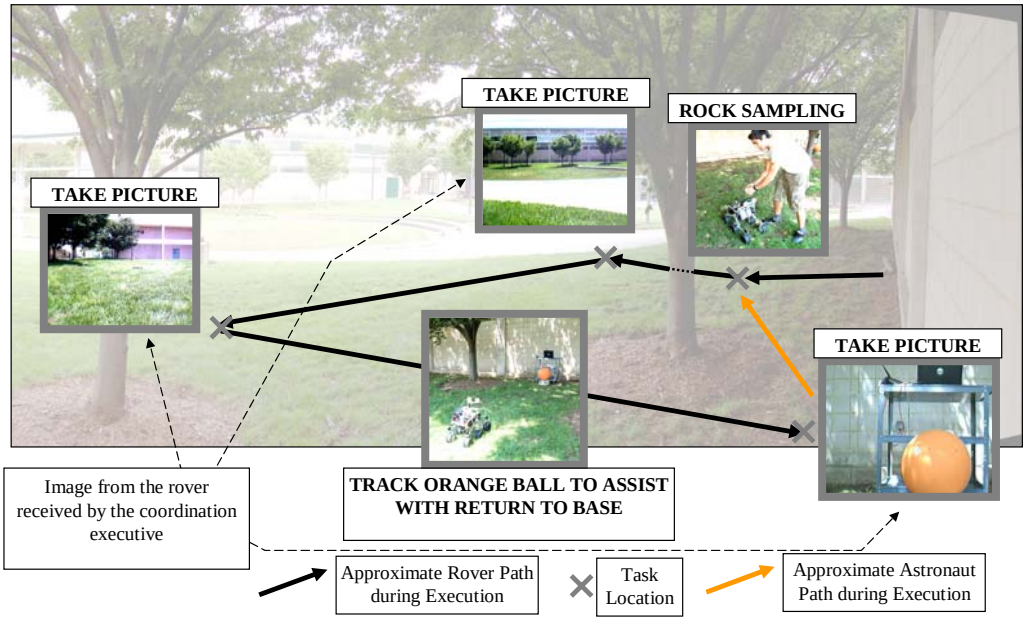


Figure 17: Outdoor Mission Scenario and Task Sequence.

V. Conclusions and Future Work

This paper has described a planning and execution system for astronaut-rover exploration teams tasked with collaboratively accomplishing planned tasks and maintaining safety and efficiency throughout. The three main research thrusts were the development of a flexible planning tool that can trade execution time for solution optimality when required, the implementation of a coordination executive to manage the multi-agent system, and the development and integration of rover hardware and onboard navigation software to enable real-world testing. The resulting system has been demonstrated capable of managing in real-time a team of astronauts and rover by scheduling optimized plans for each agent and by providing safe response to emergency or unexpected events generated by the environment or by directive from an astronaut.

Simulation results have demonstrated the robustness of the planner/scheduler over a wide range of situations and a large number of tasks. The implementation enables new users to easily implement new tasks and new agent types. Planner computation time is regulated, ensuring not only quick response when needed but also deep search of an optimal solution. A branch control strategy using a closest-point strategy has also been included in order to restrain

the search. Limiting the search space when little time is allowed for plan computation allows better plan quality while wide search space is used when a lot of time is available in order to explore more solution.

This work requires significant extension before applicable to real collaborative planetary surface missions. Future work will increase system adaptability and the range of possible scenarios. At this stage of the implementation, our model of an agent is simplistic. Parameters should be extended to include a comprehensive (relevant) set of skills, levels of attention, and willingness to be part of the team. Human and rover internal parameters may evolve during the execution; consequently the previous "optimal" plan may become less efficient over time. A combination of improvements to agent models and planning rules/heuristics must be made for this architecture to appropriately model and reason about the uniqueness of each human or robotic agent and his/her/its evolution during a mission. Collaborative dialogue is perhaps the biggest improvement needed for this architecture to work efficiently during larger-scale field trials. The current executive sends each agent involved in task accomplishment an "execution command". Even if the task is collaborative, no direct dialogue is now available since the goal was to efficiently place the correct set of agents at the right place at the right time. Direct communication between agents is a key to efficient task accomplishment, but this communication must be modeled in the planner/scheduler to realistically quantify resource costs and execution time requirements.

The ultimate goal of this and follow-on work is to provide an automated planning/scheduling system for collaborative extraterrestrial exploration that can take full advantage of each agent's capabilities while minimizing unnecessary delays. Humans and robots will soon be capable of routinely accomplishing tasks together. Planning/scheduling architectures and algorithms will improve the definition and execution of large-scale scenarios beyond what is possible with reactive plans.

VI. References

[1] D. S. Johnson, L. A. McGeoch, "The Traveling Salesman Problem: A Case Study in Local Optimization," *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra, eds., pp. 215-310. London: John Wiley and Sons, 1997.

[2] R. Fikes and N.Nilsson. "STRIPS: A new approach to the application of theorem proving to problem solving". *Artificial Intelligence*, 2(3-4): 189-208,1971

[3] J. Penberthy and D.S. Weld. "UCPOP: a sound, complete, partial order planner for ADL". In *Proceedings of the International Conference on Knowledge Representation and Reasoning (KR)*, pp 103-114,1992.

[4] A.L. Blum and M.L. Furst. "Fast Planning through planning graph analysis". *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1636-1642, 1995.

[5] F. Bacchus and F. Kabanza, "Using temporal logics to express search control knowledge for planning," *Artificial Intelligence*,116:123-191, 2000.

[6] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning, Theory and Practice*, Morgan Kaufmann, 2004.

[7] E. Anderson, C. Glass, and C.Potts. "Machine Scheduling". In E.Aarts and J.Lenstra eds., *Local Search in Combinatorial Optimization*, pp. 361-414 Wiley, 1997.

[8] S. Tabbone and D. Ziou. "On the Behavior of the Laplacian of Gaussian for Junction Models". *In Second Annual Joint Conference on Information Sciences*, pages 304--307, NC, USA, 1995.

[9] F. Ygge, H. Akkermans, "Decentralized Markets versus Central Control: A Comparative Study", *Journal Of Artificial Intelligence Research* 11 (1999) 301-333.

[10] N. Muscettola, P. Nayak, B. Pell, and B. Williams. "Remote Agent: To Boldly Go Where No AI System Has Gone Before", *Artificial Intelligence* 103(1-2):5-48, August 1998.

[11] S. Chien, R. Sherwood, D. Tran, B. Cichy, G. Rabideau, R. Castano, A. Davies, R. Lee, D. Mandl, S. Frye, B. Trout, J. Hengemihle, J. D'Agostino, S. Shulman, S. Ungar, T. Brakke, D. Boyer, J. Van Gaasbeck, R. Greeley, T. Doggett, V. Baker, J. Dohm, F. Ip, "The EO-1 Autonomous Science Agent," pp. 420-427, *Third International Joint Conference on Autonomous Agents and Multiagent Systems*, Volume 1 (AAMAS'04), 2004.

[12] T. Estlin, D. Gaines, F. Fisher, and R. Castano. "Coordinating Multiple Rovers with Interdependent Science Objectives," *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi Agent Systems*, Utrecht, the Netherlands, July, 2005 (CL#03-1202).

[13] T. Fong, C. Thorpe, and C. Baur, "Advanced Interfaces for Vehicle Teleoperation: Collaborative Control, Sensor Fusion Displays, and Remote Driving Tools", *Autonomous Robots* 11(1), July 2001.

[14] M. Ransan and E. Atkins, "A Collaborative Model for Astronaut-Rover Exploration Teams," *AAAI-2006 Spring Symposium on Human-Robot Teams*, Palo Alto, CA, March 2006.

[15] M. Ransan, *Design and Implementation of a Collaborative Model for Astronaut-Rover Exploration Teams*, Master's Thesis, Aerospace Engineering Department, University of Maryland, College Park, July 2006.