ICAR '97
Monterey, CA, July 7-9, 1997

A Self-Planning Methodology for Planetary Robotic Explorers

Shane Farritor
Department of Mechanical Engineering
Massachusetts Institute of Technology
Cambridge, MA, 02139 U.S.A.
smfarrit@mit.edu
(617) 253-5095

Steven Dubowsky
Department of Mechanical Engineering
Massachusetts Institute of Technology
Cambridge, MA, 02139 U.S.A.
dubowsky@mit.edu
(617) 253-2144

ABSTRACT

Planetary robotic explorers must plan and re-plan their actions as new mission and environmental information becomes available. Here an on-line action plan generation procedure is proposed. Action plans are scripts that include navigation, sensing, and task instructions. The plans are constructed from physically realizable actions, called action modules, that are assembled to produce a successful action plan. The approach is based on a hierarchical selection process, which includes a genetic algorithm, to select a feasible action plan. The proposed methodology attempts to fully utilize the physical capability in rugged terrain, while minimizing the risk of mission failure. The method is demonstrated in the context of an example task and some guidelines are presented.

Keywords: Mobile Robot Planning, Planetary Exploration

I. INTRODUCTION

Mobile robots are a key component to the exploration of planetary surfaces [1]. Prior knowledge of the environment and tasks of such systems is very limited. Long communication delays of planetary exploration makes tele-operation from earth impractical. A robot would need to plan much of its own actions based on sensor information that will be collected on-site by rover or lander based sensors [2]. Furthermore, such robots can be designed to be very capable in complex and rugged terrain, see Figure 1 [3]. It is important for a planning methodology to aggressively utilize this capability, while minimizing the risk of a mission failure from the system becoming hung-up or trapped.

Much research has focused on the motion planning for robots [4]. One major approach to the control of mobile robots that has substantial promise is often refereed to as behavior control [5]. Behavior control uses task achieving behaviors and an arbitration method to determine which behaviors produce the action of the robot [6]. Behavior control implementations tends to be very reactive in that they quickly respond to sensor inputs, but do not then plan actions far in the future. Other research has developed an autonomous planning system for a planetary rover expected to land on Mars in 1997 based on behavior control and human waypoint designation[3].

In this paper, a method for producing on-line mobile robot plans is presented that is based on an inventory of physically realizable actions. These action modules are assembled in a sequence or script to produce an action plan. This assembly is accomplished in a rational manner suitable for automation using a hierarchical selection process which includes a genetic algorithm. The key to this approach is that the selection process exploits the fundamental physics of the system and sensor based knowledge of the task and environment to effectively search for the most effective action plan. The method attempts to maintain some degree of reactivity, while utilizing all of the available sensor information. This method was developed in the context of planning for modular robots. For a discussion of modular robotics refer to [7,8,9,10].

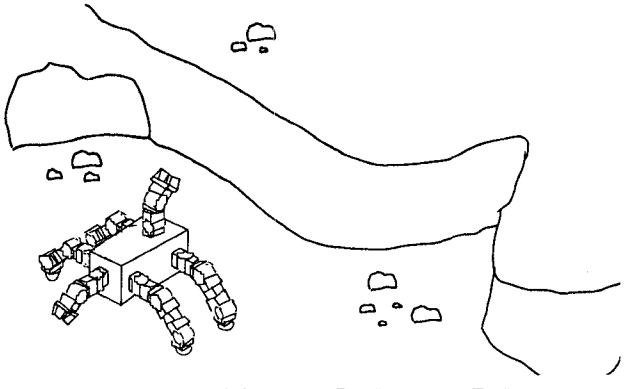

Figure 1: A Mobile Exploration Robot

This paper examines the application of this method to planetary mobile robots where issues such as computational efficiency are paramount. Some simple analysis of the approach is presented which suggests that it is more practical and effective than might be construed from a first order model of the process.

II. THE APPROACH

A successful action plan is one that allows the robot to complete the task objectives without violating any physical constraints of the robot or the task. These constraints can include actuator saturation, static stability, energy consumption, kinematic constraints, obstacle avoidance, etc. A successful action plan is produced by sequencing action modules chosen from an action module inventory. A very simple action module inventory, consisting of three motion modules, is shown in Figure 2. (A discussion of how to optimize the inventory for a given class of tasks can be found in Section VI.)

If these modules are assembled properly, and the appropriate modules are available, the robot will perform the task. A simple example of action modules assembled into an action plan is presented in Figure 3.

0-7803-4160-0-7/97 $10.00 © 1997 IEEE

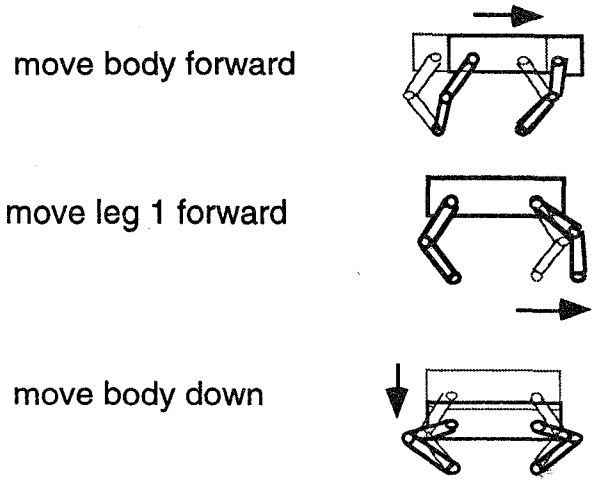

Figure 2: A Simple Action Module Inventory

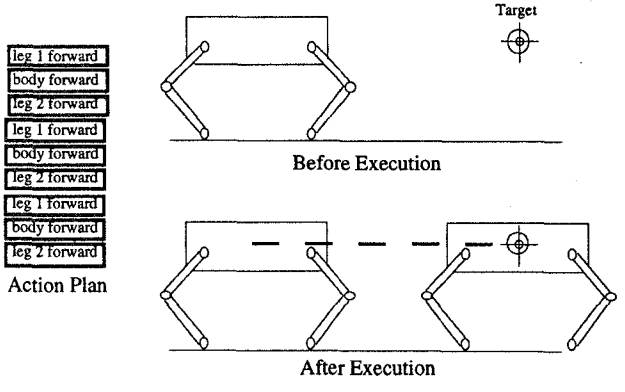

Figure 3: A Simple Action Plan or Script

In this paper, a method to automatically create a successful sequence of action modules, or action plan, is presented. For an action module inventory of any reasonable size, and an action plan of any reasonable length, the number of possible action plans is very large. The nature of the search space is discussed further in section 4. To search this space effectively, the algorithm uses a hierarchical selection process which reduces the initial inventory using task and module filters, then this reduced inventory is searched by a genetic algorithm to produce a final action plan. This process is now explained in the context of an example problem.

III. A CASE STUDY: STEP CLIMBING TASK

Consider the following 4 legged walking robot with 2 manipulators, shown in Figure 1, as it attempts to navigate a vertical 5" step. Climbing this step can be viewed as a sub-task of the larger exploration problem. The body of the robot is 8" x 4" x 4" and the limbs are 9" long.

The available action module inventory is shown in Table 1. This action module inventory would not be specifically selected for the task of Figure 1, but would be selected for an inventory of physical components for a range of tasks. For this reason, the inventory contains action modules for rolling, using solar arrays, etc. Such modules may not be pertinent to this task or even this specific robot configuration.

Table 1: Action Module Inventory

action #	description	action #	description	action #	description
001	body +x	N01	limb N +x	901	grasp object
002	body -x	N02	limb N -x	902	release object
003	body +y	N03	limb N +y	910	transmit video
004	body -y	N04	limb N -y	911	end video
005	body +z	N05	limb N +z	912	perform laser scan
006	body -z	N06	limb N -z	921	recharge batteries with solar arrays
011	body +roll	N10	limb N to initial position	922	communicate with ground station
012	body -roll	051	drive +x	923	sleep
013	body +pitch	052	drive -x		
014	body -pitch	055	turn left		
015	body +yaw	056	turn right		
016	body -yaw				

In our example, the robot has 6 limbs (4 legs and 2 manipulators). An action module such as N01, commands the robot to move limb N in the positive x-direction. This listing actually refers to 6 action modules, one for each limb of the robot (or 101, 201, 301, 401, 501, 601). Therefore, the action module inventory contains a total of 66 action modules. To efficiently search this space it is important to reduce the search space as much as possible as all possible combinations of these action modules is a very large set. This is reduction is accomplished using the hierarchical selection process.

A. Hierarchical Selection Process

1. Task and configuration filters. The first step in the hierarchical selection process is to apply module. These filters use simple tests, based on the fundamental physics of the task, to remove elements of the action module inventory before more complex evaluations procedures, such as detailed simulation, are applied. A small reduction at this level of the search results in a much larger reduction in the number of possible plans. This process eliminates large sections of the search space and therefore removes a number of sub-branches of action plans that need not be evaluated by more complex tests. For example, in this task no manipulation is required, therefore all manipulation action modules are removed. The selection process removes 14 modules that pertain to the motion of the manipulators (#101, 102, 103, 104, 105, 106, 110, 601, 602, 603, 604, 605, 606, 610), and 2 modules that pertain to grasping an object (#901, 902). During this sub-task the robot will not use a video camera or laser so modules #910, 911 and 912 are removed. Further, the task is of short duration, therefore modules number #922 and 923 are also eliminated.

The second step in the selection process is to apply robot design based module filters. If the robot is constructed of modular components the action module inventory may contain modules that do not pertain to this particular robot configuration. Here modules are removed that are not relevant for the robot configuration being used

in this task. For instance, the robot of Figure 1 does not have any wheels, therefore action modules that command the robot to roll forward, or turn left, are removed from the search space (#051,052,055,056).

2. Genetic Algorithm Search. With the design space very substantially reduced, a genetic algorithm is used to search for the best action plan. A genetic algorithm is a search process based on biological evolutionary heuristics such as selection, crossover, and mutation. This algorithm operates on a population of solutions, called a generation. Using the operators of crossover and mutation, the genetic algorithm evolves a better action plan with increasing generations. A complete description of genetic algorithms can be found in [11].

Here, a chromosome (list of actions modules) is used to represent an action plan. The order in which the modules, or genes, appear in the chromosome represents the chronological order that the robot will execute each action module. Because of the discrete nature of the method, the application of the genetic algorithm is very natural. The example action plan shown in Figure 3, with its corresponding chromosome representation, is shown in Figure 4.

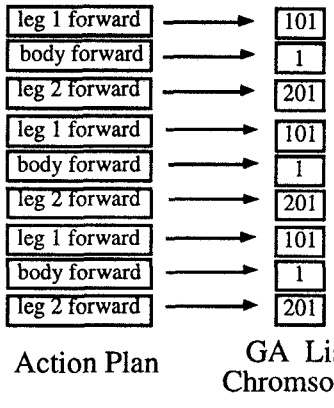

Figure 4: Chromosome Representation

The genetic algorithm requires a crossover operator, a mutation operator, and a fitness function to perform the search. The crossover operator is a process in which elements of one chromosome are combined with those of a second chromosome to produce a third, unique chromosome. In this application, a random module in the action plan along with all remaining action modules in the list, are exchanged with a random module of a second action plan and its following modules, thus producing a new action plan. This is referred to as single point crossover, the importance of using this type of crossover technique for robot planning is discussed in Section 4.

In order to maintain diversity in a population, a mutation operator is used. In mutation, a random module in a single action plan is exchanged with a random module chosen from the reduced inventory. Finally, a method to determine the quality, or fitness, of a solution is required. Here, a simulation is used to determine the fitness value of each action plan. The simulation executes the action plan to determine if it allows the robot to accomplish the task without violating any physical or task constraints. The plan is simulated until the task is completed or a constraint is violated. Then a numerical value of the fitness is assigned to the action plan such that a plan that advances the robot further towards the successful completion of its mission receives a higher fitness. The importance of assigning fitness in this manner is also discussed in Section 4. The exact form of the fitness function is dependent on the task and environment. For example, if a task requires a robot to move to a target, as in our example, a fitness function might be:

$$f = \alpha_1 |D| - \alpha_2 P + \alpha_3 \delta$$

Where: D = the distance from the robot to the target after execution of the action plan
P = the power consumed during execution
δ = 1 if the target is reached and 0 otherwise
α_i = weighting factor

This fitness function assigns a high fitness to an action plan that moves to its mission goal while consuming as little power as possible. If the task required other actions, such as manipulation, terms would be added that reflected the success of the plan in performing these actions.

The simulation checks for environmental interference, kinematic constraint violations, limb interference, static stability, power consumption, and actuator saturation. Since the simulation must be run each time an action plan needs to be evaluated, it is important to keep it as computationally simple as possible while still maintaining an acceptable accuracy.

B. Results of Step Climbing Task

With a substantially reduced search space, the genetic algorithm search is applied. The genetic algorithm used generations of 40 action plans, and the search continued for 8000 generations. The resulting action plan contained 189 action modules, and successfully moved the robot to the top of the step. In this action plan, a sequence of 118 action modules were required to move the robot to the step, and a sequence of 71 action modules were required to move the robot up the step. These 71 action modules are shown in Table 2 and the motion of the robot is shown in Figure 5.

Table 2: A step climbing action plan

script #	Action #	script #	Action #	script #	Action #	script #	Action #
1	101	19	110	37	6	55	110
2	1	20	13	38	301	56	13
3	401	21	401	39	101	57	301
4	1	22	301	40	1	58	401
5	101	23	5	41	310	59	1
6	301	24	14	42	5	60	13
7	5	25	1	43	14	61	1
8	14	26	410	44	1	62	401
9	1	27	310	45	410	63	301
10	410	28	14	46	210	64	5
11	310	29	1	47	13	65	14
12	14	30	410	48	1	66	1
13	1	31	210	49	401	67	410
14	410	32	13	50	1	68	13
15	210	33	1	51	410	69	1
16	1	34	401	52	210	70	401
17	101	35	5	53	101	71	301
18	301	36	1	54	301		

The computer time required to find this successful plan was approximately 15 minutes on a SPARC Classic workstation. The results obtained in the study indicate that the automatic plan generation using an on-board computer with the computational power of a 80486 is quite feasible. Computers are now commercially available with substantially greater computational capability in a small size appropriate for planetary rover applications. However they have yet to be qualified for space. Based on these results the implementation of self-planning systems using inventories of action modules would appear to be feasible for planetary exploration rovers in the foreseeable future.

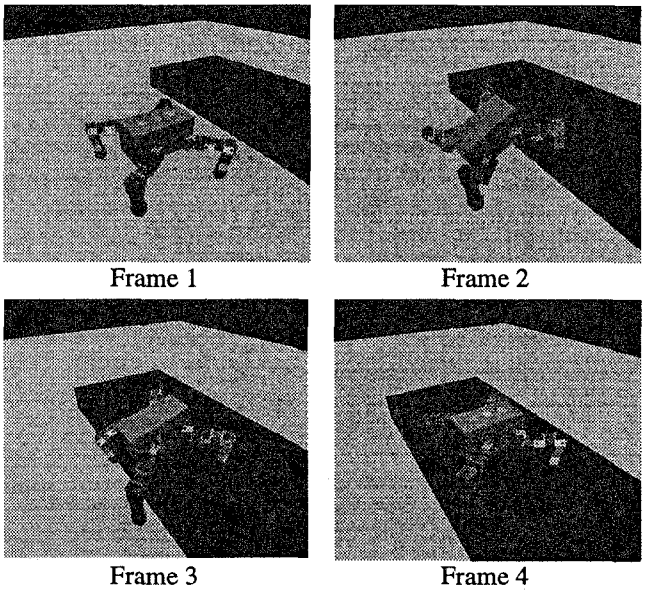

Figure 5: The Step Task

IV. WHY DOES IT WORK?

In this method the planning problem becomes a search for a script of action modules that will allow the robot to accomplish its task. The size of the search space is given by $D=N^m$, where D is the number of possible action plans, N is the number of action modules in the reduced inventory, and m is the number of action modules used in the action plan. In our example, the genetic algorithm produced an action plan that used a script of 189 action modules chosen from an inventory of 32 action modules. If we consider a plan of this length, using these action modules, there are over 10^{283} possible action plans. This is a very large search space. Also, for a difficult task such as the step climbing task, the density of good solutions in the space is not high (i.e. the number of feasible solutions with respect to the total number of solutions). With such apparent odds, the obvious question is "Why does the approach work?"

There are two contributing factors that permit our genetic algorithm search to find a feasible solution. The first is the manner that the genetic algorithm fitness function assigns a score to an action plan and the second is how genetic crossover occurs. When a given action plan is evaluated, the simulation executes the plan until a physical constraint is violated or the target is reached. The remaining action modules are not executed, and are of no consequence. The beginning portion of the plan, that which is successful, is what determines the fitness of the plan. This assigns "partial credit" to an action plan. If the plan moves the robot half way to the target then that plan receives a higher score than a plan that only moves it one quarter of the way to the target. With this approach, a plan that can move you "one step closer" to the target, will receive a higher fitness.

The second contributing factor is the method in which genetic crossover occurs. In crossover, two plans with high fitness are combined to produce two new plans in the hope that the new plans will be of improved fitness. The method to accomplish this is shown in Figure 6.

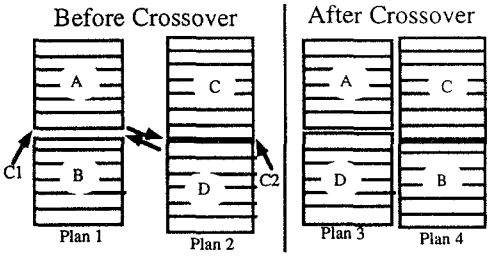

Figure 6: Genetic Crossover

Here a point (C1), called a crossover point, is chosen at a random location along the first action plan. A second crossover point (C2) is chosen in the second action plan. Then the modules that follow the crossover point on plan 1 are exchanged with the modules that follow the crossover point on plan 2. In doing this the beginning portion of each plan stay intact. Hence, the successful portion of this plan is maintained, and change occurs on the less successful portion. Since the crossover point on each plan is chosen at random, it is not ensured that crossover will occur at the exact point that the plan becomes unsuccessful. But, the effect is that you tend to build on the successful portions of the initial plans.

The combination of these two factors, fitness evaluation and crossover method, allow the planner to build on, and improve action plans that are partially successful. This suggests that the planner is not searching for an entire action plan to complete the task, but is instead building on partially successful plans by looking for ways to improve them. This can be perceived as a *short look-ahead horizon*. In general, the planner is only considering how to improve the plan through the next few "steps", that move the robot closer to the target. This effect is clear in the mathematical analysis. For example, if an inventory of 10 action modules is used and 50 actions are required to complete the task, there are 10^{50} possible action plans. However, with the method of assigning fitness and method of crossover discussed, in actuality it is possible that only 4 actions are being considered. (Discussed further in Section V.) This may mean that the search space contains only 10000 possible action plans. This represents a reduction in the size of the search space by a factor of 10^{45} and allows the problem to be solved very effectively.

The effect of the short look-ahead horizon is that the robot is less likely to make dramatic changes in its

partially successful plan. This can be called *limited backtracking*. Again, since the crossover point is chosen at random, dramatic changes can occur and the search is not locked into a particular region of the search space. However, the *short look-ahead horizon* and *limited backtracking* keep the search focused and allow it to find a solution. This greatly reduces the size of the search space and allows the search method to solve problem.

V. GUIDELINES

The short look-ahead horizon gives the planner a more reactive character. That is, it does not consider the entire problem at once, but instead tries to move "one step closer." To determine the size of this short look-ahead horizon, the robot was given the simple task of walking forward 24". Each time an improvement (increased fitness) in the plan was made the size of that improvement was recorded. The results are plotted in a histogram in Figure 7. It is seen that most improvements in the plan involved less than 5 action modules, suggesting the look-ahead horizon is on the order of 4-5 actions. However, some large improvements (>20 action modules) also occurred. These larger improvements suggest that the robot may "learn", this is discussed further in Section VI.

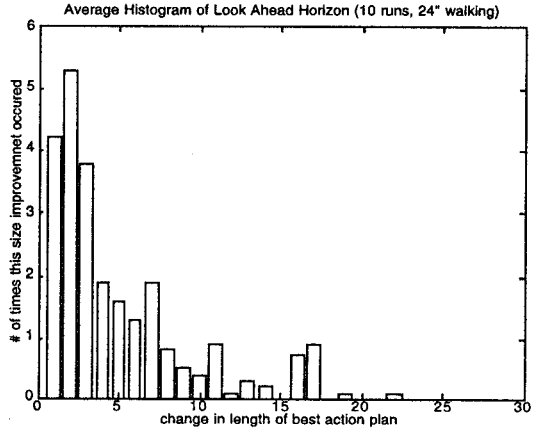

Figure 7: Histogram of Look-Ahead Horizon

This short look ahead horizon says that the robot is limited in the amount it will backtrack. For example, if the task is to travel from Boston to London, the initial plan might be to drive a car east. After arriving at cape cod this plan cannot continue. The correct solution to this problem would be to turn around, go to the airport, fly to New York (the only available connection) and then to London. However, in turning around and flying to New York, the "fitness" of the plan does not improve even though this must be done to accomplish the task..

In order for this method to effectively develop plans for exploration, the planning problem must be considered at different levels. In traveling from Boston to London, the details such as the positioning of each footstep to get from the office to the airport, or the route taken from the ticket counter to the airplane should not be initially considered. Instead higher level actions such as whether to drive a car or fly in an airplane should be considered. An action module inventory that contains higher level action modules should be used, in other words, action modules that are appropriate for the task at hand. With this high level action module inventory the larger problem can be solved using the same method presented in Section 3. Then with the higher level problem solved, the task can be broken into sub-goals and the sub-problems can be effectively solved on this lower level. As the problem is broken into smaller pieces, or sub-goals, the limited backtracking of the planning technique becomes less critical. The problem of the level of action modules that is appropriate for a given task will be explored in Section VI.

Reaching each sub-goal is in itself a sub-task. As our example problem of climbing a step is to be viewed as a sub-task in a larger exploration problem. Each sub-task is in itself a difficult planning problem. At this stage, the low level action modules and the planning methodology are very effective at producing solutions to difficult planning problems such as the one presented in section 3.

VI. DOES THE ROBOT "LEARN"?

Figure 7 suggests that large increases in the fitness of an action plan are possible. Using the method of crossover described in Section IV, it is possible to "reuse" a portion of a successful plan by adding it to another plan.

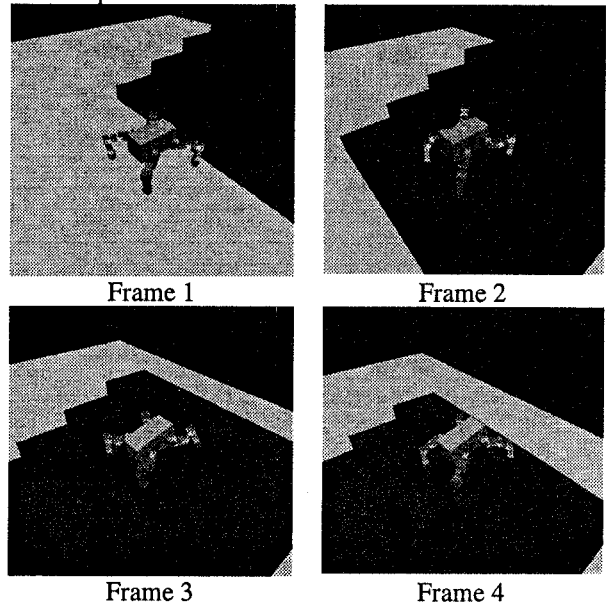

Figure 8: The Stair Climbing Task

To highlight this characteristic of the method, the robot was asked to perform a repetitive task: climbing 4 sequential 4" steps. The search was conducted and created a plan that included 514 action modules. Generations of 1000 action plans were used and 80 generations were required. The search results can be seen in Figure 8.

After the plan was developed, it was analyzed to determine if there were repeated patterns. A slight variant of the Rabin-Karp Pattern Matching Algorithm was used [12]. It was found that a pattern of 12 action modules were repeated at each of the four steps, suggesting that learning did in fact occur.

These 12 action modules were then grouped to produce a higher level action module which was added to the inventory. The task was then repeated using this augmented inventory. The results of the search are seen in Figure 9.

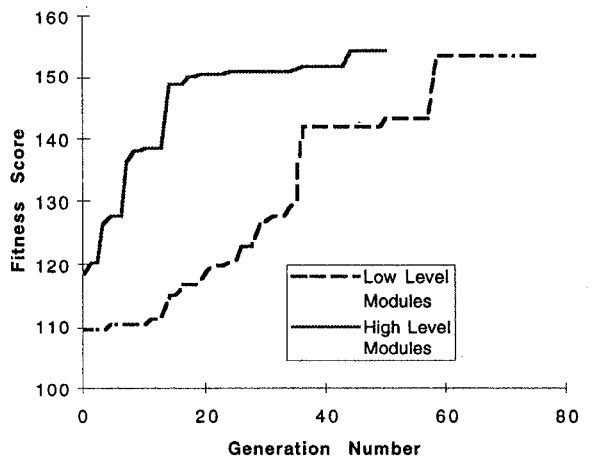

Figure 9: Genetic Algorithm Convergence

In this case, it can be seen that the inventory which the higher level action modules converged more quickly to a solution. Also, the augmented inventory produced an action plan with a higher fitness value. This suggests that higher level action modules can be very effective and should be considered for an inventory.

VII. SUMMARY AND CONCLUSIONS

Methods are required to plan the actions of planetary robotic explorers in order to accomplish mission goals. New plans must be developed as new information becomes available. Here an action plan generation procedure was discussed. This procedure develops a plan that enables a robot to accomplish the required actions of the task without violating any of the physical constraints of the problem. The proposed methodology attempts to aggressively utilize the physical capabilities of the robot, without risking mission success from the system becoming hung-up or trapped.

The procedure uses a pre-existing inventory of action modules and assembles these modules into an action plan based on the given robot configuration and task. The procedure utilizes a hierarchical selection process which includes a genetic algorithm search. The hierarchical selection process first prunes the search space to a manageable size, then uses a genetic algorithm to search this reduced search space.

This paper examined the application of this method to planetary mobile robots where issues such as computational efficiency are paramount. Some simple analysis of the approach was presented which suggests that it is more practical and effective than a first-cut mathematical analysis would suggest. This planning procedure was demonstrated on a simple mobility task. Then a discussion of the nature of the selection process, and guidelines for this process were presented.

ACKNOWLEDGMENTS

This work is supported by the NASA Jet Propulsion Laboratory.

REFERENCES

[1] Wilcox, B., et al., "Robotic vehicles for planetary exploration", *IEEE International Conference on Robotics and Automation*, , pages 175-80, May, 1992.

[2] Krotkov, E., Hoffman, R., "Terrain Mapping for a Walking Planetary Rover", IEEE Transactions on Robotics and Automation, Vol. 10., No. 6, December 1994.

[3] Matthies, L., Gat, E., Harrison, R., Wilcox, B., Volpe, R., Litwin, T., "Mars microrover navigation: performance evaluation and enhancement," Journal of Autonomous Robots, 2(4), 1995.

[4] Latombe, J.C., *Robot Motion Planning*, Kluwer Academic Publisher, 1991.

[5] Brooks, R., "A Robust Layered Control System for a mobile robot," IEEE Transactions on Robotics and Automation, Vol. 2, No. 1, 1986.

[6] Gat, E., Desai, R., Ivlev, R., Loch, J., Miller, D., "Behavior Control of Robotic Exploration of Planetary Surfaces", IEEE Transactions on Robotics and Automation, Vol. 10, No. 4, August 1994.

[7] Farritor, S., Dubowsky, S., Rutman, N., Cole, J., "A Systems-Level Modular Design Approach to Field Robotics," *IEEE International Conference on Robotics and Automation*, Minneapolis, MN, 1996. Vol. 4, pp. 2890-5.

[8] Farritor, S., Dubowsky, S., Rutman, N., "On the Design of Rapidly Deployable Field Robotic Systems," *ASME Design Engineering Technical Conference*, Irvine, CA, 1996.

[9] Tesar, D., and Butler, M. "A Generalized Modular Architecture for Robot Structures," Manufacturing Review, 1989. vol. 2, no 2.

[10] Paredis, C. and Khosla, P. "Synthesis Methodology for Task Based Reconfiguration of Modular Manipulator Systems," Proceedings of the International Symposium on Robotics Research, Hidden Valley, PA, Oct. 2-5, 1993.

[11] Goldberg, D. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addision-Wesley, Reading, MA 1989.

[12] Cormen, T., et. al., Introduction to Algorithms. MIT Press, Cambridge, MA., 1990.