N88-17279

TASK-LEVEL ROBOT PROGRAMMING: INTEGRAL PART OF EVOLUTION FROM TELEOPERATION TO AUTONOMY

James C. Reynolds
MITRE Corporation
1120 NASA Rd. 1
Houston, Texas 77058

ABSTRACT

In 1984 Congress recognized the merit of using the Space Station as a stimulus to develop a new generation of automation and robotics technology that would be efficient and flexible enough not only to meet the needs of the Space Station but also to benefit the U.S. economy. Task-level robot programming should be part of this new generation. Although it is a feasible technology for the mid-90's, it is not as well known within the NASA research and development community as it should be. This paper explains what task-level robot programming is and how it differs from the usual interpretation of "task planning" for robotics. Most importantly, it is argued that the physical and mathematical basis of task-level robot programming provides inherently greater reliability than efforts to apply better known concepts from Artificial Intelligence (AI) to autonomous robotics. Finally, an architecture is presented that allows the integration of task-level robot programming within an evolutionary, redundant, and multi-modal framework that spans teleoperation to autonomy.

INTRODUCTION

In 1984 Congress recognized the merit of using the Space Station as a stimulus to develop a new generation of automation and robotics technology that would be efficient and flexible enough not only to meet the needs of the Space Station but also to benefit the U.S. economy (NASA, 1985). The Congressional desire for technology transfer was at least partially motivated by the need to boost American labor productivity so that U.S. manufacturing would be more competitive with manufacturing in other nations.

A task-level robot programming system, which could be used with any robotic manipulator system on the Space Station, is an outstanding candidate for this new generation of technology. Such a system requires a complete world model of the workspace and task-level commands that consist of the identification of relevant objects and their desired relationships. An example would be, MOVE OBJECT ORU24 AGAINST FACE3 OF TRUSS66. These commands would then automatically be translated to the low-level motion and sensing operations required to reliably and safely achieve them. Such a system would prevent the tedious and time-consuming coding that flexible robot control normally demands.

Ten years of research in this area at Stanford, MIT, Carnegie-Mellon, IBM, and other robotic research centers has placed the development of a practical task-level robot programming system on the technological agenda. A unified conceptual framework has been developed and applied to the component problems of motion planning with obstacle avoidance, grasp planning for reachability and stability, and fine motion planning using compliance. Recently, an integrated system that implements much of the results of this work has been built at MIT. For reasons discussed below, NASA could be the decisive force in pushing this research out of the laboratory for the benefit of the Space Station Program and U.S. industry.

Research and development of autonomous robotics should build on the successes of this work. Unfortunately, alternate techniques borrowed from AI have often been applied to the problem of generating robot plans. Some of these techniques are knowledge-based and heuristic and are therefore inappropriate for robotics, especially in applications where reliability and safety are paramount. In addition, domain-independent planning is often applied to robotics, but this no longer can be viewed as a viable approach. Theorems from a recent important thesis suggest that efficient domain-independent planning with expressive power for real-world robotics is impossible.

A task-level robot programming system would support the evolutionary approach to autonomous robotics that must be taken on the Space Station. Components of a system could be used as a plan checker for robot-level programs written by ground-based personnel. As confidence in the system increased, it could be used as a plan generator of robot-level instructions that would then be simulated and modified by ground-based programmers with the advice of Station-based astronauts. Finally, when the system is judged mature, its output of instructions could be fed directly into a robot controller interactively monitored by Station personnel. This paper presents an architecture for enabling the graceful phasing in of this technology to the Space Station Program.

DESCRIPTION OF TASK-LEVEL ROBOT PROGRAMMING

The best description of a task-level robot programming system together with a discussion of alternate implementation issues is in (Lozano-Peréz and Brooks, 1985). That paper describes a framework called TWAIN for the development of such systems. The input to TWAIN would be a complete model of the robot and its environment together with a complete specification of the tasks to be accomplished.

For a practical system the model would not only include a geometrical description of objects, but also any other features of the environment that impose constraints on the motion of the robot. Mass, inertia, the coefficient of friction, restrictions on movement caused by linkages of objects, including the manipulator, and, most importantly, tolerances on the objects and bounds on capabilities such as accuracy, range, and force of both the sensors and manipulator would all be part of a task-level robot programming system's world model. Without the latter information it is impossible to plan motions in the face of the uncertainty that is the key problem of flexible robotics. The recent heightened awareness of the need for design knowledge capture during the development of the Space Station has made it possible that a model like this could be built.

The command input to the system would consist of robot-independent operations specifying the desired spatial relationships of relevant objects. A simple block-stacking example could be commanded as follows:

```
PLACE OBJECT-A AGAINST TABLE
PLACE OBJECT-B SO THAT
    FACE-1 OF B IS AGAINST FACE-2
    OF OBJECT-A
    AND
    FACE-2 OF B IS COPLANAR WITH FACE-3
    OF OBJECT-A
    AND
    FACE-3 OF B IS COPLANAR WITH FACE-1
    OF OBJECT-A
```

The importance of the commands being robot-independent is that the user does not have to specify grasp positions, complicated obstacle-avoiding paths, or terminating conditions based on dynamic and geometrical constraints. This enormously simplifies the practice of robot programming.

The commands to the task planning system advocated here are intermediate between low-level controller instructions and the input to traditional AI planners. It is important to recognize that this is not what is usually meant by "task planning" for robots. Since most task planners for robotics are based on the long chain of AI planners going back to STRIPS and ABSTRIPS, which were used to control the famous SRI robot, SHAKEY, their use of the word "task" is for a higher level of abstraction. A space-oriented example would be REMOVE ORU-24 FROM CHAMBER-3. Within the TWAIN framework this "task" might require several commands to specify.

The focus of much research and development for autonomous robotics has been on this higher abstraction task planning. The problem of focusing on this level rather than the intermediate level that TWAIN addresses is that the really hard problems of robotics are avoided. These higher level planners are capable of generating sequences of actions, but are not capable of planning under uncertainty or where there are side effects of the consequences of the planned actions. This will be discussed in greater detail in a later section of this paper.

THE IMPORTANCE OF TASK-LEVEL ROBOT PROGRAMMING

The development of a task-level robot programming system, which could be used with any robotic manipulator system on the Space Station, e.g., the Flight Telerobotic Servicer (FTS), would increase the productivity of the crew and ground personnel for Space Station operations and be a critical technology to transfer to U.S. industry.

That this capability would be crucial for the efficiency and usefulness of the Space Station itself is indicated by an analysis of the baseline configuration of the Space Station in terms of the crew hours available for maintenance and housekeeping (Reynolds, 1986). It is now widely recognized that there is, in fact, a contradiction between the hours estimated for those activities and the hours required for customer services. The logical resolution of this contradiction is to use A&R to increase crew productivity. A FTS that requires one or more crew members to continuously control it through teleoperation would possibly increase functionality of the Space Station and reduce extra-vehicular activity. It is not likely, however, to increase crew productivity enough to eliminate the contradiction between maintenance requirements and customer needs. The FTS must be programmable, and a task-level programming system would be the most productive way of programming the FTS.

Evidence that the development of a task-level programming system would indeed promise major advantages for industry can be found in the 1982 study by the Society of Manufacturing Engineers and the University of Michigan (Smith, 1982). That report recognized that robot control was the most important technological factor needed for the rapid utilization of robotics by U.S. industry, ranking ahead of computer vision, tactile sensing, and mechanical manipulation. The report was, however, overly optimistic about achieving this technology; a practical implementation of a task-level robot programming system by NASA for the Space Station could make the predictions of that report a reality.

Presently, robots in industry are either programmed by guiding or programmed in a robot-level language which includes instructions for accessing sensors and controlling the motions of the robot. Each of these methods has key shortcomings in the context of either industrial or space applications.

Programming by guiding involves the operator of the robot moving it through a sequence of positions needed to accomplish a specific task. The motions are recorded on tape and then are played back to execute this sequence repeatedly. This is one step above hardwired automation in that the robot can be used for more than one sequence of positions. However, it is equivalent to straight line programming in that no branching is allowed. For the robot to accomplish a task with this type of programming, the task must be characterized by little uncertainty in the geometry of the environment. It is impossible, with this type of programming, to use sensor feedback to correct positional errors or to choose alternate paths in the face of unexpected conditions.

Incorporating programming by guiding capability in the Space Station Program would accomplish little in benefiting U.S. industry or improving crew productivity on the Space Station. It is a mature technology which would not be advanced by its inclusion in Space Station A&R. Further, it is most appropriate for highly repetitive tasks, where the capability of deviating from a specified path is not of great importance. Even for maintenance and assembly tasks that require no great intricacy, there must be fine control with sensory feedback of any manipulator external to the Space Station, because of the sensitivity of the Station to the inertial effects of manipulator motion.

In addition, it is unlikely that many of the maintenance tasks will be done over and over for long periods of time (Holt, 1986). Typically, there will be several different maintenance tasks to be accomplished over a short time period like one day, and then it may be a much longer period before any of those tasks are done again. This situation is similar to what exists in small batch manufacturing, where robotics has not yet been applied because of the limitations of both programming by guiding and robot-level programming.

Robot-level programming is a commercially available alternative to programming by guiding. It represents an advance in that branching on sensor input is allowed, thus allowing for more robust behavior in the face of uncertainty and error. VAL-II for the Unimation series of robots and AML for certain IBM robots are two of the best known examples of these languages. These languages, although commercially available, need improvement. Some of the problems of these languages were outlined in (Lozano-Perez, 1983b). These include the following:

1) Robots and CAD/CAM systems are not able to communicate. Information in CAD systems should speed the computations necessary for motion.

2) Robot programming is highly device dependent, describing operations in terms of the motion of individual arms rather than tasks. The addition of new objects in the environment, including new robotic devices such as additional manipulators, generally requires the rewriting of the entire program.

3) Obstacle avoidance is difficult to specify, resulting in long complex programs consisting of moving and then checking. The programmer must anticipate all situations.

The key shortcoming of robot programming today is that, like any other type of software development, it is expensive and time-consuming. In industry very few of the explicit robot programming languages that are sold with robotic systems are ever used; it is easier to use guiding for programming (Rossol, 1984). For operations that require sensor data for reliability, the necessary programs are extremely complex. (Carlisle, 1985) mentions a VAL-II program in an assembly plant that was over one hundred pages long.

If the functions of the robot consist of manufacturing or assembling a few parts many times, the expense of programming can be justified. The problem is that a large part of industry manufactures and assembles many different parts in relatively small quantities. This has prevented the introduction of robots and advanced automation into many industries. Analogously, if the tasks to be performed on the Space Station are indeed diverse and non-repetitive, then the time required to program the robotic device might be greater than the time saved by the astronaut not being a slave to the robotic device.

Task-level programming is essentially a means to speed up the software development process that is ever more often the bottleneck in the engineering of large-scale systems. It is necessary for the next leap in the use of robots in industry. Without this capability the application of autonomous robotics to Space Station operations will be impossible. Further, research in the AI and robotics laboratories has reached a maturity that demands a new phase of actual development of these systems. Unfortunately, it is also clear that the impetus to utilize the fruits of over ten years of research will not come from industry. Thus, NASA can play a crucial role, which is exactly what Congress intended by its A&R mandate.

FEASIBILITY OF TASK-LEVEL ROBOT PROGRAMMING

The feasibility of a task-level robot programming system for the Space Station Program depends on over ten years of increasingly fruitful research.

The first big advance in the synthesis of robot programs from task-level specifications was the system described in (Taylor, 1976). It introduced the use of parameterized procedure skeletons, which were generalized robot programs including motions, error tests, and necessary computations but without the parameters bound to any numeric values. Depending on the geometry of the model and the tolerances and uncertainty bounds, a skeleton was chosen, and the remaining parameters were determined for grasp and approach positions. Taylor utilized linear programming methods to compute legal ranges for the parameters.

This work was significantly extended in (Brooks, 1982). Rather than using numerical methods to propagate constraints caused by position uncertainty, control uncertainty, and model uncertainty (tolerances), he used formal logic techniques to reason both forward to check error bounds and backward to restrict the range of plan variables and introduce sensing operations.

Fundamental to the line of research described here is the concept of configuration space (Lozano-Perez, 1983a). The configuration of an object is the set of parameters necessary to completely specify the position of all points of the object. The configuration space of an object is the space of all possible configurations of the object. Obstacle avoidance can be accomplished by transforming a robot into a point and, for each element of a discretized set of possible orientations of the robot, growing the obstacles by the shape and size of the robot wherever contact with the obstacles is feasible.

Rapid progress in the development of algorithms for gross motion planning with obstacle avoidance and fine motion strategies was made during the next few years based on these concepts. (Brooks, 1984) is a good example of gross motion planning work and provides further references. An algorithmic approach to the automatic synthesis of fine motion strategies (guarded moves with compliance) was described in (Lozano-Perez, Mason, and Taylor, 1984); since then this line of research has matured further (Erdmann, 1986).

Automatically synthesized fine motion can be achieved by extending the configuration space concept to the notion of recursively determined pre-images. These are the set of all starting configurations which can reach a goal configuration within the constraints of control and model uncertainty but allowing for compliant motion. The pre-image must also exclude configurations and velocities which would lead to sticking. If the set does not include the actual starting configuration then the algorithm is applied recursively to determine the pre-image of a configuration within the first pre-image. Thus multi-step plans can be generated.

TWAIN was proposed in (Lozano-Perez and Brooks, 1985) and embodies all these ideas. Each single task specification is turned by the executive planning module of the system into a sequence of gross motion, grasping, gross motion, fine motion (either synthesized or pre-determined), and ungrasping. There are separate modules for each of these three types of planning. There is a skeleton library and skeleton matcher to choose unrefined plans, and there is a constraint propagator which uses the principle of least commitment and symbolic propagation to instantiate skeleton parameters and add sensing operations. Finally, because success is not guaranteed, dependency-directed backtracking is employed to reduce search.

TWAIN has not been implemented. However, since its proposal, component problems for a task-level programming system based on the work of Brooks, Lozano-Perez, Mason, and Taylor have been tackled with increasing success. There has been an explosion of work on motion planning with obstacle avoidance. The automatic generation of grasping (Nguyen, 1987) and regrasping (Tournassoud et al., 1987) strategies has also been tackled. Finally, a systematic attack on the problem of error detection and recovery (as opposed to the ad hoc "hacks" of traditional AI planners) based on the algorithms and concepts developed in the above referenced work was informally presented in (Donald, 1986).

Some of these more recent ideas and some of the ideas of the original TWAIN proposal have been implemented in an integrated robot system by Lozano-Perez and his colleagues at MIT (Lozano-Perez et al., 1987). The Handey system is capable of locating a part that has been accurately modeled in an unstructured environment, choosing a grasp on the object, planning a collision free path to the object, grasping the object, planning a collision free path to the specified destination, and placing the object in the commanded position. The system does not incorporate the ideas of constraint propagation, but it is under development with an error detection and recovery capability being one of the planned additions.

This large and growing body of work tackling the hard problems of robotics utilizes sophisticated mathematics and is thoroughly grounded in geometry and mechanics. Its algorithms can be analyzed for correctness and completeness, and hence their robustness can be verified. It does not use expert systems or the simplistic techniques of traditional AI planners. It is not surprising that this approach has led to an actual robotic system that is capable of more impressive "intelligence" than any system depending on those techniques. It is this work that NASA should be pushing and extending.

AI AND AUTONOMOUS ROBOTICS

During the last few years many of the concepts and techniques of AI have become commonplace in engineering and data processing publications. It is of interest then to discuss the relevance of AI to robotics. The first concept and associated techniques from AI to be widely applied was "expert systems." This phrase has now given way to "knowledge-based systems," perhaps in recognition that many useful applications could be built without needing expert competence. In either case these systems contain a knowledge base of facts and rules and an inference engine to reason from the knowledge base and the data input to the system.

For robots to be autonomous it is essential that the algorithms controlling their actions are correct (the robot does what is intended by the human) and reasonably complete (if it is possible to accomplish a specified task, a robot-level program will be generated to do it). This is extremely important for robotic applications that must be robust in the face of uncertainty. In a hazardous environment like space the need for correctness and completeness is even greater. The use of expert systems, which are inherently heuristic, is an ill-advised application of a useful technology. It is simply not good enough for a motion planner to plan an obstacle avoiding path ninety per cent of the time.

Further, even without performance and reliability considerations for robotic applications, the use of expert systems in robotics is inappropriate. This is because there are few heuristic rules that can be generalized to guide motion planning; instead, it appears that small changes in the environment lead to significantly different motion strategies. For example, in the simple peg-in-hole task the geometrically trivial change of adding a chamfer to the hole would result in a radically different motion strategy.

Domain-independent planning is an area of AI that seems to be naturally applicable to robotics. Indeed, the famous early planners like STRIPS and HACKER were often applied to planning the actions of a "robot" in a "blocks world." ABSTRIPS (Sacerdoti, 1974) actually controlled a physical robot, SHAKEY. Unfortunately, an investigation of the SHAKEY project shows that the environment was too carefully engineered to be realistic and that errors in the model were handled by expensive re-planning. In addition, all of the well known planners utilized impoverished semantics; their worlds could be described in a few sentences. Representing the complex, largely quantitative model required for task-level programming would be impossible using these planners.

There are two reasons that traditional AI planners are so inappropriate for robotics. First, robot planning requires geometric representations and those are largely numeric, which has not been the emphasis of AI in general, including its planners. Second, robot planning must handle uncertainty and error, and AI in general has not been able to solve this problem. AI planners do not even pretend to try; if they attempt to handle it at all, it is by ignoring uncertainty while planning and then trying to recover during execution.

Even more devastating to those who wish to apply the techniques of AI planning to robotics is an important thesis (Chapman, 1985). That work shows that all well known AI planners work in essentially the same way. Further, their action representations do not allow for indirect or input dependent effects or for uncertain execution. Finally, extending them with more expressive action representations while keeping them computationally tractable is probably impossible and would also invalidate the proofs of correctness and completeness of these planners within their limited, artificial world.

AN EVOLUTIONARY IMPLEMENTATION OF TASK-LEVEL ROBOT PROGRAMMING FOR THE SPACE STATION

It is important that the development of autonomous robotics for space applications proceed in an evolutionary manner. In addition, any robotic system on the Space Station must be capable of allowing human control at any point during its operation. A task-level programming system for the Space Station could be implemented within these requirements. (See Figure 1.)

The first step would require the construction of a teleoperated control system augmented with the advanced ideas of computer-assisted human interaction described in (Conway et al., 1987). On top of that the components of TWAIN that are necessary for plan checking would be built. This would allow the astronaut to input robot-level programs written by ground personnel and determine their correctness. If no bugs were detected, there would be feedback provided by simulation before allowing execution. If bugs were detected, the plan checker would act as a smart compiler in suggesting corrections. Finally, as confidence in the technology reached an acceptable level, the full-scale planner could be built which would allow the astronaut to input task-level plans, check their simulated effects, and monitor execution as closely as desired.

CONCLUSION

Task-level robot programming is an important technology that promises great benefits both on the ground and in space. Considerable progress has been made toward realizing a system that could automate the flexible control of robots. This progress has been characterized by a solid grounding in geometry and mechanics, which makes it verifiably robust and a natural choice for space applications. It is feasible to implement this technology in an evolutionary way for the Space Station Program. NASA should take steps to build on the successes of the research described in this paper in order to develop autonomous robotics.

REFERENCES

Brooks, Rodney, "Symbolic Error Analysis and Robot Planning," INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH, Cambridge, Ma., Vol. 1, No. 4, December, 1982, pp. 29-68.

Brooks, Rodney, "Planning Collision Free Motions for Pick and Place Operations," ROBOTICS RESEARCH: THE FIRST INTERNATIONAL SYMPOSIUM, M. Brady and R. Paul, Eds., MIT Press, Cambridge, Ma., 1984, pp. 5-37.

Carlisle, Brian, "Key Issues of Robotics Research," ROBOTIC RESEARCH: THE SECOND INTERNATIONAL SYMPOSIUM, H. Hanafusa and H. Inoue, Eds., MIT Press, Cambridge, Ma., 1985, pp. 501-503.

Chapman, David, "Planning for Conjunctive Goals," Technical Report 802, MIT Artificial Intelligence Laboratory, Cambridge, Ma., November, 1985.

Conway, Lynn, Volz, Richard, and Walker, Michael, "New Concepts in Tele-Autonomous Systems," SECOND AIAA/NASA/USAF SYMPOSIUM ON AUTOMATION, ROBOTICS AND ADVANCED COMPUTING FOR THE NATIONAL SPACE PROGRAM, Paper IAA-87-1686, Arlington, Va., March 9-11, 1987.

Donald, Bruce, "Robot Motion Planning with Uncertainty in the Geometric Models of the Robot and Environment: A Formal Framework for Error Detection and Recovery," 1986 IEEE CONFERENCE ON ROBOTICS AND AUTOMATION, San Francisco, Ca., April 7-10, 1986.

Erdmann, Michael, "Using Backprojections for Fine Motion Planning with Uncertainty," THE INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH, Cambridge, Ma., Vol. 5, No. 1, Spring 1986, pp. 19-45.

Holt, Alan, "Mobile Robotics and Artificial Intelligence for Large Payload Assembly and Maintenance," Paper 6-0607, ROBEX '86, Houston, Tx., June, 1986.

Lozano-Perez, Tomas, "Spatial Planning: A Configuration Space Approach," IEEE TRANSACTIONS ON COMPUTERS, Vol. C-32, No. 2, February, 1983a, pp. 108-120.

Lozano-Perez, Tomas, "Robot Programming," Proceedings of the IEEE, Vol. 71, No. 7, July, 1983b.

Lozano-Perez, Tomas, Mason, Matthew, and Taylor, Russell, "Automatic Synthesis of Fine-Motion Strategies for Robots," THE INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH, Cambridge, Ma., Vol. 3, No. 1, 1984.

Lozano-Perez, Tomas and Brooks, Rodney, "An Approach to Automatic Robot Programming," Artificial Intelligence Laboratory, Massachusetts Institute of Technology, A. I. Memo No. 842, April, 1985.

Lozano-Perez, Tomas, Jones, Joseph, Mazer, Emmanuel, O'Donnell, Patrick, and Grimson, Eric, "Handey: A Robot System that Recognizes, Plans, and Manipulates," 1987 IEEE CONFERENCE ON ROBOTICS AND AUTOMATION, Raleigh N.C., March 31 to April 3, 1987.

NASA, "Advancing Automation and Robotics Technology for the Space Station and for the U.S. Economy, Progress Report 1," NASA Technical Memorandum 87772, Houston, Tx., October, 1985.

Nguyen, Van-Duc, "Constructing Stable Grasps in 3D," 1987 IEEE CONFERENCE ON ROBOTICS AND AUTOMATION, Raleigh N.C., March 31 to April 3, 1987.

Reynolds, James, "Analysis of Crew Time on the Space Station," unpublished MITRE Corp. report to the Artificial Intelligence and Information Sciences Office at Johnson Space Center of NASA, Houston, Tx., March, 1986.

Rossol, Lothar, "Technological Barriers in Robotics: A Perspective from Industry," ROBOTIC RESEARCH: THE FIRST INTERNATIONAL SYMPOSIUM, M. Brady and R. Paul, Eds., MIT Press, Cambridge, Ma., 1984, pp. 963-972.

Sacerdoti, Earl, "Planning in a Hierarchy of Abstraction Spaces," ARTIFICIAL INTELLIGENCE, Vol. 5, No. 2, 1974, pp. 115-135.

Smith, Donald and Wilson, Richard, "Industrial Robots: A Delphi Forecast of Markets and Technology," Society of Manufacturing Engineers and The University of Michigan, 1982.

Taylor, Russell, "The Synthesis of Manipulator Control Programs from Task-level Specifications," AIM-282, Stanford Artificial Intelligence Laboratory, Palo Alto, Ca., July, 1976.

Tournassoud, Pierre, Lozano-Perez, Tomas, and Mazer, Emmanuel, "Regrasping," 1987 IEEE CONFERENCE ON ROBOTICS AND AUTOMATION, Raleigh N.C., March 31 to April 3, 1987.

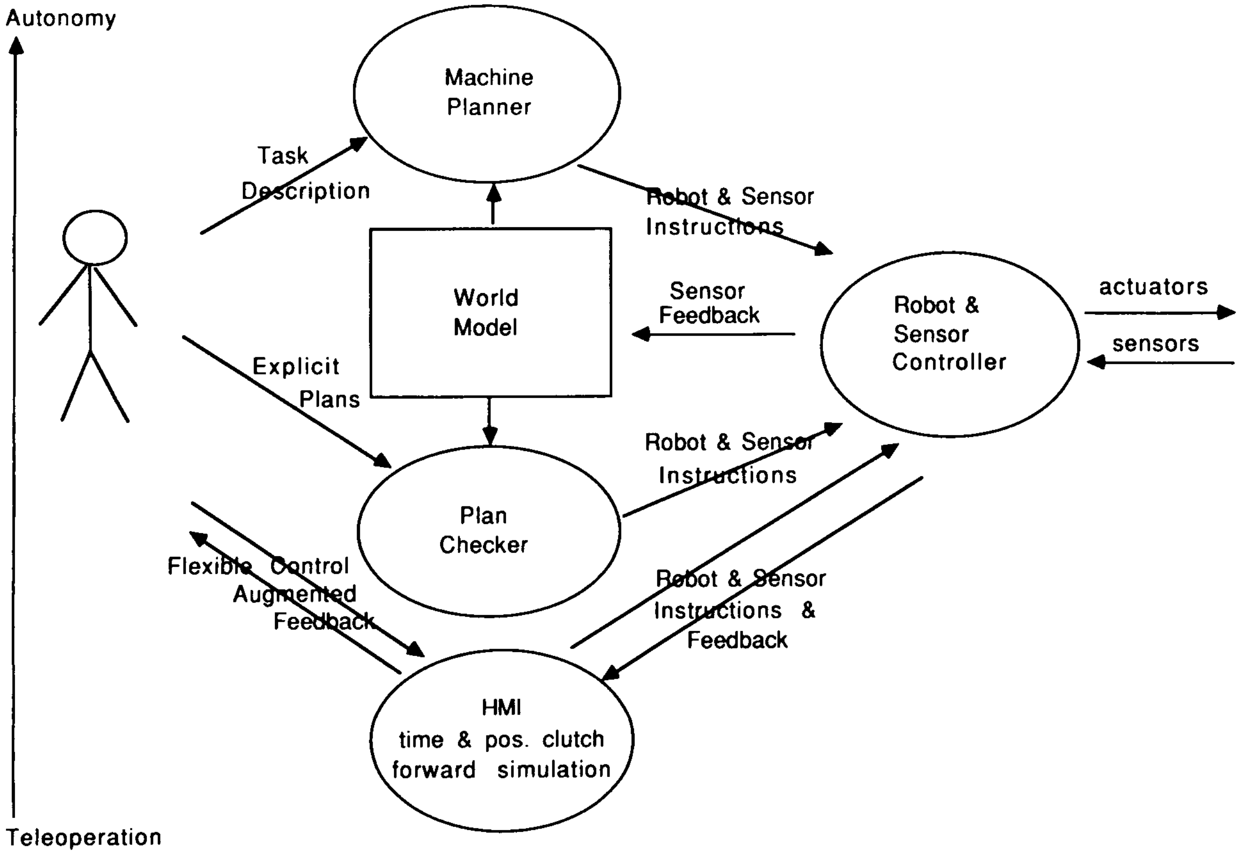

FIGURE 1 - EVOLUTIONARY ARCHITECTURE WITH TASK-LEVEL PROGRAMMING