An Integrated Prototyping Environment For Programmable Automation

(Published SPIE/OE 92 International Symposium on Intelligent Robots In Space)

F. daCosta
ACG, Inc.
San Jose, CA

V. Hwang, PhD
MITRE Corp.
Mclean, VA

P. Khosla, PhD
Carnegie Mellon Univ.
Pittsburgh, PA

R. Lumia, PhD
N.I.S.T
Gaithersburg, Md

Abstract

We are implementing a rapid prototyping environment for robotic systems, based on tenets of modularity, reconfigurability and extendibility that may help build robot systems "faster, better and cheaper". Given a task specification, (e.g. repair brake assembly), the user browses through a library of building blocks that include both hardware and software components. Software advisors or critics recommend how blocks may be "snapped" together to speedily construct alternative ways to satisfy task requirements. Mechanisms to allow "swapping" competing modules for comparative test and evaluation studies are also included in the prototyping environment. After some iterations, a stable configuration or "wiring diagram" emerges. This customized version of the general prototyping environment still contains all the hooks needed to incorporate future improvements in component technologies and to obviate unplanned obsolescence. The prototyping environment so described is relevant for both interactive robot programming (telerobotics) and iterative robot system development (prototyping).

Introduction

Escalating costs and long lead times associated with building robotics and automation systems favor automation of the process of designing automation systems itself. Lack of a higher level of automation in the product life cycle support of automation systems has resulted in increased cost of the end product, lower quality, and increased lead time. This is especially true for automated manufacture, re manufacture and repair of small quantity or one-of-a-kind items; lack of economies of scale prohibit the development of custom systems for them. Unfortunately, the bulk of components comprising today's weapon systems (and much of civil systems) fall in this category. Despite diverse application domains[1, 2, 3, 4, 5], there is a marked similarity in their needs to program robots "faster, better and cheaper". With shrinking operating budgets, there is strong incentive to automate the process of maintenance and repair, currently performed manually. The high cost of programming robots must be addressed by developing techniques to automatically program them. The Dod Critical Technologies Plan has specified the near term technology objective: "Planning and control of robotic assembly from CAD models" by 1996[6]. Similar interests, related to variable autonomy, have been expressed by NASA[4].

We are implementing a rapid prototyping environment for the design of robotic systems that includes a library of commercially available "lego blocks" consisting of hardware modules (such as robots, tools and sensors), complementary software modules (such as sensing strategies, and robot motion control) and a family of software advisors or critics designed to reduce the costs of programming robots. These include (CAD model driven) assemblability sequencing, robot motion planners and grasp strategy selection.

In addition to reducing programming costs, the prototyping environment will allow users to quickly generate and evaluate alternate approaches to design of an automation system. Thus, program managers can view how the robot will put together the parts in alternative assembly sequences and, consequently, be able to make a more informed decision. This work is relevant to small batch factory floor automation. It is also relevant to field robotics, where critics, as part of a telerobotic interface, could support cooperative robot-human interaction in harsh or remote areas.

Related Work

The main advantage for robots is their ability to be (re)programmed. There are three levels at which to do this: Joint level, Manipulator level and Task level. Joint level programs describe motion commands required by robot joint motors. It is the lowest and most flexible level at which robots can be programmed. Writing programs at this level is a painfully tedious process requiring specialized knowledge. Further, the software developed is both device and application specific. Manipulator level programs build upon joint level programs to enable users to describe actions required of the robot to perform a task as opposed to individual joint motions. A robot program is written as a sequence of robot manipulator language primitives such as move and open-gripper commands. Several manipulator level languages exist; they suffer from the following limitations: 1) the programming languages are primitive, many do not support standard programming constructs such as recursion or abstract data types; 2) there exist a plethora of robot programming languages with no common features across languages, making robot programs non-transportable and 3) robot programmers are required to have an intimate knowledge of robots, sensors and programming languages.

Off Line Programming Environments[21, 22] are a variant of manipulator level programming. They are graphic simulation platforms that provide a subset of primitives commonly used by various robot vendors. The robot programmer has to learn only the simulation language and not any of the robot programming languages. However, the current state of art in off-line systems suffers from two serious drawbacks; first, it is at best a robot motion simulator and second, does not

address the issues of sensor based robot behavior, fundamental to any robotics task involving dynamic real world situations.

Task Level Environments[9, 16, 17, 18, 19] are at yet a higher level of abstraction; their focus is on describing task goals (e.g. "assemble C5A brake disk") which is then automatically translated by the system to manipulator level commands. Task level programming is still an active research issue because 1) translation mechanisms that automatically convert high level tasks specifications to low level robot code, do not exist for the general case; 2) the complexity of real world scenarios is not always easily representable; 3) perception techniques are not always available to make "correct" situation assessments. Nevertheless, tools such as geometric reasoning for robot motion planning, and grasp strategy selection can significantly reduce the burden on the robot programmer, if they are incorporated in a prototyping environment that supports model based reasoning but also human interaction during the design process.

Design Philosophy

Tenets guiding the design of a integrated rapid prototyping environment should include:

- **Task directed process design**. Manufacturers' specifications for robots contain data such as payload or work envelope that are not directly related to the task specifications (e.g. assemble). We must transform the semantic intent of assemble to the physics of the task: a sequence of motions and force applications. We are then able to map semantic intents to robot payload and work envelope requirements. Further, the language of forces and motions is also a language that robots Understand" in terms of real time controls issues. We then have a common thread that links the selection of lower level control strategies and hardware selections, to the high level task specification. We later describe a "capabilities browser" that is a first step in this direction.

- **Systems engineering focus**. Robotics is a systems science. We must shift focus towards rapidly prototyping the entire system, in order to understand the interactions prevalent, as opposed to fine tuning individual component technologies before putting them all together. We would then be studying system performance in its entirety and have a better idea what the evolution of supporting component technologies for specific domains should be. A systems engineering focus makes a strong case for prototyping environments for robots. Our integrated prototyping environment is designed to force a consensus between the semantic intent and more implementation driven control theoretic techniques, both of which are needed to "make it all work together".

- **Re configurable modular design.** We cannot realistically claim that we know what robot architecture are best for an application domain, much less commit to specific component technologies without examining both technical and economic tradeoffs. For example, we may need to evaluate different robot control architectures [10, 8]. This requires reconfigurability: our prototyping environment should allow us to "rewire" the components of the system to determine the system behavior rapidly and cost effectively. By the same token, we must be able to easily "swap" modules for quick A-B comparisons, where modules could be robots, tools, sensors or software algorithms. For example, we have evaluated the relative performance of a family of heuristic path planning algorithms that produced collision free paths according to differing cost functions (e.g. minimum distance or safest path) within the same off-line programming environment.

We are supporting the need for modular design by providing "adapters" to mediate between incompatible off-the-shelf components. For example, we have written translators from an evolving robot language superset to robot vendors' proprietary languages to so that users need be exposed to only one robot programming environment and yet use commercially available robots. But this is a short term fix. In the long term, open systems must be encouraged.

- **Open Systems and extendibility**. Commercial robot programming environments are typically "closed systems": their fundamental capabilities cannot be extended or modified by the user. System reconfigurability is best supported by an open systems approach whereby users or a supervisory program can rapidly modify, re configure or extend the core library of functionality's, keeping in step with changing requirements and evolving designs. The concept of evolutionary open systems is critical to protecting the customer's investment and avoiding obsolescence or building a "monument to technology".

We are interested in joining rapid prototyping consortia that include industry, universities and federal research laboratories. Such consortia will help establish guidelines that will foster a spirit of cooperation within the robotics community. The resulting synergy could well provide the fillip needed to revitalize an unprofitable robot industry.

- **Product life cycle support**. Initial cost savings due to automation are often quickly offset by costs due to compromised reliability and maintainability of the robotic system and resulting low quality of the end product. One reason for this is that automation systems are still built to be application and domain specific, with little thought paid to how the systems can be modified for future applications. The result is an inordinately high cost and lead time for even minor variants to an existing automation system. In the current economic climate this is a luxury the customer cannot afford.

We can reduce the costs and lead time associated with starting from scratch for every new application domain (because system components were designed to be application specific) by providing tool kits for tasks routinely performed during robot systems design. This includes a library of software and hardware modules commonly used and software critics that guide the user in making judicious selections of robots, sensors, tools and control techniques, in addition to automating portions of the program generation process.

A Prototyping Methodology

We begin by examining how we may prototype a robot system. Later, we describe components of a prototyping environment that will automate this process for the novice.

- *Capability Driven Selection*. We link the description of a task, to what those descriptions "mean" in terms the physics of the activities (e.g. forces, motions etc.). Those requirements force certain commitments of an agent that could perform the tasks. We begin by tentatively selecting an actuation capability (e.g. a robot and tool ensemble) that supports the payload, work envelope and accuracy requirements of the task requirements. However, our choices are very preliminary: we will not try to marry a tool too heavy for a robot payload, or select a robot that has less than the requisite degrees of freedom, but beyond this we are not more detailed in their circumspection. Other issues, such as sensing strategy selection, must be addressed first.

- *Sensing Strategy Selection.* The sensor selection is driven by both the task specification and the actuation capability under consideration. Based on the physical effects of desired objectives we must infer what sensor readings should be; we can then suggest appropriate error detection and recovery strategies some available in our library of strategy modules. Some difficult choices may need to be made here. For example, if the robot picks up a tool, one implementation of an error detection procedure available could be to check if the force sensor registers a change. Alternatively, we may choose to use a arm mounted vision system to locate the tool. The decision as to which is a better sensing strategy is driven by the accuracy of the robot and tool, and sensors accessible by the robot. But is also driven by computing resources needed and real time robot control considerations.

- *Real Time Control Parameters Selection.* Having made a set of commitments to actuation and its (partially) dependent sensing strategies, we must examine if the choices made are agreeable to the real time robot operating control software. Based on the sampling frequency or computing requirements of the hardware and support software, we must decide between alternative control strategies. We may find that we do not have the computing power required for an error detection routine using vision and opt to use the force sensor instead. By the same token, the cost or unavailability of advanced sensor based servo control, such as hybrid motion control, may force us to redesign the workcell or task. Through an iterative process, we will eventually form a consensus between the semantic intent of the task and the real time control level implementation details for time critical tasks involving sensor based behaviors. These include fine motion strategies, curve following and real time obstacle avoidance.

- *Simulation and Spatial Reasoning.* By jumping from selection of hardware to making commitments at the real time control level, we skipped a few steps. We did this intentionally, because we wished to ensure that our resource selections did not violate critical real time control requirements before our skeletal design was fleshed out any further. We are now in a position to fill in some of the less time critical components. Off line programming tools, applied to CAD models of the workcell, could enable users to examine alternative sequences for robot motion or for grasping a part. Script generation for fine motion strategies[17] also takes place here.

The reader will have noted that the process outlined above is not strictly hierarchical. If a choice of a sensor or a fine motion strategy is made, it must be "checked" to ensure that it does not violate real time constraints. If it does, we need to backtrack, make another selection and try again. The concepts of concurrent engineering involving interactions between specialized reasoners and for forming consensus are relevant. But, since our intended system is interactive and not automatic, we are not overly concerned with underlying research issues. However, the need to backtrack, play out different scenarios and maintain state must be supported by the components of the prototyping environment proposed.

Components Of A Prototyping Environment

We described an approach to fleshing out a design concept to satisfy a high level task specification. Software components that could facilitate the process of transforming high level task descriptions to working prototypes include:

- *A Capabilities Browser* to advise judicious selection of robots, tools and sensing strategies. This is a family of taxonomies built on top of a commercially available knowledge engineering environment. AI tools to perform inferencing are provided.

- *A Script Generation Facility* that provides tools and a programming environment to "snap" together simple constructs like MOVEBY to compose more complex robot strategies (e.g., GRASP). We have used this facility to successfully generate robust scripts for precision assembly tasks.

• *Task Primitives Translators* that convert the semantic intent of complex scripts to a sequence of simple actions or primitives and then translate those to a target robot language. This obviates the need for the user to learn more than one robot programming language.

• *A Real Time Control Environment* that allows users to experiment with robot controller design by modifying controller PID gains or build new servo level control constructs.

• *Interactive simulation and critics* to provide interactive support during both on-line and off-line programming. This may also serve as the basis for advanced variable autonomy interfaces[4].Critics include support for robot motion planners, grasp strategy selection and the synthesis of fine motion strategies.

• Advice taking mechanisms are needed by the system to grow with the user, by incorporating past user feedback into future decision making processes.

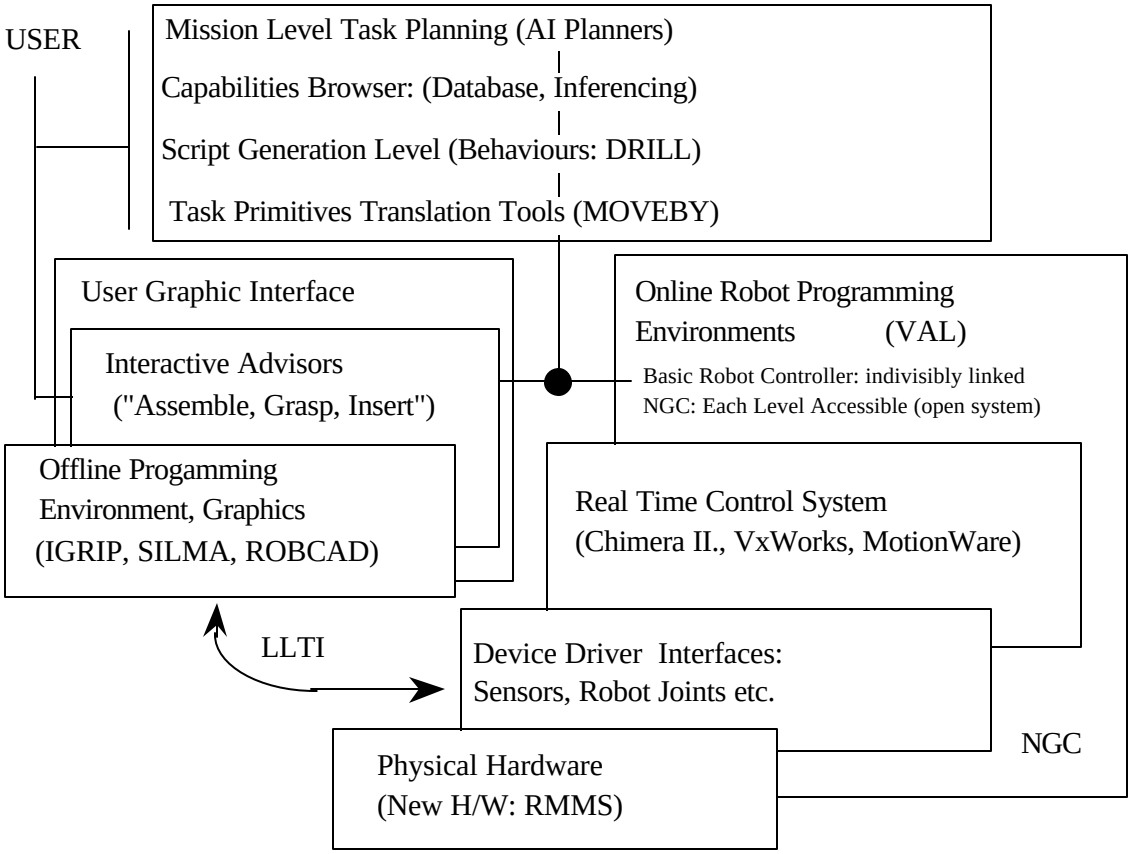

Fig 1. Prototyping Methodology: An Integrated On-line Off-line Robot Programming System

Capabilities Browser

In support of a task specification, robot system integrators must make judicious selections regarding robots, sensors, tools and control techniques. The system performance is a composite of the tradeoff made: inaccurate mechanical hardware may be compensated by sophisticated software and vice versa. Regrettably, specifications such as robot work envelope or accuracy, do not help the uninitiated make the right hardware and supporting software selections. A missing link in current robot databases is information on a basic capability, resulting from the synthesis of hardware and software components.

We have used a commercial knowledge engineering environment[23], to partially implement a a taxonomy of hardware, software and capabilities[25]. Given frame representations of a robot and commonly used sensors and grippers, the user can browse through the database to determine whether a task specification can be satisfied with off the shelf equipment, which control strategy will meet his requirements and what the cost of this capability will be. He may also review available capabilities or construct new variants.

Actuation taxonomies help users make selections based on classification of tasks in terms of force-motion-time state graphs. For example, the drilling family of tasks (drilling, reaming, boring etc.) all have the robot force and motion vectors collinear, as opposed to the de-burring family of tasks that typically require orthogonal force and motion vector applications. Sensing taxonomies describe error detection and recovery strategies that may be selected for mapping a desired end result (Acquired Tool) to the corresponding physical effects (force sensor output).

Hardware taxonomies include robots, tools, sensors and fixtures. Schema that represent their functionality and their interrelationships have been implemented to help simple inferencing. For example the user will be able to determine that a force sensor belongs to the family of sensors and can be used to sense contact or collision. He may browse through databases and discover that he could as well use a vision system but it costs more. AI tools to support dependency chaining and inferencing are available to ensure that the decisions made are "logical".

Script Generation Facility

The capabilities browser provides the data but not the technical support to help users design new capabilities or modify old ones. We now provide him with a programming environment to facilitate "snapping" together primitive building blocks to build new capabilities. Software advisors will critique how actuation and sensing portions of a strategy may be "snapped" together to forma (new) composite capability, according to well defined "legal" configurations. Primitive building blocks like compliant motion constructs will be also available so the novice user can experiment with designing new robot strategies, compose new or more abstract strategies from existing strategies without fear of violating basic design constraints. We have used such facilities[25] to build scripts for precision assembly tasks.

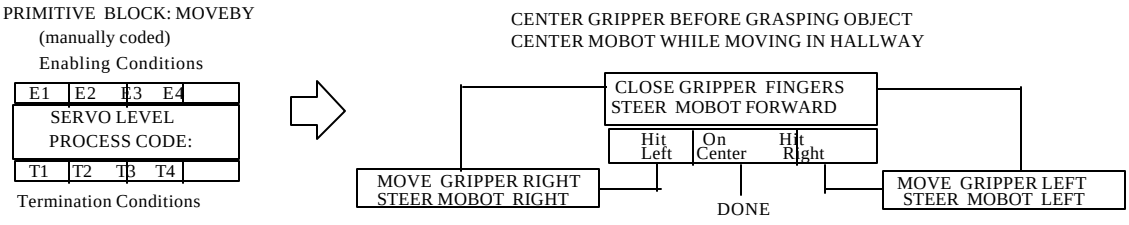

Fig. 2. Using Primitive Building Blocks (MOVEBY) to form more complex modules (CENTER)

The primitives supplied and the strategies formed are more directed towards the functional intent of a task, than on a specific implementation detail. Thus, two completely different implementations may share the same name and semantics for different environments; the concept of CENTER is semantically the same for a gripper closing on a workpiece or a mobile robot traversing a hallway (See Fig. 2). Users may experiment with semantically similar constructs and not be overly concerned (yet) about their differences related to different sensors or actuator hardware. This approach stems from observations that data flow graphs for seemingly diverse applications show marked similarity; compliant motion for a mobile robot entering a door is semantically similar to a peg in hole insertion. We wish to exploit that kind of similarity at this level of program generation.

Task Primitives Translation

We intentionally separated the semantic level of script generation from the more mundane translation mechanisms needed to "talk" to robot controllers in their native languages. At the previous level of control, we provided the user with an extendible vocabulary of task primitives, building blocks for complex program generation. At this level, we provide him with tools to build new task primitives or modify old ones.

This level of programming is to be largely transparent to the novice user for two reasons. First, the user can conveniently limit himself to one programming language and be shielded from implementation details or limitations of robot controllers. For example, most commercial robot controllers do not support hybrid motion control. However compliant like motions can be produced by translation mechanisms as shown in Figure 3. We thus circumvented a limitation of the robot controller, and provided added functionality to the user.

Second, we wish to encourage modular code generation. As robot programming languages evolve, new functionalities or *primitives* are added to a robot language vocabulary. We wish these changes to have minimal effect on application software. Device specific translators provide the desired level of abstraction: when compliant motion is supported by the vendor, the software translation mechanisms disappear. But these changes are transparent to the user or application programs. We proposed a family of translators that port programs developed in one common language to commercial robot vendor programming languages. Our common language is NOT intended to replace commercial robot languages. Instead it subsumes the primitive constructs supported by typical commercial robot language but also extend their power by providing primitives not directly supported by the robot manufacturer. It currently supports the common motion primitives: guarded and compliant motions, gross motions, and joint level robot control. Provisions for rapidly generating sensor interface routines are also provided.

MOVEBY (0,0,.1) with COMPLIANCE (3,-1,10,0) WHILE (force-z .lt. 10)

is an example of a task primitive construct. In the general case, we define a primitive as consisting of one or all of the following parameters to a process, where a process may be an actuation or sensing operation or tightly coupled interaction of the two:

- Precondition Boolean expression that must be satisfied when the process is invoked.
- Post condition Boolean expressions that, when true, terminate the process.
- Sensing activities required before, after and during the process.
- Speed, acceleration and deceleration specifications for motions.
- Compliance parameter and sensor device names.
- Controller PID gains (when supported by real time control system).

Real Time Control System

The real time operating system provides the glue that will enable robot system designers to connect different sensors, actuators, control techniques together and evaluate the real time system behavior. This level of control is not typically supported by robot vendors, however academic research test beds[12] exist. Expert users will find them invaluable in the design of better robotic devices, or control strategies. We are collaborating with universities[11] in the evolution of real time robot control environments that interface with the task primitive library and control language to include the following real time functionalities:

- Multitasking and concurrent programming ability to support a multitude of sensing, actuation and control tasks concurrently.

- To support the above, dynamic and static scheduling of processes and inter task communication protocols need to be established.

```
 ATTACH(13)            ;;initialize force sensor.
 IF IOSTAT(13) <O GOTO 1000         ;;Translator automatically
 FCMD(13, 103)                      ;;determines applicability.
 IF IOSTAT(13) <O GOTO 1000
 WRITE(13) "rs"                     ;;soft reset for JR-3 force- sensor
 WRITE(13) "ur c"                   ;;Ask JR-3 for continous force data
 CALL FORCE.READ(force[1] )         ;;resident program on robot to poll
                                     ;;force-sensor for this configuration.

 SAVESPEED=SPEED(2)     ;;commands to set speed and
 ACCEL 50                           ;;motion parms to be in sync with
 SPEED 20 ALWAYS                     ;;force sensor data, etc
 DURATION  8 TPS ALWAYS
 ENABLE CP

 WHILE (FORCE[3] < 10) DO           ;;force[3] is alias for force-z
   FCOMPLY[1l=FORCE1l]*(3)          ;;for this specific configuration.
   FCOMPLY[2]=FORCE2]*(-1)          ;;Configuration File supplies this.
   FCOMPLY[3]=FORCE3]*(1)
   MOVE HERE:TRANS(FCOMPLY[l],FCOMPLY[2],FCOMPLY[3],0,0,0)
   CALL FORCE.READ(force[])
 END

 SPEED SAVESPEED ALWAYS ;;reset motion parm- this depends
 ACCEL 100                           ;;on what operation~ needed after
 FINE ALWAYS
 WRITE(13) "rs"                     ;;close serial lines - this depends
 DETACH(13)                         ;;on that operation~ needed after
```

Fig. 3 Device Translation of MOVEBY

- Device drivers analogous to Posix device level support. This is essential to provide a layer of hardware transparency for both sensors and actuators at a preferred level of resolution.

- Ability to generate sensor monitoring processes on the fly by spawning new processes based on the current task and world state.

- Ability to execute user written exception and interrupt handlers for specialized applications.

Interactive Simulation and Critics

We proposed an interactive off-line and on-line programming environment and on-line programming environment rolled into one: the user can interact with either the real workcell or a CAD model of it through the same graphics interface. In the off-line mode, while he is generating a sequence plan software "critics" resident in the system could be watching over his shoulder, determining whether the part is capable of being manufactured in a proposed manner. If feasible, a background process is spawned to automatically generate the robot program to make the part, downloaded and executed by the robot. In the on-line mode, he can communicate with the robot, by either joystick control, task primitive level command input or by invoking (previously simulated) procedures.

The purpose of building an integrated on-line and off-line programming environment is that the distinction between the on-line and off-line is unclear when we are programming robots interactively (telerobotics) or building robot systems iteratively (prototyping). Where tools such as motion planning, should reside, depends on a multitude of factors, including the resources available, the current world situation and the task semantics. Tools often reside in more than one niche depending on these considerations. For example, we have used the same (heuristic based) motion planning algorithms on both on-line and off-line trajectory generation. Further, by using the right device abstractions we may apply critics across application domains boundaries: a general purpose potential field path planner is applicable to both mobile robot motion as well as compliant peg in the hole insertion tasks, resulting in reusable tool kits and cost savings to the end user.

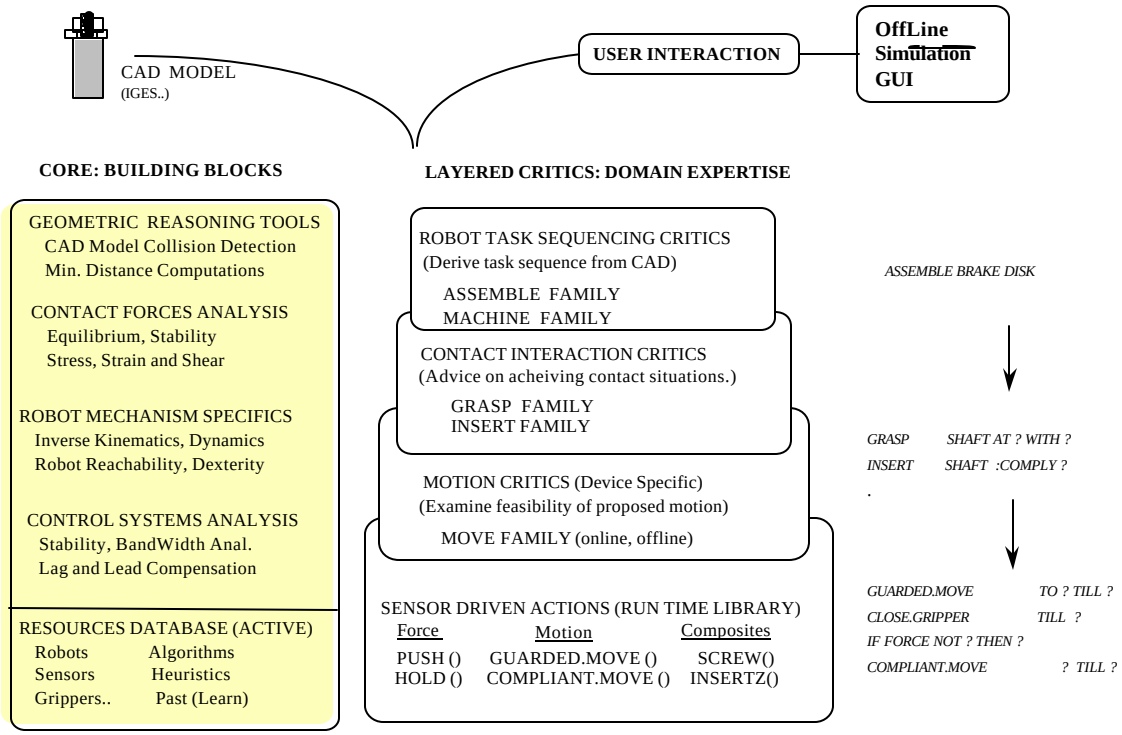

Fig. 4. Prototyping Environment for ASSEMBLE, GRASP, INSERT and MOVE

Our prototyping environment (Fig. 4) currently includes a library of core tools such as device abstracted collision detection algorithms, robot kinematics and dynamics models and some notions about static and dynamic equilibrium. On top of these core tools we are building software critics which include interactive assemblability and motion planning support.The *Assemblability Critic* critiques how to robotically assemble objects. To do this, in the general case, is non trivial. However, for most robotic assembly domains, the following assumptions are reasonable and significantly reduce the complexity of our problem space[13].

• The component does not change in dimension or shape after the assembly is completed or can be modeled as such.

• CAD models provide us with the interaction information between adjacent components of the assembly: thus we have pointers linking the face of a shaft as connected to the hole of a holder. Feature based modelling has been used to help reason about the best assembly strategy available [14] . Additionally, geometric reasoning has been applied to define what degrees of freedom a component of an assembly may have, given its contact geometry (Fig. 5, 6).

• A tool kit of special operations that constrain how the part is to be assembled exist. This includes common primitives such as insert, screw and align.

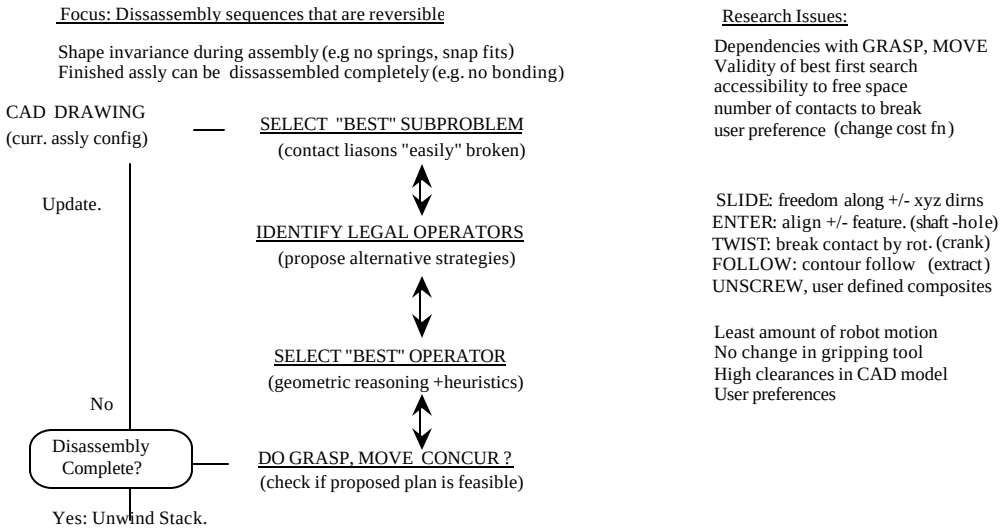

Fig. 5. Strategy for disassembly sequencing.

If the above assumptions hold, then the assembly process is the reverse of the dis-assembly process. This approach has been demonstrated for generating sequencing operations for air frame structures. It is applicable to both on-line and off-line applications.

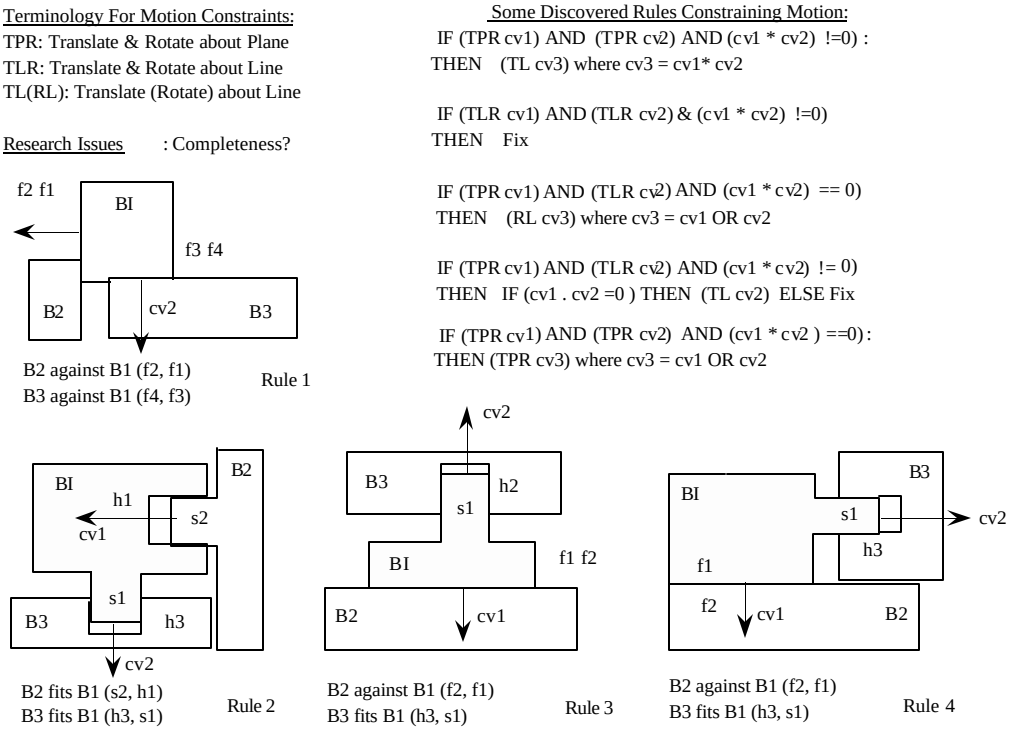

Fig.6. Some Rules defining the degrees of freedom based on contact geometry[14]

Gross motion planning[9] Critics are concerned primarily with automatically generating a collision free path from specified source and goal destinations for the current configuration of objects in the environment and the kinematic and dynamic model of the agent. The path generation process itself may be weighted to produce different paths based on the required cost function. Often, different cost functions are desirable for portions of the path. If the user is aware that one of the obstacles may be moving when the planner is invoked, he may impose a change in the type of path planning needed at or near the potentially mobile obstacle; at run time execution, the path planner will switch to a safer mode of

operation by monitoring potential collision situations more closely. We have built tools to rapidly generate the composite trajectory scripts needed for robust run time execution

Grasp motion planning[15] is a specialized application of gross motion and sensor interaction. Given a feasible approach vector, the robot and the gripper must move in to grasp the object, close the gripper fingers, ensure that the object has been grasped and move the object to the desired destination. Our focus here is parallel jaw gripping. A simple expert expert system based on the filters shown in Fig. 7 has been implemented. Unresolved problems with CAD Driven grasp strategy selection are:

• Inadequate or incomplete information on the pose of the object. Part localization techniques using vision or force sensing must be invoked to determine an accurate pose estimation of the object.

• Grasp position selection. The location at which the gripper grasps the object must provide surface normalize that allow the necessary friction forces needed to effect the grasp. Further, CAD model driven calculations of object centroid, help us select a stable grasp location.

• Error detection during grasp. We have correlated readings from a force sensor to plausible grasp location, derived from CAD model data. This was used to detect and correct for part slippage during grasping.

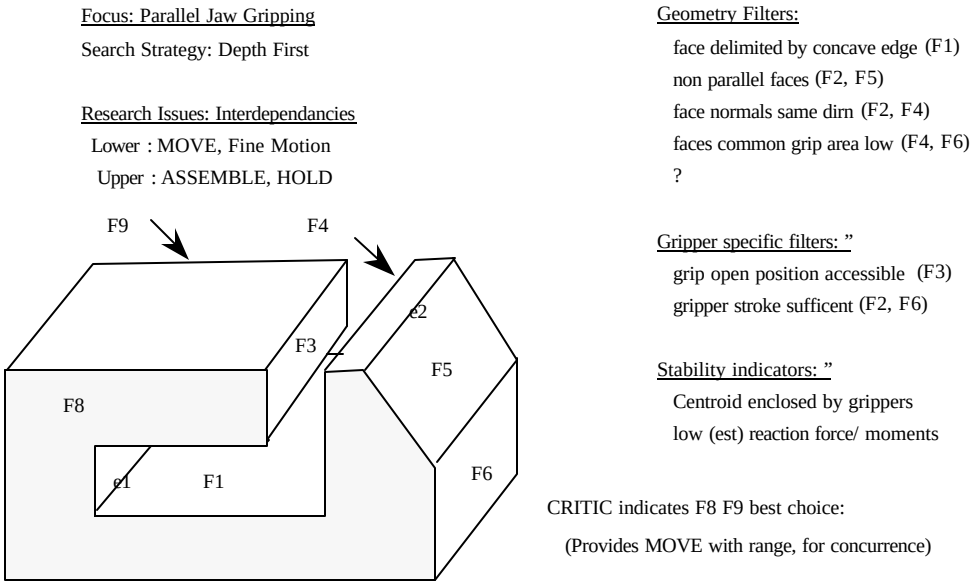

Fig 7. Applying Geometric Filters to decide the "best" location to grasp the object.

Fine motion planning[17] is concerned primarily with mating operations, generally requiring tightly coupled interaction between sensing and actuation. We have implemented a combined biased search and LMT[17] approach for the on-line automated generation of the decision tree for compliant robot motion. Given the CAD models of the two parts, we start from a location where, based on the forces and a few quick probing motions, we may determine where the point of contact is, for the geometry of interest. As we move towards the goal location, we take advantage of the fact that transitions in contact geometry are reasonably well correlated to the CAD models: by some simple rocking actions we can classify the surfaces in contact and infer our current contact situation. A decision tree is generated by repeating this over the space of probable contact configurations. Fig. 8 shows how the "rocking" approach has been applied to the common "peg-insertion" problem. In the three situations depicted, force data uniquely defines the contact situation. A precompiled decision tree enables the robot to insert the peg.

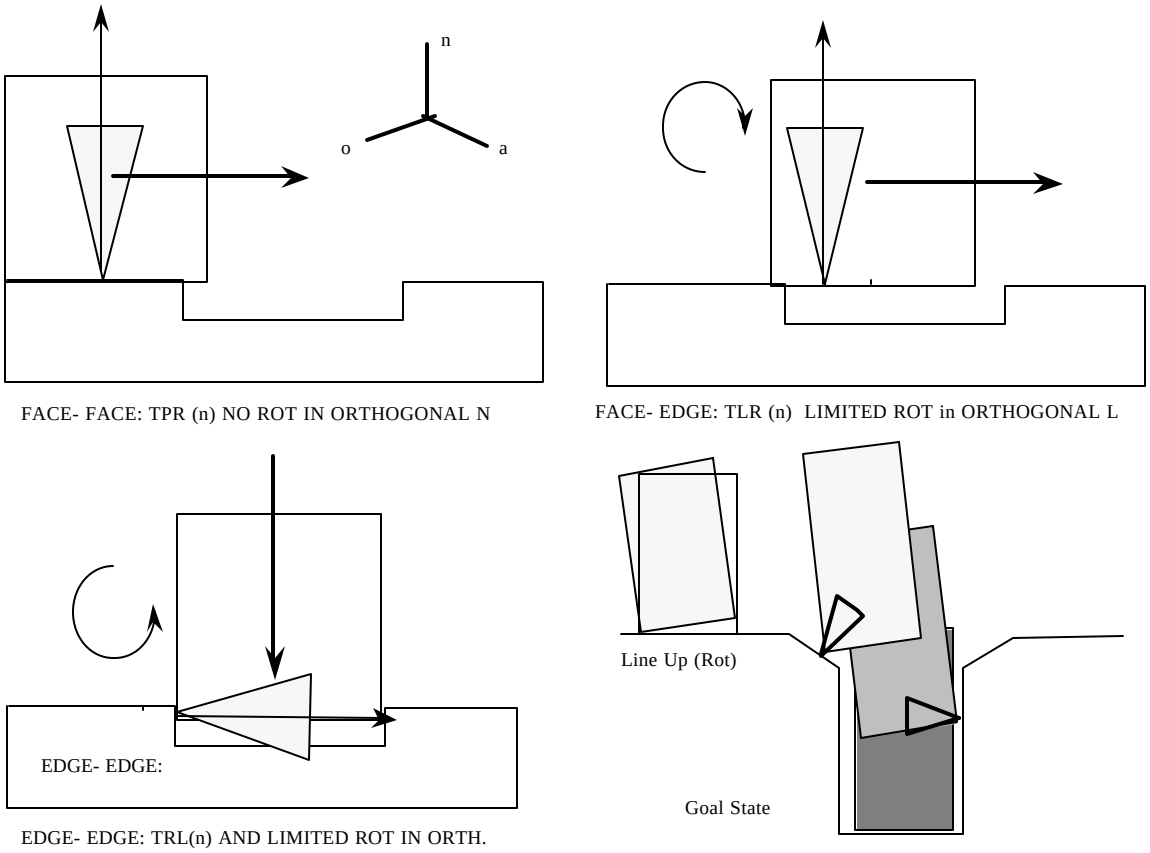

Fig.8. Successful insertion based on the results of the "rocking" action.

7 Summary

One of the missing links impeding cost effective generation of robust robot programs is the virtual absence of a supportive prototyping environment that provides constructive criticism of a proposed automation approach. In the absence of such an environment most attempts to build robot systems consider a single, usually simplistic paradigm and build their systems based on this minor modifications of this paradigm. But in general, there are several alternative approaches to automating a design requirement. These alternatives should be generated and examined in order to present an unbiased, technically sound view. Our efforts are directed towards providing the user with the following capabilities:

- The ability to examine a variety of alternate solutions for a given task.
- The ability to generate process parameters directly from the design, resulting in quick turnaround from concept to working prototype.
- The ability to re configure the system, to avoid unplanned obsolescence.
- The ability to do all of the above, at a fraction of the current cost and lead time of building advanced robotic systems for small batch situations.

References

[1] T. Cebulla, N. Kemp and J. Simpson. Robotics Application Trade Studies Report AFWAL-TR-89-8001. Manufacturing Technology Directorate, AFWAL, Wright Patterson AFB, Ohio 45433-6533

[2] S. Lee. REPTECH: Enhancing quality through repair technology. Proceedings 1988, 1989, Manufac-turing Technology Directorate, AFWAL, Wright Patterson AFB, Ohio 45433-6533

[3] NASA Johnson Annual Report. Automation and Robotics Division FY90. NASA Johnson Space Center, Houston, Texas.

[4] NASA Johnson Report JSC-54100. Space Exploration Initiative, Planet Surface systems, Humans/Automation/Robotics/Telerobotics Integrated Summary Trade Studies. NASA Johnson Space Center, Houston, Texas.

[5] Department of Energy Report DOE/CE 007. Environmental Restoration and Waste Management Robotics Technology Development Program; Robotics Five Year Plan. Volumes I-III.

[6] Department of Defense. 1991 Critical Technologies Plan. Order Number AD A234 900.

[7] M. Leahy et al. Towards RACE. Robotics and Automation Center Of Excellence, Air Force Logistics Command, TIEST/SA-ALC.

[8] J. Albus et al. NASA/NBS standard reference model for telerobot control system architecture. NIST Tech. Note 1235, NIST, Gaithersburg, MD.

[9] Rodney A. Brooks. Symbolic error analysis and robot planning, Int. J. Robotics Res., Vol 1(4), 1982

[10] Rodney A. Brooks. A Robust Layered Control System For A Mobile Robot. IEEE Journal of Robotics and Automation, March 1986.

[11] D. Stewart. Real Time Software Design and Analysis of Re configurable Advanced Sensor Based Systems: A Ph.D. Prospectus. Carnegie Mellon University, Robotics Institute.

[12] D. Stewart, R. Volpe, P. Khosla. Integration of Reusable Software Control Modules for Multiprocessor Systems. Proceedings of 1992 Intelligent Robotic Systems Conference (IROS), Raleigh, NC, August, 1992.

[13] R. Hoffman. Disassembly in the CSG domain. IEEE Conference on Robotics and Automation, 1989.

[14] A. Jain and M. Donath. Knowledge representation system for assembly using robots. Workshop On Space Telerobotics, Vol III, Pasadena, January 1987.

[15] C. Laugier and J. Pertin, Automatic grasping: A case study in accessibility analysis, Advanced software in robotics, Amsterdam: North Holland.

[16] L. Lieberman and M Wesley. AUTOPASS: An automatic programming system for computer controlledmechanical assembly, IBM J. Res. Dev., 1977, Volume 21

[17] T. Lozano Perez, M. Mason and R. Taylor. Automatic synthesis of fine motion strategies for robots, Artificial Intelligence Laboratory, M. I. T., 1983, Memo 759

[18] T. Lozano Perez and Rodney A. Brooks. An Approach to Automatic Robot Programming, Robotics and Artificial Intelligence, Springer-Verlag, New York, 1985

[19] R. Taylor. Synthesis of manipulator control programs from task level specifications, Stanford University, Artificial Intelligence Laboratory, Memo AIM228

[20] Bruno Dufay and Jean-Claude Latombe. An Approach to Automatic Robot Programming Based on Inductive Learning. Intl. J. of Robotics Research, Vol 3(4) 1984.

[21] Deneb Robotics Inc. IGRIP Reference Manual, Deneb Inc., Troy, Michigan.

[22] Silma Inc. CimStation User's Manual, Silma, Inc., Los Gatos, California.

[23] Intellicorp. KEE Reference Manuals, Intellicorp Inc., Mountain View, CA.

[24] Northrop Aircraft Company. The Automated Air Frame Assembly Cell. Automated Air Frame Assembly Program. Design Specifications, Phase I. Manufacturing Technology Directorate, Wright Patterson AFB, Ohio 45433-6533

[25] F. daCosta et al. Integrated Robotic Technology, IR&D 87-R-511, 88-R-511, 89-R-511. Automation Sciences Laboratory, Northrop Research and Technology Center, Northrop Corporation, Los Angeles.