Experiences with the JPL Telerobot Testbed — Issues and Insights —

Henry W. Stone, Bob Balaram, and John Beahan

Tele-Autonomous Systems Group
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

Abstract

JPL's Telerobot Testbed [7] is an integrated robotic testbed used to develop, implement, and evaluate the performance of advanced concepts in *autonomous*, *tele-autonomous*, and *tele-operated* control of robotic manipulators. Using the Telerobot Testbed, we have demonstrated several of the capabilities and technological advances in the control and integration of robotic systems which have been under development at JPL for several years. In particular, the Telerobot Testbed was recently employed to perform a near completely automated, end-to-end, satellite grapple and repair sequence. The task of integrating existing as well as new concepts in robot control into the Telerobot Testbed has been a very difficult and timely one. Now that we have completed our first major milestone (i.e., the end-to-end demonstration) it is important to reflect back upon our experiences and to collect the knowledge that has been gained so that improvements can be made to our existing system. It is also believed that our experiences are of value to the others in the robotics community. Therefore, the primary objective of this paper will be to use the Telerobot Testbed as a case study to identify *real* problems and technological gaps which exist in the areas of robotics and in particular systems integration. Such problems have surely hindered the development of what could be reasonably called an *intelligent* robot. In addition to identifying such problems, we will also briefly discuss what approaches have been taken to resolve them or, in several cases, to circumvent them until better approaches can be developed.

1 Introduction

JPL's Telerobot Testbed [7] is an integrated robotic testbed used to develop, implement, and evaluate the performance of advanced concepts in *autonomous*, *tele-autonomous*, and *tele-operated* control of robotic manipulators. The Telerobot Testbed consists of two Puma 560 robots mounted side-by-side; one Puma 560 mounted adjacent to the first two, called the Vision Arm; a taskboard situated between the first two Pumas, and a mockup of the Solar Max satellite mounted to a counterbalanced suspension system. The Telerobot Testbed has a number of sensing capabilities, including Force/Torque Sensing on the two Puma 560 robots, a five camera vision system which utilizes custom designed image detection and feature extraction hardware, and end-effector contact sensing. The software which controls the Telerobot Testbed is distributed among a number of computers including several MicroVAX workstations, a Symbolics workstation, and a variety of low level robot servo controllers and sensor system controllers. Each workstation contains the control software corresponding to one of the subsystems within the Telerobot Testbed's hierarchical control structure. The hardware and software integration of all these components has been a major task. The Telerobot Testbed provides a unique environment for developing integrated and intelligent robotic capabilities.

Using the Telerobot Testbed, we have demonstrated a number of capabilities and technological advances in the control and integration of robotic systems which have been under development at JPL for several years. In particular, the Telerobot Testbed was recently employed to perform a near completely automated, end-to-end, satellite grapple and repair sequence. The task of integrating existing as well as new concepts in robot control into the Telerobot Testbed has been a very difficult and timely one. Now that we have completed our first major milestone (i.e., the end-to-end demonstration) it is important to reflect back

upon our experiences and to collect the knowledge that has been gained so that improvements can be made. It is also believed that our experiences are of value to the others in the robotics community. Therefore, the primary objective of this paper is to use the Telerobot Testbed as a case study to identify *real*, problems and technological gaps which exist in the areas of robotics and systems integration. In the following paragraph we briefly describe one of the basic tasks performed by the telerobot during our previous demonstration in order to place our discussions into context.

The sample task executed by the telerobot system consisted of the removing of a bolt located behind a latched door using a standard socket tool. Task execution was complicated by the presence of a crank in an adjacent area that prevented the door from opening fully for certain orientations of the rank. The task sequence consisted of unlatching the door; using the stereo vision system to determine the orientation of the crank; combining the sensor information with a-priori information to get the best estimate of crank position/orientation; planning and executing a compliant motion to grasp and subsequently move the crank to a non-interfering location; opening the door; and finally holding the door open with one manipulator while the other manipulator arm applied the tool on the bolt. During the actual door opening operation, cooperation of both arms was needed to get the door to open. This was because a single one-arm motion to fully open the door was not possible because of a conflict between joint stop violations and compliant motion stability near the wrist singularity. As a result, the grasp of the end-effector on the door handle had to be adjusted halfway through the opening operation with the other end-effector holding stationary the partially opened door. It should be pointed out that this fixturing operation requiring the cooperation of the two arms was not anticipated during the initial development of the task sequence. Teleoperation and handoff to and from autonomy was also demonstrated for some of the free-motion segments of the task using a run-time path planner in the RTC subsystem.

This paper is divided into three sections corresponding to the three main areas in which have identified problems which must be addressed during the design and implementation of a large integrated telerobotic system. Section 2 describes problems which have plagued our system in the area of calibration and world modeling. Then, in Section 3, we switch to a discussion of the issues relating to process planning and control execution. Finally, in Section 4, we present our ideas as to what features are really important in the design of an overall system architecture and describe how the use of certain approaches to software engineering can, in practice, determine system performance. Section 5 summarizes our conclusions.

2 Calibration and World Modeling

Over the past several years and most recently during our efforts to demonstrate the autonomous and teleoperated capabilities of the JPL Telerobot Testbed system we have become acutely aware of the difficulties involved in calibrating an integrated multisensory, multiactuated robotic system. Furthermore, we have learned first hand how great of an impact calibration issues and problems can have upon the design, implementation, and performance of high level control and task planning algorithms. The difficulties that have been encountered in merging information gathered from both vision and touch sensors (i.e., wrist force/torque sensors) in order to estimate the location of an object in the environment are a prime example.

Until recently [2,6,8], the topics of sensor fusion and calibration have received little attention within the robotics research community in comparison to say artificial intelligence for task planning. In response, this section will: (1) highlight several of the problems which we have had to deal with in the design of the Run-Time Controller (RTC), Manipulator Control Mechanization (MCM), and Sensing and Perception (S&P) portions of the JPL Telerobot Testbed; (2) attempt to reinforce the need for increased research in the areas of multisensor system calibration, sensor fusion, and system architectures which explicitly take into account calibration and data fusion. Our experience indicates that the importance of many of the practical issues involved in multisensor calibration and sensor fusion to the design of useful and realizable robotic systems for space applications as well as other applications should not be underestimated.

The autonomous manipulation of an object by a manipulator requires that the location (i.e., position and orientation) of the object be known to within a specified tolerance with respect to the manipulator. In general, this tolerance is dictated by: (1) the positioning accuracy of the manipulator; (2) the accuracy of the object model database which describes the relative spatial relationships between all of the objects in the environment; and, (3) the constraints imposed by the physical configuration and/or motion limitations of the end-effector (gripper). To overcome these inherent inaccuracies, to accommodate mechanical constraints, and to increase system robustness the Telerobot Testbed utilizes contact sensing and stereo vision to assist in the run-time and real-time identification and verification of object locations.

Within this context consider the task of grasping one of the task board's door handles in preparation for opening the door. The door handles are rectangles 2.0 cm by 2.0 cm by 4.0 cm high. This apparently simple task is complicated by fact that the parallel jaw gripper being used has a maximum finger separation of only 2.54 cm. This provides a maximum clearance of only 2.2 mm per side assuming perfect end-effector orientation. This is further complicated by the Puma robot's inability to position its end-effector that accurately, in absolute coordinates, when controlled using standard (i.e., nominal) kinematic parameters. Even if the actual kinematics of the maniplator were identified [8] it is extremely difficult if not impractical to determine the relative spatial transformation between the robot's base coordinate frame and the location of the object to this accuracy. As discussed in [8] and [5] manipulator kinematic models are based upon relationships between nonphysical internal coordinate frames and not to coordinate frames which can be physically identified.

The structure of the Telerobot's global world model database, which drives the RTC planning and event monitoring software, also has a bearing upon this problem. This tree topology connected database provides an on-line object oriented means for computed object positions. The tree topology was based on the inherent physical connectivity of objects, with parent object motion resulting in automatic update of all successor objects. In the case of objects with more than one parent, the spanning tree corresponding to the relational graph was used. Completely independent of the connectivity tree is a measurement tree reflecting the metrology tree used in the building of the initial database. For practical reasons, only portions of this measurement tree coincide with the connectivity tree.

Geometric errors in the world model database are primarily a result of errors in the orientation and position of objects described with respect to other objects. The relative spatial transformations between these objects was initially derived from a combination of mechanical drawing specifications and manually taken measurements. Of these errors, orientation errors were especially troublesome because of the long *lever-arm* effects over workspace distances. With this modeling strategy errors in each of the relative transformations can accumulate into significant absolute location errors. In the case of grasping the door handle, the relationship between the manipulators base coordinate frame and the door handle's location depends upon the relative spatial transformations of between approximately 10 - 20 different primitive objects. The net result has been that when the robot is commanded to move to a cartesian location relative to the door handle, such as a desired grasp approach point, it will often be in error by 5-10 millimeters. Naturally, any subsequent motion to grasp the object will fail unless additional sensory information is used to either update the object database or guide the motion of the end-effector.

An approach which was originally pursued to overcome this problem utilizes a stereo vision system to identify the location of the door handle (or object) and update the database just prior to manipulation. The use of vision, however, has given rise to difficult calibration issues similar to those involved in determining the locations of robot base coordinate frames. In particular, one needs to accurately determine the relationships between: (1) the location of the door handle relative to the camera image frame; (2) the location of the camera image frame and the last link of the robot to which the cameras are attached (i.e., the camera robot); and, (3) the location of the base frame of the camera robot and the base frame of the robot which will perform the manipulation.

The accuracy of first of these relationships is directly related to the accuracy with which one can determine the spatial relationship between the two camera image frames. Comparisons of the relative

locations of object features (e.g., corners, edges) of a known object to feature locations computed based upon a camera model have been used to identify the camera model parameters. In practice, camera model parameter identification is performed off-line prior to system operation. The second relationship, the location of the camera frame and the last link of the robot, is especially difficult to measure since it has an unknown relationship to any of the physically accessible surfaces of the cameras. In fact, the camera frame may likely be located at a point between the cameras in free space. In the testbed, the camera robot is mounted to a wooden table which is bolted to the concrete floor about a meter from the lathebeds which support the two other robots and the task board. Consequently, the camera arm and the other arms have no common physical means for registration. Nevertheless, manual measurements have been used to approximate the spatial relationships between the robots. During the measurement process the orientational deviations between the plane of the camera robot's table and that of the lathebeds are ignored due to the limitations of the measurement equipment. Such deviations, however, can propagate into significant positioning errors between the robot base frames and the frames associated with the objects of the task board.

Although the above approach to object location verification has many drawbacks, it has been successfully used to determine the locations of a variety of objects. In a tele-autonomous system such as ours, the sensed object location information must be incorporated into the system's world model. This process of updating the spatial relationships within the database presents additional problems. The most significant of these is that of maintaining database consistency. Consistency in update is maintained when geometric relationships between objects are not violated as a result of the update. For instance, we know that the door handle does not intersect with the door; rather they share a common surface. Similar statements can be made about the door, its parent object, and so on. But the vision system, due to calibration errors as well as measurement errors will likely return an answer which would, according to the database, place the door handle partially within the door. While systematic approaches to solving this problem have been proposed [6] they are generally unsuitable for implementation due to their large computational requirements and/or their need for statistical parameterizations which are unavailable. Maintaining such consistency in the absence of a generalized approach proved to be extremely troublesome and hence efforts were initiated to provide an algorithmic approach to the problem.

In terms of database philosophy, geometric uncertainty representation per-se is not critical. Our planning methods did not reason with uncertainty and hence it was only necessary to project all of the geometric uncertainty onto a unique and *certain* geometric world model state. Similarly, regarding reasoning with uncertainty, an approach that updates the world-model after each execution step with the best estimate of the geometric state seems to be adequate. Propagating uncertainties to future actions may not be a profitable approach given the paucity of techniques for modeling such uncertainties.

Meanwhile, in order to satisfy various short term needs, we have applied several ad hoc schemes to manipulate and update the world model spatial relationships. These schemes use a concept called *localized models* whereby the a priori relative spatial relationships between objects within various portions of the database (i.e., subtrees) are assumed to be perfect. If the location of an object within a subtree is sensed the required change in the object's location is propagated to an equivalent change in the spatial relationship between the parent object of the subtree and its parent.

This concept is now being applied in conjunction with vision based object verification to provide a vision based relative calibration capability. The objective of this calibration is to apply the vision system to view and locate both the object of interest as well as the end-effector, and to determine the relative transformation between the two. Based upon this relative transformation, the end-effector can then be moved to a location that is within the a priori limits required for subsequent task execution. This approach takes advantage of the fact that the relative accuracy of the vision system is significantly greater than its absolute accuracy. The information obtained from the vision system will then be integrated into the system's world model using the *localized model* approach. In particular, the update is acheived by assuming that the manipulator's kinematics and kinematics used in control are precisely the same. Thus the absolute location of the end-effector is specified by the measured joint angles. The location of the root object of the localized model is then updated such that the object/end-effector relative transformation is identical to that which is

observed. Then, even if the absolute locations of the end-effector and its immediate surroundings are not known with great accuracy one can still have confidence in success of subsequently planned autonomous fine-motions.

A drawback of the *localized model* concept, however, is that a significant amount of bookkeeping is required to maintain a list of the satisfied versus unsatisfied geometrical constraints which will exist within the database. Following long periods of operation it can be argued that the knowledge as to the overall state of the world may actually decrease despite the fact the large quantities of sensory information have been obtained. Implicitly, information is continually being discarded each time the connectivity of localized model is changed. The ad hoc nature of this approach is not well suited for large complex systems since it requires significant amounts of custom engineering. In fact, we have experienced a number of situations in which ensuing side effects have changed the apparent behavior of our path planner and collision detection algorithms. Fortunately, the ever present kill button has saved us from damaging the testbed.

This section has presented a few of the problems which we have faced in the areas of calibration and world modeling. This presentation is by no means exhaustive. We hope, however, that this introduction provides some awareness to the problems being faced by those trying to build real robotic systems and, in particular, robotic systems for space applications. Conclusions are presented in Section 5.

3 Process Planning

Planning for telerobot processes i.e. actions such as move, grasp, rotate etc. presented special problems because of the complex and interacting nature of a very large number of problem constraints and characteristics. Among these were the usual constraints arising from the kinematic construction of the Puma 560 arm such as reach, joint stops (especially joint 5), and limited manipulability over large regions of the workspace. Added to these were spatial constraints in the form of inter-arm collision avoidance for the three arms, arm/object collision avoidance, object/object interference avoidance during manipulation, and object occlusion during vision operations. Dynamic constraints included performance degradation and even instability of compliant motion operations near singular arm configurations, and poorly known gravity and friction effects.

The effects of these constraints were further compounded by the requirement to maintain fidelity to the telerobot demonstration task. This implied that in order to preserve the resemblance to a space servicing task, the environment could not be significantly simplified (e.g. by performing the operations on a flat clutter free table). Further, inordinate amounts of computing resources could not be expended during the planning process. This was both because of execution time constraits imposed by operator/machine interaction requirements and because machine speed was constrained by the available hardware. While the hardware speed restriction could have been overcome by upgrading the laboratory system, the inavailability of high-speed space-qualified computing hardware would have equally hampered a real space application.

The demonstration task also required that elemental robot actions be concatenated to perform a meaningful task. Concatenation implies that actions selected in the current step affect the constraints on future tasks, and requires the ability to invoke some form of constraint propagating, backtracking planner. Some of the constraint propagations only affected simple geometric aspects of the task (e.g. selecting a grasp point to facilitate a future ungrasp and depart operation) whereas others changed the entire task sequence (e.g. a forced dual-arm coordination to overcome an individual arm's manipulability constraints). The concatenation of actions also placed a heavier burden on verifying the successful completion of each step. Design of recovery procedures in case of off-nominal execution was also necessitated, especially given the highly variable friction effects in metal-to-metal contact, and calibration uncertainty effects in different regions of the workspace.

The final challenge in the area of planning was to successfully blend engineered solutions to algo-

rithmic approaches. The boundary between the two was often fluid as algorithms which did not work were engineered to do the job and vice-versa, though admittedly more of the former than the latter.

The combination of factors outlined above made a standard rule or logic based planning system very difficult to design. An alternate approach in the same spirit of the *generate and test* paradigm was adopted instead. This approach utilized action selection rules and procedures based on a partial handling of the various constraints to generate candidate actions. Detailed world-model driven simulation of the action was then used to accept or reject the candidate action. For example, the path segment design algorithm focussed on simple single-arm kinematics and collision avoidance features of the task to generate a candidate motion segment. The acceptance of the motion segment was based on performing a detailed high-fidelity model-driven simulation of the segment to unearth manipulability constraints and collision problems with the other arm. From the viewpoint of the planner based on this concept, actual execution results are indistinguishable from the simulated results. Thus collision during the guarded motion execution of the path segment constituted the same information to the planner as a world-model simulation indicated collision along the path segment.

This generate and test cycle of operations owed its success to two features (1) the ability to synthesize candidate actions that have a high probability of success (2) the use of simulation based checking of such candidates. While it is intuitive to use the best possible methods to determine the candidate actions (e.g. a configuration space graph search for paths), the introduction of a performance criteria that penalized execution speed and assigned a weighted score to the probability of success showed that it was sometimes better to utilize approximate but computationally cheap methods to generate and test the candidate actions. For inherently computationally hard problems such as spatial planning in a variable environment where pre-planned paths are inappropriate, such an approach may indeed be the only feasible one [3].

The generate and test paradigm also lends itself to easy operator interaction in the overall decision making. The action choices can be selected and pruned by the operator and the simulation process can use operator input for compensating for operator modeled phenomena not modeled in the planner or for judgment based acceptance of candidate actions. One useful side benefit of the simulation process was the potential ability to plan sensor monitoring processes. For example, during the path segment simulation, objects neighbouring the path segment could be monitored for distance and direction and reflex actions or sensor event detection predicates could be synthesized accordingly. This capability was however not exercised in the laboratory and hence we have no direct assessment of its utility.

4 System Architecture and Software Engineering

In a well established and understood field, the construction of a system without first exploring its formulation and design in great detail is foolhardy. Unfortunately, this rigor is not feasible in a research environment, where requirements fluctuate at an extremely high rate. In these situations, success is often determined by the ability to rapidly react to significant design changes. The JPL Telerobot demonstrator project clearly lies within the research domain, and the system design was incrementally modified every six months or so. The size of the system (which eventually exceeded 200,000 lines of software and 5 primary CPU's) and the short development time (less than two years from initial concept to first major integrated demonstration) necessitated an extremely efficient use of time and hardware resources (physical manipulators, not computers). One of the consequences of this was the requirement for interactive development placed on most subsystems and subsystem components. This had a strong influence on the architectural evolution of the system, as we discovered the advantages of a fully interactive environment. In the following we briefly describe the issues and insights that have been encountered in the area of overall system architectural philosophy.

It is essential that all system capabilities be interactively available for use by the human, since augmentation of human capability is much more effective for robust, flexible telerobotics than using purely autonomous robots or pure force-reflecting teleoperation, which is not robust in the presence of time delays.

Our experience indicates that the low- and medium-level actuation and sensing subsystems, which may have been originally designed for use by higher level software, are in themselves a very powerful set of tools which can be used directly by a human to perform tasks, without the need for any complex planning or analysis software. This observation, that a human can often serve as a functional component in a software system, is significant because a human can usually replace vast quantities of complex software with only minor overhead for provision of appropriate user interfaces to make the human appear to the rest of the system as just another software module. Although this point has implications for software development and debugging, it crucially impacts the design of the overall system architecture, since it can make differences of orders of magnitude in the ability of a large robotic system to perform practical tasks in a realistic environment. Typically, it was relatively straightforward for the developers of the RTC to use their interactive software development environment to control the actuation and sensing subsystems, and the individual algorithms and internal modules of the RTC, ***by hand***, enabling them to perform tasks which were much more complex, ***and more robustly executed,*** than the integrated RTC autonomous software could attempt.

Existing work in telerobot systems architecture [1] has concentrated on a perspective of well-structured environments which allow the design of the telerobot to concentrate mainly on how to divide the task to be done efficiently between several cooperative robots. Our work indicates that space telerobotics is not a highly-structured task, and that concentration should be placed on the reactive, feedback portion of the architecture, which allows the robot to respond to the environment. To this end, the cooperative human-machine concept gives much greater robustness and flexibility than any existing or proposed autonomous system promises.

In most large robotic systems, there is a low level subsystem which functions as a server of basic actuation/sensing functionality to the rest of the system. Typically, it is *not* the case that a higher level system can use its interface to actually command any given motion or sensing operation which the hardware devices are physically capable of performing. There are usually many classes of actions which cannot be commanded due to interface "blind spots". Practical experience indicates that such blind spots often lie precisely in the regions which later turn out to be crucial to the functioning of the system as a whole, and thus must be avoided at all costs. Each low level subsystem must be built from the perspective that anything which its "designer" would ever want to do with it can be done *through the interfaces provided to other subsystems.* Often, this is not the case, since the builders of the low level subsystems often work in a very flexible software development environment during implementation, but do not themselves rely upon the interface to perform their own tests. The ease with which they can directly use their subsystem to perform tasks, owing to the great robustness and adaptability of humans, misleads them into thinking that it will be quite sufficient for use by the higher levels of software. Correction of any design omissions which lie in the blind spots then becomes very costly.

The ability to shift functionality across subsystem interfaces easily is also crucial. During initial design, conceptually complex tasks are often divided up semi-arbitrarily and choices are made for locations within the various subsystems for each function, usually based largely upon such considerations as "level of intelligence", computational and bandwidth requirements. However, since the introduction of a new algorithm with improved functionality can force the shifting of a large body of capability from a higher level subsystem to a lower one, the architecture of the system should conveniently support it.

We now present several issues which we found to be of prime importance during the software development process. They are well-understood by the professional software engineering community, but the small size of most robotics developments to date has not yet provided much cross-fertilization. Our aim here is to convey a healthy respect for the magnitude of the software task for large integrated systems, which can easily overshadow the robotics issues in terms of time and effort spent.

First of all, the most important issue in large systems software development is human, rather than technological: *frequent, detailed communication between virtually all members of the development team is absolutely required.* Top-down management techniques can provide some aid to the development of large software systems *if the top-down method is applied to the system as a whole, rather than just one or more*

subsystems, but the bulk of the design effort must be done by the many individuals on the various development teams. To prevent minor choices on the part of subsystems from propagating disastrously to the system, every individual must understand enough of the overall system to measure his decision's impacts on it. This is not an easy situation to maintain, but the alternative is a collection of independently powerful subsystems whose integrated capability is much less than the sum of their individual talents. The next paragraphs discuss some of the issues which specifically relate to the software development process itself.

The techniques which individuals use to manage and design small software systems are often precisely those which create the greatest difficulties in a large system developed by a team [4]. We have learned that there is a great need to impose at least the minimum of standards and techniques from the professional software development regime, in spite of many individuals' resistance to them on the grounds of the additional overhead they impose. In the absence of unusually disciplined developers with an excellent rapport, a system built without careful control of software development will be significantly, if not overwhelmingly, inferior to what it could be, in performance, development time, resources used, and future adaptability. In general, good software discipline can make the difference between a team which is performing research, and a team which is continually struggling so heavily with software and hardware implementation difficulties that they have virtually no time for broad thought about the goals and subtleties of the system they are building. Underestimating the difficulty of a hardware/software item can be one of the most dangerous mistakes to make, because once hardware/software problems pass a certain degree of desperation in some subsystem, the members of that subsystem's development team tend to withdraw from the overall system design effort due to lack of free time, and all the decisions they make from then on will then not only be made with a very narrow and short-term focus, but will not even be communicated to the system as a whole until they are largely irreversible.

Exacerbating the complexity of software development for systems of this size is the fact that systems are built as a pyramid, with low level software tools and low level subsystem interfaces being used as a foundation. Changes to these tools and interfaces which occur without extremely careful monitoring by all members of the system development team can have catastrophic effects, since pyramidal software systems tend to behave like houses built of cards: pull one out of the bottom and everything falls. In the area of performance, it was discovered that the choices which turned out to have the greatest negative performance impact on the overall system were those which were made by individuals working within the subsystems who did not recognize that their decision impacted the system as a whole, and thus did not consider it important enough to bring to the attention of the systems-level group.

Abstraction is probably the single most important software concept in building large systems, precisely because of the pyramidal nature of software. Abstraction, sometimes called implementation-hiding or factoring (which is distinct from modularity), is the practice of inserting additional layers into implementations for the express purpose of decoupling external users from the internal details of a piece of software. A trivial example is the following: suppose a library of graphics routines was rebuilt to use absolute coordinates in centimeters rather than coordinates relative to the display screen. If the library were built as a single layer, there would be no choice but to change every bit of code using the functions to perform the conversion of coordinate systems, which would then also become device-specific for each graphics monitor. However, if an additional level of abstraction had been intentionally inserted into the library when it was built, the conversion could be performed in this intermediate layer, internal to the library, so that the change in implementation need not propagate outside. This obviously imposes some additional overhead, but is almost universally preferable to performing complicated additions and modifications to previously stable software. The larger the system the more certain changes are to occur, the more severe their repercussions can be, and the more important the use of conceptual abstraction is to software design. In particular, it is essential that subsystems use sufficiently abstract interfaces to prevent minor internal modifications from being propagated to other subsystems and the system as a whole.

The need for abstraction is one of the strong drivers behind a philosophy of development which we call *tool-oriented*. The need to insert additional conceptual layers forces the developers to be continually examining at each subcomponent they are building to see how it generalizes, and what sorts of changes

and additions might be likely in the future. This effectively transforms the design of each subcomponent from a very specialized exercise into the task of *designing a set of software tools* which can be used in a very straightforward way to build the desired subcomponent. This tool-oriented methodology imposes additional overheads and larger initial investments of time and resources, but, we have estimated (based on our experiences when we did not pursue this course) that our long-term productivity has been roughly twice as high as it would have been had we initially built exactly what we thought we needed. The reason for the large gains in spite of the high initial investments required is that research systems require a high frequency of changes and it is vastly easier to rebuild something built with modular tools than it is to rebuild a single monolithic program.

Robustness is also an issue in software engineering and usually is far more important to large systems than efficiency. The tightly-coupled software environment of a large system forces every component to do double duty as both a functional unit and as a diagnostic tool for other modules and the system as a whole. The additional investment in time, both for coding and during execution of the additional error-checking, is well worth the added reliability. In a system with tens of developers and several dozen modules, if the low levels are not reliable, the higher level modules will be forced to devote significant portions of their effort to coping with the drawbacks of the modules they depend upon.

In summary, the task of managing software development for a large system is very different from that of a small one, and most of the decisions which can be safely made with only small consideration and little fear of error for small projects become magnified into major issues for large ones. It may, however, be quite difficult to convince those without experience in software engineering of this. The issues discussed above may not appear to be very significant to them, since they may not recognize how significant the gains in performance and robustness would be from their system if short-term choices were sometimes sacrificed to long-term needs.

5 Conclusions

In the area of calibration and world modeling we believe that: (1) the difficulty and cost associated with calibrating a multi-actuated, multi-sensor robotic system is grossly underestimated; (2) seemingly small calibration errors in a complex robotic system can easily propagate into large errors and thereby significantly reduce the overall capability and performance of the systems as a whole; (3) low-level subsystem calibration problems can have a significant impact upon the design of higher level subsystems; (3) the calibration of all of the actuation devices (e.g., manipulators and grippers) and sensors (e.g., force/torque, position, vision, etc) which comprise a robotic system must be performed within a coherent and common framework; (4) ad hoc and/or custom engineered methods for maintaining world model consistency, such as the *localized model approach*, are not suitable for large scale robotic systems; and (5) research in robotics needs to focus, in part, on the development of generalized methods for representing, manipulating, and propagating object spatial uncertainties in a language (e.g., geometry) suitable for use by task planning and control algorithms. In short, do not take the effects of calibration errors lightly.

Robot planning for real world applications becomes extremely difficult because of the complex and interacting nature of the various physical constraints and run time uncertainties. The effect of these becomes especially evident if long sequences of robot tasks need to be performed in a robust manner. In this paper, some of these challenges have been outlined and approaches such as fixturing and world model simulation have been discussed. A planning and execution method based upon a probabilistic approach incorporated into a *generate and test* paradigm has been advocated for its efficacy in overcoming computational constraints associated with these types of problems.

In the area of system architecture, the key features we believe are important are: (1) the interactive cooperation between human and machine during operation of the system to enable the human and the autonomous system to call upon each other as tools to perform specialized functions; (2) the importance of

the reactive feedback loop as opposed to nominal execution of preplanned tasks; (3) insuring that there is no functionality lost between the level of the sensing and actuation and the level of the operator. In the area of software engineering, we found the important issues to be: (1) interpersonal communication among the development team members; (2) discipline during the software development process; (3) high levels of conceptual abstraction; (4) build tools with which to build applications, not single applications; (5) emphasize robustness over efficiency; and, (6) interactive rather than incremental development environments.

References

[1] J. S. Albus, H. G. McCain, and R. Lumia. *NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM).* Technical Report , National Bureau of Standards, Robot Systems Division, National Bureau of Standards, Robot Systems Division, December 1986.

[2] P. K. Allen. *Robotic Object Recognition Using Vision and Touch.* Kluwer Academic Publishers, Boston, MA, 1987.

[3] J. Balaram. Probabilistic methods for robot motion determination. In *SPIE Symposium, Advances in Intelligent Robotic Systems*, Cambridge, MA, November 1988.

[4] F. P. Jr. Brooks. *The Mythical Man-Month (Essays on Software Engineering).* Addison-Wesley, New York, 1982.

[5] M. Driels, B. Mooring, Z. Roth, and L. Everett. Fundamentals of manipulator calibration. In *the Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, Philadelphia, PA, April 1988.

[6] H. F. Durrant-Whyte. *Integration, Coordination and Control of Multi-Sensor Robotic Systems.* Kluwer Academic Publishers, Boston, MA, 1988.

[7] P. S. Schenker. NASA research and development for space telerobotics. *IEEE Transactions on Aerospace and Electronic Systems*, 24(5):523–534, September 1988.

[8] H. W. Stone. *Kinematic Modeling, Identification, and Control of Robotic Manipulators.* Kluwer Academic Publishers, Boston, MA, 1987.