Abstract—Intelligent robots interact with the real world by employing their advanced sensory mechanisms to perceive their environment, and using their effectors and tools to change the state of their environment. Some of the important capabilities which endow “intelligence” to those robots include, (1) planning, i.e., given a goal, the ability to generate a set of task plans which will lead to achieving the goal, (2) coordination and execution of the perceptual actions, i.e., the abilities to coordinate sensors, acquire sensory data, and process, interpret and transform the sensory information, and (3) coordination and execution of the motor actions, i.e., the ability to navigate in their environments and the ability to coordinate effectors and tools and manipulate objects to accomplish assigned tasks. In this paper we introduce a System Architecture for Sensor-based Intelligent Robots (SASIR). The system architecture consists of Perception, Motor, Task Planner, Knowledge-Based, User Interface and Supervisor modules. SASIR is constructed using a frame data structure, which provides a suitable and flexible scheme for representation and manipulation of the world model, the sensor derived information, as well as for describing the actions required for the execution of a specific task. The experimental results show the basic validity of the general architecture as well as the robust and successful performance of two working systems: (1) the Autonomous Spill Cleaning (ASC) Robotic System, and (2), ROBOSIGHT, which is capable of a range of autonomous inspection and manipulation tasks. Simulation and animation techniques were employed, in addition to the real-world testing, during the system development. The system components were successfully transported to another research laboratory involving a different type of robot, different sensors, and a different physical environment.

I. INTRODUCTION

Advanced robotic systems should be capable of performing complex tasks in highly unstructured and dynamic work environments. These robots interact with the real world by employing their advanced sensory mechanisms to perceive their environment, and using their effectors and tools to change the state of their environment. Some of the important capabilities which endow “intelligence” to those robots include, (1) planning, i.e., given a goal, the ability to generate a set of task plans which will lead to achieving the goal, (2) coordination and execution of the perceptual actions, i.e., the abilities to coordinate sensors, acquire sensory data, and process, interpret and transform the sensory information, and (3) coordination and execution of the motor actions, i.e., the ability to navigate in their environments and the ability to coordinate effectors and tools and manipulate objects to accomplish assigned tasks. A truly robust working system can only be obtained by systematically coordinated modules performing in concert within the architecture. Real-time performance constraints of the robot’s environment impose and motivate resolution of many critical and practical problems encountered in the development of the individual modules of the system. In other words, by designing and implementing a system that must operate effectively in the real world, one can improve the capabilities of the separate components as well.

Until recently, several new directions in designing planning systems have emerged towards solving problems in real world applications [1], [2]. Among those new research directions is the integration of planning and execution. Especially for sensor-based robotic systems, the issues associated with planners are plan execution, replanning, plan error recovery, etc., which arise almost inevitably in a real, dynamic environment [3]. The research reported in this paper is directed towards the resolution of some of the important problems involved in the design and implementation of a sensor-based intelligent robotic system; specifically, the planning, coordination and execution strategies for perceptual and motor actions in such a system.

In general, an intelligent robotic system should consist of a perception module for sensing the environment, a motor module for navigation and manipulation, and an intelligence module for decision making and control. Because building the complete working system is a very complex task, most reported work deals with subsystems or proposed designs. Only a few working systems has been reported [4]-[9]. Automatic task planning capability is not explicitly embedded in their control architectures to direct the sensing and motion activities. There are robotic systems which do integrate an automatic task planner as part of the control architecture [10]-[13], however, all these systems are designed for navigation with no manipulation requirement (except [10] with a simple manipulation capability), and it is not evident that the same planner can also be used for planning integrated sensing and manipulation tasks.

Consideration of the integrated system as the primary research focus provides a unique perspective on resolving issues underlying design, architecture, and implementation of the
In this paper we present our research activities on building an integrated intelligent robotic system in two subtopics: (1) A System Architecture for Sensor-based Intelligent Robots (SASIR), which consists of six basic modules: Perception, Motor, Task Planner, Knowledge Base, User Interface and Supervisor, and (2) A Planner for Robots with Integrated Sensing and Motor capabilities (PRISM), in which planning and execution are interleaved, and sensing and motor actions are integrated.

The research focus is on designing robotic systems for inspection and manipulation tasks. Therefore, issues associated with mobility and autonomous navigation of platforms are not considered. Inspection is recognized as an important task for robotic systems involved in a wide variety of application domains. Some of the examples include inspection systems for hazardous environments such as nuclear power plants, and robotic applications involving process control and manufacturing.

It is important to note that the robotic systems discussed in this paper perform both inspection and manipulation in order to accomplish a goal. For example, if an inspection task requires moving an object occluding the sensor’s field of view, then the robot utilizes its manipulation ability to move the object. Also, since the movement may require an obstacle free trajectory, the inspection capability can be utilized to insure this. Thus, proper integration of perception and motor abilities must be emphasized. The integration of perceptual, planning and motor functions is at the core of the Active Perception field. This area is gaining increasing importance in the research community [14].

One of the important features of the reported research is related to the performance verification phase. In addition to the extensive experimental studies with the integrated implementation of the robotic systems in the Computer Vision & Robotics Research Laboratory at The University of Tennessee, a number of important system modules were transported (electronically) to another physical location1. These modules were integrated in a system involving different types of robots (HERMIES IIIB, and HERMIES III [15]), and cameras, as well as a physical environment characterized by different illumination conditions and structural details than those present in the CVRR laboratory. It is believed that such extensive and “multi-location” implementation and experimentation has provided very valuable and unique experience with which to judge issues such as system modularity, portability, flexibility, generality, and robustness.

Finally, the research utilizes a number of reasonably complex and realistic scenarios to direct the system implementation efforts. One of the scenarios involves automatic detection, localization, and clean-up of spills in the robotic work environments, and another involves inspection and manipulation tasks associated with a control panel. We believe that the nature and complexity of the case-studies addressed in this work are much more involved than the popular and somewhat simplistic scenarios involving well-structured objects in idealized (“clean”) environments.

An integrated, intelligent, robotic system consists of interconnected subsystems of sensory, perception, planning and control, and manipulation and mobility. Also, for practical and effective robotic solutions, one must closely examine the work environment as a subsystem to be integrated. Thus, in general, intelligent robots can be viewed as complex systems involving interconnections of component subsystems, each with a specific functional utility. The design of such complex systems should be approached at different levels of abstraction. At a higher level one’s interest is to propose an architecture for the system. A number of recent studies have examined issues associated with the higher-level design aspects of an intelligent robot [9], [4], [16]. This section presents an overview of an architectural framework for designing sensor-based intelligent robots.

The importance of a careful examination of the various architecture level details can not be overemphasized. These details directly influence some of the main systemic requirements of intelligent robots. These include system efficiency, modularity, flexibility, portability, man-machine interaction, complexity, ease-in-expandability, and generality. The objective is to consider an architectural framework where a robotic system can be systematically configured with different types of robots, hardware, and application domains, but with the same underlying architecture. Some of the specific higher level system architecture related issues are: (a) functional modules constituting the system, (b) proper interconnections between the functional modules, and (c) the control and knowledge representation strategies which govern the system operation.

In order for an intelligent robotic system to exhibit purposeful behavior in accomplishing a goal, it should be able to perceive, plan and reason with objects and events appearing in spatio-temporal domains. While proposing an architectural framework for the intelligent systems, the primary concern is how to to provide the above system capability most effectively. This requires representation schemes and structures for describing and manipulating both declarative and procedural sets of facts dealing with various objects and events associated with the system operation. A knowledge representation scheme which provides suitable means for representing such declarative and procedural knowledge is that of frames [17] or schemas [18].

A robot system architecture can be implemented using frames for representing the models of the environment and the robot as well as for describing the actions required for the execution of a specific task. Minsky [17] defines a frame as a data structure which consists of “a network of nodes and relations.” The top levels of the frame are usually associated with the general and static (always true) information about the entity represented by the frame. The lower levels of the frame contain terminals (slots) in which specific and dynamic information is stored. Each terminal may have a set of facets and corresponding values, or a set of subframes. The frame data structure is based on the theory that to deal with dynamic

---

1 CESAR, Oak Ridge National Laboratory.
changing situations, one can select a certain type of frame from memory and adapt the frame to generate strategies or solutions by instantiating or filling the necessary lower level terminals (slots) in a systematic fashion. The advantages of utilizing frames in the system include:

- Frames provide a powerful and flexible structure for representation and manipulation of the world model along with the sensor derived information. These knowledge representational structures have been successfully employed in intelligent system designs [19], [20]. Utilization of frames in the robotics context will be, however, quite different since in this case it is a closed-loop system cycle involving perception and action.

- Frames will be useful for both perception and action phases of the system. This is accomplished in a systematic manner by a task planning approach (described in the next section) where the exact nature of the control and execution cycles of the system is determined by the information provided by the appropriate frames associated with a goal or a subgoal. These goals or subgoals may involve either a perception or motor action tasks.

- Traditional task planners use well-formed formulas (WFF) in predicate calculus as a means of constructing their plans. However, WFF is not appropriate in representing perceptual and motor actions in robotic systems operating in the real world, since these actions contain lower-level details which cannot be described by the high-level symbolic representations of WFFs [21]. Frames, on the other hand, are well suited for representing both the high-level symbolic constructs and the lower-level, domain-specific knowledge, procedures, and constraints. In addition, frame representation of plans supports hierarchical planning, iterative and conditional planning, replanning, error recovery and backtracking.

- Frames allow us to represent the state of the work environment and robot before and after undertaking a specific action, as well as for describing the procedural details involved with the performance of an action.

- Compared to other types of data structures, such as lists or arrays, a frame is a structured representation scheme which allows better implementation of autonomous behavior than explicit rule applications in other representation schemes [22]. Furthermore, procedural knowledge and operations can also be easily encoded in structured representations. While other structured representation such as semantic network is suited for describing complex objects or events, it is not suited for representing hierarchical task plans with different levels of abstraction.

These are some of the main reasons for selecting frames as the knowledge and data representation scheme in this research.

This section presents an implementation of a robotic system using frames for describing the models of objects that the robot will have to identify and manipulate, as well as those for describing the details associated with a specific inspection and manipulation task that the robot has to perform. This System Architecture for Sensor-based Intelligent Robots, SASIR, consists of six basic modules (Fig. 1):

- Perception,
- Motor,
- Task Planner,
- Knowledge Base,
- User Interface, and
- Supervisor.

SASIR uses frames [17] for knowledge representation, and planning and control strategies. For example, frames are used for storing high-level and primitive plans in the Knowledge Base, as well as for describing spectral, spatial and relational properties of object models in the work environment. The Perception module utilizes these object frames to recognize and match object features detected in the processed sensory data. The Task Planner constructs plans which are also in the form of frames, and those primitive plans (perception or motor plans) are identified by the Supervisor and passed to the Perception or Motor module for execution. This consistency of using frames throughout the system makes a standardized, efficient implementation of the system and its components.

The SASIR architecture provides a very powerful and general framework for the design and implementation of intelligent robots. The architecture imposes a functional modularity. It forces examination of the design process at a task level so that the user does not have to master the details associated with specific robot hardware and low-level software. A very simple process allows for redefinition of the task frames associated with an application. Thus, one can experiment with a range of alternate task execution strategies which can be used in the attainment of the stated goal. Similarly, simple modification in the frames associated with the workspace allows the system to operate with a wide range of objects, tools and environments. The SASIR architecture supports a systematic and smooth interaction between the operator and robot. This allows for reconfiguration of the levels of autonomy assigned to the intelligent system. This also provides for graceful recovery from errors. In addition, the SASIR system can be easily implemented to address the on-line performance requirements of a wide range of robotic applications.

Another important feature of the SASIR system deals with the simulation of sensor-based operation in the robotic workcell. This allows for a “forward-simulation” of the robot’s behavior to verify the feasibility of the planned actions (Fig. 2). This off-line programming is very useful during the testing and debugging stages of the software development. Fig. 2 shows the functional block diagram of the robotic system. In addition to the basic system components’ functionalities, this figure depicts the integrated sensor-based robot simulator in the system.

A novel sensor-based intelligent robot simulator is integrated in the SASIR system and is working in concert with all the modules of the system; it makes the switching between real and simulation modes in the SASIR system transparent to the user. This paradigm has potential usage in improving off-line programming in industrial robots as more sensors will be utilized by those robots in the near future [23]-[25].
III. AUTOMATIC TASK PLANNING: RELATED RESEARCH

Automatic Task Planning has been a major area of research in the artificial intelligence field [26]-[30]. Task Planning in a robotic system can be defined as the generation of a set of actions which allow the system to achieve a desired goal by changing the robot work environment through successful executions of the actions.

Robot planning is typically examined by two groups of researchers. Those interested in path planning and obstacle avoidance aspects associated with mobile robots and those interested in the development of the task sequence primarily for manipulatory robots. Because of the differences in the nature of tasks to be performed, the planners developed are quite different in these two cases. Note that in the case of mobile robots, the nature of tasks are basically obstacle avoidance and goal seeking and there are typically only two degrees-of-freedom to be considered. These robots typically utilize some sort of proximity or range sensors and based-upon these signals one has to control their movement in the workcell (typically, only 2-D). The manipulatory robots, on the other hand, need to be concerned about the control of a range of tools, effectors, and multi-DOF manipulators, using a rather broad range of contact as well as non-contact sensory devices. In this case the work cell involves a 3-D volume and the range of tasks include simple obstacle avoidance on one hand to a complex assembly of manipulatory function on the other hand. The obstacle avoidance and goal seeking behaviors can be realized by reactive systems whereas the complex manipulatory tasks require deliberative planner. In the following we shall discuss a number of task planners useful for manipulatory robots. We will also provide pointers to the mobile robot related planning studies. The task planner for mobile manipulator, however, remains as a future research area (Fig. 3).

A. Task Planning for Manipulation

There are two basic approaches to planning. The first is the state-space approach where a path from an initial state to a state satisfying the goal specification is searched and identified. The other approach is the action-ordering approach where given an initial high-level plan (or skeleton plan), a set of fully detailed plans leading to the completion of the goal is generated [27]. Of these two approaches, the action-ordering based planning has advantages over state-space approach in the robotic inspection and manipulation domain because of the complexity involved in the integration of sensing and action as well as geometrical constraints on the actions and temporal relationships among the plans. It will be very difficult
Hutchinson and Kak [21] describe SPAR, a planner which uses a similar representation of actions described in STRIPS, with additional preconditions of operational, geometric, and uncertainty-reduction associated with each action. SPAR is designed to generate plans for assembly tasks in an uncertain environment. In SPAR, the preconditions are the pending goals to be satisfied. Planning is accomplished in two phases. In the first phase, the planner iteratively refines the current partial plan by either constraining the execution of an action already in the plan, or inserting a new action which satisfies the pending goal into the plan. The preconditions of the new action are added to the goal stacks. The iteration stops when there are no more pending operational or geometric goals. In the second phase, the uncertainty-reduction preconditions are considered for specific instances of a plan. These instances are created by instantiating the variables in the plan actions based upon the consistent solution for the plan's constraint network. The plan instance which satisfies all uncertainty-reduction goals, or the one with the fewest unsatisfied goals, is selected. Sensing is used to determine the positions and orientations of the objects in the initial state. However, there is no interaction between sensing and action being reflected in the plan during its execution. In addition, SPAR does not generate error-recovery plans or perform replanning automatically.

B. Task Planning for Navigation

The focus of this paper is on robotic systems for manipulation. In this section we briefly discuss a number of recent papers describing interesting ideas for task planning for mobile robots, mainly for navigation. We emphasize once again that in mobile robots where manipulatory capabilities are either absent or limited, planning requires support of only two types of functions: goal seeking and obstacle avoidance. These functions, especially obstacle avoidance, can be typically realized by reactive mechanism. Thus for the combination of mobility and manipulatory tasks, in addition to avoiding obstacles several other functions such as object recognition, grasping, assembly, disassembly, and general multi-DOF manipulatory functions need to be performed and the planner must support deliberative as well as reactive behaviors.

As mentioned earlier, constraints imposed on real world applications have generated new research directions in planning. Some of the work reported recently includes utilizing combined planning and reactive behaviors for robot navigation [10], [13], [12], [11], [32], [33].

Simmons [10] presents TCA, a Task Control Architecture for coordinating planning, sensing, and action. It combines reactive behaviors and hierarchical planning to control and coordinate mobile robot systems. The TCA is capable of interleaving planning and execution, detecting changes in the work environment, error recovery, and coordination multiple tasks. The design philosophy of TCA is very similar to ours, there are differences, however, that 1) TCA is for mobile robots with little or no manipulation capability whereas SASIR/PRISM is mainly for inspection and manipulation, and 2) TCA employs task tree structure while PRISM uses frame data structure for representing sensing and action plans.
fig 3. Three different planners needed in robotic applications. Research activities have been in two separate areas: planning for mobile platforms, and planning for manipulators. A task planner for mobile manipulator remains as a future research area.

explained in later sections, one of advantages of using frames is that it allows the planner to specify the starting task plan at any level, which is a very useful feature for system testing as well as for human-machine interactions.

Arkin [13] describes AuRA, a design of a mobile robot navigation control system based on motor schemas. The author states that complex behavior can be achieved by combining a group of motor schemas, each corresponding to a primitive behavior. The author concludes that the schema-based navigation technique is able to reflect easily to the uncertainty in perceiving the environment. This concept of using schemas shows some advantages in concurrent processing and reactive/reflexive systems. The design goal of our system, however, is more towards deliberative behavior than reactive.

Musliner et al. [11] present CIRCA, a real-time cooperative architecture which guarantee to meet hard deadlines by limiting number of its inputs. The authors have provided a detailed comparison of CIRCA to TCA, AuRA, and ATLANTIS regarding reactive and cooperative system architectures. In our planner, meeting a hard deadline was not a major issue in the design, rather, an assumption was made that each subtask will have a sufficient time-slot to be accomplished. In time-critical situations such as stopping the gripper motion because of unexpected obstacles or events, it is handled by hardware interrupt and software interrupt service routines. For example, both force/torque sensor and audio sensor can stop the motion immediately. Thus, our planner can still achieve timely performance by on-line response to the changes in the dynamic environment.

In order to design a task planner module of a robotic system capable of operating in a dynamic and unstructured environment, the abilities for integration of sensing and action, along with error recovery, interrupts servicing, replanning and conditional and iterative planning are important considerations. There is also trade-offs between deliberative and reactive capabilities of a robotic system. Although it is critical for the system to have certain reactivity to survive the real world, it is even more important for the system to have high-level, deliberative behaviors to achieve a given task. Our task planner, PRISM, within the SASIR architecture, is designed to perform automatic task planning in inspection and manipulation application domains, given a high-level goal.

IV. TASK PLANNING IN SASIR

The SASIR system architecture, described in the previous section, delineates the proper interconnections between various modules and a prescribed control mechanism to direct the overall operation of a sensor-based robot. This section discusses the development of a task planner associated with this control mechanism. It describes the details of an automatic planning approach specially designed to suit the unique requirements imposed by the sensor-based operation of the
system in which planning and execution are interleaved, and sensing and motor actions are integrated. This planner for robots with integrated sensing and motor capabilities is abbreviated to PRISM.

A. Characteristics of the Planning and Control Functions

Planning can be defined as the ability to derive a sequence of operations required to accomplish a goal, given the present state of the system and a set of permissible operators capable of state transformation. Such capability is a very important feature of an intelligent system. An intelligent robotic system utilizes this facility to generate the plans which are later executed by its manipulator and effector subassemblies. This provision would have been sufficient if the system was limited to operate in a static and well-structured environment. The emphasis of the study presented in this paper, however, is on robots for dynamic and unstructured environments, and therefore planning as described above is not by itself sufficient to accomplish a task in the work environment. The additional capability that must be included in the planner module is that of control (specifically sensor-based control). This is achieved by integration of perception and motor tasks by the planner. That is, sensing and motor functions of the robot must be tightly coupled in order for the robot to operate successfully in a changing environment. Therefore, unlike some other planners in which sensing is not required or is performed a priori, sensing is not treated as a separate preplanning or postplanning step in PRISM. Note that while performing a set of motor functions, the perception module of the robot is monitoring the changes and unexpected events in its environment. This sensor-derived information must be properly utilized during the execution of a planned activity. If the sensory information signals changes in the work environment, then the system must be capable of replanning its actions given the additional knowledge that it has acquired. As mentioned earlier, most traditional planners concentrate on strategies of generating plans but not on the execution of these plans [2]. PRISM, combined with Supervisor, Perception and Motor, interleaves planning, plan execution, and monitoring activities. The following specific list consists of desirable characteristics that the planning and control functions of a sensor-based intelligent robotic system should possess.

1) Integration of sensing and action
2) Hierarchical planning
3) Replanning
4) Automatic error recovery
5) Ability to service interrupts
6) Conditional and iterative planning,
7) Ability to support learning, and
8) Integration of planning and simulation.

The SASIR system structure allows one to design a task planner suited for automatic planning in a complex domain such as robotic inspection and manipulation tasks. While most researchers concentrate on designing and demonstrating the task planner by itself, in order to build a practical, working robotic system, however, the interaction between the task planner module and all other modules in the system should be a major consideration. In the following, the outline the salient characteristics of the task planner will be presented, and the details of the planner will be discussed in the next section.

- Integration of Sensing and Action in Planning: While developing plans for systems without sensing capability it is possible to generate the complete action sequence that needs to be followed by the system to reach its goal. This is accomplished by utilizing the detailed knowledge of all relevant objects and events characterizing the spatio-temporal environment associated with the robot and its assigned task. It should be noted that for many practical situations faced by an intelligent agent, it is impossible to provide such complete and detailed knowledge of the work environment a priori. For example, an intelligent robotic system may be assigned a task of cleaning a spill which might have occurred somewhere in its work environment. In order to accomplish this goal, the planner for this system cannot devise the detailed action sequence without the accurate knowledge of where exactly the spill is. This knowledge may not be available a priori but can be acquired by the system if it is provided with a sensing capability. For such a sensing endowed system, the planner can actually derive a sequence of operations which would systematically invoke the appropriate sensing and action tasks for accomplishing a series of subgoals leading to the eventual spill clean-up goal. The approach is basically to use various sensing operators to acquire the necessary knowledge (information) in concert with and needed by the various action operators which actually transform the states.

The SASIR architecture provides the framework for systematic incorporation of sensing while planning the actions of the robot. The Task Planner in the system utilizes two types of executable primitive plans, one for performing a motor function (via the Motor module), and another for selected sensing operations (to be performed by the Perception module). The Supervisor module of the system activates the appropriate modules based upon the requests issued by the Task Planner. The Task Planner module has the ability to decompose a high-level goal specified by the user into a series of subgoals characterized by a sequence of executable primitive perception and motor plans. It utilizes the knowledge of the work environment stored in the Knowledge Base along with the procedural knowledge of various operators coded in their respective frames.

- Hierarchical Planning: In order to accomplish a complex, nontrivial goal, the robotic system will require specification of a very detailed sequence of individual action and sensing tasks. It is desirable to provide the robot an ability to describe its plans at various levels of details and abstraction. This way the user may communicate the task assignment to the intelligent system at a high level. Associated with such a higher level plan there will be a hierarchy of subplans with increasing amounts of specific details required for the accomplishment of the overall task. At the lowest level of this hierarchy are the executable primitive plans for either the Perception
or Motor modules. As described earlier, the SASIR system utilizes frames for representing the plans. These structures are ideally suited for hierarchical encoding of knowledge.

It is important to note one additional feature while discussing this hierarchical planning capability. In the SASIR system, the detailed subplans associated with a goal will be generated and instantiated only when the system is ready to execute the subplans. Until that time, however, the planner (PRISM) works with a higher level plan which has sufficient information to proceed in its search for a goal.

- **Replanning:** While performing a series of operations, a robotic system may encounter a situation where the plans provided for accomplishing its goal are inadequate or unexpected events have occurred during the plan execution. The system would be able to realize this only after it has proceeded to execute the planned actions. In such situations, it is necessary for the Task Planner module to derive, if possible, an alternate series of actions considering the overall goal, subgoals successfully executed, the knowledge of the inadequacy of the previously selected strategy, and the present state of the environment. This capability is identified as replanning.

- **Error Recovery:** A practical robotic system designed for real-world operations must possess the capability to recover from such an erroneous state. While detection of the erroneous state is the responsibility of the Perception module, the error recovery plan is generated by the Task Planner module. This is similar to the replanning feature described above. The difference is that in error recovery, the planner attempts to provide a series of actions which will enable a possible recovery from the error without considering the overall goal of the system, whereas in replanning, consideration of the overall goal is quite important.

- **Interrupt Handling:** Again, for practical systems, one needs to provide a capability to interrupt a normal course of operation of the system in the event of a higher priority task assignment. The Task Planner has an ability to identify the nature of an interrupt and the sequence of available operations required for the service of the interrupt. It should be noted that the interrupt may occur at an arbitrary instance and the system needs to resume its normal course of operation after servicing the interrupt.

- **Conditional and Iterative Planning:** In developing plans for accomplishing nontrivial goals in a complex robotic domain, the Task Planner should be able to support conditional and iterative planning capabilities. Basically, conditionals such as "if then else" and iterators such as "repeat until" are required in specifying the detailed course of actions that a planner must provide to the robot. The need for such provision has been noted recently in the literature [27], however most practical planners do not support this capability. This limitation directly impacts the planner's (and therefore, the entire system's) capability in addressing complex and realistic situations

and tasks. In the SASIR system, since frames are utilized for representing individual tasks and operations, they can provide the conditional and iterative planning capabilities quite naturally.

- **Learning:** Learning can be considered as a fundamental characteristic of an intelligent system. The system should be able to utilize its past experience in developing plans for performing a stated objective. Most of the current planners are ahistoric in the sense that they do not utilize historical information associated with the systems past performance [27]. It is desirable that the system remember its failures and learn not to make the same mistakes again. The SASIR architecture can support this learning ability. Note that the system will be required to develop alternate plans after detecting an error. The system can record the events leading to the error as well as the series of actions (error-recovery plan) which were successful in bringing the system back on its normal course. This historical information can be utilized to derive new and alternate plans for specific subtasks. The SASIR system modules of User-Interface and Knowledge Base are quite important units in providing a practical learning mechanism. There are two basic ways: 1) learning by examples given from the operator through the User Interface module, and 2) learning through automatic error recovery and storing the new successful plans into the Knowledge Base.

- **Integration of Planning and Simulation:** Simulators are quite useful in their ability to derive the model of a workspace and robot efficiently. It is quite desirable to blend this powerful description (or representation) of the robot's environment with the logical and reasoning capabilities that an intelligent system should possess. This allows for developing systems which can be tested for safe and realizable operation on the forward simulation mode. Only if this performance is satisfactory is the system allowed to perform the actual task.

It should be noted that in some of the research studies, simulation is utilized as the only means for testing, verifying and illustrating the system performance. However, the issues involved in the development of sensor-based intelligent robots are so complex that no simulation can substitute the actual physical implementation of the system. Therefore, the main function of the simulator used in the SASIR system is not to replace the eventual implementation but to facilitate rapid development of the hardware and software modules as well as to ensure safe operation of the real system. The ultimate goal and focus should always be on developing actual robotic solutions which are implemented in real-world environments. The evaluation of the SASIR system performance is conducted exclusively with real-world components.

**B. PRISM: A Planner for Robots with Integrated Sensing and Motor Capabilities**

This section presents the details associated with the implementation of the planner, PRISM, for sensor-based robots.
As described earlier, this planner plays a crucial role in providing the SASIR system architecture with many desirable characteristics.

The SASIR system is designed mainly for inspection and manipulation tasks. The purpose of automatic task planning is to enable the robot to accomplish the user-specified, high-level task plans (goal plans) by generating the subsequent task plans (down to the level of primitive perception/motor plans) based on the object model, robot model, and environment models. Notice, however, that the changes in the status of the objects, robot, and environment will influence the outcome of the plans generated. That is, given the same goal plan and the initial conditions, the set of subsequent plans varies according to different state of the world.

As stated earlier, the main limitation of most of the existing planners is due to two assumptions normally made [27]. These are related to the requirement of complete and certain knowledge of all objects and events which characterize the robotic work environment. Specifically, first it is assumed that the system has a complete and certain knowledge of the state of the environment. Second, it is assumed that all changes in the states of the world are caused only due to a known set of actions. In the situation of dynamically changing world such as those characterizing the inspection and manipulation task domains, replanning, error recovery, and interrupt handling capabilities are needed when execution of the current plan is interrupted or a failure of a planned action occurs. The requirement to cope with unanticipated situations in a complex environment distinguishes the task planner designed for inspection and manipulation from the task planners designed for other domains of applications such as path planning or assembly. The Task Planner module described in this section, follows action-ordering approach instead of state-based approach for the reasons stated in the previous section. In PRISM, the format of the plans consists of the templates (described in the next section), along with preconditions, an action list, an add list, and a delete list. Thus the basic format is similar to those used in other planners described in the previous section, such as STRIPS and SPAR. The main difference is that in PRISM, the planning process is tightly coupled with perception, motor, knowledge base, user interface and supervisor modules of the system. Issues such as integration of sensing and action, replanning, error recovery, learning, and integrating planning and simulation are systematically considered in the design of the planner.

1) Terminologies and Illustrations: The following is a discussion of the terms and concepts which are important in the development of automatic planning capability for the SASIR system. Examples are given to clarify the use of the terminologies.

   a) Predicate: Predicate is used to describe the status or attributes of an object. It is used by the planner to access the Knowledge Base to acquire the information characterizing the current state associated with an object. Note that the predicate in this context is not represented by the well formed formulas (wffs) as is the case in the planners utilizing predicate calculus [34].

   Examples:
   a) Hold(Device_Name, Object_Name)
   b) Part_Location(Part_Name, Position, Reference_Frame)
   c) Part_Status(Part_Name, Status)
   d) Sensory_Data_Status(Sensor_Name, Status)

   b) Operator: Operators are utilized to describe various actions which transform states of objects in a work environment. These are defined by specifying the initial (preconditions) and the resultant states. (see bottom of page for examples)

   Examples:
   a) Select_Strategy(Purpose, Device_Name, Procedure_Name)
   b) Process_Sensory_Data(Purpose, Procedure_Name, Sensory_Data)
   c) Manipulate_Object(Procedure_Name, Object_Name, Device)

Examples of Plans:

<table>
<thead>
<tr>
<th>Operator Id</th>
<th>Preconditions</th>
<th>Add List</th>
<th>Delete List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move_Sensor (Sensor_Name, Position, Reference_Frame)</td>
<td>Part_Status(Sensor, Mounted_on_Gripper) or Pickup(Sensor)</td>
<td>Part_Location(Sensor, Position, Reference_Frame)</td>
<td>Part_Location(Sensor, Old_Position, Reference_Frame)</td>
</tr>
<tr>
<td>Move_Tool (Tool_Name, Position, Reference_Frame)</td>
<td>Hold(Gripper, Tool) or Pickup(Tool)</td>
<td>Part_Location(Tool, Position, Reference_Frame)</td>
<td>Part_Location(Tool, Old_Position, Reference_Frame)</td>
</tr>
</tbody>
</table>
1) **Plan**: A plan is an ordered sequence of operators or plans. There are two types of plans: primitive or complex, based upon the type of operations that the plan is required to perform.

2) **Primitive Plan**: Primitive plans are those that can be directly executed by either the Perception or Motor modules. Thus, given the operator set described above, a plan is primitive if it contains only one operator.

3) **Complex Plan**: Complex plans are those which cannot be directly executed by either Perception or Motor module. They can be user-specified or automatically generated by the Task Planner. (see bottom of next page for examples)

2) **Plan Generation, Decomposition, and Instantiation**: Given the above predicates, processes and operators, the Task Planner can generate a set of subplans, based upon the user-specified high-level task plan. The plan generation procedure will basically follow the control and execution cycles shown in Fig. 6, respectively. The Task Planner generates complex or primitive plans given a goal or sub-goal. The decomposition of the plan is not static in the sense that the outcome depends on the current situation (workspace status) perceived by the sensors. This is accomplished in the following ways:

- The predefined plans which are acquired through the Knowledge Acquisition module and stored in the Knowledge Base are fetched by the Task Planner. This occurs in most of the situations if the Knowledge Base contains a sufficient number of plans.
- If the plan fetched is a primitive perception or motor plan then it is instantiated at the time the plan is fetched from the Knowledge Base. Instantiation of a plan is the assignment of specific values to the input parameters of the plan. For example, the input parameters for an image processing and interpretation plan are the image name and size, camera identification (assume there are more than one camera), the position and orientation of the camera at the time the image was acquired, the reference system name, and the parent plan name.

For a motor plan, instantiation includes specifying the motion velocity, destination position and orientation, the reference system name, and the parent plan name.

- Automatic generation of the plans occurs based on the current plan’s preconditions and action lists. This is accomplished by the task planner recursively examining the unsatisfied goals in the precondition list and the goals in the action list, along with the execution status of the previous plan and the status of the environment, and then generating plans to satisfy those goals.

- Plans can be constructed through the User Interface module. This is a "learning by an example" strategy. It occurs when the system fails to continue the operation and has released the control to the operator. Thus the operator can, for example, construct a complex plan based on the available primitive plans. The Task Planner then saves the user-constructed plan in the Knowledge Base so that the next time a similar situation happens, the plan can be fetched and applied.

- The Task Planner also maintains a history list for all the plans generated in the Knowledge Base so that in case of failure, backtracking can be performed if necessary.

Notice that in the planning, control and execution cycle, if the generated plan contains only one operator, then it is a primitive plan and is sent to the proper module for execution; otherwise it will be classified as a complex plan and the plan generation procedure continues. Thus PRISM, like most other planners, generates plans using a depth-first search technique. However, it does not generate the entire set of plans at the beginning of the operation but constantly reacts to the responses of other modules in the system and changes in the robot’s environment, and performs planning and replanning accordingly. Therefore, even if the initial state of the world and the given task remain the same, the behavior of the system may be different at different times and situations depending on what happens during the operations.

The format of plans in PRISM utilizes a frame data structure which allows hierarchical and recursive specifications.

---

**Examples of Plans:**

Plan: Locate_Object(Object_Name, Position)
Preconditions: Hold(Gripper, NoPartBlockingView)
or Putdown(Part)
and Part_Location(Object, Estimated_Position, Reference_Frame)

Action List: Select_Strategy(Detect_Object, Sensor, Object_Detection_Procedure)
Move_Sensor(Sensor, Estimated_Position, Reference_Frame)
Acquire_Sensory_Data(Sensor, Sensory_Data)
Process_Sensory_Data(Purpose, Object_Detection_Procedure, Sensory_Data)

Add List: Sensory_Data_Status(Sensor, Data_Processed)
Part_Location(Object, Position, Reference_Frame)

Delete List: Sensory_Data_Status(Sensor, Old_Status)
Part_Location(Object, Old_Position, Reference_Frame)
of a plan and its parameters and sub-plans. A plan contains a set of variables: PlanId, PlanType, PlanName, ParentPlanName, and ReturnStatus. It also contains a set records: ParameterList, PreConditions, ActionList, AddList, and DeleteList. Each record may also include other variables and records. For example, a PreCondition record consists of NumberOfItem, AndPredicateList, and OrPlanList, as well as two parameter lists ParameterListA and ParameterListB associated with the predicate list and plan list, respectively. The relationship within the predicate-plan pair is a logical operator “OR”, and the relationship between different pairs is a logical operator “AND”, i.e., if the predicate returns a “FALSE” value, the corresponding plan will be executed; otherwise the plan will not be executed. The “AND” operator means that all the pairs in the precondition list must be checked. Specification of other records is accomplished in a similar fashion. Thus, complicated plans with different types of parameters, preconditions and actions can be easily implemented using such a hierarchical and recursive data structure. The preconditions of the current plan are examined against the present status of the robot and its environment when the Task Planner is activated by the Supervisor. The plan required to satisfy the AndPredicateList in the precondition list is activated into a current plan and sent to the Task Planner recursively. Fig. 4 shows the block diagram of the Task Planner module.

A plan will be instantiated and sent to either the Perception or Motor for execution if that plan is primitive and its preconditions are satisfied. The status of the execution is returned through the variable ReturnStatus in the plan. In the case of “SUCCESS” status, the next plan in sequence becomes the current one and the plan execution continues. A “FAILURE” or “INTERRUPT” status, however, will cause the Task Planner to activate the StatusServer, which is a process responsible for recovering errors and servicing interrupts. In the current implementation of the StatusServer, the recovery plan or interrupt-servicing plan is selected from the Knowledge Base according to the specific error or interrupt, and the present status of the robot and its environment. Examples of these plans include alternative placements of the sensors, tools and end-effector, based on inverse kinematics calculation for Motor module, and alternative lighting conditions or processing parameters for Perception module.

Once a complex plan with its preconditions satisfied, each sub-plan in the ActionList will now become a current plan until all the sub-plans are executed. If the sub-plan execution is successful, the status of the robot and its environment will be updated through the AddList and DeleteList of the plan by changing the corresponding contents of the STM in the Knowledge Base. As illustrated earlier, ActionList can also contain a process. The execution of a process involves selecting a strategy for a particular inspection or manipulation subtask by constructing (generating) a complex plan. For example, in order to pick up a particular tool, it may require a set of actions consisting of moving the gripper to the location above the tool (assume the tool has been localized), opening the fingers, moving down, closing the fingers, disengaging the tool from its holder, and moving up. Another example is in solving the “Blocks World” problem, an UnstackBlock plan can be made more efficient by using a process FindSpace to check the present state of the blocks, the sequence number of the block to be unstacked, and the sequence number of each block already on the table such that the minimum space requirement for putting the blocks on the table can be achieved. In a situation when the system fails to continue the operation autonomously, so that the operator is asked to input a plan manually, a process ComposePlan can also be used to allocate memory space, read in the information, check the plan format, and store the plan in the Knowledge Base.

V. COORDINATION AND CONTROL SCHEMES IN SASIR

In this section the details of the Supervisor, Knowledge Base and User Interface modules of the SASIR system are presented. As stated earlier, the role of the Supervisor module is to schedule and activate each individual module and control the overall flow of the system operation. During the plan execution, the Supervisor also monitors the operation of, and handles the interrupts generated by the Perception and Motor modules. The User Interface module provides the necessary interface between the robot and its operator. This is essential in the development of a system with provisions for human interaction, supervision and override of the robot operation. It also allows the operator to construct new plans to be executed by the Task Planner in case of failure or error recovery. The Knowledge Base module contains two main components: a) long-term memory, in which information about the robot, sensors, and workspace, along with high-level, user-specified task plans, and general
Fig. 6. (a) Control cycle of the SASIR System. (b) Execution cycle of the SASIR System. The Supervisor module activates each individual module, and controls the overall flow of operation. It first fetches the high-level user-specified task plan from the Task Planner module. Next it examines each step in the plan. Then the Supervisor module will pass the primitive plan to either the Perception or Motor module for execution. Complex plans will be sent back to the Task Planner module, and the sub-plans of that complex plan will be generated by the Task Planner module and passed back to the Supervisor module again. This procedure is repeated recursively until all the steps in the high-level task plan have been executed.

The Supervisor module in the SASIR system consists of three components: event scheduler, event monitor and interrupt handler, as shown in Fig. 5. The Supervisor module schedules (activates) each individual module, and controls the overall flow of operation. Such a flow of operation is controlled in the following fashion (shown in Fig. 6): The Supervisor module first fetches the high-level user-specified task plan from the Task Planner module. Next it examines each step in the plan. If the step is a primitive plan, then the Supervisor module will pass the plan to either the Perception or Motor module for execution. Otherwise, if the plan is a complex plan, the Supervisor module will send it back to the Task Planner module, and the sub-plans of that complex plan will be generated by the Task Planner module and passed back to the Supervisor module again. The above procedure is repeated recursively until all the steps in the high-level task plan have been executed. The Supervisor also handles the interrupts generated by the Perception and Motor modules. In such cases the Supervisor allows user input of commands or the specification of certain options.

As mentioned earlier, in SASIR, meeting a hard deadline was not a major concern in the design. Each subtask is allowed having a sufficient time-slot to finish. In certain time-critical situations which demand timely response, it is handled by hardware interrupt and software interrupt service routines. Examples include using force/torque sensor and audio sensor to generate hardware interrupt which can stop robot motion immediately. This way, our planner can still achieve timely performance by on-line response to the changes in the dynamic environment.

Note that the SASIR system allows the user to specify the starting task plan at any level. This capability is very useful especially during the developing phase of a system, where low-level or intermediate-level task plans can be examined and tested individually even before the high-level task plans have been constructed. In addition, this capability is quite important in semiautonomous operations involving human-machine systems.

The Knowledge Base in the SASIR system is organized in two compartments. They are: i) long-term memory (LTM),
in which information about the robot, sensors, and workspace, along with high-level user-specified task plans and general purpose low-level task plans are stored, and ii) short-term memory (STM), in which the current status of the objects, robot, and environment are stored. Notice that while LTM is associated with static information which does not change during the robotic operation, STM is associated with dynamic information which reflects the changes in the system status.

Sensory information consists of the 3-D locations of the sensors in the SASIR system. For each sensor a coordinate frame is defined. A vector containing translation values and rotation angles is stored in the Knowledge Base to define the position of that coordinate frame with respect to the end-effector coordinate frame or robot coordinate frame. Information regarding other objects within the robot workspace is stored in the Knowledge Base in a similar fashion. In addition, the sensor's geometric and calibration parameters such as camera focal length, and the image to world transformation, are stored in the Knowledge Base.

A task plan, contained in the Knowledge Base, is stored using a format defined in the previous section. In addition to LTM and STM, there are two more components in the Knowledge Base: memory acquisition and maintenance. The functions of these two components are acquiring knowledge from the user, and modifying and updating the Knowledge Base during the operation of the robot. Fig. 7 shows the block diagram of the Knowledge Base module.

For most practical robotic systems, it is desirable to have a smooth and efficient mechanism for operator-robot interaction. In some applications, it may be necessary to provide dynamically reconfigurable levels of autonomy to the intelligent system. To one extreme the operator may want to perform a task telerobotically or manually (with no autonomy). On the other extreme, the system may function totally under its own control. The User Interface module is necessary to provide such capabilities to an intelligent system. It should allow for appropriate interrupt handling, error detection and recovery, monitoring of robotic actions, and operator override features.

The User Interface module accepts the human commands, outputs operation status upon the operator's requests, and relays the control to the operator, thus giving the SASIR system a man-in-the-loop option. Fig. 8 shows the block diagram of the User Interface module. The Program I/O component handles the direct input and output during the autonomous mode of operation. The Teach Mode component relays the control to the operator at request. In case of failure or an interrupt which requires operator's assistance, the Manual Failure/Interrupt Handling component allows the operator to control the robot and the sensory devices. This can be accomplished in two ways: 1) the operator manually inputs the specific motion command to move the robot or sensory data acquisition command to obtain the data; or 2) the operator manually selects and organizes a suitable set of plans which are available in the Knowledge Base. The Operation Sequence Recorder then records the steps performed by the operator and converts these steps into the proper form of the task plans if necessary. It also updates the Knowledge Base for the new plans generated, along with the changes in robot position and environment which may have occurred during the manual mode.

VI. IMPLEMENTATION OF INTEGRATED ROBOTIC SYSTEMS

A sensor based intelligent robot is a complex system, and its implementation requires a careful examination of many issues regarding the various components of the system. In the following the utility of the SASIR system in the design of intelligent robotic systems will be illustrated. The first system is capable of autonomous detection of a chemical spill, its localization in the world coordinates, detection, localization and engagement of a vacuum cleaner attachment, cleaning up the spill by using the vacuum cleaner attachment, and verification of the clean-up operation. This will be referred to as the Autonomous Spill Cleaning (ASC) Robotic System. The second system is capable of a range of autonomous inspection and manipulation tasks involved with a control panel testbed. The systems were designed as integrated systems and are capable of online execution of a wide variety of autonomous inspection and manipulation tasks. These two systems are extensively tested and currently operational in the Computer Vision and Robotics Research Laboratory. Simulation and animation techniques were employed, in addition to the real-world testing, during the system development to increase efficiency and reliability of the software, and to ensure the operation safety of the robot. The generality and robustness of the system performance were verified by successfully transporting some of the system components to another research laboratory involving a different type of robot, sensors, and physical environment.

A. Autonomous Spill Cleaning (ASC) Robotic System

The autonomous spill cleaning (ASC) robotic system consists of six modules: Perception, Motor, Task Planner, Knowledge Base, Supervisor, and User
Interface, as shown previously in Fig. 1. The following is a high-level, user-specified task plan stored in the Knowledge Base module for a specific spill clean-up task. (For clarity, the plan is not in the exact form of the template described earlier. If, then, else, go to statements can be implemented using success/failure/interrupt servers.) The way in which this high-level task plan is utilized by the ASC robotic system to accomplish the spill clean-up operation will be shown later.

**plan name:** Spill Cleaning

1. Find spill. If spill found, then go to Step 2; otherwise terminate.
2. Tool acquisition.
3. Generate manipulator trajectory for clean-up.
4. Perform clean-up.
5. Replace tool.
6. Verify clean-up. If spill found, go to Step 2; otherwise terminate.

The plan is implemented using a frame-like structure where the steps constituting the plan can themselves be described as plans. The plans which cause direct action of the Perception or the Motor modules are primitive plans, and the plans which contain sub-plans are complex plans. Using the Spill Cleaning plan, the spill clean-up task can be carried out in the following fashion: The Supervisor module fetches the Spill Cleaning plan from the Knowledge Base module. Next it examines each step of the plan. If the step is a primitive plan, then the Supervisor module will pass the plan to either Perception or Motor module for its execution. Otherwise, if the plan is a complex plan, the Supervisor module will send it to the Task Planner module, and the sub-plans of that complex plan will be generated and instantiated by the Task Planner module and passed back to the Supervisor module. The above procedure is repeated recursively until all the steps in the spill cleaning plan have been executed.

1) Automatic Plan Generation in the ASC Robotic System: Recall that in the Autonomous Spill Cleaning system, the operation is carried out by instantiating and executing the plans stored in the Knowledge Base. Utilizing the automatic planning technique presented in the Section IV, the following shows that how the Task Planner generates some of the sub-plans given the following user-specified, high-level plan for a spill cleaning task: The system, by executing this plan, essentially performs the cycle of finding the spill, picking up the vacuum nozzle, cleaning the spill, and replacing the tool until no spill is found or the operation is interrupted. As mentioned earlier, some processes such as vacuuming are domain-dependent, thus the process Manipulate-Object selects the plan for vacuuming in the forms of rules. The following is an example:

```
Process Manipulate-Object(Procedure, Object, Device)
If Procedure = Vacuum and Object = Spill then
    Plan = Vac.Spill; Return;
end if
```

where the plan to clean-up the detected spill is (see top of next page):

The entire set of plans (names only), assuming no errors or interrupts occur, is given as follows for the first phase (finding spill and cleaning), and the second phase (verifying) of the spill cleaning task (see Spill Cleaning Operation Sequence on the page 584):

Notice that in the second phase, the initialization and detection of the location of vacuum nozzle plans (steps 13–22 in Phase 1) are not generated since those preconditions are already satisfied as results of the first phase operation. Thus for the same task, the decomposition of the plan is different; this is an example of non-static plan decomposition.

2) Experimental Verification of the ASC Robotic System In order to verify the robustness of the ASC robotic system, numerous experiments were performed at two different laboratories [35]. The experimental results indicate that the system is capable of robust performance. This section presents a complete sequence of ASC robotic system operations to illustrate its performance in laboratory environments.

In performing the spill clean-up task, the ASC robotic system starts with an initialization of the robot and sensors, as well as all the software modules. After reading the input from the operator specifying the starting plan, in this case the Spill.Cleanup plan shown earlier, the system is now in its autonomous mode. Following the procedure of decomposition and generation of the plans, and the execution and control cycle of the system described in the previous sections, the first subplan generated and executed by the system is the Locate.Object(Spill, 3D.Location) plan. The first primitive plan is moving the camera, shown in Fig. 9(a), where the Motor module of the ASC robotic system moves the camera to a position where the simulated spill is within the field of view. Currently, this camera position is predefined and stored.

Next, the image of the spill is acquired by the Perception module. The Perception module computes the histogram of the image and performs a histogram analysis based on a minimum error method to select an optimal threshold value for image binarization. The image is then segmented (binarized) using this optimal threshold and the contour of the segmented image is then traced. Complication arises in some test images due to the grid pattern of the floor. To remove these patterns the "erode-and-grow" operators are applied to the segmented image.

The image to world transformation procedure is performed after the Perception module has detected the spills. This procedure computes the actual spill location in the robot coordinate frame using the contour points of the spill regions in the image. The 3-D location and the area of the spill in robot and room reference frames are calculated based on the contour points. Once the spill location has been determined, the Motor module performs the clean-up operation. First the location of the vacuum cleaner nozzle needs to be detected. The detection of the tool is accomplished by using stereo image analysis technique [36]. Fig. 9(b) shows the robot taking the stereo image pair of the tool by placing the camera at two distinct locations, and the two images are processed to find the 3D location of the tool. The z component of the tool location
Plan: Spill.Cleanup
Preconditions: Part.Status(Robot, Initialization)
Part.Status(All.Sensors, On)
Select.Strategy(Place.Camera, Camera, Estimate.Position)
Action List: Locate.Object(Spill, 3D.Location)
Pickup(Vacuum.Nozzle)
Manipulate.Object(Vacuum, Spill, Vacuum.Nozzle)
Putdown(Vacuum.Nozzle)
Add List: Part.Location(Vacuum.Nozzle, LastPosition, Reference.Frame)
Part.Status(Robot, Initialization)
Part.Status(All.Sensors, On)
Delete List: Part.Location(Vacuum.Nozzle, Old.Position, Reference.Frame)

Plan: Vac.Spill(Spill, 3D.Location)
Preconditions: Holding(Gripper, Vacuum.Nozzle)
and Part.Location(Spill, 3D.Location, Reference.Frame)
Status.Control(Vacuum.Cleaner, On)
Move.Tool(Vacuum.Nozzle, Vac.Path, Reference.Frame)
Status.Control(Vacuum.Cleaner, Off)
Add List: Part.Location(Vacuum.Nozzle, LastPosition, Reference.Frame)
Delete List: Part.Location(Vacuum.Nozzle, Old.Position, Reference.Frame)

is verified by taking the measurement of the ultrasonic range sensor, shown in Fig. 9(c). Once the tool location has been determined, the gripper picks up the vacuum cleaner nozzle, as shown in Fig. 9(d), and in Fig. 9(e) the solid spill is being cleaned. Liquid spills were also used in the experiments. Both types of spills were successfully cleaned by the ASC robotic system. Finally, Fig. 9(f) shows the system executing tool replacement plan. The above procedure is repeated in the spill verifying phase, with an exception of tool localization step, since the location of the tool is already stored in the Knowledge Base. Therefore the Task Planner needs to generate only the plans for the gripper directly picking up the tool. Fig. 10 shows the performance of spill-cleanup by a mobile robot (HERMIES at the Oak Ridge National Laboratory) with integration of Perception module of ASC.

Several images were used to test the accuracy and consistency of image to world mapping assuming that the eye-to-hand calibration has already been performed. Thus the analysis shown below indicates the accuracy and consistency of the mapping in a relative manner rather than in an absolute sense. Five images were taken using a camera mounted on a tripod whose rotation and orientation were carefully measured. The distance from the camera to the floor ranged from 1.5 to 1.9 meters, and the rotation angle, $\beta$, for the camera ranged from 0 to 0.8 radians. The deviation of computed spill locations from the actual locations was within $\pm 0.04$ meter. Due to the limitation of the image resolution ($128 \times 128$ pixels used in these experiments), the accuracy of the calculated spill locations degraded as the spill appeared further away from the camera. Thus if the accuracy of the spill location is calculated in terms of the deviation of the computed location from the actual one with respect to the distance between the camera and the spill, then it is within $\pm 2.7\%$.

B. ROBOSIGHT: A Robotic System for Inspection and Manipulation

In this section, the design and implementation of an integrated robotic system for performing a variety of inspection and manipulation tasks is discussed. The discussion begins with a description of the test-bed utilized in the development. The main focus of this research is on the development of an autonomous system capable of performing various inspection and manipulation tasks associated with a typical control panel. This panel is designed in consultation with experts from the nuclear industry, using only "off-the-shelf" components. The tasks range from reading various meters and displays to...
Spill Cleaning Operation Sequence

Phase 1:
SpillCleanup
  01) Part.Status(Robot, Initialization)
  02) Part.Status(Camera, On)
  03) Part.Status(Range, On)
  04) Part.Status(Proximity, On)
  05) Part.Status(Sound, On)
  06) Select.Strategy(Place_Camera, Camera, Estimate_Position)
  07) Locate_Object(Spill, 3D_Location)
  08) Select.Strategy(Locate_Spill, Camera, SegmentationProcedure)
  09) Move.Sensor(Camera, Estimate_Position, Robot)
 10) Acquire.Sensory.Data(Camera, Image)
 12) Pickup(VacuumNozzle)
 13) Locate_Object(VacuumNozzle, 3D_Location)
 14) Select.Strategy(Locate_Vac, Camera, SegmentationProcedure)
 15) Move.Sensor(Camera, Estimate_Position, Robot)
 16) Acquire.Sensory.Data(Camera, Image)
 17) Move.Sensor(Camera, Estimate_Position, Robot)
 18) Acquire.Sensory.Data(Camera, Image)
 19) Process.Sensory.Data(Locate_Vac, SegmentationProcedure, Image1, Image2)
 20) Move.Sensor(Range, Vac_Position, Robot)
 21) Acquire.Sensory.Data(Range, Range_Reading)
 22) Process.Sensory.Data(Locate_Vac, VerifyProcedure, Range_Reading)
 23) Status.Control(Gripper, Open)
 24) Move.Gripper(Gripper, Vac_Position, Reference_Frame)
 25) Status.Control(Gripper, Close)
 26) Vac.Spill(Spill, 3D_Location)
 27) Select.Strategy(Clean_Spill, VacuumNozzle, Vac_Path)
 28) Status.Control(VacuumCleaner, On)
 29) Move.Tool(VacuumNozzle, Vac_Path, Reference_Frame)
 30) Status.Control(VacuumCleaner, Off)
 31) Putdown(VacuumNozzle)
 32) Move.Gripper(Gripper, Vac_Position, Reference_Frame)
 33) Status.Control(Gripper, Open)

Phase 2:
SpillCleanup
  01) Select.Strategy(Place_Camera, Camera, Estimate_Position)
  02) Locate_Object(Spill, 3D_Location)
  03) Select.Strategy(Locate_Spill, Camera, SegmentationProcedure)
  04) Move.Sensor(Camera, Estimate_Position, Robot)
  05) Acquire.Sensory.Data(Camera, Image)
  06) Process.Sensory.Data(Locate_Spill, SegmentationProcedure, Image)
  07) Pickup(VacuumNozzle)
  08) Status.Control(Gripper, Open)
  09) Move.Gripper(Gripper, Vac_Position, Reference_Frame)
 10) Status.Control(Gripper, Close)
 11) Vac.Spill(Spill, 3D_Location)
 12) Select.Strategy(Clean_Spill, VacuumNozzle, Vac_Path)
 13) Status.Control(VacuumCleaner, On)
 14) Move.Tool(VacuumNozzle, Vac_Path, Reference_Frame)
 15) Status.Control(VacuumCleaner, Off)
 16) Putdown(VacuumNozzle)
 17) Move.Gripper(Gripper, Vac_Position, Reference_Frame)
 18) Status.Control(Gripper, Open)
Fig. 9. The ASC robotic system operation in the laboratory. The robotic system is shown during: (a) execution of the Find Spill plan, (b) execution of the Stereo Image Acquisition plan, (c) execution of the Range Reading plan, (d) execution of the Pickup Tool plan, (e) execution of the Vacuum Spill plan, and (f) execution of the Replace Tool plan.

operating different types of switches and control devices. Also included are tasks associated with valve manipulation. Tele-operation or automatic operation of valves in nuclear power plants is recognized as one of the important desired capabilities of robotic systems.

The control panel and the robot arm are shown in Fig. 11. Typical autonomous robot operation will involve the following. First, the robot identifies the exact geometrical position of the panel using a camera calibration program. Next, it uses a computer vision system to develop an object location layout map for various devices appearing in the panel. The task to be performed by the robot is specified by a code displayed on a LCD meter. After decoding the command, the robot performs the requested inspection or manipulation task. The robot can, for example, acquire an image of the analog-meter, and then perform automatic determination of the needle position. For more details associated with this vision system, interested readers may refer to references [37], [38]. Issues associated with systematic utilization of the perceptual signals (such as vision in this specific case) into the overall task planning, control and execution of the selected robotic actions will be addressed in the next section.

1) Automatic Plan Generation in the ROBOSIGHT System: The generation of plans in the ROBOSIGHT system by the Task Planner follows a similar procedure described in the previous section. To be brief, we will show the initial user-specified plan and the final plan list for a typical operation run (see next page).
The ROBOSIGHT system, after the panel and the designated LCD digital meter containing the command code have been detected and located, performs the cycle of reading the LCD, decoding the command, and executing the command accordingly, until the termination command is displayed. In addition to the top most plan for inspection and manipulation, a plan which carries out the above operation cycle is defined as follows (see bottom of next page):

The following is a typical operation expressed as a set of plans (names only), assuming no errors or interrupts occur (see Panel Inspection and Manipulation Operation Sequence, page 588).

The above operation consists of reading the LCD, reading the analog meter, taking a image of the valve and finding the status of the valve, and finally picking up the tool and turning the valve.

2) Experimental Verification of the ROBOSIGHT System: The performance of the ROBOSIGHT system was tested using the test-bed described earlier. The system’s capability to perform a variety of inspection and manipulation tasks has been verified. As mentioned in the previous sections, the system starts its operation by first locating the panel. This is accomplished by placing the camera, detecting the lights on the corner of the panel, and using camera calibration and 2-D to 3-D transformation techniques to determine the panel pose with respect to the robot coordinates.

Next, the system positions the camera to take the images of left and right portions of the panel. Image segmentation is based upon the region growing procedure and object recognition is based upon the graph matching algorithm described in [23]. The system then enters the inspection and manipulation cycle. It is achieved by reading and decoding the LCD digital meter, and performing the command decoded...
accordingly, in a cyclic fashion. Edge detection and thinning routines were employed to obtain the one-pixel wide edge maps of the numeric code. The edge maps were later processed using Fourier shape descriptors to identify the numeric code accurately.

After recognizing the number displayed, the system decodes it as a command for performing a certain task. Examples of inspection tasks include "reading" of various types of analog meters, localization of 3-D position of a tool, and recognition of the status of the tool. Examples of manipulation tasks include grasping a tool, turning a valve, pushing a light switch, and moving the handle of a slider.

C. Simulation of Sensor-Driven Robots

Sensor-driven robots rely on information acquired from their sensory devices to accomplish their assigned tasks. The flow of operation and system behavior of such a robot is influenced by its perception of the world through the sensors.

There are two basic programming modes in developing control programs for such robots: on-line programming and off-line programming [39], [40]. The main difference between the two is that off-line programming involves generating and testing robot control programs in a simulated robotic environment without actually running the physical robot. Very often, a 3-D geometric model of the robotic environment is created using a graphic simulation and animation package to help the programmer visualize the motion sequences of the robot. While off-line programming has several advantages such as increased efficiency of program development and reliability of the control program, ensured safety of robot operation, and reduced cost of new software development, most of the conventional computer-graphics-based robot simulation and animation software packages do not incorporate sensory data input. The lack of robot sensing simulation has put restrictions on utilizing off-line programming on sensor-driven robots. Thus, creating a more realistic simulation and animation
software for sensor-driven robots has emerged as one of the important research issues in the robotics and automation field [39].

In this section, an implementation of a novel design of a simulator for sensor-driven robots is briefly described. The simulator is integrated within an entire robotic system, and is utilized for automatic robot programming, i.e., the flow of system operation (both perceptual and motor actions) is controlled by the task planning and supervisor modules of the system (see Fig. 2). The unique features of the simulator are that 1) it simulates and utilizes sensory information feedback, 2) it integrates planning and simulation, and 3) it makes the switching between real and simulation modes in the robotic system transparent to the user. By designing such a simulator for SASIR, the conclusion is that the simulator in an integrated sensor-driven robotic system must incorporate simulation of sensory information feedback. The system is implemented on a Silicon Graphics IRIS workstation, and its modules include Model Data Base, 4D Graphics Generator, User Interface, and parts of Model and Sensor Information Monitor, Sensory Data Panel Inspection and Manipulation Operation Sequence

<table>
<thead>
<tr>
<th>Inspection_Manipulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
</tr>
<tr>
<td>02</td>
</tr>
<tr>
<td>03</td>
</tr>
<tr>
<td>04</td>
</tr>
<tr>
<td>05</td>
</tr>
<tr>
<td>06</td>
</tr>
<tr>
<td>07</td>
</tr>
<tr>
<td>08</td>
</tr>
<tr>
<td>09</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>26</td>
</tr>
<tr>
<td>27</td>
</tr>
<tr>
<td>28</td>
</tr>
<tr>
<td>29</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>31</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>33</td>
</tr>
<tr>
<td>34</td>
</tr>
<tr>
<td>35</td>
</tr>
<tr>
<td>36</td>
</tr>
<tr>
<td>37</td>
</tr>
<tr>
<td>38</td>
</tr>
<tr>
<td>39</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>41</td>
</tr>
<tr>
<td>42</td>
</tr>
<tr>
<td>43</td>
</tr>
</tbody>
</table>
Generator, and Action Script Generator and Event Controller. For more details please refer to [25], [41], [42].

Integrating Planning and Simulation is a very important feature of an autonomous system [43], [44]. Simulators are quite useful in their ability to derive the model of a workspace and robot efficiently. It is quite desirable to blend this powerful description (or representation) of the robot’s environment with the logical and reasoning capabilities that an autonomous system should possess. This allows for developing systems which can be tested for safe and realizable operation on the forward simulation mode. Only if this performance is satisfactory is the system allowed to perform the actual task [45], [25], [24], [44].

The Task Planner, given the initial and goal states of the tasks, generates a set of plans which will achieve the goal. These plans are transformed into Action Script format, and are executed by the simulator. An application for automatic task planning is panel inspection and manipulation [37], [43]. The generated plans include 1) panel pose determination, 2) object recognition and localization for valves, meters, knobs, a slider and a tool, and 3) manipulating the valves, knobs and slider using the tool.

VII. SUMMARY AND CONCLUSIONS

In this paper a System Architecture for Sensor-based Intelligent Robots (SASIR) has been introduced. The system architecture consists of Perception, Motor, Task Planner, Knowledge-Base, User Interface and Supervisor modules. Given the complexity of an intelligent robotic system, the design considerations are: high level of autonomy, automatic task planning and sensor-driven capabilities, integration of sensing and actions, and the software system’s generality, modularity, and portability. This is accomplished by utilizing a design methodology and system architecture where differences in the types of sensors, robot hardware, work environment and task requirements can be systematically incorporated. A powerful robotic system architecture can be implemented using Frames for representing the models of the environment and the robot as well as for describing the actions required for the execution of a specific task. Frames provide a powerful and flexible structure for representation and manipulation of the world model along with the sensor derived information.

Using the frame-based structure, an automatic task planner, PRISM, was presented. PRISM integrates the sensing and motor actions in task planning and control of the SASIR system. The closely coupled Perception and Motor functionalities are crucial for the robot to operate successfully in a dynamically changing environment such as inspection and manipulation task domain. PRISM supports the following list of desirable characteristics the planning and control functions of a sensor-based intelligent robotic system should possess: integrating sensing and action, hierarchical planning, abilities for replanning, error recovery, interrupt servicing, conditional and iterative planning, and learning. Another useful feature is that the SASIR system allows the user to specify the starting task plan at any level. This capability is very important especially during the developing phase of a system, where low-level or intermediate-level task plans can be examined and tested individually even before the high-level task plans have been constructed. In addition, this capability is quite useful in semiautonomous operations involving human-machine systems.

The utility of the SASIR system in the design of intelligent robotic systems was illustrated by implementations of two working systems. The first system, the Autonomous Spill Cleaning (ASC) Robotic System, is capable of autonomous detection of a chemical spill, localization of the spill in the world coordinates, cleaning the spill by using a vacuum cleaner attachment, and verification of the clean-up operation. The second, ROBOSIGHT, is capable of a range of autonomous inspection and manipulation tasks involved with a control panel testbed. These systems are extensively tested, and currently operational in the CVRR laboratory. The generality and robustness of the systems performance was verified by successfully transporting system components to another research laboratory involving a different type of robot, different sensors, and a different physical environment. The simulation software helps the user visualize the motion and reaction of the sensor-driven robot under his control program, thereby increasing the efficiency of program development and the reliability of the control program, ensuring the safety of robot operation, and reducing the cost of new software development. This is achieved by 1) simulating and utilizing sensory information feedback, 2) integrating planning and simulation, and 3) making the switching between real and simulation modes in the robotic system transparent to the user.

In conclusion, the constraints of real-time performance, high-level autonomy, sensor-driven capability, and software system flexibility of the working system motivate resolutions of several important issues in developing intelligent robots: 1) a truly robust system can only be obtained through a carefully designed system architecture and systematically coordinated (integrated) modules performing in concert within the architecture, 2) automatic task planning with integrated sensing and action is critical for a robotic system to be capable of performing complex tasks in highly unstructured and dynamic work environments, and 3) integration of plan generation and execution is a crucial feature for a task planner operating in real-world situations in which replanning, error recovery, and interrupt servicing must be considered.

ACKNOWLEDGMENT

This research was supported by the DOE’s University Program in Robotics for Advanced Reactors (Universities of Florida, Michigan, Tennessee, Texas, and the Oak Ridge National Laboratory) under grant No. DOE DE-FG02-86NE37968, and by the National Science Foundation under grant No. NSF-INT-9019147. We appreciate the assistance and cooperation of the CESAR group of the Oak Ridge National Laboratory. Assistance of our colleagues from the Computer Vision and Robotics research laboratory is also appreciated. Finally, we thank the anonymous reviewers for offering several insightful comments.
Mohan M. Trivedi (S'76-M'79-SM'86) was born in Wardha, India, on October 4, 1953. He received the B.E. (Honors) in electronics from the Birla Institute of Technology and Science, Pilani, India, in 1974 and the M.E. and Ph.D. in electrical engineering from Utah State University, Logan, UT, in 1976 and 1979, respectively. At Utah State he received a Graduate Research Scholarship.

He is a Professor in the Electrical and Computer Engineering Department of the University of Tennessee, Knoxville, where he has established the Computer Vision and Robotics Research Laboratory. He and his team are engaged in a broad range of sponsored research studies in sensor-based mobile manipulators, active perception and machine vision, interactive graphical interfaces for human-machine systems, and remote sensing. His team's recent research involvements and accomplishments include a novel simulation and interactive control environment for sensor-based virtual and real robots; efficient means for extraction and integration of multiple depth cues; development of cooperating mobile robot teams; computational models for texture analysis and synthesis; and integrated robotic systems for applications in nuclear environments and for inspection and maintenance of high-speed railway tracks. He has published extensively and has edited over a dozen volumes including books, special issues, video presentations, and conference proceedings. He serves as the Editor-in-Chief of Machine Vision and Applications, the official journal of the International Association of Pattern Recognition. He serves as an associate editor of the journals: IEEE Transactions on Systems, Man, and Cybernetics, Pattern Recognition, Int. J. of Applied Intelligence, Int. J. of Approximate Reasoning, and the Optical Engineering Reports. He served as the Chairman of the Robotics Technical Committee of the Computer Society of the IEEE (1987–1989). Currently, he chairs the Pattern Recognition, Image Processing and Computer Vision Committee of the IEEE Systems, Man, and Cybernetics Society. He has served as the General Chair for a number of major conferences in the artificial intelligence and robotics fields.

Dr. Trivedi is a frequent consultant to various national and international industry and government agencies. He has established collaborative relationships with academic, industrial, and government institutes from Japan, France, Denmark, India, Australia, Singapore, and New Zealand. He regularly offers tutorials (both "in-class" and satellite TV-based) dealing with his specialty areas for professional groups in the U.S. and abroad. Dr. Trivedi is a recipient of the Pioneer Award (Technical Activities) and the Meritorious Service Award of the IEEE Computer Society and the Distinguished Alumnus Award from the Utah State University. He is a Fellow of the International Society for Optical Engineering (SPIE). He has been elected to the honor societies of Phi Kappa Phi, Tau Beta Pi, and Sigma Xi. He is listed in the Who's Who in the Frontiers of Science and Technology, Who's Who in American Education, American Men and Women of Science, Who's Who in the World and other similar publications.