Robotics and Autonomous Systems 10 (1992) 161–184
Elsevier

On the design and development of a human–robot synergistic system

François G. Pin, Lynne E. Parker and Fred W. DePiero

*Oak Ridge National Laboratory *, Center for Engineering Systems Advanced Research, P.O. Box 2008, Building 6025, MS-6364, Oak Ridge, TN 37831-6364, USA*

Abstract

Pin, F.G., Parker, L.E. and DePiero, F.W., On the design and development of a human–robot synergistic system, Robotics and Autonomous Systems, 10 (1992) 161–184.

A great variety of robotic systems have been developed over the past decades, varying from fully human-controlled teleoperated systems, to fully motion-programmed, industrial-type devices, to sensor-equipped 'intelligent' machines with autonomous perception and reasoning capabilities. All these systems, however, have very static roles in supporting humans, in the sense that their capabilities are limited to very narrow functions or task domains and they cannot really adapt to changing environments where entirely new tasks may be necessary. By considering these systems as part of overall human–machine systems, interesting adaptation capabilities can be envisioned for the machine through the sharing of control and experiences, and observation of the human actions. This paper proposes a general conceptual architecture for enhancing the synergy between humans and machines in such cooperative systems, and investigates an implementation of the concept to the synergistic integration of teleoperative and autonomous operation of a manipulator arm for the execution of sequential tasks. The proposed architecture reflects a unique blend of many disciplines of artificial or machine intelligence and the functioning of its major modules is discussed. The paper then focuses on initial implementations of the Job Planner, Task Allocator, and Execution Monitor modules to discuss simple proof-of-principle experiments illustrating the coalescence of these modules needed to achieve skill-based dynamic task planning, allocation, and execution.

Keywords: Shared control; Human–robot synergy; Job planning; Task allocation; Execution monitoring; Learning by observation; Human–machine interface; Sensor-based robotics.

1. Introduction

During the last few decades, the growing awareness and resulting need to maximize the safety of humans while performing necessary tasks in chemical, radioactive or other hazardous environments (e.g., battlefield, undersea operations, fire fighting, etc.) have led to the design of a great variety of automated and robotic machines. In this context, research and development have taken place on a broad range of technologies varying from fully remote-controlled systems, such as teleoperated and servo-manipulators, to fully

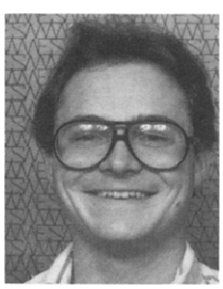
* Managed by Martin Marietta Energy Systems, Inc. for the U.S. Department of Energy under contract No. DE-AC05-84OR21400.

François G. Pin is the Group Leader of the Autonomous Robotic Systems Group at the Oak Ridge National Laboratory and a principle investigator of the Center for Engineering Systems Advanced Research (CESAR) Program. He earned the Maîtrise de Mécanique from the Université de Nancy 1, France, and the Diplôme National d'Ingénieur Electromécanicien from the Ecole Nationale Supérieure d'Electricité et Mécanique de Nancy, France. His M.S. and Ph.D. in Mechanical Engineering and Aerospace Sciences are from the University of Rochester, New York. He joined the staff of the Oak Ridge National Laboratory in 1982, at which time he conducted the major part of his research in mathematical modeling and numerical analysis. His current research work and interests include Methodologies for Intelligent Machines and, in particular, the Planning, Reasoning, Learning, and Decision Making of Autonomous Mobile Robots, Manipulation Systems, and Human–Machine Synergistic Systems. He has authored more than 100 publications and is on the Editorial Board of several international journals.

programmed robotic devices, such as industrial robots, or to fully autonomous machines involving artificial intelligence, machine perception, and advanced control. Each of these concepts incorporates proven advantages, but also typically exhibits significant drawbacks in practical applications. The best currently available master/slave manipulator systems, for example, while able to perform very complex tasks remotely, reduce work efficiency by factors of five to eight. In outdoor environments, teleoperation is currently limited by line-of-sight transmissions or the length of cumbersome optic fiber cables. In all cases, remotely controlled systems fully monopolize human operators at the remote control station and lead to rapidly fatiguing operators with resulting further degradations in performance. On the other hand, fully programmed robotic devices, such as industrial robots, while quite suited for removing humans from the workspace in the accomplishment of routine tasks, are by definition, oblivious to unexpected (i.e., unprogrammed) events and, when a slight change occurs in their rigidly structured domain, it typically results in major damage to or destruction of the robot or the environment. Improvements in this area have recently been accomplished through significant progress in sensor-based reasoning methodologies, providing increased levels of autonomy to robotic systems. However, while some of these systems have been shown to exhibit promising capabilities in coping with unexpected events and unstructured environments, the task domains in which their autonomous behavior can be considered as robust are still extremely limited. More importantly, these systems are still fully *static* from the point of view of their overall performance capability, in the sense that both the type of tasks they can handle and the reasoning flexibility they have in performing these tasks are strictly bounded by the domain knowledge and inferencing capabilities which they have been given a priori. In other words, the expected role of the autonomous modules in an overall human–machine system remains the same over time unless additional or modified knowledge is directly input into the machine, generally through very burdensome programming by a specialized machine technician. Although not critical under normal (i.e., as expected) operations, this fixed and static role becomes a serious drawback as soon as 'off-normal' conditions appear during operation of the overall human–machine system. This occurs not only when the task or environmental conditions unexpectedly change with time, but also when a component (e.g., a sensor, an actuator) fails in the system, canceling an autonomous capability of the machine, or even when a change in human operator takes place with the new operator not knowledgeable of given procedures, roles, or capabilities which himself and/or the machine are expected to exhibit. In those instances, even when direct human intervention is possible to temporarily bypass and supplement the current deficiency (i.e., by force-fitting into the system a change of respective roles), the overall operator–machine system will repeatedly fail in the accomplishment of the required tasks until some input is made to the system (i.e., to the machine or to the human). Such input could be the repaired component, or information about the modified roles expected of the human or the machine and the necessary associated knowledge.

Since the ultimate goal of the development of 'intelligent' machines is to increasingly support humans in the efficient accomplishment of increasingly complex tasks, the keys to improving the performance of autonomous modules in hu-

Lynne E. Parker is currently a Ph.D. student in the Artificial Intelligence Laboratory at the Massachusetts Institute of Technology, performing research in cooperative autonomous agents. In 1988 she received the M.S. degree in Computer Science from the University of Tennessee, with a thesis topic of mobile robot navigation. From 1983 to 1986, Ms. Parker was a systems analyst with Martin Marietta Energy Systems, and from 1986 to 1989, she was a research associate at Oak Ridge National Laboratory, performing research in Human–Machine cooperation. Ms. Parker's research interests include cooperative agents, mobile robotics, situated agent architectures, and robot learning.



Fred W. DePiero is a development engineer in the Robotics and Process Systems Division of Oak Ridge National Laboratory. In addition to his work in Human–Robot Symbiosis, Fred has also worked in the areas of autonomous robotics and Image Processing. This includes work with redundant manipulation, image compression, vision guided docking, and remote surface mapping. Fred received his BS. and M.S. degrees in Electrical Engineering from Michigan State University. He is currently pursuing a Ph.D. at the University of Tennessee.

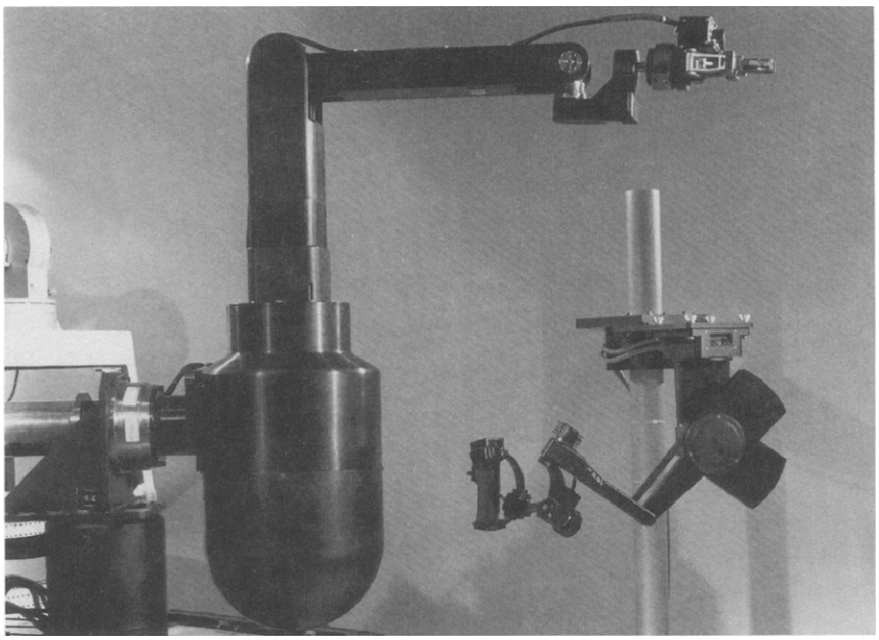


Fig. 1. The CESARm redundant manipulator testbed.

man–machine systems seem to lay with providing the system with capabilities to (1) acquire new knowledge over time via experience and observation of human actions to increase its admissible task domain, (2) evaluate current capabilities of the overall system based on the current state of knowledge and corresponding skills of the system's components (humans or machines), and (3) dynamically optimize the allocation of tasks (i.e., roles) between the human and the machines on the basis of the current and respective capabilities, availabilities and task requirements. In this context, work has been initiated at the Oak Ridge National Laboratory on a new class of automated systems which aim at embodying these concepts and which appear promising for improving the productivity, quality, and safety of operation of advanced robotic systems. This new type of system is referred to as 'man–machine symbiosis' and has the fundamental objective of allowing the division of work between humans and machines to change dynamically, thus bridging the gap between fully human controlled and fully 'fixed roles' types of autonomous modules, and ultimately to realize optimum cooperation between humans and machines through shared knowledge, skills, and experiences.

This paper describes a proposed general architecture for this type of systems and presents an initial application of this concept to the case of symbiotic integration of teleoperative and autonomous modes of task execution in unstructured environments. The system involves two symbiotic 'partners', a human teleoperator and an intelligent controller, both working toward controlling a single manipulator arm (see *Fig. 1*) for the execution of a series of sequential manipulation tasks. The resulting architecture reflects a unique blend of many disciplines of artificial intelligence into a working system, including job or mission planning, dynamic task allocation, man–robot communication, automated monitoring, and machine learning. The following section describes the proposed symbiont architecture and discusses the functioning of its major modules. The subsequent sections focus on the job planning and task allocation strategies and discuss simple proof-of-principle experiments illustrating the coalescence

of these modules needed to achieve dynamic task planning and allocation in the human–robot symbiont.

2. General architecture of the proposed symbiotic system

Studies on the problem of human–machine cooperation can be traced back to the mid-20th century when a series of articles [1–5] offered engineering design oriented approaches on how to allocate functions between humans and machines. In two posthumous articles, Craik [1,2] argued that for the planning, design, and operation of a complex man–machine system, all functions must be described in the same concepts, and recommended that human functions be described in mathematical terms comparable to those used in describing mechanical functions. Following this concept, a landmark article by Fitts [3] introduced the so-called 'Fitts lists' which qualitatively characterize those functions performed better by machines than by humans and those performed better by humans than by machines. Given a complex man–machine system and the functions to be performed by the system, utilization of the lists would lead to a fixed distribution of tasks to the human(s) and machine(s) involved. In spite of (or perhaps because of) their simplicity, Fitts lists or more elaborate versions of them such as those published by Chapanis [6], Edwards and Lee [7], or Swain and Guttman [8], remained regarded for more than three decades as the definite work on allocation of functions, and still constitute a valuable aid in the ergonomic and human-factor design phase of complex systems. Jordan [4] and Whitfield [5] were probably the first to point out the serious drawbacks in efficiency caused by a fixed or static allocation of tasks based on rigid, a priori determined capability characteristics. By arguing that complementarity rather than comparability is the key to optimizing the allocation of functions between humans and machines and by pointing out that over the long term, humans' preferred and most efficient performances occur when their tasks and functions are changing over time (on the basis of adaptability and need for challenge and motivation), Jordan [4] implicitly introduced two concepts which were at the basis of the architecture presented here: that of human–machine synergy (probably a term better suited than 'symbiosis' to define the relationship considered, as pointed out in a recent workshop on this topic [9]) and that of dynamic allocation of tasks based on changing skills, intellectual capabilities and performance of the human. Whitfield [5] emphasized this latter concept by recommending that predispositions toward past allocation solutions, as well as assumptions about how a function might be accomplished in the future, should be avoided in generating task allocations, effectively pointing out that optimum cooperative efficiency should be based on the *current* levels of skill, knowledge, and performance of the available cooperative 'partners'. The architecture proposed here embodies these basic concepts through cycles consisting of: task planning using current environmental conditions and execution constraints; dynamic allocation of functions based on current skill, knowledge, and performance levels; execution monitoring for detection of 'off normal' or unexpected events (triggering as necessary a replanning and reallocation); and evaluation of task performance for learning purposes (change detection, knowledge acquisition, skill improvement). In each cycle, an update of the knowledge and skill levels of the cooperative 'partners' (thereafter referred to as the symbiotic agents, or simply, agents) takes place. The two latter steps in the cycle constitute a feedback mechanism for the system and effectively drive the dynamics of task allocation. As discussed in the later sections, the characteristics of the feedback mechanism, i.e., the method for updating the skill levels, can be adjusted to lead relatively rapidly to a meaningful allocation process after the system has gone through the initial cycles of operation during which performance and skill levels can only be guessed or roughly estimated. Note that among the several types of systems including planning–allocation–execution–update cycles which have recently been proposed, very few [10,11] take into account the time dynamics of task allocation or incorporate feedback at execution time to provide for adaptability. In most applications, the iterations are taking place only to improve on the general heuristics provided by upgraded lists of the Fitts-type. A good review of such systems can be found in [12] or [13].

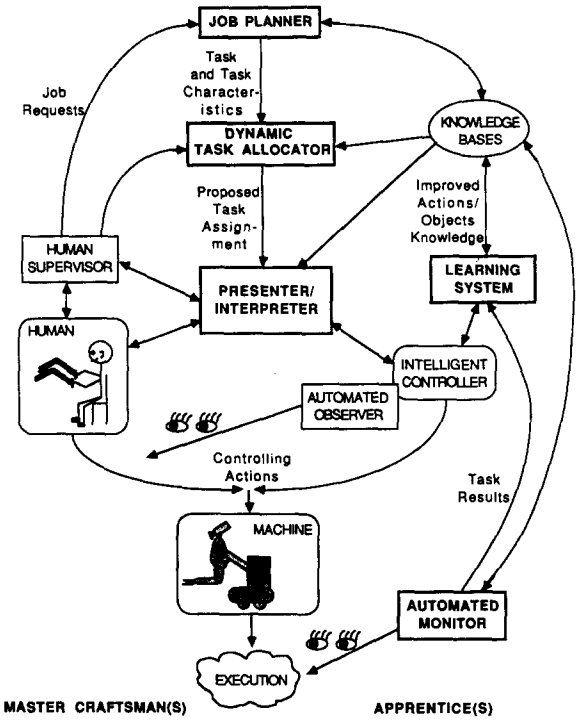


Fig. 2. Schematic of the conceptual architecture for the human–robot synergistic system.

A schematic of the proposed architecture which includes several automated modules is shown in *Fig. 2*. Here, only two agents, a 'human' and an 'intelligent controller' are shown working toward controlling a single 'machine' (a mobile manipulator), although extension to additional automated agents (several independent machines and their controllers) is straightforward by duplicating the respective modules, their interfaces with the Presenter/Interpreter and their associated characteristics in the knowledge bases. The system initiates with a request for a job to be performed, given either by another supervising system, a human supervisor, or the human agent itself. The Job Planner is responsible for decomposing the overall job to be performed (such as 'install electrical equipment') into subtasks (such as 'find wrench' or 'grasp wrench'), indicating the order in which the subtasks must be performed. The resulting job decomposition (see next section) is presented to the human partner who can approve or modify it iteratively through the Presenter/Interpreter. The plan is then passed to the Task Allocator, which further decomposes the subtasks into elemental actions, and assigns these either to the human or to the intelligent controller. The subtask decomposition and allocations are based upon information on the agents' current skills, capabilities and availabilities contained in the knowledge bases. Once they have been allocated subtasks, the human or the intelligent robot controller can send controlling actions to the manipulator arm for execution of the subtasks. When the human is executing a subtask, both the Automated Observer and the Automated Monitor are used to observe and correlate the human input to the robot and the outcome of its actions for skills and environment model updates and for task learning purposes. When the intelligent controller is sending control commands to the robot, the Automated Monitor verifies the consequences of the actions, updates the environment model and knowledge bases, and detects unexpected events, triggering replanning as appropriate.

As the schematic in *Fig. 2* indicates, the general topic of 'Man–Machine Synergy' is vast and rich in developments, encompassing many aspects of artificial intelligence, robotics, and controls. By no means have all the problems of this field been entirely solved or even defined, and developments are expected to continue over many decades, progressively adding to and refining such systems. The intent of this paper is to introduce the architecture shown in *Fig. 2*, demonstrate its feasibility as a working system in which many modules coalesce, and present in detail our recent developments for only some of its contributing modules, namely the Job Planner and Dynamic Task Allocator with their related functions in the Automated Monitor. While their functioning will also be briefly recalled here, the reader will be referred to previously published work on the remaining modules of this architecture.

3. Dynamic job planning for human–robot synergy

3.1. Planning methodology

In a human–robot synergy system, the Job Planner is responsible for planning the sequences of primitive task activity that lead to efficient job completion. Many job planning methodologies currently exist which provide various approaches

to the job planning problem (e.g., see [14–20]). The major particularity of the planning process utilized here is that it occurs in two separate levels. The first phase focusses on planning a sequence of tasks or general activities which must be performed to accomplish the job. This sequence provides a complete but succinct overview of the work to be performed and, since it is easily and rapidly generated, allows early interactions with the human supervisor who may approve and/or request changes to the plans at this level.

The second level further decomposes the tasks into elemental subtasks. The key idea behind this second level of planning is that the decomposition in elemental subtasks proceeds down to the 'smallest assignable subtasks' (SAS), i.e., the smallest tasks which can be assigned in entirety to the human or the robot. For an 'insert (object, given_loc)' task for example, the sensor-based control of one of the six degrees of freedom (3 positions, 3 orientations) of the end-effector could be considered a smallest assignable subtask, where the human could control the x and y positions using remote vision and the system could control the orientations and z position using force-torque information. A 'find (object, object_loc)' task which results from the first level of planning, would be considered a smallest assignable subtask in itself since it would be performed either by the human using remote vision or by the intelligent controller using image analysis, or knowledge base search if appropriate, and consequently would not be further decomposed in the second planning level.

Because the decomposition of a task into elemental SAS is highly dependent upon the availability, level of skill, and knowledge of the agents, and upon the availability of perception sources just prior to execution of the task (e.g., an object whose position was unknown to the machine at planning time may become known during execution of the first tasks of a job, allowing the machine to be assigned SAS involving this object in later tasks), this second level of planning takes place on-line within the Dynamic Task Allocator (discussed in a following section). In this fashion, this planning and allocation can be done using information in the knowledge bases which have been updated during execution of previous subtasks, as well as with feedback from the Automated Monitor about the current execution conditions (an object may have slipped or rolled away from its known position, an actuator or sensor may have failed, etc.). These two levels of planning thus allow both a general strategy (first level) to be rapidly generated, 'discussed', modified, and/or approved by the operator at initiation of the job, and a detailed task execution plan to be generated just prior to execution of each task, on the basis of dynamically fed back information about current conditions, availability and skills.

In this symbiotic system, the job planning approach was closely related to the definition of a language through which the modules of the symbiont communicate. This language consists of 'action–object' statements expressing relationships between a set of valid actions or tasks which can be performed in the environment and the set of objects which exist in the symbiotic world. These sets typically vary over time as new actions are learned, or as new objects appear in the world. Each of the modules of the symbiont architecture uses this language to accomplish its objective: the Job Planner plans actions to be performed on objects, the Dynamic Task Allocator assigns (action–object) SAS to the human or the robot, the Automated Observer and Automated Monitor observe the execution of actions or the states of objects, and the Learning System learns new actions or new object characteristics.

On the basis of this action–object language, the first-level job planning methodology utilizes a set of operators to generate a sequence of tasks (actions) which transforms the planning world model from an initial state to a given goal state using a depth-first search. The initial and goal states consist of modified first-order predicate calculus statements defining conditions which describe the robot's world. For example, if the Cranfield [1] Benchmark [21] which we used in the testing and proof-of-principle experiments of the various modules of the system is in a disassembled state (see *Fig. 3*) and the robot end-effector

[1] The task, called the 'Cranfield Benchmark Assembly/Disassembly', represents fundamental characteristics of practical small component manipulation tasks, and was developed as a European benchmark by the Cranfield Robotics and Automation Group at the Cranfield Institute of Technology in Bedfordshire, England.

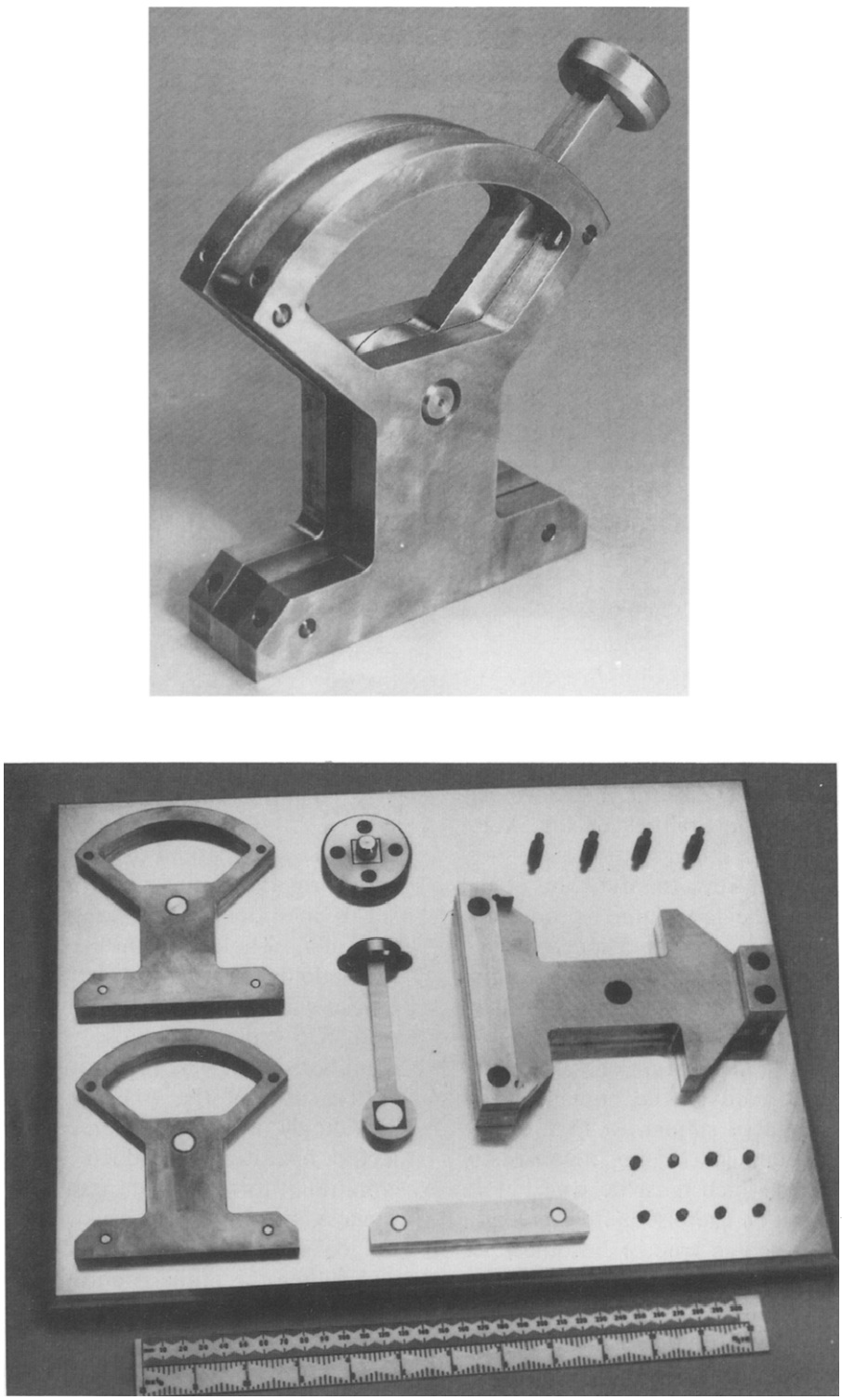


Fig. 3. The Cranfield Benchmark shown in (a) assembled state and (b) dissassembled state.

is empty, the initial environment could be described as follows:

```
At(Hand,Starting_Loc)
Handempty
Positioned(Casing1,Casing1_home)
Positioned(Casing2,Casing2_home)
Positioned(Lever,Lever_home)
Positioned(Spacer,Spacer_home)
Inserted(Peg,Peg_home)
Inserted(Large_pin_1,L_pin1_home)
Inserted(Large_pin_2,L_pin2_home)
...
Inserted(Small_pin_1,L_pin1_home)
Inserted(Small_pin_2,L_pin2_home)
...
```

and if the job consists of assembling the three large parts, for example, the goal conditions (or job result) provided by the human supervisor would be:

```
Positioned(Casing1,Jig_lower_center)
Positioned(Casing2,Jig_upper_center)
Inserted(Peg,Casing_1_center_hole).
```

Since the predicate world model is constantly verified and updated by the Automated Monitor during task execution, its accuracy is obviously highly dependent on the abilities of the Automated Monitor to sense changes in the environment. To guide and improve this function of the Automated Monitor, the Job Planner produces a list containing all changes that are expected when the plans are executed (see later in this section) and passes it to the Automated Monitor prior to execution of each task.

A task operator is a description of a given task which may be performed by the human or the robot, and has a syntax similar to that of the STRIPS planning system [14,15], i.e., it consists of the set of conditions which must be true before the task operator can be exercised and the conditions which either become true or become false subsequent to the application of the operator. Two examples of valid task operators used to plan the Assembly/Disassembly of the Cranfield Benchmark are:

```
Grasp(*object)
  PRECONDITIONS:  Handempty
                  Found(*object)
                  At(Hand,*object:
                    Hover_pos)
                  END
  DELETE_LIST:    Handempty
                  Inserted(*object,-)
                  Positioned(*object,-)
                  END
  ADD_LIST:       Grasped(*object)
                  END
Move_Arm(Curr_Loc,*to - loc)
  PRECONDITIONS:  END
  DELETE_LIST:    At(Hand,-)
                  END
  ADD_LIST:       At(Hand,*to_loc)
                  END
```

In terms of these data which reside in the knowledge bases, the job planning problem can then be expressed as follows: let $E = \{e_1, e_2, \ldots\}$ be the set of initial environmental conditions, $G = \{g_1, g_2, \ldots\}$ the set of goal conditions, and $T = \{t_1, \ldots, t_m\}$ the set of m task operators, where each t_i consists of 3 lists:

$P_i = \{p_{i1}, p_{i2}, \ldots, p_{i\max p}\}$, list of preconditions

$A_i = \{a_{i1}, a_{i2}, \ldots, a_{i\max a}\}$, add conditions

$D_i = \{d_{i1}, d_{i2}, \ldots, d_{i\max d}\}$, delete conditions

find the sequence of actions $t_1, \ldots, t_n$ which transform the initial world model such that the set of conditions G is true subsequent to execution of t_n. The algorithm used to implement this methodology is a typical depth-first search strategy, in which the operators investigated for inclusion in the plan are those with add and delete conditions that, when appropriately instantiated, match some of the goal conditions. The correspondingly instantiated preconditions of a selected operator are added to the list of goal conditions for the next iteration. The planning process thus proceeds 'backwards' from t_n, until all goal conditions are satisfied and the operator t_1 fulfilling the initial conditions has been found. General descriptions of these strategies have appeared in [14–20] and only those features and procedures particular to our human–robot synergy system are detailed below. The reader is referred to [22] for a detailed programming of our system.

3.2. Particular features of the implemented Job Planner

When a dead-end is encountered in the search, the system backtracks to the previously selected operator. However, because some of the goal conditions may have been deleted or instantiated during the previous application of operators which are now found inapplicable, and because of the memory requirements involved in saving all intermediate versions of the environmental model, pure backtracking is not utilized. Instead, only the final goal conditions G are saved and backtracking is performed by applying to this goal state the current list of selected operators less the first one in the list (i.e., the last one investigated and found inapplicable). In other words, if a dead-end is encountered while the operator list is $(t_j, t_{j+1}, \ldots t_n)$ the new list of working goal conditions is found by applying $t_{j+1} \ldots t_n(G)$. Although we have not performed formal comparisons of the memory requirements and running time of this approach compared to that of formal backtracking methods, the assurance of accuracy using this method dominated our choice in this implementation of the Job Planner.

Another characteristic of the planning algorithm resides in the instantiation of parameters for matching of goal conditions, where four types of parameters can be used. A parameter of '–' serves as a type of place-holder, meaning that it can match anything. An identifier preceded with '*' refers to an uninstantiated parameter which can be matched to anything, as long as it is instantiated to the same value in every occurrence of the parameter in the operator (e.g., see the examples of operators given above). An identifier without a prefix must remain exactly as is given in the operator. An identifier preceded by '#' refers to an uninstantiated parameter which must be matched to a value (i.e., object) of a certain type (given by the identifier) throughout the operator. This allows much more flexibility in the planning process as some goal conditions can be expressed in terms of generic objects such as in Grasped(# S_pin) or Insert(# S_pin, jig_1_loc_2) where any available small pin can be used to perform the task. The use of such generic parameters is based on the assumption that generic objects (e.g., nuts, bolts, etc.) are available in sufficient quantity that they need not be instantiated in the 'strategy' plan, thus greatly simplifying and speeding up this level of planning. This also allows the system to start some jobs which otherwise would not have been proceeding because of possible insufficient quantities of these generic objects. The object knowledge base is organized in frames which include the type of each object, if appropriate, and specific ranges of instantiation values (e.g., small_pin_1) for use at the time of task allocation. If a specific object happens to not be available for instantiation of a generic object, a simple message is transmitted to the operator from the Automated Monitor pointing out that # object has run out.

In the current implementation, no attempts have been made to derive first-level plans that are optimal in terms of logic (the most 'sensible' plan), cost (e.g., the plan with the fewest number of operators), or time (the plan quickest to execute). Instead, the human–machine synergy concept is exploited, in which the planner presents to the human supervisor the first-obtained plan that fulfills the goal. The human can then approve the entire plan or reject sections of the plan (e.g., t_j to t_m). In the latter case, the planner generates the world model conditions corresponding to the beginning and end of the rejected plan section using the 'backtracking' method described above, and regenerates a plan for these initial and goal conditions, however, considering that operator t_m has lead to a dead-end. This effectively prevents use of operator sequences ending with t_m for the revised plan.

In some conditions, more than one task operator is investigated during the planning search to achieve a desired condition (i.e., the ADD and DELETE list of several task operators include the desired condition). In these situations, a method must be incorporated for selecting the 'best' rule to use. In the current system, this selection is accomplished using two heuristic measurements: percent-achieved (PA) and percent-difficult (PD). The percent-achieved factor gives the percentage of the operator's preconditions that are currently met in the planning environmental model. The percent-difficult factor gives the percentage of the currently-met preconditions which are 'difficult' to achieve. This factor expresses the fact that, on the basis of its own experience, the system has found that some operators have a list of preconditions which are

'harder' to meet than others as obtained from the subtask failure/success rate observed over time by the Automated Monitor, analyzed by the learning module [23] and maintained in the task knowledge base. In our current implementation, only two levels of difficulty were utilized to distinguish principally the degree of numerical complexity involved in the sensing and motion planning procedures implied by each operator (e.g.,image analysis vs. knowledge base search for operators fulfilling a 'found (object)' condition). From a set of applicable task operators, the one with the highest PA factor is selected. If the operators have the same PA factor, the operator selected is that with the lowest PD factor. If the operators still tie, they are considered practically equivalent from a strategy planning point of view and the operator found first is arbitrarily selected.

Deriving this linear, non-hierarchical task sequence is much less complex than deriving the full hierarchical description of all motions involved in the job and allows a succinct first level of activities to be submitted to the human operator or supervisor for approval. Furthermore, this task sequence derived by the Job Planner allows for rapid reconfiguration if problems in task execution or changes in the human or robot capabilities occur at a later stage of operation. This reconfiguration typically is triggered by and occurs in close cooperation with the Dynamic Task Allocator and the module in the Automated Monitor (described below), which is responsible for detecting events requiring modification of the task plan. To provide information to the Automated Monitor about some conditions that are to be expected during execution, an Execution Monitoring Table (EMT) is built as the job is planned. As each new operator is added to the job plan, an EMT entry for that operator is created. The entry consists of the instantiated preconditions of the new operator and a set of continuing conditions. The continuing conditions are those conditions which were on the ADD LISTs of earlier operators but have not yet appeared on the PRECONDITIONS list of a subsequent operator. Examples of EMT's for operators instantiated in the plan for assembly of the Cranfield Task are given in the Appendix. Once the job is planned, the Execution Monitoring Table is available to the Automated Monitor for use during task execution to detect problems in performance of the job as planned (see next section).

3.3. Sample results of the Job Planner

The job planning algorithm has been implemented in the 'C' programming language on a 68020 processor using the OS-9 operating system. Experimental scenarios utilizing the 'Cranfield Benchmark [21] Assembly/Disassembly' were selected to test the functioning of the individual modules and the intermodule communications in the symbiont architecture. The following example illustrates the results of the job planning module. As shown in *Fig. 3*, the Cranfield Benchmark consists of 18 parts: 2 casings, 1 lever, 1 spacer, 1 peg, 1 jig, 4 large pins, and 8 small pins. The type of actions required to assemble/disassemble the Benchmark include Grasp, Release, Insert, Place, Move_Arm, and Find, and corresponding action operators (see list in the Appendix) were used for the planning. Testing of the proper functioning of the module was accomplished by generating plans for a wide variety of initial and goal state scenarios. The job plan derived for a full assembly of the benchmark (see *Fig. 3b*) from its disassembled state (*Fig. 3a*) is shown in the Appendix. This plan contains 119 actions and requires approximately 20 'wall clock' seconds on the OS-9/68020-based system. For the sample set of operators utilized here, and the variety of scenarios considered, the increase in processing time with increasing job plan length was found nearly linear, requiring approximately one second for every six actions in the plan. Our conjecture for this near-linearity is that, with the type of plans and operators considered here, the search for the assembly solution is almost entirely directed by the instantiation of the object parameters since little redundancy exists in the action operators, i.e., no two operators or sequence of operators can result in the same effect. For the general case, however, the planning time requirements would also be dependent upon the number of operator rules utilized, their degree of redundancy, and their complexity. For the applications considered here, the observed processing times are typically orders of magnitude faster than what would be required for human planning and input into the system, and should be quite acceptable

when implemented on newer processors with speed-up factors of 4 to 10.

4. Execution monitoring for the human–robot symbiont

In the human–robot symbiont, several key functions are provided by the Automated Monitor, all of which revolve around the observation of the task execution. First, the Automated Monitor is responsible for detecting when problems in task execution have occurred. This aspect of the Automated Monitor focuses on the integration of the planning and sensing operations with emphasis on monitoring that the conditions assumed to be true in the planning (and listed in the EMT) still hold at execution time. After determining that a problem has occurred, the Automated Monitor proceeds to perform its second function – working to derive a modified job plan to reach the goal in spite of the task execution problem. This ability to recover from execution errors is a key to improving the performance of the symbiotic system and to the reduction of the fragility of the system as discussed in several papers on this topic (e.g. see [24–27]). The philosophy behind our approach to this function has been to reuse the existing job plan to the fullest extent possible to reduce the time of replanning. The third and fourth duties of the Automated Monitor are to maintain the consistency between the real world and the knowledge bases through observation of the actual performances of the agents (human and robot) during task execution and to provide appropriate observation data to the learning module. These data, in turn are used by the system to learn new tasks and objects from observations of the human actions, and to adjust the agents' skill and knowledge parameters as necessary to reflect their current capabilities. These latter two functions have been described in detail in a previous paper [23] and consequently, this section will only focus on the former two: detecting problems in task execution and recovering from those problems.

4.1. Detecting problems in task execution

While a number of sensor-based execution monitoring approaches which have been investigated in the past (see for example [24,25,26]) would apply to symbiotic systems, the monitoring strategy developed for this project was selected to correlate tightly with the job planning methodology since most of the information about the general conditions which are expected at execution are generated with the plans in the Job Planner. This section describes the methodology used for the detection of problems in task execution.

How adeptly the Automated Monitor can detect that a problem has occurred is directly related to sensor availability and feedback. For example, manipulation-related conditions, such as 'Grasped(*object)', 'Handempty', or 'At (Hand,*loc)', can be verified using sensors such as force/torque, gripper width, tactile, and end-effector position sensors, whereas object-relationship conditions, such as 'At(Wrench,Toolbox)', 'On(Box1,Box2)', or 'Connected(Bolt1,Plate4)', need to be verified using vision sensors. The basic concept utilized here consists of generating expectations of sensor-based readings when the tasks are planned and in comparing them at critical points in time with the actual sensor readings during execution. If any discrepancy is detected, the human in the human–robot symbiotic system is informed of the inconsistency to verify that a problem exists, and is requested to intervene in the task execution if he does not approve the contingency plan proposed by the system.

To accomplish this, the Automated Monitor uses the following methodology: Let the available sensor suite be denoted by $S = \{s_1, s_2, \ldots, s_p\}$, where p = total number of sensors; the generic types of conditions describing the environment by $C = \{c_1, c_2, \ldots, c_m\}$, where m = number of condition types; and the set of available values (e.g., object weight, end-effector position, etc.) that can be calculated from the sensor data at given points in time by $R = \{r_1, r_2, \ldots, r_n\}$, where n = number of obtainable characteristics. Let $V_i^1, V_i^2, \ldots$, be sets expressing valid representations of R under an expected condition c_i, where:

$$V_i^1 = \{v_{i1}^1, \ldots, v_{i2}^1, \ldots, v_{in}^1\},$$

$$V_i^2 = \{v_{i1}^2, v_{i2}^2, \ldots, v_{in}^2\},$$

and $E_i = \{V_i^1, V_i^2, \ldots\}$. Then, if $R \in E_i$ for an expected condition c_i, no problem exists, and execution can proceed normally. Otherwise, a

discrepancy has been detected and the human will be notified of the potential problem.

As an example, assume that the available sensor suite, S, includes {tactile sensor, gripper_width encoder, joint encoders}, and that the generic conditions describing the environment, C, are {Grasped(*object), Handempty, At(Hand, *loc)}. Further assume that the sensor-based values that can be calculated are either a 0 or a 1 from the tactile sensor, meaning 'no object sensed' or 'object sensed', a non-negative number from the gripper_width encoder data, indicating the width of the gripper (0 implies closed); and the three position coordinates (x, y, z) and three orientation angles of the end-effector from the joint encoders data. Then, for each condition in C, the sets of expected sensor readings E_i can be derived as follows:

for c_1 = Grasped(*object),

$V_1^1 = \{1, >0, -\}$,

for c_2 = Handempty,

$V_2^1 = \{0, -, -\}$,

$V_2^2 = \{-, 0, -\}$,

for c_3 = At(Hand,*loc)

$V_3^1 = \{-, -, *loc\}$,

where '–' means the reading is not applicable, or does not matter. Now, assume that at a certain point during task execution, the Automated Monitor expects the condition 'Grasped(*object)' to be true. It obtains the actual sensor readings and finds that:

R = {tactile_reading = 1,
gripper_width = 2.5,
position = (3,4,5,90,45,60)}

Comparing these readings to the expected readings, V_1^1, it determines that $1 = 1$ (tactile), and $2.5 > 0$ (gripper_width), so the 'Grasped' condition appears to be true. One planned modification not yet implemented in the current system could be to make the expected gripper_width equal the actual width of the object being grasped to within a given threshold. This would allow either verification that the correct object is being grasped, rather than just any object or, when this capability will be included in our object characteristic learning system [23], to learn new grasping positions and characteristic dimensions of a verified object. The two sets of valid expected sensor readings, V_2^1 and V_2^2, for c_2, 'Handempty', correspond to the gripper being either open or closed with no held object. In these situations, if the tactile sensor detects nothing, the gripper width sensor reading is immaterial, whereas, if the gripper width is zero, the tactile reading is unnecessary. In either case, the position sensor is not applicable.

To setup and make these comparisons, the Automated Monitor needs information not only on the conditions that characterize execution of a type of subtask, but also about those conditions which convey the intent that the task is expected to fulfill with respect to the overall plan. Consider, for example, the action of moving the arm from point 'a' to point 'b'. The motion is the same regardless of whether or not an object is being held during the move, but the sensor readings are *not* the same (e.g., in this case the tactile readings). The Automated Monitor needs to be informed whether the intent is to transfer an object or simply to move to a new position in order to properly detect problems in the execution. Since the Job Planner knows the intent of each task as well as the overall plan to be executed, it provides this information to the Automated Monitor in the form of Execution Monitoring Tables (EMT), which include the conditions expected to hold both prior to subtask execution (preconditions) and during subtask execution (continuing conditions).

Recall that the Job Planner produces a list of actions, $t_1, \ldots, t_n$. When the final plan has been generated, the EMTs are then constructed consisting of the following for each task t_k, $1 \leq k \leq n$:

PRECONDITIONS: Preconditions, p_k, of action t_k
CONTINUING CONDITIONS: ALL ADD LIST conditions, A, of previous tasks $t_1, \ldots, t_{k-1}$, which have not yet appeared on the PRECONDITIONS list, P, or DELETE LIST, D, of a subsequent task.

The continuing conditions express the fact that the effects of tasks in a job plan should remain true until they are needed as preconditions of a

later task, or until they are nullified by a later task. Future enhancements to this current implementation could (1) distinguish between desired effects of operators and side-effects (which are not needed by any future task), and (2) allow conditions to remain in the set of continuing conditions when needed by more than one subsequent task. To each type of predicate condition corresponds a type of expected sensor reading (see the sets V_i described previously). After instantiating these with the parameters of the EMTs, the Automated Monitor can compare the actual sensor readings to the expected sensor readings both prior to and during task execution to detect problems in task execution.

If a problem event is detected during task execution, the Automated Monitor first informs the human of the violated condition. The human can then instruct the Monitor to simply ignore the problem and continue with task execution (e.g., if the supposed problem does not require corrective actions). If this is not the case, the human acknowledges the detected problem and can either instruct the Monitor that he will perform 'manual' recovery, in which case the Monitor informs the Dynamic Task Allocator of the human's request and triggers the learning system, or the human can instruct the Automated Monitor to derive a new or revised plan, as described in the following section.

In addition to the knowledge about execution conditions which is available in the Execution Monitoring Tables, the Automated Monitor also has knowledge from the Dynamic Task Allocator about the expected times for subtask completion. This information is used during the performance of a job that was allocated based upon a policy of minimizing the expected execution time (see later section on Task Allocation). During subtask execution, the Monitor uses this information to detect unexpectedly long attempts to complete a subtask and warn the human of suspected problems. This feature is of particular assistance during robot performance, and particularly during the early phases of operation since the robot skills may be low and the human might not be aware of the reasonable length of time required for the robot to perform the current subtask. The human can respond by instructing the Automated Monitor to either ignore the elapsed time, reallocate the subtask, or replan the job.

4.2. Replanning due to task execution problems

After detecting that a problem has occurred and receiving instruction from the human to replan, the Automated Monitor works to recover from the error by deriving a new or revised plan to achieve the goal. A number of recovery techniques are possible in this scenario, varying from restarting execution to undergoing complete replanning (refer to [27] for a recent review of existing techniques). Ideally, the existing job plan should be reused to the fullest extent possible to prevent a waste of time in replanning. Only those portions of the plan which are no longer useful in reaching the final goal should be replaced with a new sequence of operators.

The replanning technique developed for this symbiont system involves two levels of recovery: utilizing predefined recovery strategies for the task execution problems detectable with the available sensor suite, and undergoing complete replanning when such a recovery strategy is unavailable. Aside from speeding up the recovery process, this two-level strategy allows utilization of simple procedures, in cases where recovery is 'straightforward' and/or can be easily learned from observation of the human actions in previous recoveries.

For this type of direct recovery, the strategy can be stated as follows:

Let

$C = \{C_1, C_2, \ldots, C_k\}$ be the set of k conditions that the Automated Monitor is able to detect as valid or violated, and let each subset of C be defined as a 'problem event', P_i.

Further let

$P = \{P_1, P_2, \ldots, P_j\}$ be the set of j problems for which a recovery plan exists in the knowledge bases, and

$FX = \{F_1, F_2, \ldots, F_j\}$ the set of corresponding recovery plans,

where:

$F_i = \{t_{i0}, t_{i1}, \ldots\}$ is the list of subtasks to be performed for recovery from problem event P_i.

Then, when a problem event $P_k \in P$ is detected as a combination of violated conditions $\{C_j, C_m, C_n \ldots\}$ the plan $F_k \in FX$ can be inserted into the job plan prior to the subtask, SF, during which the problem event occurred, and will recover from P_k if the subtasks in F_k are instantiated with the parameters of the violated conditions $\{C_j, C_m, C_n \ldots\}$ and of the subtask SF.

As an example, assume that during the execution of a 'Move_Arm' action, the Monitor detects that the carried object, say a 'Lever', has been dropped – that is, the continuing condition 'Grasped(Lever)' in the EMT is no longer true. The Automated Monitor consults the human, who acknowledges the problem and instructs the Monitor to propose a recovery plan. If a recovery plan exists in the knowledge bases for the problem event 'dropped object', defined by a violated 'grasped(*object)' condition, the Monitor will use that plan, say F_1, which consists of the following tasks:

$F_1 = \{$Find(*object)
Move_Arm(*object:Hover_pos)
Grasp(*object)$\}$

The Monitor determines the identity of the dropped object by extracting the object's name parameter from the 'Grasped' condition (in this case, 'Grasped(Lever)') of the Execution Monitoring Table condition which became untrue. The recovery plan F_1 is then instantiated to become:

$F_1 = \{$Find(Lever)
Move_Arm(Lever:Hover_pos)
Grasp(Lever)$\}$

These tasks are then inserted into the job plan prior to the Move_Arm action that was being performed, and are given to the Dynamic Task Allocator for allocation to the human or the robot. Once human approval is obtained, task execution begins again to re-grasp the Lever. If no recovery plan already exists for the problem event, the Monitor instructs the Job Planner to undergo replanning from the current state to the goal state using the methodology for partial modification of a plan described in the previous section.

4.3. Sample results of the Automated Monitor

This methodology, like that for the Job Planner, has been implemented in 'C' on an OS-9/68020-based system, and tested with the Cranfield Benchmark scenario discussed in the previous section and depicted in *Fig. 3*. A portion of the Execution Monitoring Tables generated by the Job Planner and used by the Automated Monitor is shown in the Appendix. In the experimental setup, the sensors available to the Automated Monitor were the tactile, gripper-width, force-torque, and joint encoders processed to produce end-effector position. With these sensors, the Automated Monitor was demonstrated to reliably monitor the conditions 'Grasped', 'At(Hand,*loc)', and 'Handempty', thus allowing the detection of problem events such as 'object dropped' or 'object not grasped as expected'. The Automated Monitor was also able to monitor the elapsed execution time to warn the human of task execution times exceeding the expected times. When detecting these problem events during executions of Cranfield Assembly/ Disassembly tasks, the Automated Monitor also successfully communicated with the human and resolved the problems using either recovery planning strategies described above.

In these experiments, however, programming was used to input the predefined recovery plans in the knowledge bases. Our continuing work in this area focuses on enhancing the Automated Monitor-Learning System communications to allow automated learning of recovery plans by observation of the human actions using the task learning capabilities of our learning module [23].

5. Dynamic task allocation for the human–robot symbiont

An essential requirement of the task allocator in this human-machine system is its ability to be event-driven, responding to changes in the work environment, physical constraints, or unexpected events by altering the task allocation to adjust to new conditions. This dynamic nature of the task allocator allows the man–machine symbiont to cope with a changing environment, causing the resource most appropriate for performing a sub-

task to be assigned the subtask [28,29], while the other can observe performance.

The purpose of the task allocator in the man–machine symbiosis is to attempt to dynamically optimize the division of work between the man and the machine. For this, the task allocator must know what allocation policies (i.e., optimality criteria) are placed on the task allocation, what the requirements of the subtasks are, and information concerning the characteristics of the environment in which the problem is to be solved. The task allocator must also have information about the skills and capabilities of the human and the intelligent robot controller in order to determine which one is most appropriate for performing a subtask in a given scenario. This information, which continuously varies with time, is periodically updated by the Learning System [23] or the Automated Monitor, and resides in the knowledge bases of the system. These data and the allocation methodologies are described below.

5.1. *Allocation policies*

The allocation policies place performance measures, limitations, and/or restrictions on the task allocation problem solution. The intent of these allocation policies is to alter the task allocation strategy to adapt to differing problem contexts. Examples of performance measures include minimization of job completion time, minimization of human involvement, maximization of result quality, etc., while limitations and restrictions express constraints on the allocation such as preventing or mandating use of certain resources or agents for given subtasks.

The performance measures are provided at time of allocation, i.e., just prior to execution of the tasks, by a source external to the task allocator (which typically is the human agent or supervisor and ultimately could be a synergy supervising system). For each type of performance measure (optimization criterion), corresponding data must exist in the knowledge bases for the allocation to take place. This data is quantitative and in the form of 'capability factors' which express the abilities each agent possesses for accomplishing each type of task with respect to each performance measure. The limitations and restrictions are implemented as {0, 1} indexes expressing whether a resource (e.g., vision, tactile, search, etc.) or an agent is currently available and can be considered for allocation of a subtask. Note that since several resources may typically be needed to accomplish the several elemental subtasks which constitute a SAS (smallest assignable subtask, discussed in the previous sections), the unavailability of only one resource to an agent prevents allocation of the SAS to that agent, thus severely limiting the solution space.

5.2. *Agents' capability factors*

The agents have been defined as the 'intelligent' entities (humans or computers) which are available for performing subtasks to solve a problem, or to achieve a goal. In this paper, only two agents are considered: a human and an intelligent robot controller. In order to allocate tasks among these agents, the task allocator must have information concerning their capabilities. The agents' capabilities are defined to be either the abilities the agents have to perform certain physical actions such as Move-Arm or Find(*object), or the knowledge they have of certain objects such as 'wrench' or 'bolt'. The actual information stored about the capabilities of the agents is directly related to the allocation policies which might at some time be present in the problem scenario. For example, the policy 'minimize time of task completion' requires that 'timeliness of achievement' factors exist in the knowledge bases, while the policy 'maximize quality of result' requires that 'level of achievement' factors exist.

Although the agent capabilities are quantified differently depending upon whether the capability refers to a physical ability or to knowledge about an object, one evaluation number is obtained for each factor (such as level of achievement) for each capability. The normalized evaluation numbers are then used in determining the appropriate task allocation. For a physical ability, the evaluation number indicates the skill with which the ability is performed, on a scale from 0 to 1, corresponding to 'unacceptable' to 'superior'. For the knowledge about an object, the evaluation number indicates how complete the knowledge of that object is, also on a scale from 0 to 1, from 'unknown' to 'always known'. The lowest value on the scale is reserved to indicate that the agent does not possess the capability. With the normalization, the evaluation number

characterizing capability i of agent x under the allocation policy p is denoted θ_{pi}^{x}. Thus, as an example, if the effective allocation policy p is 'maximize the quality of the result', θ_{pi}^{x} provides the normalized level of achievement factor associated with capability i of agent x for the optimization. Default values (typically 0 for the machine expressing no knowledge) for these factors are initially input into the system. As tasks are performed, these agent capability factors change with time reflecting the acquisition of new knowledge and experience through the Learning System [23].

5.3. Tasks

As described in the previous section, the Job Planner decomposes the job to be performed into its component activities and sends the plan as input to the Task Allocator. The Task Allocator uses the same methodology to break down each activity into a sequence of tasks to fulfill the goal of the activity. Each activity is associated with a frame in the task knowledge base, which contains the list of operators related to planning of this specific activity, as well as their description. Each list is augmented when a task performed by the human under a given activity has been learned by the machine from observations. The decomposition takes place just prior to execution of the activity so that the updated knowledge of actions, objects, agent and resource availability is used in the operators and object instantiation. The key idea behind this second level of planning is that each activity is therefore decomposed into a set of smallest assignable subtasks (SAS), i.e., the smallest units that can be feasibly performed by a single agent. If the machine has learned some subtasks under an activity, it should be able to exercise each subtask as a whole. The remaining part of the activity, whose decomposition in subtasks is still unknown to the system, remains identified as a single SAS which will be assigned, by default and as a whole, to the human since the machine has yet no knowledge of it. The methodology which is described in the remainder of this section thus consists of allocating those SAS of which the machine has some knowledge, either to the human or the machine, based on their respective current skills at performing these SAS.

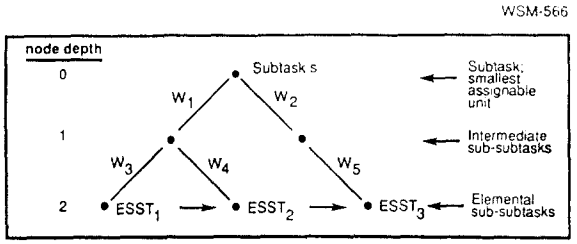


Fig. 4. Schematic of SAS subtask characterization.

Because their perception and control means are considerably different, the agents may perform the same subtask quite differently, using very different resources. Furthermore, since the capability factors θ_{pi}^{x} need to express generic levels of skills for each agent independently of the type of task to which they contribute, the subtasks are characterized, differently for each agent if appropriate, by a set of 'elemental sub-subtasks' (ESSTs). Since each ESST is comprised of generic capabilities (see *Fig. 4*), it can be parameterized by a merit factor independently of the environment and the context of the problem. The concept of an 'elemental sub-subtask' is very important since it represents the smallest subdivision of the elements of a task which correlate with the physical mechanics of the symbiont operations. For the system considered here, the set of ESSTs included each of the perception capabilities (e.g., remote vision, force feedback, etc. for the human, or tactile data analysis, force torque interpretation, inverse kinematic calculation, etc. for the controller), control of the manipulator end-effectors' position and orientation, and each of the search methods (e.g., visual search, knowledge base search, etc.).

5.4. Determining the allocation recommendation

To allocate the SAS to either the human or the intelligent controller, the Task Allocator analyzes their respective ESST tree decomposition, weighing the capabilities needed by each agent for each SAS according to their current skills and availabilities, and generates an allocation recommendation based on the evaluation factors θ_{pi}^{x} and importance factors as follows:

Let C^{sx} denote the set of all capabilities required for resource x to perform subtask s. To

each capability i in this set C^{sx} we associate the importance factor f_i^{sx}.

$$f_i^{sx} = \sum_{j=1}^{b^{sx}} \left[\left(\prod_{h=1}^{r_j^{sx}} w_{jh}^{sx} \right) (m_{ij}^{sx}) \right], \quad 0 \le w_{jh}^{sx} \le 1, \quad 0 \le m_{ij}^{sx} \le 1 \tag{1}$$

where:

b^{sx} = number of ESSTs in the subtask tree for subtask s of resource x,

r_j^{sx} = the depth of the jth ESST in the tree of subtasks for resource x,

w_{jh}^{sx} = the weight associated with the branch into the node at depth h of the path from the root to the jth ESST of the subtask tree s for resource x,

m_{ij}^{sx} = the merit factor associated with the ith capability required by the jth ESST of subtask s for resource x.

The weight value associated with each branch emanating from a node in the subtask breakdown tree indicates the importance of that branch in the successful performance of the subtask or sub-subtask represented by that node. The merit factor associated with each capability indicates the importance of that capability in the successful performance of the elemental sub-subtask, relative to the other capabilities required by that elemental sub-subtask. At the beginning of the problem execution, these weight values and merit factors are given default values. As subtasks are performed over time, the Automated Monitor and the Learning System alter the weights and merit factors after each subtask completion to reflect new knowledge about the tasks on the basis of their success/failure rate [23]. In deriving allocations, the task allocator therefore uses the most recently adjusted weight values and merit factors.

To obtain the task allocation recommendation for a job including the ordered set T_t of SASs, the task allocator seeks to determine, for all agents involved x, $1 \le x \le n$, the membership of the sets T_t^x, which are the sets of SAS which should be performed by resource x. This is done by comprising the sets T_t^x such that, for all SAS in T_t, $s \in T_t^x$ if x respectively maximizes or minimizes:

$$\sum_{i \in C^{sx}} (\Theta_{pi}^x)(f_i^{sx}), \quad \Theta_{pi}^x \neq 0, \tag{2}$$

for a maximization or a minimization allocation policy p; and

$$T_t = \cup T_t^x, \quad 1 \le x \le n, \tag{3}$$

where n is the number of agents involved.

One of the key features of this task allocation methodology is its ability to be event-driven, responding to changes in the information about the allocation policies, available resources, agents' capabilities, or tasks by altering the task allocation. Such a dynamic nature of the task allocation is essential to allow the man-machine symbiont to cope with a changing work context and is directly related to the capabilities of the Automated Monitor and Learning System to update the information contained in the knowledge bases. As described previously, such changes could take place in two ways: through learning schemes and through execution monitoring. The learning scheme allows the robot to learn and improve its capabilities by observing the human while the monitor observes the agents' performance to quantify and/or modify their skills (achievement factors) and capabilities (merit factors) in achieving given tasks. As also mentioned previously, the information in the third knowledge base, the task information, is subject to change during the execution of the subtasks when environmental changes or unexpected events occur which require the Job Planner to update the list of subtasks to be performed, and the Task Allocator will respond dynamically to these changes by replanning the task allocation appropriately.

5.5. *Sample results from the Task Allocator*

Within the architecture developed to illustrate human–machine symbiosis, it is crucial that all of the reasoning modules be synchronized for the sharing of knowledge and the transfer of control, particularly during task execution when problem events, task failures, or normal task completions occur. Synchronized program flows were developed and implemented for the Job Planner, Dynamic Task Allocator, and Automated Monitor to allow the knowledge-sharing and transfer of control to occur. At this stage, a synchronizing interface has not yet been necessary or implemented for the Learning System, which works on a much different time scale. These program flows have been presented and discussed in detail in

[22]. Inter-module communication was accomplished by the sending and receiving of messages between modules, implemented in the OS-9 system as pipes. With these program flows and message-passing, the high-level reasoning modules (Job Planner, Dynamic Task Allocator, and Automated Monitor) successfully cooperated in sample symbiont executions which included unexpected events during subtask execution.

This system has been implemented using five coarsely parallel processors, four of them being Motorola 133 processors, each with a 68020 CPU and 68881 FPU. These processors use the OS-9 (TM) operating system developed by Microware. The fifth processor in the system is a separate Macintosh II computer which has been used to create a graphical Human-Machine Interface as part of the Presenter/Interpreter. The Motorola processors reside on a VME bus, which provides the backplane connection required for various I/O devices. The VME bus is well standardized and supports a great many off-the-shelf I/O boards. The system uses parallel I/O cards for the CESARm's brakes, encoders, and safety interlocks, and also for the force/torque sensor. A D/A card is used to drive the pulse-width-modulated amplifiers for the actuators. Serial connections are made to the tactile sensor and to the Macintosh. An outline of the partitioning of computations is given below. The letters indicate that the process is active during either the (A) autonomous mode, (T) teleoperated mode, or (B) both.

CPU1:
(1) CESARm's PID (B)
(2) CESARm's Cable, joint, and velocity limit checks (B)
(3) CESARm's path segment generation (A)
(4) CESARm's forward kinematics, for force reflection in master (T)
(5) CESARm's I/O for sensors and actuators (B)
(6) Intelligent Controller (A)
(7) Knowledge Base Manager (B)

CPU2:
(1) CESARm's redundancy resolution and optimization (B)
(2) CESARm's obstacle detection (B)

CPU3:
(1) Master's forward kinematics (T)
(2) Master's inverse kinematics (T)
(3) Master's I/O (T)

CPU4:
(1) Job Planner (B)
(2) Dynamic Task Allocator (B)
(3) Automated Monitor (T)

Macintosh:
(1) Graphical Interface (B)

The partitioning was not designed to exactly divide the system's computations between each processor. Actually, the processes listed above do not lend themselves particularly well to fine-grained parallel computation and it was decided to pack CPU2 and CPU3 with the processes that were reasonably stable in their content while keeping CPU1 as underutilized as possible. This permitted quicker alterations to the system because it allowed CPU1 to accommodate any new tasks as they were identified while minimizing the chore of repartitioning. With this partitioning, CPU2 requires 90% of the 0.01 sec. loop time to complete its computations while CPU3 ran flat out with a loop rate of 60Hz. The bandwidth of the manufacturer's serial communication link to the master was a limiting factor here [30].

The proper functioning of each module was first verified in simple tests using simulated data. For the Dynamic Task Allocator, sample SAS decomposition trees, simulated capabilities, and random merit factors were utilized in a variety of test problems, representative of various stages of knowledge of the machine. Integrated experiments were then performed on assembly/dissassembly tasks using the Cranfield Benchmark. All parts of the Benchmark were utilized except the 8 small pins which created two different types of problems: their small size made them difficult to grasp and practically impossible to insert with the CESARm testbed (even in teleoperated mode), and their attributes (weight, size) fell below the accuracy of the arm and the discernable threshold of the force/torque and tactile sensors, making their characteristic location, weight, tactile inprint, and release end-effector position (assumed to be the same as the end-effector position to re-grasp) unlearnable for the machine. For these experiments which focused on the integration of the Planner, Allocator, and Monitor, an early version of the learning module was used which had capabilities to learn object characteristics and only some motion tasks [23] such as Grasp and Release. This, nevertheless, was sufficient to show feasibility of changes in

knowledge and skills. Other primitive tasks such as Find, Insert, Move_Arm, Search (in knowledge base or using vision) were not learnable by the system and their autonomous execution was not programmed in the knowledge bases, implicitly forcing their allocation to the human at all times. As mentioned previously, recovery plans were also input to the system.

Series of problems were performed, each consisting of successive full or partial assemblies and disassemblies of the Benchmark. For each problem, different initial states of knowledge of the machine were studied, as well as some unexpected events (moving objects away from their learned location, 'dropped' object). As mentioned in the previous sections, the Job Planner and Execution Monitor performed well on all these tests, properly detecting execution problems and replanning as appropriate. Surprising task allocation results were initially obtained which eventually got tracked down to the method for updating the skill factors for learned tasks: when the characteristics of an object are learned, the skill factor for this object (representing the amount of knowledge about the object) is immediately updated from its initial zero value. For tasks however, the skill factor is estimated from the success/failure rate of the machine performance on that task. Since, with an initial skill of 0, the machine is never allocated that task, its success rate forcibly remains zero, even after having acquired sufficient knowledge to be able to perform the task.

Although we have not yet developed an enhanced task knowledge skill evaluation formalism to remedy this problem, a simple alternative was used for the purpose of demonstrating the integrated dynamics of the Planner, Monitor, and Allocator. We monitored the knowledge bases for the learned task data and, at the appropriate time, artifically added a Grasp(object), Release (object) sequence (the two tasks considered for learning here) in the list of tasks to be performed by the machine. This 'gave a chance' to the machine at performing the tasks and initiating the update of its skill factor. With this modification, the system properly performed all allocations, including those following some task learning. For the sample problem shown in the Appendix, for example, which consisted of a full assembly with no initial machine knowledge (skill factors at zero), followed by a full disassembly and a reassembly except for the small pins, and for an allocation policy minimizing human involvement (allocate the task s to the machine x if $f_i^{sx} \neq 0$ and $\Theta_{pi}^x \neq 0$ for all capabilities i involved in s), the beginning of the first assembly was properly entirely allocated to the human. The data about the two learnable tasks were observed suitable for autonomous performance after four to six instances of their performance by the human, and were then artificially allocated once to the machine. Only the Release task, however, was consequently allocated by the system to the machine during the first assembly. This comes from the fact that in the assembly plan, each object is sequentially manipulated through a Grasp(*object), Move_Arm(*new location), Release(*object) sequence, with the machine having no knowledge of the object ($\Theta_{pi}^x = 0$) prior to the Grasp task, whereas it has acquired some ($\Theta_{pi}^x \neq 0$) prior to the Release task and therefore, can be allocated that task. During the disassembly, the system properly allocated all the Grasp and Release tasks to the machine since it had knowledge about the tasks and the characteristics of all the objects. A series of unexpected events occurred, however, which allowed us to further test the Automated Monitor: the large pins almost always slipped out of the gripper when attempting to grasp them because of their deep position in their hole and their small size compared to the accuracy of the arm. The 'object_not_grasped' problem was correctly detected and the corresponding recovery plan (which had been a priori input to the knowledge bases in these experiments) proposed. Among the few pins that were successfully grasped, many fell out of the gripper while being moved, which also correctly triggered the detection of and recovery plan proposal for the 'dropped_object' problem. Finally, in the reassembly phase, the Grasp and Release tasks were properly assigned to the machine from the onset of execution.

6. Conclusions

A conceptual architecture for enhancing the synergy between humans and machines in complex cooperative systems has been presented. The architecture calls for many modules spanning sev-

eral disciplines of robotics and intelligent systems to coalesce in a highly fluid and dynamic fashion. In particular, the modules responsible for planning the job, monitoring the execution, and allocating tasks to the human or the machine based on their respective and *current* knowledge, skills and availability, and the current execution conditions, need to be tightly coordinated. Some detailed descriptions of the methodologies underlying these modules have been presented, and their operation in proof-of-principle implementations and experiments illustrating the conceptual integrated functioning of the system have been discussed in the context of synergy between a human operator and a robotic manipulator arm. Although much further research and development are needed and will probably continue over the coming decades on this general topic of human–robot synergy, the initial implementation and preliminary experiments presented here illustrate the feasibility of the proposed architecture while clearly pointing out the areas in need of development for future practical implementations. The experience and insight gained with these initial investigations have also provided us with clear indications of what levels of synergism could be envisionable today with the current technologies in machine learning, dynamic task planning and allocation, and error detection and recovery. We expect that significant developments in these three areas will occur in the few years to come, thus greatly benefiting the human-robot synergy area, as they also appear as the central deficiencies in many other robotics problems of near term interests.

Acknowledgments

The authors would like to express their thanks to the many people who have contributed directly or indirectly to the development reported here and in particular, R. Glassell, S. Hruska, M. Kedl, R. Kress, E.M. Oblow, C. Steidley, and D. Thompson for their help with the Human-Machine synergy modules and the CESARm testbed.

Appendix

Example job planning operators

Place(*object,*loc)	PRECONDITIONS:	Grasped(*object) Found(*loc) At(Hand,*loc > Hover_pos) END
	DELETE_LIST:	END
	ADD_LIST:	Positioned(*object,*loc) END
Grasp(*object)	PRECONDITIONS:	Handempty Found(*object) At(Hand,*object:Hover_pos) END
	DELETE_LIST:	Handempty Inserted(*object,-) Positioned(*object,-) END
	ADD_LIST:	Grasped(*object) END
Insert(*object,*loc)	PRECONDITIONS:	Grasped(*object) Found(*loc > Hover_pos) At(Hand,*loc > Hover_pos) END
	DELETE_LIST:	END
	ADD_LIST:	Inserted(*object,*loc) END
Move_Arm(Curr_Loc,*to_loc)	PRECONDITIONS:	END
	DELETE_LIST:	At(Hand,-) END
	ADD_LIST:	At(Hand,*to_loc) END
Release(*object)	PRECONDITIONS:	Grasped(*object) END
	DELETE_LIST:	Grasped(*object) END
	ADD_LIST:	Handempty END
Find(*object)	PRECONDITIONS:	END
	DELETE_LIST:	END
	ADD_LIST:	Found(*object) END

Example of job plan for Cranfield assembly

Plan number	*Task description*
1	Find(Casing1)
2	Move_Arm(Curr_Loc,Casing1:Hover_pos)
3	Grasp(Casing1)
4	Find(Jig_lower_center)
5	Move_Arm(Curr_Loc,Jig_lower_center > Hover_pos)
6	Place(Casing1,Jig_lower_center)
7	Release(Casing1)
8	Find(Lever)
9	Move_Arm(Curr_Loc,Lever:Hover_pos)
10	Grasp(Lever)
11	Find(Jig_axis)
12	Move_Arm(Curr_Loc,Jig_axis > Hover_pos)
13	Place(Lever,Jig_axis)
14	Release(Lever)
15	Find(Spacer)
16	Move_Arm(Curr_Loc,Spacer:Hover_pos)
17	Grasp(Spacer)
18	Find(Casing_1_bottom_edge)
19	Move_Arm(Curr_Loc,Casing_1_bottom_edge > Hover_pos)
20	Place(Spacer,Casing_1_bottom_edge)
21	Release(Spacer)
22	Find(Large_pin_1)
23	Move_Arm(Curr_Loc,Large_pin_1:Hover_pos)
24	Grasp(Large_pin_1)
25	Find(Casing_1_bottom_left_hole > Hover_pos)
26	Move_Arm(Curr_Loc,Casing_1_bottom_left_hole > Hover_pos)
27	Place(Large_pin_1,Casing_1_bottom_left_hole)
28	Release(Large_pin_1)
29	Find(Large_pin_2)
30	Move_Arm(Curr_Loc,Large_pin_2:Hover_pos)
31	Grasp(Large_pin_2)
32	Find(Casing_1_bottom_right_hole > Hover_pos)
33	Move_Arm(Curr_Loc,Casing_1_bottom_right_hole > Hover_pos)
34	Insert(Large_pin_2,Casing_1_bottom_right_hole)
35	Release(Large_pin_2)
...	
113	Find(Small_pin_8)
114	Move_Arm(Curr_Loc,Small_pin_8:Hover_pos)
115	Grasp(Small_pin_8)
116	Find(Casing_2_top_side_right_hole > Hover_pos)
117	Move_Arm(Curr_Loc,Casing_2_top_side_right_hole > Hover_pos)
118	Insert(Small_pin_8,Casing_2_top_side_right_hole)
119	Complete_Assembly(Benchmark)

Sample execution monitoring tables for the cranfield assembly task

...

Subtask: Place(Casing1,Jig_lower_center)
Preconditions: Grasped(Casing1)
Found(Jig_lower_center)
At(Hand,Jig_lower_center > Hover_pos)

Continuing Conditions:

Subtask: Release(Casing1)
Preconditions: Grasped(Casing1)

Continuing Conditions: Positioned(Casing1,Jig_lower_center)

Subtask: Find(Lever)
Preconditions:

Continuing Conditions: Handempty
Positioned(Casing1,Jig_lower_center)

Subtask: Move_Arm(Curr Loc,Lever:Hover_pos)
Preconditions:

Continuing Conditions: Found(Lever)
Handempty
Positioned(Casing1,Jig_lower_center)

Subtask: Grasp(Lever)
Preconditions: Handempty
Found(Lever)
At(Hand,Lever:Hover_pos)

Continuing Conditions: Positioned(Casing1,Jig_lower_center)

Subtask: Find(Jig_axis)
Preconditions:

Continuing Conditions: Grasped(Lever)
Positioned(Casing1,Jig_lower_center)

Subtask: Move_Arm(Curr_Loc,Jig_axis > Hover_pos)
Preconditions:

Continuing Conditions: Found(Jig_axis)
Grasped(Lever)
Positioned(Casing1,Jig_lower_center)

...

References

[1] K.J.W. Craik, Theory of the human operator in control systems: I. The operator as an engineering system, *Brit. J. Psychol.* 38 (1947) 56–61.

[2] K.J.W. Craik, Theory of the human operator in control systems: II. Man as an element in a control system, *Brit. J. Psychol.* 38 (1947) 142–148.

[3] P.M. Fitts, Functions of men in complex systems, *Aerospace Engineering* 21 (1962) 34–39.

[4] N. Jordan, Allocation of function between man and machines in automated systems, *J. of Applied Psychology* 47 (1963) 161–165.

[5] D. Whitfield, Human skill as a determinate of allocation of function, *Ergonomics* 10 (2) (1967) 154–160.

[6] A. Chapanis, On the allocation of functions between men and machines, *Occupational Psychology* 39 (1965) 1–11.

[7] E. Edwards and F.P. Lees, *The Human Operator in Process Control* (Taylor and Francis, London, 1974).

[8] A.D. Swain and H.F. Guttman, *Handbook of Human Reliability Analysis with the Emphasis on Nuclear Power Plant Applications* (NRC NUREG-CR-1278), (Sandia Laboratories, Albuquerque, NM, 1980).

[9] *Proc. Workshop on Human–Machine Symbiotic Systems*, Oak Ridge, TN (1988) Proc. No. ORAU 89/C-140 (Oak Ridge Associated Universities, P.O. Box 62, Badger Ave., Oak Ridge, TN 37831).

[10] T. Sato and S. Hirai, MEISTER: A model enhanced intelligent and skillful teleoperational robot system, *Proc. 4th International Symposium on Robotics Research* (1987) 1–8.

[11] T. Sato and S. Hirai, Motion understanding and structured DD master–slave manipulator for cooperative teleoperator, *Proc. 3rd International Conference on Advanced Robotics* (ICAR '87), Versailles, France (1987) 112–124.

[12] H.E. Price, The allocation of functions in systems, *Human Factors* 27 (1) (1985) 33–45.

[13] H.E. Price, R.E. Maisano and H.P. van Cott, *The Allocation of Functions in Man–Machine Systems: A Perspective and Literature Review*, NUREG-CR-2623, Oak Ridge National Laboratory, ORNL/SUB/81-9027/1 (1982) (NTIS. 5285 Port Royal Rd., Springfield, VA 22161).

[14] R.E. Fikes and N.J. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence* 2 (1971) 189–208.

[15] R.E. Fikes, P.E. Hart and N.J. Nilsson, Learning and executing generalized robot plans, *Artificial Intelligence* 3 (1972) 251–288.

[16] E.D. Sacerdoti, *A Structure for Plans and Behavior* (Elsevier, New York, 1977).

[17] M. Stefik, Planning with constraints (MOLGEN: Part 1), *Artificial Intelligence* 16 (1981) 111–140.

[18] M. Stefik, Planning and meta-planning (MOLGEN: Part 2), *Artificial Intelligence* 16 (1981) 141–170.

[19] S.A. Vere, Planning in time: Windows and durations for activities and goals, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 5 (3) (1983) 246–267.

[20] D.E. Wilkins, Domain-independent planning: Representation and plan generation, *Artificial Intelligence* 22 (1984) 269–301.

[21] K. Collins, A.M. Palmer, and K. Rathmill, The development of a European benchmark for the comparison of assembly robot programming systems, *Proc. June 1984 Robots Europe Conference*, Brussels (Springer-Verlag, Berlin, 1985).

[22] L.E. Parker, *Job Planning and Execution Monitoring for a Human–Robot Symbiotic System*, ORNL/TM-11308 (November 1989) (NTIS, 5285 Port Royal Rd., Springfield, VA 22161).

[23] F.G. Pin, P.F.R. Belmans, S. Hruska, C. Steidley and L.E. Parker, Robotic learning from distributed sensory sources, *IEEE Transactions on Systems, Man, and Cybernetics* 21 (5) (1991) 1216–1223.

[24] R.E. Fikes, Monitored execution of robot plans produced by STRIPS, *Proc. of IFIP Congress 71* (1971).

[25] M.-Y. Chern, An efficient scheme for monitoring sensory conditions in robot systems, *Proc. IEEE International Conference on Robotics*, Atlanta (1984) 298–303.

[26] M. Gini, Symbolic and qualitative reasoning for error recovery in robot programs, in: U. Reubeld and K. Hoermeah, eds., *Languages for Sensor-Based Control in Robotics* (Springer-Verlag, Berlin, 1987).

[27] S.Y. Harmon, Dynamic task allocation and execution monitoring in teams of cooperating humans and robots, *Proc. 1988 Workshop on Human–Machine Symbiotic Systems*, ORAU 89/C-140, CESAR-89/19, Oak Ridge, TN (1988).

[28] W.B. Rouse, Human–computer interaction in the control of dynamic systems, *Computing Surveys* 13 (1) (1979) 71–99.

[29] L.E. Parker and F.G. Pin, A methodology for dynamic task allocation in a man–machine system, in: Z.W. Ras and M. Zemankova, eds., *Methodologies for Intelligent Systems* (Elsevier, New York, 1987) 488–495.

[30] F.W. DePiero, W.W. Manges, R.L. Kress, M.R. Kedl and W.R. Hamel, Architecture for a human–robot symbiotic system, *Proc. 22nd Southeastern Symposium on System Theory*, Cookeville, TN (1990) 78–83.