An Overview of KALI: a System to Program and Control Cooperative Manipulators

Vincent Hayward
Laeeque Daneshmend

McGill Research Center for Intelligent Machines
3480 University Street, Montréal, Québec Canada H3A 2A7

Samad Hayati

Jet Propulsion Laboratory, Robotics and Teleoperators group
4800 Oak Grove Drive, Pasadena, California 91109, U.S.A.

Summary

A software and hardware system, called Kali, for programming and controlling cooperative manipulators is described. It has been designed at McGill University in a collaborative effort with the Jet Propulsion Laboratory. A set of programming primitives which permit a programmer, human or automated, to specify cooperative tasks are first outlined. In the context of cooperative robots, trajectory generation issues are discussed and our implementation briefly described. Software engineering for system integration is also discussed. Finally, the paper describes the allocation of various computational tasks among the elements of a multi-processor computer. Target applications presently envisioned include space robotics, power line maintenance, and other resource industry applications.

1 Introduction

Since the earliest attempts,[1] the goal of designing a well structured robot programming language with accurate semantics has always eluded researchers. Robot languages must account for a hierarchy of models and abstractions.[2,3] A possible decomposition is outlined here. At the highest level, the system and its task can be described in terms of formal logic, for which a *grain size* at which such a description is appropriate must be chosen. At a lower level of abstraction, it is possible to use automata models, in which robot tasks are described in terms of *state changes*. At an even lower level, robot systems can be described in terms of *processes* which have a finite life time and which explicitly deal with the notion of time. At the lowest levels, descriptions can be made in terms of continuous functions (i.e. kinematics) and feed-back control. Thus, many representations, some of them of a geometric nature, must be explicitly or implicitly

---

In Hindu mythology, Kali the Divine Mother, is often represented as a creature with many arms.

Fourth International Conference on Advanced Robotics.

K. J. Waldron (ed.), *Advanced Robotics: 1989*

included in robot programs. This multiplicity of representations, and the need to integrate various types of sensing and acting modalities, has hindered the development of general purpose robot programming languages.

The programming and control of multiple cooperative robot manipulators represents a significant step in complexity and computational requirements in comparison to that of a single robot. Even though the essence of the problem remains the same, some concepts developed in the context of a single manipulator which fail to extend to the case of several manipulators clearly show their weaknesses. The development of Kali is helping to pinpoint some of these concepts.

In view of these considerations, we have designed a robot controller to conduct research in robot languages which can handle multiple manipulators, and which is based on an open architecture, both from the software and the hardware point of view. Kali addresses the lowests levels of representation hierarchy outlined above: feedback control, processes, and state changes.

The implementation of this controller is in its final stage. It is taking place simultaneously at McGill University and at the Jet Propulsion Laboratory. A portion of the design is based on the RCCL system,[4,5] which now has led to numerous projects,[6,7,8,9] or inspired others.[10] The design philosophy of Kali is different from industrial systems specifically designed for off-line programming,[11] in that we insist on implementing on-line control methods to provide for experimental sensor integration and provide a base for advanced telerobotic systems.[12]

The open architecture concept allows us to treat questions such as the user interface, tasks simulators, and programming aids as *applications* and not motivations of our software.

In this paper, we shall present the basic algorithms that we have implemented to achieve programming of coordinated multiple manipulators, the software structure that embodies them, and the hardware configuration that runs them.

2 Algorithms

Instead of designing a robot control language, we provide the user, human or automated, a collection of algorithms accessible through a standard interface. At the present time, as in RCCL, this interface consists of a set of functions and global variables. Another interface for future development will consist of a set "objects" in the sense of object-oriented computer programs implemented using Smalltalk, for example.

Several basic algorithms concerned with the abstract notion of "motion system" form the core of Kali. A motion system is an instance of a collection of actuated joints and links with known dynamic and kinematic properties whose behavior can be described by a motion descriptor specifying a number of constraints such as: velocity, time of arrival, maximum wander, etc...

It is up to the programmer to decide what constitutes a motion system. Most of the time, motion systems will be such that their overall kinematic mobility is six. For example one manipulator, or two manipulators rigidly connected through a common load. Of course, this is always a matter of appreciation. It is not easy to tell the mobility of system if one accounts for flexibility. It also depends on the nature and the capabilities of the underlying feed-back control.

A motion system can also have a mobility smaller than six. Such a case occurs, for example, when a manipulator is controlled to perform a compliant motion. In this

case, it is possible to consider that the manipulator looses some degrees of mobility due to natural constraints. This case is handled in Kali by including in the geometric description of the motion a representation of the "difference" between the ideal motion and the actual one. A similar case occurs when the controlled mechanism becomes kinematically singular.

We also need to consider the case when a motion systems has more than six mechanical freedoms. It is legitimate to consider a "redundant manipulator" as a single motion system: If some implicit constraint is given to resolve the kinematic redundancy, the property of redundancy is then hidden from the programmer.

The consideration of all the constraints contributing to the execution of a task must lead to the decision of what constitute a motion system, to a certain extent, independently from the available control algorithms. For example, consider the task of carrying a common load with two manipulators. It is clear, according to the above definition, that when the two manipulators reach for the grasp position, they must be considered as independent motion systems. At grasping time, as well as when the load is lifted from its support, the *topology* of the mechanical system varies, and if the grasps are sufficiently rigid, we will probably choose to merge the two manipulators and their load into a single motion system to which task constraints can be applied. It should be noted that these abstractions can be made independent from the servo-control algorithms which can be employed to achieve correct behavior.

It is the objective of Kali to capture these abstractions at geometric and trajectory level and to provide sufficient computing power to implement advanced control methods. We consider it convenient to classify the constraints which contribute to the execution of a task into three groups:

- *The task constraints* are objectives which have to met for the task to execute satisfactorily: to be (or not to be) at a place at a given time, with a given velocity, or exerting a given force, etc...)
- *The manipulator constraints* depend on the properties of the mechanism (reach, dexterity, manipulability, etc...)
- *Design constraints* are imposed by the designer to enforce some kind of performance. They usually concern optimality criteria such as: minimum time motions, minimum energy, minimum structural stress caused by accelerations and jerk, etc...

As appropriate to a particular case, motion systems can include one manipulator, several manipulators, or other types of equipment such as grippers, hands or micro-manipulators as commented in a previous paper.[13]

2.1 Spatial Relationships

We observed that the essence of manipulator programming consists of specifying time-varying kinematic relationships designed to satisfy various types of constraints.

Positions are conveniently described by frame transformation graphs. As in RCCL, those graphs adopt a ring structure (see figure 1). Graphs have two kind of nodes: *input nodes* and *output nodes*. Input nodes take their values from information contained within the programming system or from sensor data. Output nodes specify the nominal position of a controlled set of joints and links. It can be shown that no matter how complex the kinematic loops are, their associated graphs must have the following properties to be proper:

- Being connected;
- All nodes must belong to at least one closed path;
- Each output node must be solvable, that is belong to a closed path containing only input nodes, or output nodes which are also solvable;
- There is no closed path with no output node.

In fact, any proper graph can be transformed into an equivalent set of closed paths, called 'rings', if the nodes are duplicated but share the same values.

The graph shown figure 1 is equivalent to the following equation:

$$\mathbf{M\ T\ D\ C} = \text{Identity}$$

where **M** represents the "manipulator transform" (an output node), **T** the transformation from the manipulator's last link to the controlled frame, the "tool transform" (an input node), and **C** the coordinate transformation from where the tool should reach, to where the robot is located. The transform **D** or "drive transform" (an input node) has an initial value that reflects the position of the arm before the motion begins, and is interpolated toward the unit transform in order to produce the desired motion. Such a graph is set up by one position making primitive.

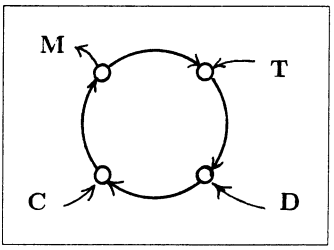

Fig. 1

For example, with two manipulators, the rings may share common transformation frames (see figure 2). The transform **C** represents the transformation relating the base of one manipulator to the other. This illustrate the case of having two manipulators rigidly connected and moving independently.

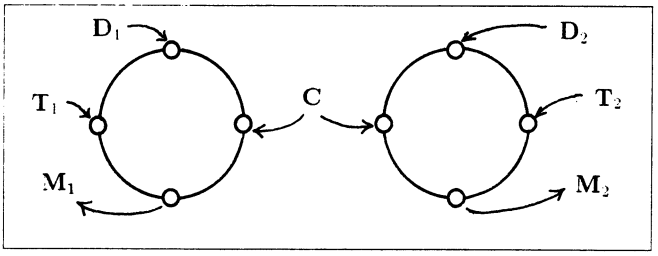

Fig. 2

On figure 3, which illustrates the case of manipulators sharing a load, $\mathbf{M}_1$ and $\mathbf{M}_2$ are manipulator transforms, **D** is the "drive transform" which is unity upon motion termination, such that both position equations are satisfied.

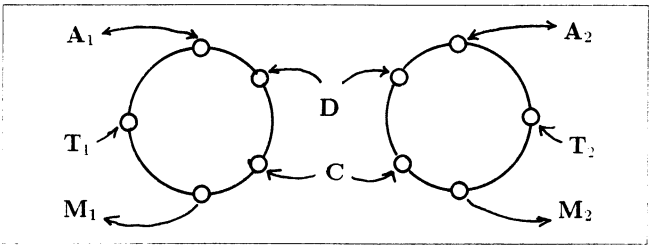

Fig. 3

All loops must multiply out as the identity transforms. Of course, because of the uncertainty in the models and the servo errors, it is unlikely that a loop containing both manipulators will have this property. Therefore we introduce the transforms, $\mathbf{A}_1$ and $\mathbf{A}_2$, in order to take into account the discrepancies. During accommodation (regardless on how it is achieved), the values of $\mathbf{A}_1$ and $\mathbf{A}_2$ will vary slightly. Accommodation can be performed while both arms are position servoed by varying the values until mutual forces are canceled. Their values can also result from the readings of the positions of the manipulators if the servo algorithm is able to self adapt. In the latter case, the procedure is analogous to compliant motion.

The next example illustrate the case of a manipulator attached to the end of another one (Figure 4). A position is specified by indicating which of the frames is the "controlled frame" and to which applies the motion constraints. The total number of position controlled degrees of freedom remains always six. The various transforms are evaluated and contribute to the nominal values of the manipulator transforms. Arbitrary graphs can be created by means of several loops. The nodes of several loops may point to the same transform, in order to express mutual relationships.

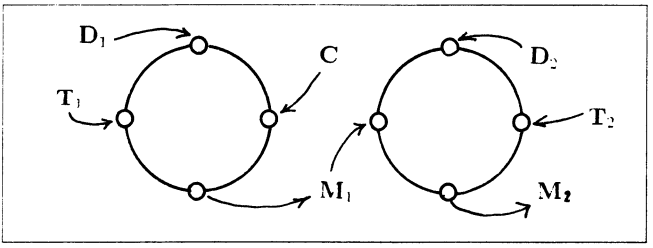

Fig. 4

The order of evaluation of the functions bound to the transforms is specifiable by the user. The normal order is: sensor-based functions, path-planning functions, and update of the manipulator transforms.

2.2 Basic Path Planning

A trajectory is viewed as a string of path segments connected by transitions. It is assumed that the velocity of the controlled frame is the variable of concern during the traversal of path segments. Accelerations are supposed to be small because the direction of the velocity should not change abruptly. On the other hand, during transitions, the acceleration is the variable of concern because of the velocity change. As a consequence

the path must be allowed to wander off the ideal trajectory or the manipulator brought to a stop.

We have developed a transition computation method based on the blending of successive path segments. The type of blend is controlled by two factors. The *preview* factor conveys the amount of look ahead the system must perform before a transition and control the shape of the wander. The *acceleration* factor conveys the amount of admissible trajectory wander. These two factors, and the knowledge of the dynamics of the system lead to the automatic determination of the transition time. A smaller wander will lead to a longer acceleration period. This method is quite robust and is not affected by ill-defined trajectories, such as those produced by tracking sensory data, since it does not rely on boundary conditions in position, velocity, and acceleration.[14] The interpolation of rotations relies on quaternion algebra.

Of course, this method does not produce "optimal" trajectories, but provide a realistic account of what is needed for general purpose manipulation.

2.3 Advanced Trajectory Planning

The purpose of an accommodation or optimization procedure is to modify the nominal linear Cartesian coordinates trajectory in order to better suit the manipulators or the tasks, according to some design criterion. This may be achieved in two different and possibly combined ways. Either interpolation along the path is produced by using a non-linear time scale factor, causing the manipulator to accelerate and decelerate by demand. or the path is modified. In most cases, we shall attempt to satisfy the *manipulator constraints:* actuator torque and range bounds, or obtain motions that minimize energy, consumption, wear, etc... A variety indexes can be exploited.

In the first case, the user has the possibility to specify an arbitrary time scale function. In the second case, the transform equations describing the motions can be written as follows:

$$\mathbf{M}_i \; \mathbf{O}_i \; \mathbf{T}_i \; \mathbf{C}_i \; = \; \text{Identity}.$$

In these equations, the $\mathbf{O}_i$'s stand for path modifiers.

As far as the basic path planning process is concerned, the only requirement is that the quasi-linearity hypothesis remains valid, *i.e.*, the path modifications are not too drastic. A simple example of an optimizer module would be a process to calculate the deviation such that the joint variables vary linearly. This method produces "joint mode" motions while not requiring the handling of special cases in the trajectory generation code. This is particularly useful in the context of multi-manipulator systems for which the notion of joint interpolated motions breaks down.

Similarly, during compliant motions, such accommodation procedure have the purpose to take into account the discrepancies between the programmed trajectory and the actual trajectory as a result of the geometrical constraints.

This procedure can also be used to handle singularities. For example, the configuration of the manipulator may be observed from a kinematic view point. If the manipulator moves next to or close to a singular point, the concerned joint or joints is or are brought to rest. The resulting path discrepancy is then included in the position equation. Thus this procedure allows the arm to move gracefully through points of singularity. Similarly, collision avoidance algorithms can be incorporated into the system in a structured fashion.

2.4 Synchronization

Motions are treated as processes, they are created, go through a sequence of states, and are eliminated from the system after a while.

Each motion is associated to an identification number (`id = move(...)`) which allows the controlling program to perform further references to this particular motion (such as: `suspend(id)`). Since motions are treated as individual processes, synchronization between motions themselves is also easy to achieve through the combined use of motion control flags and motion parameters such as velocity or time of arrival.[15]

As in RCCL, "robot programs" are encoded in a 'user process' which runs asynchronously with respect to the actual arm motions. This process is provided with all the means necessary to synchronize itself with the motions, or the motions with itself.

2.5 Control

At the present time, several schemes are under consideration. Among those, hybrid control schemes with feed-forward dynamic decoupling will be evaluated.[16] Possible methods for computing the robot dynamics and kinematics are provided in the McGill implementation.[17,18]

Adaptive control is believed to provide us with a means to cope with the difficulty in the modeling of such a complex system as several cooperating manipulators.[19] Hybrid force/position control is an essential attribute for true cooperative manipulation.[20,21,22] Force control is necessary even for the case of the pure transport problem, where the object being manipulated is unconstrained by the environment, since internal forces need to be regulated.

Under ideal conditions, i.e.

- structurally accurate dynamic and kinematic models of the manipulators, the object to be manipulated, and the environment which constrains motion (if any),
- accurate measures of model parameters;
- rigid body dynamics for all components of the system, and rigid contact between the manipulators and the object,

the dynamic decoupling scheme of Hayati[16] can be shown (in theoretical analysis and simulation) to provide stable decoupled task space control.

We have formulated a discrete-time single-input/single-output (SISO) adaptive control scheme to cope with sensor compliance and non-rigid body dynamics of the manipulators and load. The scheme relies upon dynamic decoupling of the task-space dynamics using a feed-forward model. This scheme has performed well in simulation studies.[19] The SISO nature of the adaptive controller lends itself to superposition on a multiple coordinated manipulator control scheme, and to parallel implementation in the KALI multi-processor environment.

Practical aspects of this adaptive scheme requiring experimental investigation include: noise-filtering of force measurements, "jacketing" software to initiate and terminate the adaptive loop and choose initial parameter estimates, and the effect of actuator bandwidth limitations.

3 Software

The design goal of the software is to create an extensible collection of algorithms providing a true research tool. New algorithms can be designed to replace existing ones or to augment them. Programs, coded in the C language, are grouped into several utility libraries in a modular fashion. The very structure of the software design permits implementors to interchange such complex functions as trajectory generators.

Not counting the real-time operating system level, the software is divided in roughly five layers which may consist each of several libraries. Note that these layers do not necessarily reflect the control layers.

The first layer, known as MUX (McGill U. Extensions) runs on top of a commercial real time multi-processor operating system called VxWorks.* The function of this software layer is to establish an environment suitable for running various synchronous and asynchronous processes. This operating system interface is a set of functions and global variables that represent, in a generic fashion, the system services necessary to run robot control code: shared memory management, and a "wall-clock" facility.[23]

The second software layer is also a support layer and is independent from the operating system. It consists of several small utility libraries. Currently there is a library for buffered input and output of data, useful for debugging and dumping data into files. The 'geo' library is a set of functions for geometrical computations on vectors, transforms, and quaternions. This layer also includes kinematic and dynamic models for our Puma robots.

The third software layer implements the servo control code and uses the above layers.

The fourth layer is the heart of Kali. It consists of two libraries. The 'rings' library contain code to maintain and update in real time kinematics loops. As discussed earlier, loops are oriented graphs whose nodes point to transformation values. Each node is attached to a function which specifies the 'method' to update the value. Provision is made for the same function to be attached to several nodes. Also, if a value is shared by several loops, the update will take place only once. Intermediate values, results of transform multiplies, are stored into an internal hash table in order to avoid redundant computations. There is no need to keep track explicitly of the fact that the same kinematic relationship may be used several times. The other library, the 'cmotion' library, contains the 'method' to compute smooth interpolated Cartesian coordinate trajectories according to a variety of constraints specified in a 'motion record'. Robot programs using the 'cmotion' library can be extremely verbose.

Each motion is treated as a process. As it is common practice in software engineering, a process, when created, is represented as a record which remains in existence for an extended length of time in the system by means of a ring buffer.

The fifth layer is application dependent. It may consist of user level functions to simplify task descriptions. For example, functions specific to dual-arm control can be developed at this level. It may also consist of communication software to make the Kali primitives accessible from a host machine running more advanced software development tools.

4 Run Time Structure

The run-time task structure has been devised in such a way that few and simple task synchronization mechanisms are needed because efficiency considerations prevail. These

* A trademark of Wind Rivers Systems Inc.

mechanisms are easily implemented using most general purpose real-time operating systems.

The run-time structure consists of a set of processes, some of which are high priority synchronous processes, some others are low priority and may execute asynchronously. The processors are distributed over an array of processors connected by a bus.

The task allocation reflects the attempt to minimize bus traffic, and synchronized inter-process communications. Short delays are paramount to obtain adequate control. In many other proposed architectures, pipelining is seen as a method for improving control by augmenting the computational through-put. Unfortunately, this approach fails to take into account that the benefits of high rates are often lost in the computations delays spent in the stages of a pipelined architecture which cause the correction signals to be computed on stale data.

To improve the control rate, we adopt a different approach based on the consideration of the physics of the plant to be controlled, on the structure of the control algorithm,[24] [17] and the evolution of today's computing technology. First, rapidly changing quantities (control error, for example) are updated more often than slowly changing ones (inertia characteristics, for example). Second, parallelism is achieved by observing that certain computations, within one sample period, can be performed independently from others, and by allocating them on a limited number of concurrently running processors. Although it has been observed that a great deal of parallelism can be achieved in this fashion at the cost of great hardware complexity, we have preferred to make use make a limited use of it in favor of simplicity. Finally, the technology of processors is rapidly improving performance and we base our design on conservative projections.

4.1 Synchronous Processes

As in RCCL, there exists a main synchronous process whose task is to compute nominal set-points for the manipulators. The main function calls n instances of a function called `switcher()`, one per motion system with their associated environment. The user's monitor functions are invoked first, then the motion queue is examined and book keeping operations pertaining to the state of the active motions are performed. Finally, a call to the function `valuate()` is made in order to invoke, in the requested order, all the functions bound to active kinematic loops. Of course, among them, there will be the Cartesian motion generator and manipulator motion generators.

Others synchronous processes implement the servo control algorithm. There is also also a synchronous I/O process which runs at the same frequency as the servo process. This process is in charge of gathering sensor information: joint position, current, wrist force readings, etc... and post them in a shared area of memory.

4.2 Asynchronous processes

The main asynchronous process is the so-called 'user process'. It is the process that contains the 'robot program'. Its mains functions are: presetting the transformation values, setting up the kinematic loops, issuing the motion requests, synchronizing itself with the task execution, and performing I/O with the external world.

The other asynchronous processes are related to the dynamic computations. These processes can run asynchronously because the performance of the system will degrade only slowly if the data they produce is a little bit 'old'.[25] Four of these processes are

needed per robot. The first one computes three sets of forces created by the velocity terms under various conditions: before and after a potential transition, and for the current set-point. The second one compute the current gravity terms. The third one updates the inertia matrix. One other compute the maximum force the robot can produce.

5 Hardware

We selected five Heurikon single board computers with the Motorola MC68020 and floating point coprocessor both running at 20 MHz and 1 Mbyte of static no wait state operation. These processors communicate over a common VME backplane. The development is done in C and programs are created on a SUN workstation under Unix* connected to the VME backplane via Ethernet. To further off load VME bus traffic, we have selected a system which features the VSB secondary bus. The VSB bus serves to access the shared memory for all asynchronous communications. Further details of the hardware requirements and implementation have been presented.[23] The use of the VxWoks operating system provides for an easy upgrade (68030, SPARC). Presently, the initial version of Kali at McGill University can run up to 1 KHz sampling rate.

6 Conclusion

Kali is an attempt at creating an integrated manipulator programming and control system. A significant amount of care has been exercised to set up an open environment from the software and hardware point of view. For example, we consider using Kali for controlling manipulators, walking machines, or dexterous hands. In each of these cases, relevant common programming primitives have been identified and implemented. Another implementation, under development, utilizes the Harmony operating system.[26] Several other implementations are currently under consideration.

7 Acknowledgments

Credit must be given to Tony Topper, Ajit Nilakantan, John Lloyd and Ron Kurtz for their contributions to the project.

The research described in this paper has been funded for the largest part by the Jet Propulsion Laboratory under contract with the National Aeronautics and Space Administration. Support through grants from the Natural Sciences and Engineering Research Council of Canada (NSERC), and a contract with the National Research Council of Canada (NRC), is also gratefully acknowledged.

8 References

1. Paul, R. P. 1972. Modeling, trajectory calculation, and servoing of a computer controlled arm. *A.I. Memo 177*, Stanford Artificial Intelligence Laboratory, Stanford.

2. Hayward, V., Paul, R. P., 1984 (June). Robot control and computer languages. *Fifth CISM-IFToMM Symposium on Theory and Practice of Robot Manipulators*, Udine, Italy.

---

* Trademark of AT&T

3. Hayward, V. 1988. Autonomous control issues in a telerobot. *IEEE Conference on systems man and cybernetics*, Beijin, China.

4. Hayward, V., Paul, R. P. 1987. Robot manipulator control under Unix: RCCL a robot control 'C' library. *International Journal of Robotic Research*, 5(4).

5. Hayward, V., Paul, R. P. 1984 Introduction to RCCL: A robot control 'C' library. *Proc. of First IEEE Conference on Robotics*, Atlanta, GA.

6. Lloyd, J., Parker, M., McClain, R. 1988. Extending the RCCL environment to multiple robots and processors. *Proc. 1988 IEEE Int. Conf. on Robotics and Automation*, Philadephia, Pa.

7. Guptill, R., Stahura, P. 1987. Multiple robotics devices: Position specification and coordination. *Proc. 1987 IEEE Int. Conf. on Robotics and Automation.* Raleigh, NC.

8. Lee, J. S., Hayati, S., Hayward, V., Lloyd, J. E. 1986 Implementation of robot control C library on the micro vax II. *Advances in Intelligent Robotics Systems, SPIE's Cambridge Symposium on Optical and Optoelectronic Engineering*, Cambridge, Ma.

9. Kossman, D., Malowany, A. 1987. A multi-processor Robot control system for RCCL under iRMX. *IEEE Int. Conf. on Robotics and Automation*, Raleigh, Pa.

10. Trevelyan, J. P., Nelson, M. 1988. Adaptive motion sequencing for process robots. In *Robotic Research, the Fourth International Symposium*, MIT Press.

11. Craig, J. J. 1988. Issues in the design of off-line programming systems. In *Robotic Research, the Fourth International Symposium*, MIT Press.

12. Backes, P., Hayati, S., Hayward, V., Tso, Kam. 1989. The Kali multi-arm robot programming and control environment. *1989 NASA Conference on Space Telerobotics*, Pasadena, Ca.

13. Hayward, V., Hayati, S. 1988. Kali: An Environment for the Programming and Control of Cooperative Manipulators. *1988 American Control Conference*, Atlanta, Ga.

14. Hayward, V., Daneshmend, L., Nilakantan, A. 1988. Model based trajectory planning using preview. *SPIE Conference, Space Automation IV*, Cambridge, Ma.

15. Nilakantan, A., Hayward V. 1988. Synchronizing Multiple Manipulators. In *Robotics and Manufacturing, Recent Trends in Research, Education and Applications*, ASME Press.

16. Hayati, S. 1986 (San Francisco, April). Hybrid position/force control of multi-arm cooperating robots. *IEEE International Conference on Robotics and Automation*, pp. 82–89.

17. Izaguirre, A., Hashimoto, M., Paul, R. P., Hayward, V. 1988. A new computational structure for real-time dynamics. *International Journal of Robotic Research.* (to appear).

18. Lloyd, E. J., Hayward, V. 1988. Kinematics of common industrial robots. *Robotics*, North-Holland, Vol. 4.

19. Daneshmend, L., Hayward, V. 1988. Adaptation in the control of multiple coordinated manipulators. In *Robotics and Manufacturing, Recent Trends in Research, Education and Applications.* ASME Press.

20. Fujii, S., and Kurono, S. 1975. Coordinated Computer Control of a Pair of Manipulators. *Proc. I. Mech. E.*, pp. 411-417.

21. Ishida, T. 1977. Force Control in Coordination of Two Arms. *Proc. 5th IJCAI*, Cambridge, MA, vol. 2, pp. 717-722.

22. Arimoto, S., Miyazaki, F., and Kawamura, S. 1987 Cooperative motion Control of Multiple Robot Arms or Fingers. *Proc. 1987 IEEE Int. Conf. on Robotics and Automation, pp. 1407-1412.*

23. Topper, A., Daneshmend, L., Hayward, V. 1988 (Ottawa, Canada, November). A computing architecture for a multiple robot controller for space applications (Kali project). *Fifth CASI Conference on Astronautics.*

24. Khatib, O., Burdick, J. 1986. Motion and force control of robot manipulators. *IEEE International Conference on Robotics and Automation*, San Francisco, Ca. pp. 1381–1386.

25. Kircanski, N. Kircanski, M., Vukobratovic, M., Timcenko, O. 1986. An approach to development of real time robot models. *IFToMM Symp. ROMANCY,* Krakow.

26. Gentlemen, W. M. 1985. Using the harmony operating system. Technical Report ERB-966 (NRCC No. 24685). National Research Council, Ottawa, Canada.