N90-29060

# THE KALI MULTI-ARM ROBOT PROGRAMMING AND CONTROL ENVIRONMENT

Paul Backes[†], Samad Hayati[†], Vincent Hayward[‡], and Kam Tso[†]

[†]Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

[‡]McGill Research Center for Intelligent Machines
Montréal, Québec Canada

## Abstract

The KALI distributed robot programming and control environment is described within the context of its use in the JPL telerobot project. The purpose of KALI is to provide a flexible robot programming and control environment for coordinated multi-arm robots. Flexibility, both in hardware configuration and software, is desired so that it can be easily modified to test various concepts in robot programming and control, e.g., multi-arm control, force control, sensor integration, teleoperation, and shared control. In the programming environment, user programs written in the C programming language describe trajectories for multiple coordinated manipulators with the aid of KALI function libraries. A system of multiple coordinated manipulators is considered within the programming environment as one motion system. The user plans the trajectory of one controlled Cartesian frame associated with a motion system and describes the positions of the manipulators with respect to that frame. Smooth Cartesian trajectories are achieved through a blending of successive path segments. The manipulator and load dynamics are considered during trajectory generation so that given interface force limits are not exceeded.

## I Introduction

Research and evaluation of robotics concepts are often hindered by difficulty in the implementation process. Current robot programming and control environments have limited task description and implementation capabilities or have hardware/software environments which are difficult to modify, thus making them difficult to use in quickly changing implementation environments. A wide array of research efforts in robotics are presently being pursued at JPL. The present implementation environment, while quite powerful, is difficult to modify to include the added capabilities that the research efforts are producing. This situation has prompted a collaborative effort between JPL and McGill University to create a new robot programming and control environment, Kali, which has simple but powerful task description capabilities and an open modular architecture to provide for evolving capabilities.

A basic goal of the robotics research at JPL is the development of a space telerobot system. The JPL telerobot testbed is used to develop, test, evaluate, and demonstrate state of the art space technology through development of likely space robotics tasks such as satellite grappling and repair and orbital replacement unit changeout. Several subsystems, from low level task execution through sensing, vision, and planning, coordinate efforts to execute the tasks. The Kali system will be used for the low level task execution and sensing part of the JPL telerobot system. At this level many research areas demand a powerful, flexible system for implementation. These research areas include shared control, traded control, multiple arm control, redundant arm control, teleoperation, task space description and control, force/position hybrid control, sensor design and integration, adaptive control methods, flexible arm control, and human factors.

The present robot control environment used for JPL telerobot research is Multi-Arm RCCL which provides for coordinated motion of two arms [1]. This is an extension of the original version of RCCL developed by Hayward at Purdue University [2]. Kali utilizes many of the concepts used in RCCL. Results of work on Kali prior to this paper can be found in [3,4,5,6,7].

## II Standard Kali

Kali is a low level software and hardware environment for trajectory description and control of multiple coordinated machines, e.g., manipulators, walking machines, or robotic hands. The initial Kali system will be implemented specifically for control of multiple coordinated manipulators. At the high level of Kali, a collection of C language functions are provided which allow the user to describe and implement coordinated motion of multiple manipulators using a C language program. The lower level of Kali has processes distributed over multiple processors in a VMEbus environment.

It is desired with the user interface to Kali to provide the user with an environment to, as easily and generally as possible, describe and execute a task which requires the motion of one or multiple coordinated arms. This is done by separating the task description from the specifics of the underlying mechanical system. The user describes the task in task space, i.e., the motion and forces of a Cartesian frame along with task space constraints. Underlying mechanical system constraints can be specified independently from a specific task and can be considered automatically by Kali which removes the burden of the details of the underlying mechanical system from the user.

Dynamics constraints of the underlying mechanical system are considered in trajectory generation. The load limitations of a manipulator can be specified and a trajectory will be determined which does not require the arm to produce forces greater than the limits specified. This requires the user to provide dynamics models of all manipulators and objects.

An important feature of Kali is its modular functionality. Modularity simplifies software and hardware customization resulting in a convenient environment for research or customization for a specific installation. Software modules such as trajectory generation and servo control have defined functionality allowing them to be replaced without altering other parts of the system.

The hardware environment was selected for simplicity of implementation, computation speed, and upward compatibility with advancing processor technology. Kali utilizes a distributed hardware environment which allows for parallelization of computations, simplified sensor integration, and modular hardware functionality.

## II.1 Motion Systems

Motion description in Kali is specified in terms of a motion system. A motion system describes the motion of a Cartesian coordinate frame, the Control frame, and the constraints associated with the Cartesian motion. User task description involves describing the motion of the Control frame in terms of time synchronization, destinations, velocities, and transmitted forces. Coordinated motion of multiple manipulators is achieved by kinematically constraining the manipulators to the Control frame. Multiple motion systems can be described and executed allowing manipulators to operate independently or semi-coordinated by specifying related completion times.

All information describing the motion of a motion system is placed in a motion record and motion records are placed in a queue to be processed on a first-in first-out basis by the trajectory generator. Motions are processed by a finite state model where the motions go through a sequence of states depending on control flags.

## II.2 Spatial Relationships

Spatial relationships in a motion system are described by frame transformation graphs in the form of ring structures. Each ring structure, or loop, in the structure is equivalent to the equation

$$B\ M\ T\ D\ C\ =\ Identity \tag{1}$$

where B represents the manipulator base transform from a fixed world frame to the manipulator base, M represents the manipulator transform, T the tool transform from the manipulator final link to the controlled frame, C the goal position of the control frame, and D the drive transform which is interpolated from an initial value which satisfies the initial state to the identity transform in order to produce the desired motion. Each transformation ring in a motion system is set up by one position making primitive. All loops of a motion system share a common drive transform. Spatially coordinated motion of manipulators which are not part of the same motion system is possible by having common transforms in their ring equations other than the drive transform, e.g., common control frame goal positions. The ring equations allow for additional frames in the loop such as for sensor based motion as described below.

Each transform in the loop has associated with it a bound function which is used to update the transform each sample interval. Standard functions such as those bound to the M and D transforms are supplied by Kali, although the user can specify new ones simply by placing pointers to these in the ring equation. Other functions such as those bound to the A transforms of figure 2 below are supplied by the user to provide for specialized motion such as sensor based motion for compliant control. The order of evaluation of the transform-bound functions is specifiable by the user. The normal order is sensor-based functions (e.g., bound to A above), path planning functions (bound to D above), and lastly, update of the manipulator transform M.

Many types of multiple manipulator motion are possible using Kali. Three examples utilizing one and two motion systems are depicted in figures 1 - 4. In figure 1 the two arms share a common motion system to move an object that both arms are grasping. The figure shows the transformations from the ring equation between coordinate frames which are attached to physical objects. Figures 2 - 4 show only the frame transformation graphs. Figure 2 shows the transformation graph for figure 1 with the addition of the A transforms. The small perturbations caused by the A transforms would be difficult to show in figure 1 which shows the physical objects. Figure 3 depicts the case where two manipulators are moving to grasp a common object. Since the arms have different distances to move, independent motion systems (drive transforms) are used to move the arms. The arms share a common control frame goal position which specifies the grasp points on the object for the arms. Concatenated manipulators using two motion systems such as for a gross-positioning arm with an attached micromanipulator are shown in figure 4.

## II.3 Dynamics

The trajectory generator automatically considers the dynamic constraints of objects associated with frames in the ring equations when determining the parameters of the motion system drive transform. Functions describing the dynamics of the objects, including the manipulators, must be provided by the user. These functions return acceleration, gravity, and velocity terms. Standard dynamic model functions, such as for solid cubes, will be provided by Kali. The user may also specify the maximum force that can be taken at an interface. For example, a manipulator's load limit can be specified in this way and the trajectory generator will produce trajectories which will not violate this limit.

## II.4 Trajectory Specification

Trajectories in Kali are considered as a string of Cartesian linear path segments connected by transitions. Velocity is controlled along a path segment and acceleration is controlled during transitions between path segments. The Cartesian trajectory function generates the motion parameters of a drive transform for a motion system such that the specified spatial, temporal, and dynamic constraints are satisfied. Constraints at the task level are described as part of the motion system by the user. The Kali-supplied trajectory generator is described in [5].

The transition between segments is accomplished by the blending of successive path segments with blending controlled by preview and acceleration factors and accelerations limited by dynamic constraints. The preview factor conveys the amount of look ahead the system must perform before a transition. For example, for a preview factor range of 0 - 1, a value of 0 indicates that the trajectory generator has no knowledge of a new path segment ahead and stays on the current path segment through the goal position. At this time, the start of the next path segment, the transition begins, thus causing an overshoot. For a preview factor of 0.5, trajectory wander off of each of the successive segments is equally permissible during the transition. The acceleration factor conveys the amount of admissible trajectory wander. A small acceleration factor specifies a smaller admissible wander, thus causing a longer transition time. This blending method is robust to ill-defined trajectories since it does not rely on position, velocity, and acceleration boundary conditions.

Other motion system constraints include beginning and ending times, coupling factors, and maximum transmitted forces. Beginning and ending times can be used to synchronize motions of different motion systems. The amount of force transmission between each kinematic relationship is specified with a coupling factor. This factor is normally set to 1, which means

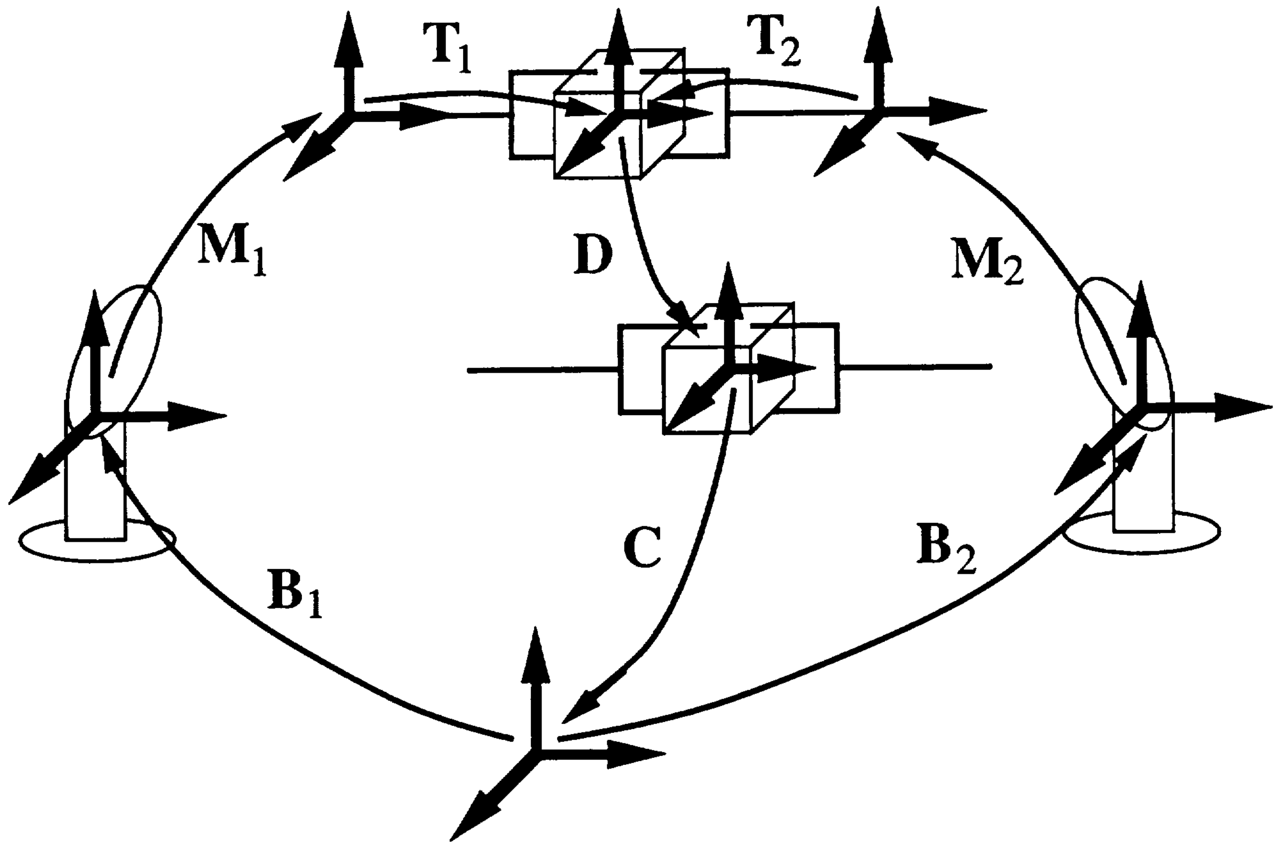

Figure 1: Two arms manipulating a common object

$$\begin{tikzcd}
 & A_1 \arrow[dr] & & A_2 \arrow[dl] & \\
T_1 \arrow[ur] & & D \arrow[d] & & T_2 \arrow[ul] \\
M_1 \arrow[u] & & C \arrow[dl] \arrow[dr] & & M_2 \arrow[u] \\
 & B_1 \arrow[ul] & & B_2 \arrow[ur] &
\end{tikzcd}$$

Figure 2: Transformation graph for two arms manipulating a common object

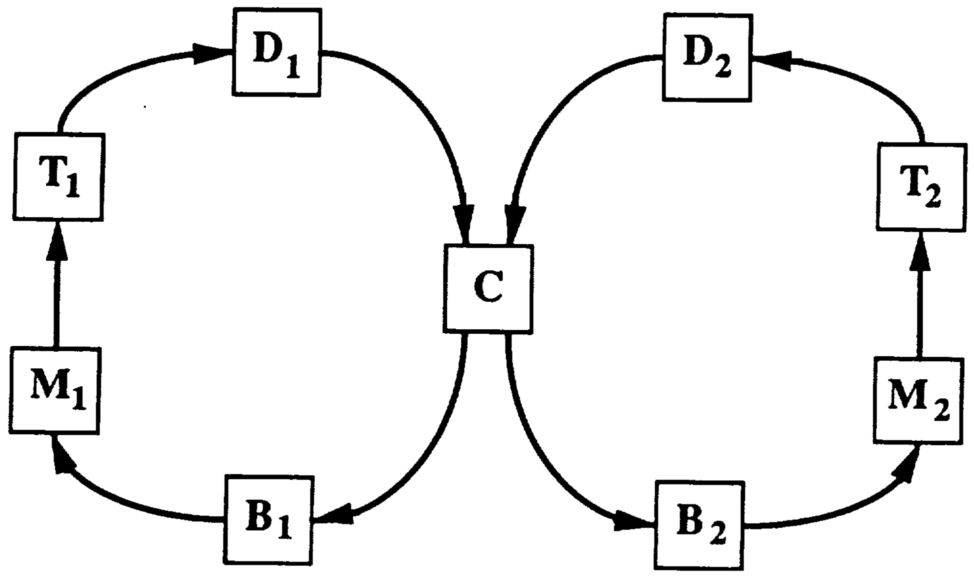

Figure 3: Two arms moving to grasp a common object

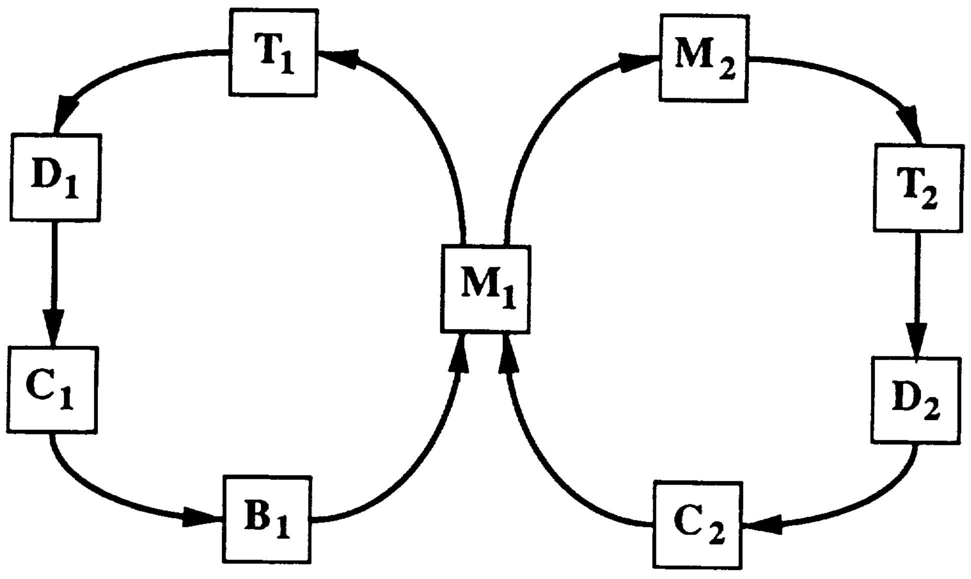

Figure 4: Micro-manipulator concatenated at terminal link of gross-positioning arm

that all forces applied to an object attached to that relationship must be accounted for by the next object. The coupling factor can also be used for load sharing in multiple-arm control. Coupling factors of 0.5 between the load and each of the two arms can be specified indicating that the trajectory generator considers each arm to dynamically carry only half of the load, thus allowing increased speed versus a one-arm motion. The maximum force that can be transmitted at an interface can also be specified. The transition time is initially determined from the temporal constraints and increased as needed to satisfy the dynamic constraints specified by the maximum transmitted forces and determined by the dynamics functions and the coupling factors.

### II.5 Control

Kali does not provide a standard servo level control algorithm for controlling the manipulators. Instead, Kali provides setpoints for the servo control and the interfaces to the control algorithms. Several control schemes for possible use within Kali are presently under study [6].

### II.6 Software

The Kali software environment is written in the C programming language and organized into five layers, each of which contains multiple libraries. The first layer, MUX (McGill University Extensions) runs on top of the commercial real time multi-processor operating system VxWorks (from Wind Rivers Systems, Inc). MUX provides an environment for running various synchronous and asynchronous processes on multiple processors [7]. The second software layer is also a support layer but is independent of the operating system. There are libraries for buffered input and output of data, geometric computations on vectors, transformations, quaternions, and kinematics and dynamics models of the manipulators. The third software layer implements the servo control using the lower level software layers. The fourth layer updates the kinematic loops in real time using two libraries. The cmotion library is used to compute Cartesian trajectories given the constraints in the motion records. The rings library maintains and updates the kinematic loops. The fifth layer continues to be developed. It consists of user level functions to simplify task description, e.g., accommodation functions and functions specific to dual arm control.

### II.7 Run Time Processing

Synchronous and asynchronous run time processes are distributed over multiple processors on a VME backplane to increase speed through parallelization. Bus bandwidth is reduced by placing separate functionalities on separate processors. The resulting processor allocation method is simple rather than optimal to facilitate implementation and evolution. Multiple update rates are utilized so that rapidly changing quantities (e.g., position, sensed forces) are updated more often than slowly changing ones (e.g., inertia characteristics, Jacobian). Asynchronous processes may include the user process, dynamics computations, and computation of the Jacobian. The user process initializes the Kali environment, describes the motion systems, issues motion requests, and communicates with higher level systems. Synchronous processes include the Cartesian setpoint generator, servo control, and sensor I/O. Communication between processes is done with message passing and shared memory. An example allocation of processes may be having the user process and trajectory generator on CPU 1, dynamics on

CPU 2 and CPU 3, kinematics on CPU 4, and servo control on CPU 5 and CPU 6.

## II.8 Hardware Environment

The Kali hardware environment consists of multiple processors connected on a VME backplane. Presently Heurikon single board computers with the Motorola MC68020 and MC68881 are used at 20 MHz with 1 Mbyte of RAM. A secondary VSBbus serves to access shared memory for all asynchronous communications. C language software is developed and compiled on a SUN workstation under Unix and downloaded to the processors on the VMEbus via Ethernet. Use of the VxWorks real time operating system provides for processor upgrade (e.g., MC68030, SPARC) as processor technology improves. Presently an initial version of Kali is running at McGill University with 1 KHz sample rate.

## III JPL Implementation for Telerobot

As explained in section I above, one of the basic goals of robotics research at JPL is the development of a space telerobot system. The standard mode of control for the telerobot is shared control [8] where the operator and autonomous control system both have inputs to the system. The modular architecture of Kali will be utilized to incorporate the shared control mode into the system. A module which may be fully autonomous in standard Kali may consist of an interface submodule which merges inputs from autonomous and operator submodules which each perform the functionality specified for the module, e.g., trajectory generation.

The hardware incorporated into the system includes two manipulators (eventually to be a 17 degree of freedom dual-arm torso system from Robotics Research Corp.), servoed grippers, multiple cameras for vision, other sensors (e.g., force/torque wrist sensors), and two force reflecting hand controllers. Several subsystems are involved in task execution including task planning, sensing, and control. Kali will be used for the low level task execution, control, and sensing. An initial proposed hardware setup for the JPL Kali system is shown in figure 5. Kali runs in one VME chassis and communicates with multiple devices. For the two arm system, Kali processes for the separate arms (e.g., kinematics, dynamics, sensing) are put on separate processors. The user program and trajectory generator for both arms are located on a single processor. Actuator commands are sent to robot motor controllers to control the arms and senor data from the arms is sent back via parallel ports. A VME chassis is used to control and sense each of the two hand controllers. A third manipulator equipped with vision cameras is controlled independently from Kali in the initial system.

## IV Kali Extensions

Kali's modular architecture provides for further evolution of the Kali environment. Evolution may be in the form of enhancements of the functions presently in Kali or additions of functionality. For example, the user may be provided with the option of selecting the servo control algorithm to be used, e.g., joint PID, operational space, multi-arm. An environment contact model could be supplied either by Kali or by the user. A world geometric model database could be added to simplify task description. Real-time collision avoidance of both the arms and environment could be included. A more flexible processor allocation scheme could be incorporated — either specifiable by the user or an automatic optimizing scheme. Also, force trajectory generation at the user level could be provided.

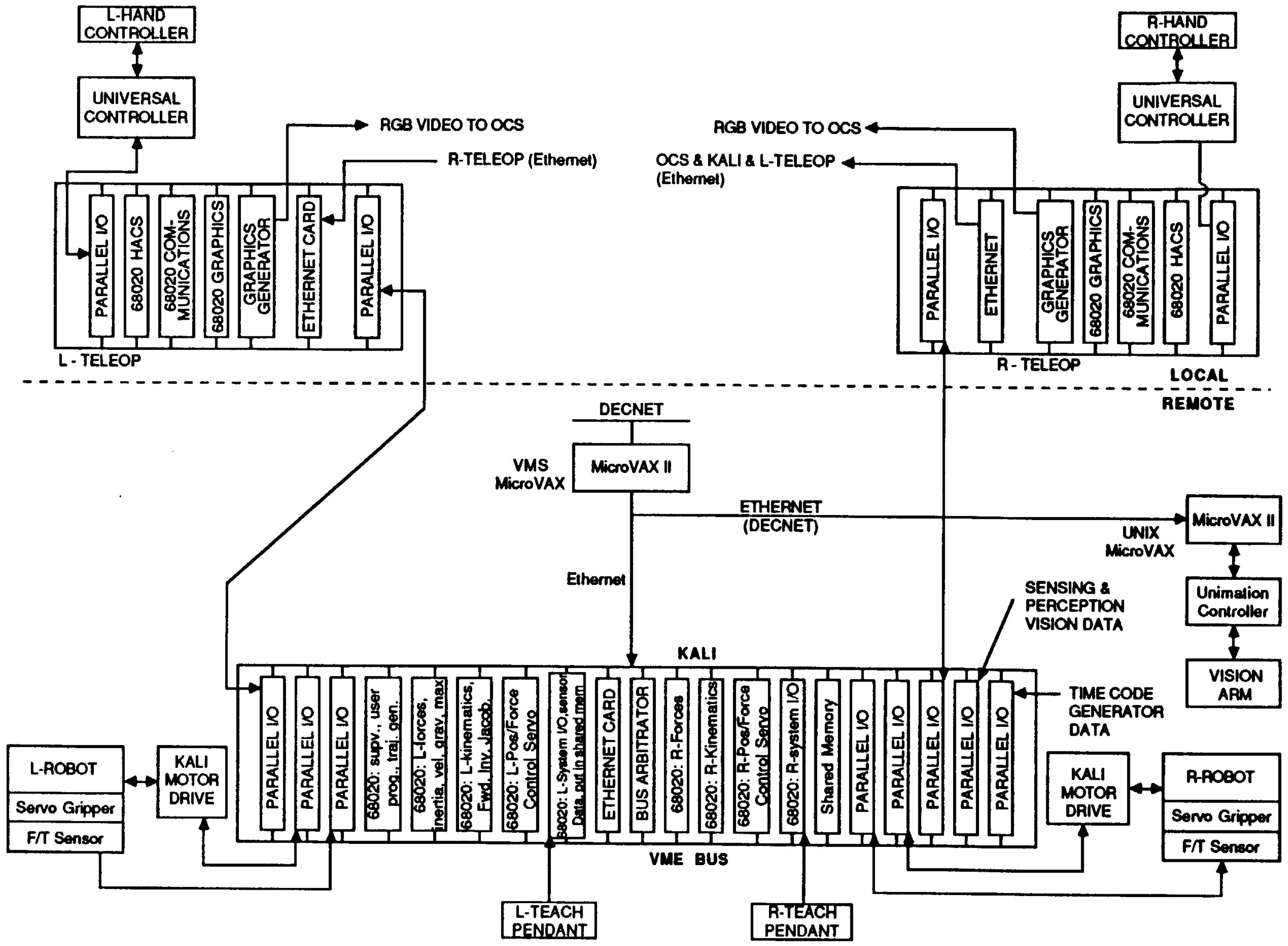

Figure 5: Proposed Kali configuration for low level of JPL Telerobot

## V Conclusions

The Kali multi-arm robot programming and control environment has been described in the context of its use in the low level of the JPL telerobot project. Its advanced features, e.g., multi-arm coordination, dynamically constrained Cartesian trajectory generation, and its open, modular architecture, make it a valuable tool for implementation of advanced robotics concepts. The initial version of Kali, presently running at McGill University, will be further enhanced and ported to JPL's telerobotics laboratory to serve as the low level of the telerobot shared control environment. Taking advantage of its modular design, the Kali environment shall continue to be enhanced as research in the various functional areas of Kali develops.

## Acknowledgements

The research described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. Graphic arts by Donna Milton are also appreciated.

## References

[1] J. Lloyd, M. Parker, and R. McClain. Extending the rccl programming environment to multiple robots and processors. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 465–474, 1988.

[2] V. Hayward and R. Paul. Robot manipulator control under unix rccl: a robot control "c" library. *International Journal of Robotics Research*, 5(4):94–111, Winter 1986.

[3] A. Nilakantan and V. Hayward. Synchronizing multiple manipulators. In *Second International Symposium on Robotics and Manufacturing, Research, Education and Applications*, 1988.

[4] V. Hayward and S. Hayati. Kali: an environment for the programming and control of cooperative manipulators. In *Proceedings American Control Conference*, 1988.

[5] V. Hayward, L. Daneshmend, and A. Nilakantan. Model based trajectory planning using preview. In *SPIE Conference, Space Automation IV*, 1988.

[6] V. Hayward, L. Daneshmend, and S. Hayati. An overview of kali: a system to program and control cooperative manipulators. In *Fourth International Conference an Advanced Robotics*, 1988.

[7] A. Topper, L. Daneshmend, and V. Hayward. A computing architecture for a multiple robot controller for space applications (kali project). In *Fifth CASI Conference on Astronautics*, 1988.

[8] S. Hayati and S. Venkataraman. Design and implementation of a robot control system with traded and shared control capability. In *IEEE International Conference on Robotics and Automation*, 1989.

## TELEROBOT PERCEPTION