AIAA-94-1253-CP

AN OPEN ARCHITECTURE MOTION CONTROLLER

Lothar Rossol
Trellis Software & Controls, Inc.
Rochester Hills, Michigan
AIAA/NASA Conference on Intelligent Robots
In Field, Factory, Service, and Space
March 1994

Abstract

Commercial controllers for robots are typically custom-designed with closed architectures on proprietary hardware and software platforms. However, the cost of *open controllers* that use standard computer hardware and software platforms is rapidly decreasing. It is now practical to build an open controller for sophisticated robot and general motion control using off-the-shelf components. Such open controller designs allow the user to standardize on hardware platforms such as VME, and on operating systems and user interfaces, such as UNIX or Windows. This paper describes *Nomad*, an open architecture motion controller. Nomad consists of a set of software modules designed to control robots, various specialty machines, and machine tools. The base operating system for Nomad is LynxOS, a POSIX-compliant real-time UNIX system. LynxOS, and hence Nomad, runs on a number of hardware platforms, including PC-ATs, VME-based PCs, Motorola and RISC processors. Nomad provides for sensor-controlled robotic motions, with user replaceable kinematics. It is programmable in C, with full UNIX compatibility, including X Windows and MOTIF. Specialized programming interfaces and languages have been added. Open architecture controllers, as represented by Nomad, will have a major impact on the robot control industry.

Introduction

Typically, motion controllers for robots and other machines have been developed based on closed architectures and proprietary platforms. The hardware, the operating system, and the software were custom designed. The result was a closed system that was inflexible and expensive to develop and maintain. In addition, closed architecture controllers could not take advantage of the rapid improvements in cost and performance driven by high volume markets outside the motion control industry. Also, standardization on common platforms by customers and end users was impossible.

In the past, the advantages of such custom architectures have been cost and performance. However, the cost of off-the-shelf standard computer hardware has dropped dramatically and performance has increased substantially in the recent past. The result of these rapid advances is that it has now become feasible to build sophisticated robot and general motion controllers using off-the-shelf components. These compare favorably in price and performance to custom designs.

If standard real-time operating systems are also used in their designs, then open architecture controllers will automatically leverage future advances in hardware, driven by R&D funding in high volume consumer markets. That is, such open designs will become more and more attractive in price and performance over time.

In addition to attractive cost and performance, open controller designs allow the customer to standardize throughout his factory on platforms such as VME, and on operating systems and user interfaces, such as UNIX or Windows. This is an attractive advantage not possible with the various custom designs. Also, such controllers can be tailored exactly to the requirements of the customer's application and the machine being controlled. This is expensive or impossible with closed custom architectures.

Published in similar form in *Proceedings*, International Robots and Vision Automation Show and Conference, April 1993, Detroit, MI.
Copyright © 1993 by Trellis Software and Controls, Inc. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

With Nomad, Trellis Software & Controls, Inc. is introducing open architecture software for motion control. Nomad software modules can be combined with standard off-the-shelf computer hardware and software to build robot or other motion controllers that are standardized and yet tailorable exactly to specific applications and machines.

Nomad Overview

Nomad was designed to be the foundation for open architecture motion controllers for robots, various specialty machines, and machine tools.

The base operating system for Nomad is LynxOS, a POSIX-compliant real-time UNIX system. LynxOS, and hence Nomad, runs on a number of hardware platforms, including PC-ATs, VME-based PCs, Motorola and RISC processors. Nomad-based controllers will automatically benefit from future advances in functionality and pricing of these hardware platforms.

A controller built with Nomad software modules and off-the-shelf standard computer components provides full power of sensor-controlled robotic motions, with user replaceable kinematics. It is programmable in C, with full UNIX compatibility, including XWindows and MOTIF. Specialized programming interfaces or languages can also be added.

Nomad-based motion controllers can be uniquely tailored to specific machines and applications. Through the power of C, X Windows, and MOTIF, tailored application packages can be developed, resulting in controllers that are very simple to use from the end user's perspective. These application packages can be menu-based or teach pendant programmable or whatever is appropriate in each instance. With application-specific interfaces, the end user need not be faced with having to learn the intricacies of an operating system or a programming language.

Nomad Components

Nomad components include *TMOS*, a Cartesian trajectory generator that provides the full power of sensor-controlled motions and a variety of industrial servo and I/O interfaces. The trajectory generator allows for six degree-of-freedom Cartesian position offsets in real time and coordination with multiple auxiliary axes. Kinematic solutions for different machine configurations can be incorporated into TMOS. The I/O control provides precise synchronization of both discrete digital and analog I/O with motion.

C-WORKS, the second component of Nomad, provides the user environment for Nomad. It consists of a C library for communicating with TMOS, a system configuration tool for Nomad, control panel functions, and a set of demonstration programs.

A wide variety of optional *Utilities* represent the third Nomad component. These utilities include a graphical servo tuning tool, a graphical machine simulator, a command line interface to Nomad, teach pendant support, and others.

Nomad was designed to be open and modular. This allows Trellis to provide additional combinations of user environments with other motion systems in the future. For example, other user environments planned consist of a robot language and an NC environment for Nomad.

The remainder of this paper will cover TMOS, the Nomad trajectory generator.

TMOS Interface to Nomad

This section describes the *High Level C Library (HLCL)* interface to TMOS. The HLCL provides many conversion functions for various forms of user data, performs integrity checks on user supplied data, provides parameter schedules to simplify motion programming, and hides internal TMOS data structures from the C program, minimizing the need for program modifications with TMOS product enhancements.

In addition to advanced motion commands, the HLCL interface to TMOS contains a number of unique features that allow programmers to communicate position information in many different formats, use "schedules" to reduce complexity of simple applications, generate

complex trajectories with minimum effort, program sensor-guided motions, generate off-line paths, and provide user control of error handling.

Connecting to TMOS

Figure 1 illustrates the interaction between a C program using the HLCL and TMOS.

Some HLCL function calls work synchronously with TMOS by sending it a message and waiting for a response before returning to the user program. Some functions, such as a request to set a digital output, require no response and return immediately after sending the message. Finally, some functions, such as status requests, interact with an information base that TMOS updates.

Motions generally execute in parallel with the user program. That is, HLCL calls that initiate motion will return when the motion is queued. When motions complete, a message is sent back to the user process. A user definable *callback* function is then used to process the completion.

Multiple user programs may *login* to TMOS. Since the information described above is kept within the space of each user process, each process has its own *context* and will not interfere with other programs that might use or modify the same parameters.

Position Allocation and Data Types

Languages for motion control typically provide one or more specific data types for position data which can be declared and allocated within a program. These types must then be supported in communications with the outside world through files or networks. The inevitable problem is one of compatibility with off-line generated data or with new and improved releases of software.

The HLCL for TMOS allocates position data internally in an undocumented format called a *TMOSPos*. HLCL calls can be used to convert between a variety of user formats and a TMOSPos. TMOS will then return a *handle* to the TMOSPos for later use in motion and other calls. The burden of storage and communication of the user's data is left with the user.

For example, an *Euler* format for position data can be declared and allocated in the user's C program. He can store and retrieve such data using files or the network. He would then call a TMOS routine to convert that data to a TMOSPos and return a handle to the user. Later, the user's C program can issue a motion call to TMOS using the handle.

In this way, issues of upward compatibility with future versions of TMOS are avoided, and the user is free to archive his data in whatever format and precision he chooses. (That is, the format of a TMOSPos is not intended to be

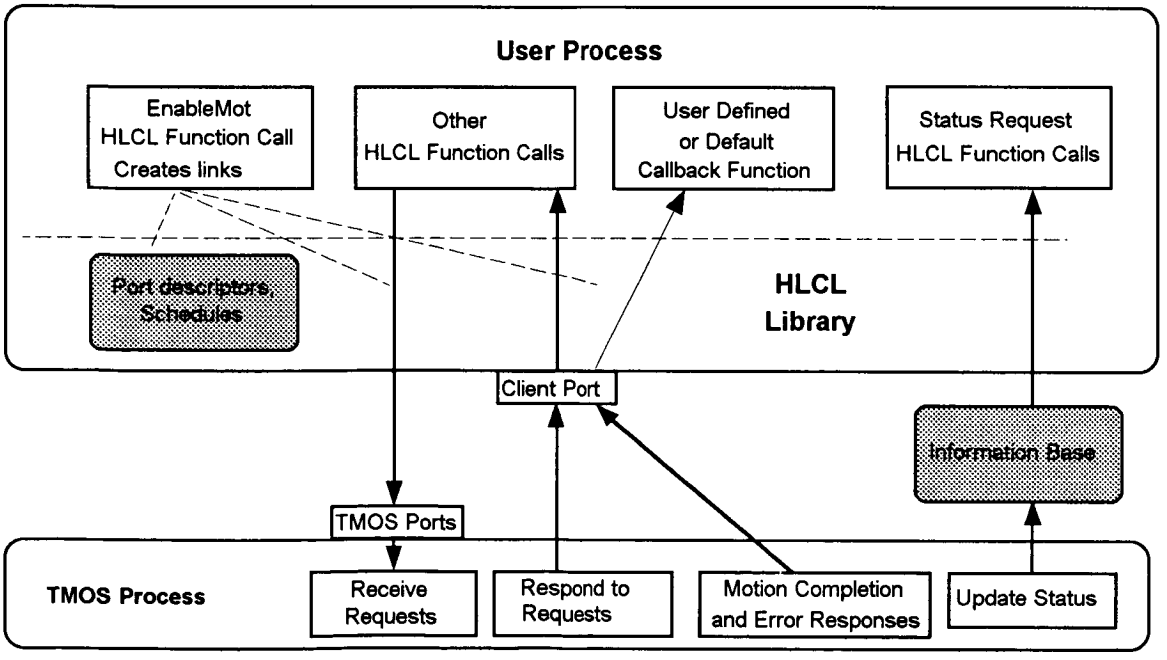

Fig. 1 -- Connecting a user process to TMOS

secret, but merely to shelter programs from changes in its structure with future enhancements.)

In addition to several other pieces of data, a TMOSPos keeps configuration information, so that when such a position becomes the argument of a machine motion, redundant solutions for the position can be reconciled. HLCL calls that return position information will also return configuration information. The user may keep this configuration information with the data or choose to ignore it.

Schedules and Other Modal Parameters

Sophisticated motion control algorithms employ many user or system definable parameters (as many as 100). It is inconvenient for the user to have to specify every parameter with each motion request. This problem is typically solved with modal parameters or system-wide parameters.

In a multiprogramming environment for multiple machines, we must further consider whether a parameter is specific to a thread of execution or whether it is specific to a particular machine. For example, it is appropriate for each separate program to keeps its own default value of speed to be used with every motion request. However, since the tool definition refers to a current physical attribute of a machine, that characteristic must be shared globally with all programs.

A TMOS schedule includes the modal parameters of speed, acceleration, motion type (linear, joint, etc.), and motion termination type.

Other modal parameters used in TMOS include a base frame of reference definition (called *Frame*) and a tool frame of reference definition (called *Tool*). These modal parameters are defined inside the TMOS process itself rather than in a schedule, because they are specific to a particular machine rather than a thread of execution. *Tool* and *Frame* are set with specific HLCL function calls.

Coordinate Systems and Coordinate Frames

All Cartesian motions produced by TMOS are defined for a specific Tool Center Point (defined by a homogeneous *Tool* transformation) and are defined with respect to a specific user definable frame of reference (defined by a homogeneous *Frame* transformation). *Tool* and *Frame* transformations are kept modally within TMOS for each machine under its control. That is, *Tool* and *Frame* need to be set only once, and those values will be used for each Cartesian motion of the machine until the modal values are changed. Changes in *Tool* and *Frame* take effect only at the beginning of the next motion using *Tool* and/or *Frame*.

Dynamic offsets to both transformations can occur at any time during a motion that uses *DeltaTool* and/or *DeltaFrame*. Changes to either of these transformations will take place immediately after the HLCL function calls that change them.

Interpolation and Termination of Motions

TMOS trajectories are broken into two categories, those that are terminated at a user specified destination (called *destination terminated*) and those that are unterminated (called *vector*). Vector motions are terminated by a succeeding motion or by an HLCL function call. Vector motions are useful for user defined sensor termination (also necessary for manual motion implementation.)

For Cartesian motions, the trajectories are of the Tool Center Point (TCP). In some systems, the destination position is checked for reachability of the tool center point before the motion is attempted. However, for many machines with nonlinear kinematics, this does not guarantee reachability of all positions on the trajectory. Therefore, TMOS does not check for reachability of destinations. TMOS confines itself to dynamic testing of reachability of each position on the desired trajectory for both *vector* and *destination terminated* motions. The test is made one deceleration period ahead of the machine's current position on the trajectory, so that the machine can be stopped on the trajectory just prior to the offending position.

The vector motions continue forever until stopped by a joint limit or are canceled or aborted. Many types of vector motion are available in TMOS, including move-to-joint-vector, move-to-world-vector, move-to-frame-vector, move-to-tool-vector, and move-to-wrist-joint-vector.

Destination-terminated motions are completed when the destination position of the motion is reached within the tolerance indicated by the termination condition. Motion types include joint-interpolated moves, straight line interpolated moves of the tool center point (TCP), circularly interpolated move of the TCP, straight line interpolation of the major axes in combination with joint interpolation of the wrist axes, and other more complex motion types.

All the above motions can be dynamically offset by various sensory inputs, such as vision.

Termination Conditions

The terminating condition of a motion for any of the above interpolation types determines how closely the machine must come to its destination before returning to the calling subroutine. TMOS provides a number of termination conditions, including:

- Return motion complete when the machine is within tolerance specified as *Fine* or as *Coarse* in the TMOS configuration.
- Return motion complete and start next motion when interpolation of this motion is complete (do not wait for servo tolerance).
- Start next motion when this motion begins deceleration. This provides the ability to blend motions.
- Initiate a new motion immediately when the next motion instruction is received. This is useful for vector motions in manual motion pendant implementation and user defined sensor applications.
- Other more complex termination types unique to TMOS, such as *fillet termination* which permits continuous motion at constant speed across motion segment boundaries.

Real-time Trajectory Modification

TMOS permits real-time modification of motions in progress. Routines are provided which accept a TMOS position as an incremental offset to either the part (*Frame*) or the tool position. This feature can be used to implement sensor-guided tracking for example. The C program can read the sensor in a loop and dynamically modify the motion in progress.

Arm Configurations

Since robots can generally reach a given Cartesian position in more than one way, Cartesian information alone is insufficient to completely determine the joint angles needed for a machine to reach that position. For example, the PUMA robot can generally reach a position with its wrist either above or below the elbow. Some redundant manipulators can reach nearly every position in an infinite number of ways.

The additional information needed to dictate how a machine will reach a given position is referred to as *configuration* information. Some controllers use specific commands to specify the configuration to be used in reaching a position. TMOS assumes configuration is part of the data. Therefore, all library routines that expect Cartesian input data also provide a parameter for configuration information.

For some simple machines, Cartesian positions can only be achieved in one way. Also, it is desirable to represent the positions of some objects without regard to how a machine might reach that object. Therefore, the *Euler* and *Transform* types defined for user representation of positions by the HLCL do not incorporate configuration information inside the data. Configuration data may be optionally added to these data types when they are converted to TMOS positions by the TMOS conversion functions, or when they are converted to machine joint angles.

Speed

TMOS by default provides coordinated motion of all axes of a machine on every motion. This means that for each motion request TMOS will coordinate the motion such that each joint of a machine will begin and end its motion at the same instant in time. The meaning of *speed* therefore, must apply not to individual axes, but to some entity which represents a combination of the axes of a machine. In some cases, speed applies to the tool center point with respect to the Frame. In some cases it is not possible to define a single point at which speed can be measured (joint interpolated motion for example) and speed is defined relative to some maximum. In addition, for motions which involve changes in both orientation and translation of the tool, both rotation and translation speed constraints must be taken into account.

TMOS considers up to three kinds of speed in the coordination of a motion, depending on the motion type:

- Tool translation speed - the speed of translation of the defined tool center point relative to the specified Frame.
- Tool rotation speed - the speed of rotation of one or more angles of orientation, measured in rotation units per second.
- Axis speed - the speed of motion of an axis (either rotation or translation), measured relative to the maximum for that axis.

Speed Limits

TMOS imposes speed limits only on a joint basis. That is, TMOS continuously monitors the speed of all axes. If any axis exceeds its speed limit as defined in the TMOS configuration file, then all axes are scaled back to maintain coordinated motion at the limiting joint speed. No limit is imposed on Cartesian translation or Cartesian orientation speed control.

TMOS also permits acceleration control on a per-motion basis.

Motion Commands and Continuous Motion

The HLCL is designed so that routines return as soon as possible. In the case of motion requests, the routines will return as soon as an "ID" can be assigned to the motion. The calling program can then continue processing other events and handle motion completion as an asynchronous event. If the programmer wishes to issue a series of motion requests, which is often the case, he must wait for previous motions to complete. A library routine is provided, which permits the caller to wait for the completion of a specific motion, identified by the ID.

Of course, the programmer can also cause continuous motion, by using a suitable termination type, so that the machine will not decelerate at the end of each motion and not wait for the completion of the previous motion. Continuous motion with the appropriate termination type will cause succeeding motions to be smoothly blended with previous ones. Smooth blending of motions at *constant speed of the TCP* is a unique option in TMOS.

Summary

TMOS, when combined with C-WORKS and other utilities, forms Nomad, a foundation for open architecture motion control for robots and other machines. Nomad software runs in a UNIX environment on off-the-shelf hardware, to provide low cost and high functionality motion control, standardized and yet amenable to tailoring for highly specialized applications.