Proc. IEEE International Conference on Robotics, pp. 293-297.

Introduction to RCCL : A Robot Control "C" Library

Vincent Hayward
Richard P. Paul

School of Electrical Engineering, Purdue University
West Lafayette, Indiana 47907, USA

ABSTRACT

RCCL is a robot programming system that enables a user to specify robot manipulator tasks employing a set of primitive system calls similar in spirit to those of the UNIX input-output system. The goals addressed in the RCCL system are: manipulator task description; sensor integration; updatable world representation; flexibility; wide range of applications; medium level robot programming; off-line programming; efficiency; manipulator independence; portability; foreground-background programming; Cartesian path programming; arbitrary path specification; tracking; force control.

1. Introduction

Most current robot programming systems are based on a dedicated programming language. Quite a large number of them exit (AL, AML, HELP, JARS, LM, MCL, RAIL, VAL). They consist of a language interpreter running at low priority specifying motion parameters to a trajectory generator. The trajectory generator, running at high priority, and usually interrupt driven, computes the sequence of joint variables so as to produce the desired motion. The sequence of joint variables is in turn transmitted to a servo process capable of actuating the robot's joints. The execution flow of the robot program is synchronized with the actual motion of the manipulator. Most language based systems, if not all, are strongly tied to the computer hardware on which they run, as well as to the type of manipulator they control. The more sophisticated robot programming languages become, the more they resemble high level computer programming languages (ALGOL, PASCAL, etc.) augmented with the data structures and operators necessary to control robots. Some languages can handle concurrent processing.

RCCL is not a language but a set of system calls suitable for the control of robot manipulators. Manipulator programs become ordinary computer programs, and the manipulator is considered as a peripheral device. Since manipulator control primitives are defined at the system level, a program written in any language which is able to provide the proper list of arguments can use the manipulator primitives.

Instead of designing yet another robot programming language, we use the C language to write manipulators programs. The RCCL system is itself written the C language. C is a high level structured language suitable for projects of any size, which allows us to deal with low level implementation details. Programs are easily portable, and yet can be efficiently implemented. Two criticisms are often made of compiled languages based systems. The compilation time increases the edit-test cycle time. If a program fails either because it is wrong from the manipulation point of view, or from the programming point of view, the whole task has to be stopped. Practice has shown that these limitations are largely offset by the gain in flexibility and generality. If for some applications, an interpreted language is needed, the interpreter of a general purpose or a dedicated language can also make use of RCCL system calls. We would obtain ,in that case, a large gain in modularity. The RCCL design approach has advantages in modularity, flexibility and hardware independence.

2. Overview

2.1. Manipulator task description

The location of an object is described by its position and orientation with respect to some reference coordinate frame. In the remaining, the word 'position' will implicitly stand to 'position and orientation'. Tasks are described in terms of positions to be reached in space to grasp, displace or exert forces on objects located in the robot work space. Tasks are also described by the sequence and the type of motions necessary to carry out the work. Position descriptions require special data structures and sequential operations of a robot require special primitives. Both can however be implemented with the tools provided by high level languages, namely, data structures, functions ,and structured flow of control. (The C language does not know anything about a file, for example. Users wishing to manipulate files in their programs have to include a system file called "stdio.h". This file contains a description of the necessary data structures. Files can be manipulated by system primitive functions like *read, write, filbuf, or, flsbuf* [1]).

This work is partially supported by a Grant from the CNRS project ARA (Automatique et Robotique Avancée), France. Facilities to perform this research are provided by the Purdue University CIDMAC project. Richard Paul is the Ransburg Professor of Robotics. This material is also based on work supported by the National Science Foundation under Grant. No. MEA-8119884. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

The first implementation runs on a VAX 11/780 computer under UNIX. It has been used to control a PUMA manipulator and a Stanford Arm.

2.1.1. Structured position description

RCCL handles what is referred to as structured position description [2]. The basic construct is the homogeneous transformation that is a mathematical construct describing the position of coordinate frames. A homogeneous transformation can either be interpreted as the description of the position of a coordinate frame with respect to another, or as a transformation performed on the first coordinate frame. Homogeneous transformations are a very general tool [3], however in manipulation we will restrict them to orthogonal transformations, built in terms of a 3 by 3 rotation matrix constructed with three orthogonal vectors n, o, and a, and a position vector p.

Relative positions of objects can be described with transformations products. For example, let OBJ, a transformation, describe the position of an object relative to a reference coordinate frame. Let HOLE represent the position of a hole with respect to the frame OBJ. The matrix product OBJ HOLE which is also a homogeneous transformation, describes the position of the hole relative to the reference coordinate frame. One important property of orthogonal homogeneous transformation is that the inverse transformation can be obtained at reduced computational costs.

One dedicated transformation T6, represents the position of the end-effector with respect to the reference coordinate frame located at the base of the manipulator. A given manipulator position can be specified in base coordinates by writing:

$$T6 = POS$$

However, such a description is usually insufficient. For instance, one might need to express that a tool attached to the arm end-effector must reach the position POS. This is achieved by writing:

$$T6\ TOOL = POS$$

A more complete description of a motion to a goal position might be written as:

$$REF\ T6\ TOOL = CONV\ OBJ\ PG$$

Where:

REF is the position of the manipulator with respect to reference coordinate frame.

T6 describes the position of the manipulator end-effector with respect to the reference coordinate frame attached to the shoulder or to the base of the manipulator.

TOOL expresses the position of a tool attached to the end-effector.

CONV represents a conveyor belt, defined as a coordinate frame moving with respect to the reference coordinate frame.

OBJ is the position of the object to be grasped lying on the conveyor belt.

PG is the position of the end-effector, relative to OBJ, where the object is to be grasped.

Position equations are solved for T6 to obtain the desired position of the manipulator with respect to the reference coordinate frame:

$$T6 = REF^{-1}\ CONV\ OBJ\ PG\ TOOL^{-1}$$

One RCCL system call directly implements position equations in terms of dynamic data structures. The positions can be modified at the level of the move statement in terms of small translations and rotations described in the *tool* frame. This provides a convenient short hand for specifying approach and deproach positions, or for specifying motions which purposely over shoot the described position when the arm is to perform guarded motions [21].

2.1.2. Motion description

A task is made up of a number *path segments* between successive positions. There are many ways for generating trajectories for a manipulator[4][5]. RCCL provides two types of motions. The first one, called *joint mode,* consists of computing the set of joint values for each end of path segment and generating all intermediate values by linear interpolation. The second type, that we will call *Cartesian mode,* requires the system to solve a modified position equation each sample interval and to compute the corresponding joint coordinates. The position equation is internally modified in such a way that one frame, called the *tool frame,* moves along straight lines and rotates around fixed axis. These motion types are discussed elsewhere [3][6]. Here, we will assume that we are dealing with a manipulator for which an analytical solution exists, relating a Cartesian position to a set of joints coordinates [7][8][9][10]. In the current implementation, manipulator motions are obtained by specifying a sequence of desired joint values to the servo processes controlling the manipulator joints. However, most of what follows does not assume a particular control method.

Path segment transitions involve three positions. The manipulator is on its way from position P1, is about to perform a transition next to P2 in order to head toward P3. Transition are rendered necessary to avoid velocity discontinuities, and are computed using a quartic polynomial. At the time of a transition, the subsequent *path segment* is fully described by the goal position P3, P1 and P2 being known from the current motion, by the time of the transition, and by the time of the segment itself. RCCL allows the user to specify velocities, as well as segment times. If the velocity is specified, the Cartesian distance of each tool frame is determined to compute the segment time automatically.

When the manipulator is to move while exerting forces or torques on objects, the manipulator must be controlled in a such a way that forces and torques are controlled directly in place of positions. The manipulator is then said to be controlled in a *comply* mode. Several methods [11][12][13][14] are proposed for such a control. RCCL implements a variation of Shimano's joint matching method [22]. RCCL provides for compliance specifications in the *tool* coordinate frame which is specified in the position equation. Compliance is specified in terms of forces along, and torques around the principal axes of the *tool* frame. The manipulator looses one if the positional degree of freedom for each direction along, or around which the manipulator is complying in force. The trajectory is then constrained by the geometrical features of the objects in contact. A more complete discussion of this subject can be found in [15].

2.2. Sensor integration; Updatable world representation;

One of the main goals of RCCL is to facilitate the integration of sensors [16]. Sensors are used to influence the behavior of the manipulator according to information acquired from the manipulator or from its environment. Sensor information can be classified in many different ways : according to the data type necessary to represent it, booleans, scalars, vectors, arrays, tensors, etc...; by meaning, touch, limit, distance, position, temperature, vibration, force, etc...; by the order of magnitude of the acquisition time, minutes, seconds, milliseconds, microseconds; by accuracy ;and so on. Considering this variety, the RCCL approach is to deliberately ignore, when possible, the type of information we may have to deal with, but on the contrary, to provides means for an efficient utilization of this information.

2.2.1. Foreground - background programming

As robot programs will have to interact with the environment while the manipulator is moving, programs are not implicitly synchronized with the robot motions. Each time a motion is required, a *motion request* is entered into a 'First-in first-out' queue. The request consists of a record containing all the information necessary to perform the corresponding path segment. This feature allows us to specify ahead a sequence of motions and to perform input output operations and calculations as the robot is executing the requests. When the motion queue becomes empty, the manipulator is brought to rest. We will see that it does not necessarely mean that the manipulator is brought to a stop in absolute coordinates. Slow sensors such as computer vision systems requiring lengthy computations can then be efficiently used as there is no need to stop the manipulator while the data is acquired and processed. A 'wait' primitive is provided when it is necessary to synchronize the execution of the program with the manipulator's motions. A similar technique to allow for the simultaneous control of several manipulators or positioning devices may be implemented in the future.

2.2.2. Influencing positions

End of segment positions can be modified according to information acquired at run time. This is achieved by changing the value of transformations within position equations. Transformations likely to be modified at run time must be declared as such (*hold* transforms). The system makes a copy the transformation at the time the corresponding *move request* is issued, and enters it in the motion queue. It is therefore possible to use the same transformation to describe a coordinate frame whose value is different from one path segment to another. Using a copy of the transformation, makes it possible to change the value at an arbitrary instant even if the corresponding position equation is currently being evaluated. A typical use of this type of transformation is the description of an object position that is variable and obtained from sensor readings at discrete time intervals.

User interaction and slow sensors like computer vision require the use of *hold* transformations. Position data can be acquired ahead at time in a completely asynchronous manner.

2.2.3. Influencing trajectories

Fast sensors can provide for direct synchronous sensory feedback. This corresponds to the class of *functionally* defined transformations. In this case, a transformation is attached to a function that will be evaluated each sample time. The purpose of the function is to calculate the value of the transformation as a function of sensor readings. The position equation in section 2.1.1. makes use of such a functionally defined transform to describe a position with respect to a conveyor belt. If the motion is performed in *Cartesian* mode, the tracking is perfectly accurate, since the position equation is evaluated at sample time intervals. When the motion is performed in *joint* mode, the system estimates the expected position at the end of the segment by linear extrapolation. If the functionally defined transform is computed as a function of time, we can obtain mathematically described motions (circles, ellipses etc...).

The transitions to, or from path segments involving moving coordinate frames must deal with unpredictable velocity changes. Smooth transitions are obtained by adding a third order polynomial trajectory modification during the transition time. We have seen that manipulator stops are obtain by repeating a move to the same position. When the position involves moving coordinate frames, the stop will be relative to those moving coordinate frames. If a stop in absolute coordinates is required, a move to a fixed position must be performed before specifying the stop. The system internally maintains a position equation which always reflects the current position of the manipulator. It is therefore possible to have the manipulator stop at an arbitrary instant at the position it currently occupies. Functionally described transformations can be used anywhere in a position equation. Trajectories can be affected with respect to any coordinate frame which provides unlimited applications.

2.2.4. Influencing path segment times

The second way to influence the manipulator behavior is to modify the length of the path segments, to start and to stop the manipulator according to external events. The RCCL system allows to interrupt the execution and cause a transition to the next path segment at any moment by merely setting a global flag. A motion termination code enables the user to determine the cause of the path segment termination. For example, the system internally checks for joint limits and brings the manipulator to an absolute stop when one of them is reached. The termination code allows us to check for the proper termination of the motions that may cause a joint limit and to take an appropriate action. For any motion terminated on a condition, a meaningfull termination code is returned. An arbitrary monitoring function can be specified as part of a motion request, the termination code is then chosen by the user. Start, stop, motion interruption and resumption are achieved using the same mechanism.

2.2.5. Internal sensing

Internal information is acquired from the manipulator itself. Two particularly useful kinds of informations are internally maintained in RCCL: position and force.

2.2.5.1. Position

For any motion terminated on a condition, the world model may have to be updated to account for the actual position where the manipulator stopped. The system is then asked to *update* a transformation in a position equation. The equation is solved for requested transformation using the actual value of T6 when the path segment ends. This new position information might be very useful in any subsequent motion related to this location. For example, consider the case of a manipulator picking up an object which it had previously placed on a surface whose height is only approximatively known. The manipulator is able to retrieve the object immediately if the final position of the object was updated.

2.2.5.2. Force

Joint torques are also obtained from the manipulator state. The complete determination of the forces and torques exerted on an object based on the joint torques leads to lengthy computations [17], RCCL, however provides a mechanism that compares the actual forces and torques against expected values. This information may be used to cause a path segment termination when some specified limit is reached. The subsequent path segment will usually contain compliance specifications.

3. The RCCL implementation

When a manipulator in under RCCL control, four processes are concurrently running. At the lower level a *servo process* controls the position or the torque of each manipulator joint as input parameters. The *setpoint process*, running at interrupt level, computes the Cartesian trajectories and determines the corresponding joint parameters. A real time communication channel swaps information between the *servo process* and the *setpoint process*. *The user process* running under time sharing contains the RCCL system calls. The *setpoint process* communicates with the *user* process via a motion request queue containing all the necessary information.

3.1. Servo process

The present implementation makes use of Unimation PUMA robot controllers. These controllers include six micro processors, one per joint. Each joint servo micro processor receives position commands specified in incremental encoder values. The joint processors can also read and transmit the joint position information. The Stanford arm controller has been modified [18][19] so that joint 1, 2, and 3 can be force servoed. The Puma arm controller can drive the joints motors with current specifications. A method for relating joint torques to motor currents has been developed and implemented by Zhang Hong [20]. The method take into account the friction effect of the joint drives.

3.2. Joint processor control and host machine interface

A LSI11 microprocessor supervises the joint processors and establishes the communication with the host machine. At each sample time interval, the microprocessor gathers data from the manipulator, transmits it to the host machine, accepts commands, and sends the corresponding values to the joint processors. It also executes a calibration procedure at startup time.

3.3. Real time channel communication

The real time channel, besides transmitting information between the controllers and the host machine performs several functions such as the conversions of encoder values to trigonometric angles, the conversion of torque to current, joint limits. It also monitors maximum velocities and maximum currents and checks for data integrity. A manual stepping mode and an automatic rest position return are built in.

3.4. Setpoint process

The *setpoint* process is interrupt driven. Each time a path segment terminates, the process attempts to obtain a new motion request from the queue. If there is one, the transition parameters are computed according to the type of path segment and the transition parameters are computed. Many types of transitions occur : joint mode, Cartesian mode, moving coordinate frames, constrained motions. The final result is always a set of joint positions and torques.

3.5. User process

The *user* process consist of the user program calling the RCCL primitives. Memory space is dynamically allocated for each new position equation. This space can be released when needed no longer needed. Several functions are provided to handle transformations: rotations, Euler angles, roll pitch and yaw angles, transform multiplication, transform inversion, etc...

4. Tools

4.1. Trajectory planning

There exists a version of the RCCL library, which instead of computing the trajectories in real time, computes them off-line. This is achieved by calling the setpoint function in a loop instead of activating it upon interrupt. The same manipulator programs, provided that they do not depend on external events and information, can be run in this fashion. Some debugging tools are then provided. The system can be asked to keep a trace of the motion requests, to store the sequence of setpoints on file in order to replay them afterwards, or to plot them.

4.2. Teaching

A manual control program is included within RCCL. It consists of a very simple command line language interpreter enabling an operator to interactively move the manipulator in Cartesian coordinates. Motions can be specified in world or tool coordinates. Positions can be recorded via the *update* primitive. The manual control program is implemented entirely in terms of RCCL primitives.

4.3. Transformation data base

A simple data base system has also been developed. Transformation values can be recorded and read on line in manipulator programs. The values can be displayed and modified off-line for maintenance.

5. Conclusion

The main goal of this project was to show that manipulator control could be developed in a more general context than within the framework of a stand alone

robot controller with to it's own language. The current RCCL implementation does not yet offer the convenience of dedicated robot controllers because it requires a large machine. However, as microprocessor based computers become more powerful and can run operating systems like UNIX, the RCCL approach shows many advantages over conventional robot controller designs. The conclusion we wish to draw is that robot control can be viewed as an addition to an already existing, tested, and standardized system, rather than the design from scratch of a system which provides only for robot control.

6. Acknowledgments

We would like to thank Bill Fisher for his contribution to this work. Calculations and measurements of the dynamic and friction parameters of the manipulators have been performed by Zhang Hong who made the force control implementation possible. The specialized real-time device driver and the additions to the UNIX kernel code are the work of George Goble whose help has been most valuable.

7. References

[1] Kernighan ,B. K., "The C Programming Language", Prentice-Hall, 1978.

[2] Paul ,R. P., "Manipulator Language", Workshop On The Research Needed to Advance The State Of Knowledge In Robotics, April 15-17, 1980, organized by J. Birk and R. Kelley, supported by N.S.F.

[3] Paul, R. P., "Robot Manipulators: Mathematics, Programming, and Control", MIT Press 1981.

[4] Derby, S., "Simulating Motion Elements of General-Purpose Robot Arms", International Journal of Robotic Research, Vol. 2, No. 1, Spring 1983.

[5] Castain, R. H., Paul, R. P., "Polynomial Robotic Trajectories: A New Approach", TR-EE 82-37, Dec 1982.

[6] Hayward, V., Paul, R. P., "Robot Manipulator Control Using the C Language Under UNIX", IEEE Workshop on Languages for Automation, Chicago, Nov. 1983.

[7] Shimano, B. E., "The Kinematic Design and Force Control of Computer Controlled Manipulators", Stanford Artificial Laboratory, Stanford University, AIM 313, 1978.

[8] Paul, R. P., Stevenson, C. N., "Kinematics of Robot Wrists", International Journal of Robotic Research, Vol. 2, No. 1, Spring 1983.

[9] Paul, R. P., Shimano, B. E., Mayer , E. G., "Kinematic Control Equations for Simple Manipulator", IEEE Transactions on Systems, Man, and Cybernetics, Vol SMC-11, No 6, June 1981.

[10] Fisher, W. D., Private communication.

[11] Inoue, H., "Force Feedback In Precise Assembly Tasks", MIT Artificial Intelligence Laboratory, Memo 308, Aug 1974.

[12] Raiberg, M. H., Craig J. J., "Hybrid Position/Force Control of Manipulators", Journal of Energy Resources Technology, Vol. 103, June 1981.

[13] Salisbury, J. K., "Active Stiffness Control of a Manipulator In Cartesian Coordinates", 19th IEEE Conference on Decision and Control, Dec. 1980, Albuquerque, New Mexico.

[14] Geschke, C. C., "A System for Programming and Controlling Sensor-Based Robot Manipulators", IEEE Transactions on Pattern Matching and Machine Intelligence, Vol. PAM1-5, No. 1, Jan 1983.

[15] Mason M. T., "Compliance and Force Control for Computer Controlled Manipulators", MIT TR-515, April 1979.

[16] Rosen, C. A., Nitzan, D., " Use of Sensors In Programmable Automation", Computer Magazine, December 1977.

[17] Paul, R. P., "Computational Requirements of Third Generation Manipulators"

[18] Fisher, W. D., "The Modification of a Robotic Manipulator and Digital Controller to Incorprate Both Force and Possition Control", MSEE Thesis, Purdue University, May 1981.

[19] Luh, J. Y. S., Fisher W. G., Paul, R. P., "Joint Torque Control by Direct Feedback for Industrial Robots", IEEE Transaction on Automatic Control, Vol. AC-28, No. 2, February 1983.

[20] Zhang, H., Paul, R. P., "Determination of Simplified Dynamics of Puma Manipulator", Purdue University.

[21] Will, P. M., Grossman D. D., "An Experimental System for Computer Controlled Mechanical Assembly", IEEE Trans. Computers C-24 9, 1975, 879-888.

[22] Shimano, B. E., "The Kinematic Design and Force Control of Computer Controlled Manipulators", Ph.D. Dissertation, Memo AIM-313, 1978, Stanford Univ. t