Robot Control and Computer Languages

R P Paul and V Hayward

School of Electrical Engineering, Purdue University, West Lafayette, IN 47906, USA

Summary: From the earliest stages of their development, robot manipulators have been tied to computers by robot-control languages. These special languages have endeavoured to deal with the complexities of real-time control, multiple processes, the description of robot-manipulation tasks and the integration of sensors. In every case, these languages have been able to provide only partial solutions to the general problem. We propose a new solution to the problem by integrating the robot control into an existing high-level language. The robot manipulator is integrated in such a manner that conventional programming techniques can be used to solve the special requirements of manipulator control. We use the 'C' language and run the manipulator under the UNIX operating system. The robot manipulator is integrated into the language in the same manner as is input/output; that is, integration into the language is handled by a small set of functions included in a library. The robot program thus becomes a conventional 'C' program. The implementation language of the library is also written in 'C', which provides a 'user transparent' system, allowing complete freedom in the mode of controlling the manipulator.

Introduction

The earliest work in robotics was that of Ernst[1] who in 1960 interfaced a teleoperator 'slave' arm to a small digital computer. The 'arm', equipped with touch and force sensors, was able to search a workspace to locate a box and a number of blocks which were then placed into the box. The control algorithms for this task were programmed in a high-level computer language. The program for the task consisted of many conditional statements testing sensors and requesting small manipulator motions. It would be hard to imagine expressing such algorithms in any other than a high-level computer language. This trend continued into the sixties, with the manipulator embedded into such languages as LISP and FORTRAN. Not only were sensors integrated into the manipulator control but coordinate transformations between vision systems and the manipulator had also to be made. Once again, the data structures and control of a high-level language were used naturally.

At the same time that these developments were taking place, the industrial robot was developed. This device was directly programmed by moving the manipulator through a task and recording the task positions. While this appeared to be a much simpler approach to programming, it was in reality no more than the input of task positions. The industrial robot was in fact only a programmable positioning and orienting device, a very useful component in automation systems but, lacking all forms of task sensor input and task plan or model, it had no need for algorithms or for the languages in which they might be expressed. The program for an industrial robot was in fact hard wired and simply consisted of moving the manipulator from one position to the next through the taught sequence.

A. Morecki et al. (eds.), *Theory and Practice of Robots and Manipulators*

The use of high-level languages for robot manipulator control had a major drawback due to the lack of concurrency. A motion would be planned and then executed, but during execution no other processing could be performed. Any form of sensor interaction had to be included in the real-time program that moved the manipulator, a difficult task. A second difficulty was that the manipulator had to be brought to rest before a subequent motion could be planned. Although this did not appear to be a problem in the early development of robotics, it soon became apparent that the time lost due to the continual need to interrupt motion by bringing the manipulator to rest was a serious limitation. The solution found to the problem was to define the robot control task in terms of two concurrent processes. One process would run in real time to control the motion of the robot; the other process would run in background to compute the next motion so that it would be ready as soon as the current motion was completed. With such a system it was possible to move the robot through a number of path positions without needing to bring it to rest at any intermediate position. Sensor integration could also be performed in terms of additional concurrent processes. Unfortunately, none of the available operating systems supported this type of concurrent process.

Although no general solution to the operating-system problem was undertaken, a number of stand-alone robot-control systems were developed which provided for concurrency. Within these systems various feedback and control strategies were developed. But with the advent of these special-purpose systems the generality of the high-level languages was lost; gone were the data structures, input/output, control statements, subroutines, etc. Their lack was apparent as soon as the fundamental robot-control problems were solved and more ambitious tasks were undertaken. The special-purpose languages were then extended to include many of the features of high-level languages and while this seemed like a reasonable approach it necessitated that the user become familiar with a new language. These systems were also fairly inefficient computationally, lacking the many man years of development on which some of the more standard high-level languages are based. The implementation language in these new systems was different from the user language, making extension difficult or impossible.

The approach we take here is to identify the robot as an input/output device and to integrate it directly with an existing high-level language. Concurrency is provided within the operating system. An optimizing compiler is available for both the user and as implementation language. There are no special data types as the entire system is represented in terms of standard language features. We have included the manipulator into the 'C' programming language in the form of a library, RCCL – the Robot 'C' Control Library.

Overview

Manipulator-task description

The location of an object is described by its position and orientation with respect to some reference co-ordinate frame. In the following, the word 'location' will implicitly mean 'position and orientation'. Tasks are described in terms of locations to be reached in space to grasp, displace or exert forces on objects located in the robot workspace. Tasks are also described by the sequence and the type of motions necessary to carry out the work. Location descriptions require special data structures

and sequential operations of a robot also 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.

Structured location description

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

Relative locations of objects can be described with transformation products. For example, let *OBJ*, a transformation, describe the location of an object relative to a reference co-ordinate frame. Let *HOLE* represent the location of a hole with respect to the frame *OBJ*. The matrix product *OBJ HOLE*, which is also a homogeneous transformation, describes the location of the hole relative to the reference co-ordinate frame. One important property of orthogonal homogeneous transformations is that the inverse transformation can be obtained very simply.

One dedicated transformation *T6*, represents the location of the end of the manipulator with respect to the reference co-ordinate frame located at the base of the manipulator. A given manipulator location can be specified in base co-ordinates by writing

$$T6 = POS$$

However, such a description is usually insufficient. For instance, one might need to express that a tool is attached to the end of the manipulator which must reach the location *POS*. This is achieved by writing

$$T6\ TOOL = POS$$

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

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

Where *REF* is the location of the manipulator with respect to the reference co-ordinate frame; *T6* describes the location of the end of the manipulator with respect to the reference co-ordinate frame attached to the shoulder or to the base of the manipulator; *TOOL* expresses the location of a tool attached to the end of the manipulator; *CONV* represents a conveyor belt, defined as a co-ordinate frame moving with respect to the reference co-ordinate frame; *OBJ* is the location of the object to be grasped lying on the conveyor belt; *PG* is the required location of the tool, relative to *OBJ*, where the object is to be grasped.

Location equations are solved for *T6* to obtain the desired location of the end of the manipulator with respect to the reference co-ordinate frame

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

One RCCL system call directly constructs location equations in terms of dynamic data structures. The locations can be modified at the level of the move statement in terms of small translations and rotations described with respect to the tool frame.

This provides a convenient shorthand for specifying approach and deproach locations, or for specifying motions which purposely overshoot the described location when the manipulator is to perform guarded motions.[4]

Motion description

A task is made up of a number of path segments between successive locations. There are many ways to generate trajectories for a manipulator.[5,6] RCCL provides two types of motions. The first, called joint mode, consists of computing the set of joint values for each path segment end and generating all intermediate values by linear interpolation. The second type, which we will call Cartesian mode, requires the system to solve a modified location equation at each sample interval and to compute the corresponding joint co-ordinates. The location equation is internally modified in such a way that one frame, called the tool frame, moves along straight lines and rotates around a fixed axis. These motion types are discussed elsewhere.[3,7]

When the manipulator is to move while exerting forces or torques on objects, the manipulator must be controlled in such a way that forces and torques are controlled directly in place of locations. The manipulator is then said to be controlled in a comply mode. Several methods[8–11] are proposed for such control. RCCL implements a variation of Shimano's joint matching method.[12] RCCL provides for compliance specifications in the tool co-ordinate frame which is defined in the location equation. Compliance is specified in terms of forces along, and torques around, the principal axes of the tool frame. The manipulator loses one degree of freedom for each direction along or around which it is complying, in force or torque respectively. 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 reference 13.

Sensor integration; updatable world representation

One of the main goals of RCCL is to facilitate the integration of sensors.[14] Sensors are used to modify the behaviour 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, location, temperature, vibration, force, etc; by the order of magnitude of the acquisition time, whether minutes, seconds, milliseconds or microseconds; by accuracy and so on. Considering this variety, the RCCL approach is deliberately to ignore, when possible, the type of information we may have to deal with but, on the other hand, to provide means for an efficient utilization of this information.

Modifying locations

End of segment locations can be modified according to information acquired at run time. This is achieved by changing the value of transformations within location equations. Transformations likely to be modified at run time must be declared as such (hold transforms). The system makes a copy of 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 co-ordinate frame whose value is different from one path segment to another. Use of a copy of the transformation makes it possible to change the value at an arbitrary instant even if the corresponding location equation is currently being evaluated. A typical

use of this kind of transformation is the description of an object location 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. Location data can be acquired ahead of time in a completely asynchronous manner.

Modifying 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 location equation in the *Structured location description* section makes use of such a functionally defined transform to describe a location with respect to a conveyor belt. If the motion is performed in Cartesian mode, the tracking is perfectly accurate, since the location equation is evaluated at sample time intervals. When the motion is performed in joint mode, the system estimates the expected location 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 co-ordinate frames must deal with unpredictable velocity changes. Smooth transitions are obtained by adding a modifying third-order polynomial trajectory during the transition time. We have seen that the manipulator is stopped by repeating a move to the same location. When the location involves moving co-ordinate frames, the stop will be relative to those moving co-ordinate frames. If a stop in absolute co-ordinates is required, a move to a fixed location must be performed before specifying the stop. The system internally maintains a location equation which always reflects the current location of the manipulator. It is therefore possible to have the manipulator stop at an arbitrary instant at the location it currently occupies. Functionally described transformations can be used anywhere in a location equation. Trajectories can be modified with respect to any co-ordinate frame which provides unlimited applications.

Internal sensing

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

Location. For any motion terminated on a condition, the world model may have to be updated to account for the actual location where the manipulator stopped. The system is then asked to update a transformation in a location equation. The equation is solved for the requested transformation by using the actual value of *T6* when the path segment ends. This new location 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 approximately known. The manipulator is able to retrieve the object immediately if the final location of the object has been updated.

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;[15] 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.

RCCL implementation

When a manipulator is under RCCL control, four processes are running concurrently. At the lower level, a servoprocess controls the location or the torque of each manipulator joint. 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 servoprocess and the setpoint process. The user process running under time sharing is the user program and makes the RCCL system calls. The setpoint process communicates with the user process via a motion-request queue containing all the necessary information.

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 its 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 exhibits many advantages over conventional robot-controller designs. The conclusion we 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.

This material is based on work supported by the National Science Foundation under the 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. This work is also supported by a grant from the CNRS project ARA (Automatique et Robotique Avancée), France. Richard Paul receives support as the Ransburg Professor of Robotics. Facilities to perform this research are provided by the Purdue University CIDMAC Project.

References

[1] Ernst H A A (1961) A computer operated mechanical hand. ScD Thesis, Massachusetts Institute of Technology, Massachusetts

[2] Paul R P *Manipulator Language* Workshop On The Research Needed to Advance The State Of Knowledge In Robotics, 15-17 April 1980 (organized by J Birk & R Kelley, supported by NSF)

[3] Paul R P (1981) *Robot Manipulators: Mathematics, Programming and Control* MIT Press, Massachusetts

[4] Will P M & Grossman D D (1975) An experimental system for computer controlled mechanical assembly. *IEEE Trans. Computers* **C-24 9**, 879-888

[5] Derby S (1983) Simulating motion elements of general-purpose robot arms. *Int. J. Robotic Res.* **2** (1)

[6] Castain R H & Paul R P (1982) Polynomial robotic trajectories: a new approach, TR-EE 82-37, December
[7] Hayward V & Paul R P *Robot Manipulator Control Using the C Language Under UNIX* IEEE Workshop on Languages for Automation, Chicago, November, 1983
[8] Inoue H (1974) *Force Feedback In Precise Assembly Tasks*, MIT Artificial Intelligence Laboratory, Memo 308, August
[9] Raiberg M H & Craig J J (1981) Hybrid position/force control of manipulators. *J. Energy Resources Technol.* **103**, June
[10] Salisbury J K. Active stiffness control of a manipulator. In *Cartesian Coordinates* 19th IEEE Conf. on Decision and Control, December 1980, Albuquerque, New Mexico (1983)
[11] Geschke C C (1983) A system for programming and controlling sensor-based robot manipulators. *IEEE Trans, Pattern Matching and Machine Intelligence,* **PAMI-5** (1).
[12] Shimano B E (1978) The kinematic design and force control of computer controlled manipulators, PhD Dissertation, Memo AIM-313, Stanford University, California
[13] Mason M T (1979) Compliance and force control for computer controlled manipulators, MIT TR-515, April
[14] Rosen C A & Nitzan D (1977) Use of sensors in programmable automation. *Computer Magazine*, December
[15] Paul R P Computational Requirements of Third Generation Manipulators