Mechanisms for User Interface

Interactive 3D CAD. Computer integrated manufacturing operations require off-line programming and simulation in order to layout production facilities, model and evaluate design concepts, optimize motion of devices, avoid interference and collisions, minimize process cycle times, maximize productivity, and ensure maximum return on investment. Graphical interfaces, available on some industrial robots, are very effective for conveying information to the operator quickly and efficiently. A graphical interface is most important for design and simulation functions in applications which require frequent reprogramming and setup changes. Several very useful off-line programming software systems are available from third party suppliers (CimStation [SILMA 1992], ROBCAD, IGRIP). These systems use CAD and/or dynamics computer models of commercially available robots to simulate job execution, path motion, and process activities, providing rapid programming and virtual prototyping functions. Interactive off-line CAD is useful for assigning tasks at the management level and for task decomposition, job sequencing, and resource assignment at the task planning level.

Off-Line Robot Teaching and Workcell Programming. Commercial robot or machine tool controllers may have several operator interface mechanisms. These are generally useful at the level of off-line definition or teaching of jobs, which can then be sequenced by the job coordinator or machine coordinator to accomplish assigned tasks. At the lowest level one may program the robot in its operating language, specifying path via points, gripper open/close commands, and so on. Machine tools may require programming in CNC code. These are very tedious functions, which can be avoided by object-oriented and open architecture approaches in well-designed workcells, where such functions should be performed automatically, leaving the user free to deal with other higher level supervisory issues. In such approaches, macros or subroutines are written in machine code which *encapsulate the machine behaviors* (e.g., set speed, open gripper, go to prescribed point). Then, higher level software passes specific parameters to these routines to execute behaviors with specific location and motion details as directed by the job coordinator.

Many robots have a teach pendant, which is a low-level teleoperation device with push buttons for moving individual axes and other buttons to press commanding that certain positions should be memorized. On job execution, a playback mode is switched in, wherein the robot passes through the taught positions to sweep out a desired path. This approach is often useful for teaching multiple complex poses and Cartesian paths.

The job coordinator may be implemented on a programmable logic controller (PLC). PLC programming can be a tedious and time-consuming affair, and in well-designed flexible reconfigurable workcells an object-oriented approach is used to avoid reprogramming of PLCs. This might involve a programming scheme that takes the task plan matrices in section 33.9 as inputs and automatically implements the coordinator using rule-based techniques (e.g., forward chaining, Rete algorithm).

Teleoperation and Man-in-the-Loop Control. Operator interaction at the servocontrol level can basically consist of two modes. In man-in-the-loop control, a human provides or modifies the feedback signals that control a device, actually operating a machine tool or robotic device. In teleoperation, an inner feedback loop is closed around the robot, and a human provides motion trajectory and force commands to the robot in a master/slave relationship. In such applications, there may be problems if extended communications distances are involved, since delays in the communications channel can destabilize a teleoperation system having force feedback unless careful attention is paid to designing the feedback loops to maintain *passivity*. See Lewis [1996] for more details.

33.12 Robot Workcell Programming

The robotic workcell requires programming at several levels [Leu 1985]. At the lower levels one generally uses commercial programing languages peculiar to device manufacturers of robots and CNC machine

tools. At the machine coordination level, robot controllers are also often used with discrete I/O signals and decision making commands. At the job coordination level prorammable logic controllers (PLCs) are often used in medium complexity workcells, so that a knowledge of PLC programming techniques is required. In modern manufacturing and process workcells, coordination may be accomplished using general purpose computers with programs written, for instance, in C.

Robot Programming Languages

Subsequent material in this section is modified from Bailey [1996]. Each robot manufacturer has its own proprietary programming language. The variety of motion and position command types in a programming language is usually a good indication of the robot's motion generation capability. Program commands which produce complex motion should be available to support the manipulation needs of the application. If palletizing is the application, then simple methods of creating position commands for arrays of positions are essential. If continuous path motion is needed, an associated set of continuous motion commands should be available. The range of motion generation capabilities of commercial industrial robots is wide. Suitability for a particular application can be determined by writing test code.

The earliest industrial robots were simple sequential machines controlled by a combination of servomotors, adjustable mechanical stops, limit switches, and PLCs. These machines were generally programmed by a record and play-back method with the operator using a teach pendant to move the robot through the desired path. MHI, the first robot programming language, was developed at Massachusetts Institute of Technology (MIT) during the early 1960s. MINI, developed at MIT during the mid-1970s was an expandable language based on LISP. It allowed programming in Cartesian coordinates with independent control of multiple joints. VAL and VAL II [Shimano et al. 1984], developed by Unimation, Inc., were interpreted languages designed to support the PUMA series of industrial robots. **A manufacturing language (AML)** was a completely new programming language developed by IBM to support the R/S 1 assembly robot. It was a subroutine-oriented, interpreted language which ran on the Series/1 minicomputer. Later versions were compiled to run on IBM compatible personal computers to support the 7535 series of SCARA robots. Several additional languages [Gruver et al. 1984, Lozano-Perez 1983] were introduced during the late 1980s to support a wide range of new robot applications which were developed during this period.

V+, A Representative Robot Language

V+, developed by Adept Technologies, Inc., is a representative modern robot programming language with several hundred program instructions and reserved keywords. V+ will be used to demonstrate important features of robot programming. Robot program commands fall into several categories, as detailed in the following subsections.

Robot Control. Program instructions required to control robot motion specify location, trajectory, speed, acceleration, and obstacle avoidance. Examples of V+ robot control commands are as follows:

MOVE:	Move the robot to a new location.
DELAY:	Stop the motion for a specified period of time.
SPEED:	Set the speed for subsequent motions.
ACCEL:	Set the acceleration and deceleration for subsequent motions.
OPEN:	Open the hand.
CLOSE:	Close the hand.

System Control. In addition to controlling robot motion, the system must support program editing and debugging, program and data manipulation, program and data storage, program control, system definitions and control, system status, and control/monitoring of external sensors. Examples of V+ control instructions are as follows:

EDIT:	Initiate line-oriented editing.
STORE:	Store information from memory onto a disk file.
COPY:	Copy an existing disk file into a new program.
EXECUTE:	Initiate execution of a program.
ABORT:	Stop program execution.
DO:	Execute a single program instruction.
WATCH:	Set and clear breakpoints for diagnostic execution.
TEACH:	Define a series of robot location variables.
CALIBRATE:	Initiates the robot positioning system.
STATUS:	Display the status of the system.
ENABLE:	Turn on one or more system switches.
DISABLE:	Turn off one or more system switches.

Structures and Logic. Program instructions are needed to organize and control execution of the robot program and interaction with the user. Examples include familiar commands such as FOR, WHILE, IF as well as commands like the following:

WRITE:	Output a message to the manual control pendant.
PENDANT:	Receive input from the manual control pendant.
PARAMETER:	Set the value of a system parameter.

Special Functions. Various special functions are required to facilitate robot programming. These include mathematical expressions such as COS, ABS, and SQRT, as well as instructions for data conversion and manipulation, and kinematic transformations such as the following:

BCD:	Convert from real to binary coded decimal.
FRAME:	Compute the reference frame based on given locations.
TRANS:	Compose a transformation from individual components.
INVERSE:	Return the inverse of the specified transformation.

Program Execution. Organization of a program into a sequence of executable instructions requires scheduling of tasks, control of subroutines, and error trapping/recovery. Examples include the following:

PCEXECUTE:	Initiate the execution of a process control program.
PCABORT:	Stop execution of a process control program.
PCPROCEED:	Resume execution of a process control program.
PCRETRY:	After an error, resume execution at the last step tried.
PCEND:	Stop execution of the program at the end of the current execution cycle.

Example Program. This program demonstrates a simple pick and place operation. The values of position variables *pick* and *place* are specified by a higher level executive that then initiates this subroutine:

```
1  .PROGRAM move.parts()
2  ;                    Pick up parts at location "pick" and put them down at "place"
3  parts = 100                                  ; Number of parts to be processed
4  height1 = 25                                 ; Approach/depart height at "pick"
5  height2 = 50                                 ; Approach/depart height at "place"
6  PARAMETER.HAND.TIME = 16                     ; Setup for slow hand
7  OPEN                                         ; Make sure hand is open
8  MOVE start                                   ; Move to safe starting location
9  For i = 1 TO parts                           ; Process the parts
10 APPRO pick, height1                          ; Go toward the pick-up
11 MOVES pick                                   ; Move to the part
12 CLOSEI                                       ; Close the hand
```

```
13  DEPARTS height1                              ; Back away
14  APPRO place, height2                         ; Go toward the put-down
15  MOVES place                                  ; Move to the destination
16  OPENI                                        ; Release the part
17  DEPARTS height2                              ; Back away
18  END                                          ; Loop for the next part
19  TYPE "ALL done.", /I3, parts, "parts processed"
20  STOP                                         ; End of the program
21  .END
```

Off-Line Programming and Simulation. Commercially available software packages (discussed in section 33.11) provide support for off-line design and simulation of 3D worckell layouts including robots, end effectors, fixtures, conveyors, part positioners, and automatic guided vehicles. Dynamic simulation allows off-line creation, animation, and verification of robot motion programs. However, these techniques are limited to verification of overall system layout and preliminary robot program development. With support for data exchange standards [e.g., **International Graphics Exchange Specification (IGES), Virtual Data Acquisition and File Specification (VDAFS), Specification for Exchange of Text (SET)**], these software tools can pass location and trajectory data to a robot control program, which in turn can provide the additional functions required for full operation (operator guidance, logic, error recovery, sensor monitoring/control, system management, etc.).

33.13 Mobile Robots and Automated Guided Vehicles

A topic which has always intrigued computer scientists is that of **mobile robots** [Zheng 1993]. These machines move in generally unstructured environments and so require enhanced decision making and sensors; they seem to exhibit various anthropomorphic aspects since vision is often the sensor, decision making mimics brain functions, and mobility is similar to humans, particularly if there is an onboard robot arm attached. Here are discussed mobile robot research and factory automated guided vehicle (AGV) systems, two widely disparate topics.

Mobile Robots

Unfortunately, in order to focus on higher functions such as decision making and high-level vision processing, many researchers treat the mobile robot as a dynamical system obeying Newton's laws $F = ma$ (e.g., in the potential field approach to motion control, discussed earlier). This simplified dynamical representation does not correspond to the reality of moving machinery which has nonholonomic constraints, unknown masses, frictions, Coriolis forces, drive train **compliance**, wheel slippage, and backlash effects. In this subsection we provide a framework that brings together three camps: computer science results based on the $F = ma$ assumption, nonholonomic control results that deal with a kinematic steering system, and full servo-level feedback control that takes into account all of the vehicle dynamics and uncertainties.

Mobile Robot Dynamics

The full dynamical model of a rigid mobile robot (e.g., no flexible modes) is given by

$$M(\mathbf{q})\ddot{\mathbf{q}} + V_m(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + F(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) + \tau_d = B(\mathbf{q})\tau - A^T(\mathbf{q})\lambda \tag{33.32}$$

which should be compared to Eqs. (33.7) and (33.14). In this equation, M is an inertia matrix, V_m is a matrix of Coriolis and centripetal terms, $\mathbf{F}$ is a friction vector, $\mathbf{G}$ is a gravity vector, and τ_d is a vector of disturbances. The n-vector $\tau(t)$ is the control input. The dynamics of the driving and steering motors