6

Control Architecture

This chapter is devoted to presenting a reference model for the *functional architecture* of an industrial robot's *control system*. The *hierarchical structure* and its articulation into *functional modules* allows the determination of the requirements and characteristics of the *programming environment* and the *hardware architecture*. The architecture refers to robot manipulators, yet its articulation in *levels* also holds for mobile robots.

6.1 Functional Architecture

The *control system* to supervise the activities of a robotic system should be endowed with a number of tools providing the following functions:

- capability of moving physical objects in the working environment, i.e., *manipulation* ability;
- capability of obtaining information on the state of the system and working environment, i.e., *sensory* ability;
- capability of exploiting information to modify system behaviour in a pre-programmed manner, i.e., *intelligence* ability;
- capability of storing, elaborating and providing data on system activity, i.e., *data processing* ability.

An effective implementation of these functions can be obtained by means of a *functional architecture* which is thought of as the superposition of several *activity levels* arranged in a *hierarchical structure*. The lower levels of the structure are oriented to physical motion execution, whereas the higher levels are oriented to logical action planning. The levels are connected by data flows; those directed towards the higher levels regard measurements and/or results of actions, while those directed towards the lower levels regard transmission of directions.

With reference to the control system functions implementing management of the above listed system activities, in general it is worth allocating three

functional models at each level. A first module is devoted to sensory data management (sensory module). A second module is devoted to provide knowledge of the relevant world (modelling module). A third module is devoted to decide the policy of the action (decision module).

More specifically, the *sensory modules* acquire, elaborate, correlate and integrate sensory data in time and space, in order to recognize and measure the system state and environment characteristic; clearly, the functions of each module are oriented to the management of the relevant sensory data for that level.

On the other hand, the *modelling modules* contain models derived on the basis of a priori knowledge of system and environment; these models are updated by the information coming from the sensory modules, while the activation of the required functions is entrusted to the decision modules.

Finally, the *decision modules* perform breakdown of high-level tasks into low-level actions; such task breakdown concerns both breakdown in time of sequential actions and breakdown in space of concurrent actions. Each decision module is entrusted with the functions concerning management of elementary action assignments, task planning and execution.

The functions of a decision module characterize the level of the hierarchy and determine the functions required to the modelling and sensory modules operating at the same level. This implies that the contents of these two modules do not uniquely allow the determination of the hierarchical level, since the same function may be present at more levels depending on the needs of the decision modules at the relative levels.

The functional architecture needs an *operator interface* at each level of the hierarchy, so as to allow an operator to perform supervisory and intervention functions on the robotic system.

The instructions imparted to the decision module at a certain level may be provided either by the decision module at the next higher level or by the operator interface, or else by a combination of the two. Moreover, the operator, by means of suitable communication tools, can be informed on the system state and thus can contribute his/her own knowledge and decisions to the modelling and sensory modules.

In view of the high data flow concerning the exchange of information between the various levels and modules of the functional architecture, it is worth allocating a shared *global memory* which contains the updated estimates on the state of the whole system and environment.

The structure of the reference model for the functional architecture is represented in Fig. 6.1, where the *four hierarchical levels* potentially relevant for robotic systems in industrial applications are illustrated. Such levels regard definition of the *task*, its breakdown into elementary *actions*, assignment of *primitives* to the actions, and implementation of control actions on the *servo*-manipulator. In the following, the general functions of the three modules at each level are described.

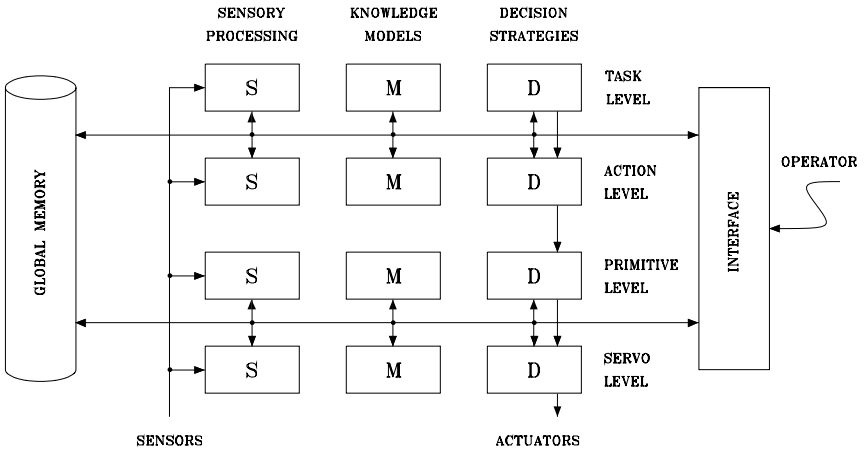

Fig. 6.1. Reference model for a control system functional architecture

At the *task level*, the user specifies the task which the robotic system should execute; this specification is performed at a high level of abstraction. The goal of the desired task is analyzed and broken down into a sequence of actions which are coordinated in space and time and allow implementation of the task. The choice of actions is performed on the basis of knowledge models as well as of the scene of interest for the task. For instance, consider the application of a robot installed in an assembly line which is required to perform a specific assembly task. To define the elementary actions that have to be transmitted to the decision module at the next lower level, the decision module should consult its knowledge base available in the modelling module, e.g., type of assembly, components of the object to assembly, assembly sequence, and choice of tools. This knowledge base should be continuously updated by the information provided by the sensory module concerning location of the parts to assembly; such information is available by means of a high-level vision system operating in a scarcely structured environment, or else by means of simple sensors detecting the presence of an object in a structured environment.

At the *action level*, the symbolic commands coming from the task level are translated into sequences of intermediate configurations which characterize a motion path for each elementary action. The choice of the sequences is performed on the basis of models of the manipulator and environment where the action is to take place. With reference to one of the actions generated by the above assembly task, the decision module chooses the most appropriate coordinate system to compute manipulator's end-effector poses, by separating translation from rotation if needed; it decides whether to operate in the joint or operational space, it computes the path or via points, and for the latter it defines the interpolation functions. By doing so, the decision module should compare the sequence of configurations with a model of the manipulator as

well as with a geometric description of the environment, which are both available in the modelling model. In this way, action feasibility is ascertained in terms of obstacle-collision avoidance, motion in the neighbourhood of kinematically singular configurations, occurrence of mechanical joint limits, and eventually utilization of available redundant DOFs. The knowledge base is updated by the information on the portion of scene where the single action takes place which is provided by the sensory module, e.g., by means of a low-level vision system or range sensors.

At the *primitive level*, on the basis of the sequence of configurations received by the action level, admissible motion trajectories are computed and the control strategy is decided. The motion trajectory is interpolated so as to generate the references for the servo level. The choice of motion and control primitives is conditioned by the features of the mechanical structure and its degree of interaction with the environment. Still with reference to the above case study, the decision module computes the geometric path and the relative trajectory on the basis of the knowledge of the manipulator dynamic model available in the modelling module. Moreover, it defines the type of control algorithm, e.g., decentralized control, centralized control, or interaction control; it specifies the relative gains; and it performs proper coordinate transformations, e.g., kinematic inversion if needed. The sensory module provides information on the occurrence of conflicts between motion planning and execution, by means of, e.g., force sensors, low-level vision systems and proximity sensors.

At the *servo level*, on the basis of the motion trajectories and control strategies imparted by the primitive level, control algorithms are implemented which provide the driving signals to the joint servomotors. The control algorithm operates on error signals between the reference and the actual values of the controlled quantities, by utilizing knowledge of manipulator dynamic model, and of kinematics if needed. In particular, the decision module performs a microinterpolation on the reference trajectory to exploit fully the dynamic characteristic of the drives; it computes the control law, and it generates the (voltage or current) signals for controlling the specific drives. The modelling module elaborates the terms of the control law depending on the manipulator current configuration and pass them to the decision module; such terms are computed on the basis of knowledge of manipulator dynamic model. Finally, the sensory module provides measurements of the proprioceptive sensors (position, velocity and contact force if needed); these measurements are used by the decision module to compute the servo errors and, if required, by the modelling module to update the configuration-dependent terms in the model.

The specification of the functions associated with each level points out that the implementation of such functions should be performed at different time rates, in view of their complexity and requirements. On one hand, the functions associated with the higher levels are not subject to demanding real-time constraints, since they regard planning activities. On the other hand, their

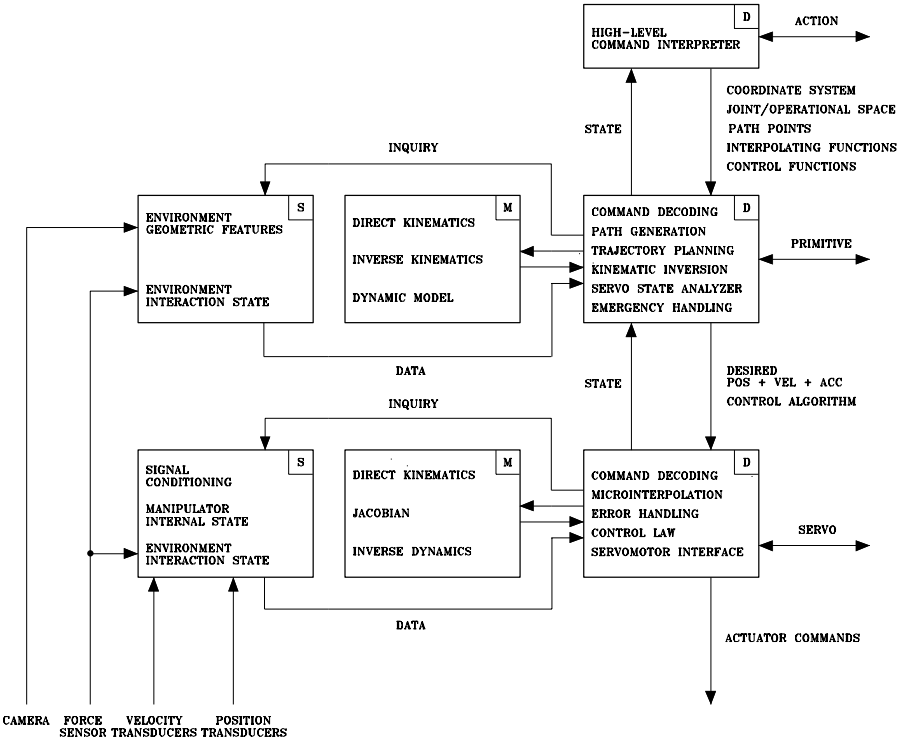

Fig. 6.2. Hierarchical levels of a functional architecture for industrial robots

complexity is notable, since scheduling, optimization, resource management and high-level sensory system data processing are required to update complex models.

At the lowest level, demanding real-time operation prevails in order to obtain high dynamic performance of the mechanical structure. The above remarks lead to the conclusion that, at the servo level, it is necessary to provide the driving commands to the motors and to detect the proprioceptive sensory data at sampling rates of the order of the millisecond, while sampling rates of the order of the minute are admissible at the task level.

With respect to this reference model of functional architecture, current industrial robot's control systems are not endowed with all the functions illustrated, because of both technology and cost limitations. In this regard, the task level is not implemented at all since there do not yet exist effective and reliable application software packages allowing support of the complex functions required at this level.

It is worth characterizing those functional levels of the reference models which are typically implemented in *advanced industrial robot's control systems*. The details of Fig. 6.2 show what follows:

- The modelling and sensory modules are always present at the lowest level, because of demanding requirements at the servo level for high dynamic performance robots to be employed even in relatively simple applications.
- At the primitive level, the modelling module is usually present while the sensory module is present only in a reduced number of applications that require robot interaction with a less structured environment.
- At the action level, the decision module is present only as an interpreter of the high-level commands imparted by the operator. All the task breakdown functions are entrusted to the operator, and thus the modelling and sensory module are absent at this level. Possible checking of action feasibility is moved down to the primitive level where a modelling module exists.

In view of the highly-structured reference model of functional architecture illustrated above, evolution of the control system towards more and more powerful capabilities is possible. In fact, one may foresee that information technology progress may allow the addition of hierarchically higher levels than the task level. These should functionally characterize complex tasks to be broken down into elementary tasks and yet, at an even higher level, missions to be broken down into complex tasks. A six-level hierarchical structure of the above kind has been proposed as the reference model for the functional architecture of the control system of a service robotic system for space applications (NASREM). In this framework, one may allocate the functions required to *advanced robotics* systems devoted to field or service applications, as discussed in Sect. 1.4.

6.2 Programming Environment

Programming a robotic system requires definition of a *programming environment* supported by suitable *languages*, which allows the operator imparting the task directions that the robot should execute. The programming environment is entrusted not only with the function of translating statements by means of a suitable language, but also with the function of checking correct execution of a task being executed by the robot. Therefore, robot programming environments, besides having some features in common with computer programming environments, present a number of issues related to the observation that program execution produces effects on the physical world. In other words, even if a very accurate description of physical reality is available in the programming environment, a number of situations will unavoidably occur which have not been or cannot be predicted.

As a consequence, a robot programming environment should be endowed with the following features:

- real-time operating system,
- world modelling,
- motion control,

- sensory data reading,
- interaction with physical system,
- error detection capability,
- recovery of correct operational functions,
- specific language structure.

Therefore, the requirements on a programming environment may naturally stem from the articulation into models of the preceding reference model of functional architecture. Such an environment will be clearly conditioned by the level of the architecture at which operator access is allowed. In the following, the requirements imposed on the programming environment by the functions respectively characterizing the sensory, modelling and decision modules are presented with reference to the hierarchical levels of the functional architecture.

Sensory data handling is the determining factor which qualifies a programming environment. At the servo level, real-time proprioceptive sensory data conditioning is required. At the primitive level, sensory data have to be expressed in the relevant reference frames. At the action level, geometric features of the objects interested to the action have to be extracted by high-level sensory data. At the task level, tools allowing recognition of the objects present in the scene are required.

The ability of *consulting knowledge models* is a support for a programming environment. At the servo level, on-line numerical computation of the models utilized by control algorithms is to be performed on the basis of sensory data. At the primitive level, coordinate transformations have to be operated. At the action level, it is crucial to have tools allowing system simulation and CAD modelling of elementary objects. At the task level, the programming environment should assume the functions of an expert system.

Decision functions play a fundamental role in a programming environment, since they allow the definition of the flow charts. At the servo level, on-line computation ability is required to generate the driving signals for the mechanical system. At the primitive level, logic conditioning is to be present. At the action level, process synchronization options should be available in order to implement nested loops, parallel computation and interrupt system. At the task level, the programming environment should allow management of concurrent processes, and it should be endowed with tools to test for, locate and remove mistakes from a program (debuggers) at a high-interactive level.

The evolution of programming environments has been conditioned by technology development of computer science. An analysis of this evolution leads to finding three generations of environments with respect to their functional characteristics, namely, *teaching-by-showing*, *robot-oriented programming*, and *object-oriented programming*. In the evolution of the environments, the next generation usually incorporates the functional characteristics of the previous generation.

This classification regards those features of the programming environment relative to the operator interface, and thus it has a direct correspondence with the hierarchical levels of the reference model of functional architecture. The functions associated with the servo level lead to understanding that a programming environment problem does not really exist for the operator. In fact, low-level programming concerns the use of traditional programming languages (Assembly, C) for development of real-time systems. The operator is only left with the possibility of intervening by means of simple command actuation (point-to-point, reset), reading of proprioceptive sensory data, and limited editing capability.

6.2.1 Teaching-by-Showing

The first generation has been characterized by programming techniques of *teaching-by-showing* type. The operator guides the manipulator manually or by means of a teach pendant along the desired motion path. During motion execution, the data read by joint position transducers are stored and thus they can be utilized later as references for the joint drive servos; in this way, the mechanical structure is capable of executing (playing back) the motion taught by a direct acquisition on the spot.

The programming environment does not allow implementation of logic conditioning and queuing, and thus the associated computational hardware plays elementary functions. The operator is not required to have special programming skill, and thus he/she can be a plant technician. The set-up of a working program obviously requires the robot to be available to the operator at the time of teaching, and thus the robot itself has to be taken off production. Typical applications that can be solved by this programming technique include spot welding, spray painting and, in general, simple palletizing.

With regard to the reference model of functional architecture, a programming environment based on the teaching-by-showing technique allows operator access at the primitive level.

The drawbacks of such an environment may be partially overcome by the adoption of simple programming languages which allow:

- the acquisition of a meaningful posture by teaching,
- the computation of the end-effector pose with respect to a reference frame, by means of a direct kinematics transformation,
- the assignment of a motion primitive and the trajectory parameters (usually, velocity as a percentage of the maximum velocity),
- the computation of the servo references, by means of an inverse kinematics transformation,
- the teaching sequences to be conditioned to the use of simple external sensors (presence of an object at the gripper),
- the correction of motion sequences by using simple text editors,
- simple connections to be made between subsets of elementary sequences.

Providing a teaching-by-showing environment with the the above-listed functions can be framed as an attempt to develop a structured programming environment.

6.2.2 Robot-oriented Programming

Following the advent of efficient low-cost computational means, *robot-oriented* programming environments have been developed. The need for interaction of the environment with physical reality has imposed integration of several functions, typical of high-level programming languages (BASIC, PASCAL), with those specifically required by robotic applications. In fact, many robot-oriented languages have retained the teaching-by-showing programming mode, in view of its natural characteristic of accurate interface with the physical world.

Since the general framework is that of a computer programming environment, two alternatives have been considered:

- to develop *ad hoc languages* for robotic applications,
- to develop robot *program libraries* supporting standard programming languages.

The current situation features the existence of numerous new proprietary languages, whereas it would be desirable to develop either robotic libraries to be used in the context of consolidated standards, or new general-purpose languages for industrial automation applications.

Robot-oriented languages are *structured programming* languages which incorporate high-level statements and have the characteristic of an interpreted language, in order to obtain an interactive environment allowing the programmer to check the execution of each source program statement before proceeding to the next one. Common features of such languages are:

- text editor,
- complex data representation structures,
- extensive use of predefined state variable,
- execution of matrix algebra operations,
- extensive use of symbolic representations for coordinate frames,
- possibility to specify the coordinated motion of more frames rigidly attached to objects by means of a single frame,
- inclusion of subroutines with data and parameter exchange,
- use of logic conditioning and queuing by means of flags,
- capability of parallel computing,
- functions of programmable logic controller (PLC).

With respect to the reference model of functional architecture, it can be recognized that a robot-oriented programming environment allows operator access at the action level.

In view of the structured language characteristic, the operator in this case should be an expert language programmer. Editing an application program may be performed off line, i.e., without physical availability of the robot to the operator; off-line programming demands a perfectly structured environment, though. A robotic system endowed with a robot-oriented programming language allows execution of complex applications where the robot is inserted in a work cell and interacts with other machines and devices to perform complex tasks, such as part assembly.

Finally, a programming environment that allows access at the task level of a reference model of functional architecture is characterized by an *object-oriented* language. Such an environment should have the capability of specifying a task by means of high-level statements allowing automatic execution of a number of actions on the objects present in the scene. Robot programming languages belonging to this generation are currently under development and thus they are not yet available on the market. They can be framed in the field of expert systems and artificial intelligence.

6.3 Hardware Architecture

The hierarchical structure of the functional architecture adopted as a reference model for an industrial robot's control system, together with its articulation into different functional modules, suggests hardware implementation which exploits distributed computational resources interconnected by means of suitable communication channels. To this end, it is worth recalling that the functions implemented in current control systems regard the three levels from servo to action, with a typically limited development of the functions implemented at the action level. At the servo and primitive levels, computational capabilities are required with demanding real-time constraints.

A general model of the *hardware architecture* for the control system of an industrial robot is illustrated in Fig. 6.3. In this figure, proper *boards* with autonomous computational capabilities have been associated with the functions indicated in the reference model of functional architecture of Fig. 9.2. The boards are connected to a *bus*, e.g., a VME bus, which allows support of the communication data flow; the bus bandwidth should be wide enough so as to satisfy the requirements imposed by real-time constraints.

The *system* board is typically a CPU endowed with:

- a microprocessor with mathematical coprocessor,
- a bootstrap EPROM memory,
- a local RAM memory,
- a RAM memory shared with the other boards through the bus,
- a number of serial and parallel ports interfacing the bus and the external world,
- counters, registers and timers,

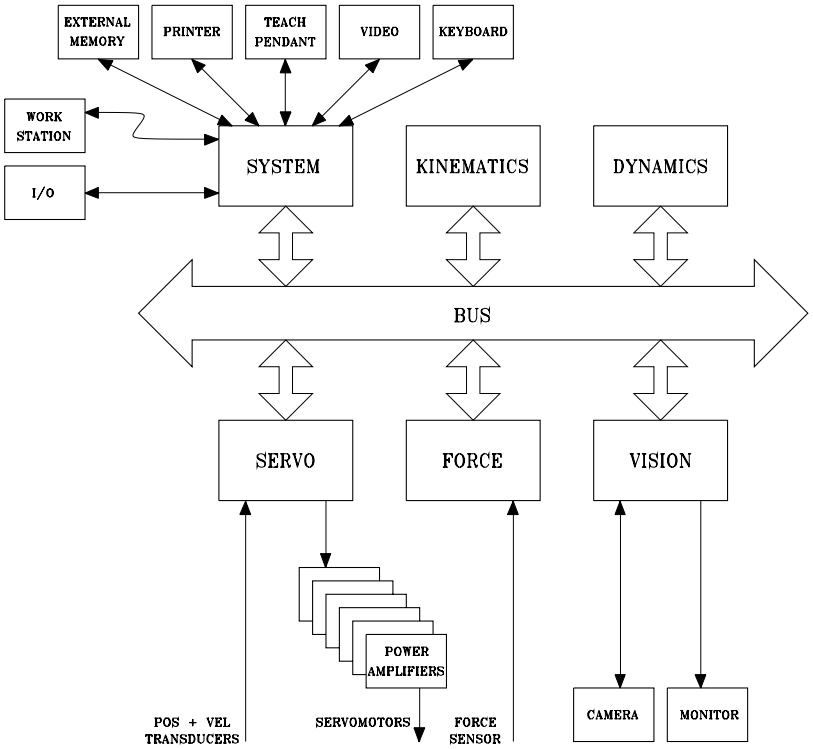

Fig. 6.3. General model of the hardware architecture of an industrial robot's control system

- an interrupt system.

The following functions are to be implemented in the system board:

- operator interface through teach pendant, keyboard, video and printer,
- interface with an external memory (hard disk) used to store data and application programs,
- interface with workstations and other control systems by means of a local communication network, e.g., Ethernet,
- I/O interface with peripheral devices in the working area, e.g., feeders, conveyors and ON/OFF sensors,
- system bootstrap,
- programming language interpreter,
- bus arbiter.

The other boards facing the bus may be endowed, besides the basic components of the system board, with a supplementary or alternative processor

(DSP, Transputer) for implementation of computationally demanding or dedicated functions. With reference to the architecture in Fig. 6.3, the following functions are implemented in the *kinematics* board:

- computation of motion primitives,
- computation of direct kinematics, inverse kinematics and Jacobian,
- test for trajectory feasibility,
- handling of kinematic redundancy.

The *dynamics* board is devoted to

- computation of inverse dynamics.

The *servo* board has the functions of:

- microinterpolation of references,
- computation of control algorithm,
- digital-to-analog conversion and interface with power amplifiers,
- handling of position and velocity transducer data,
- motion interruption in case of malfunction.

The remaining boards in the figure have been considered for the sake of an example to illustrate how the use of sensors may require local processing capabilities to retrieve significant information from the given data which can be effectively used in the sensory system. The *force* board performs the following operations:

- conditioning of data provided by the force sensor,
- representation of forces in a given coordinate frame.

The *vision* board is in charge of:

- processing data provided by the camera,
- extracting geometric features of the scene,
- localizing objects in given coordinate frames.

Although the boards face the same bus, the frequency at which data are exchanged needs not to be the same for each board. Those boards connected to the proprioceptive sensors indeed need to exchange date with the robot at the highest possible frequency (from 100 to 1000 Hz) to ensure high dynamic performance to motion control as well as to reveal end-effector contact in a very short time.

On the other hand, the kinematics and dynamics boards implement modelling functions and, as such, they do not require data update at a rate as high as that required by the servo board. In fact, manipulator postures do not vary appreciably in a very short time, at least with respect to typical operational velocities and/or accelerations of current industrial robots. Common sampling frequencies are in the range of 10 to 100 Hz.

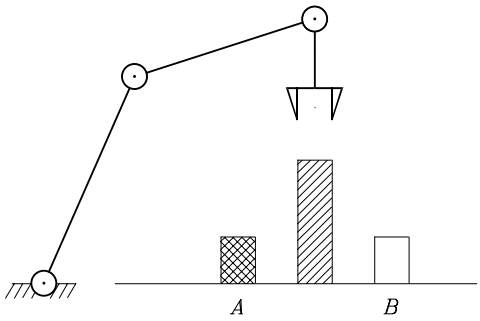

Fig. 6.4. Object pick-and-place task

Also the vision board does not require a high update rate, both because the scene is generally quasi-static, and because processing of interpretive functions are typically complex. Typical frequencies are in the range of 1 to 10 Hz.

In summary, the board access to the communication bus of a hardware control architecture may be performed according to a multirate logic which allows the solution of bus data overflow problems.

Bibliography

The features of robot control architectures are presented in [230, 25]. The NASREM architecture model has been proposed in [3]. For robot programming see [225, 139, 91]. More advanced control architectures based on artificial intelligence concepts are discussed in [8, 158].

Problems

6.1. With reference to the situation illustrated in Fig. 6.4, describe the sequence of actions required from the manipulator to pick up an object at location A and place it at location B.

6.2. For the situation of Problem 6.1, find the motion primitives in the cases of given via points and given path points.

6.3. The planar arm indicated in Fig. 6.5 is endowed with a wrist force sensor which allows the measurement of the relevant force and moment components for the execution of a peg-in-hole task. Draw the flow chart for writing a program to execute the described task.

6.4. A palletizing problem is represented in Fig. 6.6. Sixteen equal objects have to be loaded on the pallet. The manipulator's end-effector has to pick

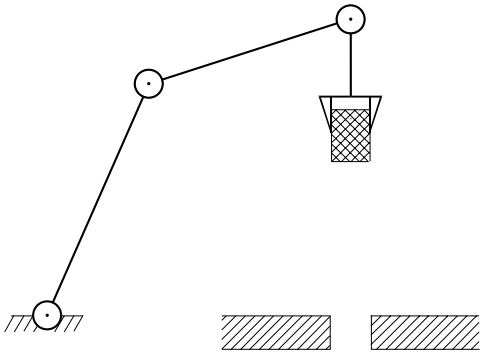

Fig. 6.5. Peg-in-hole task

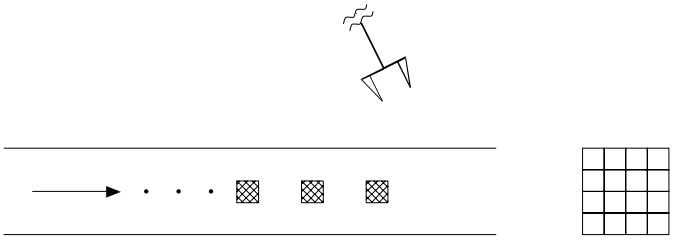

Fig. 6.6. Palletizing task of objects available on a conveyor

up the objects from a conveyor, whose feeding is commanded by the robot in such a way that the objects are always found in the same location to be picked. Write a PASCAL program to execute the task.