Robotica (1992) volume 10, pp 45–56

Automatic simulation of a robot program for a sequential manufacturing process

Witold Jacak* and Jerzy W. Rozenblit†

(Received in Final Form: February 26, 1991)

SUMMARY

This paper presents a framework for the design of a hierarchical simulator of a robotized sequential technological process. The framework employs concepts of discrete event simulation modelling. The simulator consists of two layers: the simulator of a robot and technological process, and the interpreter and planner of robot tasks. A formal specification of both layers is presented. The proposed simulation approach is expected to result in significant improvements in the robot task plan generation and in higher efficiency of a technological process.

KEYWORDS: Robot Motion; Simulation; Process Modelling; Sequential manufacturing.

1. INTRODUCTION

A robot is an important tool in modern technological processes. The effectiveness of employing a robot to control a technological process depends on the following factors: a) The arrangement of workcells and devices on the robot's workscene; b) The robot's control program; and c) Parameters of individual program instructions (e.g. the speed of motion, acceleration, etc.).

In order to verify the correctness of a robot's program and to determine how effective the robot is in carrying out the program, we propose to develop a simulator of a technological process. We employ the simulator to investigate several modes of the robot's operation that arise from varying the parameter values of its control program. The choice of such parameters affects the efficiency of a robot servicing a technological process. Those parameters are usually defined as based on the geometry of the robot's workscene and the pre-planned trajectories of the robot's motion. They determine the time and energy required for the robot to complete a task and subsequently impact the overall utilization of technological devices.[1]

The existing planners of action for manufacturing systems treat time and energy consumption parameters as input data.[2–4] The planners do not have the facilities to verify and/or modify the parameters once they have been input. The planning systems do not comprise modules that could enable a user to plan and simulate elementary robot actions based on a geometric model of the workscene. Therefore we cannot generate alternative trajectories of robot movements (associated with robot actions) using the conventional planning systems. If such trajectories were available for analysis, we could improve the efficiency of a manufacturing process by appropriately modifying its action plans.

The action plan for a manufacturing process determines the robot's program of actions in servicing the process. The program is usually a sequence of instructions expressed in a task-oriented robot programming language[5,6] (*TORPL*). Variant interpretations of the language commands result in different realizations of a robot action plans. Variant plans may be generated by modifying the robot's workscene, or by redefining the motion's dynamic requirements (e.g. time-minimal motion, energy-minimal motion). This induces us to specify a system that would automatically verify the semantics of the robot's program.

We propose a hierarchical robot planning and programming system that facilitates simulation and testing of robot tasks in a given time window. It also allows the operator to vary and modify the robot's motion parameters. The system consists of two basic layers:

Layer 1: The simulation layer that comprises:
- a) A simulator of the technological process
- b) A simulator of the robot actions based on a discrete event model of the manufacturing process.

Layer 2: The interpretation and planning layer for each individual robot action. A geometric model of the work-cells is employed as basis for plan generation.

The simulation layer (Layer 1) automatically synthesizes a model of a technological process by analyzing the technological operations applied to details. This results in a discrete event model specification which serves as a basis for simulating the robot's actions. To carry out the simulation, the system must have the knowledge of how individual actions are carried out in the process. This knowledge must be available in order to schedule a correct sequence of actions. (Recall that each action has an associated set of commands of the robot programming language).

Layer 2 supports scheduling. It interprets and simulates the command based on a geometric model of the robot and its workscene. The planning component of

* Institute of Technical Cybernetics, Technical University of Wroclaw, Wyb. Wyspianskiego 27, 50-370 Wroclaw (Poland).
† Department of Electrical and Computer Engineering, University of Arizona, Tucson, Arizona 85721 (U.S.A.).

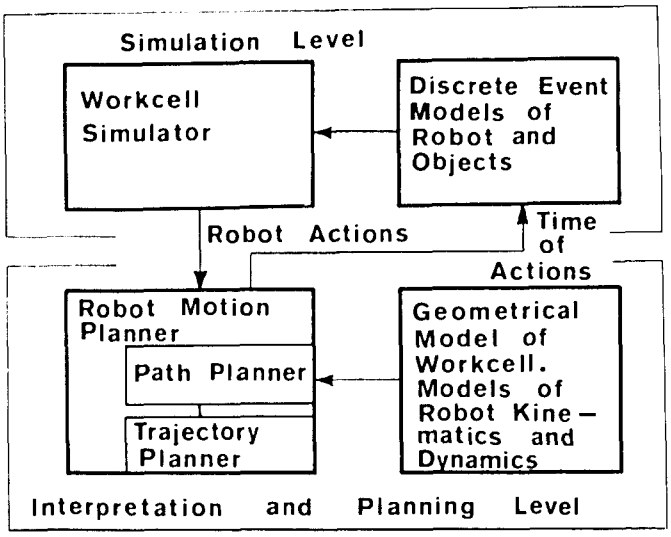


Fig. 1. Structure of a robot task simulation and testing system.

this layer automatically formulates the robot's motion trajectory. It also provides time parameters for the robot's actions. These parameters are used in the simulation carried out at Layer 1.

The structure of the proposed system is illustrated in Figure 1.

2. BASIC NOTIONS

In general, a technological process can be represented by a pair

$$\text{Process} = (\text{Operations}, <) \tag{1}$$

where

$$\text{Operations} = \{\text{Op}_i \mid i = 1, \ldots, N\} \tag{2}$$

is the set of technological operations (e.g. tooling operations, assembly operations), necessary to process the details.

$$< \subset \text{Operations} \times \text{Operations}$$

is the operations precedence relation, and $\text{Op}_i < \text{Op}_j$ means that operation i "must precede" operation j.

The operation precedence relation is irreflexive, antisymmetrical and transitive, and generates an antisymmetrical quasi-order in a set of technological operations. The precedence relation is used to create and AND/OR graphs that represents all valid operations sequences.[2,4]

In this paper, we restrict the analysis to a class of sequential technological processes which transform only one type of a part. Such process represents a wide range of applications (e.g. tooling processes). In this case, a work detail operated on by robot is described by a state

$$\text{State}_d = (\bar{v}_j \mid j \in I_d) \tag{3}$$

where: I_d – is the current set of detail properties, $\bar{v}_j = (v_{j1}, \ldots, v_{jn(j)})$ is a vector of parameters described by the j-th property of the detail. Hence, the technological operation Op_k can be treated as a function

$$\text{Op}_k: \mathbb{R}^m \to \mathbb{R}^l. \tag{4}$$

This function defines the change of properties and values of their parameters that describe the state of detail, i.e. $v = \text{Op}_k(w)$ is a composed vector of detail's parameters after the operation $(v = (v_{11}, \ldots, v_{1n(1)}, v_{21}, \ldots, v_{2n(2)}, \ldots, v_{ln(l)}) \in \mathbb{R}^l)$ has been executed on a vector $w \in \mathbb{R}^m$.

A sequential technological process is represented by a composition function PROC,[7] where:

$$v = \text{PROC}(x) = (\text{Op}_N \circ \text{Op}_{N-1} \circ \cdots \circ \text{Op}_1)(w) \tag{5}$$

and $w \in \mathbb{R}^n$, $v \in \mathbb{R}^m$ are composed vectors of a detail's parameters before and after the process has been carried out, respectively; N is the number of technological operations necessary to process a detail. The sequence of operations from PROC should satisfy the operations precedence relation i.e. if $\text{Op}_k < \text{Op}_l$ then $k < l$.

We assume that each workcell (device) is assigned an operation to process a detail. A technological line consists of a series of workcells (devices) $(M_i \mid i = 1, \ldots, N)$. In addition, there are two special workcells, the feeder conveyer M_0 and the output conveyer M_F. Both M_0 and M_F can serve as input/output devices for other technological lines. Each workcell has its own program for processing a detail. The program determines the time necessary to carry out the operation assigned to a workcell. The operation is executed by a robot. The robot also transports a detail among the workcells. We do not provide any facilities for queueing details at the workcells.

The above assumptions are defined for the sake of brevity of this paper. They can be easily relaxed and the model presented here can be extended to a much more complex class of manufacturing systems. We now proceed to present the formal basis underlying our appproach to simulation modelling of the sequential technological process.

2.1 Formal concepts for robot and assembly line modelling

Our main objective in defining the framework to build the automatic tester and robot action planner is to provide a means for rapid modelling and simulation of the entire technological line. To model the technological line we employ the Discrete Event System Specification (DEVS) formalism.[8–10]

The DEVS hierarchical, modular formalism, closely parallels the abstract set theoretic formulation developed by Zeigler.[8] In such a formalism, one must specify basic models from which larger ones are built, and how these models are connected together in a hierarchical fashion. A basic model, called an atomic DEVS is defined by the following structure[8]

$$M = (X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, t_a) \tag{6}$$

where

X is a set, the external input event types
S is a set, the sequential states
Y is a set, the external output event types
δ_{int} is a function, the internal transition specification
δ_{ext} is a function, the external transition specification

λ is a function, the output function
t_a is a function, the time advance function
with the following constraints:

(a) The total state set of the system specified by M is

$$Q = \{(s, e) \mid s \in S, 0 \leq e \leq t_a(s)\}; \quad (7)$$

(b) δ_{int} is a mapping from S to S:

$$\delta_{int}: S \rightarrow S; \quad (8)$$

(c) δ_{ext} is a function

$$\delta_{ext}: Q \times X \rightarrow S; \quad (9)$$

(d) t_a is a mapping from S to the non-negative reals with infinity:

$$t_a: S \rightarrow \mathbb{R}, \quad (10)$$

and

(e) λ is a mapping from Q to Y:

$$\lambda: Q \rightarrow Y. \quad (11)$$

An interpretation of the DEVS and a full explication of the semantics of the DEVS are in refs. 8, 9.

The second form of models, called a coupled model, tells how to couple several component models together to form a new model. This latter model can itself be employed as a component in a larger coupled model, thus giving rise to a hierarchical construction. A coupled DEVS is defined as a structure:[8]

$$DN = (D, M_i, I_i, Z_{ij}, \text{SELECT}) \quad (12)$$

where
D is a set, the component names;
for each i in D,
M_i is a component;
I_i is a set, the influences of i;
and for each j in I_i
Z_{ij} is a function, the i-to-j output translation;
and
SELECT is a function, the tie-breaking selector
with the following constraints:

$$M_i = (X_i, S_i, Y_i, \delta_i, \lambda_i, t_{ai}) \quad (13)$$

I_i is a subset of D, i is not in I_i

$$Z_{ij}: Y_i \rightarrow X_j \quad (14)$$

SELECT: subset of $D \rightarrow D$
such that for any non-empty subset E, SELECT(E) is in E.

The formal model specification in multi-facetted methodology consists in specifying the system entity structure, attached variable types (called descriptive variables), pruning, and then specifying a discrete event model for the components indentified by the pruned entity structure. A selection of input, output and state variables results in the model's static structure. The definiton of transition and output functions adds dynamic components to the DEVS specification.

In the ensuring section, we describe the design of a simulator of the robotized technological line.

3. ROBOTIZED TECHNOLOGICAL LINE SIMULATOR

Recall that our considerations are restricted to sequential technological process. A process is represented by the following graph:

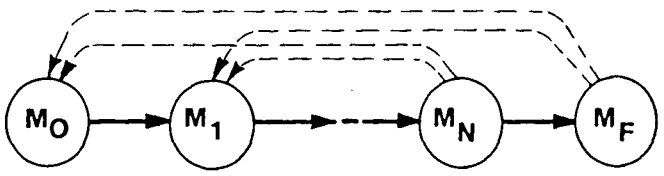


Fig. 2. Graph representation of a sequential technological process.

Each vertex M corresponds to a technological device (workcell). The arcs represent possible robot movements between workcells. An arc depicted in Figure 2 by a continuous line symbolizes the transport of a detail between workcells. A dashed line represents a possible "empty" move of the robot servicing the line. An empty move is necessary to transfer the robot to a workcell which is requesting service.

The robot's actions in a system such as depicted in Figure 2 can be realized by elementary operations. Each elementary action has an associated set of instructions in the task-oriented-robot programming language[6] (*TORPL*). The basic macroinstructions of *TORPL* are:

MOVE {EMPTY / HOLDING} TO "position"
PICKUP "part" AT "position"
PLACE "part" ON "position"
WAIT FOR "sensor input signal"
START "output signal"

The MOVE operation can be specialized with respect to the type of the end-effector paths or the state of the effector (EMPTY, HOLDING).[6] In some cases, an exact geometrical path of motion is required.

The instructions listed above can be combined into a higher level macro: PICK-AND-PLACE.[11-12] This macro defines the following set of instructions:

MOVE EMPTY TO "position A"
PICKUP "part" AT "position A"
MOVE HOLDING "part" TO "position B"
PLACE "part" ON "position B"

The above instructions are used to synthesize the robot's action program. The synthesis process requires an introduction of conditional instructions that depend on the states of each device M_i of the assembly line. Therefore, in order to define a simulator of the program, we have to model conditions that enable each program instruction. To do that, we model each device M_i using DEVS. Thus M_i is given by the following tuple:

$$M_i = (X_i, S_i, Q_i, \delta^i_{int}, \delta^i_{ext}, t^i_a) \quad \text{for } i = 0, 1, \ldots, N, F \quad (15)$$

where each component of the tuple is defined as presented in Section 2.

The state set of each M_i is defined as $S_i = \{A, B, C\}$ where

A denotes that DEVICE IS NOT WORKING AND IS FREE
B denotes that DEVICE IS NOT WORKING AND IS NOT FREE
C denotes that DEVICE IS WORKING AND IS NOT FREE

The state sets of devices M_0 and M_F are $S_0 = \{B\}$ and $S_F = \{A\}$, respectively.

Let us assume that the "i-th position" denotes a location at which a detail is placed on a workcell. The set of external events for M_i is defined by the commands of the *TORPL*, namely:

$$X_i = \{e1_i, e2_i, e0\} \quad \text{for } i = 1, \ldots, N$$

where:

$$\begin{aligned} e1_i &= \text{PLACE "part" ON "}i\text{-th position"} \\ e2_i &= \text{PICKUP "part" AT "}i\text{-th position"} \\ e0 &= \text{NOTHING DO} \end{aligned} \tag{16}$$

Given the state set and the external event set, we now define the transition functions. The internal transition function is specified as follows:

$$\delta^i_{\text{int}}: S_i \rightarrow S_i \quad \text{for } 1, \ldots, N \tag{17}$$

where

$$\begin{aligned} \delta^i_{\text{int}}(A) &= A \\ \delta^i_{\text{int}}(B) &= B \\ \delta^i_{\text{int}}(C) &= B \end{aligned}$$

The external transition function is defined by:

$$\begin{aligned} &\delta^i_{\text{ext}}((A, t), e1_i) = C \quad \text{for } i = 1, \ldots, N \\ &\delta^i_{\text{ext}}((B, T), e2_i) = A \\ &\delta^i_{\text{ext}}((s, t), e0) = s \\ &\delta^i_{\text{ext}}(. , .) = \text{Failure for other states and events} \\ &\text{The pairs } (A, t), (B, t), (s, t) \in Q_i \end{aligned} \tag{18}$$

For the device M_0 and M_F the functions are:

$$\begin{aligned} &\delta^0_{\text{int}}(B) = B \\ &\delta^0_{\text{ext}}((B, t), e2_0) = B \quad \text{and} \\ &\delta^0_{\text{ext}}((B, t), e1_0) = \text{Failure} \end{aligned} \tag{19}$$

and

$$\begin{aligned} &\delta^F_{\text{int}}(A) = A \\ &\delta^F_{\text{ext}}((A, t), e1_F) = A \end{aligned} \tag{20}$$

and

$$\delta^F_{\text{ext}}((A, t), e2_F) = \text{Failure}$$

The time advance functions for M_i determine the time needed to process a detail in the i-th device. They are defined as follows:

$$t^i_{a(s_i)} = \begin{cases} \infty & \text{if } s_i = A \\ \infty & \text{if } s_i = B \\ \tau_i - \text{the tooling time} & \\ \quad \text{if } s_i = C & \text{for } i = 1, \ldots, N. \end{cases} \tag{21}$$

The above is a complete model of technological devices of the assembly line. The activation of each device M_i is caused by an external event generated by the model of the robot. Such a model is realized by a generator of the experimental frame[10] associated with the technological line model. We now briefly explain the concept of experimental frame. (For a detailed description, we refer the reader to refs. 8 & 10).

We separate the model description from a simulation experiment under which the model is observed. A set of circumstances under which a model is observed and experimented with is called an experimental frame. Zeigler[8] has shown that an experimental frame can be realized as a coupling of three components: a generator (supplying a model with an input segment reflecting the effects of the external environment upon a model), an acceptor (a device monitoring a simulation run), and a transducer (collecting and processing model output data). Figure 3 depicts the structural realization of an experimental frame. The specification of experimental frames in the DEVS-Scheme environment is equivalent to that of specifying basic models and their corresponding couplings.

Experimental frames reflect I/O performance design requirements. For example, an experimental frame for evaluating the average task processing time by a robot could have the following constituents: a generator producing a workload of tasks for the robot, an acceptor monitoring the observation interval, and a transducer recording the times of task completions, and computing the average task execution time.

The events generated by the robot depend on the states of the workcells M_i. Thus, we must define an acceptor which will observe the states of each workcell. The acceptor is defined by the following DEVS:[8]

$$A = (X_a, S_a, \delta^a_{\text{ext}}) \tag{22}$$

where:

$$\begin{aligned} X_a &= \{(x, t) \mid s \in S \quad \text{and} \quad t \in \text{Time}\} \\ S &= X\{S_i \mid i = 1, \ldots, N\} \end{aligned}$$

and Time is the time base.

The input port of the acceptor receives state descriptions of each M_i. The role of the acceptor is to select events which invoke the robot to service a workcell. Let us define the set of acceptor states as a class of subsets of

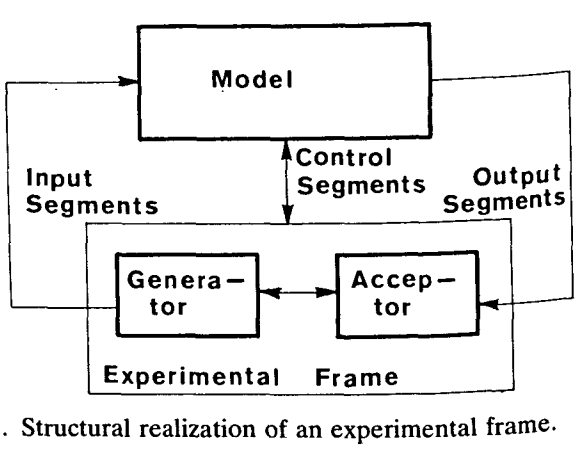


Fig. 3. Structural realization of an experimental frame.

indexes of workcells M_i, i.e.:

$$S_a \subset 2^{\{1,\ldots,N\}} \tag{23}$$

Then, the acceptor's transition function is given by:

$$\delta^a_{\text{ext}}: X_a \times S_a \to S_a.$$

The state $s_a = \varnothing \in S_a$ is the non-accepting state. All other states are accepting states. δ^a_{ext} is then defined as

$$\begin{aligned}\delta^a_{\text{ext}}(((s_1, \ldots, s_N), t), s_a) \\ = (s_a \cup \{i \mid s_i = B \text{ and } s_{i+1} = A\}) \\ - (\{j \mid s_j = C \text{ or } s_j = A\} \cap s_a)\end{aligned} \tag{24}$$

By definition of δ^a_{ext}, the state of the acceptor determines which workcells require service. A state of the acceptor can be an empty set. The state s_a contains indexes of only those workcells which have completed processing of a detail and from which the detail can be transported to another workcell (i.e. there exists a free workcell).

The states of the acceptor also determine state components of the generator that models the robot. We now proceed to define the robot's model in more detail. The model is defined by the following DEVS:

$$\text{Robot} = (S_R, \delta_{\varnothing R}, t^R_a, \{Z_{Rj}\}) \tag{25}$$

where S_R is the set of states defined as

$$S_R = S_a \times \text{POSITIONS} \times \text{HS} \tag{26}$$

where:

S_a – is a state set of the acceptor
POSITIONS – is the set of positions of the robot's effector-end in the base-Cartesian space E_0
POSITIONS = {"i-th position" $\in E_0 \mid i = 0, 1, \ldots, N, F$}
HS is the set of states of the effector
HS = {EMPTY, HOLDING}

We partition the state-set S_R into

$$\begin{aligned}S^1_R = \{(s_a, k, \text{EMPTY}) \mid s_a \in S_a \text{ and} \\ k \in \text{POSITIONS}\}\end{aligned}$$

and

$$\begin{aligned}S^2_R = \{(s_a, k, \text{HOLDING}) \mid s_a \in S_a \text{ and} \\ k \in \text{POSITIONS}\}\end{aligned} \tag{27}$$

The above division facilitates convenient specification of the remaining model components. The internal transition function is defined as:

$$\delta_{\varnothing R}: S_R \to S_R \tag{28}$$

Let $s^1_R = (s_a, k\text{-position, EMPTY}) \in S^1_R$. Then

$$\delta_{\varnothing R}(s^1_R) = \begin{cases} s^1_R & \text{if } s_a = \varnothing \\ s^2_R = (s'_a, i\text{-position, HOLDING}) & \text{if } s_a \neq \varnothing \end{cases}$$

where

$$i = \arg\min_j \{|k - j| \mid j \in s_a\}$$

and

$$s'_a = s_a \stackrel{i}{-} \{i\}.$$

Let $s^2_R = (s_a, k\text{-position, HOLDING}) \in S^2_R$. Then

$$\delta_{\varnothing R}(s^2_R) = s^1_R = (s_a, k+1\text{-position, EMPTY}).$$

Notice, that for the state s^1_R an internal transition can be represented by the following sequence of commands in *TORPL*:

MOVE EMPTY FROM "k-position" TO "i-position"
PICKUP "part" AT "i-position".

For $s^2_R \in S^2_R$, the sequence is:

MOVE HOLDING "part" FROM "k-position" TO "$k+1$-position"
PLACE "part" ON "$k+1$-position"
START ("start signal for $k+1$-device")

The time-advance function is specified as follows:

$$t^R_a(s^1_R) = \begin{cases} \infty & \text{if } s_a = \varnothing \\ \tau_m(k, i) + \tau_u(i) & \text{for } s^1_R \in S^1_R \\ & \text{if } s_a \neq \varnothing \end{cases} \tag{29}$$

where

$\tau_m(k, i)$ is the time of motion from position "k" to position "i",
$\tau_u(i)$ is the time of pick up operation from i-th device (usually const.),

and

$$\begin{aligned}t^R_a(s^2_R) = \tau_m(k, k+1) \\ + \tau_p(k+1) \quad \text{for } s^2_R \in S^2_R\end{aligned} \tag{30}$$

where

$\tau_m(k, k+1)$ is the time of motion from position "k" to position "$k+1$",
$\tau_p(k+1)$ is the time of place operation on $k+1$-th device.

The last component of Robot,

$$\{Z_{R_j}\} = \{Z_{R_j} \mid j = 0, \ldots, N, F\} \tag{31}$$

is the set of functions that generate external events for the workcell models M_i. More specifically functions

$$Z_{R_j}: S_R \to X_j, \quad \text{for } j = 0, 1, \ldots, N, F$$

are defined as follows:

Let $s_R = (s_a, k, \text{EMPTY}) \in S^1_R$

$$Z_{R_j}(s_R) = \begin{cases} e0 & \text{for } j \neq i \text{ or if } s_a = \varnothing \\ & \text{where } i = \arg\min_j \{|k - j| \mid j \in s_a\} \\ e2_i & \text{for } j = i \text{ and } s_a \neq \varnothing \end{cases} \tag{32}$$

Let $s_R = (s_a, k, \text{HOLDING}) \in S^2_R$. Then

$$Z_{R_j}(s_R) = \begin{cases} e0 & \text{for } j \neq k+1 \\ e1_{k+1} & \text{for } j = k+1 \end{cases} \tag{33}$$

Functions Z_{R_j} generate external events for the workcells and trigger their corresponding simulators. The structure of the model we have synthesized is shown in Figure 4.

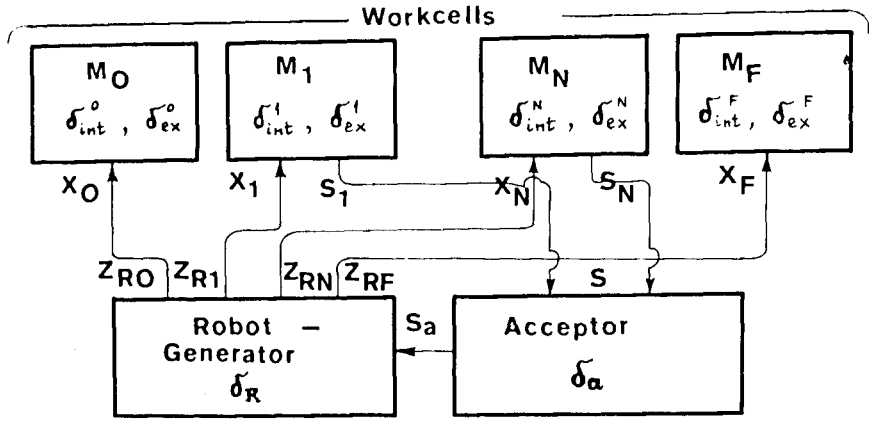


Fig. 4. Structure of the hierarchical simulation model of a machining line.

Our model machining facilitates a convenient generation of the robot's program. In order to construct the program, we translate the generator's transition function into commands of the task-oriented robot programming language. By definitions (28), (32), (33), we derive the following set of instructions:

MOVE EMPTY FROM "parking-position" TO "0-position"

LOOP: (FOR $i = 0, 1, \ldots, N, F$)
IF(("i-th input signal" EQ "not work-not free") AND ("$i+1$-th input signal" EQ "not work-free"))
THEN;
MOVE EMPTY TO "i-th position"
PICKUP "part" AT "i-th position"
MOVE HOLDING "part" TO "$i+1$-th position"
PLACE "part" ON "i-th position"
START (start signal for $i+1$-th device)
END FOR LOOP

Notice that this model of a workcell can be used as a basis for testing the program with a varying range of motion parameters. The most important parameters are the time it takes to complete an operation (τ_i) and the time the robot takes to service a workcell. The time τ_i depends on the type of a device on which the i-th operation is being processed. This time is fixed. It can be changed by replacing the device. Similarly, the times of PICKUP and PLACE operations are determined by the type of detail and device on which the detail is processed.

The times of robot's inter-operational moves (transfers) $\tau_m(i, j)$ depend on the geometry of the workscene and on the cost function of the robot motion. This cost function determines the dynamic of motion along the geometric trajectories.

The output and efficiency of the line depends on the arrangement of the devices on the workscene and the optimization criteria imposed on the robot's motion. For example, the most frequent move may be the longest move. However, we can only find this out after we have simulated the assembly line over a period of time longer than one cycle. Thus it is very important to have the facilities available to modify the following: a) the geometry of the workscene; b) the plan for each move of the robot; and c) the time parameters $\tau_m(i, j)$. Having modified the parameters, we run simulations repeatedly and observe the behaviour of the line. The second layer of the simulation system (Layer 2) is responsible for planning and modification of the motion parameters.

4. THE ROBOT MOTION PLANNER

The planner's fundamental function is to synthesize the robot's motion trajectories. The trajectories realize the MOVE instructions of the robot program. They also determine the duration of the moves. These data must be accessible in order to simulate the entire technological line. To generate the trajectories, Layer 2 must have the geometrical models of the robot and technological line available. It must also have initial and final effector's positions of each move given. The robot's motion trajectory planning process will be decomposed into two sub-problems: 1) planning of the geometric motion trajectory, and 2) planning of the motion dynamics along a computed trajectory. The motion planner consists of two layers:

- layer for collision-free geometrical path robot motion planning
- layer for optimal trajectory motion planning.

The first layer employs only the kinematic model of the manipulator. The second layer is based on the model of the robot's dynamics.

Such decomposition of the motion planner is appropriate for it facilitates an independent analysis of different variants of geometric as well as dynamic interpretations of the same motion trajectory.

4.1 Geometrical model of robot and its work-scene

Each workcell (device) M_i is represented by a specified in its own coordinates frame E_i. The location of M_i's in base Cartesian coordinate frame E_0 is represented by the transformation of the coordinate system E_i to the base system E_0. We accomplish this by employing Denavit-Hartenberg's matrix;[13] $H_{0i} = [\mathrm{Rot}_i, \mathrm{Pos}_i]$. The geometric model of M_i in base Cartesian space E_0 is obtained by transforming each vertex of the polyhedron 0_i using the matrix H_{0i}. The model of the work-scene has the following form:[7]

$$\mathrm{SCENE} = (\mathrm{OBJECTS}, \mathbb{H}) \qquad (34)$$

where

OBJECTS – is the set of geometric models of devices in E_i coordinate frame ($0_i \in$ OBJECTS),

and

$\mathbb{H}$ – is the set of transformations of E_i's to E_0 which describe the location of objects in base coordinate frame ($H_{0i} \in \mathbb{H}$).

The above model facilitates convenient modification of locations of objects on the workscene. The robot's geometry is represented by a skeleton model, i.e., broken line connecting the joints of the robot's manipulator.

A skeleton model for a robot with n-degrees of freedom is a vector[14]

$$c = (P_1, \ldots, P_{n+1}) \tag{35}$$

where $P_i \in E_0$ is a point in base frame E_0 which defines the location of the i-th joint of the manipulator. Point P_{n+1} defines the location of the end of effector. The skeleton model representation is often called the Cartesian state (configuration) of the manipulator.[14–16] The Cartesian state can be uniquely represented by a vector of joint angles

$$\bar{q} = (q_1, \ldots, q_n) \tag{36}$$

which describes the manipulator's state in the Joint Space.[17,18] The length of each link is given by l_i. The skeleton model of robot is shown in Figure 5.

The robot is located in E_0 in such a way that its base (usually point P_1 in the skeleton model) is in the origin of the base frame E_0.

The motion planner must also have initial and final positions of effector-end of each movement given. These positions are related to the "i-th position" denotes a location at which a part is placed on M_i workcell for $i = 0, 1, \ldots, N, F$.

Then, each MOVE instruction of robot program is parameterized by a pair (P_{init}, P_{final}), where P_{init} stands for the initial effector's position and P_{final} denotes its final position. The positions P_{init} and P_{final} are determined by the locations of the devices (workcells) in geometrical model of the workscene, and program of robot actions. These pairs must be available for all possible movements of robot's manipulator. It is a first phase of a robot program interpretation.

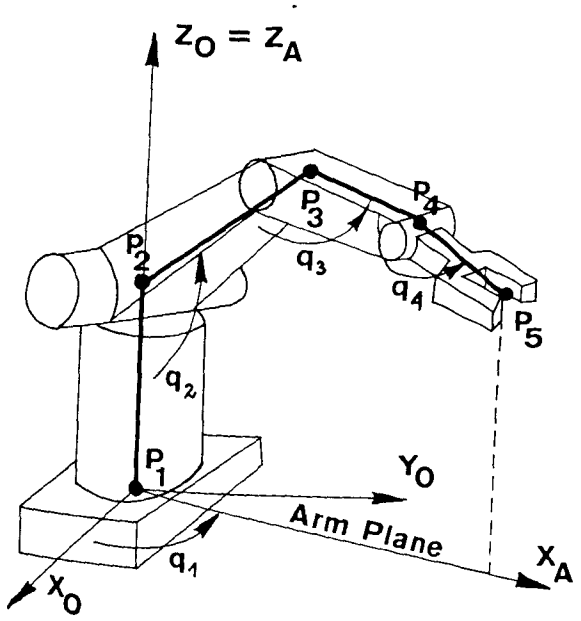


Fig. 5. Skeleton model of a robot manipulator.

The problem of planning a move for each pair (P_{init}, P_{final}) consists in finding collision-free and optimal motion trajectories.

The problem of planning a move between P_{init} and P_{final} is now decomposed into two subproblems: the geometry-oriented and time-oriented plan generation.[19] In geometry-oriented planning, we aim to find the shortest, collision-free track of robot motion. This problem is solved by the collision-free path planning system. Time-oriented planning requires that we find optimal speed and acceleration along the motion trajectories. The time-trajectory should minimize the motion cost function. This task is solved by the trajectory planning system. Task decomposition has the following consequences: the motion planner constructs an optimal trajectory along a given motion track, which, in general, is not a solution to the global optimization problem without geometric constraints.

The two-layer problem solving scheme proposed here was motivated by the following factors:

– lack of effective methods for solving global optimization problems, especially for robots in which generalized torques are functions of the manipulator's state,
– the proposed scheme facilitates parametrization of the equations of robot's dynamics by a scalar length of the motion track. This is a fast and effective way to solve the problem,
– the scheme provides an ability to compute optimal trajectories between given points for different geometric variants of the motion track and for alternative cost functions. This allows us to test the effectiveness of the manufacturing system with the aid of simulation.

We now proceed to describe both systems in more detail.

4.2 The collision-free path planning system

For the purpose of automatic translation of the language commands such as "MOVE TO" we need adequate models of robot kinematics and robot's workscene. There are two requirements which a robot kinematics model should satisfy. Firstly, in order to apply methods of geometrical path of robot motion planning, a robot kinematics model would facilitate direct analysis of robot location with respect to objects for its environment. Secondly, a robot kinematics model should be convenient to 3D graphic simulation of robot movements in the base Cartesian frame. The computer graphic simulation is a fundamental tool for the off-line testing of correctness of the robot program. For these reasons the most suitable model of robot kinematics is a discrete dynamic system,[16] defined as:

$$M_{\text{robot}} = (C, U, Y, f, g) \tag{37}$$

where C denote a set of states (configurations) of manipulator, U denote a set of input signals causing a change of configuration, and Y is the set of output signals of M_{robot}.

An output of M_{robot} should ensure a possibility of the geometrical representation of the robot body in 3D-base frame. For this purpose it is convenient to use a skeleton model of the manipulator as the output of M_{robot}.

The function $f: C \times U \rightarrow C$ is a one-step transition function of the form

$$c(k+1) = f(c(k), u(k)), \tag{38}$$

and the function $g: C \rightarrow Y$ is an output function of the form

$$g(c(k)) = y(k) \tag{39}$$

The properties of so defined model of the robot kinematics depend on the method of specification of its components and especially the state-set C and input-set U.

Assume that a state c of robot's manipulator is described by a vector of points in the Cartesian base frame, which represent positions of every joint of the manipulator, i.e.

$$c = (P_i \mid i = 1, \ldots, n+1) \quad \text{and} \quad P_i \in E_0$$

Such an assumption yields that the model (37) can be reduced to the system

$$M_{\text{robot}} = (C, U, f).$$

In this case, the output set Y is equal to C and the output function g of M_{robot} is the identity function.

In order to avoid the problem of solving the kinematics equations one can perform an arbitrary discretization of the Joint Space[15,18] or of the base Cartesian Space.[14] In this paper, we restrict the analysis to the assumption about Joint Space discretization, i.e. we assume that every joint angle q_i has already been discretized at the increment δq_i, and that a single change of angle q_i amounts only to $\pm\delta q_i$ or 0. Such an assumption yields that changes of q_i can be described by means of the input-set $U_i = \{+1, 0, -1\}$, where ± 1 mean a change by $\pm\delta q_i$ and 0 means no change. The model of robot kinematics will be synthetized in three steps.[15] The first step will be concerned with synthetizing the model of a single joint. A second step, we shall produce the model of the manipulator arm in the arm plane, while at the third step we shall describe the kinematics model of the whole manipulator in 3-D base frame E_0.

For the i-th joint the set of states equals the set of feasible angle positions of the joint, defined as

$$J_i = \{0, 1, \ldots, N_i\} \quad \text{where} \quad q_i^{\max} = q_i^{\min} + N_i \cdot \delta q_i \tag{41}$$

A change of the joint state causes the link l_i to rotate by an angle δq_i in the direction defined by the input $u_i \in U_i$; that in the composed kinematic chain amounts to a rotation of a point P_{i+1} around a point P_i within the arm plane. For this reason, it is natural to take the output of a model of the i-th joint as a transformation defining the rotation of a point P_{i+1} in the arm plane. Let p_{i+1} and p_i denote the positions of the points P_{i+1} and P_i within the arm plane, respectively,[15] then the new position of point p_{i+1} after a rotation around point p_i by a constant angle δq_i toward the u_i direction is defined as

$$p'_{i+1} = p_i + R_i(u_i)[p_{i+1} - p_i] \tag{42}$$

and

$$R_i(u_i) = \begin{bmatrix} \cos \delta q_i & \vdots & u_i \sin \delta q_i \\ \cdots\cdots & \vdots & \cdots\cdots \\ -u_i \sin \delta q_i & \vdots & \cos \delta q_i \end{bmatrix}$$

Thus the output set $Y_i = \{R(u_i) \mid u_i \in U_i\}$.

The model of the i-th joint kinematics is the representable as a Finite State Machine (FSM) in the form:[15]

$$M_{i\text{-joint}} = (U_i, J_i, Y_i, \lambda_i, \delta_i) \tag{43}$$

where $\lambda_i: J_i \times U_i \rightarrow J_i$ is the state transition function and

$$\lambda_i(j, u) = \begin{cases} \max(0, j+u) & \text{for } u = -1 \\ \min(N_i, j+u) & \text{for } u = +1 \\ j & \text{for } u = 0 \end{cases} \tag{44}$$

$\delta_i: J_i \times U_i \rightarrow Y_i$ is the output function and

$$\delta_i(j, u) = \begin{cases} R_i(0) & \text{for } (j = 0 \wedge u = -1) \\ & \text{or } (j = N_i \wedge u = +1) \\ R_i(u) & \text{in other case} \end{cases}$$

The position of individual joints of the manipulator within the arm plane is described by the vector $c_{\text{arm}} = (p_1, \ldots, p_{n+1})$ related univocally to the Cartesian state c.[14] Then the model of changes of the arm kinematic within the arm plane can be represented as a reduced FSM of the form

$$M_{\text{arm}} = (U_{\text{arm}}, C_{\text{arm}}, \lambda_a) \tag{45}$$

where $U_{\text{arm}} = X\{U_i \mid i = 2, \ldots, n\}$ and

$$\lambda_a: C_{\text{arm}} \times U_{\text{arm}} \rightarrow C_{\text{arm}}$$

is the one-step transition function defined recurrently as follows:

Let

$$c'_{\text{arm}} = \lambda_a(c_{\text{arm}}, \tilde{u}) \quad \text{where} \quad c_{\text{arm}} = (p'_1, \ldots, p'_{n+1}),$$
$$c_{\text{arm}} = (p_1, \ldots, p_{n+1}) \quad \text{and} \quad \tilde{u} = (u_2, \ldots, u_n),$$

then

$$p'_i = p'_{i-1} + \left(\prod_{k=2}^{i-1} \delta_{i-k}(j_{i-k}, u_{i-k})\right) \times [p_i - p_{i-1}] \quad \text{for } i = 2, \ldots, n+1 \text{ and } p'_1 = p_1. \tag{46}$$

δ_k is the output function of the FMS model for k-th joint kinematics $M_{k\text{-joint}}$ (44).

The discretization of the Joint Space results in the finiteness of the Cartesian-state set C. The cardinality of C is equal to $(N_1 \cdot \bar{C}_{\text{arm}})$. Thus the model of robot kinematics with respect to the base coordinate frame E_0 can be represented as a Finite State Machine of the following form[15,20]

$$M_{\text{robot}} = (U, C, f) \tag{47}$$

where

$$U = X\{U_i \mid i = 1, \ldots, n\},$$

$f: C \times U \rightarrow C$ is the one-step transition function.

Let $c = (P_1, \ldots, P_{n+1})$, $c' = (P'_1, \ldots, P'_{n+1})$ and $c' = f(c, \bar{u})$, then

$$P'_i = \left[\begin{array}{c|c} \delta_1(j_1, u_1) & \bar{0} \\ \hline 0 & 1 \end{array}\right] \cdot crd^i z^{-1}(\lambda_a(z(c), \bar{u}_a)) \qquad (48)$$

where $z: C \rightarrow C_{arm}$ is the transformation of Cartesian-state into arm plane,[14,15] and $crd^i c = P_i$.

The transition function f of the robot – FSM is a composition of the transition function λ_a of the arm – FSM and the output function of the first joint FSM, responsible for the rotation of arm plane. More exactly, such a model is described in ref. 15. The model based on Cartesian space discretization may be found in refs 14, 16, & 20.

The model of the manipulator kinematics will be applied to planning of collision-free paths of robot motions. The problem of finding a collision-free path of a manipulator movement, from an initial configuration c_{init} (equal to c'_{final} of previous performed movement) to a terminal effector location $P_{final} \in E_0$ can be formulated as follows.[15] Let $C_F = \{c \mid crd^{n+1} c = P_{final}\}$ be a set of terminal configurations. Then the problem amounts to finding such a sequence $u_F^* = (\bar{u}_0, \ldots, \bar{u}_F)$ of inputs of the FSM that:

(i) the terminal configuration reaches point P_{final}

$$f^*(c_{init}, u_F^*) \in C_F \qquad (49)$$

where $f^*(c_{init}, u_F^*) = f(f^*(c_{init}, u_{F-1}^*), \bar{u}_F)$

(ii) every configuration corresponding to sequence u_F^* is feasible

$$f^*(c_{init}, u_j^*) \in C_{feasible} \quad \text{for} \quad j = 0, 1, \ldots, F \qquad (50)$$

(iii) the length of the geometrical path of motion is minimal.

As the result, we obtain the sequence of robot's configurations $c^* = (c_0, c_1, \ldots, c_F)$, where $c_0 = c_{init}$, and $c_i = f^*(c_{init}, u_i^*)$, and also the sequence of configurations in a Joint Space

$$\text{path}^* = (\bar{q}_0, \ldots, \bar{q}_F) \qquad (51)$$

where $\bar{q}_i = \text{Conv}(c_i)$ and Conv: $C \rightarrow Q$ is the transformation of the Cartesian-State Space into a Joint Space. The function Conv we can construct by using the vector dot and cross products of the vectors $[P_{i+1} - P_i]$ and $[P_{i-1} - P_i]$.[14,16]

In order to solve the problem, we shall employ procedures of graph searching used in *AI*. For this purpose we shall exploit the state-transition graph of an automaton M_{robot} generated implicitly by applying the function f. Except for the input $\bar{0}$ which does not change the state, every node (state) of the graph has maximal $2^n - 1$ successors, i.e.

$$\text{Suc}(c) = \{f(c, \bar{u}) \in C_{feasible} \mid \bar{u} \in U - \{\bar{0}\}\} \qquad (52)$$

A configuration c is said to be feasible if it does not collide with any obstacle in the workscene. Using the discretization of the q_1 joint angle, the checking for the collision-freeness of the configuration c can be reduced to the "brocken line-polygon" intersection detection problem in the plane. This problem can be solved by a few efficient algorithms.

The development of the search graph will start from the node c_{init}, by the action of function f for all possible inputs $\bar{u}$. The way of expanding the graph will depend on the form of the evaluation function.

The evaluation function $e(c)$ at any node c gives the sum of the cost of path from c_{init} to c plus the cost of the path from c to a terminal node i.e.

$$e(c) = k(c) + h(c) \qquad (53)$$

The cost of path $k(c)$ between two nodes is the sum of the cost of all the arcs connecting the nodes along the path. As the cost function between two nodes c and $c' = f(c, \bar{u})$, joined by an arc, we shall assume an Euclidean distance travelled along by the effector while passing from configuration c to c'.[15] The heuristic function $h(c)$ with estimates of the distance from the actual configuration c to the set C_F will be defined as the rectilinear distance between the current effector position and the terminal position P_{final}.

Now, we can use the A^* algorithm[15] to find the shortest path in the state-transition graph of system M_{robot} connecting the initial node c_{init} and the final node $c_F \in C_F$. The result the A^*-algorithm work is the sequence c^* of collision-free configurations in the Cartesian Space, or the sequence path* (51) of robot configurations in the Joint Space. The collision-free motion trajectory of the manipulator is minimal only with respect to the geometric distance for the end-effector. This does not imply that this is a time-minimal trajectory. However, by changing the evaluation function, we can obtain another motion path, e.g. trajectories tracking a given path of the effector motion.

4.3 The time-trajectory planning system

From the path planner we obtain an ordered sequence of robot's configurations c^* (or path*), which represents the collision-free track of robot motion. Usually, a continuous geometric track of robot motion is constructed by connecting configuration from path* (points in the Joint Space) with some means such as cubic splines.[21] In this case, the geometric track is given in the form of the parameterized curve

$$\text{path}^* = \bar{q} = q(s), \quad s \in [0, s_{max}] \qquad (54)$$

where the initial and final configurations of the path correspond to the points $s = 0$ and $s = s_{max}$, respectively. The time-trajectory planning system, briefly called trajectory planner, receives these geometric paths as input and determines a time history of position, velocity, acceleration and input torques which are then fed to the trajectory tracker. While the problem of avoiding obstacles in the robot's workspace is not a control theory

problem in the normal sense, the problem of moving a mechanical system at minimum cost is. On this level of robot specification, a robot is represented by a model of manipulator dynamics. There are a number of ways of obtaining the dynamic equations of a robot arm, i.e. the equations which relate joint forces and torques to positions, velocities and accelerations. One of them, the Lagrange method, yields a set of differential equations in the form:[13,22]

$$M(\bar{q})\ddot{q} + N(\dot{q})\dot{q} + R\dot{q} + G(\bar{q}) = U \qquad (55)$$

where U is the vector of generalized forces, M is the inertia matrix, N the Coriolis force array, G the gravitational force and R is the viscous friction matrix.

We assume that the set of realizable torques can be given in terms of the state $\bar{q}$. Then we have

$$U = (u_1, \ldots, u_n) \in E(\bar{q}, \dot{q})$$

where u_i is the i-th actuator torque/force. Given the position $\bar{q}$ and velocity $\dot{q}$, E determinates a set of input space. If the equations of the parameterized path $q(s)$ are plugged into the dynamic equations, then they become[22,23] form:

$$\begin{aligned} m\dot{\mu} + n\mu^2 + r\mu + g &= u \\ \dot{s} &= \mu \end{aligned} \qquad (56)$$

Here μ is the time-derivative of the parameter s. The cost C will be assumed to take the form

$$C = \int_0^{s_{max}} L(s, \mu, u)\, ds \qquad (57)$$

The trajectory planning problem then becomes that of minimizing cost subject to dynamic model and the constraints of u. This problem is a classical dynamical optimization problem for non-linear system and can be solved, for example, by using the dynamic programming method or graph searching method.[22-24] As we can show by using the parameterized path, the dimensionality of the problem has been reduced. There exist only two states s and μ, regardless of how many joints the robot has.

To apply dynamic programming one must divide the "phase plane" $s - \mu$ into a discrete grid $(s_k, \mu_k, (s_k))$. Next, under the assumption, that $\dot{\mu}_k = \text{const}$ in the interval $[s_k, s_{k+1}]$, one can compute u and μ as functions of s from interval $[s_k, s_{k+1}]$.[22] Given the formulas (56) for the pseudo-velocity μ and the joint torques u, the incremental cost of going from one point on the grid to the next can be found as

$$C(\mu_k(s_k), \mu_{k+1}(s_{k+1})) = \int_{s_k}^{s_{k+1}} L(s, \mu(s), u(s))\, ds. \qquad (58)$$

Once costs have been computed, the usual dynamic programming algorithm can be applied with respect to the following recursive form

$$I_k(\mu_k) = \min_{\mu_{k+1}} [C(\mu_k(s_k), \mu_{k+1}(s_{k+1})) + I_{k+1}(\mu_{k+1})] \qquad (59)$$

where $I_k(\mu_k)$ is the optimal cost from (s_k, μ_k) to (s_{max}, μ_{final}). Given the optimal sequence (s_k, μ_k) from the grid, it is then possible to calculate joint positions, velocities and torques. Hence we can obtain an optimal trajectory and the time of manipulator movement along a geometrical path. The time-trajectories of the robot movement allows us to establish times for each action of the robot. The simplifications of the dynamic model and discretization of the s parameter result in some errors in the solution of the optimal trajectory problem. More detail describing the errors can be found in refs. 22 & 23. The times of motion computed by the planner are good estimates of the actual times despite the fact that they are subject to errors resulting from the simplification of the dynamic model and discretization of the motion trajectory. The two-tier decomposition of the motion planner is particularly suitable for evaluating the operation of a manufacturing system where several variant interpretations of the robot's motion are applied. Without changing the simulation model we can vary the parameters of the experimental frame. More specifically, for each program command we can change the geometry of the motion (change in the evaluation function), or change the motion dynamics along the motion trajectories (selection of criteria for time-minimal or energy-minimal planning). Thus we can select the most effective variant for realizing the robot's motion steps. Conventional manufacturing systems simulation techniques do not provide such facilities,[1-4] since they do not embed the experimental frame/planner interface in their simulation layer.

5. AN EXAMPLE

Let us consider the technological process PROC consisting of the following operations: Op_1 = mill a detail and Op_2 = turn a detail. To every operation Op_i the following devices are assigned: $Op_1 \rightarrow$ miller A as M_1 and robot R, $Op_2 \rightarrow$ lathe B as M_2 and robot R, feeder conveyor I as M_0 and output conveyor 0 as M_F.

The structure of a coupled DEVS simulation model of this technological line (workcell) is shown in Figure 6, where mXY denote a robot's motion between the X-device and the Y-device.

The geometrical model of the technological workcell is shown in Figure 7. The figure presents the objects and

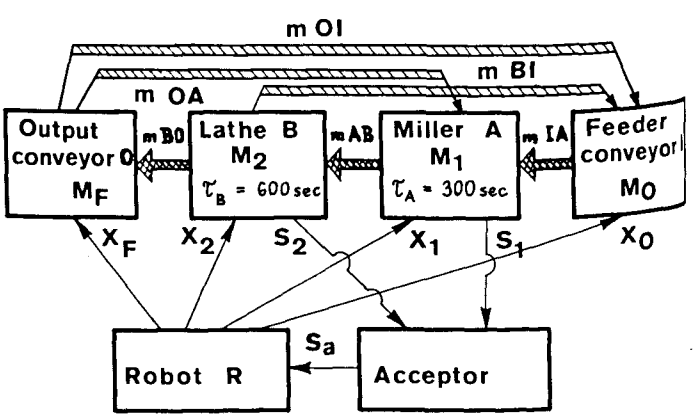


Fig. 6. Structure of a DEVS-model of the workcell.

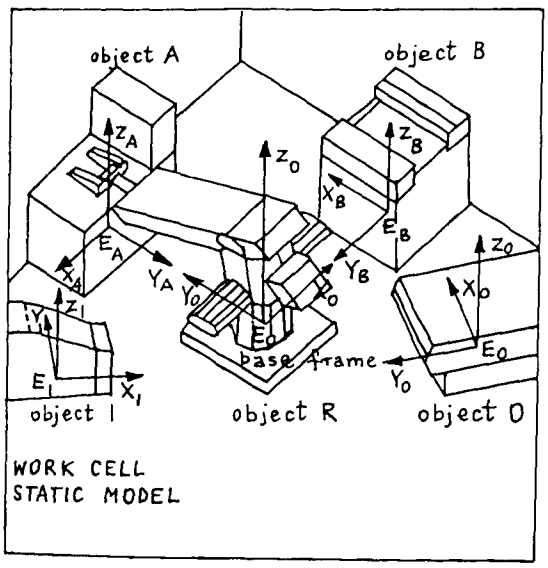


Fig. 7. Geometrical model of the workcell.

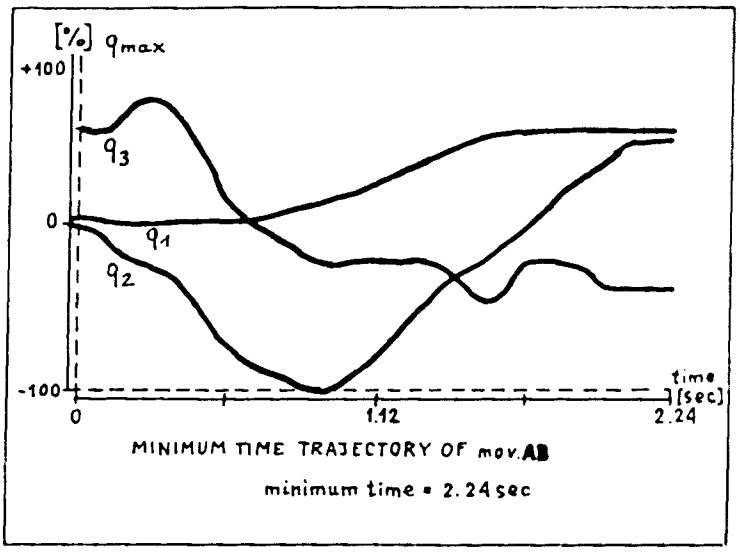


Fig. 9. Minimum-time trajectory of robot motion.

their locations and orientations in the base Cartesian frame E_0. The points *I*, *A*, *B*, 0 determine the initial and final positions of the effector-end for each robot's motion. Figure 8 depicts one of the planned path of a robot motion. This path realizes a traslocation of part from the miller A to the lathe B. This collision-free path is obtained by an application of the graph searching method A^*.

The minimum-time trajectory of movement between A and B along the designed path is shown in Figure 9. The result of event simulation of the work-cell action is presented in Figure 10. The productivity of the work-cell is equal to one detail each 634 sec. Figure 10a shows the time of work of objects A – miller, B – lathe and R – robot. Figure 10b illustrates the time of detail stays in the workcell and its division in the translocation time, treating time and waiting time. The global time of every robot movement, during 8 hours work is presented in Figure 10c.

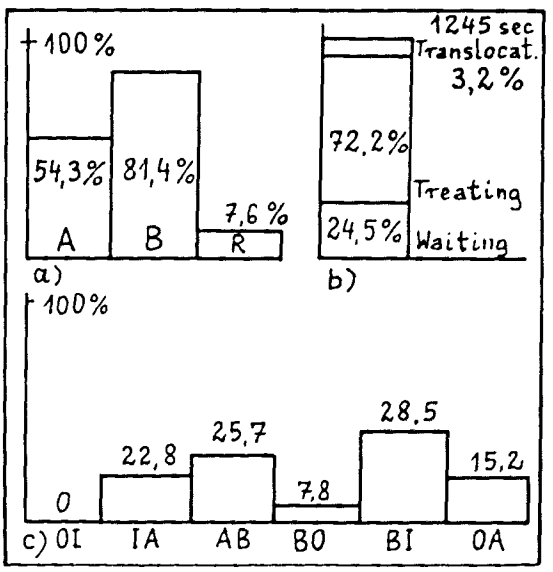
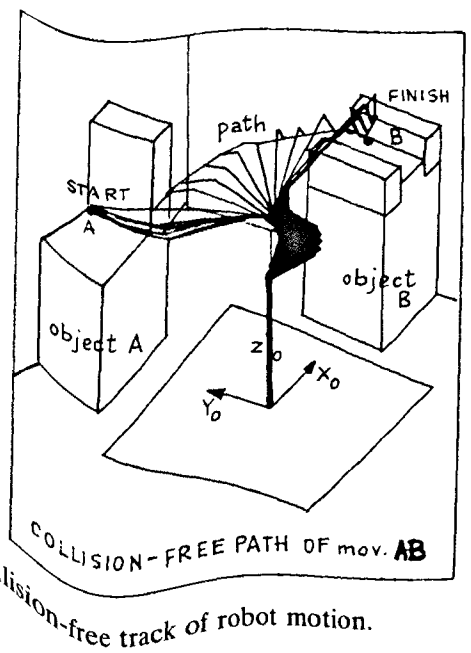


Fig. 10. The result of a workcell action simulation.



8. Collision-free track of robot motion.

6. FINAL REMARKS

We have presented a framework for automatic synthesis of a discrete event-based simulator of a robotized flexible manufacturing system. The simulator is a two-layer system. The first layer comprises a discrete event model of the manufacturing system and a model of a robot servicing this system. The robot's model is parametrized by variables obtained from the simulator's second layer. This layer plans the robot's motion steps. The steps realize particular robot actions required to service the manufacturing system. The second layer consists of two subsystems: a) A subsystem responsible for planning the geometry of motion; and b) subsystem responsible for planning the motion dynamics. Such an architecture facilitates convenient simulation of several realization variants of the robot's moves. This, in turn, allows us to evaluate the operation of the manufacturing system under different interpretations of the robot's program.

References

1. A.C. Sanderson, H. Zhang and L.S. Homem de Mello, "Assembly Sequence Planning" *AI Magazine* **11** (1), 62–82 (1990).

2. A. Kusiak and G. Finke, "Selection of Process Plans in Automated Manufacturing Systems", *IEEE Trans. on Robotics and Autom.* **4** (4), 397–408 (1988).
3. T. Nagata and K. Honda, "Multirobot Plan Generation in a Continuous Domain: Planning by Use of Plan Graph and Avoiding Collisions among Robots", *IEEE Trans. on Robotics and Autom.*, **4** (1), 1–13 (1988).
4. S.A. Hutchinson and A.C. Kak, "SPAR: A Planner That Satisfies Operational and Geometric Goals in Uncertain Environments" *AI Magazine*, **11** (1), 30–61 (1990).
5. B. Faverjon, "Object level programming of industrial robot" *IEEE Int. Conf. on Robotics and Automation* **2**, 1406–1411 (1986).
6. R. Speed, "Off-line Programming for Industrial Robots" *Proc. of ISIR 87* 2110–2123 (1987).
7. W. Jacak and I. Sierocki, "Software Structure for Design of Automated Work-Cell" *Cybernetics and Systems '90* (Ed. R. Trappl) (Kluver Ac. Publ., Vienna, 1990) pp. 216–225.
8. B.P. Zeigler, "*Multifacetted Modelling and Discrete Event Simulation*", (Academic Press, London, 1984).
9. J.W. Rozenblit and B.P. Zeigler, "Design and Modelling Concepts" *Encyclopedia of Robotics* (John Wiley, N.Y., 1988).
10. J.W. Rozenblit and B.P. Zeigler, "Entity-Based Structures for Model and Experimental Frame Construction" In: *Modelling and Simulation in Artificial Intelligence Era* (Ed. M.S. Elzas et al.) (North Holland, Amsterdam, 1986).
11. T. Lozano-Perez, "Task-level Planning of Pick-and-Place Robot Motions" *IEEE Trans. on Computer* **38** (3), 21–29 (1989).
12. R. Brooks, "Planning Collision-free Motions for Pick-and-Place Operations" *Int. J. Robotics Res.* **2** (4), 19–44 (1983).
13. M. Brady, *Robot Motion: Planning and Control* (MIT Press, Cambridge MA, 1986).
14. W. Jacak, "A Discrete Kinematic Model of Robots in the Cartesian Space" *IEEE Trans. on Robotics and Automation* **5** (4), 435–444 (1989).
15. W. Jacak "Strategies of Searching for Collison-free Manipulator Motions: Automata Theory Approach" *Robotica* **7**, 129–138 (1989).
16. W. Jacak, "Modelling and Simulation of Robot Motions" *Lecture Notes in Computer Science* **410** (Springer Verlag, Berlin, 1990) pp. 751–758.
17. T. Lozano-Perez, "Spatial-Planning: A Configuration Space Approach" *IEEE Trans. on Computer* **32** (2), 108–119 (1983).
18. T. Lozano-Perez, "A simple Motion Planning Algorithm for General Robot Manipulators" *IEEE Trans. on Robotics and Autom.* **3** (2), 224–238 (1987).
19. W. Jacak, "Robot Task and Movement Planning" In: *AI, Simulation and Planning in High Autonomy Systems* (Ed. P.B. Zeigler, J.W. Rozenblit) (IEEE Comp. Soc. Press, Los Alamitos, CA 1990) pp. 168–176.
20. W. Jacak, "Discrete Kinematic Modelling Techniques in Cartesian Space for Robotic System" In: *Advances in Control and Dynamics Systems* (Ed. C.T. Leondes) (Academic Press, Orlando, Florida) (in print).
21. J.Y. Luh, C. Lin and P. Chang, "Formulation and Optimization of Cubic Polynomial Joint Trajectories for Industrial Robot" *IEEE Trans. on Automatic Control* **28** (12), 1066–1074 (1983).
22. K. Shin and N. McKay, "A Dynamic Programming Approach to Trajectory Planning of Robotic Manipulators" *IEEE Trans. on Automatic Control* **31** (6), 491–500 (1986).
23. K. Shin and N. McKay, "Minimum-Time Control of Robotic Manipulators with Geometric Path Constraints" *IEEE Trans. on Automatic Control* **30** (6), 531–541 (1985).
24. H. Tan and R. Potts, "Minimum Time Trajectory Planner for the Discrete Dynamic Robot Model with Dynamic Constraints" *IEEE Trans. on Robotics and Automation* **4** (2), 174–185 (1988).