WORKCELL TASK PLANNING AND CONTROL

WITOLD JACAK
Institute of System Sciences, Johannes Kepler University Linz
A-4040 Linz, Austria

and

JERZY W. ROZENBLIT
Department of Electrical and Computer Engineering, University of Arizona
Tucson, Arizona 85721, U.S.A

ABSTRACT

This paper describes an intelligent control system of a manufacturing cell, which can plan tasks and motions of robots that service the cell. The system consists of two basic layers: the *Task Planning* and the *Task-Level Programming* layer. Task planning is based on the description of technological operations and their precedence relation. The resulting fundamental plan describes the decomposition of a manufacturing task into an ordered sequence of robot actions. The implementation of the plan is carried out using a *task-level programming* approach in which the detailed paths and trajectories, gross and fine motion, grasping, and sensing instructions are specified.

1. Introduction

Intelligent autonomous systems should be able to synthesize and execute their actions so that the overall system's objectives are be achieved, even under circumstances which may require replanning [5,6].

An autonomous, flexible manufacturing system (FMS) is a set of programmable machines (technological devices), D, called *workstations* and product *stores*, M, connected by a flexible material handling facility, R, (e.g., a robot or an automated guided vehicle), and controlled by a computer net connected with a sensory system [3,4]. An FMS can perform technological operations such as fabrication, machining, or assembly. All machines and material handling systems (robots) are highly automated. The groups of workstations serviced by robots are called *cells*. In a manufacturing environment, FMSs are generally constructed based on a hierarchical control architecture. The control hierarchy consists of the following levels: facility, cell, workstation, and equipment. In this paper, we propose an intelligent control system of a cell, which can plan tasks and motion of robots servicing the cell. We now proceed to define basic notions.

The input data of a cell controller — these data are obtained from the facility level — are a *technological task*. A task realized by a cell is given by a three-tuple

$$Task = (O, \prec, \alpha) \tag{1}$$

where: O - is the finite set of technological operations necessary to process (machine or assemble) a product, $\prec \subset O \times O$ - is the weak-ordering relation of operation precedence, and $\alpha \subset O \times D$ - is the relation of device assignment (i.e. $(o, d) \in \alpha$ means that the operation o can be performed on the workstation d).

Parts are transferred between machines by the robots which service the cell. A robot ($r \in R$) can service only those machines which are located within its service space. The automatic synthesizer of the cell control program must determine for each robot r a set of cell-state dependent time trajectories of the robot's motions and actions such that all operations of a given task are carried out. The control program synthesis problem has a large number of possible solutions. They differ with respect to the technological operations, sensor-dependent robots actions, geometric forms of the manipulator's paths, and dynamic of movements along the paths. To reduce the complexity of the solution process, we apply a hierarchical decomposition which results in two subproblems: the *Task Planning Problem* and the *Task-Level Programming Problem*.

2. Task Planning

The *Task Planner* performs the first phase of the cell's program synthesis, namely, the generation of the fundamental plan of robot and workstation actions. These actions must be derived in order to execute a manufacturing task.

The sequence of actions depends directly on the order of the technological operations defined by *Task* (Eq.1). Hence, we can reduce the *Manufacturing Task Planning* problem to the problem of finding an ordered sequence of operations which is feasible and which minimizes the number of potential deadlocks.

To establish the sequence of technological operations and then the sequence of robot actions, we decompose the task planning problem into two subproblems: a) the *production route planning* problem and b) the *synthesis of the fundamental plan of cell actions*.

2.1. Production Route Planner

The route planning consists in finding an ordered sequence of operations from *Task* with a minimum number of deadlock cases. This sequence is called *a pipeline sequential machining process*

$$Process = (o_1, o_2, \ldots, o_L) \tag{2}$$

if the following conditions hold:

(A) if for two operations o_i and o_j from *Task*, $o_i \prec o_j$, then $i < j$

(B) for each $i = 1, \ldots, L-1$ there exists a robot which can transfer a part from machine d_i or store m_i (assigned to operation o_i) to machine d_{i+1} assigned to operation o_{i+1}

In addition, we define a set of ordered sequences of technological devices or stores (called *resources*) needed for the execution of the successive operations from the list *Process*. This set of new sequences, denoted **P**, is called *production routes*. A production route $\mathbf{p} \in \mathbf{P}$ is an ordered list of resources which has $2L-1$ stages, where L denotes the length of the list *Process*. The route is created on-line during the execution of technological operations. In general, a route is defined as follows:

$$\mathbf{p} = (p(i) | i = 0, 1, \ldots, 2L) \tag{3}$$

where

(a) $p(0) = m_f$ and $p(2L) = m_s$ (m_f, m_s denote feeder and output conveyors, respectively)

(b) for $i = 2j-1$ and $j = 1, \ldots, L$, $p(i) = \alpha(o_j) = d_j$

(c) for $i = 2j$ and $j = 1, \ldots, L$, $p(i) = 0$ if direct transfer to workstation $\alpha(o_{j-1}) = d_{j+1}$ is possible or $p(i) = m_j$ (store), otherwise

The "minimal" production route has at least $L = 2$ nonzero stages. We denote it by $\mathbf{p}_{min} = (m_f, \alpha(o_1), \alpha(o_2), \ldots, \alpha(o_L), m_s)$. Each execution of a process is called a *job*. A job J is characterized by a production route $\mathbf{p}$. In the workcell discussed in this paper, a so called *circular wait-deadlock* between pipeline processes can occur. A *circular wait* occurs if there's a closed chain of jobs in which each job is waiting for a machine held by the next job in the chain [1]. To avoid it, the minimal production route $\mathbf{p}_{min}$ is partitioned into a unique set of Z sublists called *zones*.

$$\mathbf{p}_{min} = (z_k | k = 1, \ldots, Z) \tag{4}$$

where $z_k = (s_k, u_k)$, $u_k = (u_k^i | i = 1, \ldots, I(k))$ is the sublist of resources which appear only once in the minimal production route $\mathbf{p}_{min}$ (called *unshared resources*), $s_k = (s_k^j | j = 1, \ldots, N(k))$ is a sublist of resources which are used more than once in the route (they are called *shared resources*), $I(k)$, $N(k)$ denote the number of unshared and shared resources in subzones u_k and s_k, respectively). To avoid deadlocks, we employ the *restricted allocation policy* proposed in [2]. Consistent with the production route definition (Eq.4), we define the function $Next_J$ which specifies the next resource required by the job J. Let $C(d)$ denote the capacity of the workstation d, i.e., the maximum number of jobs which can be allocated to it. Additionally, by $h(d)$ we denote the number of jobs which are *currently* allocated to the workstation d. Assume that the resource $Next_J(t) = v$ belongs to zone z_k, i.e.,

$$v \subset z_k = (s_k^1, \ldots, s_k^{N(k)}, u_k^1, \ldots, u_k^{I(k)}).$$

Then, the allocation policy is defined by the following rules:

(A) If $(v = u_k^j \ \& \ j \neq 1$ or $v = s_k^j \ \& \ h(v) < C(s_k^j) - 1)$ or $(v = u_k^1 \ \& \ h(u_k) < C(u_k) - 1)$ where $C(u_k) = \sum_{j=1}^{n(k)} C(u_k^j)$, then resource v can be allocated to job J.

(B) If $v = s_k^j \ \& \ h(v) = C(s_k^j) - 1$, then resource v can be allocated to job J if $(\forall s_k^i \neq s_k^j)(h(s_k^i) < C(s_k^i)) \ \& \ h(u_k) < C(u_k)$

(C) If $v = u_k^1 \ \& \ h(u_k) = C(u_k) - 1$, then resource v can be allocated to job J if $(\forall s_k^i)(h(s_k^i) < C(s_k^i))$.

This allocation policy results in the following facts: [2]

Fact .1 *The circular-wait deadlock can never occur under the restricted allocation policy described in rules (A), (B), (C).*

We use this fact to formulate the planning quality criterion. The job waiting times should be minimal in order for an optimal makespan to be generated. Based on Fact.1, we can derive the following: Let zone z_k have $n(k) = N(k)+1$ elements, i.e., $N(k)$ is the number of shared workstations and 1 denotes all the unshared workstations. Assume that the probability of a workstation being full is equal to w (i.e., $P(h(d) = C(d)) = w$).

Fact .2 *If for a job J, $Next_J(l) = v \in z_k$ then the probability that the job J will wait for processing is:*

$$P(J(l^+) = \text{wait}) = w(1 - (1 - w)^{n(k)-1})$$

The more elements (workstations) belong to a zone (the subzone u_k is treated as one element), the higher the probability that a job will wait for resource allocation. Hence, we can introduce a measure of production route quality. Let $\mathbf{p}_{min} = (z_k | k = 1, \ldots, Z)$ be the minimal production route of *Process* and $z_k = (s_k u_k)$. The measure of route quality is defined as:

$$\nu(\mathbf{p}_{min}) = \max_k \{n(k) | k = 1, \ldots, Z\} \tag{5}$$

The function $\nu(\mathbf{p}_{min})$ is used to evaluate the technological process being planned. Our task is to find an ordered sequence of operations from *Task* which is feasible and which minimizes the function $\nu(\mathbf{p}_{min})$. This problem can have more than one solution. To solve it, we use the backtracking algorithm [12]. The output of the algorithm contains all feasible, ordered sequences of technological operations which realize the *Task* and have the same quality with respect to the function ν. Each *Process* from the output list determines a different fundamental plan of robot and workstation actions and, thus, different principles of workcell control.

3.2. *Synthesis of Fundamental Plan of Cell Actions*

Based on the sequence of operations *Process* obtained through task planning, we create a fundamental plan of actions for the cell's components, i.e.,

$$Plan = (Action_i | i = 1, \ldots, L)$$

Each $Action_i$ consists of three parts which determine a) the preconditions of an action, b) the robot's motion parameters necessary for the realization of the i-th operation of *Process*, and c) the action execution parameters. The i-th action of a workcell has the following form:

$$Action_i = (Cond(o_i), Transfer(o_{i-1}, o_i), Execute(o_i))$$

The $Cond(o_i)$ segment describes the preconditions which must be satisfied in order for the operation o_i to be executed. It also establishes the geometric parameters the robot's motion. The preconditions are formulated as a function of the workcell states based on the conditions of Fact.1

The elementary actions of a robot, $Transfer(o_{i-1}, o_i)$, are interpreted as the "PICK AND PLACE part FROM x TO y" macro instruction, where x denotes the geometric data of the output buffer of the workstation and y is the position and orientation of the input buffer of the next workstation. The parameters x and y are determined by the segment $Cond(o_i)$ [8]. The action EXECUTE can be translated into the instructions of a program of a NC-workstation in a similar manner.

This completes the definition of the fundamental plan of cell actions specified with respect to the chain of technological operations *Process*. Given the set of *Actions* we can define the control process for both the robots and the workstations. We can also specify the time trajectories of robot motions for each operation. Each chain of technological operations from *Task* generates a different fundamental plan of cell actions and different cell control algorithms. The flow time of every part is the sum of processing time, waiting time, and the time of inter-operational moves called the transfer time. The waiting and transfer times depend directly on the order of the operations in *Process* and on the interpretations of robot movements.

3. Task Programming Level

The cell control algorithm synthesis process requires that we introduce conditional instructions which depend on the states of each machine d_i of the cell (segment *Cond*) and on the operational instructions that realize the actions (segments *Transfer* and *Execute*). To minimize the flow time, variants of the fundamental plan and motion interpretation should be tested by a simulator.

3.1. Discrete Event Simulation of Workcell Action

To model the cell we employ the Discrete Event System Specification (DEVS) formalism[7]. Each workstation $d \in D$ is modelled as an atomic DEVS [8]:

$$Dev_d = \langle X(d), S(d), \delta^d_{int}, \delta^d_{ext}, ta^d \rangle \tag{6}$$

where: $X(d)$ is a set, the external input event types, $S(d)$ is a set, the sequential states, δ^d_{int} is a function, the internal transition specification, δ^d_{ext} is a function, the external transition specification, and ta^d is a function, the time advance function.

The capacity of the workstation d is equal to $C(d) = C_b(d) - 1$, where $C_b(d)$ denotes the capacity of the workstation's buffer. By $NC\_Reg(d)$ (NC program register) we denote the set of operations performed on the workstation d. The state set of the workstation d is defined as follows:

$$S_d = S_s \times S_b \tag{7}$$

where S_s is the workstation state set, S_b is the state set of the workstation's buffer. The state set S_s is given by

$$S_s = S_A \cup S_B \cup S_C$$

where:

- $S_A = \{free\}$ signifies that the **workstation is free**
- $S_B = \{(prc, q) | q \in NC\_Reg(d)\}$ and (prc, q) signifies that the **workstation is busy processing an operation** q
- $S_C = \{(com, q)\ q \in NC\_Reg(d)\}$ and (com, q) signifies that the **workstation has completed the** q**-operation and is not free**

The state of the workstation's buffer is described by a vector, whose coordinates specify a state of each position in the buffer. Assume that the $i-th$ index denotes the location of the i-th position in the buffer.

Then, let $s(d) = (s_m, (s_{bi}\ i = 1, \ldots, C_b(d)))$ be the internal transition function $\delta^d_{int} : S(d) \rightarrow S(d)$ specified as follows:

$$\delta^d_{int}(s(d)) = \begin{cases} ((com, q), (s_{b1}, s_{b2}, \ldots, s_{bC_b(d)})) \Leftrightarrow & s_m = (prc, q) \in S_B \\ ((prc, q), (s_{b1}, \ldots, s_{bi-1}, s_{bi+1}, \ldots, s_{bC_b(d)})) \Leftrightarrow & s_m = free \in S_A \\ & \wedge (\exists i)(i = \min_j \{j | s_{bj} = (q, 0)\}) \\ (free, (s_{b1}, \ldots, s_{bi-1}, (q, 1), s_{bi+1}, \ldots, s_{bC_b(d)})) \Leftrightarrow & s_m = (com, q) \wedge s_{bi} = \# \end{cases}$$

The external transition function δ^d_{ext} for each workstation d is defined in a similar way. We can describe the model of a production store following the above specification. The activation of each workstation d is caused by an external event generated by the models of the robots. Such models are realized by a coupling of a DEVS generator and acceptor[8]. The events generated by the robots depend on the states of the workstations (d, s) and the fundamental plan *Process*.

The model of the robot r is defined by the following DEVS:

$$Robot_r = \langle S_r, \delta_{ext}, ta_r, \{Z_{rd}\} \rangle \tag{8}$$

The DEVS model of each robot contains the state set $S_r = S_{ar} \times Positions \times HS$, where S_{ar} is the subset of the acceptor output, *Positions* is the set of positions of the robot's effector-end in the base-Cartesian space, HS is the effector state set. The last component of *Robot*, the function Z_{rd} generates external events for the workstation model d [8]. The robot's model also generates external events (i.e., PICKUP, and PLACE) for machines Dev_d, which trigger their corresponding simulators.

3.2. Experimental Frame of Cell Model

The fundamental plan (machining sequence) *Process* determines the cell's control principles. The variants of *Process* obtained from the process planner can be tested by a discrete event simulator. The coupling of the acceptor and generator produces elementary actions of robots which carry out selected technological operations. The interpretation of these actions is done using a motion programming approach in which detailed paths and trajectories, gross and fine motion, grasping and sensing are specified. Variant interpretations of the motion commands result in different realizations of a task. The most important parameters are the time it takes to complete an operation o, τ^o_d, and the time the robot requires to service a workstation. The time τ^o_d depends on the type of machine on which the operation o is processed. It is fixed but can be changed by replacing the machine. Similarly, the times of PICKUP and PLACE operations are determined by the type of part and machine on which the part is processed. The times of the robot's inter-operational moves (transfers), τ^{motion}_r, depend on the geometry of the work scene and the cost function of the robot's motion. This cost function determines the dynamics of motion along the geometric tracks and the duration of the moves. These data must be accessible in order to simulate the entire production system.

A *motion planner* is used for each individual robot action to create all valid interpretations of robot commands [10,11,12]. The planner generates variants of collision free time trajectories of the manipulator. It interprets and simulates the movement commands based on a geometric model of the robot and its workscene. In addition, it provides time parameters for the robot's actions. The interpretations obtained from the motion planning layer allow us to test and select the control algorithm - *Process* - which minimizes the makespan.

4. Summary and Conclusions

A comprehensive framework for design of an intelligent cell-controller will require integration of several layers of support methods and tools. We are developing methods and tools that should facilitate:

- automatic generation of different plans of sequencing operations
- synthesis of action plan for robots servicing the devices
- planning and interpretation of robots' motion actions in the geometric model of workscene
- synthesis of the workcell simulation programs
- testing and verification of control variants based on the interpreted programs of robots' actions and simulation modeling of the overall cell architecture

Such techniques are imperative to achieving high autonomy in manufacturing systems.

5. References

1. E.G. Coffman, M. Elphick, A. Shoshani. System Deadlock.*Computing Surveys*, **3**(2),67-78, (1971)
2. B.H. Krogh, Z. Banaszak. Deadlock Avoidance in Pipeline Concurrent Processes. *Proc. of Workshop on Real-Time Programming* IFAC/IFIP. (1989)
3. A. Kusiak *Intelligent Manufacturing Systems*. Prentice Hall. (1990)
4. J.E. Lenz. *Flexible Manufacturing*. Marcel Dekker. Inc.. (1989)
5. L.S. Homem De Mello and A.C. Sanderson. AND/OR Graph Representation of Assembly Plans. *IEEE Trans. on Robotics and Automation.*, **6**(2), 188-199. (1990)
6. A.C. Sanderson, L.S. Homem De Mello and H. Zhang. Assembly Sequence Planning. *AI Magazine*, **11**(1), (1990)
7. J.W. Rozenblit and B.P. Zeigler. Design and Modelling Concepts, in: *International Encyclopedia of Robotics, Applications and Automation*. (ed. Dorf, R.) John Wiley and Sons. New York. 308-322, (1988)
8. B. Faverjon. Object Level Programming of Industrial Robots. *IEEE Int. Conf. on Robotics and Automation*. **2**. 1406-1411. (1986)
9. W. Jacak and J.W. Rozenblit. Automatic Simulation of a Robot Program for a Sequential Manufacturing Process. *Robotica*.(1992)
10. W. Jacak. Strategies for Searching Collision-Free Manipulator Motions: Automata Theory Approach. *Robotica*, **7**. 129-138, (1989)
11. 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, (1991)
12. W. Jacak and J.W. Rozenblit. Simulation-based Intelligent Control of Workcell Planning. *IEEE Trans. on Robotics and Automation.*, (submitted)
13. K. Shin, N. McKay. A Dynamic Programming Approach to Trajectory Planning of Robotic Manipulators. *IEEE Trans. on Automatic Control*, **31**(6), 491-500. (1986)