Evaluation of assembly routines with multitasking execution in a physical robotic cell
by Sergio San Martin

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Industrial and Management Engineering
Montana State University

Abstract:
This research covered the development of multitasking execution programs and evaluation of twelve assembly sequences, in terms of efficiency and effectiveness, applied to a robotic cell under two software control methods. The assembly sequences were defined via analytical methods and verified later with a physical simu I at ion.

The analytical methods used to define the sequences were the SPT rule, the LPT rule and the Branch and Bound algorithm. The software control methods were a single task execution program, and a multitask execution program. The single task execution programs performed all the activities in a sequencial mode. The multitask execution program allowed two activities to run simultaneously.

The physical simulation was performed in a robotic cell containing two TeachMover robot arms, a central assembly area, and two bin cells built with Fischertechnik components. The part assembled was a representation of a circuit board with four microchips made of machinable wax.

The control software was coded using ARMBASIC for the robot arms and TurboBASIC for the main program.

The analytical results showed that using a single robot, the sequence with the best completion time was generated with the Branch and Bound algorithm. Also, this result was verified with the physical simulation that generated the best completion time using the Branch and Bound algorithm.

Using two robot arms, the analytical results showed that the Branch and Bound algorithm under a multitask execution mode generated the best assembly sequence. In this case, the physical simulation showed different results. The best sequence found with the physical simulation was an adjusted sequence, running under a multitask execution mode, that removed the physical conflicts to avoid collisions in the assembly area. The physical interference could not be observed by the analytical methods. Therefore, the use of physical simulation to evaluate robotic motions is recommended.

Evaluation of Assembly Routines with Multitasking Execution in a Physical Robotic Cell

by

Sergio San Martin

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Industrial and Management Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

March 1989

APPROVAL

of a thesis submitted by

Sergio San Martin

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

2/23/89
Date

Chairperson, Graduate Committee

Approved for the Major Department

2/24/89
Date

Head, Major Department

Approved for the College of Graduate Studies

2/28/89
Date

Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the library shall make it available to borrowers under rules of the library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Dean of Libraries when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature 

Date 2/23/89

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

ABSTRACT

This research covered the development of multitasking execution programs and evaluation of twelve assembly sequences, in terms of efficiency and effectiveness, applied to a robotic cell under two software control methods. The assembly sequences were defined via analytical methods and verified later with a physical simulation.

The analytical methods used to define the sequences were the SPT rule, the LPT rule and the Branch and Bound algorithm. The software control methods were a single task execution program, and a multitask execution program. The single task execution programs performed all the activities in a sequencial mode. The multitask execution program allowed two activities to run simultaneously.

The physical simulation was performed in a robotic cell containing two TeachMover robot arms, a central assembly area, and two bin cells built with Fischertechnik components. The part assembled was a representation of a circuit board with four microchips made of machinable wax.

The control software was coded using ARMBASIC for the robot arms and TurboBASIC for the main program.

The analytical results showed that using a single robot, the sequence with the best completion time was generated with the Branch and Bound algorithm. Also, this result was verified with the physical simulation that generated the best completion time using the Branch and Bound algorithm.

Using two robot arms, the analytical results showed that the Branch and Bound algorithm under a multitask execution mode generated the best assembly sequence. In this case, the physical simulation showed different results. The best sequence found with the physical simulation was an adjusted sequence, running under a multitask execution mode, that removed the physical conflicts to avoid collisions in the assembly area. The physical interference could not be observed by the analytical methods. Therefore, the use of physical simulation to evaluate robotic motions is recommended.

CHAPTER I

INTRODUCTION

Industrial robot applications usually involve several pieces of hardware in addition to the robot. These other hardware components include conveyers, pallets, machine tools, fixtures, and so on. In some applications several robots must be integrated into a single work cell. It is important the equipment in the cell be organized into an efficient layout [21].

Assembly is one of the growing application areas for industrial robotics. What makes robots useful in the assembly applications is their capability to execute programmed variations in the work cycle to accommodate different assembly configurations. To be able to utilize this capability, a variety of problems must be solved. Technological feasibility, flexibility, assemblability, and producibility are included among these problems. The flexibility of robots is defined as a degree of variation in part types to be assembled and assembly sequences that a robot can accept in controlling and programming.

The purpose of this research was to introduce good control schemes for a robotic assembly cell through

physical simulation. These schemes for the robotic cell were defined by efficient and effective combinations of assembly routines. The efficiency was determined by the cell utilization and the effectiveness was determined by other performance criteria such as the completion time and mean flow time. The performance criteria were used to evaluate the cell performances along with production scheduling rules such as the shortest processing time (SPT) and the longest processing time (LPT). Furthermore, the Branch and Bound algorithm and its tree were considered to seek for a better sequence.

The robotic cell consisted of two table-top TeachMover arms, one assembly area, two identical bin cells and a Zenith microcomputer to control the system. To construct the entire cell, Fischertechnik components were used as basic building blocks. The cell was an assembly robot cell in which two arms were located at the sides of the cell. This arrangement was suited to perform some processing or assembly operations on each work part. The product assembled in the robotic cell was a representation of a circuit board with four microchips. The sizes of the board and microchips were enlarged because of the low resolution of the TeachMover for tiny workpieces.

In this research, the analytical results of the efficiency and the effectiveness from the SPT rule, the LPT

rule, and the Branch and Bound algorithm for a variety of assembly routines were compared to the physical simulation results under the same operating conditions. The objective of this comparison was to observe all possible physical problems which occurred when the analytical results were directly applied to the cell. This comparison resulted in several suggestions regarding better robotic assembly cell designs. This research also suggested the best assembly routines selected from a variety of assembly routines (sequences).

Finally, a multitask execution programming method was adapted for TeachMover arms in this research and the cell performance with this method was compared to that with other assembly methods.

CHAPTER 2

REVIEW OF RELEVANT LITERATURE

Robotic Assembly Operations

The use of robots in industry has a great variety of applications. One of the best examples is found in the assembly industry.

The use of robots to perform assembly tasks has been the subject of much research since the mid 1970's, but only recently has it been shown to be technically and economically viable within general manufacturing environments. As with advances in other technologies, the real world of industry lags far behind the frontiers of research into robotized assembly, because the 'new discoveries' need to be both proved and adapted to industry.

Robots have gradually evolved from primitive manipulators with virtually zero intelligence, to devices that have high levels of 'pseudo intelligence' that can communicate with other machines and react to changing work environments. The development and widespread availability of microcomputers have permitted the development of very

sophisticated control and sensing systems.

Before the installation of robots, it is important to ensure that the assembly tasks are compatible with the robots capabilities, otherwise failures such as technical conflicts and financial loss will result. In addition, the product being assembled must be designed for automation.

An automated process requires that all activities be provided with the necessary materials and that they be performed satisfactorily at both physical and functional levels [11].

The use of appropriate software and hardware accessories allows a 'standard' robot to be customized to a specific task. The robot's flexibility is limited only by its size, payload, repeatability and degrees of freedom.

Drezner [2] presented specific concepts for the robot assembly systems. In robotic assembly stations, robots pick a series of component parts from bins and then insert and assemble them. In programmable assembly of discrete products, component parts are selectively picked by robotic arms and inserted or attached to their respective position. Assembly tasks include repeated sequences of pick-move-insert operations. Thus the performance of the assembly system will depend on the planning and control of the assembly sequences.

Basic types of assembly systems vary according to

their degrees of automation. Boothroyd [12] defined six typical assembly systems of three main classes:

1) Manual, either with part transfer devices or with part feeders.

2) Special purpose automatic, either with indexing transfer or with free transfer of parts.

3) Programmable assembly with work done either on a fixture or on a moving conveyor.

A programmable assembly system would typically include the following five classes of components:

1) One or more robot arms.
2) Parts and products, input/output mechanism, eg. conveyor.
3) Parts storage devices, eg. feeders, bins, pallets.
4) Stationary peripheral machines, eg. press.
5) Assembly tools, eg. powered screw driver.

Drezner [2] identified the problems of designing and planning that occurred within the programmable assembly systems. He called these problems assembly plan problems. He also offered solutions to those problems. According to Drezner, the assembly plan problem was defined as the problem of how to physically arrange assembly cell components with the objective to optimize pick, place and insert motions [2]. The assembly design was performed prior to the assembly plan and consisted in the identification of the following items:

1) The specific parts and quantities to be assembled into the product.

2) The physical placement and location of each part on the assembled product.

The assembly plan problem basically was dependent upon three characteristics:

1) The assembly mode in terms of product variety.
2) The variety and quantity of assembly parts.
3) The sequence of assembly if any is given.

With regard to the mode of assembly operation, programmable assembly cells operated in one of three basic modes: single product, changeable product and multiple product. The single product mode comprised the assembly operations for one basic product where product design changes were likely. The changeable product mode refereed to the assembly operations for several products and models. The cells, however, operated the model or product at a same time. Finally, the multiple product model comprised the assembly operations for interleaved assemblies of several products and models during the same period.

The variety of assembly parts of different types and the quantity of separated parts composed of one assembly were important factors in planning assembly operations. Drezner [2] also described the determination of the optimal location of each cell component and the optimal sequence of assembly as critical to minimize the assembly plan problem.

Figure 1 shows the different assembly modes by product and model variety [2].

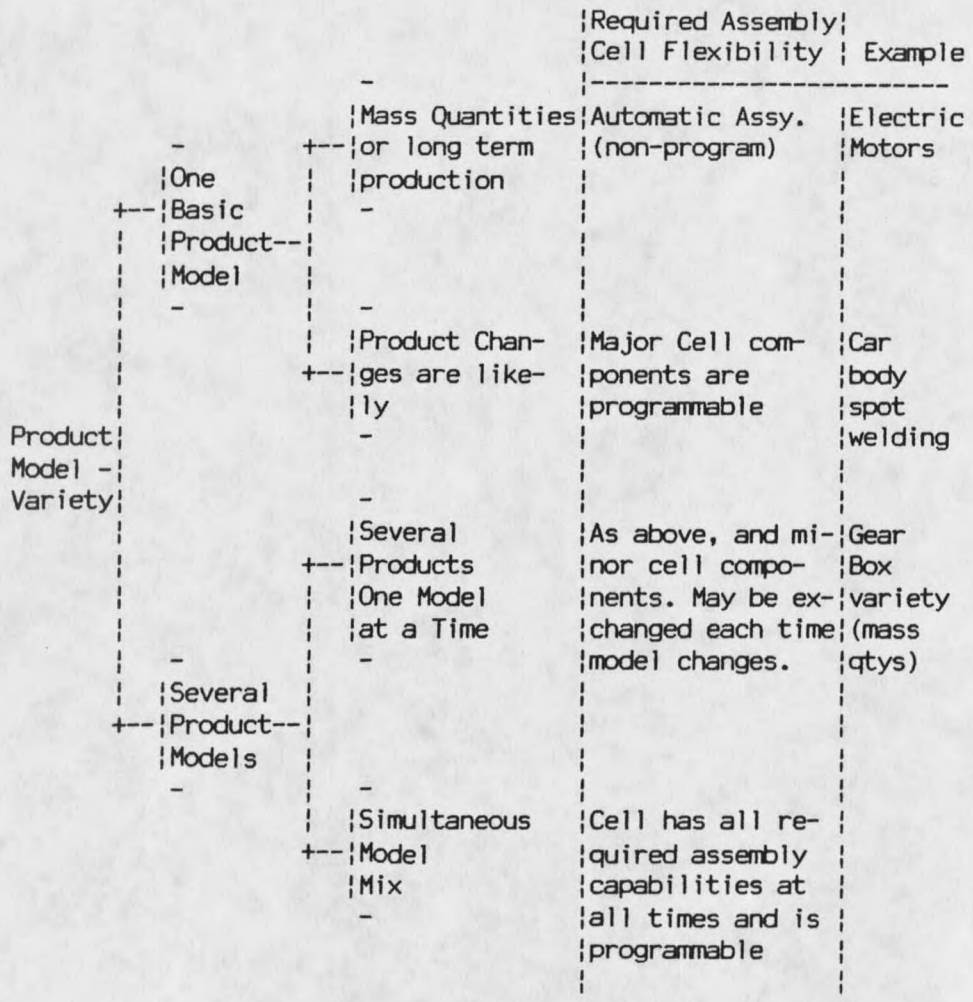

Figure 1. Assembly Modes by Product or Model Variety

Formulation of the Assembly Plan Problem

The simple assembly plan problem can be formulated when the number of bin cells "**n**" is equal to the number of parts in the assembly place. The total time of movements made by a robot with an empty gripper on the way back is to be minimized.

One of the optimization techniques for solving the simple assembly plan problem is the Branch and Bound approach. This approach was originally developed for solving the traveling salesman problem. This approach has been used very extensively in order to minimize the total traveling distance of a salesman and was adapted for this research.

Branch and Bound Approach

This approach [16] was presented as a traveling salesman problem (tsp) in which a salesman must visit each of "n" cities once and only once and return to his point of origin, and do so in a way that minimizes the total distance traveled (or total time, or cost, etc.). Each city corresponds to a location; and the distance between cities corresponds to the time required to change over from one location to another [16]. In general, distances, times, etc. were given by a matrix S having a size of n by n. The object was to find a matrix X having a size of n by n with constraints of

$$x_{ik} = 0 \text{ or } 1, \quad i = 1,2, \ldots,n, \quad k = 1,2, \ldots,n,$$

$$\sum_{i=1}^{n} x_{ik} = 1, \quad k = 1,2, \ldots,n,$$

$$\sum_{k=1}^{n} x_{ik} = 1, \quad i = 1,2, \ldots,n,$$

and minimize

$$\sum_{i=1}^{n}\sum_{k=1}^{n} x_{ik}s_{ik}$$

where

s_{ik} = the traveling time from location i to location k

and

x_{ik} = a decision variable representing a location

Up to this point the problem is similar to the resource allocation problem. For the traveling salesman problem, however, there was an additional restraint that no tour could return to its starting point until all n cities had been visited.

Multitask Real-Time Programming

Real Time

The term "real time" covers a wide range of computer (and other) systems, but shares the common feature that results are demanded by deadlines imposed by the "real" world outside the system. The range of time scales is immense, from microseconds to years [17].

Wright [9] defined real time or on line as follows:

"An on-line computer system accepts input directly from the user or process that creates it and returns output directly to the user or process that requires it, regardless of where they are located". Wright divided on-line systems into the following five major categories:

. process control systems
. data acquisition systems
. business- oriented information systems
. scientific/engineering time sharing systems
. remote batch systems

All of these systems work with the operating system of a computer. Wright identified five operating systems for small computers:

. terminal interface monitor: this is used by only the simplest single application dedicated machines. This type of monitor handles process interruptions and also allows the user to start a program, display or alter memory locations, set breakpoints and load or dump programs.

. input/output monitor: this handles its own initialization and control as well as communication between itself, system programs, user programs and a simple I/O subsystem.

. keyboard monitor: this is a single-user operating system which requires some form of bulk storage such as magnetic tapes or discs. It includes all the facilities of an I/O monitor plus routines to accept and act on console

commands as well as the ability to modify I/O assignments dynamically.

. background/foreground monitor: this is a dual program executive that includes all the facilities of the keyboard monitor and also controls processing and I/O in a time-shared or interrupt-driven environment. The term "foreground" means that a program can be seen, running, by the operator in the screen, while in the "background", the program runs without being shown in the screen.

. multiprogramming executive: this includes all the facilities of the background/foreground monitor with additional capability for concurrent execution of several hundred foreground jobs.

Multitasking

When several independent processes are going on at once, it is convenient to handle each one with an independent program. When all the programs are running on the same computer (processor), the result is called multitasking.

Multitasking is done via a piece of software called an operating system, whose job is to make each program, called task or process, appear to be running on its own separate computer. This trick is accomplished by saving the processor "state" for one task, and loading the state of

another task. There is a vast range of different kinds of operating systems, which differ according to the facilities they provide for scheduling tasks, synchronizing their actions, and passing data between them. Operating systems intended for real time applications tend to provide a wide range of these facilities.

There are two major ways of organizing a set of tasks for multitasking: either all tasks are identical programs operating on different data, or each task does a single operation on a set of data. Figure 2 shows a memory map for a multi-task real-time operating system.

An application program should be defined in terms of tasks. White [14] described a task as a logically complete program segment or the smallest program to which system resources (I/O devices, CPU, memory, etc.) might be allocated.

In a multitasking, single-processor system, only one task actually can be in control of the CPU, i.e. be executing, at a given instant.

Wright [9] established four states that the tasks could be in: executing, ready, suspended and inactive.

Task States

EXECUTING (or running): The executing task is the highest priority task determined by a CPU that is "ready"

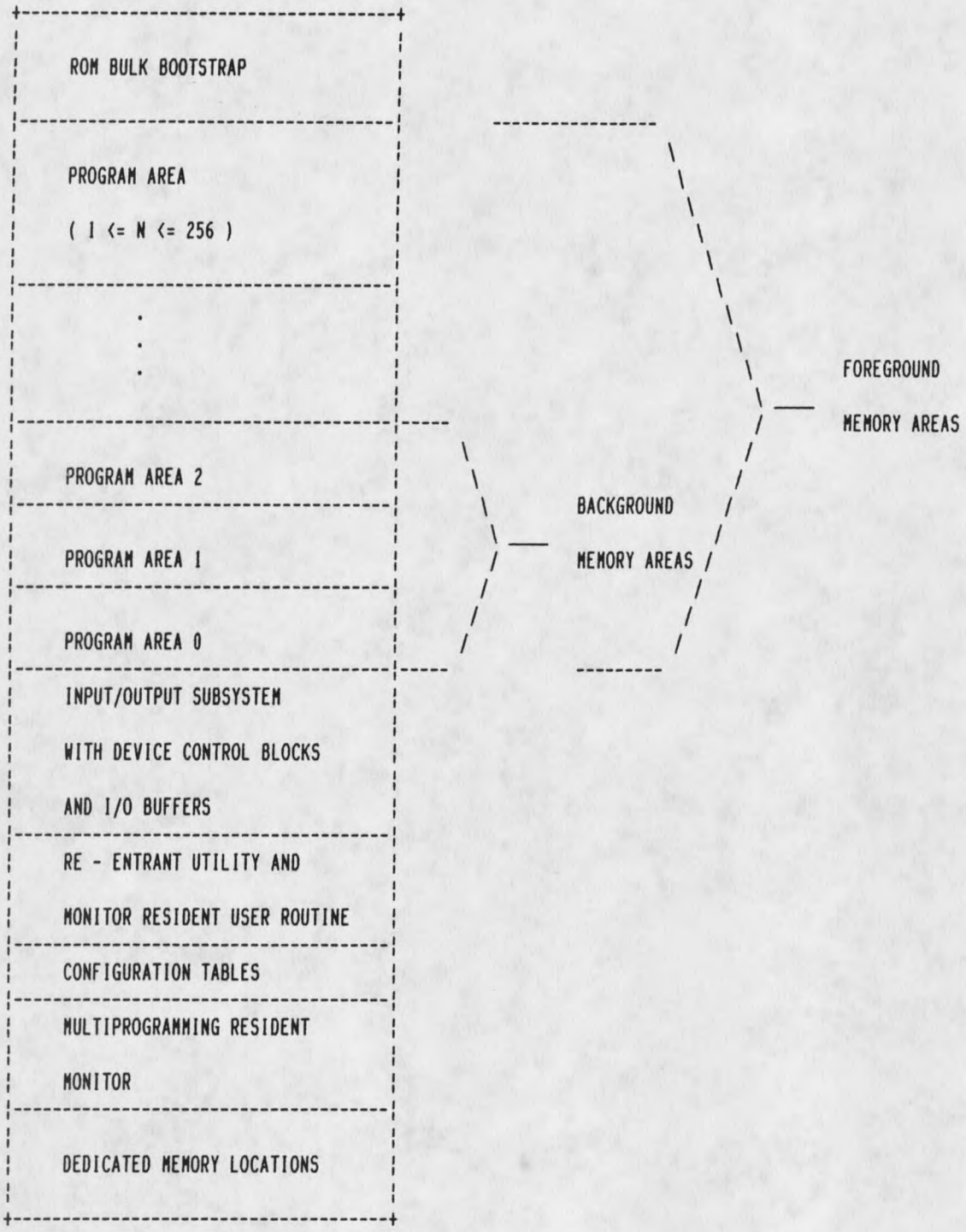

Figure 2. Memory Map for Multitasking Real-Time Operation

to run.

READY (or active or installed): this routine states that one or more tasks may reside until their probability levels dictate that they should have control of the CPU.

SUSPENDED (or blocked): The task has been halted until some real time event or communication from an executing task occurs.

INACTIVE (or dormant): The task, even though potentially residing in main memory, has either not yet been introduced to the operating system (scheduler), or that it has completed an assignment and has been removed from the queue of jobs to be executed.

The tasks states and transition paths for a multitasking environment are shown in Figure 3.

Every application program runs under an operating system, and each operating system has its own structure. For a real-time operating system, Wright [9] described the structure as a real time multitasking executive. The multitasking executive allows a CPU to execute other tasks while one (or more) of them is waiting for input (or output) to be handled by the operating system.

Figures 4 and 5 show a comparison between a single task operating system and a multitask operating system.

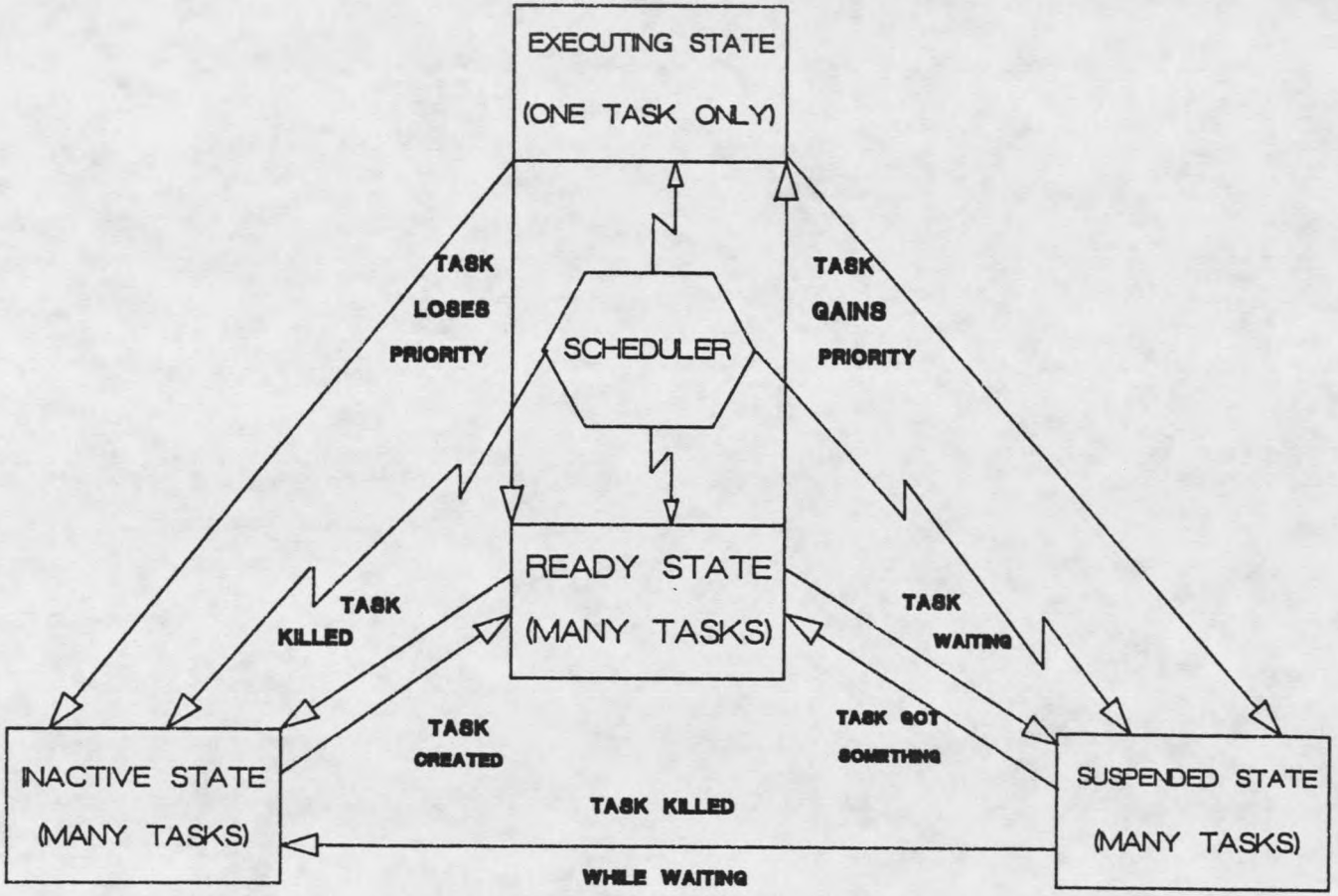

Figure 3. Task States and Transition Paths

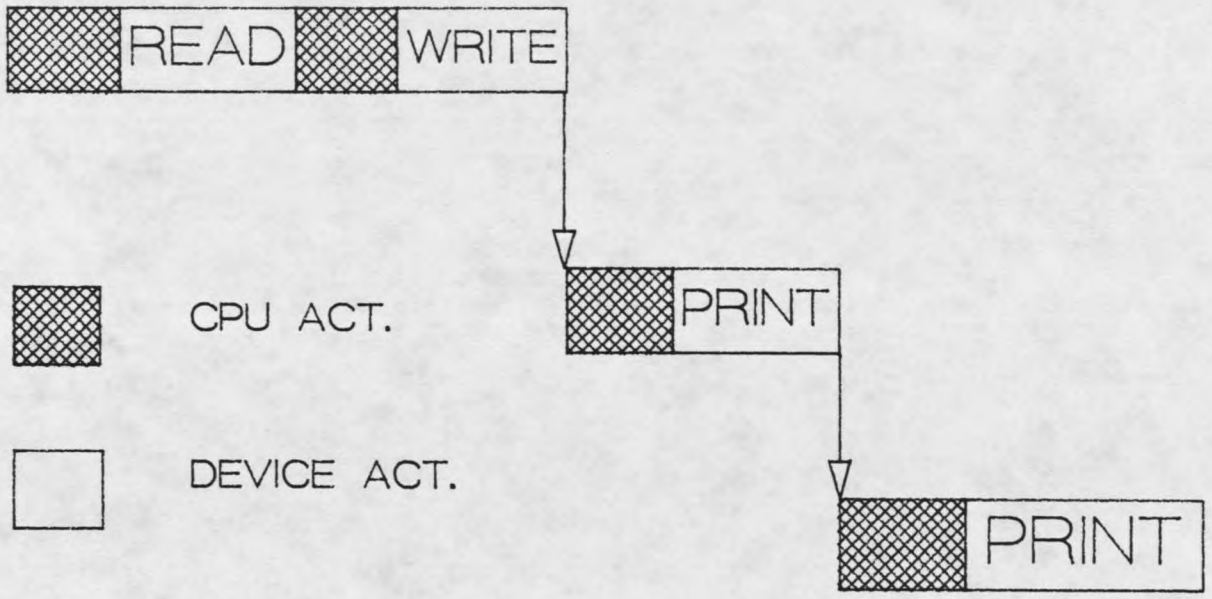

Figure 4. Single task Operating System

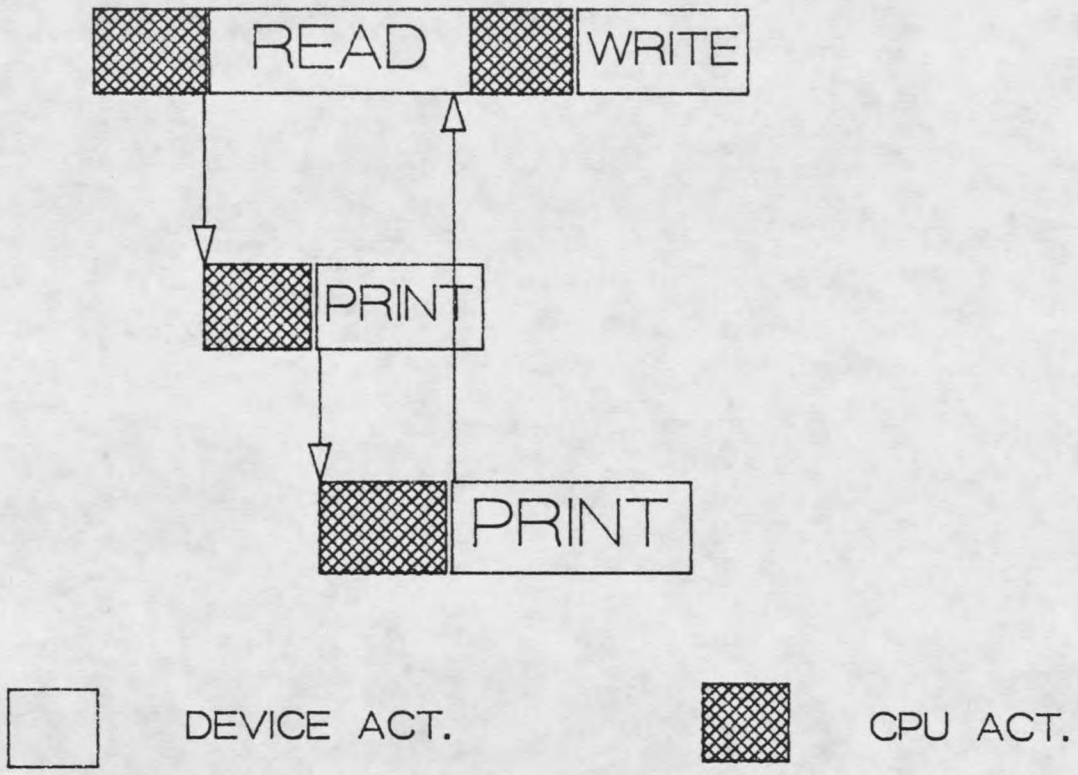

Figure 5. Multitask Operating System

In summary, the basic components of a multitask real-time system include:

. The operating system, including an interrupt handler or priority control program, a scheduler, a set of I/O routines, and in some systems, memory allocation and file handling routines. Each of these concepts is briefly described below.

. The user or application-oriented tasks.

. System calls, macro commands.

Priority Control Program [17]

The main difference between the simple batch system and a real-time system is summarized below:
In the batch system, the program initiates and waits for all input/output, and each stage of data processing is well defined, whereas in a real-time system, the program must respond to a series of random and frequently unrelated events. The means by which this is accomplished is through hardware or software routines. Each routine responds to and acknowledges hardware interrupts and controls internal system priorities so that requests for CPU service by I/O devices can be granted. Facilities must exist to have low priority devices interrupted by requests from higher-priority ones, and in particular, to respond quickly to emergency alarm conditions such as powerfail interrupts.

Scheduler Program [17]

The basic function of the scheduler is to determine

which task (user task or system task) has highest priority and then to raise it to the executing state. In addition, the scheduler must maintain the task "ready" queue. For each task in the "ready" state the scheduler must know its priority, the status of all the active CPU registers, and the link words to the next-higher-priority ready task in the queue. These have been defined in the task data blocks which the scheduler maintains and uses to restore the status of any ready task being raised to the executing state. In addition, the scheduler must know the priority of the task that is being executed. Tasks may be introduced to the scheduler in three ways: by a system macro call or equivalent assembly language macro command (a preprogrammed routine allocated in memory) [20] to raise a task from the inactive state to ready state; by a user task request for input or output of information (data), or by requesting for system resources (CPU) such as one made by the interrupt handling routine when a device causes an interrupt.

Real-Time Macro Commands [17]

The system macro commands is a preprogrammed general routine that is allocated in one address of the memory and

may be grouped into eight categories:

- Input/Output commands
- Task creation and deletion commands
- Intertask communication commands
- Overlay and special queuing commands
- Clock commands
- Task identification commands
- Task/operator communication commands
- Operator/task interaction commands

MICROBOT TeachMover

The TeachMover robot arm is a microprocessor-controlled, six-jointed mechanical arm designed to provide an unusual combination of dexterity and low cost [15].

The TeachMover arm can be used in either of the following modes:

1) Teach control Mode, in which the hand-held teach control can be used to teach, edit, and run manipulation programs.

2) Serial Interface Mode, in which the TeachMover arm can be controlled by a host computer or a computer terminal via one or two built-in RS-232 asynchronous serial communications lines.

The microprocessor card is housed in the base. The teach control cable and the D.C. power cord extend from the rear of the base. The two RS-232C connectors are also located at the rear of the base. The body swivels relative to the base on a hollow shaft attached to the base. This shaft is called the base joint. Six stepper motors with gear assemblies are mounted on the body and control each of the six joints. The power wires for the motors pass from

the computer card in the base through a hollow shaft to the body. This arrangement provides a direct cable-drive system. The upper arm is attached to the top of the body and rotates relative to the body on a shaft called the shoulder joint. Similarly, the forearm is attached to the upper arm by another shaft known as the elbow joint. Finally, the hand, also called gripper, is attached to the forearm by two wrist joints. Two separate motors operate the wrist joints to control the pitch and roll of the hand.

The TeachMover arm has a lifting capacity of one pound when fully extended, and a resolution of 0.011 inches. The end of the hand can be positioned anywhere within a partial sphere with a radius of 17.5 inches. The maximum speed is from 2 to 7 inches per second, depending upon the load.

Scheduling Rules

There are many possible scheduling objectives. The most important is to increase the utilization of the resources that is, to reduce the resource idle time [18]. For a finite set of tasks, the utilization of resources is inversely proportional to the time required to accomplish all the tasks. This time is referred to as the makespan or maximum flow time of schedule. In a finite problem, resource utilization is improved by scheduling the set of tasks so as to reduce makespan.

Another important scheduling objective is to reduce in process inventory; that is, to reduce the average number of tasks waiting in a queue while the resources are busy with other tasks.

One final objective for scheduling is to reduce some function of tardiness. In many situations, some or all of the tasks have due dates and a penalty is incurred if a task is finished after that date. One can reduce the maximum tardiness, or one can reduce the number of tasks that are tardy.

There are numerous scheduling algorithms or rules in order to achieve the objectives described above [18]. Conway, Maxwell and Miller [16], however, proved mathematically and experimentally that the shortest processing time rule (SPT) minimized the mean flow time in various manufacturing system structures. Especially, this rule proved to be the best performer in job shops that processed n jobs using either single processor (machine) or multiple processors (machines) on the basis of a variety of performance criteria such as mean flow time, mean lateness, and mean tardiness.

Shortest Processing Time (SPT) [18] When scheduling n tasks on a single processor, the mean flow is minimized by sequencing the shortest processing time (SPT) task first,

that is,

$$t_1 <= t_2 <= t_3 \ldots <= t_n$$

When scheduling n tasks on a single processor, mean lateness is minimized by sequencing in the order of

$$t_1 <= t_2 <= t_3 \ldots <= t_n$$

Longest Processing Time [18] (LPT)

When scheduling n tasks on a single processor, the mean flow is maximized by sequencing the longest processing time (LPT) task first, that is

$$t_1 => t_2 => t_3 \ldots >= t_n$$

The following chapter will present a detailed application of the previous concept into a physical robotic cell simulator.

CHAPTER 3

METHOD OF ANALYSIS

Description of the Robotic Cell

The robotic cell that was used for this research consisted of the following elements:

1. Two TeachMover robotic arms
2. One central assembly area built with Fischertechnik components
3. Two identical parts bins built with Fischertechnik components
4. One Zenith computer model ZW-241-82 with two RS-232 ports.

Figure 6 shows a layout of the cell.

The part assembled was a mimic of a circuit board. The dimensions of the board were 6.81 X 2.81 X .5 in. and it was made of blue machinable wax. The microchips were numbered from 1 to 4 and had different dimensions. The dimensions were as follows:

1.	chip # 1	2.87 X 1 X 1.18 in.
2.	chip # 2	2.25 X 1 X 1.18 in.
3.	chip # 3	1.31 X 1 X 1.18 in.
4.	chip # 4	1.00 X 1 X 1.18 in.

Figure 7 shows a layout of the circuit board with the location of the microchips.