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
© Copyright by Sergio San Martin (1989)

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 sequential 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

© COPYRIGHT

by

Sergio San Martin

1989

All Rights Reserved.

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

Chris Hoatson
Chairperson, Graduate Committee

Approved for the Major Department

2/24/89
Date

David J. Nelson
Head, Major Department

Approved for the College of Graduate Studies

2/28/89
Date

Henry L. Parsons
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

	Page
1. INTRODUCTION	1
2. REVIEW OF RELEVANT LITERATURE	4
Robotic Assembly Operations	4
Formulation of the Assembly Plan Problem ..	9
Branch and Bound Approach	9
Multitask Real-Time Programming	10
Real Time	10
Multitasking	12
Task States	13
Priority Control Program	19
Scheduler Program	19
Real-Time Macro Commands	20
Microbot TeachMover	21
Scheduling Rules	22
Shortest Processing Time	23
Longest Processing Time	24
3. METHOD OF ANALYSIS	25
Description of the Robotic Cell	25
Assembly Operations	28
Assembly Sequences	28
Assembly Scheduling	38
Multitasking Program	39
Performance Criteria	43
4. DATA COLLECTION AND ANALYSIS	44
Analytical Results	44
Physical Simulation Results	50
5. SUMMARY AND CONCLUSIONS	63

TABLE OF CONTENTS (Continued)

REFERENCES CITED	66
APPENDICES	69
Appendix A: Branch and Bound Trees Single Case	70
Appendix B: Branch and Bound Trees	
Multitasking	75
Appendix C: Completion Time for Single Robot	
Trees	84
Appendix D: Completion Time for Double Robot	
Trees	89
Appendix E: Source Code for Programs "ULL",	
"ULL1", "DLL", "DLL1"	98
Appendix F: Source Code for Program "TESIN1" ...	103

LIST OF TABLES

Table	Page
1. Assembly Methods	30
2. Traveling Times for Robot A and Robot B	35
3. Assembly Sequences	44
4. Analytical Results	45
5. Performance of the Rules	48
6. Physical Simulation Results	53
7. Number of Physical Conflicts	56
8. Analytical Results vs. Physical Simulation Results	60
9. Summary of the best sequences	62
10. Completion Time for Branches 1 and 2 - Robot B .	85
11. Completion Time for Branches 3 and 4 - Robot B .	86
12. Completion Time for Branches 1 and 2 - Robot A .	87
13. Completion Time for Branches 3 and 4 - Robot A .	88
14. Completion Time for Pair (1,2) Multitasking	90
15. Completion Time for Pair (1,3) Multitasking	91
16. Completion Time for Pair (1,4) Multitasking	92
17. Completion Time for Pair (1,5) Multitasking	93
18. Completion Time for Pair (2,1) Multitasking	94
19. Completion Time for Pair (3,1) Multitasking	95
20. Completion Time for Pair (4,1) Multitasking	96

LIST OF TABLES (Continued)

21. Completion Time for Pair (5,1) Multitasking	97
--	----

LIST OF FIGURES

Figure	Page
1. Assembly Modes by Product or Model Variety	8
2. Memory Map for Multitasking Real-Time Operation ...	14
3. Task States and Transition Paths	16
4. Single Task Operating System	17
5. Multitask Operating System	18
6. Layout of the Robotic Cell	26
7. Layout of the Circuit Board	27
8. Operations Sequence	31
9. General Branch and Bound Tree for Robot A	33
10. Branch and Bound Tree for Chip #1 Robot A	34
11. General Multitask Branch and Bound Tree	37
12. Program Executing Sequence	40
13. Multitask Executive Program	41
14. Analytical Completion Time for Robot A with the Branch and Bound Algorithm	47
15. Completion time using Robot A and Robot B under Multitask Execution w/a Branch and Bound Algorithm	49
16. Physical Paths followed by Robot A and Robot B under the Branch and Bound solution	55
17. Physical Paths followed by Robot A and Robot B under the LPT Rule solution	57
18. Physical Paths followed by Robot A and Robot B under an Adjusted sequence	59

LIST OF FIGURES (Continued)

19. Branch and Bound Tree for Chip #2 Robot A	71
20. Branch and Bound Tree for Chip #3 Robot A	72
21. Branch and Bound Tree for Chip #4 Robot A	73
22. General Branch and Bound Tree for Robot B	74
23. Multitask B. and B. Tree for pair 1,2	76
24. Multitask B. and B. Tree for pair 1,3	77
25. Multitask B. and B. Tree for pair 1,4	78
26. Multitask B. and B. Tree for pair 1,5	79
27. Multitask B. and B. Tree for pair 2,1	80
28. Multitask B. and B. Tree for pair 3,1	81
29. Multitask B. and B. Tree for pair 4,1	82
30. Multitask B. and B. Tree for pair 5,1	83
31. Source code for program "ULL"	99
32. Source code for program "ULL1"	100
33. Source code for program "DLL"	101
34. Source code for program "DLL1"	102
35. source code for program "TESIN1"	104

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 sequential 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 referred 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].

		Required Assembly Cell Flexibility		Example
Product Model - Variety	-	-	-	-
	-	Mass Quantities	Automatic Assy.	Electric
	+--	or long term	(non-program)	Motors
	-	production	-	-
	-	-	-	-
	-	-	-	-
Product Model - Variety	-	Product Chan-	Major Cell com-	Car
	+--	ges are like-	ponents are	body
	-	ly	programmable	spot
	-	-	-	welding
	-	-	-	-
	-	-	-	-
Product Model - Variety	-	Several	As above, and mi-	Gear
	+--	Products	nor cell compo-	Box
	-	One Model	nents. May be ex-	variety
	-	at a Time	changed each time	(mass
	-	-	model changes.	qtys)
	-	-	-	-
Product Model - Variety	-	Several	-	-
	+--	Product--	-	-
	-	Models	-	-
	-	-	-	-
	-	-	-	-
	-	-	-	-
Product Model - Variety	-	Simultaneous	Cell has all re-	-
	+--	Model	quired assembly	-
	-	Mix	capabilities at	-
	-	-	all times and is	-
	-	-	programmable	-
	-	-	-	-

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

$$\begin{aligned}
 x_{ik} &= 0 \text{ or } 1, \quad i = 1, 2, \dots, n, \quad k = 1, 2, \dots, n, \\
 \sum_{i=1}^n x_{ik} &= 1, \quad k = 1, 2, \dots, n, \\
 \sum_{k=1}^n x_{ik} &= 1, \quad i = 1, 2, \dots, n,
 \end{aligned}$$

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,

$$t_1 \leq t_2 \leq t_3 \dots \leq t_n$$

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

$$t_1 \leq t_2 \leq t_3 \dots \leq t_n$$

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

$$t_1 \geq t_2 \geq t_3 \dots \geq t_n$$

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.

Figure 6. Layout of the Robotic Cell

Figure 7. Layout of the Circuit Board

The assembly process started by placing a circuit board on the assembly area. After the board was placed, the chips were be placed in random order and finally the board was removed. All these operations formed a cycle.

Assembly Operations

Assembly Sequences

A round-robin method was chosen for the continuous operation of the robotic cell. Once a certain assembly sequence was determined, the sequence was repeated for the number of cycles. The positions of the microchips on the board remained fixed for all the sequences.

The assembly sequences that were evaluated were divided into two groups: one for assembling the circuit board with one robotic arm, using one set of bins for parts (one bin cell) and the central assembly area, and the other for assembling the circuit board with two robot arms, using two bin cells and the central assembly area.

As stated before, the first operation was placing an unassembled circuit board. This operation was the initial movement of a TeachMover arm and fixed for every cycle. The last operation, removing the finished board, was also fixed.

The assembly operations of a circuit board were summarized as follows:

<u>OPERATION No.</u>	<u>DESCRIPTION</u>
1	Pick the circuit board from the storage bin and place it in the central assembly area.
2, 3, 4, 5	Pick randomly any available chip among four chips from a bin cell and place the selected chip on a fixed location of the board. This operation will be repeated until all four chips are assembled for the finished board.
6	Pick the finished assembly, remove it from the assembly area, and place it in finished product area.

To minimize the makespan, the Branch and Bound approach was adapted and modified to define $2 * n$ 'cities' as n bin-cells and n locations on an assembly area. The number of 'cities' was multiplied by two because a bin location had its correspondent assembly location and both were considered as single entity.

The distances (or travel times) between the 'cities' (between a bin location and an assembly location) were dependent on the part (microchip) type (1, 2, 3 or 4) because they were located at a certain place in the assembly.

Table 1 presents the assembly methods evaluated in this research.

Table 1. Assembly Methods.

	METHOD	DESCRIPTION
Group I	I - a	assemble the board with Robot A
	I - b	assemble the board with Robot B
Group II	II - a	assemble the board with both robots in sequential operation
	II - b	assemble the board with both robots with multitasking programming

Branch and Bound trees were developed for each group. Each Branch and Bound tree covered all possible combinations of the different sequences that were used to assemble a circuit board. A sequence consisted of 6 different operations described below.

1) Operation 1: This operation had two movements: one for traveling from the home position to the circuit board storage with an empty hand, and the other for picking and placing the board in the assembly area (loaded hand).

2) Operations 2,3,4 and 5: These operations also had two movements: one for traveling empty from the assembly area to the part bin, and the other for picking and placing the microchip in the circuit board.

3) Operation 6: This operation consisted of movements from the assembly area with empty hand, picking the finished product, placing it in the finished product storage area, and traveling back to the home position. Figure 8 shows

Figure 8. Operations Sequence

the assembly operations.

Figure 9 illustrates the Branch and Bound tree for Group I - a. It is referred to as a general Branch and Bound tree for robot A. This figure shows Operation 0 as the home position. The next operation, Operation 1, was fixed for all the sequences, that is, the first part placed was an unassembled board. After Operation 1 was done, there were four possible paths to follow, each one corresponding to a microchip assembled (Operations 2, 3, 4 and 5). These paths were branched according to the combination of microchips that remained. The bounding rule was established with the restriction that the last operation should be Operation 6. Figure 10 shows all possible combinations that follow operation 2. The rest of the Branch and Bound trees are illustrated in the Appendix A.

Table 2 shows the traveling time, in seconds, for the first robot (robot A) and for the second robot (robot B). The traveling time of both robots was different because the positions of the microchips were not symmetric with respect to the robots. The values presented in Table 2 correspond to the average of ten observations for each traveling time. The variance obtained from the measurements was minimal. The values were measured with a stopwatch. Therefore, there are some experimental errors in the amount of time required for a robot to travel from a point to the other

Figure 9. General Branch and Bound Tree for Robot A

Figure 10. Branch and Bound Tree for Chip #1 Robot A

Table 2. Traveling Times for Robot A and Robot B

TRAVELING TIME

POSITION	ROBOT A						
	0	1	2	3	4	5	6
0	-	30.04	21.97	17.24	14.20	13.81	-
1	-	-	24.90	19.38	17.17	15.87	-
2	4.65	-	-	20.48	17.61	16.26	22.78
3	6.50	-	24.38	-	18.64	17.29	26.13
4	4.71	-	21.59	18.98	-	15.15	25.71
5	3.82	-	21.30	19.01	15.88	-	23.37
6	10.5	-	-	-	-	-	-

POSITION	ROBOT B						
	0	1	2	3	4	5	6
0	-	27.01	21.20	19.24	16.12	14.00	-
1	-	-	22.36	20.34	18.19	17.10	-
2	5.67	-	-	20.94	18.40	15.79	27.92
3	3.41	-	23.29	-	17.74	15.85	23.64
4	3.85	-	22.23	19.66	-	14.94	23.81
5	5.04	-	22.46	20.27	17.49	-	24.39
6	9.73	-	-	-	-	-	-

point. It was assumed that the amount of errors would not affect the results significantly.

The second method, assembling the board with two robotic arms, was divided into two approaches: one for assembling the board with sequential operations performed by the two robots, i.e., every operation should wait until the previous operation was completely done, and the other for assembling the board with a multitask execution program. This program allowed both robots to work simultaneously with the same board, just as if a worker used both hands at the same time to assemble a product. This approach integrated all the activities into one brain or a central computer.

For the first method in the second Group (Table 1), the same Branch and Bound trees that were used in the first Group will be used.

Branch and Bound trees were also developed for the second method in the second Group, (Group II-b). The activities of the two robotic arms were grouped into pairs according to the assembly sequence in the tree. Figure 11 shows a general multitasking Branch and Bound tree.

At the beginning of the cycle both robots were at their home position. In this case, the first operation by which an unassembled board was loaded to the central assembly area was fixed, but it was performed by either of the robotic arms. With this situation, the number of

Figure 11. General Multitask Branch and Bound Tree

combinations along with the first operation was increased. For example, the combination (1,2) meant that robot A placed the board in the assembly area, and robot B placed chip 1. It was assumed that the robotic arms started at the same time.

The largest traveling time within a pair of two robot operations was selected as the traveling time for the combined task to solve the Branch and Bound tree. This simplification allowed the user to find the shortest assembly time in a multitask environment.

There was a difference between the analytical results and the physical results because of the interference of the robot arms in the same assembly area. The interference occurred when one of the robotic arms approached an area, and then found the other robot occupying the same area.

During the assembly of the microchips, there were several situations when both robots approached the same area and one of them was delayed and waited until there was enough space for the other robot to perform a predetermined operation.

The traveling times shown in Table 2 also applied to this approach. The detailed Branch and Bound trees for this method are shown in the Appendix B.

Assembly Scheduling

The assembly schedules used in this research were

the shortest processing time (SPT), and the longest processing time (LPT). The objective was to determine the total assembly time generated by each method.

The sequences of operation based on either the SPT rule and the LPT rule were determined from the Branch and Bound trees.

Multitasking Program

The multitask program that was developed for this thesis did not replace the operating system on the computer because the program was an application program executed by the existing operating system (MS-DOS). This program was classified as a multitask executive system. That meant that an objective of this program was to manage all the activities or 'tasks' within a predefined assembly sequence. The multitask executive program decided the time at which a task was to be completed and prepared for the next one to be executed. Figure 12 presents the program executing sequence. The initial step was loading the programs of the assembly operations. The programs were coded by two BASIC languages: one was the ARMBASIC for the robot arms, and the other was a BASIC compiler, Turbo BASIC [19]. Each of the assembly operations were compiled in an independent executable file to call an operation on the basis of the assembly sequence.

Figure 13 presents a flowchart of the executive

Figure 12. Program Executing Sequence

Figure 13. Multitask Executive Program

program. The program consisted of four subroutines and a set of data files that contained the information for the assembly operations.

The subroutines and their descriptions are as follows:

Executive Program: This was the main routine that controls and sets the entry of the rest of the subroutines. It was coded in Turbo Basic.

Time Controller Routine: This routine kept track of the time that each assembly operation takes in order to avoid spatial conflicts over the assembly area. It was correlated with the assembly sequence that was to be performed.

Scheduler Routine: This routine was in charge of activating the task that had to be performed according to the assembly sequence. It maintained a task queue for all the assembly cycles.

Robots A & B Routines: These routines were the ones that actually moved the robot arm. Before sending the instruction to the robot arm, the data file was read for allowing the previous task to be controlled by the CPU. After the data were read, the scheduler routine waited until the CPU was available to schedule the task. Once the task took possession of the CPU it sent the information to the robot arm to perform the desired operation.

Data Files: These files contained the current readings of the stepper motors of the robot arms.

Before the simulation started, an assembly routine was selected and the sequence in the scheduler routine was specified.

Performance Criteria

The results of this research were evaluated in terms of the efficiency of the robotic assembly cell and the effectiveness of the assembly routines. The best assembly sequences were selected based on both efficiency and effectiveness.

The efficiency of a robotic cell was determined by the percentage of time utilization of the resources that were in the cell, in this case the robot arms.

The effectiveness was determined by the manufacturing (assembly) time, total cycle time and idle time on the basis of the assembly routines. These time parameters were a result of the control method used to control the cell.

CHAPTER 4

DATA COLLECTION AND ANALYSIS

Analytical Results

To evaluate the performance of the robotic cell, twelve different assembly sequences were considered in analytical methods. The assembly sequences were selected by the SPT rule, LPT rule, and the Branch and Bound algorithm. These sequences are listed in Table 3.

Table 3. Assembly Sequences

Sequence	cell configuration	rule*
1	robot A	single task SPT
2	robot A	single task LPT
3	robot A	single task B. & B.
4	robot B	single task SPT
5	robot B	single task LPT
6	robot B	single task B. & B.
7	robot A and robot B	single task SPT
8	robot A and robot B	single task LPT
9	robot A and robot B	single task B. & B.
10	robot A and robot B	multitasking SPT
11	robot A and robot B	multitasking LPT
12	robot A and robot B	multitasking B. & B.

* B. & B. = Branch and Bound

Table 4 shows the twelve assembly sequences and their total completion time determined analytically. The

Table 4. Analytical Results

ANALYTICAL RESULTS													ASSEMBLY TIME (SEC)			
ROBOT A OPERATIONS																
SEQUENCE																
SPT	0	⇒	1	⇒	5	⇒	4	⇒	3	⇒	2	⇒	6	⇒	0	138.43
LPT	0	⇒	1	⇒	2	⇒	3	⇒	4	⇒	5	⇒	6	⇒	0	143.08
B. & B.	0	⇒	1	⇒	3	⇒	5	⇒	4	⇒	2	⇒	6	⇒	0	137.46
ROBOT B OPERATIONS																
SEQUENCE																
SPT	0	⇒	1	⇒	5	⇒	4	⇒	3	⇒	2	⇒	6	⇒	0	142.20
LPT	0	⇒	1	⇒	2	⇒	3	⇒	4	⇒	5	⇒	6	⇒	0	137.11
B. & B.	0	⇒	1	⇒	2	⇒	5	⇒	4	⇒	3	⇒	6	⇒	0	135.68
ROBOT A AND B OPERATIONS																
SEQUENCE																
SPT SINGLE TASK	A ⇒ 0 ⇒ 5 ⇒ 4 ⇒ 2 ⇒ 0													136.65		
	B ⇒ 0 ⇒ 1 ⇒ 3 ⇒ 6 ⇒ 0															
LPT SINGLE TASK	A ⇒ 0 ⇒ 1 ⇒ 4 ⇒ 6 ⇒ 0													146.45		
	B ⇒ 0 ⇒ 2 ⇒ 3 ⇒ 5 ⇒ 0															
B. & B. SINGLE TASK	A ⇒ 0 ⇒ 5 ⇒ 4 ⇒ 2 ⇒ 0													136.65		
	B ⇒ 0 ⇒ 1 ⇒ 3 ⇒ 6 ⇒ 0															
SPT MULTITASKING	A ⇒ 0 ⇒ 5 ⇒ 3 ⇒ 2 ⇒ 0													80.13		
	B ⇒ 0 ⇒ 1 ⇒ 4 ⇒ 6 ⇒ 0															
LPT MULTITASKING	A ⇒ 0 ⇒ 1 ⇒ 4 ⇒ 6 ⇒ 0													87.19		
	B ⇒ 0 ⇒ 2 ⇒ 3 ⇒ 5 ⇒ 0															
B. & B. MULTITASK	A ⇒ 0 ⇒ 2 ⇒ 4 ⇒ 3 ⇒ 0													78.74		
	B ⇒ 0 ⇒ 1 ⇒ 5 ⇒ 6 ⇒ 0															

assembly sequence with the lowest completion time using robot A was the sequence obtained through the Branch and Bound algorithm. Its completion time was 137.46 seconds compared to 138.43 seconds obtained by the SPT rule. Figure 14 shows a Gantt chart of this analytical assembly sequence.

The assembly sequence with the shortest completion time using robot B was also the sequence obtained through the Branch and Bound algorithm. Its completion time was 135.68 seconds. Tables 10 through 13 shown in Appendix C present all the different sequences that were evaluated in the single robot case.

It has been demonstrated that the SPT rule outperformed the LPT rule in terms of completion time [18]. In the above solutions, the SPT rule and the LPT rules were applied under the restriction that the first and last operation (Operation 0 and Operation 6) were fixed for every sequence. In the case of robot B this restriction caused a different solution than it was expected. The LPT rule generated a shorter completion time than the SPT rule did. In this solution, Operation 1 and Operation 6 had the longest traveling but they did not follow either the SPT rule or the LPT rule. The following example shows the results of having two fixed operations that didn't follow the assembly rules (Table 5):

Figure 14. Analytical Completion Time for Robot A with the Branch and Bound Algorithm

Table 5. Performance of the Rules

Rule:		SPT		LPT	
Sequence (from-to)	completion time	sequence (from-to)	completion time		
0 - 1	27.01	0 - 1	27.01		
1 - 5	44.11	1 - 2	49.37		
5 - 4	61.60	2 - 3	70.31		
4 - 3	81.26	3 - 4	88.05		
3 - 2	104.55	4 - 5	102.99		
2 - 6	132.47	5 - 6	127.38		
6 - 0	142.20	6 - 0	137.11		

The above example shows that the LPT rule performed better than the SPT rule due to the sequence of 3-2 and 2-6 in the SPT rule, and the sequence 4-5 and 5-6 in the LPT rule. The SPT rule selected the worst sequences while the LPT was forced to select the best sequences.

The analytical computations using both robot A and robot B under sequential operations showed that the SPT rule sequence and the Branch and Bound sequence generated the shortest completion time, 136.65 seconds.

The best assembly sequence using two robot arms under the multitasking execution mode was the sequence obtained with the Branch and Bound algorithm. The completion time required for this sequence was 78.74 seconds. Figure 15 shows a Gantt chart for this sequence. Tables 14 through 21 listed in Appendix D show the Branch and Bound trees for all possible sequences and total completion time

Figure 15. Completion Time using Robot A and Robot B under Multitask Execution w/a Branch and Bound Algorithm

values.

Under evaluation of analytical completion times in the robotic cell, the multitasking execution method would be selected. The sequence that minimized the total assembly time in a multitasking execution mode was obtained through the Branch and Bound algorithm. However, the selection of a control scheme and an assembly sequence could not be made based only on the best analytical solutions because of physical conflict problems. The performance investigation of a robotic cell is difficult under analytical approaches. Analytical approaches usually give the decision maker the optimal solution. However, the solutions are difficult to visualize. The decision maker cannot be sure whether his control algorithms or his best routines are really working for the system as he expects. Therefore, a physical simulation tool has been suggested in this research. The physical simulation can show visually how the algorithm or the routine works for the system. In this research, the physical conflicts between the robot arms were investigated through physical simulation.

Physical simulation results are presented in the following section.

Physical Simulation Results

Physical simulation for a robotic assembly cell can

suggest improvements in the design of manufacturing systems. A physical simulation model shows visually the spatial relationships between machines, transfer mechanisms, and material handling equipment. A critical element in the entire design of a robotic cell is the control method that will be used to operate the cell. The physical simulation performed in this research evaluated two independent control methods with different assembly sequences identified by analytical methods: a single task control method, and a multitask control method.

All the operations executed by the robot arms were recorded prior to the performance test. The program source codes are listed in Appendix E. Program "ULL" was developed to record the movements for robot A. Program "ULL1" was developed to record robot B's movements.

Two programs were developed to evaluate the single task control method. Program "DLL" controlled the movements for robot A, and program "DLL1" controlled the movements for robot B. These programs basically contained the number of steps required for moving the stepper motors attached to the robot. Each axis of this robot would be moved to the specified locations of the cell as many as the number of steps defined for the axis. Furthermore, these programs were executable files and were activated by inputting the data file name.

Program "TESIN1" was developed for the multitask

control method. The program source code is listed in Appendix F. To execute this program, the data file names for robot A's movements and robot B's movements had to be coded in the program. The scheduler routine in the multitask executive file decided what specific sequence had to be tested in that time moment. Therefore, this program was continuously updated with the next higher prioritized sequence. This means that the scheduler knew which sequence had the highest priority to be performed.

Table 6 presents the results obtained from the physical simulation. The first part of the table shows the single task-single robot execution mode. The second part shows the single task-double robot execution mode and finally, the multitask double robot execution mode.

In using robot A under a single task execution mode, the best sequence was determined by the Branch and Bound solution. The completion time was 150.04 seconds.

In using robot B, under a single task execution mode, the best sequence was also determined by the Branch and Bound solution. The completion time was 148.82 seconds. No physical conflicts occurred during the above experiments (robot A or robot B).

The performance of both robots under a single execution mode was worse than the above two experiments. The sequence with the shortest completion time was found

Table 6. Physical Simulation Results

PHYSICAL SIMULATION RESULTS													ASSEMBLY TIME (SEC.)			
SEQUENCE		ROBOT A OPERATIONS														
SPT	0	⇨	1	⇨	5	⇨	4	⇨	3	⇨	2	⇨	6	⇨	0	154.09
LPT	0	⇨	1	⇨	2	⇨	3	⇨	4	⇨	5	⇨	6	⇨	0	160.15
B & B	0	⇨	1	⇨	3	⇨	5	⇨	4	⇨	2	⇨	6	⇨	0	150.04
SEQUENCE		ROBOT B OPERATIONS														
SPT	0	⇨	1	⇨	5	⇨	4	⇨	3	⇨	2	⇨	6	⇨	0	158.96
LPT	0	⇨	1	⇨	2	⇨	3	⇨	4	⇨	5	⇨	6	⇨	0	154.00
B & B	0	⇨	1	⇨	2	⇨	5	⇨	4	⇨	3	⇨	6	⇨	0	148.82
SEQUENCE		ROBOT A AND B OPERATIONS														
SPT SINGLE TASK	A	⇨	0	⇨	5	⇨	4	⇨	2	⇨	0					157.96
	B	⇨	0	⇨	1	⇨	3	⇨	6	⇨	0					
LPT SINGLE TASK	A	⇨	0	⇨	1	⇨	4	⇨	6	⇨	0					169.29
	B	⇨	0	⇨	2	⇨	3	⇨	5	⇨	0					
B. & B. SINGLE TASK	A	⇨	0	⇨	5	⇨	4	⇨	2	⇨	0					157.96
	B	⇨	0	⇨	1	⇨	3	⇨	6	⇨	0					
SPT MULTITASKING	A	⇨	0	⇨	5	⇨	3	⇨	2	⇨	0					131.59
	B	⇨	0	⇨	1	⇨	4	⇨	6	⇨	0					
LPT MULTITASKING	A	⇨	0	⇨	1	⇨	4	⇨	6	⇨	0					123.73
	B	⇨	0	⇨	2	⇨	3	⇨	5	⇨	0					
B. & B. MULTITASKING	A	⇨	0	⇨	2	⇨	4	⇨	3	⇨	0					127.00
	B	⇨	0	⇨	1	⇨	5	⇨	6	⇨	0					

by the SPT rule and the Branch and Bound algorithm. The completion time for this sequence was 157.96 seconds, 9.14 seconds more than in the single robot execution (robot B).

The multitask execution mode with two robot arms showed the best results compared to all the single task modes. The sequence with the shortest completion time was determined by the LPT rule. The completion time for this sequence was measured as 123.73 seconds.

As described in the analytical results, the LPT rule was outperformed by the SPT rule because of the lack of good choices for the chips as the assembling tasks proceeded.

From the above results, it was concluded that the Branch and Bound algorithm performed best in a single task control method. However, the same solution was not recommended for the multitask control method. Therefore, in-depth analysis of the multitask performance was followed.

In the multitask control method, the physical layout of the cell played an important role in selecting the best assembly sequence. Figure 16 shows the physical paths followed by robot A and robot B under the Branch and Bound solution. Robot A traveled the longest distance in Operation 3 to place chip 2. This operation left the assembly area occupied by robot A seriously delaying robot B. This means that a physical conflict should be removed

Figure 16. Physical Paths followed by Robot A and Robot B under the Branch and Bound solution

to reduce the delay times forced on robot B by robot A. Similarly, the execution of Operation 4 by robot A and Operation 5 by robot B were in physical conflict. As a matter of fact, all those intersections between two paths drawn by both robots were the physical conflicts in Figure 16. Table 7 shows the number of physical conflicts experienced by various sequences.

Table 7. Number of Physical Conflicts

Sequence	Control Method	No. of physical conflicts	% of delay w/respect to analytical sol.
SPT	single task	0	10.10
LPT	single task	0	10.65
B. & B.	single task	0	8.38
SPT	multitask	3	39.10
LPT	multitask	3	29.53
B. & B.	multitask	3	38.00

The LPT rule under the multitask control method performed better than the Branch and Bound solution, but there was still physical conflict as shown in Figure 17. Therefore, a better solution was sought in this thesis. The new solution should minimize the physical interference of the robotic arms under the multitask execution method. The first improvement was to place an unassembled board with one robot and remove the finished board with the other robot. Each robot should place two microchips among four microchips, respectively. According to the circuit board

Figure 17. Physical Paths followed by Robot A and Robot B under the LPT Rule solution

configuration, the two chips that were located closer to robot A were chip 1 and chip 4. They should be assembled by robot A. The same approach was applied for robot B, that is, robot B should select chip 2 and chip 3 to complete an assembly. The results of this combination of operations are shown in Figure 18. This sequence was called the adjusted sequence. Under this sequence, the number of physical conflicts was minimized. The completion time for this sequence was measured at 116 seconds. Still, this value was relatively smaller than those for other sequences, with the minimum number of physical conflicts.

The performance evaluation parameter values such as the completion time, mean flow time for measuring the effectiveness and the system utilization for measuring the efficiency, are summarized for each sequence in Table 6. They were calculated by using the following equations:

Completion Time

$$M_s = \sum_{i=1}^n t_i + d_i$$

where

M_s = the completion time for the n tasks in a sequence S

t_i = the processing time of task i (i = 1, 2, ..., n)

d_i = the idling time required for task i

Figure 18. Physical Paths followed by Robot A and Robot B, under an adjusted sequence

Table 8. Analytical Results vs. Physical Simulation Results

ANALYTICAL RESULTS

SEQUENCE	ROBOT	COMPLETION TIME	MEAN FLOW TIME	SYSTEM UTILIZATION
B. & B. SINGLE TASK	A	137.46	85.33	100.00
B. & B. SINGLE TASK	B	137.11	84.01	100.00
B. & B. SINGLE TASK	A & B	136.65	84.11	50.00
B. & B. MULTITASK	A & B	78.74	27.42	90.98
SPT MULTITASK	A & B	80.13	27.94	87.72
LPT MULTITASK	A & B	87.19	30.61	83.98

PHYSICAL SIMULATION RESULTS

SEQUENCE	ROBOT	COMPLETION TIME SEC.	MEAN FLOW TIME SEC.	SYSTEM UTILIZATION %
B. & B. SINGLE TASK	A	150.04	85.33	100.00
B. & B. SINGLE TASK	B	148.82	84.01	100.00
B. & B. SINGLE TASK	A & B	157.96	84.11	50.00
B. & B. MULTITASK	A & B	127.00	49.78	59.43
SPT MULTITASK	A & B	131.59	50.64	57.72
LPT MULTITASK	A & B	123.73	47.09	61.64
ADJUSTED MULTITASK	A & B	116.00	44.83	67.25

Mean Flow Time

$$F_s = \frac{1}{n} \sum_{i=1}^n F_{i,s}$$

where

$F_{i,s}$ = flow time for task i in a sequence S
 $(i = 1, 2, \dots, n)$

System Utilization

$$U = \frac{M - I}{M} * 100 \text{ (per robot)}$$

where

M = completion time

I = total idle time

Both the analytical results and the physical simulation results showed that the multitask execution mode was better than the single task execution mode in terms of completion time. As shown in Table 8, the difference of the completion time between two sequences is small, but considering that a robotic facility would be running twenty hours a day, six days a week, this difference could be very important.

In the physical simulation, the results showed that a Branch and Bound solution with a single robot under a single task execution mode was the best choice because it generated the shortest completion time. Based on these results, it was concluded that the best analytical method

could be directly applied for a single robot under a single task execution control method.

As stated before, the multitask control method was better in both the analytical method and the physical simulation. But an analytical solution was not recommended to seek the best assembly sequence when using two or more robots under a multitasking execution control method because of physical conflicts. Even with a digital simulation tool, it would be very difficult to detect any physical conflict or to avoid collisions between two robot arms. Without a physical simulation it would be difficult to find a better assembly sequence such as the adjusted sequence that improved the overall system utilization by 5.6 % and reduced the completion time by 8.6 % referred to in Table 8.

In summary, the above discussions are presented in Table 9.

Table 9. Summary of the Best Sequences.

High Efficiency o high utilization	High Effectiveness o shorter completion time o shorter mean flow time o shorter idle time
B. & B. single task (robot A or robot B)	Adjusted sequence multitask (robot A and robot B)
Note: No physical conflicts	Note: Adjusted to minimize the number of physical conflicts

CHAPTER 5

SUMMARY AND CONCLUSIONS

An evaluation of a variety of assembly routines under two software control methods, a single task execution program and a multitask execution program, was presented in this thesis. Three analytical methods were used to select the assembly sequences, the shortest processing rule (SPT), the longest processing rule (LPT), and the Branch and Bound algorithm. The results obtained from this research were the basis for the following conclusions:

In designing and implementing an assembly robotic cell, two factors should be taken into account. First, the parts to be assembled must be designed for improving the assemblability in a robotic assembly process. The number of components should be minimum and arranged in such a way that the number of robot movements should be also minimized. Second, the cell must be configured to minimize the number of physical conflicts or collisions. In controlling a robotic cell, the control schemes should be effective and powerful. The software control schemes should be selected according to the part types, cell configuration, and production quantity. When a single robot

was utilized, the recommended control scheme was the single task execution method. This method obviously resulted in 100 % utilization of the robotic cell. Furthermore, the assembly sequence should be defined by a Branch and Bound algorithm which generates the shortest completion time.

For the control of a complex robotic cell that uses peripheral devices and more than one robot arm, a multitask control method is strongly recommended. The results obtained by this control method showed the best performance for the cell. When an assembly sequence was slightly modified to minimize the amount of physical conflict with the consideration of the cell layout, the system utilization was 67 %, and the completion time to assemble a part was reduced 22 % in comparison to the single case. The cell modification to minimize the number of physical conflicts was described in Chapter 4. An analytical method to define the best assembly sequence in terms of efficiency and effectiveness was not recommended because the physical conflicts could not be observed visually. This is obviously true when the multitask execution method is used for multiple number of robot arms.

Production managers usually have conflicts in deciding between efficiency and effectiveness to introduce the best control policies into their systems. For the robotic assembly cell investigated in this research, the completion

time, mean flow time, and scheduling rules were major factors in determining the effectiveness. The system utilization was a factor for evaluating the efficiency. As seen in Table 5, a sequence generated either a higher utilization value with longer completion time value or a lower utilization value with a shorter completion time. The manager would have a choice between higher effectiveness (lower system utilization) and lower effectiveness (higher system utilization). There are few algorithms or methods to achieve both higher effectiveness and higher efficiency simultaneously.

REFERENCES CITED

1. Backer, T. P. , and G. M. Scallion, "An Architecture for Real-Time Software Systems", IEEE Software, May 1986.
2. Drezner, Z., and S. Y. Nof, "On Optimizing Bin Picking and Insertion Plans for Assemble Robots", IIE Transactions, Volume 16, No. 3.
3. Crowl, D. A., "A Real-Time Fortran Executive", IEEE MICRO, August 1985.
4. Fathi, E. T., E. Bosse and J. Caseault, "A Distributed System for Real-Time Applications", IEEE MICRO, December 1987.
5. Hosseini, J. and N. Fard, "Analysis and Design of a Real-Time Manufacturing Subsystem", Proceedings of the 8th Annual Conference of Computers and Industrial Engineering.
6. Muth, E. J., "Using a Personal Computer as a Real-Time Multitasking Controller", Proceedings of the 8th Annual Conference of Computers and Industrial Engineering.
7. Paidy, S. and R. Reeve, "The Use and Developement of Physical Simulator", Annual Simulation Symposium.
8. Tavora, C. J., "A Basic Technique for Real-Time System Design", Computer Design, October 1980.
9. Wainwright, R. E., "A Microcomputer-based Model Robot System with Pulse-Width Modulation Control", IEEE MICRO, February 1985.
10. Owen, T., "Assembly with Robots", Prentice-Hall Inc., 1985
11. Groover, M.P., M. Weiss, "Industrial Robotics", Mc Graw-Hill, 1986.
12. Boothroyd, G., "Economics of Assembly Systems", Journal of Manufacturing Systems 1, 1982.

13. Anderson, D.R., D.I. Sweeney, and T.A. Williams, "An Introduction to Management Science", 3rd Edition, West Publishing Company, St. Paul, Minn., 1982.

14. White, J.W., "Utility and Systems Software", Real Time Computing, Van Nostrand Reinhold Company, 1983.

15. Microbot Inc., "TeachMover User Manual", Microbot Inc., 1984.

16. Conway, R. W., W. L. Maxwell and L. W. Miller, "Theory of Scheduling", Addison-Wesley Publishing Company, 1967.

17. Savitzky, S. R., "Real-Time Microprocessor Systems", Van Nostrand Reinhold Company, 1985.

18. Bedworth, D. D., J. E. Bailey, "Integrated Production Control Systems", 1982.

19. Borland International, Inc., "Turbo BASIC", Borland International, Inc., 1987.

20. Microsoft Corporation, "Microsoft Macro Assembler 5.0", Microsoft Corporation, 1987.

21. Groover, M. P., "Automation Production Systems, and Computer Integrated Manufacturing", Prentice Hall, 1987.

APPENDICES

APPENDIX A

Branch and Bound Trees Single Case

Figure 19. Branch and Bound Tree for Chip #2 Robot A

Figure 20. Branch and Bound Tree for Chip #3 Robot A

Figure 21. Branch and Bound Tree for Chip #4 Robot A

Figure 22. General Branch and Bound Tree for Robot B

APPENDIX B

Branch and Bound Trees Multitasking

Figure 23. Multitask B. and B. Tree for pair 1,2

Figure 24. Multitask B. and B. Tree for pair 1,3

Figure 25. Multitask B. and B. Tree for pair 1,4

Figure 26. Multitask B. and B. Tree for pair 1,5

Figure 27. Multitask B. and B. Tree for pair 2,1

Figure 28. Multitask B. and B. Tree for pair 3,1

Figure 29. Multitask B. and B. Tree for pair 4,1

Figure 30. Multitask B. and B. Tree for pair 5,1

APPENDIX C

Completion Time for Single Robot Trees

Table 10. Completion Time for Branches 1 and 2 - Robot B

SEQUENCE	ASSEMBLING TIME
0 ⇨ 1 ⇨ 2 ⇨ 3 ⇨ 4 ⇨ 5 ⇨ 6 ⇨ 0	137.11
0 ⇨ 1 ⇨ 2 ⇨ 3 ⇨ 5 ⇨ 4 ⇨ 6 ⇨ 0	137.32
0 ⇨ 1 ⇨ 2 ⇨ 4 ⇨ 3 ⇨ 5 ⇨ 6 ⇨ 0	137.40
0 ⇨ 1 ⇨ 2 ⇨ 4 ⇨ 5 ⇨ 3 ⇨ 6 ⇨ 0	136.35
0 ⇨ 1 ⇨ 2 ⇨ 5 ⇨ 3 ⇨ 4 ⇨ 6 ⇨ 0	136.10
0 ⇨ 1 ⇨ 2 ⇨ 5 ⇨ 4 ⇨ 3 ⇨ 6 ⇨ 0	135.68
0 ⇨ 1 ⇨ 3 ⇨ 2 ⇨ 4 ⇨ 5 ⇨ 6 ⇨ 0	138.10
0 ⇨ 1 ⇨ 3 ⇨ 2 ⇨ 5 ⇨ 4 ⇨ 6 ⇨ 0	144.13
0 ⇨ 1 ⇨ 3 ⇨ 4 ⇨ 2 ⇨ 5 ⇨ 6 ⇨ 0	137.23
0 ⇨ 1 ⇨ 3 ⇨ 4 ⇨ 5 ⇨ 2 ⇨ 6 ⇨ 0	140.18
0 ⇨ 1 ⇨ 3 ⇨ 5 ⇨ 2 ⇨ 4 ⇨ 6 ⇨ 0	137.60
0 ⇨ 1 ⇨ 3 ⇨ 5 ⇨ 4 ⇨ 2 ⇨ 6 ⇨ 0	140.57

Table 11. Completion Time for Branches 3 and 4 - Robot B

SEQUENCE	ASSEMBLING TIME
0 ⇨ 1 ⇨ 4 ⇨ 2 ⇨ 3 ⇨ 5 ⇨ 6 ⇨ 0	138.34
0 ⇨ 1 ⇨ 4 ⇨ 2 ⇨ 5 ⇨ 3 ⇨ 6 ⇨ 0	136.86
0 ⇨ 1 ⇨ 4 ⇨ 3 ⇨ 2 ⇨ 5 ⇨ 6 ⇨ 0	138.06
0 ⇨ 1 ⇨ 4 ⇨ 3 ⇨ 5 ⇨ 2 ⇨ 6 ⇨ 0	140.82
0 ⇨ 1 ⇨ 4 ⇨ 5 ⇨ 2 ⇨ 3 ⇨ 6 ⇨ 0	136.91
0 ⇨ 1 ⇨ 4 ⇨ 5 ⇨ 3 ⇨ 2 ⇨ 6 ⇨ 0	141.35
0 ⇨ 1 ⇨ 5 ⇨ 2 ⇨ 3 ⇨ 4 ⇨ 6 ⇨ 0	138.79
0 ⇨ 1 ⇨ 5 ⇨ 2 ⇨ 4 ⇨ 3 ⇨ 6 ⇨ 0	138.00
0 ⇨ 1 ⇨ 5 ⇨ 3 ⇨ 2 ⇨ 4 ⇨ 6 ⇨ 0	139.61
0 ⇨ 1 ⇨ 5 ⇨ 3 ⇨ 4 ⇨ 2 ⇨ 6 ⇨ 0	142.00
0 ⇨ 1 ⇨ 5 ⇨ 4 ⇨ 2 ⇨ 3 ⇨ 6 ⇨ 0	138.14
0 ⇨ 1 ⇨ 5 ⇨ 4 ⇨ 3 ⇨ 2 ⇨ 6 ⇨ 0	142.2

Table 12. Completion Time for Branches 1 and 2 - Robot A

SEQUENCE	ASSEMBLING TIME
0 → 1 → 2 → 3 → 4 → 5 → 6 → 0	144.80
0 → 1 → 2 → 3 → 5 → 4 → 6 → 0	143.08
0 → 1 → 2 → 4 → 3 → 5 → 6 → 0	142.69
0 → 1 → 2 → 4 → 5 → 3 → 6 → 0	143.34
0 → 1 → 2 → 5 → 3 → 4 → 6 → 0	145.06
0 → 1 → 2 → 5 → 4 → 3 → 6 → 0	142.69
0 → 1 → 3 → 2 → 4 → 5 → 6 → 0	140.43
0 → 1 → 3 → 2 → 5 → 4 → 6 → 0	141.15
0 → 1 → 3 → 4 → 2 → 5 → 6 → 0	139.78
0 → 1 → 3 → 4 → 5 → 2 → 6 → 0	137.79
0 → 1 → 3 → 5 → 2 → 4 → 6 → 0	141.83
0 → 1 → 3 → 5 → 4 → 2 → 6 → 0	137.46

Table 13. Completion Time for Branches 3 and 4 - Robot A

SEQUENCE	ASSEMBLING TIME
0 → 1 → 4 → 2 → 3 → 5 → 6 → 0	140.44
0 → 1 → 4 → 2 → 5 → 3 → 6 → 0	140.70
0 → 1 → 4 → 3 → 2 → 5 → 6 → 0	140.70
0 → 1 → 4 → 3 → 5 → 2 → 6 → 0	138.06
0 → 1 → 4 → 5 → 2 → 3 → 6 → 0	140.77
0 → 1 → 4 → 5 → 3 → 2 → 6 → 0	139.03
0 → 1 → 5 → 2 → 3 → 4 → 6 → 0	142.54
0 → 1 → 5 → 2 → 4 → 3 → 6 → 0	140.43
0 → 1 → 5 → 3 → 2 → 4 → 6 → 0	143.12
0 → 1 → 5 → 3 → 4 → 2 → 6 → 0	138.43
0 → 1 → 5 → 4 → 2 → 3 → 6 → 0	140.49
0 → 1 → 5 → 4 → 3 → 2 → 6 → 0	138.43

APPENDIX D

Completion Time for Double Robot Trees

Table 14. Completion Time for Pair (1,2) Multitasking

SEQUENCE							ASSEMBLING TIME
(0,0)	⇒	(1,2)	⇒	(3,4)	⇒	(5,6)	⇒ (0,0) 82.96
(0,0)	⇒	(1,2)	⇒	(3,4)	⇒	(6,5)	⇒ (0,0) 86.05
(0,0)	⇒	(1,2)	⇒	(3,5)	⇒	(4,6)	⇒ (0,0) 83.54
(0,0)	⇒	(1,2)	⇒	(3,5)	⇒	(6,4)	⇒ (0,0) 86.05
(0,0)	⇒	(1,2)	⇒	(4,3)	⇒	(5,6)	⇒ (0,0) 84.35
(0,0)	⇒	(1,2)	⇒	(4,3)	⇒	(6,5)	⇒ (0,0) 87.19
(0,0)	⇒	(1,2)	⇒	(4,5)	⇒	(3,6)	⇒ (0,0) 81.33
(0,0)	⇒	(1,2)	⇒	(4,5)	⇒	(6,3)	⇒ (0,0) 83.42
(0,0)	⇒	(1,2)	⇒	(5,3)	⇒	(4,6)	⇒ (0,0) 84.35
(0,0)	⇒	(1,2)	⇒	(5,3)	⇒	(6,4)	⇒ (0,0) 84.85
(0,0)	⇒	(1,2)	⇒	(5,4)	⇒	(3,6)	⇒ (0,0) 81.98
(0,0)	⇒	(1,2)	⇒	(5,4)	⇒	(6,3)	⇒ (0,0) 82.31

Table 15. Completion Time for Pair (1,3) Multitasking

SEQUENCE							ASSEMBLING TIME
(0,0)	⇨	(1,3)	⇨	(2,4)	⇨	(5,6)	⇨ (0,0) 88.48
(0,0)	⇨	(1,3)	⇨	(2,4)	⇨	(6,5)	⇨ (0,0) 82.22
(0,0)	⇨	(1,3)	⇨	(2,5)	⇨	(4,6)	⇨ (0,0) 89.06
(0,0)	⇨	(1,3)	⇨	(2,5)	⇨	(6,4)	⇨ (0,0) 88.22
(0,0)	⇨	(1,3)	⇨	(4,2)	⇨	(5,6)	⇨ (0,0) 90.98
(0,0)	⇨	(1,3)	⇨	(4,2)	⇨	(6,5)	⇨ (0,0) 89.54
(0,0)	⇨	(1,3)	⇨	(4,5)	⇨	(2,6)	⇨ (0,0) 81.33
(0,0)	⇨	(1,3)	⇨	(4,5)	⇨	(6,2)	⇨ (0,0) 83.42
(0,0)	⇨	(1,3)	⇨	(5,2)	⇨	(4,6)	⇨ (0,0) 90.98
(0,0)	⇨	(1,3)	⇨	(5,2)	⇨	(6,4)	⇨ (0,0) 87.20
(0,0)	⇨	(1,3)	⇨	(5,4)	⇨	(2,6)	⇨ (0,0) 81.32
(0,0)	⇨	(1,3)	⇨	(5,4)	⇨	(6,2)	⇨ (0,0) 81.65

Table 16. Completion Time for Pair (1,4) Multitasking

SEQUENCE							ASSEMBLING TIME
(0,0)	⇒	(1,4)	⇒	(2,3)	⇒	(5,6)	⇒ (0,0) 88.31
(0,0)	⇒	(1,4)	⇒	(2,3)	⇒	(6,5)	⇒ (0,0) 88.22
(0,0)	⇒	(1,4)	⇒	(2,5)	⇒	(3,6)	⇒ (0,0) 89.06
(0,0)	⇒	(1,4)	⇒	(2,5)	⇒	(6,3)	⇒ (0,0) 88.22
(0,0)	⇒	(1,4)	⇒	(3,2)	⇒	(5,6)	⇒ (0,0) 89.92
(0,0)	⇒	(1,4)	⇒	(3,2)	⇒	(6,5)	⇒ (0,0) 88.90
(0,0)	⇒	(1,4)	⇒	(3,5)	⇒	(2,6)	⇒ (0,0) 83.54
(0,0)	⇒	(1,4)	⇒	(3,5)	⇒	(6,2)	⇒ (0,0) 86.05
(0,0)	⇒	(1,4)	⇒	(5,2)	⇒	(3,6)	⇒ (0,0) 89.92
(0,0)	⇒	(1,4)	⇒	(5,2)	⇒	(6,3)	⇒ (0,0) 86.14
(0,0)	⇒	(1,4)	⇒	(5,3)	⇒	(2,6)	⇒ (0,0) 83.07
(0,0)	⇒	(1,4)	⇒	(5,3)	⇒	(6,2)	⇒ (0,0) 83.57

Table 17. Completion Time for Pair (1,5) Multitasking

SEQUENCE								ASSEMBLING TIME	
(0,0)	⇒	(1,5)	⇒	(2,3)	⇒	(4,6)	⇒	(0,0)	88.31
(0,0)	⇒	(1,5)	⇒	(2,3)	⇒	(6,4)	⇒	(0,0)	88.22
(0,0)	⇒	(1,5)	⇒	(2,4)	⇒	(3,6)	⇒	(0,0)	88.48
(0,0)	⇒	(1,5)	⇒	(2,4)	⇒	(6,3)	⇒	(0,0)	88.22
(0,0)	⇒	(1,5)	⇒	(3,2)	⇒	(4,6)	⇒	(0,0)	90.15
(0,0)	⇒	(1,5)	⇒	(3,2)	⇒	(6,4)	⇒	(0,0)	89.13
(0,0)	⇒	(1,5)	⇒	(3,4)	⇒	(2,6)	⇒	(0,0)	83.53
(0,0)	⇒	(1,5)	⇒	(3,4)	⇒	(6,2)	⇒	(0,0)	86.05
(0,0)	⇒	(1,5)	⇒	(4,2)	⇒	(3,6)	⇒	(0,0)	90.15
(0,0)	⇒	(1,5)	⇒	(4,2)	⇒	(6,3)	⇒	(0,0)	88.70
(0,0)	⇒	(1,5)	⇒	(4,3)	⇒	(2,6)	⇒	(0,0)	83.68
(0,0)	⇒	(1,5)	⇒	(4,3)	⇒	(6,2)	⇒	(0,0)	86.94

Table 18. Completion Time for Pair (2,1) Multitasking

SEQUENCE								ASSEMBLING TIME	
(0,0)	⇒	(2,1)	⇒	(3,4)	⇒	(5,6)	⇒	(0,0)	81.03
(0,0)	⇒	(2,1)	⇒	(3,4)	⇒	(6,5)	⇒	(0,0)	84.12
(0,0)	⇒	(2,1)	⇒	(3,5)	⇒	(4,6)	⇒	(0,0)	81.01
(0,0)	⇒	(2,1)	⇒	(3,5)	⇒	(6,4)	⇒	(0,0)	84.12
(0,0)	⇒	(2,1)	⇒	(4,3)	⇒	(5,6)	⇒	(0,0)	80.72
(0,0)	⇒	(2,1)	⇒	(4,3)	⇒	(6,5)	⇒	(0,0)	83.56
(0,0)	⇒	(2,1)	⇒	(4,5)	⇒	(3,6)	⇒	(0,0)	78.74
(0,0)	⇒	(2,1)	⇒	(4,5)	⇒	(6,3)	⇒	(0,0)	80.83
(0,0)	⇒	(2,1)	⇒	(5,3)	⇒	(4,6)	⇒	(0,0)	80.72
(0,0)	⇒	(2,1)	⇒	(5,3)	⇒	(6,4)	⇒	(0,0)	81.22
(0,0)	⇒	(2,1)	⇒	(5,4)	⇒	(3,6)	⇒	(0,0)	78.74
(0,0)	⇒	(2,1)	⇒	(5,4)	⇒	(6,3)	⇒	(0,0)	79.07

Table 19. Completion Time for Pair (3,1) Multitasking

SEQUENCE							ASSEMBLING TIME
(0,0)	⇨	(3,1)	⇨	(2,4)	⇨	(5,6)	⇨ (0,0) 84.93
(0,0)	⇨	(3,1)	⇨	(2,4)	⇨	(6,5)	⇨ (0,0) 84.67
(0,0)	⇨	(3,1)	⇨	(2,5)	⇨	(4,6)	⇨ (0,0) 85.51
(0,0)	⇨	(3,1)	⇨	(2,5)	⇨	(6,4)	⇨ (0,0) 84.67
(0,0)	⇨	(3,1)	⇨	(4,2)	⇨	(5,6)	⇨ (0,0) 87.02
(0,0)	⇨	(3,1)	⇨	(4,2)	⇨	(6,5)	⇨ (0,0) 85.58
(0,0)	⇨	(3,1)	⇨	(4,5)	⇨	(2,6)	⇨ (0,0) 79.77
(0,0)	⇨	(3,1)	⇨	(4,5)	⇨	(6,2)	⇨ (0,0) 81.86
(0,0)	⇨	(3,1)	⇨	(5,2)	⇨	(4,6)	⇨ (0,0) 87.02
(0,0)	⇨	(3,1)	⇨	(5,2)	⇨	(6,4)	⇨ (0,0) 83.24
(0,0)	⇨	(3,1)	⇨	(5,4)	⇨	(2,6)	⇨ (0,0) 78.74
(0,0)	⇨	(3,1)	⇨	(5,4)	⇨	(6,2)	⇨ (0,0) 79.07

Table 20. Completion Time for Pair (4,1) Multitasking

SEQUENCE								ASSEMBLING TIME	
(0,0)	⇨	(4,1)	⇨	(2,3)	⇨	(5,6)	⇨	(0,0)	81.97
(0,0)	⇨	(4,1)	⇨	(2,3)	⇨	(6,5)	⇨	(0,0)	80.63
(0,0)	⇨	(4,1)	⇨	(2,5)	⇨	(3,6)	⇨	(0,0)	82.67
(0,0)	⇨	(4,1)	⇨	(2,5)	⇨	(6,3)	⇨	(0,0)	81.88
(0,0)	⇨	(4,1)	⇨	(3,2)	⇨	(5,6)	⇨	(0,0)	87.02
(0,0)	⇨	(4,1)	⇨	(3,2)	⇨	(6,5)	⇨	(0,0)	86.00
(0,0)	⇨	(4,1)	⇨	(3,5)	⇨	(2,6)	⇨	(0,0)	80.11
(0,0)	⇨	(4,1)	⇨	(3,5)	⇨	(6,2)	⇨	(0,0)	82.62
(0,0)	⇨	(4,1)	⇨	(5,2)	⇨	(3,6)	⇨	(0,0)	87.02
(0,0)	⇨	(4,1)	⇨	(5,2)	⇨	(6,3)	⇨	(0,0)	83.24
(0,0)	⇨	(4,1)	⇨	(5,3)	⇨	(2,6)	⇨	(0,0)	80.72
(0,0)	⇨	(4,1)	⇨	(5,3)	⇨	(6,2)	⇨	(0,0)	81.22

Table 21. Completion Time for Pair (5,1) Multitasking

SEQUENCE								ASSEMBLING TIME	
(0,0)	⇒	(5,1)	⇒	(2,3)	⇒	(4,6)	⇒	(0,0)	81.68
(0,0)	⇒	(5,1)	⇒	(2,3)	⇒	(6,4)	⇒	(0,0)	81.59
(0,0)	⇒	(5,1)	⇒	(2,4)	⇒	(3,6)	⇒	(0,0)	81.85
(0,0)	⇒	(5,1)	⇒	(2,4)	⇒	(6,3)	⇒	(0,0)	81.54
(0,0)	⇒	(5,1)	⇒	(3,2)	⇒	(4,6)	⇒	(0,0)	87.02
(0,0)	⇒	(5,1)	⇒	(3,2)	⇒	(6,4)	⇒	(0,0)	86.00
(0,0)	⇒	(5,1)	⇒	(3,4)	⇒	(2,6)	⇒	(0,0)	80.13
(0,0)	⇒	(5,1)	⇒	(3,4)	⇒	(6,2)	⇒	(0,0)	82.65
(0,0)	⇒	(5,1)	⇒	(4,2)	⇒	(3,6)	⇒	(0,0)	87.02
(0,0)	⇒	(5,1)	⇒	(4,2)	⇒	(6,3)	⇒	(0,0)	85.58
(0,0)	⇒	(5,1)	⇒	(4,3)	⇒	(2,6)	⇒	(0,0)	80.72
(0,0)	⇒	(5,1)	⇒	(4,3)	⇒	(6,2)	⇒	(0,0)	83.98

APPENDIX E

Source Code for Programs "ULL", "U111", "DLL", "DLL1"

Figure 31. Source code for program "ULL"

```

100 COLOR 15,1,1 'ULL.BAS
110 DEFINT A-Z
120 DIM L(7)
130 '
140 OPEN "COM1:9600,N,8,1,RS,DS,CS" AS #1
150 CLS:INPUT"INPUT FILE";A$
160 OPEN A$ FOR OUTPUT AS #2
170 CLS:LOCATE 10,20
180 PRINT "CLEAR ROBOT MEMORY BY PRESSING 'CLEAR' & 'ZERO' "
190 LOCATE 14,20
200 INPUT "ENTER # OF MOVES RECORDED WHEN DONE (0 > ANS > 53)? ", NUM.MOVES
205 PRINT#1,"@RESET":INPUT#1,I
210 PRINT #1,"@QDUMP" : INPUT #1, SHAKE
220 '
230 FOR LINUM = 0 TO 52
240   FOR Y = 0 TO 7
250     INPUT #1, L(Y)
260     IF (Y = 0) THEN L(Y) = L(Y)+(TOT.MOVES)
270     IF (LINUM < NUM.MOVES) THEN WRITE #2, L(Y) :LAST.MOV = L(0)
280   NEXT Y
290 NEXT LINUM
300 '
310 WRITE #2, LAST.MOV+1
320 WRITE #2, 2307
330 WRITE #2, LAST.MOV+1
340 FOR I = 1 TO 5
350   WRITE #2, 0
360 NEXT I
370 PRINT"THE END"
380 END

```

Figure 32. Source code for program "ULL1"

```

100 COLOR 15,1,1 'ULL.BAS
110 DEFINT A-Z
120 DIM L(7)
130 '
140 OPEN "COM2:9600,N,8,1,RS,DS,CS" AS #1
150 CLS:INPUT"INPUT FILE";A$
160 OPEN A$ FOR OUTPUT AS #2
170 CLS:LOCATE 10,20
180 PRINT "CLEAR ROBOT MEMORY BY PRESSING 'CLEAR' & 'ZERO' "
190 LOCATE 14,20
200 INPUT "ENTER # OF MOVES RECORDED WHEN DONE (0 > ANS > 53)? ", NUM.MOVES
210 PRINT #1,"@QDUMP" : INPUT #1, SHAKE
220 '
230 FOR LINUM = 0 TO 52
240   FOR Y = 0 TO 7
250     INPUT #1, L(Y)
260     IF (Y = 0) THEN L(Y) = L(Y)+(TOT.MOVES)
270     IF (LINUM < NUM.MOVES) THEN WRITE #2, L(Y) : LAST.MOV = L(0)
280   NEXT Y
290 NEXT LINUM
300 '
310 WRITE #2, LAST.MOV+1
320 WRITE #2, 2307
330 WRITE #2, LAST.MOV+1
340 FOR I = 1 TO 5
350   WRITE #2, 0
360 NEXT I
370 PRINT"THE END"
380 END

```


Figure 33. Source code for program "DLL"

```
10 COLOR 15,1,1
20 DEFINT A-Z
30 DIM L(7)
50 '
60 OPEN "COM1:9600,N,8,1,RS,DS,CS" AS #1
70 '
80 '----- Up-loading from robot to files:
90 PRINT #1,"@RESET":INPUT#1,I
100 CLS
150 INPUT "FILE";F$
340 OPEN F$ FOR INPUT AS #2
350 WHILE (EOF(2)=0)
360 INPUT #2 ,L1,L2,L3,L4,L5,L6,L7,L8
370 PRINT#1,"@QWRITE",L1,L2,L3,L4,L5,L6,L7,L8:INPUT#1,I
380 WEND
390 '
400 ' >>>>>> EXECUTION ROUTINE <<<<<<<<<<<<<<
410 '
420 PRINT#1,"@RUN":INPUT#1,I
430 CLOSE #1
440 CLOSE #2
450 END
```

Figure 34. Source code for program "DLL1"

```

10 COLOR 15,1,1
20 DEFINT A-Z
30 DIM L(7)
50 '
60 OPEN "COM2:9600,N,8,1,RS,DS,CS" AS #1
70 '
80 '----- Up-loading from robot to files:
90 INPUT "FILE";F$
100 CLS
150 PRINT#1,"@RESET":INPUT#1,I
340 OPEN F$ FOR INPUT AS #2
350 WHILE (EOF(2)=0)
360   INPUT #2 ,L1,L2,L3,L4,L5,L6,L7,L8
370   PRINT#1,"@QWRITE",L1,L2,L3,L4,L5,L6,L7,L8:INPUT#1,I
380 WEND
390 '
400 ' >>>>> EXECUTION ROUTINE <<<<<<<<<<<<
410 '
420 PRINT#1,"@RUN":INPUT#1,I
430 CLOSE #1
440 CLOSE #2
450 END
```

APPENDIX F

Source Code for Program "TESIN1"

Figure 35. Source code for program "TESINI"

```

70  COM(1) ON
80  '$com1 4000
100 KEY OFF:CLS:COLOR 15,1 'D112.bas
110 DEFINT A-Z
115 OPEN"COM2:9600,N,8,1,RS,DS,CS" AS #3
120 OPEN"COM1:9600,N,8,1,RS,DS,CS"AS #1
125 INPUT"INPUT No. OF CYCLES";N
130 FOR C=1 TO N
136 CLS
137 PRINT #1,"@RESET":INPUT #1,I:PRINT #3,"@RESET":INPUT #3,I
138 LOCATE 4,10:PRINT"ASSEMBLIBNG BOARD No.:";C
140 OPEN "a0"FOR INPUT AS #2
150 WHILE EOF(2)=0
160     INPUT #2,L1,L2,L3,L4,L5,L6,L7,L8
170     PRINT #1,"@QWRITE";L1;L2;L3;L4;L5;L6;L7;L8:INPUT#1,I
180     WEND
190     PRINT#1,"@RUN":INPUT#1,I
195     LOCATE 8,10:PRINT "PLACING BOARD"
200     CLOSE #2
300     TIME$="00:00:00"
310     T$=RIGHT$(TIME$,2)
320     IF T$="28" THEN 1140
330     GOTO 310
1140     OPEN "b3"FOR INPUT AS #2
1150     WHILE EOF(2)=0
1160         INPUT #2,L1,L2,L3,L4,L5,L6,L7,L8
1170         PRINT #3,"@QWRITE";L1;L2;L3;L4;L5;L6;L7;L8:INPUT#3,I
1180         WEND
1220         PRINT#3,"@RUN":INPUT#3,I
1225         LOCATE 10,10:PRINT"PLACING CHIP # 1"
1230         CLOSE #2
1390         TIME$="00:00:00"
1400         T$=RIGHT$(TIME$,2)
1410         IF T$="22" THEN 1500
1415         GOTO 1400
1500     PRINT #1,"@RESET":INPUT #1,I:OPEN "a1"FOR INPUT AS #2
1550     WHILE EOF(2)=0
1560         INPUT #2,L1,L2,L3,L4,L5,L6,L7,L8
1570         PRINT #1,"@QWRITE";L1;L2;L3;L4;L5;L6;L7;L8:INPUT#1,I
1580         WEND
1620         PRINT#1,"@RUN":INPUT#1,I
1625         LOCATE 12,10:PRINT"PLACING CHIP # 2"
1630         CLOSE #2

```

Figure 35. (Continued)

```

1640 'STOP
1890 TIME$="00:00:00"
1900 T$=RIGHT$(TIME$,2)
1910 IF T$="27" THEN 1920
1915 GOTO 1900
1920 PRINT #3,"@RESET":INPUT #3,I:OPEN "b5"FOR INPUT AS #2
1930 WHILE EOF(2)=0
1940     INPUT #2,L1,L2,L3,L4,L5,L6,L7,L8
1950 PRINT #3,"@QWRITE";L1;L2;L3;L4;L5;L6;L7;L8:INPUT#3,I
1960 WEND
2010 PRINT#3,"@RUN":INPUT#3,I
2015 LOCATE 14,10:PRINT"PLACING CHIP # 3"
2020 CLOSE #2
2030 'STOP
3000 TIME$="00:00:00"
3010 T$=RIGHT$(TIME$,2)
3020 IF T$="22" THEN 3040
3030 GOTO 3010
3040 PRINT #1,"@RESET":INPUT #1,I:OPEN "a5"FOR INPUT AS #2
3050 WHILE EOF(2)=0
3060     INPUT #2,L1,L2,L3,L4,L5,L6,L7,L8
3070 PRINT #1,"@QWRITE";L1;L2;L3;L4;L5;L6;L7;L8:INPUT#1,I
3080 WEND
3120 PRINT#1,"@RUN":INPUT#1,I
3125 LOCATE 16,10:PRINT"PLACING CHIP # 4"
3130 CLOSE #2
3140 TIME$="00:00:00"
3150 T$=RIGHT$(TIME$,2)
3160 IF T$="22" THEN 3180
3170 GOTO 3150
3180 PRINT #3,"@RESET":INPUT #3,I:OPEN "b6"FOR INPUT AS #2
3190 WHILE EOF(2)=0
3200     INPUT #2,L1,L2,L3,L4,L5,L6,L7,L8
3210     PRINT #3,"@QWRITE";L1;L2;L3;L4;L5;L6;L7;L8:INPUT#3,I
3220 WEND
3230 PRINT#3,"@RUN":INPUT#3,I
3240 LOCATE 18,10:PRINT"REMOVING FINISHED BOARD"
3250 CLOSE #2
3280 TIME$="00:00:00":'PRINT:PRINT"    SETTING NEXT CYCLE"
3290 T$=RIGHT$(TIME$,2):LOCATE 22,10:PRINT TIME$

```


Figure 35. (Continued)

```
3300 IF T$="33" THEN 3660
3310 GOTO 3290
3320 PRINT#1,"@RESET":INPUT#1,I
3330 PRINT#3,"@RESET":INPUT#3,I
3660 NEXT C
5140 CLOSE #1
5150 CLOSE #3
5160 END
```

MONTANA STATE UNIVERSITY LIBRARIES

3 1762 10142971 8