MOTION PLANNING FOR MULTI-ROBOT ASSEMBLY SYSTEMS

by

Martin Bonert

A thesis submitted in conformity with the requirements for the degree of

Master of Applied Science

Graduate Department of Mechanical and Industrial Engineering

at the University of Toronto

© Copyright Martin Bonert 1999
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.
ABSTRACT

The classical Travelling Salesperson Problem (TSP) models the movements of a salesperson travelling through a number of cities. The optimization problem is to choose the sequence in which to visit the cities in order to minimize the total distance travelled. This thesis addresses a multi-robot assembly planning problem which in essence is a TSP-type optimization problem. However, in this augmented TSP (TSP+), both the "salesperson" (a robot with a tool) as well as the "cities" (another robot with a workpiece) move. Namely, in addition to the sequencing of tasks, further planning is required to choose where the "salesperson" should rendezvous with each "city".

A generalized point-to-point (PTP) motion-planning technique is presented in this thesis for manufacturing systems with multiple, coordinated assembly robots that can be modelled as a TSP problem. As an example area, the optimization of the electronic-component placement process is addressed. This TSP+ problem is investigated for single and two placement robots. Namely, the electronic-component placement machine comprises one or two placement robots, a moving XY-table (on which the printed circuit board (PCB) is fixed), and two moving component delivery systems (CDSs). The use of a genetic algorithm (GA) is chosen as the search engine for the solution of the TSP+ optimization problem defined above.

The simulation tools developed within the framework of the thesis were tested on five different component-placement system configurations. In the most generalized configuration, the placement robot meets the CDS at an optimal rendezvous location for the pick-up of the component and subsequently meets the PCB (on the mobile XY-table)
at an optimal rendezvous location. In addition to the solution of the component-placement sequencing problem and the rendezvous-point planning problem, the collision-avoidance issue is addressed for the system configuration with two placement robots.

The proposed novel optimization methodology can facilitate the use of higher degrees of freedom in robotic assembly systems, so that substantial improvements in production times may be obtained. Its effectiveness is successfully shown herein via simulated assembly cases.
ACKNOWLEDGEMENTS

No work of this magnitude is accomplished without advice and help from other people. I have been fortunate to be able to find competent help when needed, to make the completion of this thesis less difficult. I wish to thank all of the people that I’ve consulted at one point or another and those that simply offered words of encouragement when I needed them.

My thesis supervisors, Professors B. Benhabib and L. H. Shu have taught me much and they have my whole hearted gratitude for their guidance and large amount of patience.

Special thanks goes out to all of the CIMLab members with whom I’ve had many a discussion over lunch, and otherwise. Franck Truillet came all the way from France to work together with me on this project for four months during the first summer. Thank you, Franck, Gene, Vladimir, Mehran, Alejandro, Mark and everyone else in the lab.

Thanks to Dan and Brigitte for their C++ and programming advice. I also gratefully acknowledge the financial support of NSERC.

I wish to give special thanks to Anna for being there whenever I needed support. I also wish to thank Bart for keeping me company while I worked on my thesis despite his broken leg. And, finally I thank my family for their support and patience.
# TABLE OF CONTENTS

ABSTRACT ......................................................................................................................... ii  
ACKNOWLEDGEMENTS ...................................................................................................... iv  
TABLE OF CONTENTS ......................................................................................................... v  
LIST OF FIGURES ................................................................................................................ xii  
LIST OF TABLES .................................................................................................................. xvii  
LIST OF ACRONYMS AND SYMBOLS .................................................................................. xviii  

1. INTRODUCTION .............................................................................................................. 1-1  
   1.1. Problem Statement and Motivation .............................................................................. 1-1  
   1.2. A Brief Review of the Literature ............................................................................... 1-2  
      1.2.1. The Travelling Salesperson Problem .................................................................. 1-2  
      1.2.2. The Augmented Travelling Salesperson Problem (TSP+) ............................... 1-4  
      1.2.3. Augmented Travelling Salesperson Problem with Multiple Robots .................. 1-5  
   1.3. Research Objectives and Thesis Structure ................................................................ 1-8  

2. PROBLEM DEFINITION AND SOLUTION APPROACH ............................................... 2-1  
   2.1. Introduction ............................................................................................................... 2-1  
   2.2. Problem Definition .................................................................................................... 2-1  
      2.2.1. Set-Up Geometry .............................................................................................. 2-2  
         2.2.1.1. Single-Placement-Robot Set-Up ................................................................. 2-2  
         2.2.1.2. Two-Placement-Robot Set-Up ................................................................. 2-3  
      2.2.2. Problem Structure ............................................................................................. 2-4  
         2.2.2.1. Single-Robot Problem Structure .............................................................. 2-4  
         2.2.2.2. Two-Robot Problem Structure ............................................................... 2-5
3.1. Introduction ........................................................................................................... 3-1

3.2. The Objective Function .................................................................................... 3-1

3.2.1. Objective Function Input Variables .......................................................... 3-1

(i) Constant Input Data ......................................................................................... 3-2

(ii) Genome Input Data .......................................................................................... 3-5

3.2.2. Calculating Motion Times ........................................................................... 3-9

3.2.3. The Governing Assembly-Time Equation ................................................. 3-11

(i) The Time Before the End of the Component-Pick Operation ......................... 3-14

(ii) The Cycle After the Component Pick Up Operation ........................................ 3-15

3.3. Methodology .................................................................................................... 3-17

3.3.1. Simulation Capabilities .............................................................................. 3-17

3.3.2. Software Configuration .............................................................................. 3-19

(i) Constant Data Input Routines .......................................................................... 3-20

(ii) Genome Data Initialization and Input Routines ............................................ 3-21

(iii) GALib Library Files ....................................................................................... 3-22

(iv) Objective Function Routines .......................................................................... 3-22

(v) Output Routine ............................................................................................... 3-23

3.4. An Example ...................................................................................................... 3-23

3.4.1. The Set Up .................................................................................................... 3-23

3.4.2. Problem Configurations ............................................................................. 3-26

(i) Devices Fixed at Non-Optimal Locations ..................................................... 3-26

(ii) Devices Fixed at Optimal Locations ............................................................ 3-27

(iii) X-Y Table Fixed at an Optimal Location ..................................................... 3-27
(iv) CDSs Fixed at Optimal Locations ................................................. 3-27
(v) All Devices Moving ........................................................................ 3-27

3.4.3. Individual Simulation Results ...................................................... 3-28
(i) Devices Fixed at Non-Optimal Locations ........................................ 3-28
(ii) Devices Fixed at Optimal Locations .............................................. 3-31
(iii) X-Y Table Fixed at An Optimal Location ........................................ 3-34
(iv) CDSs Fixed at Optimal Locations ................................................. 3-37
(v) All Devices Moving ........................................................................ 3-40

3.4.4. Example Results Summary .......................................................... 3-43

3.5. Summary .......................................................................................... 3-44

4. THE TWO-ROBOT AUGMENTED TRAVELLING SALESPERSON PROBLEM 4-1

4.1. Introduction ...................................................................................... 4-1

4.2. The Objective Function ................................................................. 4-1

4.2.1. Objective Function Input Variables ............................................ 4-1

(i) Constant Input Data ......................................................................... 4-2

(ii) Genome Input Data ......................................................................... 4-4

4.2.2. Calculating Motion Times ............................................................ 4-8

(i) The robot time before the pick operation ........................................ 4-12

(ii) The robot time after and including the pick operation ..................... 4-14

4.3. Two-Robot Problem Solution Methodology .................................... 4-14

4.3.1. Simulation Capabilities ............................................................... 4-14

4.3.2. Software Configuration .............................................................. 4-15

4.4. A Simulation Example ..................................................................... 4-16
4.4.1. The Set Up ................................................................. 4-16
4.4.2. Simulated System Configurations ........................................ 4-17
4.4.3. Simulation Results ......................................................... 4-18
   (i) Devices fixed at non-optimal locations .............................. 4-18
   (ii) Devices fixed at optimal locations .................................. 4-21
   (iii) X-Y table fixed at an optimal location .............................. 4-23
   (iv) CDSs fixed at optimal locations ...................................... 4-27
   (v) All devices moving ....................................................... 4-30
4.4.4. Summary of Simulation Results ....................................... 4-33
4.5. Summary ........................................................................... 4-35
5. THE TWO-ROBOT TSP+ COLLISION AVOIDANCE PROBLEM ...... 5-1
   5.1. Introduction .................................................................... 5-1
   5.2. Collision Avoidance Approach ......................................... 5-1
   5.3. The Objective Function .................................................... 5-2
       5.3.1. Objective Function Input Variables ............................. 5-2
       5.3.2. Calculating Motion Times ......................................... 5-2
       5.3.3. The Governing Assembly-Time Equation ..................... 5-3
       5.3.4. The Safety Delay .................................................... 5-4
   5.4. Collision Avoidance Solution Methodology ....................... 5-11
   5.5. A Collision Avoidance Example ....................................... 5-12
       5.5.1. The Set Up ............................................................ 5-12
       5.5.2. Problem Configurations ............................................. 5-13
       5.5.3. Individual Simulation Results ................................... 5-14
(i) Collision Avoidance Routine is not Employed ...........................................5-14

(ii) Collision Avoidance Routine is Employed ...........................................5-14

5.6. An Industrial Application Example ............................................................5-17

5.7. Summary ....................................................................................................5-19

6. SUMMARY AND CONCLUSIONS ....................................................................6-1

6.1. Introduction ................................................................................................6-1

6.2. Single-Robot Augmented Travelling Salesperson Problem (TSP+) .............6-1

6.3. Two-Robot TSP+ ..........................................................................................6-2

6.4. Two-Robot TSP+ and Collision Avoidance ..............................................6-3

6.5. Future Work ................................................................................................6-4

6.5.1. Specific Improvements ...........................................................................6-4

6.5.2. General Suggestions ..............................................................................6-6

6.6. Final Conclusions .......................................................................................6-6

APPENDICES

A. DYNAMIC CDS ALLOCATION ADDITIONAL PLOTS ....................................A-1


A.3. X-Y Table Fixed at An Optimal Location ..................................................A-4

A.4. CDSs Fixed at Optimal Locations .............................................................A-6

A.5. All Devices Moving ....................................................................................A-7

B. DYNAMIC CDS ALLOCATION ADDITIONAL PLOTS .................................B-1

B.1. Devices Fixed at Non-Optimal Locations ..................................................B-1
B.2. Devices Fixed at Optimal Locations ........................................B-1
B.3. X-Y Table Fixed at An Optimal Location..............................B-1
B.4. CDSs Fixed at Optimal Locations .......................................B-2
B.5. All Devices Moving .........................................................B-2
C. ROBOT X-COORDINATE VERSUS TIME PLOTS ..................C-1
   C.1. User-Defined-CDS Allocation ......................................C-1
   C.2. Dynamic CDS-Allocation ............................................C-4
D. COLLISION AVOIDANCE ADDITIONAL PLOTS ......................D-1
   D.1. No Collision Avoidance ...........................................D-1
   D.2. Collision Avoidance .................................................D-1
E. SMC ETHEREZ PCB EXAMPLE ADDITIONAL PLOTS ...............E-1
   E.1. Single Robot Solution .............................................E-1
   E.2. Two-Robot Solution .................................................E-1
REFERENCES ........................................................................R-1
LIST OF FIGURES

Figure 2.1: The Single-Robot Electronic-Component Placement Machine
Configuration. .................................................................2-2

Figure 2.2: The Two-Robot Electronic-Component Placement Machine
Configuration. .................................................................2-3

Figure 2.3: The Various Aspects of the Single-Robot Problem.......................2-4

Figure 2.4: The Various Aspects of the Two-Robot Problem.........................2-6

Figure 2.5: Illustration of a Single Cycle. ........................................2-7

Figure 2.6: A Genome for a Sequencing Problem With 6 Components............2-15

Figure 2.7: Ordered Mutation Operator. ........................................2-18

Figure 2.8: Ordered Crossover Operator........................................2-19

Figure 2.9: The Genetic Algorithm Process.....................................2-20

Figure 3.1: Illustration of Cyclic Device Motion Times..............................3-10

Figure 3.2: An Exemplary Time-Line Plot of a Single Cycle.......................3-13

Figure 3.3: One of the Two Identical CDSs.......................................3-25

Figure 3.4: The PCB With Component Positions...................................3-25

Figure 3.5: Robot Path - Case 1 - User-Defined-CDS-Component Allocation......3-28

Figure 3.6: Time Line - Case 1 - User-Defined-CDS-Component Allocation........3-29

Figure 3.7: Convergence Plot - Case 1 - User-Defined-CDS-Component Allocation...3-30

Figure 3.8: Robot Path - Case 1 - Dynamic-CDS-Component Allocation..........3-30

Figure 3.9: Robot Path - Case 2 - User-Defined-CDS-Component Allocation.....3-31

Figure 3.10: Time Line - Case 2 - User-Defined-CDS-Component Allocation....3-32

Figure 3.11: Convergence Plot - Case 2 - User-Defined-CDS-Component Allocation.3-33
Figure 3.12: Robot Path - Case 2 - Dynamic-CDS-Component Allocation. ..................3-33
Figure 3.13: Robot Path - Case 3 - User-Defined-CDS-Component Allocation. ..........3-34
Figure 3.14: Time Line - Case 3 - User-Defined-CDS-Component Allocation. ..........3-35
Figure 3.15: Convergence Plot - Case 3 - User-Defined-CDS-Component Allocation.3-36
Figure 3.16: Robot Path - Case 3 - Dynamic-CDS-Component Allocation. ...............3-36
Figure 3.17: Robot Path - Case 4 - User-Defined-CDS-Component Allocation ..........3-37
Figure 3.18: Time Line - Case 4 - User-Defined-CDS-Component Allocation ..........3-38
Figure 3.19: Convergence Plot - Case 4 - User-Defined-CDS-Component Allocation.3-39
Figure 3.20: Robot Path - Case 4 - Dynamic-CDS-Component Allocation. ...............3-39
Figure 3.21: Robot Path - Case 5 - User-Defined-CDS-Component Allocation ..........3-40
Figure 3.22: Time Line - Case 5 - User-Defined-CDS-Component Allocation ..........3-41
Figure 3.23: Convergence Plot - Case 5 - User-Defined-CDS-Component Allocation.3-42
Figure 4.1: An Exemplary Time-Line Plot of a Single Cycle........................................4-10
Figure 4.2: Robot Paths - Case 1 - User-Defined-CDS-Component Allocation.........4-18
Figure 4.3: Time Lines - Case 1 - User-Defined-CDS-Component Allocation..........4-19
Figure 4.4: Convergence Plot - Case 1 - User-Defined-CDS-Component Allocation.4-20
Figure 4.5: Robot Paths - Case 1 - Dynamic-CDS-Component Allocation.................4-20
Figure 4.6: Robot Paths - Case 2 - User-Defined-CDS-Component Allocation..........4-21
Figure 4.7: Time Lines - Case 2 - User-Defined-CDS-Component Allocation..........4-22
Figure 4.8: Convergence Plot - Case 2 - User-Defined-CDS-Component Allocation.4-22
Figure 4.9: Robot Path - Case 2 - Dynamic-CDS-Component Allocation.................4-23
Figure 4.10: Robot Paths - Case 3 - User-Defined-CDS-Component Allocation.........4-24
Figure 4.11: Time Lines - Case 3 - User-Defined-CDS-Component Allocation.........4-25
Figure 4.12: The Convergence Plot - Case 3 - User-Defined-CDS-Component Allocation

Figure 4.13: Robot Paths - Case 3 - Dynamic-CDS-Component Allocation

Figure 4.14: Robot Paths - Case 4 - User-Defined-CDS-Component Allocation

Figure 4.15: Time Lines - Case 4 - User-Defined-CDS-Component Allocation

Figure 4.16: Convergence Plot - Case 4 - User-Defined-CDS-Component Allocation

Figure 4.17: Robot Paths - Case 4 - Dynamic-CDS-Component Allocation

Figure 4.18: Robot Paths - Case 5 - User-Defined-CDS-Component Allocation

Figure 4.19: Time Lines - Case 5 - User-Defined-CDS-Component Allocation

Figure 4.20: Convergence Plot - Case 5 - User-Defined-CDS-Component Allocation

Figure 4.21: Robot Paths - Case 5 - Dynamic-CDS-Component Allocation

Figure 5.1: The Four Possible Locations of a Robot

Figure 5.2: The Robots Modelled as Walls for Collision Avoidance Detection

Figure 5.3: Example of Multiple Work Distance Overlaps

Figure 5.4: Time Line of Two-Robot Interaction

Figure 5.5: The Robot Paths; No Collision Avoidance Routine

Figure 5.6: Robot X-Coordinate vs. Time; No Collision Avoidance Routine

Figure 5.7: The Robot Paths; With Collision Avoidance Routine

Figure 5.8: Robot X-Coordinate vs. Time; With Collision Avoidance Routine

Figure 5.9: The SMC EtherEZ Network Card PCB

Figure 5.10: Convergence Plot; Two-Robot Set Up Assembling the SMC PCB

Figure A.1: Time Line - Case 1 - Dynamic-CDS-Component Allocation
Figure A.2: Convergence Plot - Case 1 - Dynamic-CDS-Component Allocation............A-2
Figure A.3: Time Line - Case 2 - Dynamic-CDS-Component Allocation....................A-3
Figure A.4: Convergence Plot - Case 2 - Dynamic-CDS-Component Allocation..........A-4
Figure A.5: Time Line - Case 3 - Dynamic-CDS-Component Allocation....................A-5
Figure A.6: Convergence Plot - Case 3 - Dynamic-CDS-Component Allocation..........A-5
Figure A.7: Time Lines - Case 4 - Dynamic-CDS-Component Allocation....................A-6
Figure A.8: Convergence Plot - Case 4 - Dynamic-CDS-Component Allocation..........A-7
Figure A.9: Robot Path - Case 5 - Dynamic-CDS-Component Allocation....................A-8
Figure A.10: Time Lines - Case 5 - Dynamic-CDS-Component Allocation....................A-9
Figure A.11: Convergence Plot - Case 5 - Dynamic-CDS-Component Allocation...........A-10
Figure B.1: Convergence Plot - Case 1 - Dynamic-CDS-Component Allocation..........B-2
Figure B.2: Convergence Plot - Case 2 - Dynamic-CDS-Component Allocation..........B-3
Figure B.3: Convergence Plot - Case 3 - Dynamic CDS Component Allocation............B-3
Figure B.4: Convergence Plot - Case 4 - Dynamic-CDS-Component Allocation..........B-4
Figure B.5: Convergence Plot - Case 5 - Dynamic-CDS-Component Allocation..........B-4
Figure B.6: Time Lines - Case 1 - Dynamic-CDS-Component Allocation....................B-5
Figure B.7: Time Lines - Case 2 - Dynamic-CDS-Component Allocation....................B-6
Figure B.8: Time Lines - Case 3 - Dynamic-CDS-Component Allocation....................B-7
Figure B.9: Time Lines - Case 4 - Dynamic-CDS-Component Allocation....................B-8
Figure B.10: Time Lines - Case 5 - Dynamic-CDS-Component Allocation....................B-9
Figure C.1: Robot X-Coordinate versus Total Time - Case 1 - User-Defined..............C-1
Figure C.2: Robot X-Coordinate versus Total Time - Case 2 - User-Defined..............C-2
Figure C.3: Robot X-Coordinate versus Total Time - Case 3 - User-Defined..............C-2
Figure C.4: Robot X-Coordinate versus Total Time - Case 4 - User-Defined............. C-3
Figure C.5: Robot X-Coordinate versus Total Time - Case 5 - User-Defined............. C-3
Figure C.6: Robot X-Coordinate versus Total Time - Case 1 - Dynamic.................... C-4
Figure C.7: Robot X-Coordinate versus Total Time - Case 2 - Dynamic.................... C-4
Figure C.8: Robot X-Coordinate versus Total Time - Case 3 - Dynamic.................... C-5
Figure C.9: Robot X-Coordinate versus Total Time - Case 4 - Dynamic.................... C-5
Figure C.10: Robot X-Coordinate versus Total Time - Case 5 - Dynamic.................. C-6
Figure D.1: The Time Lines; No Collision Avoidance........................................... D-2
Figure D.2: The Time Lines; Collision Avoidance............................................... D-3
Figure D.3: The Convergence Plot; No Collision Avoidance.................................... D-4
Figure D.4: The Convergence Plot; Collision Avoidance....................................... D-4
Figure E.1: The Convergence Plot, Single Robot Solution to the SMC PCB Problem. E-1
Figure E.2: The Robot X-Coordinate versus Time Plot for the Two Robot Solution of
the SMC PCB Problem. ......................................................................................... E-2
LIST OF TABLES

Table 3.1: The Simulation Results Summary for User-Defined CDS Allocation. ........3-43
Table 3.2: The Simulation Results Summary for Dynamic CDS Allocation..................3-44
Table 4-1: Simulation Results Summary for User Specified CDS Allocation. .............4-33
Table 4-2: The Simulation Results Summary for Dynamic CDS Allocation..............4-34
Table 4-3: Comparing Single- and Two-Robot Performance for the All-Moving Case

With Dynamic-CDS-Component Allocation. ....................................................4-34
Table 6.1: The Simulation Results Summary for the Single-Robot Problem...............6-2
Table 6.2: Comparing Single- and Two-Robot Performance. .....................................6-3
LIST OF ACRONYMS AND SYMBOLS

List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
<td>1-4</td>
</tr>
<tr>
<td>CDS</td>
<td>Component-Delivery System</td>
<td>2-2</td>
</tr>
<tr>
<td>ECPSIM</td>
<td>Electronic Component Placement SIMulator</td>
<td>3-17</td>
</tr>
<tr>
<td>ECPSIM2R</td>
<td>Electronic Component Placement SIMulator for 2 Robots</td>
<td>4-14</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
<td>1-1</td>
</tr>
<tr>
<td>GALib</td>
<td>The name of M. B. Walls C++ genetic algorithms class library</td>
<td>3-17</td>
</tr>
<tr>
<td>OR</td>
<td>Operations Research</td>
<td>1-1</td>
</tr>
<tr>
<td>PCB</td>
<td>Printed Circuit Board</td>
<td>1-3</td>
</tr>
<tr>
<td>PTP</td>
<td>Point-to-Point</td>
<td>1-1</td>
</tr>
<tr>
<td>TSP</td>
<td>Travelling-Salesperson-Problem</td>
<td>1-1</td>
</tr>
<tr>
<td>TSP+</td>
<td>Augmented Travelling Salesperson Problem</td>
<td>1-4</td>
</tr>
</tbody>
</table>

List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>The time it takes to complete assembly cycle i (current cycle)</td>
<td>3-11</td>
</tr>
<tr>
<td>$C_j$</td>
<td>The total cycle time for cycle j</td>
<td>5-9</td>
</tr>
<tr>
<td>$^{\text{pick}}C_i$</td>
<td>The first part of the cycle time</td>
<td>3-14</td>
</tr>
<tr>
<td>$^{\text{place}}C_i$</td>
<td>The second part of the cycle time</td>
<td>3-15</td>
</tr>
<tr>
<td>$^{\text{pk}}C_i$</td>
<td>Component-pick-up time</td>
<td>3-12</td>
</tr>
<tr>
<td>$^{\text{pl}}C_i$</td>
<td>Component placement time</td>
<td>3-12</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>$\max_{i} d_i$</td>
<td>Maximum work distance overlap</td>
<td>5-7</td>
</tr>
<tr>
<td>$N$</td>
<td>The number of components on the PCB</td>
<td>3-11</td>
</tr>
<tr>
<td>$PK_i$</td>
<td>Current pick location</td>
<td>3-10</td>
</tr>
<tr>
<td>$PK_{i-1}$</td>
<td>Previous pick location</td>
<td>3-10</td>
</tr>
<tr>
<td>$PL_i$</td>
<td>Current placement location</td>
<td>3-10</td>
</tr>
<tr>
<td>$PL_{i-1}$</td>
<td>Previous placement location</td>
<td>3-10</td>
</tr>
<tr>
<td>$t$</td>
<td>The overall assembly time for a complete population of a PCB</td>
<td>3-11</td>
</tr>
<tr>
<td>$c_{rc} t_i$</td>
<td>The time it takes the current robot to cross the maximum work distance overlap</td>
<td>5-9</td>
</tr>
<tr>
<td>$c_{rp} t_j$</td>
<td>The time it takes the previous robot to cross the maximum work distance overlap</td>
<td>5-10</td>
</tr>
<tr>
<td>$d_{ti}$</td>
<td>The time it takes the CDS to move to the pick location (in cycle $i$)</td>
<td>3-10</td>
</tr>
<tr>
<td>$r_{rc} t_i$</td>
<td>Robot entry time</td>
<td>5-8</td>
</tr>
<tr>
<td>$r_{rp} t_j$</td>
<td>Robot leaving time</td>
<td>5-8</td>
</tr>
<tr>
<td>$d_{off} t_i$</td>
<td>CDS off time</td>
<td>3-12</td>
</tr>
<tr>
<td>$t_{off}$</td>
<td>The time the XY-table has had off</td>
<td>3-13</td>
</tr>
<tr>
<td>$p_{rk} t_i$</td>
<td>Robot motion-to-pick-location time</td>
<td>3-10</td>
</tr>
<tr>
<td>$p_{rl} t_i$</td>
<td>Robot motion-to-placement-location time</td>
<td>3-10</td>
</tr>
<tr>
<td>$r_{ti}$</td>
<td>The robot time</td>
<td>4-11</td>
</tr>
<tr>
<td>$r_{t_{i}^{1st}}$</td>
<td>The first part of the robot cycle time</td>
<td>4-12</td>
</tr>
</tbody>
</table>
$r_t^{2nd}$  The second part of the of the robot cycle time.....................4-14

$rd_t$  Collision avoidance safety delay (for current cycle)..................5-3

$rd_{t_j}$  Safety delay (for a collision event in cycle $j$).......................5-11

$r_i$  X-Y table's motion time-to-placement-location .........................3-11

$rcW_i$  The work distance of the current robot .....................................5-7

$rpW_a$  Work distances of previous robot (for cycle $a$).........................5-7

$rpW_n$  Work distances of previous robot (for cycle $n$).........................5-7

$rc\dot{x}_i$  The speed of the current robot in $X$ axis direction..................5-9

$rp\dot{x}_j$  The speed of the previous robot in $X$ axis direction ...............5-10
1. INTRODUCTION

1.1. Problem Statement and Motivation

Autonomous robotic systems are being increasingly utilized in industrial environments, requiring the development of modelling methodologies and tools to optimize their operational efficiency. In this context, the literature is extensive on robot motion planning and optimization methods. The specific research area of this thesis is optimal multi-robot assembly.

The classical assembly-planning optimization problem has been extensively studied, with numerous recent attempts at applying the earlier research results to robotic-based assembly, [e.g., 1]. As expected, most proposed solution methods have their roots in the classical Operation Research (OR) field. Over the past several decades, OR research groups have devised many effective solution approaches to the combinatorial Travelling Salesperson Problem (TSP), [e.g., 2], which is naturally a two-dimensional point-to-point motion optimization problem.

This thesis presents a generalized flexible point-to-point (PTP) motion-planning technique for multiple coordinated assembly robots, which can be modelled as a TSP. The approach minimizes assembly times for single and multiple coordinated robot assembly systems using Genetic Algorithms (GAs). As an example area, the optimization of an electronic component placement process is used in this thesis [e.g., 3, 4].
1.2. A Brief Review of the Literature

The literature pertinent to this thesis can be divided into three general areas: The Travelling Salesperson Problem, the Augmented Travelling Salesperson Problem, and the Augmented Travelling Salesperson Problem with Multiple Robots. Since the application area chosen in the thesis is electronic component placement, many of the papers reviewed below are related to this process.

1.2.1. The Travelling Salesperson Problem

The classical TSP models the movements of a salesperson travelling through a number of cities. The optimization problem is to choose the order in which to visit the cities to minimize the total distance travelled. In such a tour, the salesperson starts in a chosen city, visits all cities only once, and finally returns to the first city. Many variations of this combinatorial problem have been addressed in literature, including the Asymmetric, Symmetric, Euclidean, Chebyshev, Prize Collecting and Time-dependent TSP variations [e.g., 2].

In the context of the industrial applications at hand, Ji et al. [5] present a linear assignment-based optimized method for electronic-component placement, discussing in detail the issues of placement sequencing and optimum bin location. A single-robot placement machine is modelled. Although the placement sequence is modelled as a TSP problem, the assignment of components to part bins is modelled as a quadratic problem.
Dubowsky and Blubaugh [6] present a point-to-point TSP motion-planning technique also using a single robot. They discuss strategies for generating minimum-time motions, rather than minimum distance, for tasks such as electronic-component placement and spot welding. They also briefly address optimal workcell reconfiguration. An exact tour-building algorithm using a branch-and-bound technique is proposed to solve the TSP problem.

Leu et al. [7] solve the electronic-component placement optimization problem using genetic algorithms. The bin assignment problem is also addressed. Three types of assembly machines were modelled: (1) A basic single robot pick-and-place problem with fixed feeders and a fixed Printed Circuit Board (PCB), (2) Problem 1 extended to have feeder component assignment optimization, and (3) A multi-head fixed placement turret with a moving XY-table and feeder component optimization.

Nelson and Wille [8] compare various solution approaches to the optimization of PCB assembly. A single robot placement-sequencing problem with fixed pick-and-placement locations is modelled and solved with GAs, Evolutionary Programming and Simulated Annealing.

Naft [9] describes a modified TSP model to optimize PCB design by considering wire-length criteria (conductive path lengths between components) and other design parameters, including thermal reliability. The solution approach uses an adapted Hopfield neural network to solve the modified TSP.
Another approach used to solve the classic TSP or variations thereof in PCB assembly has been the use of expert systems. Huang et al. [10] propose the use of an Artificial Intelligence (AI) based expert system to solve the placement sequence and bin allocation problem for such machines.

1.2.2. The Augmented Travelling Salesperson Problem (TSP+)

In contrast to the single-robot TSPs, where the primary objective is to find the best sequence for N tasks, for multi-robot problems, one must also solve the "rendezvous-point" planning problem. In this augmented TSP, (TSP+), the "salesperson" (one robot) as well as the "cities" (PCB placement locations moved around by another robot) have motion capability. Namely, further planning is required to choose where the "salesperson" should rendezvous with the "city". This minimum-time optimization problem is further complicated when two placement robots are used concurrently in a task-sharing mode.

There exist two variations to this rendezvous-planning problem: (1) Continuous-path planning where two robots move synchronously along a continuous curve, and (2) Point-to-point path planning, where the robots move independently and only rendezvous at discrete points.

Cao et al. [11,12] address the issue of inspection-task-sequence planning for two coordinated robots. Two SCARA robots are used to investigate the TSP+ problem, where one robot holds the inspection tool and the other holds the part to be inspected. A
series of locations on a sphere are inspected by the robot pair, where both robots move together to bring the inspection tool to the rendezvous location. Using the simulated-annealing technique, they were able to plan numerous point-to-point inspection routes. As expected, they showed that the cooperative robot configuration, where both robots moved, was faster than when only one robot moved and the other acted as a fixture.

Tabarah et al. [13] present two algorithms for the coordination of two robots working cooperatively, on continuous-path motion applications, with one robot holding the tool and the other the workpiece. Both robots move simultaneously in a way to ensure that the tool maintains continuous contact with the workpiece while following a continuous prescribed trajectory along its surface.

A coordinated motion-control approach for a two-axis worktable and a 7-dof welding robot is presented by Ahmad and Luo, [14]. The redundancy of the robot (7-dof) allows greater maneuverability and singularities are avoided by keeping the robot in a good maneuverable position away from its motion limits. A cooperative control approach is presented by Suh and Shin, [15], also for a robot and workpiece positioner. Combined coordinated control of both devices by a single controller is proposed.

1.2.3. Augmented Travelling Salesperson Problem with Multiple Robots

In the previous section, the TSP+ was addressed for two robots working together in a coordinated manner. The robots only shared a work area during the coordinated activities and during the remaining time it was assumed that the robots would stay out of
a common work area, thus, avoiding potential collisions. However, if an additional robot is introduced into the TSP+, which continuously shares its workspace with the other, it would be necessary to directly address the collision avoidance problem.

The TSP+ with multiple robots can be better described by the example considered in this thesis, where the component placement machine comprises two individual placement robots and a mobile table on which the PCB is located. This machine, thus, has two two-robot systems each comprising two mobile devices: a placement robot and the X-Y table. One can easily note the potential of collision between the two robots attempting to place components on the same PCB. This multiple-robot collision avoidance problem has been addressed in two primary ways in the literature: (1) Collision avoidance through path planning, and (2) Collision avoidance through scheduling (or time delays).

Lee and Lee [16] propose solving the collision-avoidance problem by speed changes, which introduce a time delay in one of the two robots. Each robot is represented by a single sphere at the wrist, and straight-line motion is assumed. For paths that are close together, collision maps of time versus distance travelled along the robot path are generated.

Baba and Kubota [17] present a collision-avoidance-planning method using a GA for a robot manipulator in an environment that includes moving obstacles (other robots). The method divides the workspace into uniform cells and assigns potentials to each cell
depending on its proximity to an obstacle. To enable this method to work with a second robot, the initial path for the second robot is generated and a potential field is introduced to represent the second robot. The goal is then to get the first robot from a starting point through the workspace to an end point without any collisions. A path consisting of a string of cells is generated through the workspace and is evaluated by the GA based on the criteria of total distance, total of cell potentials and maximum potential encountered.

An off-line collision-free path planning method is suggested by Zurawski and Phang [18], which also divides the workspace into cells for path representation. Robot motion is restricted to orthogonal movements within the grid. The first robot to enter the shared workspace is assigned the master status and the second robot the slave status. The master robot is given priority to complete its task. This method checks for position conflicts, first, by generating a map of cells showing the positions of the robots, which may indicate a possible collision. If a position overlap occurs, then, the timing of the event is checked to confirm or rule out a collision. In a collision case, the slave robot's path is replanned to avoid the master robot.

Another method for avoiding collisions between two robots, using a simple time delay is presented by Chang et al., [19]. The robots are approximated by polyhedra and collision is detected by calculating the distance between the two robots. A collision map scheme is adapted to describe any possible collisions. In the event of a possible collision, one of the robots is delayed to prevent a collision.
Baptiste et al. [20] present an approach for scheduling the motions of two or more hoists sharing a single track and a set of tanks in PCB etching. Constraint logic programming is used to model the problem and to find the optimal cyclic sequence of moves for the hoists, so that the PCBs are moved from tank to tank as efficiently as possible.

1.3. Research Objectives and Thesis Structure

The goal of this thesis is to develop a flexible point-to-point motion planning method for systems with coordinated robotic devices using genetic algorithms. The method should be able to model and optimize problems associated with both single and multiple robots. Using this method, the TSP, TSP+ and multi-robot TSP+ problems must be able to be solved.

Chapter 2 will present the various aspects of the single- and two-robot problems and provide an overview of the proposed solution approach.

Chapter 3 develops the proposed general method for solving the single-robot coordinated TSP+ problem. A number of simulation results of electronic-component-placement problems for the single-robot are presented for different system configurations.
Chapter 4 expands on the concepts in Chapters 2 and 3 to develop a method for solving the TSP+ problem for two-robot systems. Examples of two-placement-robot system configurations are presented.

Chapter 5 addresses the collision avoidance issue for two-robot problems.

The last chapter of the thesis, Chapter 6, contains the conclusions of this thesis and recommendations for future investigation.
2. PROBLEM DEFINITION AND SOLUTION APPROACH

2.1. Introduction

In electronic assembly, components must be placed onto the PCB in a time-efficient manner. The first task is the configuration of the PCB, where component locations are determined subject to constraints and objectives. In this research, it is assumed that this task has already been carried out. It is also assumed that the component-placement machine picks and places one component at a time. Although various other placement strategies exist, and are further detailed in the literature, [e.g., 7], the objective of this thesis is the investigation of the fundamental TSP+ problem as it applies to electronic component placement. As stated in Chapter 1, the primary goal is to minimize the travel time of the robotic devices of the placement system, thereby minimizing the assembly time of the given electronic component assembly process.

In this chapter, the fundamental nature of the problems studied in this thesis is defined and the outline of the proposed solution approach follows.

2.2. Problem Definition

In this section, the problem will be defined for both the single- and two-robot problem. The problem will be broken down into the following sub-issues: (1) Set-up Geometry, (2) Problem Structure and (3) Assembly Cycle.
2.2.1. Set-Up Geometry

The geometry of the assembly cell considered herein includes such parameters as: cell size, types and number of devices and device geometry and positions. The following sections outline these parameters for the single- and two-placement-robot problems.

2.2.1.1. Single-Placement-Robot Set-Up

Figure 2.1 shows the most generalized physical set up of the single-robot-placement machine modelled. The system comprises four main sub-systems: an X-Y gantry robot for component pick-and-place operations; a numerically controlled X-Y table, on which the PCB is located; and, two identical single-dof multiple-component delivery systems, with controllable motions in the Y direction.

![Diagram of Single-Robot Electronic-Component Placement Machine Configuration](image)

Figure 2.1: The Single-Robot Electronic-Component Placement Machine Configuration.

The gantry robot's workspace includes the workspace of both Component-Delivery Systems (CDSs) and the workspace of the X-Y table. The individual component-delivery devices (bins) are assumed to be attached to each other and move
together along the Y axis. PCB boards enter and exit the component-placement machine at fixed locations at the top and bottom of the workspace respectively.

2.2.1.2. Two-Placement-Robot Set Up

Figure 2.2 shows the physical set up of the two-robot placement machine modelled. The system is identical to the single-robot system displayed in Figure 2.1, except for an additional X-Y gantry robot. This additional robot also performs component pick-and-place operations.

![Diagram of Two-Robot Electronic-Component Placement Machine Configuration](image)

Figure 2.2: The Two-Robot Electronic-Component Placement Machine Configuration.

The two gantry robots share a common workspace, which includes the workspace of their respective component-delivery systems and the workspace of the X-Y table. Each CDS is accessed by only the robot assigned to it. For example, Robot 1 only uses CDS 1 (see Figure 2.2). The two robots are not allowed to crossover each other and switch sides.
2.2.2. Problem Structure

With the geometry of the system defined, it now becomes possible to define the various other aspects of the path-planning problem. The placement process is broken down below into a series of sub-problems.

2.2.2.1. Single-Robot Problem Structure

For the single-robot problem, there exist four sub-problems: Assigning components to CDSs; Determining the placement sequence; Finding the rendezvous locations; and, Planning the robot path. Figure 2.3 shows the layers of the problem.

```
<table>
<thead>
<tr>
<th>CDS Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequencing</td>
</tr>
<tr>
<td>Rendezvous Location Planning</td>
</tr>
<tr>
<td>Robot Path Planning</td>
</tr>
</tbody>
</table>
```

Figure 2.3: The Various Aspects of the Single-Robot Problem.

The CDS assignment is the process of assigning the various component types to occupy specific bins in the two CDSs. The component placement sequence is a combinatorial optimization problem to minimize assembly time. The rendezvous-planning problem is the process of determining the meeting positions of the placement robot and the CDSs, and the meeting positions of the placement robot and the mobile X-Y table. When the combined dof of the two moving sub-systems is above the minimum needed (e.g., 2 dof for planar problems), an infinite number of possible rendezvous-
location solutions exist for every potential pick or placement exchange between two moving devices. Therefore, for a given sequence of picks and placements, a corresponding set of optimal rendezvous locations must be determined.

For a given point-to-point (PTP) (rendezvous) motion, the fastest robot path is normally determined using the robot dynamics. In this case, for a given sequence, the optimality of a potential set of rendezvous locations can be determined by measuring the overall motion time. To achieve optimal results, individual robot paths between these rendezvous locations must also be optimized. This robot path sub-optimization problem is not addressed in this thesis since it has been extensively addressed by the robotics research community [e.g., 21]. Herein, it is simply assumed that minimum robot-motion time can be achieved by minimizing the Cartesian distance travelled by the individual devices.

2.2.2.2. Two-Robot Problem Structure

For the two-placement-robot case, the problem is broken down into five sub-problems: Robot job assignment; Determining the overall sequence; Rendezvous location planning; Robot movement coordination; and, Robot path planning. The various layers can be seen in Figure 2.4.
Figure 2.4: The Various Aspects of the Two-Robot Problem.

Although the components are placed one at a time, according to an overall sequence, each component is placed by only one of the two robots. This requires the placement tasks to be divided up and each robot to be assigned the components it is responsible for placing. Since components are assumed to be placed sequentially, the placement sequence is identified as a variable. The third task is the solution of the rendezvous-location-planning problem for every pair of interacting mobile devices. Once the rendezvous positions have been determined, since there are two gantry robots sharing the workspace, it is necessary to coordinate the actions of the two robots to prevent collisions. The final problem is the robot-path-planning problem. As discussed in the single-robot problem above, it is simply assumed that minimum robot-motion time can be achieved by minimizing the distance travelled by the individual robots.

2.2.3. The Assembly Cycle

In order to calculate placement cycle times, the overall board-population assembly problem is divided into individual cycles, where a single cycle comprises a
complete pick-and-place operation for a single component. This pick-and-place operation consists of: (i) the pertinent robot and the CDS moving together to the pick location and component pick up, and (ii) the pertinent robot and X-Y table moving to the placement location and placement of the component on the board.

Figure 2.5 shows the process of a generic pick-and-place cycle. The robot starts this cycle at its previous placement location and the CDS starts at its previous pick location. The CDS moves to the current pick location, (1), while the robot simultaneously moves there as well to rendezvous with the CDS, (2). The robot then picks the component from the CDS. While (1) and (2) are happening, the X-Y table moves to the current placement location, (3), where it meets with the robot arriving from the current pick location, (4) (while (3) and (4) are happening, the CDS is allowed to move, without waiting, to the next pick location). The robot then places the component and the cycle repeats itself for the next component in the placement sequence.

Figure 2.5: Illustration of a Single Cycle.
2.3. Proposed Solution Approach

The goal of this thesis is the development of a methodology to optimize the process outlined above. Rather than optimizing one parameter at a time, a method that allows us to optimize all parameters simultaneously is chosen. This is advantageous since many of the parameters are interdependent and the modification of one problem parameter affects another. Use of a genetic algorithm was chosen as the solution approach for the problem at hand (for readers not familiar with GAs, a brief review is presented in Section 2.4).

Each of the sub-problems in Section 2.3 is represented as a group of variables in the proposed solution approach. The parameters for the single- and two-robot placement problems are encoded in a genome for each robot. Subsequently an objective that yields the highest fitness value to the solution with the minimum assembly time is defined. This fitness is used to determine which genomes are chosen for reproduction. The genetic operators, then, generate new offspring, which are evaluated, and the entire process is repeated.

The genome (data string) structures, used for the single- and two-robot problems are outlined below. The objective functions are also briefly discussed below, however, they are further discussed in detail in Chapters 3 to 5. To choose the parents, once a fitness has been determined, a roulette selection method is used. Then, during each GA iteration, half of the population is replaced using genetic operators. The genetic operators used are limited to mutation and crossover operators: An ordered mutation operator and
an ordered crossover operator for the sequencing, CDS allocation, robot assignment, and split-point sub-genomes (a sub-genome that contains part of the CDS and robot assignment data, Sections 2.3.1.2 and 2.3.2.2). For the rendezvous location problem sub-genome, a binary mutation operator is used together with a simple uniform crossover.

2.3.1. Single-Robot Solution Approach

To encode the single-robot parameters, a compound genome consisting of a series of sub-genomes is proposed. These sub-genomes contain variables to address each of the sub-problems outlined for the single-robot problem (Section 2.2.2.1). The compound genome for the single-robot problem consists of: a sequencing sub-genome, a CDS allocation sub-genome, a split-point sub-genome, and a rendezvous-point planning sub-genome.

2.3.1.1. The Sequencing Sub-Genome

The sequencing sub-genome contains the placement sequence information as a string of integers where each number identifies a placement location on the PCB. The order of the numbers determines which component is picked and placed next. Each corresponding placement location on the board has a matching component identifier. Each component also has a component type, which is used for locating the component bin during the pick operation.
2.3.1.2. The CDS Allocation and Split-point Sub-Genomes

The CDS allocation and split-point sub-genomes contain the information necessary to allocate components to the two CDSs and to order the components in the CDS compartments. The CDSs are modelled as a series of bins connected together. Each bin is represented by a pickup point. Each point is characterized by a distance from the CDSs' frames' origins. Therefore, the order of the components in the bins affects the location of the pickup points, which in turn affects the distances travelled by the robot.

Both the CDS allocation and split-point sub-genomes provide for the encoding of bin allocations and their optimization. The CDS allocation sub-genome consists of a string of integers each corresponding uniquely to one component.

The split-point sub-genome consists of an integer between zero and an integer representing the total length of the CDS allocation sub-genome. Using this number as an array position index, the CDS allocation sub-genome is divided into two arrays, with all numbers before the split-point forming one array, and the remaining numbers forming the second array. Each of these arrays specifies the contents of a CDS.

2.3.1.3. The Rendezvous-Point-Planning Sub-Genome

The rendezvous-point-planning sub-genome contains the position information of all devices that can move. In its raw form, the rendezvous sub-genome is a binary string of ones and zeros, where each eight bits (a byte) are the encoded form of a real number. These numbers are then decoded to give a series of coordinates for the CDSs and the X-Y
The robot positions are then calculated from the CDSs positions, X-Y table positions, the bin offsets and the component offsets.

As explained above in Section 2.3.1.2, a CDS is modelled as a series of bins with known constant offsets from the CDS' frame's origin. Using the CDS position and the bin assignment of a given component, the pick position for the robot can be calculated as the CDS position plus the bin offset. Each component position on the PCB has a known predefined component offset from the X-Y table's frame's origin. Using the X-Y table position and the placement location of a given component, the placement location for the robot can be calculated as the X-Y table position plus the component offset. Combining the pick-and-place positions yields the robot positions.

2.3.1.4. The Objective Function

The objective function of the GA takes the above defined compound genome, decodes it and applies a formula to determine a fitness value. This fitness value is then assigned to that particular solution. In the single-robot case, the objective function determines the total time necessary to populate the complete PCB. Solutions with shorter assembly times are given a better fitness value than those with longer assembly times. The objective function for the single robot is described in detail in Chapter 3.
2.3.2. Two-Robot Solution Approach

For the two-robot case, the solution strategy for solving the problem is separated into two parts: the optimization method and the collision avoidance method. Once again, for the optimization the use of a genetic algorithm is adopted. The collision avoidance strategy in this thesis is a rule based approach and incorporated into the objective function of the GA. Since the parameters of sequencing and device positions during pick-and-place operations affect both the collision avoidance problem and the performance of a given placement strategy, it is advantageous to check both simultaneously when searching for an optimum solution, hence the integration into a single objective function.

To encode the two-robot parameters, a compound genome consisting of a series of sub-genomes is proposed. These sub-genomes contain variables to address each of the sub-problems outlined for the two-robot problem (Section 2.2.2.2). The compound genome consists of a sequencing sub-genome, a robot-assignment sub-genome, a split-point sub-genome and a rendezvous-point planning sub-genome.

2.3.2.1. The Sequencing Sub-Genome

The sequencing sub-genome for the two-robot problem is the same as one for the single-robot problem. It contains the placement sequence information as a string of integers where each number represents a placement location (and matching component) on the PCB. The order of the numbers sets the master sequence which governs both robots.
2.3.2.2. The Robot-Assignment and Split-Point Sub-Genomes

The robot-assignment and split-point sub-genomes serve two functions. They contain the information as to which robot is placing which components, and they also serve the same purpose as the CDS allocation sub-genome and split-point sub-genome for the single-robot problem, assigning components to CDSs.

The robot-assignment sub-genome is similar to the CDS allocation sub-genome of the single robot. It consists of a string of unique integers, where each number represents a component. The split-point sub-genome for the two-robot problem is also the same as the one used for the single-robot problem. It is used to split the robot-assignment sub-genome into two strings of arrays that specify which robot places which components.

As described above in Section 2.2.1.2, for the two-robot problem, each robot is paired with a CDS. Examining Figure 2.2, one can see that there is a CDS and a robot pair on the left and a CDS and robot pair on the right. Therefore, each of the two strings is used (as with the single-robot problem) to define component allocation to the CDSs. This ensures that the robot has access to the components it is assigned to pick and place.

2.3.2.3. The Rendezvous-Point-Planning Sub-Genome

The rendezvous-point-planning sub-genome for the two-robot problem is the same as the corresponding sub-genome for the single-robot problem and contains position information about all the mobile devices. The positions of the X-Y table, the CDSs, the components’ offsets’ and the bins’ offsets are used together with the robot
assignment sub-genome and sequencing sub-genome to determine the two robots' pick-and-place locations, namely, defining their placement paths.

2.3.2.4. The Objective Function

As mentioned in Section 2.3.2, the two-robot objective function not only has to optimize the placement process but also simultaneously solve the movement coordination problem (see Section 2.2.2.2). The objective function, as with the single-robot case, decodes the genome, and uses a formula to evaluate the fitness of the solution. The two-robot optimization problem and its associated objective function are discussed in detail in Chapters 4 and 5.

2.4. Genetic Algorithms [22]

Genetic Algorithms (GAs) mimic the natural selection process, where a population of one species is exposed to an environment, and the fittest survive and reproduce. The characteristics of the individuals, encoded in their DNA, determine fitness. As the population evolves through successful reproduction, the fittest DNA becomes dominant.

For use as an optimization method, the characteristics encoded in a genome describe one possible solution to the problem. The environment is a user-defined objective function that, given the genome of each population member, returns a fitness for that member. Through genetic operators new population members are generated.
After the population has "evolved" for a number of generations, the fittest solution becomes dominant. This solution is the optimum or near-optimum solution to the problem described by the genome.

A GA, thus, consists of the following components: a population of genomes each containing a solution to the problem, a method for generating the initial population of genomes, an objective function that evaluates a genome's fitness, a genome selection method, and a method of applying genetic operators to generate new genomes (solutions) from the previous generation.

2.4.1. A Genome

A genome is best described as an information set. For application in a numerical optimization problem, it is usually a string of numbers containing all independent variables to be optimized. For example, in a sequencing problem of six components, one could use the genome depicted in Figure 2.6. Each number (or gene) represents one of the six events, then the order of the numbers yields the placement order.

![Gene and Genome Diagram](gene.png)

Figure 2.6: A Genome for a Sequencing Problem With 6 Components.
2.4.2. Initial Population Generation

Once the nature of the problem has been encoded into a genome, an initial population of solutions (genomes) must be generated. The easiest way to generate an initial population is to use randomization. This is done by using various randomization functions, specific to the genome type. Generally for each gene, a value is randomly picked from a predefined acceptable range of values.

2.4.3. The Objective Function

The objective function is the most important part of the GA. This function takes the characteristics stored in each genome and evaluates that genome's fitness to reproduce. By designing an objective function to assign the best fitness to the genome with the best solution, the GA becomes an optimization tool. Since the user defines the criterion for optimization, the GA can optimize any parameters of the solution the user chooses.

2.4.4. The Selection Process

The selection process chooses a number of genomes which will reproduce to create new genomes based on the fitness scores of the population. There exist many selection processes, the most commonly used of which is the roulette wheel.

The process is called the roulette wheel because the genomes are assigned a probability of reproduction based on their fitness. Each genome is assigned a section of
the wheel equal in size to its probability of reproduction. This wheel is then spun to choose a parent every time one is required for a reproduction operation.

2.4.5. Genetic Operators

Genetic operators modify the genome in a particular fashion to create a new genome. These operators are applied to the fittest genomes (provided by the selection process) to produce new "offspring" thereby furthering the evolutionary optimization process. There exist many different operators that have been devised by researchers. These can be classified as sexual or asexual. Sexual operators receive two genomes and create one or more new genomes by taking parts of each genome and combining them to create the new genome(s). Asexual operators receive one genome as input and produce a new genome by modifying one or more of its genes (often randomly).

Usually, a mixture of sexual and asexual operators are used with probabilities assigned to the use of one operator or another. The two basic operators most commonly used are the mutation operator and the crossover operator.

2.4.5.1. Mutation Operator

The mutation operator is an asexual operator. It takes a genome and modifies one (or more) of its gene(s) by replacing the value with a randomly generated value. This value is generated in a way to ensure that it is within the acceptable range for that gene. There exist many different mutation operators since they are adapted to the nature of the
data on which they operate. Usually, mutation operators are assigned a relatively small probability of being used, since they often result in large changes in the genome, aiding in a global search but interfering with a local optimization.

Figure 2.7 shows an ordered mutation operator. Ordered operators are for sequences where the same number cannot occur twice. This operator takes the original genome (or parent) and swaps two genes to generate the offspring.

![Diagram of Ordered Mutation Operator]

**Figure 2.7: Ordered Mutation Operator.**

### 2.4.5.2. Crossover Operator

The crossover operator is a sexual operator. Namely, it receives two genomes as input, which act as parents for the new offspring. The simplest form of a crossover operator takes half of one parent and half of another parent, and combines the first half of parent one with the second half of parent two, resulting in an offspring. It then takes the remaining halves and combines them to form a second offspring.
As with the mutation operator, there exist different types of crossover operators depending on the nature of the genomes on which they are operating. In contrast to mutation operators, however, crossover operators are assigned a large probability of being used, since they combine good solutions with good solutions. The ordered crossover operator, like its mutation counterpart, is designed for sequences where each number is only allowed to occur once in the genome.

An example of an ordered crossover operator can be seen in Figure 2.8.

![Crossover Operator Diagram](image)

**Figure 2.8: Ordered Crossover Operator.**

### 2.4.6. The GA Process

Figure 2.9 shows a typical genetic algorithm, where the initial population of genomes is proposed to be generated randomly.
2.5. Summary

In this chapter, the set-up geometries of the single-robot and two-robot problem were defined. The various sub-problems of the optimization processes were then outlined. Calculation of a pick-and-place cycle time was also discussed as a common metric for both single-robot and two-robot problems.
The solution approach proposed utilizes a genetic algorithm (GA) to optimize the placement process. The application of GA to be further detailed in the next chapters was briefly discussed herein, including the choices of genome structures, population initialization methods, genome selection methods and genetic operators. Chapters 3 to 5 will discuss the single-robot problem, the two-robot problem, and present the novel solution techniques developed within the framework of this thesis.
3. THE SINGLE-ROBOT PROBLEM

3.1. Introduction

In Chapter 2, the overall problem addressed in this thesis was defined and a solution approach was proposed. All the aspects of the implementation of the Genetic Algorithm (GA) were discussed therein with the exception of the detailed explanation of the objective function. This core function is discussed in this chapter for the single-robot problem. Several simulations will allow us to illustrate the effects of various parameters on the efficiency of the process.

3.2. The Objective Function

An objective function is required for the implementation of the GA. This function takes system parameter values and evaluates them to provide a fitness level. In this thesis, the goal is to minimize the total assembly time for PCB population. The GA is set up to minimize assembly time, thus, the cycle time is used directly as the basis for the fitness score of a given genome.

3.2.1. Objective Function Input Variables

Two main types of input data for the objective function exist. The first type of data is the constant, system specific geometry and device properties data. This data is known to the objective function a priori. The second type of data received by the
objective function is the information contained in the genome, namely, task-specific parameter values.

(i) Constant Input Data

The constant input data consists of all system properties, all device properties, and any other configuration information relevant to the simulation. This data is read into the simulation from various configuration data files. The data can be categorized into the following groups: system data, board data, CDS data, robot data and component data.

System data affects the entire system and is not specifically associated with a particular device. The system data for the one-robot problem is the number-of-bits parameter, and the CDS-allocation flag. The number-of-bits parameter defines how many bits are used to encode rendezvous information. The CDS allocation flag is set either to dynamic or user-defined. If the flag is set to user-defined, the constant CDS data for component types and component locations is used. Otherwise, for dynamic CDS allocation, data is taken from the CDS allocation sub-genome, together with the starting offset and bin offset to calculate CDS assignments and component positions.

Board data is specific to the X-Y table and PCB. The data includes: board size (length and width), X-Y table speed, the table start point (fixed or variable), the table end point, workspace limits, table operation mode, number of components and component types and locations on the PCB. A device speed is simply the maximum value at which the device can move. It is used with the standard kinematic equations to calculate time from distance.
Each CDS has associated data including: size (length and width), speed, start point (fixed or variable), the end point, workspace limits, operation mode, number and types of components, location of components of the CDS, and starting-offset and bin-offset variables.

The component type and location data apply to the PCB and the CDSs. The PCB or CDS is described by a list of components, equal in length to the number of components. Each entry contains an identifier, a component type and an X-Y position. Since a PCB can contain the same type of components multiple times, the identifier is needed to make each component location unique for the choice of placement order. The position consists of an X-Y co-ordinate defined with respect to the origin of the device. The descriptions of the CDSs and PCB are used together with genome information to generate paths for each of the devices.

The starting-offset and bin-offset parameters are used when dynamic CDS allocation is chosen. The starting offset is a vector giving the position of the first bin in a dynamically generated CDS defined in co-ordinates with respect to the CDS origin. The bin offset is the offset between successive bins in a CDS. It is also given in a vector form. These two parameters are used together with CDS allocation sub-genome data if dynamic CDS allocation mode is chosen.
Robot data includes: speed, start point (fixed or variable), end point and workspace limits. The start and end points of a device are variables that can be set to a user-specified value or can be set to be dynamically defined.

The start point is location of the device at the start of the assembly of the PCB, and the end point is the location it moves to before the assembly is considered complete. When the start point is set to be dynamically defined, the start position of the device is the first optimum location determined by the GA. For the robot and the CDS, the dynamic start point is the first pick point for the device. For the X-Y table, the dynamic start point is the first placement location. When dynamic end points are chosen, the end point is simply the last pick or placement location the device moved to. For the CDS, this is the last pick point, and for the robot and the X-Y table this is the last placement point.

The workspace limits for a given device are the dimensions of the workspace within which the given device can move. In the case of the CDS and the PCB, these limits are used together with size to calculate the feasible workspace. This range is then used together with the number of bits to convert binary rendezvous data into real numbers.

Component data is related to component properties. In this thesis, however, the only relevant properties considered are the pick and placement times for each component. For each component different pick and placement time can be specified.
Above, the device operation mode applies to the CDSs and X-Y table. This mode is defined as one of three allowable setups. In Mode 1, the device is allowed to move freely and the GA optimizes each individual position for each pick or placement event for the device. In Mode 2, the device is fixed at an optimal location determined by the GA for the entire assembly process. In Mode 3, the device is fixed at a user-specified location. The operation mode at its current setting allows individual axis settings and, thus, serves two primary purposes: (1) physically it allows us to investigate different individual degrees of freedom in the system and (2) regarding software, it allows us to optimize the length of the rendezvous sub-genome, thereby, reducing computation time. The device modes, therefore, are used to define the size of the rendezvous sub-genome, and are also used together with number of bits and workspace limits to decode the rendezvous sub-genome into a series of co-ordinates.

(ii) Genome Input Data

A genome consists of four parts (as described in Chapter 2): the sequencing sub-genome, the CDS allocation sub-genome, the split-point sub-genome, and the rendezvous point sub-genome.

The sequencing sub-genome is a string of unique integers. It requires no decoding. Its values are used directly to determine the sequence, since the identifiers of the components on PCB are assigned to match the possible numbers in the sequencing sub-genome. Each number in the sequence represents one assembly cycle (see Chapter 2) for the component with the matching identifier.
The CDS-allocation and split-point data is only used if the dynamic CDS allocation mode is used. If the user runs the GA with user-defined CDS allocation, these sub-genomes are ignored. Like the sequencing sub-genome, the CDS allocation sub-genome is also a string of unique integers. It does not have to be decoded, but it has to be split into one or two CDSs depending on the value of the split-point sub-genome.

The split-point sub-genome consists of a single integer between zero and the length of the CDS allocation sub-genome. Its value specifies the point at which the CDS allocation sub-genome is split into two arrays. These arrays are then used as the basis for assigning components to the CDS bins. It is possible to have only one array, if the split-point is zero, resulting in only one CDS instead of two. The sequence of the numbers in each array determines which bin the component is in, starting with the first element of the array being placed closest to the CDS origin, with each successive component (as identified by the arrays) placed in the next bin. The bin positions are calculated with respect to the CDS origin, with the first bin located away from the origin by the starting offset, and each successive component located away from the previous component by the bin offset. The final step of this conversion is to overwrite the CDS component type and component locations for all of the CDSs with the dynamically attained values.

The rendezvous-point sub-genome when decoded comprises one placement array, and one or two (depending on the number of CDSs) pick array. This sub-genome requires the most decoding. The first conversion is to determine, based on the operation mode, how many values are encoded. The next step is to take the workspace limits and
sizes of the devices to calculate the range of possible values for each of the values. Subsequently, the number of bits together with a conversion template (depending on the operation mode) and the ranges calculated are used to convert the binary rendezvous sub-genome into a string of real numbers. These real numbers are arranged into the placement and pick arrays, where each array contains a series of vectors. The rendezvous placement array represents the positions of the X-Y table for each of the placement events (one per cycle). Each vector contains one placement position. Each CDS has a corresponding pick array which represents the positions of the CDS for each of its assigned pick events. From the combination of pick and placement arrays all of the device paths are calculated with constant data including CDS and PCB component locations. The device paths comprise the X-Y table path, the CDS paths and the robot path.

The X-Y table path is calculated from the placement array, the sequencing sub-genome and the board start and end points. First, the placement array is rearranged into the order specified by sequencing sub-genome. Then, the start point is added to the front of the array and the end point is added to the back of the array. The overall resulting array is the X-Y table path in the placement sequence order.

Each CDS path is determined individually. For a given CDS, the path is calculated from the related pick array, the sequencing sub-genome, the PCB component type information, the CDS start and end points. For each component in the sequence, using the sequence identifier, the PCB component-type information is accessed to
determine the type of the component. Using this information, the location of the component type in the CDS is determined. Then, the location of the first dynamic point in the CDS path array is found. The process is repeated until all dynamic points have been put into one of the two respective CDS paths. The final step for each CDS is to add the respective CDS start point onto the front of the CDS path array, and the CDS end point onto the back of the array. The resulting arrays are the CDS paths.

To calculate the robot path, the pick arrays, the placement array, the CDSs component type arrays, the CDSs component location arrays, the PCB component type information, the robot start and end points have to be known. The robot path essentially consists of alternating pick and placement events, at alternating pick and placement locations. These pick and placement locations naturally depend on the positions of the CDSs and the X-Y table. A pick location of the robot consists of the CDS position for that component plus the total offset of the corresponding component bin. As described earlier, the component position is known with respect to the CDS origin.

To calculate a robot pick point, the objective function starts with the sequence and identifies the component being placed in this cycle. Once this is known, the PCB component type information is accessed and the component type is identified. Using the component type, the correct CDS and corresponding bin is located. Once this is known, the first dynamic robot-path point is calculated by adding the rendezvous-pick-point value and the component-location point.
The second dynamic point in the robot path is the corresponding placement location for the cycle. The process is essentially the same as for the pick location using slightly different data. First, using the placement sequence, the objective function identifies the component on the PCB. Using this information, the board component location and the rendezvous placement location are determined. By adding these two values, the robot placement location is determined. The process of calculating the robot pick and placement points is repeated for each cycle until all pick and placement events have been completed. At this point, the only parameters which remain to be added are the robot start and end points.

3.2.2. Calculating Motion Times

Once the various device paths have been determined, the next step is to convert these paths into time values. For each cycle, all device positions are known by the objective function. To convert this collection of positions into time, one needs to know the speed and path of each device from one point to the next. Herein, it is assumed that all devices move in a straight line between two points at the devices’ maximum allowable speed. The devices accelerate instantaneously to maximum speed.

Figure 3.1 shows the paths of each of the devices during a pick-and-place operation.

From Figure 3.1, one can see that there are four motion times that need to be calculated from the position information contained in the genome: (i) The robot motion

Figure 3.1: Illustration of Cyclic Device Motion Times.

(i) The robot starts its motion at the previous placement location, PL_{i-1}, and moves to rendezvous with the CDS at the current pick location, PK_i. The value of \( r t_i \) is calculated simply by finding the linear distance between PL_{i-1} and PK_i, and then dividing it by the robot's maximum speed.

(ii) The CDS starts at the previous pick location, PK_{i-1}, and moves to the current pick location, PK_i. The value of \( dt_i \) is calculated by simply dividing the linear distance between the two points by the CDS' maximum speed.
(iii) The robot subsequently moves from the current pick location, PK_i, to the rendezvous location with the X-Y table, at the current placement location, PL_i. The value of \( t_i \) is calculated by dividing the distance by the robot’s maximum speed.

(iv) The final time to calculate is the X-Y table’s motion time-to-placement-location. The value of \( t_i \) is calculated by dividing the linear distance between PL_{i-1} to PL_i by the X-Y table’s maximum speed.

3.2.3. The Governing Assembly-Time Equation

From the motion times calculated above, the next step is to formulate the objective function. The cycle must be further refined in order to do this. This section will explain the calculation of the total assembly-time step by step.

The overall assembly time for a complete population of a PCB, \( t \), is calculated herein as follows:

\[
t = \sum_{i=1}^{N} C_i ,
\]

(3.1)

where \( C_i \) is the time it takes to complete cycle \( i \) and \( N \) is the number of components on the PCB.
The time equation for a single cycle must be defined with respect to a frame of reference. Herein, the start of the cycle is chosen to be the instant when the previous component insertion operation has been completed.

Figure 3.2 shows the individual device motion times for a single cycle. The plot for each device gives time on the X axis, and on/off status on the Y axis. Examining cycle $C_i$ for the robot, one sees that the robot is continuously utilized. It is always in motion in the X-Y domain except during the pick and placement operations. The first segment of the cycle is the robot motion-to-pick-location time, $r_{ti}$. The next segment is the component-pick-up time, $c_{ti}$, which is the time it takes for the component to be picked up from the CDS by the robot. The robot is not moving in the X-Y plane at this point, and neither is the CDS. Examining the CDS plot in parallel one sees that the component-pick-up time is considered as a concurrent action. The third segment is the robot motion-to-placement-location time, $r_{pi}$. The final segment of the cycle $C_i$ for the robot is the component placement time, $c_{pi}$, during which both the robot and the X-Y table are motionless in the X-Y domain.

The second plot in Figure 3.2 is for the CDS. Since the CDS becomes free to move to the next location in cycle $C_{i-1}$ after the $(i-1)$th component pick up, it starts moving right away. The amount of time it requires to complete its movement in cycle $C_i$ is the time it takes to move to the pick location, $d_{ti}$, minus the time it has had off in the last cycle, $off_{d1}$. Since the CDS may have had an off time in the last cycle, it is shown as idle for part of the time before the robot arrives at the rendezvous location for the $i$th
part. The next segment is matched with the robot. After that, the CDS immediately starts moving toward its next rendezvous location in the sequence.

![Diagram of X-Y Table, CDS, and Robot]

Figure 3.2: An Exemplary Time-Line Plot of a Single Cycle.

The X-Y table plot in Figure 3.2 closely resembles the CDS plot. The first segment consists of the X-Y table's motion time-to-placement-location, \( t_i \). In the next segment, the X-Y table is assumed to be idle for a while, since it has had the opportunity to start its motion toward the placement location ahead of the robot. The robot is unable to start moving toward the placement location until after the component pick operation has been completed. This advantage that the X-Y table has over the robot is represented by the time it has had off, \( \text{off}_{t_i} \). The final segment in the X-Y table plot is the component placement time, when the X-Y table is motionless at the same time as the robot in the X-Y domain.
Examining Figure 3.2, one can note that the cycle $C_i$ can be divided into two parts: (i) The time before the end of component pick operation, and (ii) the time after the component pick operation.

(i) The Time Before the End of the Component-Pick Operation

The first part of the cycle consists of the time before end of the component-pick operation. The limiting event is that both the robot and the corresponding CDS have to be at the pick location before the pick operation can proceed. Therefore, the maximum of the CDS' and robot's motion time to pick location must be considered as the overall pre-pick time.

The robot motion time-to-pick-location with respect to the start of cycle $C_i$ is simply $p_i t_i$. For the CDS, on the other hand, the portion of movement that occurs within cycle $C_i$ is of interest. Thus, it is necessary to correct the CDS motion time to pick location, $d t_i$, by subtracting the time the CDS has had off in the last cycle, $\text{off} t_i$.

To complete the calculation of the first part of the cycle, the actual component-pick time, $p_i c_i$ has to be considered as well. Thus, the first part of the cycle time, $p_{\text{pick}} C_i$ is:

$$p_{\text{pick}} C_i = \max[p_i t_i, (d t_i - \text{off} t_i)] + p_i c_i. \quad (3.2)$$
The CDS off time \((t_{off_i})\) is essentially the second part of the last cycle in which the CDS was utilized. Therefore, the second part of the cycle will be explained before this CDS-off time.

(ii) The Cycle After the Component Pick Up Operation

The second part of the cycle \(C_i\) is the time after the component-pick operation leading up to the component-placement operation. This time is calculated with respect to the last known motion of the robot, which is the end of the \(i\)’th component-pick operation. Namely, in order to determine the minimum time for the second part of the cycle, one has to determine the maximum of the time it takes the robot to move to the placement location, \(t_{pl_i}\), and the time it takes the X-Y table to move to the same location, \(t_{off_i}\).

The X-Y table motion time-to-placement-location has to be corrected, the table’s off time has to be subtracted. The actual component-placement time, \(t_{pl_i}\), has to be added to the cycle time. Therefore, the second part of the cycle, \(t_{place_i}\), is:

\[
place_i = \max\left[p_{pl_i}, (t_i - t_{off_i})\right] + t_{pl_i}.
\]  

(3.3)

Above, to calculate the X-Y-table-off time in cycle \(C_i\), one has to examine the first part of the cycle. The X-Y-table-off time is equal to the complete first part of the cycle \(t_{pick_i}\), namely:
Also, to calculate the CDS-off time in cycle $C_i$, one has to determine the last time the CDS was picked from. If the CDS was used in the last cycle $C_{i-1}$, where the $i$'th component is picked from the same CDS as the $(i-1)$'th component, then, the off time is equal to the second part of the previous cycle, $\text{place}_{C_{i-1}}$:

$$t_{\text{off},i} = \max\left[p_{i_i}^{\text{pl}}, t_{i-1} - \text{off}_{t_{i-1}}\right] T_{i-1}^{\text{pl}} C_{i-1}$$

(3.5)

Otherwise, if the component is picked from a different CDS, the off time has to include the additional cycles the CDS had time off. Then, the overall off time is equal to the second part of the last cycle the CDS was picked from, plus all cycle times up to the current cycle that the CDS was off:

$$t_{\text{off},i} = \sum_{j=k+1}^{i-1} C_j + \max\left[p_{i_i}^{\text{pl}}, t_{k} - \text{off}_{t_{k}}\right] T_{k}^{\text{pl}} C_{k}$$

(3.6)

where the index $k$ in the summation corresponds to the last component picked from the CDS under consideration.

The complete equation for a single cycle, $C_i$ is then the sum of the first part of the cycle, $\text{pick}_{C_i}$, and the second part of the cycle, $\text{place}_{C_i}$:
\[ C_i = \max[a_i, t_i, (d_i - \alpha_i t_i)] + \max[b_i, t_i, (t_i - \alpha_i t_i)] + \beta_i c_i + \pi_i. \] (3.7)

3.3. Methodology

With the above-defined objective function, the simulation tool ECPSIM (Electronic Component Placement SIMulator), based on a GA was written in C++. ECPSIM was developed using Wall's [23] GALib as a starting point. The goal in writing ECPSIM was to create a flexible tool to model any single-placement-robot point-to-point motion problem.

3.3.1. Simulation Capabilities

With the above definition of the objective function and its various input parameters, it is possible to model almost any planar co-operative robot system, consisting of: one central robot (the placement robot) that interacts with all other devices in the system, one receiving robot (the X-Y table) that acts as a destination for the central robot, and up to two provider devices that supply the central robot with parts.

The constant parameters described above are only constant for a single simulation, and the user of the simulation can vary them to define different problems. All of the geometry parameters are defined in input files and can be changed. Namely, it is possible to model almost any geometry with devices located at user-specified locations. The number of components and the information regarding their positions and types on the PCB and in the CDSs are also modifiable, allowing any PCB to be modelled.
The mode parameter described above allows any of the CDS or X-Y table axis motion behaviours to be defined independently, allowing devices with different motion characteristics to be investigated. It is assumed that the robot always has full motion. This allows the investigation of the effect of the degrees-of-freedom parameters on the optimization process.

With this highly flexible simulation, it becomes possible to investigate almost any parameters related to a planar point-to-point motion planning problem. In this thesis, the primary interest is to better understand a PCB assembly process and how different configurations can improve the performance. The system chosen is modelled from the perspective of a flexible manufacturing system and is designed to allow the comparison of current systems with new proposed systems with higher degrees of freedom.

One of the goals of this thesis was to explore the effect of the introduction of more degrees of freedom to an assembly system. Therefore a simple set up which could be run in different modes was chosen. The system is detailed below in the presentation of the example. Five parameters and their effect on the optimization are explored herein: fixed-device position optimization, CDS motion configuration, X-Y-table-motion configuration, CDS component allocation, and an all moving configuration.

By exploring these parameters, an understanding of the effect of each device on the optimization process is gained. Device parameters including the dof of a device can be better understood. This understanding of the system will allow the identification of
key areas for system improvement, showing which system changes will result in the
greatest performance improvement return.

3.3.2. Software Configuration

ECPSIM was written in C++, consisting of several programming modules, and the GALib library files. The modules and library are compiled together into a single executable. To run ECPSIM certain command line arguments and data files are required.

The GALib contains the main GA loop. To use the GALib, one has to provide the objective function and the genome. The program data flow controlled by the main module essentially consists of the following sections: (i) The constant input routines which read in all user-defined parameters, creating the constant data objects including: the robot, X-Y table, CDS, component and general system parameters. These input routines are supported by modules containing file scanning functions, a test display function and data structures (ecparray, and vector2d). (ii) The genome initialization routines create the main genome. (iii) Then, the main GA is run, it creates a series of population members based on the genome. It then calls the objective function, passing it a current population member. (iv) The objective function then uses the information provided to it by the constant data objects and the population member (genome) to calculate a fitness value. The GA then continues to run with the population reproducing and being evaluated. When the GA finishes running, it passes the genome with the highest fitness to the output function. (v) The output function calls the objective function to evaluate the genome into more useful terms, and then writes the results to file.
The main.cc module contains the main program which calls all of the other functions. It contains the code that handles all of the command line arguments. It calls the input routines, the genome initialization routines, the GA, and finally the output function.

(i) Constant Data Input Routines

The ecpsystem.cc module contains general system information. It reads in the file system.txt (or any other filename as passed in the command line argument). The system objects data is available to all other objects.

The board.cc module contains the board class, which defines the board object containing all constant data about the PCB and X-Y table. The constructor of the board class reads in the board.txt data file using the scanstr routine, storing the information in standard C++ data types in the custom array class ecparray, and the custom vector class vector2d. The board class includes public functions to allow the objective function to access the data when required.

The cds.cc module is the same as the board class except that it contains CDS data, and its constructor reads in the data from either cdsdat0.txt or cdsdat1.txt. The CDS class is called twice to create two CDS objects.

The robot.cc module is like the board class except that it is for the robot. The robot object reads in its data from the robot.txt file, which contains all of the constant robot data.
The component.cc module contains the component class which reads in the component.txt file. This object contains all pick and placement times.

The ecarray.cc module contains a class definition for an array type. The array type is resizable, and handles all of the pointer issues associated with memory allocation.

The vector2d.cc module contains a class definition for a real vector type. It also contains vector related operations, and data conversion functions.

The scanstr.cc module contains a function for scanning in data from any of the above input files, ignoring blank lines and lines preceded by a double slash.

The test.cc module contains functions for displaying the system, robot, CDS and board objects when the program is run with the -t command argument. This module serves primarily to allow verification of proper file handling by the constant data objects.

(ii) Genome Data Initialization and Input Routines

The ecspاسele.cc module contains functions to generate allele sets, which are data sets passed to the ecpgenome2 class which define the possible values of each of the sub-genomes.

The popinit.cc module is only used if the init command argument is used. This module contains a function that rewrites some of the population members with user-
defined rather than random starting values. This allows previous GA results to be used as a starting point for a new run.

The ecpgenome2.cc module contains the ecpgenom2 class. This class defines all aspects of the genome including all of its sub-genomes. It is used by the GA as a stencil for creating new population members.

(iii) GALib Library Files

The GALib files include all of the functions to run a genetic algorithm, when provided with a genome class and an objective function. They also produce a statistics output file showing the mean, minimum, and maximum fitness values of the population at a user-defined sampling rate.

(iv) Objective Function Routines

The max.cc module contains a simple function that returns the maximum of two values passed to it. It is used by the objective function.

The congen2rdv.cc module contains a number of conversion functions that are called by the objective function to convert the binary rendezvous sub-genome into real numbers, and then group them into the pick and placement arrays.
The ecpobj.cc module contains the objective function. This function calls on all of the constant data objects, the genome and congen2rdv.cc to calculate the fitness of a population member passed to it.

(v) Output Routine

The output.cc module contains a function that calls the objective function to evaluate the best genome, and then writes all genome information to data files in a usable form. It creates two data files, one containing all device paths, and one containing the time-line plot data.

3.4. An Example

To illustrate the methodology described above, the simulation tool ECPSIM is used to test various system configurations which are described in the next sub-sections.

3.4.1. The Set Up

A simple PCB population sequence of six components is optimized. The general system set up described in Chapter 2, is used. The system consists of one placement robot, two CDSs and the X-Y table.

The 2-dof gantry-type placement robot can move at a maximum speed of 2 m/s. Its workspace extends from -10 to 310 mm in the X direction, and from 0 to 300mm in the Y direction, to included both the CDSs’ and the X-Y table’s workspaces. The robot is
assumed to have dynamic start and end locations. It starts at the first component pick location and finishes at the last placement location. The robot is modelled as a point end effector with zero size.

The CDSs move at maximum speed of 1 m/s (Y axis). The first CDS moves along the line \( x = -10 \) and the second CDS moves along the line \( x = 310 \). The workspace of each CDS extends from 0 to 300 mm in the Y axis. The CDSs start at the first pick location and end at their last pick location. The size and component assignments depend on whether a dynamic-CDS-component allocation or a user-specified component allocation is used.

Figure 3.3 shows the configuration of the two CDSs for a user-specified CDS component allocation. Each CDS consists of a bin with three compartments spaced by 60 mm. The CDSs are modelled with a size of 0 mm x 120 mm. The components are numbered from zero to five and are distributed between the two CDSs, with components 0 to 2 in CDS 1 and components 3 to 5 in CDS 2.

For dynamic CDS component allocation, the number of components in each CDS are determined from the CDS allocation sub-genome and the split-point sub-genome. The CDS is configured out of 0 mm x 60 mm bins which are stacked together in the Y direction until the required number of components have been accommodated.
Component Pick Locations in Device Coordinates

<table>
<thead>
<tr>
<th>Component Number</th>
<th>Pick Position on CDS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$x_1$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$x_2$</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 3.3: One of the Two Identical CDSs.

The 100 mm x 100 mm X-Y table moves at a maximum speed of 0.5 m/s (both axes). The workspace of the X-Y table is 300 mm x 300 mm. The table starts at the first placement location and finishes at the last placement location.

The PCB on the X-Y table is of size 100mm x 100mm, Figure 3.4.

Component Placement Locations in Device Coordinates

<table>
<thead>
<tr>
<th>Component Number</th>
<th>Place Position on PCB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$x$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>75</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
</tr>
</tbody>
</table>

Figure 3.4: The PCB With Component Positions.
3.4.2. Problem Configurations

A series of different configurations are explored:

(i) All devices are fixed at user-specified locations, where only the placement robot is allowed to move.

(ii) All devices are fixed at GA-optimized locations, where only the placement robot is allowed to move.

(iii) The X-Y table is fixed at a GA-optimized location, where the CDSs and the placement robot are allowed to move.

(iv) The CDSs are fixed at GA-optimized locations, where the X-Y table and the placement robot are allowed to move.

(v) All devices are free to move.

All above five configurations were tested with user-specified CDS allocation as well as with dynamic CDS allocation modes.

(i) Devices Fixed at Non-Optimal Locations

The X-Y table is fixed at the center of the workspace, with its origin located at (100,100). Similarly the CDSs are fixed in the middle of their respective workspaces, with their origin fixed at (-10,90) for CDS 1 and at (310,90) for CDS 2, respectively. The robot is free to move with every pick or placement event. This configuration serves as a comparison case, where the GA optimizes the sequence of the component placement. This is the classic TSP problem.
(ii) Devices Fixed at Optimal Locations

In this case, the X-Y table is fixed at an optimal location, determined by the GA at the beginning of the assembly process, and is not allowed to move. The CDSs are also fixed at optimal locations determined by the GA. The robot is free to move. The goal here is to show that even with current PCB placement technology a carefully chosen setup can lead to time savings.

(iii) X-Y Table Fixed at an Optimal Location

The degrees of freedom (dof) of the system is gradually increased to show the effect on the performance of the system. The CDSs are allowed to move to GA-determined locations for each pick event. The X-Y table is still fixed at a GA-optimized location and the placement robot is free to move.

(iv) CDSs Fixed at Optimal Locations

Here, the X-Y table is allowed to move. It is sent to optimal positions determined by the GA. The CDSs are fixed at GA optimized locations and the robot is free to move.

(v) All Devices Moving

This is the fully optimized case (within the limits of the simulation) where all devices move to GA-optimized rendezvous locations.
3.4.3. Individual Simulation Results

Each of the above problems were run 25 times using ECPSIM. The best results are given for each case, both for user-specified CDS allocation and for dynamic CDS allocation.

(i) Devices Fixed at Non-Optimal Locations

This run yielded a minimum total time of 1.621 s with a placement sequence of (5, 3, 4, 0, 1, 2). Figure 3.5 shows the path that the placement robot took. The position of the X-Y table outline is shown by the dashed-line box.

![Figure 3.5: Robot Path - Case 1 - User-Defined-CDS-Component Allocation.](image)

Figure 3.6 shows the activities of each of the devices in a series of time-line plots. The first plot shows the robot picking up a component, moving, placing it, and then moving to the next pick location, starting the cycle over again. The CDSs positions are
both fixed, thus, their time-line plots only show the pick events. The X-Y table position is fixed as well, with its time-line plot only showing the component placement times.

Convergence data were also generated, with the population mean, minimum and maximum being recorded every one hundred generations, and are shown in Figure 3.7. It can be seen that the population converges almost immediately to a steady value for this case.

![Graphs of Robot, Component Delivery System 1, Component Delivery System 2, and X-Y Table Time Lines]

Figure 3.6: Time Line - Case 1 - User-Defined-CDS-Component Allocation.

For the example above, when the component assignment is carried out dynamically by the GA a total time of 1.596 s is obtained for the placement sequence of (0, 2, 1, 5, 3, 4), Figure 3.8 shows the robot path. This time is slightly shorter than the
case with user-defined CDS allocation of components. (See Appendix A for more details).

Figure 3.7: Convergence Plot - Case 1 - User-Defined-CDS-Component Allocation.

Figure 3.8: Robot Path - Case 1 - Dynamic-CDS-Component Allocation.
(ii) Devices Fixed at Optimal Locations

With each of the devices fixed at a GA-determined optimal location, this run generated a minimum total time of 1.582 s, with a sequence (3, 5, 4, 0, 1, 2). Figure 3.9 shows the robot path.

Figure 3.10 shows the on-off times of the various devices. The robot is continuously busy. This suggests that the robot is the limiting device in the system, which is logical since the other devices are fixed.

Figure 3.9: Robot Path - Case 2 - User-Defined-CDS-Component Allocation.
As shown in Figure 3.11, the GA once again converges rapidly.

For the above example, when the dynamic CDS component allocation was utilized, a total time of 1.336 s is obtained for a placement sequence of (0, 3, 4, 1, 5, 2). Figure 3.12 shows the robot path. (See Appendix A for more details).
Figure 3.11: Convergence Plot - Case 2 - User-Defined-CDS-Component Allocation.

Figure 3.12: Robot Path - Case 2 - Dynamic-CDS-Component Allocation.
(iii) X-Y Table Fixed at An Optimal Location

This example illustrates the performance improvement achieved via the motion of the CDS. In this run, the total time was 1.562 s with a placement sequence of (5, 3, 4, 0, 1, 2). Figure 3.13 shows the robot path.

![Diagram](image)

Figure 3.13: Robot Path - Case 3 - User-Defined-CDS-Component Allocation.

Comparing Figure 3.9, where the CDSs are fixed, with Figure 3.13, where the CDSs are moving, one can note that the pick points in Figure 3.13 are closer together. Namely, the CDSs position themselves in such a way as to produce the shortest path between the current placement point and the next pick point and shortest path between the next pick point and the next placement point.

Examining the time lines in Figure 3.14, it can be seen that the robot is still continuously busy. Also, the CDSs are moving to the next pick position immediately.
after the previous pick event. The distance the CDSs must travel, when compared to the robot, are substantially shorter, therefore, the CDSs stop moving well ahead of the robot as they arrive at their rendezvous locations. Since the distances travelled by the CDSs are relatively small, only a relatively small time saving is achieved over the case with the optimally fixed CDS.

![Diagram](image)

**Figure 3.14: Time Line - Case 3 - User-Defined-CDS-Component Allocation.**

Figure 3.15 shows that with more degrees of freedom, the problem is taking longer to converge than before. The GA however still rapidly converges close to the final solution in the first 1000 cycles, with a minor adjustment occurring at approximately 10,000 cycles.

3-35
For the above example, when the CDS component allocation is carried out dynamically by the GA, a total time of 1.216 s is obtained for a placement sequence of (0, 3, 4, 1, 5, 2). Figure 3.16 shows the robot path. (See Appendix A for more details).

Figure 3.15: Convergence Plot - Case 3 - User-Defined-CDS-Component Allocation.

Figure 3.16: Robot Path - Case 3 - Dynamic-CDS-Component Allocation.
(iv) CDSs Fixed at Optimal Locations

In this case, the effect of the X-Y table motion on the optimization process is illustrated. The total time was 1.251 s with a placement sequence (5, 4, 3, 0, 1, 2). Figure 3.17 shows the robot and X-Y table paths.

The robot path is shown as the solid line, where the X-Y table path is only shown by the PCB outline at each of the placement locations. One can see that there is substantial movement by the X-Y table, and that the entire form of the robot path changes when compared to Figure 3.9, where the X-Y table is fixed.

![Robot Path - Case 4 - User-Defined-CDS-Component Allocation](image)

Figure 3.17: Robot Path - Case 4 - User-Defined-CDS-Component Allocation.

Figure 3.18 shows the time lines of the various devices. The CDSs are fixed, and hence only the pick times are shown. The robot is continuously busy. The interesting point to note in this time line is the motion of the X-Y table. From Figure 3.17, after component 4 is placed, the X-Y table starts to cross the workspace toward the other CDS.
From Figure 3.18, the X-Y table moves until the robot catches up with it, and then allows Component 3 to be placed. It then immediately resumes its crossing, stopping again only when the robot catches up once again for the placement of Component 0. This motion results in assembly time savings.

![Figure 3.18: Time Line - Case 4 - User-Defined-CDS-Component Allocation.](image)

Figure 3.18 shows that the GA takes longer to converge. This is consistent with the increase in the number of variables being optimized. The GA does not fully converge until after 1000 cycles, with further improvements occurring in the remaining cycles.

For the above example, when the CDS component allocation is carried out dynamically by the GA, a total time of 1.240 s is obtained for a placement sequence of (0, 2, 1, 5, 3, 4). Figure 3.20 shows the robot plot. (See Appendix A for more details).
Figure 3.19: Convergence Plot - Case 4 - User-Defined-CDS-Component Allocation.

Figure 3.20: Robot Path - Case 4 - Dynamic-CDS-Component Allocation.
(v) All Devices Moving

In the final case examined, all devices can move and interact to produce the optimal time. This simulation resulted in a total time of 1.205 s for a placement sequence of (5, 3, 4, 0, 1, 2). Figure 3.21 shows the robot path.

Figure 3.21: Robot Path - Case 5 - User-Defined-CDS-Component Allocation.

From Figure 3.22, one can note that the robot is continuously busy. But the CDSs have only a limited role in determining the maximum time, since they complete all of their motions before the robot or the X-Y table arrive at their rendezvous locations. Comparing Figure 3.18 and Figure 3.22, one notes that the X-Y table is substantially busier in Figure 3.22, being almost constantly in motion. The CDSs and X-Y table appear to have a higher interaction to reduce the robot’s Y-travel distance.
The total time improvement achieved via all devices moving over all devices being fixed at optimal locations is 0.377 s (approximately 25%). This improvement may be attributed to co-operation between the CDSs and X-Y table.

Figure 3.23 shows the convergence data for the all-devices-moving case. This particular run was started with a genome generated from a previous GA run. Thus, the initial minimum value is much lower than normal. The GA converges rapidly to a value close to the optimum.
Figure 3.23: Convergence Plot - Case 5 - User-Defined-CDS-Component Allocation.

For the above example, when the CDS component allocation is carried out dynamically by the GA, a total time of 1.205 s is obtained for a placement sequence of (5, 3, 4, 0, 1, 2). This result was accomplished by rerunning the simulation with starting data from the all-moving-user-defined case. The result is the same as the above example since the GA did not change the CDS component allocation.

Figure 3.22 shows that neither of the CDSs are fully occupied. Examining the user-defined CDS allocation, the components are separated into Components 0, 1, and 2, which are located on the left side of the PCB, and 3, 4, and 5 which are located on the right side of the PCB. The dynamic CDS allocation for the CDSs-fixed-at-optimal-locations case (iv) also splits the data into these two groups. This suggests that the user-defined data is already optimized in this respect. The next level of component
sequencing at the CDS level can take place either through dynamic CDS allocation or through the motion of CDSs. Looking at the results in (iii) showing the case of the X-Y table fixed at an optimal location, with a user-defined CDS allocation, a sequence of (5, 3, 4) is achieved by CDS motion. (See Appendix A for more details).

3.4.4. Example Results Summary

Table 3.1 presents the overall assembly time and the corresponding optimal component-placement sequence, for each of the problem types, for user-defined CDS allocation. As expected, when more devices are allowed to move (dof increases) the overall assembly time is reduced.

<table>
<thead>
<tr>
<th>Problem Type</th>
<th>PCB on X-Y Table</th>
<th>Delivery Systems</th>
<th>Optimal Sequence</th>
<th>Total Time (s)</th>
<th>% Improvement w.r.t. the Non-Optimal Fixed Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Optimal Fixed</td>
<td>Fixed (Middle)</td>
<td>Fixed (Middle)</td>
<td>0 2 1 5 4 3</td>
<td>1.621</td>
<td>0.0</td>
</tr>
<tr>
<td>Optimally Fixed</td>
<td>Fixed (Optimal)</td>
<td>Fixed (Optimal)</td>
<td>3 5 4 0 1 2</td>
<td>1.582</td>
<td>2.4</td>
</tr>
<tr>
<td>CDS Moving</td>
<td>Fixed (Optimal)</td>
<td>Moving</td>
<td>5 3 4 0 1 2</td>
<td>1.563</td>
<td>3.6</td>
</tr>
<tr>
<td>X-Y Table Moving</td>
<td>Moving</td>
<td>Fixed (Optimal)</td>
<td>0 1 2 5 4 3</td>
<td>1.252</td>
<td>22.8</td>
</tr>
<tr>
<td>All Moving</td>
<td>Moving</td>
<td>Moving</td>
<td>5 3 4 0 1 2</td>
<td>1.205</td>
<td>25.7</td>
</tr>
</tbody>
</table>

Table 3.2 presents the overall assembly time, and the corresponding optimal component-placement sequence, for each of the problem types, for dynamic CDS
component allocation. It should be noted that for the optimally fixed case and for the CDS-moving case all components were placed in a single CDS by the dynamic CDS allocation sub-genome.

Table 3.2: The Simulation Results Summary for Dynamic CDS Allocation.

<table>
<thead>
<tr>
<th>Problem Type</th>
<th>PCB on X-Y Table</th>
<th>Delivery Systems</th>
<th>Optimal Sequence</th>
<th>Total Time (s)</th>
<th>% Improvement w.r.t. the Non-Optimal Fixed Case User-Defined CDS Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed (Middle)</td>
<td>Fixed (Middle)</td>
<td>021534</td>
<td>1.596</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>Optimally Fixed (Optimal)</td>
<td>Fixed (Optimal)</td>
<td>034152</td>
<td>1.336</td>
<td>17.6</td>
<td></td>
</tr>
<tr>
<td>Moving (Optimal)</td>
<td>Moving</td>
<td>034152</td>
<td>1.216</td>
<td>25.0</td>
<td></td>
</tr>
<tr>
<td>X-Y Table Moving</td>
<td>Fixed (Optimal)</td>
<td>021534</td>
<td>1.240</td>
<td>23.5</td>
<td></td>
</tr>
<tr>
<td>All Moving</td>
<td>Moving</td>
<td>534012</td>
<td>1.205</td>
<td>25.7</td>
<td></td>
</tr>
</tbody>
</table>

3.5. Summary

In this chapter, the GA objective function and the methodology for the single-robot TSP+ problem were defined. The methodology was then illustrated by testing it with various system configurations. Dynamic CDS-component allocation was also investigated. It was shown that the greater the dof of motion of the system, the better the overall performance. In the next chapter, the two-robot TSP+ problem will be presented.
4. THE TWO-ROBOT AUGMENTED TRAVELLING SALESPERSON PROBLEM

4.1. Introduction

In Chapter 2, the overall TSP+ problem was defined and a generic solution approach was proposed. The utilization of genetic algorithms (GA) was discussed with the exception of the specific description of the objective function. Chapter 3 presented the objective function for the single-robot TSP+ problem and solved the optimization problem for different system configurations. This chapter will first define the objective function for the two-robot case and then solve the TSP+ problem for the same system configurations as in Chapter 3. (The collision avoidance problem, however, will be only addressed in Chapter 5). As in Chapter 3, simulations will also be used herein to illustrate the proposed solution approach.

4.2. The Objective Function

The objective function evaluates a genome and assigns a fitness value to it. The GA, as with the one-robot problem, is set up to minimize assembly time. Therefore, assembly time is used directly as the basis for the fitness score.

4.2.1. Objective Function Input Variables

Two types of input data exist: A priori known constant input data and task-specific information contained in the genome.
(i) Constant Input Data

The constant input data consists of overall system properties, device properties, and other constant configuration information. (In this thesis, the input data is read from various files). The data is categorized into: system data, board data, CDS data, robot data and component data.

The system data affects the entire system and is not associated with a particular device. The system data includes the number of bits parameter, the CDS allocation flag and a collision avoidance flag. This is the same as for the single-robot problem, except for an additional collision avoidance flag. The collision avoidance flag informs the GA whether to use the collision avoidance routine. In this chapter, the flag is set to off and thus the routine is disabled.

Board data is specific to the X-Y table and the PCB. The board data remains the same as for the single-robot problem. It includes board size (length and width), X-Y table speed, table start point (fixed or variable), table end point, workspace limits, table operation mode, number of components and component types and locations on the PCB.

The CDS data is also the same as for the single-robot problem and each CDS has associated data including: size, speed, start point (fixed or variable), end point, workspace limits, operation mode, number and types of components, location of components of the CDS, and starting offset and bin offset variables.
For each robot, the following data is included: speed, start point (fixed or variable), end point and workspace limits. (The start and end points of a device are variables that either can be set to a user-specified value or can be dynamically defined. When the start point is set to be dynamically defined, the start position of the device is the first optimum location determined by the GA). The dynamic start points for the CDS and X-Y table are the same as for the single-robot problem, namely, the first pick and placement locations respectively. For the two robots, the dynamic start points are their first pick locations.

Dynamic end points for the CDS and X-Y table are the same as in the single-robot problem, namely, the last pick point and the last placement point, respectively. For the robots, the end points are the first pick locations, since the robots must return to the next pick location in order not to block the other robot’s access to the PCB.

The workspace limits for the CDS and X-Y table are the same as in the single-robot case. For the robots, the workspace of each robot includes the entire workspace of the X-Y table and its associated CDS.

Component data is the same as for the single-robot case, consisting of the pick-up and placement-operation times for each component.

The device operation modes apply to the CDSs and the X-Y table. The operating modes for devices are identical to those specified in Chapter 3. In Mode 1, the device is
allowed to move freely to GA-optimized positions. In Mode 2, the device is fixed at an optimal location determined by the GA, and in Mode 3, the device is fixed at a user-specified location.

(ii) Genome Input Data

In this thesis, a genome consists of four parts (as described in Chapter 2): The sequencing sub-genome, the robot-assignment sub-genome, the split-point sub-genome, and the rendezvous-point sub-genome.

The sequencing sub-genome is a string of unique integers that require no decoding. Same as for the single-robot problem, its values are used directly to determine the placement sequence.

The robot-assignment and split-point data are only used if the CDS allocation flag is set to dynamic. The robot-assignment sub-genome serves two purposes: (1) It assigns a given component to one robot or to the other, and (2) It assigns the components to the matching CDS of the robot. Like the sequencing sub-genome, the robot assignment sub-genome is also a string of unique integers. It does not have to be decoded, but it may or may not have to be split into two robots depending on the value of the split-point sub-genome.

The split-point sub-genome consists of a single integer between zero and the length of the robot-assignment sub-genome. Its value specifies the point at which the
robot-assignment sub-genome is split into two arrays. These arrays are then used as the basis for assigning components to each of the robots and their respective CDS bins.

Each robot only picks components from the CDS assigned to it. The optimization allows us to have only one array, if the split-point is zero, resulting in the utilization of only one robot and one CDS, instead of two. The sequence of the numbers in each array also determines which bin the components are in, starting with the first element of the array being placed closest to the CDS origin with each successive component (as identified by the arrays) placed in the next bin.

The bin positions are calculated with respect to the CDS origin, with the first bin located away from the origin by the starting offset, and each successive component located away from the previous component by the bin offset. The final step of this conversion is to overwrite the CDS component type and component locations for all of the CDSs with the dynamically attained values.

The handling of the rendezvous-point sub-genome is the same as for the single-robot problem, which, when decoded, comprises one placement array and one or two (depending on the number of CDSs) pick array(s). This sub-genome is decoded from binary into real numbers using the number of bits parameter together with a conversion template (depending on the operation mode) and the workspace ranges. These real numbers are then rearranged into the placement and pick arrays, which form the basis of the X-Y table placement positions and the CDS pick positions. From the pick and
placement arrays together, with constant data including CDS and PCB component locations, all of the device paths are calculated. The device paths comprise the X-Y table, the CDS and the robot paths.

The X-Y table and the CDS paths are calculated in the same manner described for the single-robot problem. First, the pick or placement array is rearranged according to the sequencing sub-genome parameters, and then, device start and end points are added to the path arrays (for details see Chapter 3).

For each of the robots, the process for determining the corresponding path requires the knowledge of the pick and placement arrays, the robot-assignment sub-genome, the CDSs component type arrays, the CDSs component location arrays, the PCB component type information, and the robot start and end points. First, for each robot, a sub-sequence of its movements to pick and place the components assigned to it by the robot-assignment sub-genome is generated. This robot sub-sequence consists of the components assigned to the robot as they appear in the overall sequencing sub-genome. For example, if the overall sequence is (5, 1, 3, 0, 4, 2) and Robot 1 is assigned to place components (0, 1, 2), then, Robot 1’s sub-sequence is (1, 0, 2). Similarly, if Robot 2 is assigned to place components (3, 4, 5), then, Robot 2’s sub-sequence is (5, 3, 4).

Following the above, for a given robot, the corresponding path determination procedure is similar to that of the single-robot problem, using the robot sub-sequence
(instead of the overall sequence) for determining the component identity. The robot path consists of alternating pick and placement events, at alternating pick and placement locations. These pick and placement locations depend on the positions of the CDSs and the X-Y table. In order determine a robot pick point, the objective function starts with the robot sub-sequence and identifies the component being placed in this cycle. Once this is known, the PCB component type information is accessed and the component type is identified. Subsequently, the correct CDS and corresponding bin is located. Then, the first dynamic robot path point is calculated by adding the rendezvous pick point value and the component location point together.

The second point in the robot's path is the corresponding placement location for the cycle. The process is essentially the same as for the pick location, but using different data. First, using the robot sub-sequence, the objective function identifies the component on the PCB. Then, using this information, the board component location and the rendezvous placement location are determined. By adding the board component location and the rendezvous placement location together, the robot placement location is determined. The process of calculating the robot pick and placement points is repeated for each cycle until all pick and placement events have been completed. At this point, the only parameters which remain to be added to beginning and end of the robot path array are the robot start and end points.

The above process is repeated for the second robot.
4.2.2. Calculating Motion Times

Once the various device paths have been determined, the next step is to convert these paths into time values. For each cycle, all device positions are known by the objective function. To convert this collection of positions into time, one needs to know the speed and path of each device from one point to the next. Figure 3.1 shows the paths of each of the devices during a typical pick-and-place operation.

The overall assembly time for a complete population of a PCB, \( t \), is calculated herein as follows:

\[
t = \sum_{i=1}^{N} C_i ,
\]

where \( C_i \) is the time it takes to complete cycle \( i \) and \( N \) is the number of components on the PCB. The time equation for a single cycle must be defined with respect to a time
frame of reference. Herein, the start of the cycle is chosen to be the instant when the previous component insertion operation has been completed.

Figure 4.1 shows exemplary individual device motion-time plots for a single cycle. Even though Figure 4.1 shows all the devices, it is important to recall that, for cycle $C_i$, Robot 2 and CDS 2 are the active devices, and Robot 1 and CDS 1 are only shown to illustrate how robot interaction can lead to off times.

The plot for each device gives the time on the X axis, and on/off status on the Y axis. Since Robot 2 is free to move to its next pick location after completing cycle $C_{i-2}$ (note that cycle $C_{i-1}$ is carried out by Robot 1), it starts its motion immediately. The amount of time Robot 2 requires to complete its pick movement in cycle $C_i$ is then the time it takes to move to the pick location, $p^*_i t_i$, minus the time it has had off, $o^*_i t_i$. The next segment is the component pick time, $p^c_i$, which is the time required for the component to be picked up from CDS 2 by Robot 2. Robot 2 is not moving in the X-Y plane during this time, and neither is CDS 2. Examining CDS 2's plot in parallel, one notices that the component pick time is considered as a concurrent event. The third segment of the Robot 2 plot is the motion time to placement location, $p^*_r t_i$. The final segment of the cycle $C_i$ for Robot 2 is the component placement time, $p^c_i$, during which both Robot 2 and the X-Y table are motionless in the X-Y domain. Examining cycle $C_i$ for Robot 2, one sees that, Robot 2 is continuously utilized (in cycle $C_i$). This robot is always in motion except during the pick and placement operations (shown by shaded areas).
Figure 4.1: An Exemplary Time-Line Plot of a Single Cycle.

Also included in Figure 4.1 is the time-line plot for CDS 2. Since CDS 2 becomes free to move to its next location for cycle $C_i$ after the $(i-2)$th component pick up, it starts moving right away. The amount of time it requires to complete its movement in cycle $C_i$ is then the time it takes to move to the pick location, $d t_1$, minus the time it has had off, $d t_1$. Since in this plot, the CDS 2 has had off time, it is shown as idle for part of the time before the robot arrives at the rendezvous location for the $i$'th part. The next segment of CDS 2's time-line plot is the component pick-up time that is matched with
that of Robot 2. After that, the CDS immediately starts moving toward its next rendezvous location in the sequence.

The X-Y table time-line plot is also shown in Figure 4.1. The first segment of cycle $C_i$ is the X-Y table's motion time to placement location, $t_i$. In the next segment, the X-Y table is idle since it arrives at the placement location before Robot 2. The last segment in the X-Y table plot is the component placement time, when the X-Y table is motionless, as is Robot 2, in the X-Y domain.

The cycle time $C_i$ is the maximum of Robot 2's time and the X-Y table time, plus the component placement time, $p_i C_i$. The robot time, $r_i$, is the time it takes the robot to carry out all the tasks required to move from the last placement location to be ready to place the next component at the current placement location. The X-Y table time, $t_i$, is the time it takes to move from the last placement location to the current placement location, as shown in Figure 3.1. Therefore, $C_i$ is:

$$C_i = \max[r_i, t_i] + p_i C_i.$$  \hspace{1cm} (4.1)

The robot cycle time, $r_i$, can be divided into: (i) The time before the pick operation, and (ii) The time after and including the pick operation.
(i) The robot time before the pick operation

The completion of the first part of the robot cycle time, \( r_{t_i}^{1st} \), depends on both the CDS' motion and the robot's motion to the pick location. Therefore, the first part of the robot cycle time is the maximum of the time the robot takes to reach the pick location and the time the CDS takes to reach it. The former is the robot motion time to pick location, \( r_{t_i}^{pk} \), minus the robot off time, \( r_{t_i}^{off} \). The latter is the CDS motion time to pick location, \( d_{t_i} \), minus the CDS off time, \( d_{t_i}^{off} \). Therefore,

\[
 r_{t_i}^{1st} = \max[r_{t_i}^{pk} - r_{t_i}^{off}, d_{t_i} - d_{t_i}^{off}].
\] (4.2)

The second term in Equation (4.2), \( r_{t_i}^{off} \), is the time the current robot has been off since its last placement operation. If the i'th component is placed by the same robot as the (i-1)'th component, then;

\[
 r_{t_i}^{off} = 0. \] (4.3)

Otherwise, it is placed by the other robot and;

\[
 r_{t_i}^{off} = \sum_{j=k+1}^{i-1} C_j, \] (4.4)

where the index \( k \) represents the last cycle in which the current robot moved.
The fourth term in Equation (4.2), $t_i^{d\text{off}}$, is the time period that the current CDS has not been involved in a pick operation and has had time to move toward its next pick location. In order to calculate $t_i^{d\text{off}}$, the total time of the last cycle in which the CDS was picked from, $C_k$, is taken, and the time the CDS was busy is subtracted from it. The CDS is busy for the first part of the robot cycle time, $t_k^{\text{int}}$, and the component pick time, $p_k c_k$. If the $i$'th component is picked from the same CDS as the $(i-1)$'th component, then, $k=i-1$ and:

$$t_i^{d\text{off}} = C_{i-1} - \max[(p_k t_{i-1} - t_{i-1}^{\text{off}}), (d t_{i-1} - t_{i-1}^{\text{off}})] - p_k c_{i-1}.$$  \hfill (4.5)

Otherwise, it is picked from the other CDS, where it is also necessary to add all the other cycles that the CDS has been off:

$$t_i^{d\text{off}} = \sum_{j \neq k}^{i-1} C_j - \max[(p_k t_k - t_k^{\text{off}}), (d t_k - t_k^{\text{off}})] - p_k c_k ,$$ \hfill (4.6)

where the index $k$ is the last cycle in which a component was picked from the CDS under consideration.
(ii) The robot time after and including the pick operation

The second part of the robot cycle time, \( R t^{2nd}_i \), depends on the pick operation time, \( p^k c_i \), and the robot motion time to placement location, \( p^l t_i \), which occur sequentially,

\[
R t^{2nd}_i = p^k c_i + p^l t_i .
\]  

Adding the first and second parts of the robot cycle time the overall robot motion time needed by Equation (4.1) is obtained as follows:

\[
R t_i = R t^{1st}_i + R t^{2nd}_i = \max[(r^k_t_i - r^t_i), (d t_i - d^t_i)] + p^k c_i + p^l t_i .
\]  

4.3. Two-Robot Problem Solution Methodology

For the above defined objective function, the GA-based simulation tool ECPSIM2R (Electronic Component Placement SIMulator for 2 Robots) was developed and utilized in this thesis. The goal in developing the ECPSIM2R was to create a flexible tool to model and solve any two-(placement)-robot point-to-point motion problem.

4.3.1. Simulation Capabilities

ECPSIM2R can be utilized to model planar co-operative systems comprising two-placement robots that interact with all other devices in the system, one receiving robotic
X-Y table that acts as a destination for the placement manipulators, and two mobile component provider devices.

All the run-time-constant parameters can be varied for different simulations, as were described for the single-robot case in Chapter 3, allowing any PCB assembly system configuration to be modelled. Similarly, the operating mode parameters can also be modified allowing devices with different motion characteristics to be investigated.

4.3.2. Software Configuration

The overall configuration of ECPSIM2R is identical to ECPSIM (described in Chapter 3) except that certain modules were replaced. These modules serve the same roles but are adapted to be able to handle two robots.

The main2r.cc module replaces main.cc, containing the main program and performing all of the functions main.cc performs for the single-robot problem. The primary change is that it calls the robot module twice, creating two robot objects instead of just one.

The ecpobj2.cc module replaces the ecpobj.cc module. This module is completely different and contains the above described two-robot objective function. However, its role in the program remains the same as that of the ecpobj.cc module in ECPSIM.
The output2.cc module replaces output.cc in performing a similar task of calling the objective function (ecpobj2.cc) and writing to two data files. The first file contains all the device paths, and the second file contains the time-line data and additional data for a robot interference plot.

4.4. A Simulation Example

In order to illustrate the methodology described above, ECPSIMZR was used to test various system configurations as described in the next sub-sections.

4.4.1. The Set Up

A simple PCB population sequence of six components is optimized (same as in Chapter 3). The general system set up described in Chapter 2 is used. The system consists of two placement robots, two CDSs and an X-Y table.

The 2-dof gantry type placement robots can move at a maximum speed of 2 m/s. Robot 1’s workspace extends from -10 to 300 mm in the X direction and from 0 to 300 mm in the Y direction, enclosing CDS 1’s and the X-Y table’s workspaces. Robot 2’s workspace extends from 0 to 310 mm in the X direction and from 0 to 300 mm in the Y direction, enclosing CDS 2’s and the X-Y table’s workspaces. The robots’ start and end locations are determined dynamically. They start and finish at their first component-pick locations. The robots are kinematically modelled as point-end effectors with zero size.
The CDSs and X-Y table set ups are identical to those of the single-robot set up described in Chapter 3.

4.4.2. Simulated System Configurations

A series of five different configurations are explored:

(i) All devices are fixed at user-specified locations, where only the placement robots are allowed to move.

(ii) All devices are fixed at GA-optimized locations, where only the placement robots are allowed to move.

(iii) The X-Y table is fixed at a GA-optimized location, where the CDSs and the placement robots are allowed to move.

(iv) The CDSs are fixed at GA-optimized locations, where the X-Y table and the placement robots are allowed to move.

(v) All devices are free to move.

All above five configurations were tested with user-defined as well as with dynamic CDS allocation modes. (For a more detailed description of the problem configurations see Chapter 3).
4.4.3. Simulation Results

Each of the above problems were run 25 times using ECPSIM2R. The best results are given for each case, both for user-defined CDS allocation and for dynamic CDS allocation.

(i) Devices fixed at non-optimal locations

This run yielded a minimum time of 0.928 s with a placement sequence of (5, 1, 4, 2, 3, 0). Figure 4.2 shows the paths the placement robots took. The fixed position of the X-Y table is shown by the dashed-line box.

![Figure 4.2: Robot Paths - Case 1 - User-Defined-CDS-Component Allocation.](image)

Figure 4.3 shows the time-line plots of each of the devices.
Convergence data was also generated, with the population mean, minimum and maximum being recorded every one hundred generations, as shown in Figure 4.4. It can be seen that the population converges almost immediately to a steady value for this case.

For the above example, when the component assignment is carried out dynamically by the GA, a total time of 0.906 s is obtained for the placement sequence of (5, 1, 3, 2, 4, 0). Figure 4.5 shows the robot paths. This time is slightly shorter than the
case of the user-defined CDS allocation of components (See Appendix B for more details).

Figure 4.4: Convergence Plot - Case 1 - User-Defined-CDS-Component Allocation.

Figure 4.5: Robot Paths - Case 1 - Dynamic-CDS-Component Allocation.
(ii) Devices fixed at optimal locations

With each of the devices fixed at GA determined optimal locations, this run generated a minimum total time of 0.924 s, with a sequence (2, 4, 1, 3, 0, 5). Figure 4.6 shows the robot paths. Figure 4.7 shows the time-line plots of the various devices.

Figure 4.6: Robot Paths - Case 2 - User-Defined-CDS-Component Allocation.

As shown in Figure 4.8, the GA once again converges rapidly. For the above example, when dynamic-CDS-component allocation is utilized, a total time of 0.837 s is obtained for a placement sequence of (3, 2, 4, 5, 0, 1). Figure 4.9 shows the robot paths. In this case, the GA determined that using a single robot is most efficient, and therefore set the split-point to zero. In contrast the user-defined-CDS-component allocation set up has a fixed split-point of three, dividing the components uniformly between the two CDSs, forcing the GA to use two robots (See Appendix B for more details).
Figure 4.7: Time Lines - Case 2 - User-Defined-CDS-Component Allocation.

Figure 4.8: Convergence Plot - Case 2 - User-Defined-CDS-Component Allocation.
(iii) X-Y table fixed at an optimal location

This example illustrates the performance improvement achieved via the motion of the CDS when the X-Y table is fixed at a GA-determined optimal location. In this run, the total time was 0.905 s with a placement sequence of (0, 4, 2, 3, 1, 5). Figure 4.10 shows the robot paths.

Comparing Figure 4.6, for the fixed CDSs, with Figure 4.10, where the CDSs are moving, one can note that the pick points in Figure 4.10 are closer together. Namely, the CDSs, position themselves to yield the shortest path between the current placement point and the next pick point and the shortest path between the next pick point and the next placement point.

Figure 4.9: Robot Path - Case 2 - Dynamic-CDS-Component Allocation.
Examining the time lines in Figure 4.11, it can be seen that both robots are continuously busy (except when Robot 1 waits for Robot 2 to complete the first placement). The CDSs are moving to the next pick position immediately after the previous pick event. The distance the CDSs must travel, when compared to the robots, are substantially shorter, therefore, the CDSs stop moving in advance of the robots as they arrive at their rendezvous locations. Since the distances travelled by the CDSs are small, only minor time savings are achieved over the optimally-fixed CDS system configuration.

Figure 4.12 once again shows that the GA converges rapidly.
For the above example, when the CDS component allocation is carried out dynamically a total time of 0.785 s is obtained for a placement sequence of (4, 2, 5, 1, 0, 3). Figure 4.13 shows the robot paths. It is interesting to note that the GA splits the components into two sets, with two components in CDS 1 and the other four components in CDS 2, resulting in substantial time savings over the user-defined case. (See Appendix B for more details).
Figure 4.12: The Convergence Plot - Case 3 - User-Defined-CDS-Component Allocation.

Figure 4.13: Robot Paths - Case 3 - Dynamic-CDS-Component Allocation.
(iv) CDSs fixed at optimal locations

In the case of CDSs being fixed at GA-optimized locations, the effect of the X-Y table motion on the optimization process is illustrated. The total time was determined as 0.889 s with a placement sequence (3, 1, 4, 2, 5, 0). Figure 4.14 shows the robot and X-Y table paths. One can note the substantial movement of the X-Y table.

![Figure 4.14: Robot Paths - Case 4 - User-Defined-CDS-Component Allocation.](image)

Figure 4.15 shows the time lines of the various devices. The CDSs are fixed, and hence only the pick times are shown. The time line of the X-Y table indicates that it is continuously busy. The X-Y table starts on the right side of the workspace and slowly moves to the left, thereby reducing some of the robots' placement times.
Figure 4.16 shows that the GA converges relatively rapidly.

For the above example, when the CDS component allocation is carried out dynamically by the GA, a total time of 0.837 s is obtained for a placement sequence of (3, 2, 4, 5, 0, 1). Figure 4.17 shows the robot paths. The path of Robot 1 may at first appear to be non-optimal, however, examining Equation (4.1) one notes that the maximum of the X-Y table and robot times is used to evaluate the genome. Namely, the robot path is only optimized so that it arrives at the rendezvous location before the X-Y table does when the
X-Y table is the slower device. This is the case in Figure 4.17. (See Appendix B for more details).

Figure 4.16: Convergence Plot - Case 4 - User-Defined-CDS-Component Allocation.

Figure 4.17: Robot Paths - Case 4 - Dynamic-CDS-Component Allocation.
(v) All devices moving

In the final case examined, all devices may move and interact to produce the optimum time. This simulation resulted in a total time of 0.873 s for a placement sequence of (2, 4, 1, 5, 0, 3). Figure 4.18 shows the robot paths.

![Robot Paths - Case 5 - User-Defined-CDS-Component Allocation](image)

Figure 4.18: Robot Paths - Case 5 - User-Defined-CDS-Component Allocation.

From Figure 4.19, one can note that the X-Y table is continuously busy. The CDSs have only a limited role in determining the cycle time, since they complete all of their motions before the robot or the X-Y table arrive at their rendezvous locations.

The total time improvement achieved via two-robot all-devices-moving approach over the single-robot all-devices-moving approach is 0.332 s (approximately 28%).
Figure 4.20 shows the convergence data for the all-devices-moving case. The GA converges rapidly to the optimum.

For the above example, when the CDS component allocation is carried out dynamically by the GA, a total time of 0.785 s is obtained for a placement sequence of (4, 2, 5, 1, 0, 3). (This result was accomplished by starting the simulation with optimal data from the X-Y-table-fixed-dynamic-CDS-allocation case). Figure 4.21 shows the robot paths. (See Appendix B for more details).
Figure 4.20: Convergence Plot - Case 5 - User-Defined-CDS-Component Allocation.

Figure 4.21: Robot Paths - Case 5 - Dynamic-CDS-Component Allocation.
4.4.4. Summary of Simulation Results

Table 4-1 presents the overall assembly time and the corresponding optimal component-placement sequence for each of the problem types for user-defined CDS allocation. As expected, when more devices are allowed to move the overall assembly time is reduced.

Table 4-1: Simulation Results Summary for User Specified CDS Allocation.

<table>
<thead>
<tr>
<th>System Configuration</th>
<th>Optimal Sequence</th>
<th>Total Time (s)</th>
<th>% Improvement w.r.t. the Non-Optimal Fixed Case ((Fixed-Other) / Fixed) x 100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Optimal Fixed</td>
<td>5 1 4 2 3 0</td>
<td>0.928</td>
<td>0.0</td>
</tr>
<tr>
<td>Optimally Fixed</td>
<td>2 4 1 3 0 5</td>
<td>0.924</td>
<td>0.4</td>
</tr>
<tr>
<td>CDS Moving</td>
<td>0 4 2 3 1 5</td>
<td>0.905</td>
<td>2.5</td>
</tr>
<tr>
<td>X-Y Table Moving</td>
<td>3 1 4 2 5 0</td>
<td>0.889</td>
<td>4.2</td>
</tr>
<tr>
<td>All Moving</td>
<td>2 4 1 5 0 3</td>
<td>0.873</td>
<td>5.9</td>
</tr>
</tbody>
</table>

Table 4-2 presents the overall assembly time and the corresponding optimal component-placement sequence for each of the problem types for dynamic CDS component allocation.

Comparing the single-robot to the two-robot case for all devices moving with dynamic CDS allocation an improvement of 34.9% is achieved, Table 4-3.
Table 4-2: The Simulation Results Summary for Dynamic CDS Allocation.

<table>
<thead>
<tr>
<th>System Configuration</th>
<th>Optimal Sequence</th>
<th>Total Time (s)</th>
<th>% Improvement w.r.t. the Non-Optimal Fixed Case, with User-Defined CDS Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Optimal Fixed (Dynamic CDS Allocation)</td>
<td>5 1 3 2 4 0</td>
<td>0.906</td>
<td>2.4</td>
</tr>
<tr>
<td>Optimally Fixed</td>
<td>3 2 4 5 0 1</td>
<td>0.837</td>
<td>8.4</td>
</tr>
<tr>
<td>CDS Moving</td>
<td>4 2 5 1 0 3</td>
<td>0.785</td>
<td>15.4</td>
</tr>
<tr>
<td>X-Y Table Moving</td>
<td>3 2 4 5 0 1</td>
<td>0.837</td>
<td>8.4</td>
</tr>
<tr>
<td>All Moving</td>
<td>5 3 4 0 1 2</td>
<td>0.785</td>
<td>15.4</td>
</tr>
</tbody>
</table>

Table 4-3: Comparing Single- and Two-Robot Performance for the All-Moving Case With Dynamic-CDS-Component Allocation.

<table>
<thead>
<tr>
<th>System Configuration</th>
<th>Total Time (s)</th>
<th>% Improvement w.r.t. the Single-Robot Non-Optimal-Fixed Case</th>
<th>% Improvement w.r.t. the Single-Robot All-Moving Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Robot Non-Optimal-Fixed Case, User-Defined CDS-Allocation</td>
<td>1.621</td>
<td>0.0</td>
<td>--</td>
</tr>
<tr>
<td>Single-Robot All-Moving Case, Dynamic CDS-Allocation</td>
<td>1.205</td>
<td>25.7</td>
<td>0.0</td>
</tr>
<tr>
<td>Two-Robot All-Moving Case, Dynamic CDS-Allocation</td>
<td>0.785</td>
<td>51.6</td>
<td>34.9</td>
</tr>
</tbody>
</table>
4.5. Summary

In this chapter, the GA objective function and the methodology for the two-robot TSP+ problem were defined. As expected, it was illustrated that introducing another robot results in overall performance improvement. Dynamic-CDS-component allocation plays a greater role here due to its unsymmetrical component assignment. In the next chapter, collision avoidance issues related to the use of two robots will be addressed.
5. THE TWO-ROBOT TSP+ COLLISION AVOIDANCE PROBLEM

5.1. Introduction

In Chapter 4, the two-robot TSP+ problem was presented and solved without addressing the collision avoidance problem. In this chapter, the need for a collision avoidance method is illustrated and a collision avoidance method is proposed for the problem at hand. All the systems simulated in Chapter 4 and an additional example are examined for collision avoidance. Finally, the results of a real electronic-component placement problem are presented.

5.2. Collision Avoidance Approach

As discussed in Chapter 1, the two basic collision avoidance approaches noted in the literature are (i) the path-planning-based approach, and (ii) the time-delay-based approach. The collision-avoidance approach proposed here adds a safety delay to one of the two robots, delaying its motion until the other robot is no longer on its path. This method may appear to be an approach based on time delays, however, since collision avoidance is integrated into the GA (as described below) the end result is a combination of path changes and time delays.

For a given GA genome, potential collisions are averted by adding a safety delay to one of the two robots. Thus, the time it takes to complete the assembly for that particular configuration becomes longer. (Namely, the fitness of the genome that is based on the assembly time becomes lower). This is where the correction for collision
avoidance as part of the GA-optimization evaluation becomes valuable. Since the variables that affect the optimization of the system also determine whether collisions occur, a separate collision-avoidance correction carried out after the path-planning optimization could interfere with the result. Herein, with the collision avoidance integrated into the optimization, as is possible with the GA, it is still possible to ensure that the solution is not only collision free but also optimized.

5.3. The Objective Function

The GA objective function evaluates a genome and assigns a fitness value to it. In this chapter, the objective function is identical to the one presented in Chapter 4 with an additional term related to collision avoidance.

5.3.1. Objective Function Input Variables

The objective function input variables remain the same as for Chapter 4.

5.3.2. Calculating Motion Times

Device motion times are calculated as described in Chapter 4 using the standard kinematic models of the robots.
5.3.3. The Governing Assembly-Time Equation

The overall assembly time for a complete population of a PCB, $t$, is calculated as in Chapters 3 and 4:

$$ t = \sum_{i=1}^{N} C_i, \quad (3.1) $$

where $C_i$ is the time it takes to complete cycle $i$ and $N$ is the number of components on the PCB. The cycle time $C_i$ is calculated as described in Chapter 4 where:

$$ C_i = \max[r_i t_i, c_i] + p_i c_i. \quad (4.1) $$

Here, the expression for the robot cycle time is the same as in Chapter 4 with one additional term. As described above, the collision avoidance method adds a safety delay, $t_i^{sd}$, to the total robot time:

$$ r_i t_i = \max[(r_i p_i - \alpha_i t_i), (d_i t_i - \alpha_i t_i)] + p_i c_i + p_i t_i + t_i^{sd}. \quad (5.1) $$

The safety delay term will be described below. All other terms in Equations (4.1) and (5.1) are calculated as in Chapter 4.
5.3.4. The Safety Delay

Before the use of the safety-delay term, one must determine if a collision is possible at all. It is assumed that the robots do not stop anywhere except at the pick and placement locations. Namely, immediately after a placement operation, the current robot starts to move back to its pick location. As shown in Figure 5.1, the robot can be in one of four possible locations: (1) at the pick location, (2) on the way to the placement location, (3) at the placement location, and (4) returning from the placement location.

![Figure 5.1: The Four Possible Locations of a Robot.](image)

There is no possibility of collision when the robot is in Position 1. However, for all other positions a collision is possible. Therefore, before the current robot moves from its pick location to its placement location, a collision check is carried out. This collision check searches for the existence of another robot in the workspace (in Positions 2 through 4).
To carry out a collision check, the robot state must be modelled in space and time. It was decided to model the robots as "walls" moving across the workspace, as shown in Figure 5.2, in order to simplify the collision detection process. This modelling simplification reduces the state of the robot (for collision avoidance calculations) to two variables, the robot X-coordinate and a time at which the robot occupies the corresponding position.

![Diagram showing robots as walls for collision avoidance](image)

Figure 5.2: The Robots Modelled as Walls for Collision Avoidance Detection.

In Figure 5.2, for the placement of a component, Robot 1 moves from CDS 1 to the placement location. Its maximum travel toward the other robot is the X-coordinate of the placement location of component \(i\), \(r_1w_i\). The current robot (Robot 1, in the example) can collide with the other robot (Robot 2) that may still be in the common workspace. Therefore, in determining whether a safety delay is required, one must check all previous cycles, up to the last cycle, in which the current robot moved. For example, all previous
cycles up until when Robot 1 last moved have to be checked. In these cycles, one must ensure that Robot 2 is not in Robot 1's path.

For a collision to occur, the position and time for the two potentially colliding robots have to match. Therefore, the collision check for the current robot (in cycle i) checks all previous cycles until the current robot last moved (cycle k). To check for an overlap, cycles (k+1) to (i-1) are checked to ensure that the current robot placement location in the X axis ($r_i w_i$) does not crossover with any of the other robot's previous placement locations ($r_{2w(k+1)}$ to $r_{2w(i-1)}$).

For the case in Figure 5.2, it is assumed that the last cycle in which Robot 1 placed a component occurred in cycle (i-2), i.e., $k=(i-2)$. Checking all cycles from (k+1) to (i-1), it is determined that the only required collision check with Robot 2 is in cycle (i-1). Since Robot 1 is on the left, one would check that $r_1 w_i$ is less than $r_2 w_i-1$.

For the above collision check, one of two cases may result:

Case 1: If no overlap occurs in any of the previous cycles, (k+1) to (i-1), then, no collision is possible (as in Figure 5.2), and the safety delay is not required, i.e., $s_i t_i = 0$.

Case 2: An overlap occurs in one or more of the previous cycles, (k+1) to (i-1), and thus a collision is possible. Further testing to check the time variables is necessary to confirm or disprove a collision.
If an overlap occurs (Case 2), one must determine its extent. Once this is known, one can check when the robots are in the overlap region at the same time. Figure 5.3 shows an example of an overlap event. The overlap area is highlighted in grey. In this example, the safety delay for Robot 1 is calculated. Robot 2 moves and places Component 1, and then moves to place Component 2. Robot 1 needs to place Component 3. Examining Figure 5.3, one can see that the overlap area consists of two sub-areas. To avoid a collision with Robot 2, Robot 1 not only has to stay to the left of Component 1’s placement location, but also to the left of Component 2’s placement location. To ensure that all cases are considered when checking for collisions, it is necessary to take the largest of the overlaps. Therefore, all overlaps are calculated and their maximum is chosen. The maximum work distance overlap \( \max_{\text{of} \; d_i} \) is thus calculated as:

\[
\max_{\text{of} \; d_i} = \max (r_c w_i - r_p w_j), \quad j = a \text{ to } n. \tag{5.2}
\]

Above, \( r_c w_i \) is the work distance of the current robot, and the variables \( r_p w_a \) to \( r_p w_n \) are the distances in cycles \( (k+1) \) to \( (i-1) \) where an overlap of workspaces occurs (cycles where no overlap occurs are not checked by Equation (5.2)).

For the example in Figure 5.3, using Equation (5.2), the width of the overlap area from placement locations 1 to 3 is calculated, and then, the overlap from 2 to 3 is calculated. Taking the maximum, one finds that the maximum overlap, is the one from 2 to 3.
Once the maximum overlap has been determined, the time values can be checked. In this thesis, a collision is defined as a situation in which the current robot enters the maximum overlap region before the other robot has had a chance to leave the region. In the case where both position and time variables indicate that the robots are in the overlap region at the same time, a preventative strategy has to be developed. The approach here is to introduce a delay to the current (cycle's) robot's motion to allow the other robot to leave the overlap region before the current robot enters it.

To check each previous cycle, from \((k+1)\) to \((i-1)\), for a time overlap, one needs to calculate the robot entry time, \(r_t^1\), which is when the current robot enters the maximum work distance overlap region. One also needs to calculate the other robot's leaving times, \(r_t^j\), for all previous cycles for \(j=(k+1)\) to \((i-1)\), (where \(k\) is the cycle in which the current
robot last moved, and i is the current cycle). The latter, robot leave times, are the times when the other robot leaves the maximum overlap region for each of the previous cycles from \((k+1)\) to \((i-1)\). Collisions occur when \(e_t \) is less than \( t_p \), i.e., when both robots are in the maximum overlap region.

The simplest way to calculate \(e_t\) is by starting at a known time, namely, considering the time the current robot reaches its placement location, and subtracting the time it took to cross the maximum overlap. One must first calculate the time the current robot takes to cross the maximum overlap as:

\[
e_t = \frac{\text{max } d_i}{\dot{x}_i} .
\]  

In Equation (5.3), \(\text{max } d_i\) is the maximum overlap, as defined above in Equation (5.2) and the variable \(\dot{x}_i\) is the speed of the current robot in the X-axis direction.

Therefore,

\[
e_t = \sum_{j=1}^{i-1} C_j + \text{max}[e_t -(d_t - \text{off} )] + e_t + p_t - \dot{x}_i .
\]  

In Equation (5.4), \(C_j\) is the total cycle time for cycle \(j\) (as defined in Equation (4.2)). The other variables are defined as in Chapter 3: \(p_t\) (robot pick time), \(o_t\) (robot off time), \(d_t\) (CDS move time), \(o_d\) (CDS off time), \(p_c\) (pick time), and \(p_t\) (robot placement time).
To calculate the robot leaving time, \( t_{j} \), for a previous cycle (j), from (k+1) to (i-1), one can start with the known time of the end of the placement operation and add to it the time the previous robot needs to cross the maximum overlap distance while moving to the next pick location. First, however, one needs to solve for the time the previous robot takes to cross this distance:

\[
\frac{\alpha}{\mu} t_{j} = \max_{d_{i}} \frac{d_{i}}{\mu \dot{x}_{j}}. \tag{5.5}
\]

In Equation (5.3), \( \max_{d_{i}} \) is the maximum overlap and the variable \( \mu \dot{x}_{j} \) is the speed of the previous robot in the X-axis direction.

Therefore,

\[
\frac{1}{\mu} t_{j} = \sum_{i=0}^{k} C_{i} + \frac{\alpha}{\mu} t_{j}. \tag{5.6}
\]

In Equation (5.6), \( C_{i} \) is the total cycle time for cycle 1 (as defined in Equation (4.2)), where \( l \) is any cycle from zero to \( k \).

After calculating \( \frac{1}{\mu} t_{i} \) and \( \frac{1}{\mu} t_{j} \), where \( j = (k+1) \) to (i-1), one can check for collisions. A collision occurs when \( \frac{1}{\mu} t_{j} > \frac{1}{\mu} t_{i} \).
After all of the cycles, \( j = (k+1) \) to \((i-1)\), have been tested for collision, it is possible to calculate the time delay required to avoid a collision for a particular cycle \( j \):

\[
^s_d t_j = ^r t_j - ^r t_1. \tag{5.7}
\]

By using the maximum of the safety delays, the required delay to ensure that no collisions will occur in any of the previous robot cycles is obtained:

\[
^s_d t_i = \max( ^s_d t_j, \text{ for } j = (k+1) \text{ to } (i-1) ). \tag{5.8}
\]

Figure 5.4 shows the time line for the example shown in Figure 5.3. The dashed lines indicate the times when the respective robots enter the overlap area. Robot 2 places Components 1 and 2 and Robot 1 places Component 3. The grey blocks indicate the placement events. Using the method described above, starting from \(^s t_j\), the time the current robot (1) enters the overlap region, and \(^r t_2\), the time the previous robot (2) leaves the overlap region, the maximum delay can be calculated. The safety-delay-corrected entry time for the current robot is shown in Figure 5.4 as \(^s t'_3\).

5.4. Collision Avoidance Solution Methodology

As in Chapter 4, the simulation tool ECPSIM2R (Electronic Component Placement SIMulator for 2 Robots) was used with the current objective function.
<table>
<thead>
<tr>
<th>Cycle 1</th>
<th>Cycle 2</th>
<th>Cycle 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Y Table</td>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>Robot 2</td>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>Robot 1</td>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Figure 5.4: Time Line of Two-Robot Interaction.

5.5. A Collision Avoidance Example

ECPSIM2R was used to test the same system configurations of Chapter 4.

5.5.1. The Set Up

For the cases tested in Chapter 4, it was noted that the optimal robot paths did not have any collisions, (Appendix C). The optimization method separated the components into two groups based on their proximity to their respective CDS bins. This separation prevented any robot crossover events and as a result no collisions occurred. Therefore, in
order to illustrate the collision avoidance routine, an artificially user-defined CDS assignment case was created, where crossover events occur and collisions are possible.

The two-robot system and the PCB arrangement described in Chapter 4 are used here as well. The 2-dof gantry type placement robots can move at a maximum speed of 2 m/s. Robot 1’s workspace extends from -10 to 300 mm in the X direction and from 0 to 300 mm in the Y direction, including CDS 1’s and the X-Y table’s workspaces. Robot 2’s workspace extends from 0 to 310 mm in the X direction and from 0 to 300 mm in the Y direction, including CDS 2’s and the X-Y table’s workspaces. The robots’ start and end locations are determined dynamically. They start and finish at their first component pick locations.

The components are numbered from zero to five and are distributed between the two CDSs. Instead of optimally having Components 0 to 2 in CDS 1 and Components 3 to 5 in CDS 2 (as in Chapter 4), the component assignments were swapped to force a crossover event, where Components 3 to 5 are now in CDS 1 and Components 0 to 2 are in CDS 2.

5.5.2. Problem Configurations

The above setup was run for the following two configurations:

(i) All devices moving with user-defined-CDS allocation. Collision avoidance routine is not employed.
(ii) All devices moving, with user-defined-CDS allocation, Collision avoidance routine is employed.

5.5.3. Individual Simulation Results

(i) Collision Avoidance Routine is not Employed

This run yielded a minimum total time of 1.061 s with a placement sequence of (4, 2, 3, 0, 5, 1). Figure 5.5 shows the paths that the placement robots took. The position of the X-Y table is shown by the dashed-line box. (See Appendix D for convergence plot and time lines).

Figure 5.6 shows the plot of the two robots' positions (X-coordinate) versus time allowing for quick collision identification. Any location where the line representing Robot 1’s movement crosses Robot 2’s line indicates a collision event. There are two collision events. The first occurs when Robot 2 collides with Robot 1 while trying to place Component 2. The second collision occurs when Robot 2 tries to place Component 1 and collides with Robot 1 placing Component 5.

(ii) Collision Avoidance Routine is Employed

This run yielded a minimum total time of 1.151 s with a placement sequence of (5, 1, 3, 2, 4, 0). Figure 5.7 shows the paths that the placement robots took. The position of the X-Y table is shown by the dashed-line box. (See Appendix D for convergence plot and time lines).
Figure 5.5: The Robot Paths; No Collision Avoidance Routine.

Figure 5.6: Robot X-Coordinate vs. Time; No Collision Avoidance Routine.
Figure 5.8 shows the robots' positions versus time, allowing for easy potential-collision observation. In this case, as expected, the robot time lines do not cross. Namely, no collisions occur.

Figure 5.7: The Robot Paths; With Collision Avoidance Routine.

Comparing Figure 5.5 and 5.7, one sees that the GA, when run with the collision avoidance routine, not only added necessary safety delays but chose a different placement sequence and rendezvous locations.
5.6. An Industrial Application Example

The methodology and the simulation tools ECPSIM and ECPSIM2R were used to plan an assembly process for a commercial PCB. The placement of all the surface mounted components on a commercial network card was modelled, Figure 5.9. The components were numbered from 0 to 36.

The simulation results for the single-robot, all devices moving with dynamic CDS allocation, resulted in an assembly time of 11.530 s with a sequence of (2, 35, 28, 14, 9, 12, 11, 21, 17, 32, 31, 10, 18, 36, 29, 1, 6, 8, 5, 3, 0, 25, 33, 34, 30, 7, 15, 24, 26, 22, 20, 4, 19, 27, 16, 13, 23). The results for the two-robot, all devices moving with dynamic CDS allocation, resulted in a time of 6.123 s with a sequence of (33, 6, 32, 22, 29, 0, 19, 24, 3, 13, 20, 2, 11, 7, 15, 16, 9, 27, 14, 18, 21, 36, 10, 8, 4, 5, 35, 1, 17, 12, 31, 26, 28,
23, 30, 25, 34). Figure 5.10 shows the convergence data for the two-robot simulation where the initial non-optimized case has an assembly time of 38.71 s. (See Appendix E for more details).

Figure 5.9: The SMC EtherEZ Network Card PCB.

Figure 5.10: Convergence Plot; Two-Robot Set Up Assembling the SMC PCB.
5.7. Summary

In this chapter, the collision avoidance methodology was defined and illustrated through examples. Since the collision avoidance routine was embedded in the GA's objective function, the optimization chose a collision-free solution with the minimum additional safety delay.
6. SUMMARY AND CONCLUSIONS

6.1. Introduction

Autonomous robotic systems are being increasingly utilized in industrial environments, thus, necessitating the development of modelling methodologies and tools for the purpose of optimizing their operational efficiency. To address this problem, this thesis presented a generalized point-to-point (PTP) motion-planning technique for multiple coordinated assembly robots. The proposed approach minimizes assembly times for single- and multiple-robot assembly systems using Genetic Algorithms (GAs). Specifically, as an example area, the optimization of the electronic component placement process was addressed.

6.2. Single-Robot Augmented Travelling Salesperson Problem (TSP+)

The classical TSP models the movements of a salesperson travelling through a number of cities. The optimization problem is to choose the order in which to visit the cities to minimize the total distance travelled. For the TSP+ problem with multiple robots, the “rendezvous-point” planning problem must also be solved. The “salesperson” (one robot) as well as the “cities” (another robot) move. Namely, further planning is required to choose where the “salesperson” and the “city” should rendezvous.

The TSP+ problem was investigated for a single-placement robot in Chapter 3, where a system consisting of a placement robot, a moving XY-table, and two moving component delivery systems (CDSs) was modelled. Use of a genetic algorithm (GA) was
chosen as the solution approach for the problem. This method can optimize all parameters simultaneously. Since many of the optimization parameters are interdependent and the modification of one affects others, this simultaneous optimization is advantageous.

The methodology was successfully illustrated by testing it with various system configurations, investigating higher dof systems and Dynamic CDS-component allocation. From the resulting simulation data one noted that the proposed novel optimization methodology combined with the introduction of a greater number of dof, results in a substantial improvement over the non-optimized case. Table 6.1 shows a summary of the key results for the single-robot problem example.

Table 6.1: The Simulation Results Summary for the Single-Robot Problem.

<table>
<thead>
<tr>
<th>Problem Type</th>
<th>PCB on X-Y Table</th>
<th>Delivery Systems</th>
<th>% Improvement w.r.t. the Non-Optimal Fixed Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Optimal Fixed User-Defined-CDS-Allocation</td>
<td>Fixed (Middle)</td>
<td>Fixed (Middle)</td>
<td>0.0</td>
</tr>
<tr>
<td>All Moving</td>
<td>Moving</td>
<td>Moving</td>
<td>25.7</td>
</tr>
</tbody>
</table>

6.3. Two-Robot TSP+

In Chapter 4, the single-robot TSP+ was expanded to include an additional placement robot. This further complicated the problem by requiring a division of placement tasks. For the two-robot case, the solution strategy for solving the problem was separated into two parts: the optimization method and the collision avoidance method.
The expanded multi-robot TSP+ was also solved through the use of a GA. A two-placement-robot TSP+ electronic component placement system was modelled. The system consisted of two placement robots, a moving XY-table, and two moving CDSs.

Table 6.2 shows how the optimal use of the multi-robot system improves on the single-robot case. Comparing the single-robot to the two-robot case for all devices moving with dynamic CDS allocation, an additional improvement of 25.9 % was achieved.


<table>
<thead>
<tr>
<th>Problem Type</th>
<th>Total Time (s)</th>
<th>% Improvement w.r.t. the Single-Robot-All-Moving Case</th>
<th>% Improvement w.r.t. the Single-Robot-Non-Optimal-Fixed Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Robot-Non-Optimal-Fixed-Case, User-Defined-CDS-Allocation</td>
<td>1.621</td>
<td>--</td>
<td>0.0</td>
</tr>
<tr>
<td>Single-Robot-All-Moving-Case, Dynamic-CDS-Allocation</td>
<td>1.205</td>
<td>0.0</td>
<td>25.7</td>
</tr>
<tr>
<td>Two-Robot-All-Moving-Case Dynamic-CDS-Allocation</td>
<td>0.785</td>
<td>34.9</td>
<td>51.6</td>
</tr>
</tbody>
</table>

6.4. Two-Robot TSP+ and Collision Avoidance

Chapter 4 presented the two-robot TSP+ problem without addressing the collision avoidance problem. One can, however, easily note the potential of collision between two two-placement robots attempting to place components on the same PCB, even if the PCB...
is on a moving X-Y table. Chapter 5 addressed this multiple robot collision avoidance problem.

The collision avoidance strategy proposed in this thesis is a rule-based approach and was incorporated into the objective function of the GA. Since the parameters of sequencing and device positions during pick-and-place operations affect both the collision avoidance problem and the performance of a given placement strategy, it is advantageous to check both simultaneously, when searching for an optimum solution. Hence, their integration into a single objective function.

An example of a real PCB with 37 components was also presented showing that the simulation tool and the methodology is easily implementable for real industrial problems.

6.5. Future Work

This section is divided into two sections: (i) specific improvement suggestions to the existing methodology and simulation tools, and (ii) general suggestions for further research.

6.5.1. Specific Improvements

There are a number of aspects of the current method which could be improved upon. The current collision avoidance routine, although effective, is relatively simple.
Further improvements to this routine could result in additional time savings. For example, currently, the maximum overlap distance is used as the exclusion area for the delayed robot. However, if the maximum overlap distance is the result of the first component placed by the first robot, and then, the next component placed by the same robot results in a smaller overlap area, the second robot could follow the first robot in its retreat up to the next overlap area, thereby getting closer to its destination and saving some delay time.

Also, the current simulation tool is set up to have one bin per component. With a few changes to this tool, the components that are of the same type could share the same CDS bin. This issue was originally planned for solution but not implemented due to time constraints.

In this thesis, the path-planning method was not part of the focus and as a result a simple kinematic model was chosen. The simulation tool could be made more accurate by implementing a path-planning method based on robot dynamics. Many such methods are described in the literature.

The actual software, ECPSIM and ECPSIM2R, could be improved with a better graphical user interface (rather than text input files). GALib [23] was used without a lot of attention to optimizing the actual GA configuration, and a decrease in simulation time could probably be achieved with some modifications.
6.5.2. General Suggestions

The proposed methodology has only been implemented as an approach for modelling planar problems, but could be expanded into three dimensions. This would make it applicable to a larger number of practical assembly problems.

It would also be of interest to compare the simulation results to real experimental implementations of an electronic component placement system or other point-to-point assembly set ups.

6.6. Final Conclusions

This thesis presented an effective and novel approach to solving multi-robot point-to-point planning problems. The proposed methodology was used to simulate a variety of problems. Its effectiveness was successfully shown with substantial assembly time reductions for all cases simulated. It was also shown that the proposed methodology can be easily applied to industrial problems.
APPENDIX A:

A. Dynamic CDS Allocation Additional Plots

This appendix includes the time-line plots and convergence plots for the dynamic CDS allocation for the single robot problem presented in Chapter 3.

A.1. Devices Fixed at Non-Optimal Locations - Dynamic CDS Allocation

Figure A.1 shows that the CDSs and X-Y tables are stationary, with only the pick up and placement times plotted. The robot is continuously busy. The dynamic CDS allocation saves some time over the user-defined case.

Figure A.1: Time Line - Case 1 - Dynamic-CDS-Component Allocation.
Figure A.2: Convergence Plot - Case 1 - Dynamic-CDS-Component Allocation.

Figure A.2 shows the convergence data for CDS allocation. The population average, maximum and minimum were sampled every ten iterations. The GA converges rapidly.

A.2. Devices Fixed at Optimal Locations - Dynamic CDS Allocation

Figure A.3 shows the time lines. CDS 1 is empty and has no components in it, so it has nothing plotted. CDS 2 is fixed and therefore only has the pick-up times of each of the components plotted. Similarly the X-Y table is also fixed, and only has the placement times plotted. The robot is the only moving device and is continuously busy.
Figure A.3: Time Line - Case 2 - Dynamic-CDS-Component Allocation.

Figure A.4 shows the convergence plot of the GA for this case. The GA converges fairly rapidly in the first 100 iterations, almost reaching the final value. The GA reaches the final value before 10 000 iterations.
A.3. X-Y Table Fixed at An Optimal Location

The time line in Figure A.5 shows that CDS 1 is idle and not used. The X-Y table is fixed and only the placement times are plotted. CDS 2 is quite busy. CDS 2 is still moving with time to spare, reaching its rendezvous locations before the robot. The robot is continuously busy, being the limiting device in this optimization.

Figure A.6 shows the convergence of the GA for this case. The GA converges rapidly.
Figure A.5: Time Line - Case 3 - Dynamic-CDS-Component Allocation.

Figure A.6: Convergence Plot - Case 3 - Dynamic-CDS-Component Allocation.
A.4. CDSs Fixed at Optimal Locations

Figure A.7 shows the CDSs as fixed, with only the pick times plotted. The robot is continuously busy. Its activity is closely matched by the X-Y table, which is also very active. After Component 2 is placed, the X-Y table starts crossing the workspace, stopping for placement only when the robot catches up with it. It then resumes its movement across the workspace, until the robot catches up with it again.

Figure A.7: Time Lines - Case 4 - Dynamic-CDS-Component Allocation.

The convergence plot shown in Figure A.8 indicates that the GA converges.
Figure A.8: Convergence Plot - Case 4 - Dynamic-CDS-Component Allocation.

A.5. All Devices Moving

Figure A.9 shows the path of the robot, which starts in the top right corner with component 5. The X-Y table placement positions are shown by the dashed boxes. First the components from CDS 2 are placed. Then, the robot and X-Y table cross the workspace, and then the components from CDS 1 are placed.
Figure A.9 shows the time lines of the devices. All plots show the devices in motion, with the CDSs' plots showing the movement to pick locations and the pick up times. Looking at Figure 3.22 and Figure A.10 one sees that neither of the CDSs are fully occupied in both the user-defined and dynamic-CDS-component allocation cases. In the user-defined CDS allocation, the components are separated into Components 0, 1, and 2 which are located on the left side of the PCB, and 3, 4, and 5 which are located on the right side of the PCB. The dynamic CDS allocation for case where the X-Y table moves and the CDSs are fixed also splits the data into these two groups. This suggests that the user-defined data is already optimized in this respect. The next level of component sequencing at the CDS level can take place either through dynamic CDS allocation or through the motion of CDSs. Looking at the results in Chapter 3 for Case (iii) X-Y table fixed at an optimal location, with a user-defined CDS allocation, a sequence of 5, 3, 4 is achieved by the CDS motion.
As seen in the time line above in Figure A.10, the CDSs are idle for a substantial amount of time, and therefore are not the limiting devices in the optimization. This means that the local CDS sequence rearrangement by CDS motion rather than by CDS dynamic allocation, does not increase the overall cycle time. This shows that once components have been sorted according to left or right side, as the case with both the user-defined data and the dynamic results, the CDS motion and CDS dynamic allocation become redundant. This explains why the both the user-defined and dynamic CDS allocation cases can have the same optimal time, and why with the dynamic-CDS-
allocation-all-moving case was run with user-defined CDS allocation data, there was no additional time improvement.

Figure A.11 shows the convergence curve for the all-moving case with dynamic-CDS allocation. The minimum starts at a low value as explained earlier, because the simulation was started with data from the user-defined-all-moving case. It takes almost the full 100 000 iterations for the values to converge.

Figure A.11: Convergence Plot - Case 5 - Dynamic-CDS-Component Allocation.
APPENDIX B:

B. Dynamic CDS Allocation Additional Plots

This appendix includes the time-line plots and convergence plots for the dynamic CDS allocation case for the two-robot problem presented in Chapter 4.

B.1. Devices Fixed at Non-Optimal Locations

Figure B.1 shows that the GA converges rapidly. Figure B.6 shows that the CDSs and X-Y tables are stationary, with only the pick and placement times plotted. The dynamic CDS allocation saves some time over the user-defined case.

B.2. Devices Fixed at Optimal Locations

Figure B.2 shows that the GA converges rapidly. Figure B.7 shows the time lines. The CDSs are fixed and therefore only the pick times of the components are plotted. Similarly, the X-Y table is also fixed, and thus only the placement times are plotted. The robots are the only moving devices.

B.3. X-Y Table Fixed at An Optimal Location

Figure B.3 shows that the GA converges rapidly. Figure B.8 shows the time lines. The X-Y table is fixed and only the placement times are plotted. CDS 2 is quite busy. However it is still moving with some time to spare, reaching its rendezvous locations.
before Robot 2. The robots are almost continuously busy, being the limiting device in this optimization.

**B.4. CDSs Fixed at Optimal Locations**

The convergence plot shown in Figure B.4 indicates that the GA converges, though not as rapidly as in other cases. Figure B.9 shows time lines were the CDSs are fixed, with only pick times plotted.

**B.5. All Devices Moving**

Figure B.5 shows the convergence curve for the all-moving case with dynamic-CDS allocation. Figure B.10 shows the time lines of the devices.

![Figure B.1: Convergence Plot - Case 1 - Dynamic-CDS-Component Allocation.](image)
Figure B.2: Convergence Plot - Case 2 - Dynamic-CDS-Component Allocation.

Figure B.3: Convergence Plot - Case 3 - Dynamic CDS Component Allocation.
Figure B.4: Convergence Plot - Case 4 - Dynamic-CDS-Component Allocation.

Figure B.5: Convergence Plot - Case 5 - Dynamic-CDS-Component Allocation.
Figure B.6: Time Lines - Case 1 - Dynamic-CDS-Component Allocation.
Figure B.7: Time Lines - Case 2 - Dynamic-CDS-Component Allocation.
Figure B.8: Time Lines - Case 3 - Dynamic-CDS-Component Allocation.
Figure B.9: Time Lines - Case 4 - Dynamic-CDS-Component Allocation.
Figure B.10: Time Lines - Case 5 - Dynamic-CDS-Component Allocation.
APPENDIX C:

C. Robot X-Coordinate versus Time Plots

This appendix includes the robot X-coordinate versus time plots for the two-robot problems presented in Chapter 4. All problems ran without collision avoidance routines.

C.1. User-Defined-CDS Allocation

![Graph showing robot X-coordinate versus time for Case 1 - User-Defined.]

Figure C.1: Robot X-Coordinate versus Total Time - Case 1 - User-Defined.
Figure C.2: Robot X-Coordinate versus Total Time - Case 2 - User-Defined.

Figure C.3: Robot X-Coordinate versus Total Time - Case 3 - User-Defined.
Figure C.4: Robot X-Coordinate versus Total Time - Case 4 - User-Defined.

Figure C.5: Robot X-Coordinate versus Total Time - Case 5 - User-Defined.
C.2. Dynamic CDS-Allocation

Figure C.6: Robot X-Coordinate versus Total Time - Case 1 - Dynamic.

Figure C.7: Robot X-Coordinate versus Total Time - Case 2 - Dynamic.
Figure C.8: Robot X-Coordinate versus Total Time - Case 3 - Dynamic.

Figure C.9: Robot X-Coordinate versus Total Time - Case 4 - Dynamic.
Figure C.10: Robot X-Coordinate versus Total Time - Case 5 - Dynamic.
APPENDIX D:

D. Collision Avoidance Additional Plots

This appendix includes the time-line plots and convergence plots for the no collision avoidance, and collision avoidance cases presented in Chapter 5.

D.1. No Collision Avoidance

Figure D.1 shows time lines for the no collision avoidance case. Figure D.3 shows the convergence data for the no collision case.

D.2. Collision Avoidance

Figure D.2 shows the time lines for the collision avoidance case. Figure D.4 shows the convergence plot of the GA for this case.
Figure D.1: The Time Lines; No Collision Avoidance.
Figure D.2: The Time Lines; Collision Avoidance.
Figure D.3: The Convergence Plot; No Collision Avoidance.

Figure D.4: The Convergence Plot; Collision Avoidance.
APPENDIX E:

E. SMC EtherEZ PCB Example Additional Plots

This appendix includes additional plots for the SMC EtherEZ network card problem presented in Chapter 5.

E.1. Single Robot Solution

Figure E.1 shows the convergence plot for the single robot solution to the SMC EtherEZ assembly planning problem.

E.2. Two-Robot Solution

Figure E.2 shows that the two robot solution is collision free.

Figure E.1: The Convergence Plot, Single Robot Solution to the SMC PCB Problem.
Figure E.2: The Robot X-Coordinate versus Time Plot for the Two Robot Solution of the SMC PCB Problem.
REFERENCES


