An Agent-Based Approach to the Design of Rapidly Deployable Fault Tolerant Manipulators

Christiaan J. J. Paredis

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
in Electrical and Computer Engineering

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

August 1996.

This Research was supported in part by the Department of Energy under grant DE-F902-89ER14042, by Sandia National Laboratories under contract AL–3020, by the Department of Electrical and Computer Engineering, and by The Robotics Institute at Carnegie Mellon University. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the funding agencies.

Abstract

There exists a need for manipulators that are more flexible and reliable than the current fixed configuration manipulators. Indeed, robot manipulators can be easily reprogrammed to perform different tasks, yet the range of tasks that can be performed by a manipulator is limited by its mechanical structure. In remote and hazardous environments, such as a nuclear facility or a space station, the range of tasks that may need to be performed often exceeds the capabilities of a single manipulator. Moreover, it is essential that critical tasks be executed reliably in these environments.

To address this need for a more flexible and reliable manipulator, we propose the concept of a *rapidly deployable fault tolerant manipulator system*. Such a system combines a Reconfigurable Modular Manipulator System (RMMS) with support software for rapid programming, trajectory planning, and control. This allows the user to rapidly configure a fault tolerant manipulator custom-tailored for a given task. This thesis investigates all aspects involved in such a system. It describes an RMMS prototype which consists of seven manipulator modules with a total of four degrees-of-freedom. The reconfigurability of the hardware is made transparent to the user by the supporting control software that automatically adapts itself to the current manipulator configuration. To achieve high reliability, a global fault tolerant trajectory planning algorithm is introduced. This algorithm guarantees that a manipulator can continue its task even when one of the manipulator joints fails and is immobilized. Finally, all these aspects are considered simultaneously in the *task based design* software, that determines the manipulator configuration, its base position, and the fault tolerant joint space trajectory that are optimally suited to perform a given task.

The most important contribution of this thesis is a *novel agent-based approach* to solve the task based design problem. The approach is based on a genetic algorithm for which the modification and evaluation operations are implemented as autonomous asynchronous agents. Specific design knowledge about the task based design problem has been included in the agents, resulting in a significant reduction of the size of the design space and of the cost of evaluating a candidate design. Furthermore, thanks to their autonomous and asynchronous nature, these agents can be easily executed distributedly on a network of workstations. The flexibility and performance of the agent-based implementation, combined with the problem specific knowledge included in the modification and evaluation agents results in a powerful new approach to task based design of rapidly deployable fault tolerant manipulators.

Finally, the thesis presents a performance analysis of the agent-based design framework by comparing its results with those of exhaustive search, random search, and multiple restart statistical hill-climbing. This analysis is performed for three examples, including a comprehensive example of a satellite docking operation with a fault tolerant modular manipulator mounted in the cargo bay of the space shuttle.

Acknowledgments

I am very grateful to my advisor, Professor Pradeep Khosla. His vision and guidance played an important role in the realization of this work. Even though he was on a leave of absence during most of my doctoral studies, he managed to make time for regular meetings and provided constant encouragement and support. I also wish to thank the members of my thesis committee, Professors Takeo Kanade, Matt Mason, and Harry Asada, for their valuable and constructive comments.

I have been very fortunate to be part of the Advanced Manipulators Laboratory. The many people that have passed through the lab while I was a graduate student (too many to list here) have always been a source of friendship and support. During our weekly meetings, I often received much appreciated feedback on my research. In particular, I would like to thank C.J. Taylor, David LaRose, Dan Morrow, John Dolan, and Frank Dellaert for taking the time to read over the drafts of this thesis, and Debbie Scappatura for always offering a helping hand and a listening ear.

Many thanks also to Ben Brown, Randy Casciola, and Jim Moody for their contribution to the realization of the Reconfigurable Modular Manipulator System. Thanks to their efforts, this thesis consists not just of a theoretical exposition, but also includes experimental results in support of the theory. Equally important have been the contributions of the maintainers of the VASC computing facilities (Kate Fissell, Jim Moody, and Bill Ross) and the developers of the A-teams toolkit (Philip Chang, John Dolan, Jim Hemmerle, Sarosh Talukdar, and Mike Terk).

Finally, I would like to express my gratitude to all my family members. My parents always strongly supported my efforts to reach this goal, even though it meant that we would be far apart geographically.

Most of all I would like to thank my wife, Jan Jernigan, for her loving support and encouragement throughout my years at Carnegie Mellon University. I dedicate this thesis to her.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Robot manipulators can be easily reprogrammed to perform different tasks, yet the range of tasks that can be performed by a manipulator is limited by its mechanical structure. For example, horizontal SCARA-configuration manipulators, such as the Adept One and the CMU Direct Drive Arm II, are well suited for delicate table-top assembly operations requiring accuracy and selective stiffness, but have very limited vertical reach. On the other hand, vertical elbow-configuration manipulators, such as the Puma family, have a relatively long reach in all directions and are suitable for painting, welding, and parts handling, but do not have the accuracy and stiffness required for precise assembly. Therefore, to perform a given task, one needs to choose a manipulator with an appropriate mechanical structure—kinematical as well as dynamical.

Applying a manipulator with an optimal configuration to a particular task is only possible if the task is known in advance. This is often not the case in unpredictable environments such as a nuclear environment or a space station. To be able to address these unknown tasks, one would need a manipulator with a wide range of capabilities—probably beyond the limitations of a single manipulator. To solve this problem, one could deploy a large number of manipulators, each with different kinematic and/or dynamic characteristics. However, this is often prohibitively expensive, for instance, because the robots become contaminated by radiation, or because the total mass of all manipulator combined is too large to be transported into space.

In addition to having a wide range of kinematic and dynamic capabilities, it is important that manipulators in remote and hazardous environments be extremely *reliable*. Recently, with the Hubble telescope and the Mars Observer, NASA has experienced first hand how devastating the consequences can be when a critical component fails during a multi-billion-dollar mission. Space applications are particularly vulnerable to failure, because of the adverse environment (cosmic rays, solar particles etc.) and the demand for long term operation. In this context, NASA has started to incorporate fault tolerance in their robot designs (Wu et al. 1995). Reliability is also important in robotic applications for Environmental Restoration and Waste Management (ER&WM) program of the Department of Energy. Consider, for instance, the use of a manipulator in a nuclear environment where equipment has to be repaired or space has to be searched for radioactive contamination. The manipulator system deployed in these kinds of critical tasks must be reliable, so that the successful completion of the task or the safe removal of the robot system is assured. A third domain in which reliability is a major issue is medical robotics, because of the risk of the loss of human life. Although medical staff will probably always be on standby to take over in the case of a manipulator failure, the robot should at least be fail-safe, meaning it should fail into a safe configuration.

The goal of this thesis is to develop a system that overcomes the above mentioned shortcomings. We propose the concept of a *rapidly deployable fault tolerant manipulator system*, as

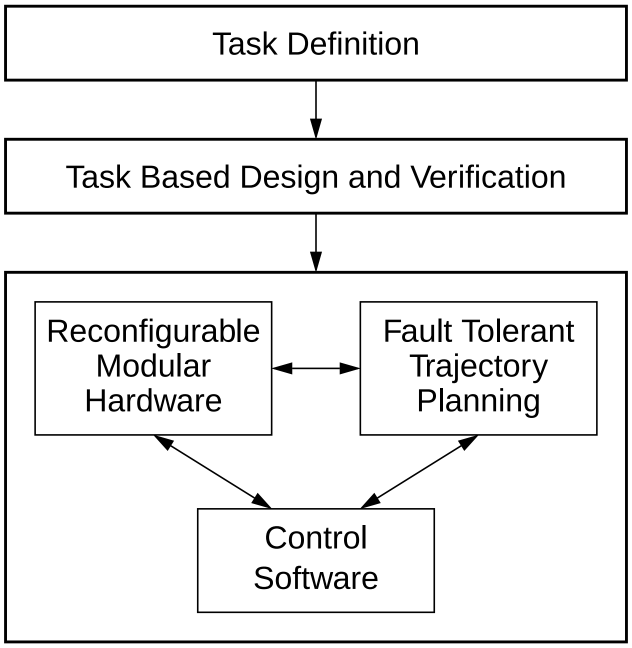

Task Definition
Task Based Design and Verification
Reconfigurable Modular Hardware
Fault Tolerant Trajectory Planning
Control Software

Figure 1-1. The concept of a rapidly deployable fault tolerant manipulator system.

illustrated in Figure 1-1. A rapidly deployable fault tolerant manipulator system consists of hardware and software that allow the user to rapidly build and program a manipulator which is custom-tailored for a given task and which can execute this task fault tolerantly.

The central building block of a rapidly deployable fault tolerant system is a *Reconfigurable Modular Manipulator System* (RMMS). The RMMS utilizes a stock of interchangeable link and joint modules of various sizes and performance specifications. By combining these general purpose modules, a wide range of special purpose manipulators can be assembled. Recently, there has been considerable interest in the idea of modular manipulators for research as well as for industrial applications (Benhabib and Dai 1991; Cohen et al. 1992; Fukuda et al. 1992; Hui et al. 1993; Kotosaka et al. 1992; Matsumaru 1995; Paredis, Brown, and Khosla 1996). However, most of these systems lack the property of reconfigurability, which is key to the concept of rapidly deployable systems. The RMMS is particularly easy to reconfigure thanks to its integrated quick-coupling connectors.

Besides the ability to be quickly adaptable to a task, the RMMS has the advantage of being easily maintainable, and possibly inexpensive and reliable through high volume production. Furthermore, our approach could form the basis for the next generation of autonomous manipulators, in which the traditional notion of sensor-based autonomy is extended to *configuration-based autonomy*. Indeed, although a deployed system can have all the sensory and planning information it needs, it may still not be able to accomplish its task because the task is beyond the system's physical capabilities. A rapidly deployable system, on the other hand, could adapt its physical capabilities based on task specifications and, with advanced sensing, control, and planning strategies, accomplish the task autonomously.

A second important building block of a rapidly deployable manipulator system is a framework for the generation of control software. To reduce the complexity of software generation for real-time sensor-based control systems, a software paradigm called *software assembly* has been proposed in the Advanced Manipulators Laboratory at CMU. This paradigm combines the concept of reusable and reconfigurable software components, as is supported by the Chimera real-time operating system (Stewart and Khosla 1995), with a graphical user interface and a visual programming language, implemented in Onika (Gertz and Khosla 1994). This software framework and its application to the control of the RMMS are described in Chapter 2, in which we also discuss the electro-mechanical structure of the RMMS.

Although the software assembly paradigm provides the software infrastructure for rapidly programming manipulator systems, it does not solve the programming problem itself. Explicit programming of sensor-based manipulator systems is cumbersome due to the extensive amount of detail which must be specified for the robot to perform the task. The software synthesis problem for sensor-based robots can be simplified dramatically, by providing robust *robotic skills*, that is, encapsulated strategies for accomplishing common tasks in the robots task domain (Morrow and Khosla 1995). Such robotic skills can then be used at the task level planning stage without having to consider any of the low-level details. The issue of high-level planning and programming of rapidly deployable systems is not addressed in this thesis. We refer the interested reader to Carriker (1995), who presents a high-level robot planner for flexible assembly, and Morrow et al. (1995), who introduces sensorimotor primitives for robotic skill generation.

The third component of a rapidly deployable fault tolerant manipulator system is a an algorithm for planning fault tolerant trajectories. In this thesis, we focus on *fault tolerance* as a technique to achieve reliability in manipulator systems. The traditional approach to reliability has been that of fault intolerance, where the reliability of the system is assured by the use of high quality components. However, increasing system complexity and the necessity for long term operation have proven this approach inadequate. The system reliability can be further improved through redundancy. This design approach was already advocated in the early fifties by von Neumann in connection with the design of reliable computers: “The complete system must be organized in such a manner, that a malfunction of the whole automaton cannot be caused by the malfunctioning of a single component, ... , but only by the malfunctioning of a large number of them” (von Neumann 1956, p. 70). This is the basic principle of fault tolerance: *add redundancy to compensate for possible failures of components*. However, this does not mean that any kind of redundancy added to a system results in fault tolerance. Therefore, in Chapter 3, we will shed some light on the redundancy requirements for fault tolerant manipulators. That is, how much redundancy is needed and how should this redundancy be distributed over the manipulator structure?

The analysis of fault tolerant manipulators indicates that whether a task can be completed after a joint failure depends strongly on the joint angle at which the failure occurs. This observation is the basis for the global fault tolerant trajectory planning algorithm presented in Chapter 4. The idea is to avoid unfavorable joint angles *before* a failure occurs by carefully planning a fault tolerant joint space trajectory. If the manipulator follows this trajectory before failure, it is guaranteed that it can complete the task regardless of which joint fails and regardless of the time at which the failure occurs.

The redundancy provisions needed for fault tolerance can be incorporated only at a price of increased complexity. This drawback is partly overcome by the modular and structured design philosophy embodied in the RMMS project. Modularity in hardware and software has the advantage of facilitating testing during the design phase and therefore reducing the chances for unanticipated faults. Modules also constitute natural boundaries to which faults can be confined. By including fault detection and recovery mechanisms in critical modules, the effect of local faults remains internal to the modules, totally transparent to the higher levels of the manipulator system.

Finally, the most important component of the rapidly deployable fault tolerant manipulator system is the Task Based Design (TBD) software. The TBD problem can be illustrated by considering the *task space* and the *manipulator space*, shown in Figure 1-2. The task space is the multi-dimensional space of all possible tasks. Every point in this space defines a specific task. Similarly, consider the space of manipulator designs (the manipulator space), in which every point corresponds to one specific manipulator configuration. Every manipulator can execute a large number of tasks, and thus, every point in the manipulator space maps to a set of points in the task space. Finding this set of points in task space is the problem addressed by manipulator analysis. Because each configuration of the Reconfigurable Modular Manipulator System constitutes a different manipulator, the RMMS corresponds to a set of points in the manipulator space, each of which maps to a set of tasks in the task space. The RMMS as a whole maps onto the union of all these sets in the task space, which corre-

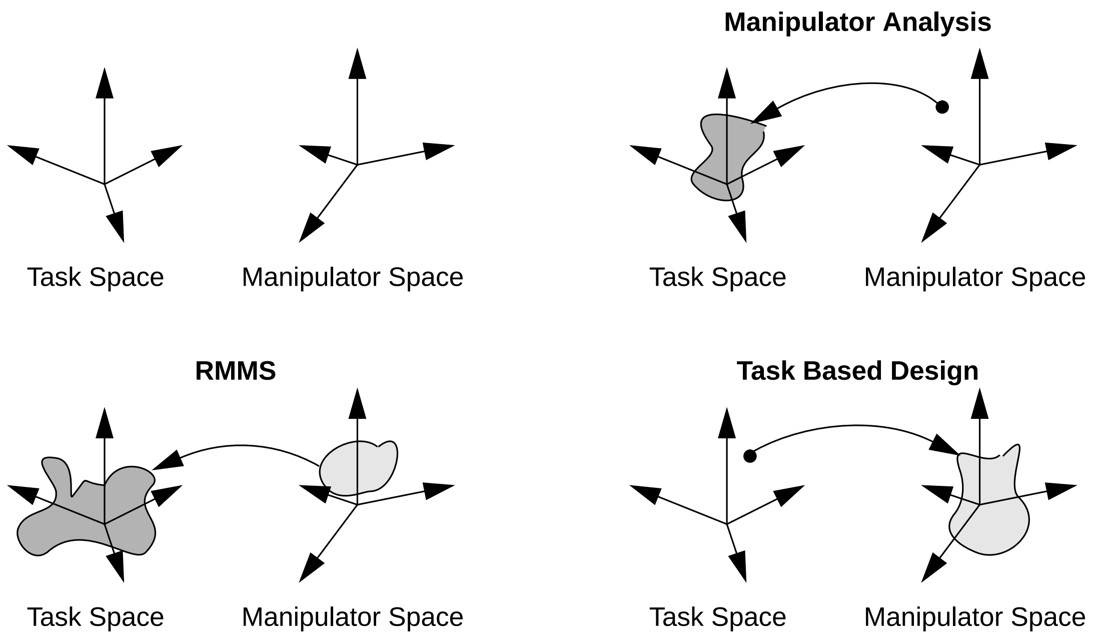


Figure 1-2. Mappings between task space and manipulator space.

sponds to a set of tasks that is larger than the task set of each individual configuration. This illustrates the increased flexibility and autonomy of the RMMS over traditional fixed configuration manipulators. In order to use the RMMS effectively, it is important to address the problem of finding the opposite mapping, that is, the mapping from task space to manipulator space. This is the problem addressed by *Task Based Design* (TBD): which manipulator configurations are optimally suited to perform a given task? The inputs to the TBD problem are the descriptions of the task and of the available manipulator modules; the output is a modular manipulator configuration, its base position, and the joint space trajectory optimally suited to perform the given task. In Chapter 5 of this thesis, we introduce a novel agent-based framework that provides a very comprehensive and fully integrated approach to the TBD problem. We will further demonstrate the power of this approach through an extensive performance analysis and a comprehensive example in Chapters 6 and 7.

1.2 Contributions

This thesis combines research efforts in three different areas: reconfigurable modular manipulators systems, manipulator fault tolerance, and task based design. The result is the realization of a rapidly deployable fault tolerant manipulator system. To achieve this goal we make the following contributions:

Rapidly Deployable Systems:

- We develop the hardware of a reconfigurable modular manipulator system: the RMMS.
- We implement and test a distributed reconfigurable control system that automatically adapts itself to the current manipulator configuration by building configuration independent kinematic and dynamic manipulator models.
- We seamlessly integrate RMMS simulation software with the real-time control software and hardware.

Manipulator Fault Tolerance:

- We formulate a simple yet comprehensive scenario for fault tolerance; it can handle a large variety of faults with one single recovery mechanism: immobilize the failing degree-of-freedom (DOF) by enabling its brake, and continue the task with the remaining DOFs.
- We prove that two degrees-of-redundancy are necessary and sufficient for general purpose fault tolerance.
- We provide an 8-DOF template for general purpose fault tolerant manipulator.
- We prove that, under certain conditions, one degree-of-redundancy is necessary and sufficient for task specific fault tolerance.
- Based on the idea that one can achieve fault tolerance by avoiding unfavorable joint angles *before* failure, we develop a global fault tolerant trajectory planning algorithm.
- We developed an efficient implementation of the global fault tolerant trajectory planning algorithm, which is used to evaluate the fault tolerant properties of candidate manipulator designs in an agent-based design framework.
- We implemented a fault tolerant recovery mechanism that allows a manipulator to continue its task uninterruptedly when a simulated joint failure occurs. We demonstrated this controller on the RMMS.

Task Based Design:

- We consider a very complete definition of the TBD problem, including energy consumption as an optimality criterion, and all of the following task constraints: trajectory reachability, joint position limits, joint velocity limits, joint torque limits, singularity avoidance, obstacle collision, and self-collision.
- We formulate an integrated solution approach to the TBD problem, based on Genetic Algorithms. This approach considers simultaneously the manipulator kinematics and dynamics, trajectory planning, and control.
- We include problem specific knowledge in the genetic algorithm to reduce the size of the search space.

- We introduce the concept of “progressive evaluation,” which drastically reduces the average computation cost of fitness evaluations.
- We introduce an agent-based implementation of the genetic algorithm, which increases the computational power through distributed computing, and provides a modular composable framework for adapting the design system to the design task at hand.
- We perform a detailed performance analysis of the agent-based design framework, by comparing it to exhaustive search, random search, and multiple restart statistical hill-climbing.
- We propose the Fisher sign test, to compare the performance of statistical search algorithms.
- We solve a comprehensive TBD problem, which consists of designing a modular fault tolerant manipulator for a satellite docking operation with the space shuttle. The design task includes the determination of the optimal position and orientation of the space shuttle with respect to the satellite, and the determination of the corresponding fault tolerant trajectory.

1.3 Overview

We conclude this introduction with an overview of the organization of the remainder of this thesis:

- Chapter 2 provides an overview of the development of the RMMS hardware and corresponding control software.
- Chapter 3 introduces the concept of manipulator fault tolerance and investigates how many redundant degrees-of-freedom are necessary and sufficient to achieve fault tolerance.
- Chapter 4 develops a global fault tolerant trajectory planning algorithm, which is an essential component of a task specific fault tolerant manipulator system.
- Chapter 5 is the most important chapter of this thesis. It combines the concept of

rapidly deployable systems with the global fault tolerant trajectory planning algorithm, to create an *agent-based Task Based Design framework* for the design of rapidly deployable fault tolerant manipulators.

- Chapter 6 presents an extensive analysis of the TBD problem, including a performance analysis of the agent-based design system, and several criteria for characterizing the TBD problem itself.
- Chapter 7 illustrates the power of the agent-based design system with a comprehensive example of a manipulator design for a satellite docking operation.
- Chapter 8 summarizes the achievements and conclusions presented in this thesis.
- Appendix A lists the module specifications of all the RMMS modules that are used in the TBD examples in this thesis.

Chapter 2

RMMS: A Reconfigurable Modular Manipulator System

2.1 Introduction

The central building block of a rapidly deployable fault tolerant manipulator system is the RMMS, a Reconfigurable Modular Manipulator System. The RMMS consists of a set of modules of different sizes and performance specifications that can be quickly configured into a manipulator that is optimally suited to perform a given task. To achieve this important property of *reconfigurability*, the system requires special hardware and software provisions.

The first part of this chapter focuses on the mechanical and electrical hardware development of the RMMS. Special quick-coupling connectors combined with a modular communication interface guarantee the rapid reconfigurability of the RMMS hardware. The second part of this chapter describes the modular and reconfigurable control software, as implemented with the Chimera real-time operating system and the Onika visual programming language. An experiment illustrates the manipulator configuration independent programming and control capabilities of the implementation.

2.2 Self-Contained Hardware Modules

In most industrial manipulators, the controller is a separate unit housing the sensor interfaces, power amplifiers, and control processors for all the joints of the manipulator. A large number of wires is necessary to connect this control unit with the sensors, actuators and brakes located in each of the joints of the manipulator. The large number of electrical connections and the non-extensible nature of such a system layout make it infeasible for modular manipulators. The solution we propose is to distribute the control hardware to each individual module of the manipulator. These modules become then self-contained units which include sensors, an actuator, a brake, a transmission, a sensor interface, a motor amplifier, and a communication interface, as is illustrated in Figure 2-1. As a result, only six wires are required for power distribution and data communication.

2.2.1 Mechanical Design

The goal of the RMMS project is to have a wide variety of hardware modules available. So far, we have built four kinds of modules (two of which are shown in Figure 2-2): the manip-

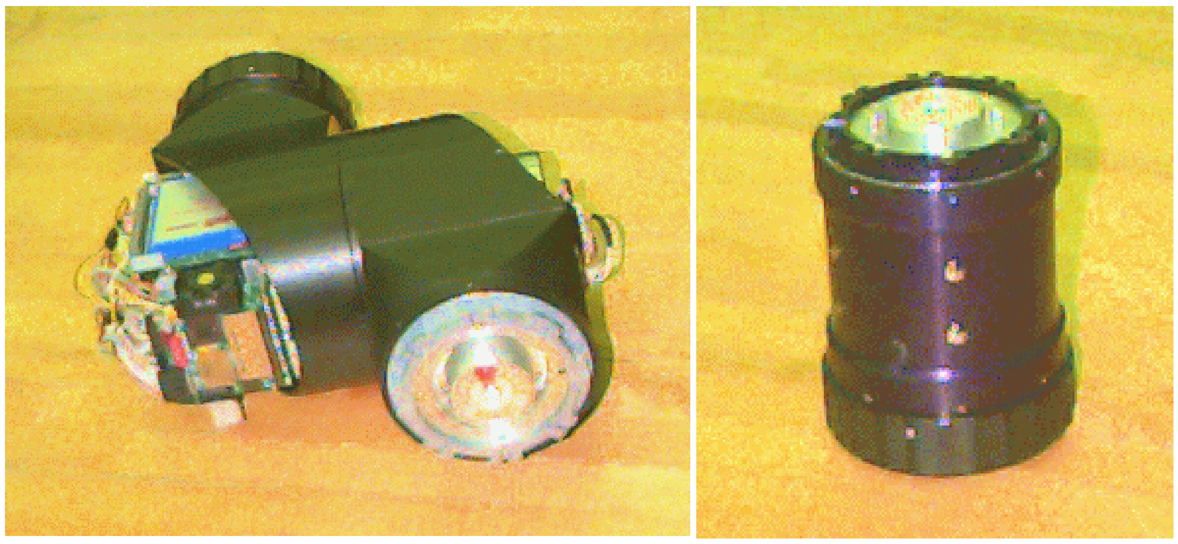


Figure 2-2. An RMMS pivot joint and link module.

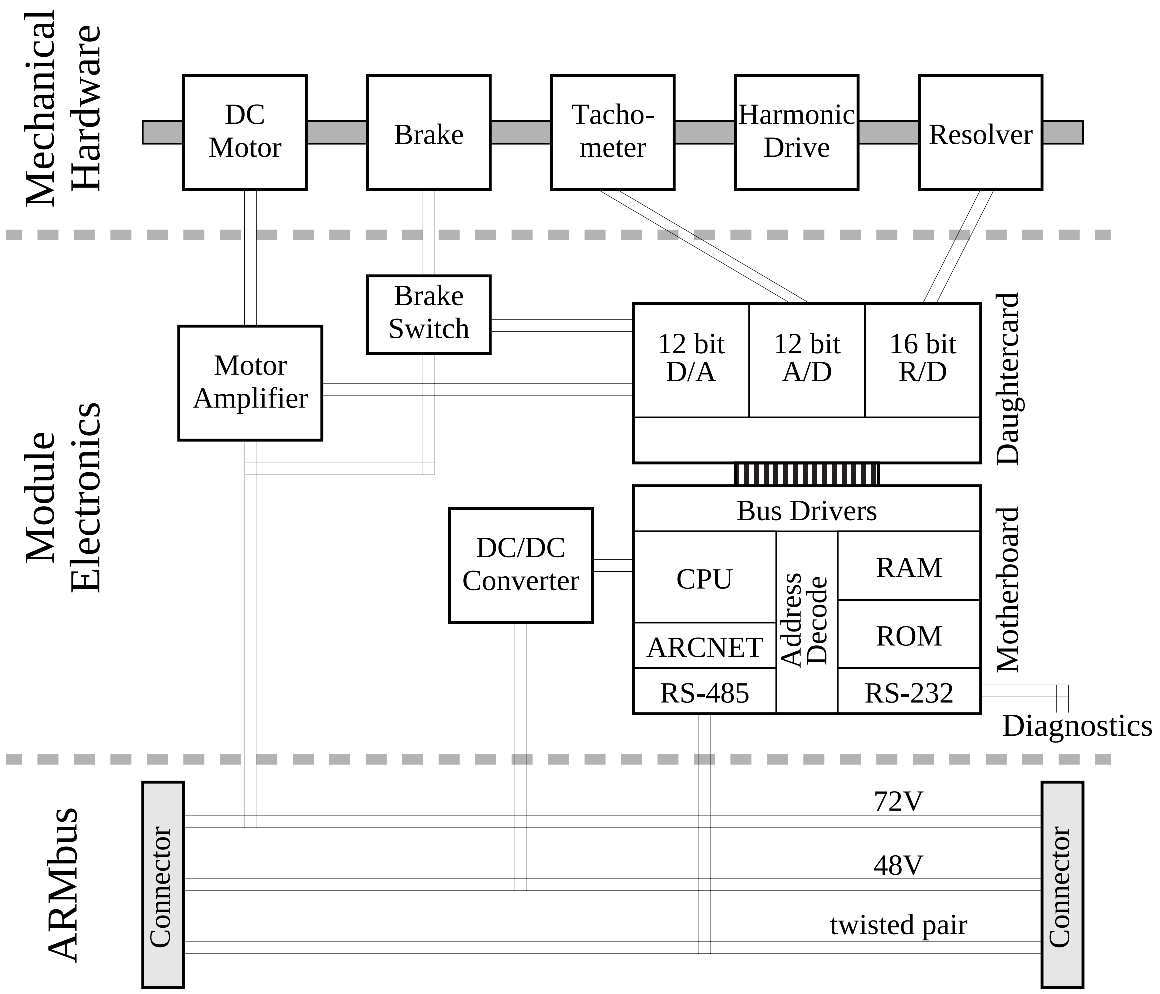


Figure 2-1. Diagram of a self-contained RMMS module.

ulator base, a link module, three pivot joint modules, and one rotate joint module. The base module and the link module have no degrees-of-freedom; the joint modules have one degree-of-freedom each. The mechanical design of the joint modules compactly fits a DC-motor, a fail-safe brake, a tachometer, a harmonic drive, and a resolver.

The pivot and rotate joint modules use different outside housings to provide the right-angle or in-line configuration, respectively, but are identical internally. Figure 2-3 shows in cross-section the internal structure of a pivot joint. Each joint module includes a DC torque motor and a 100:1 harmonic-drive speed reducer; the modules are rated at a maximum speed of

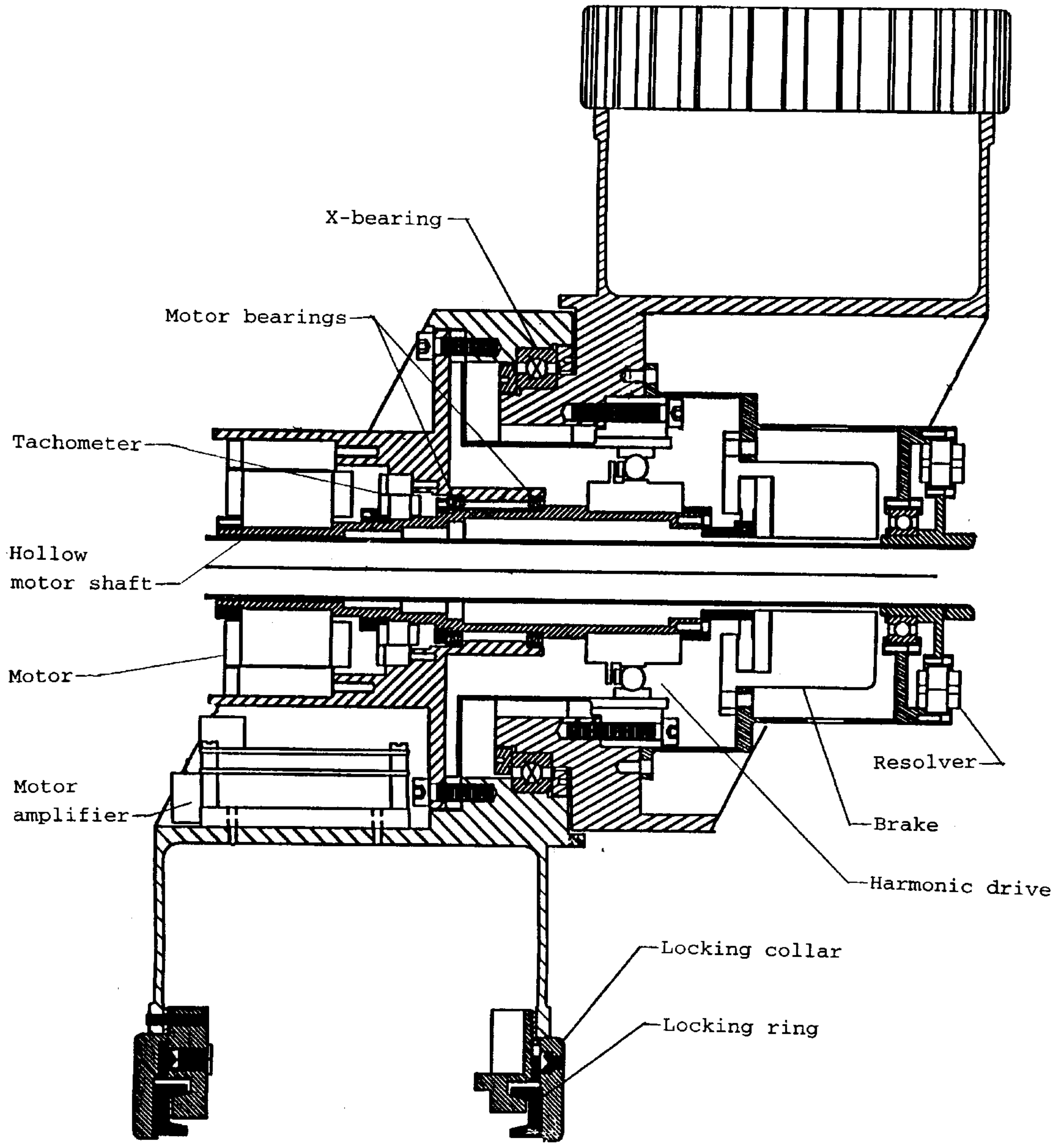


Figure 2-3. Assembly drawing of a pivot joint module.

0.9rad/s and maximum torque of 290Nm. Each module has a mass of approximately 10.7kg. A single, compact, X-type bearing connects the two joint halves and provides the needed overturning rigidity. A hollow motor shaft passes through all the rotary components, and provides a channel for passage of cabling with minimal flexing.

2.2.2 Electronic Design

The custom-designed on-board electronics are also designed according to the principle of modularity. Each RMMS module contains a motherboard which provides the basic functionality. Onto the motherboard, one can stack daughtercards to add module specific functionality.

The motherboard consists of a Siemens 80C166 microcontroller, 64K of ROM, 64K of RAM, an SMC COM20020 universal local area network controller with an RS-485 driver, and an RS-232 driver. The function of the motherboard is to establish communication with the host interface and to perform the low-level control of the module, as is explained in more detail in Section 2.2.4. The RS-232 serial bus driver allows for simple diagnostics and software prototyping.

A stacking connector permits the addition of an indefinite number of daughtercards with various functions, such as sensor interfaces, motor controllers, and RAM expansion. In our current implementation, only modules with actuators include a daughtercard. This card contains a 16 bit resolver to digital converter, a 12 bit A/D converter to interface with the tachometer, and a 12 bit D/A converter to control the motor amplifier; we have used an of-the-shelf motor amplifier (Galil Motion Control model SSA-8/80) to drive the DC-motor. For modules with more than one degree-of-freedom, for instance a wrist module, more than one such daughtercard can be stacked onto the same motherboard.

2.2.3 Integrated Quick-Coupling Connectors

To make a modular manipulator be reconfigurable, it is necessary that the modules can be easily connected with each other. We have developed a quick-coupling mechanism with which a secure mechanical connection between modules can be achieved by simply turning a ring hand-tight; no tools are required. As shown in Figure 2-4, keyed flanges provide pre-

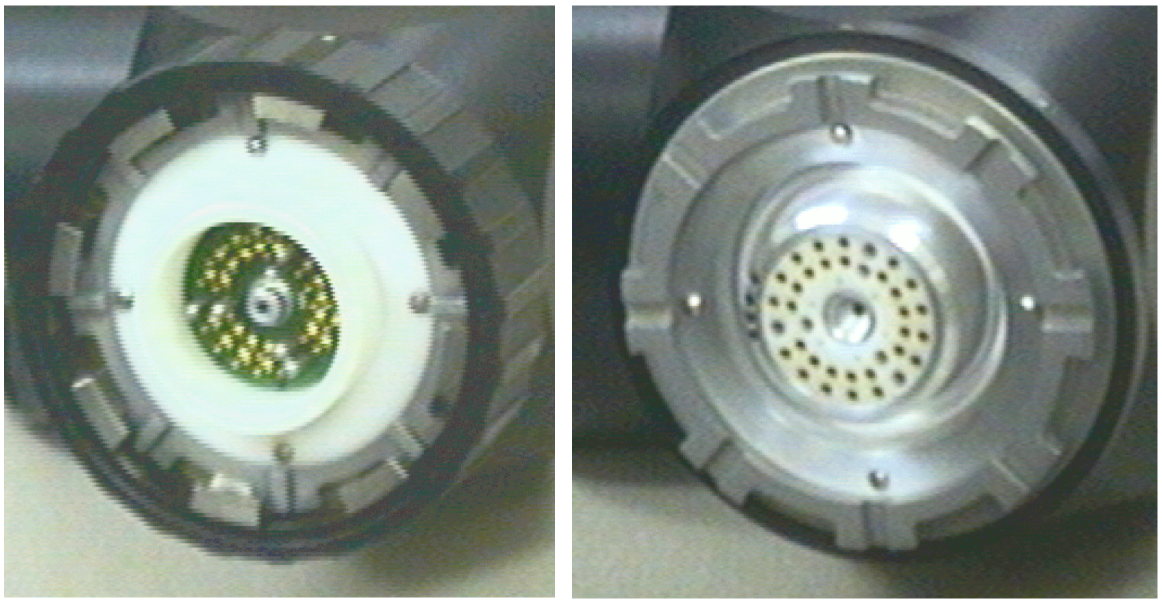


Figure 2-4. A male and female RMMS connector.

cise registration of the two modules. Turning of the locking collar on the male end produces two distinct motions: first the fingers of the locking ring rotate (with the collar) about 22.5 degrees and capture the fingers on the flanges; second, the collar rotates relative to the locking ring, while a cam mechanism forces the fingers inward to securely grip the mating flanges. A ball-transfer mechanism between the collar and locking ring automatically produces this sequence of motions.

At the same time the mechanical connection is made, pneumatic and electronic connections are also established. Inside the locking ring is a modular connector that has 30 male electrical pins plus a pneumatic coupler in the middle. These correspond to matching female components on the mating connector. Sets of pins are wired in parallel to carry the 72V-25A power for motors and brakes, and 48V–6A power for the electronics. Additional pins carry signals for two RS-485 serial communication busses and four video busses. A plastic guide collar plus six alignment pins prevent damage to the connector pins and assure proper alignment. The plastic block holding the female pins can rotate in the housing to accommodate the eight different possible connection orientations (8@45 degrees). The relative orientation is automatically registered by means of an infrared LED in the female connector and eight photodetectors in the male connector.

2.2.4 ARMbus Communication System

Each of the modules of the RMMS communicates with the host interface over a local area network called the ARMbus; each module is a node of the network. The communication is done in a serial fashion over an RS-485 bus which runs through the length of the manipulator. We use the ARCNET protocol (ARCNET Trade Association 1992) implemented on a dedicated IC (SMC COM20020). ARCNET is a deterministic token-passing network scheme which avoids network collisions and guarantees each node its time to access the network. Blocks of information called packets may be sent from any node on the network to any one of the other nodes, or to all nodes simultaneously (broadcast). Each node may send one packet each time it gets the token. The maximum network throughput is 5Mb/s.

The first node of the network resides on the host interface card, as is depicted in Figure 2-5. In addition to a VME address decoder, this card contains essentially the same hardware one can find on a module motherboard. The communication between the VME side of the card and the ARCNET side occurs through dual-port RAM.

There are two kinds of data passed over the local area network. During the manipulator initialization phase, the modules connect to the network one by one, starting at the base and ending at the end-effector. On joining the network, each module sends a data-packet to the

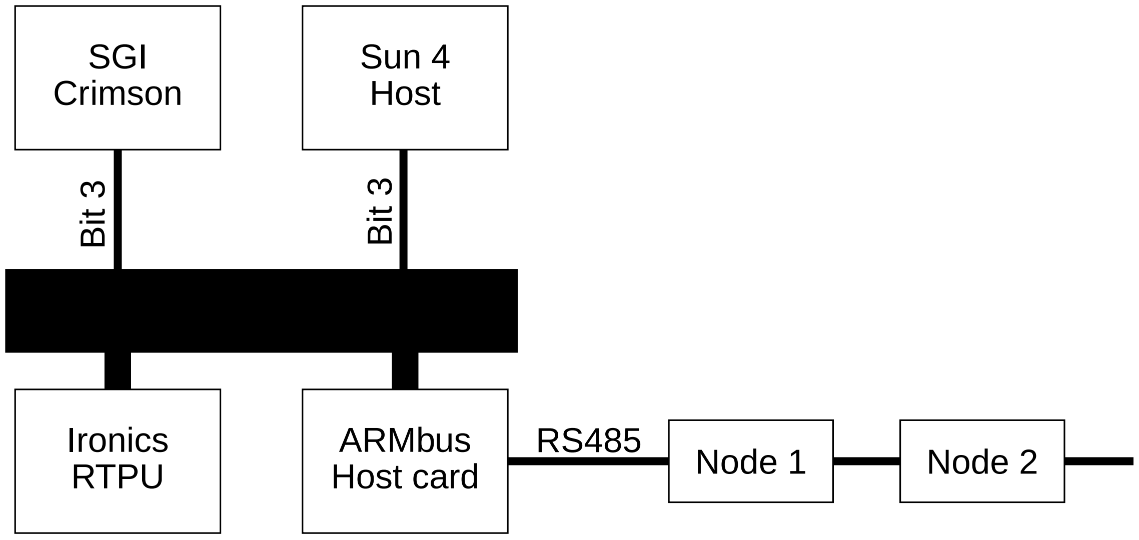


Figure 2-5. The RMMS computing hardware

host interface containing its serial number and its relative orientation with respect to the previous module. This information allows the controller to determine automatically the current manipulator configuration.

During the operation phase, the host interface communicates with each of the nodes at 400Hz. The data that is exchanged depends on the control mode—centralized or distributed. In centralized control mode, the torques for all the joints are computed on the VME-based real-time processing unit (RTPU), assembled into a data-packet by the microcontroller on the host interface card, and broadcast over the ARMbus to all the nodes of the network. Each node extracts its torque value from the packet and replies by sending a data-packet containing the resolver and tachometer readings. In distributed control mode, on the other hand, the host computer broadcasts the desired joint values and feed-forward torques. Locally, in each module, the control loop can then be closed at a frequency much higher than 400Hz. The modules still send sensor readings back to the host interface to be used in the computation of the subsequent feed-forward torque.

2.3 Modular and Reconfigurable Control Software

As we have shown in Section 2.2, the keys to rapidly deployable hardware are modularity and reconfigurability. The same concepts apply also to the software aspect of rapidly deployable systems. The development of real-time control software for sensor-based robotic systems is complicated and time-consuming. To reduce this complexity, a software paradigm, called *software assembly*, has been proposed in the Advanced Manipulators Laboratory at Carnegie Mellon University. This paradigm combines the concept of reusable and reconfigurable software components with a graphical user interface and visual programming language, as is shown in Figure 2-6.

2.3.1 The Chimera Real-Time Operating System

Chimera is a real-time operating system that supports the use of reconfigurable and reusable software components. It is founded upon the notion of port-based objects, in which the port

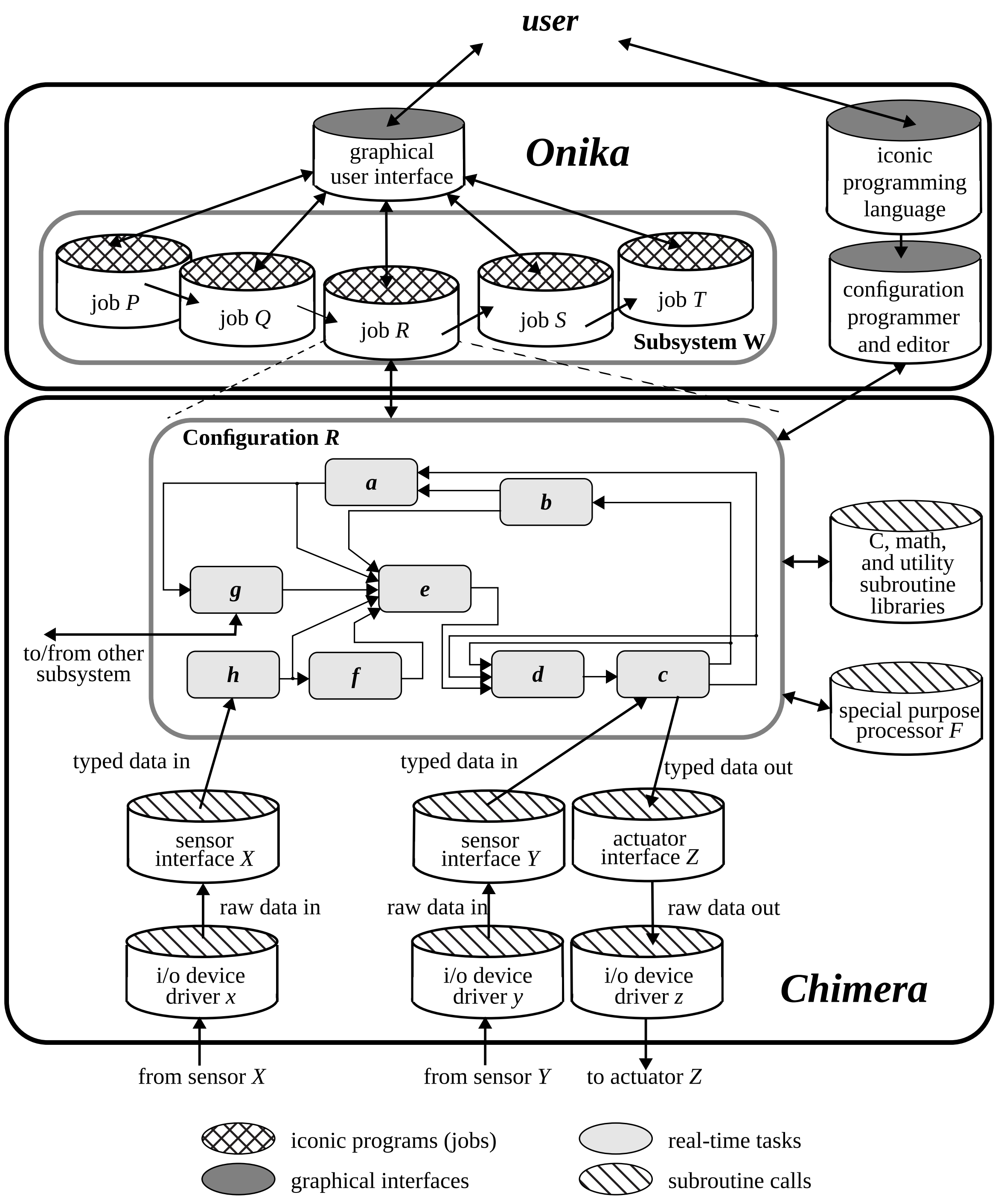


Figure 2-6. Framework for software assembly, as implemented in Chimera and Onika. (from Stewart 1994)

automaton computation model of concurrent processes is combined with object based design of software. The internal methods of a port-based object are hidden (as in object oriented programming), while communication is achieved through input, output, and resource ports (as in automata). Each object executes as a separate task on one of the processors in a distributed environment. Communication between objects is performed through a global state variable table that is stored in shared memory. The variables in this table are a union of the input port and output port variables of all the objects. The port names are used to perform the binding between the objects. Instead of accessing the global state variable table directly, each objects uses a local copy of the table, in which only the variables used by the object are kept up-to-date. Because the global table is locked during updates, the data is always transferred between global and local state variable tables as a complete set. The result of the global state variable table mechanism is that each of the objects can execute autonomously, without blocking when another task is using a shared resource. More details about Chimera can be found in Stewart and Khosla (1995) and Stewart (1994).

2.3.2 The Onika Visual Programming Language

Onika is a high-level graphical interface that facilitates the use of the port-based objects created within Chimera. Onika provides manipulation, encapsulation, and sequencing capabilities for the objects at two distinct levels: high level and middle level (the low level is defined to be the coding of specific objects within Chimera).

The middle-level interface allows a system designer to combine low-level software objects into object *configurations*, such as the one shown in Figure 2-7. This level provides a “canvas” on which the user can drag-and-drop object icons to interactively assemble the real-time software while it is running on the real time processing units. No knowledge of textual coding is required, but merely a good working knowledge of control theory and familiarity with control block diagrams. The configurations specified at the middle level represent particular tasks that the system designer requires to complete some higher level *application*. For instance, “move the robot to a point in space” could be a task accomplished by a single configuration, while making a series of moves to complete an assembly could be an application. Entire object configurations can be iconified and exported to the dictionary in the high

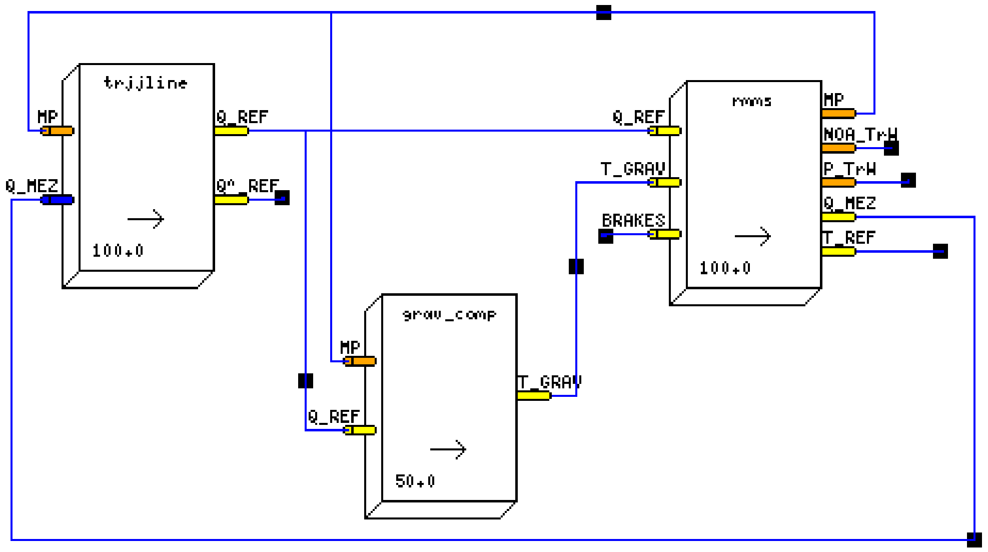


Figure 2-7. An object configuration for joint space control, created at the middle level of Onika.

level interface.

At the high level, which is intended for non-programmers and unskilled users, a storyboard is presented with a dictionary of puzzle-piece-like task icons. The task icons are shape and color coded for proper connection. They are the building blocks for the visual programming language of the high level of Onika. A user can drag-and-drop the icons onto the storyboard to create various applications. Figure 2-8 illustrates the program used to execute the task in the experiments of Section 2.4.

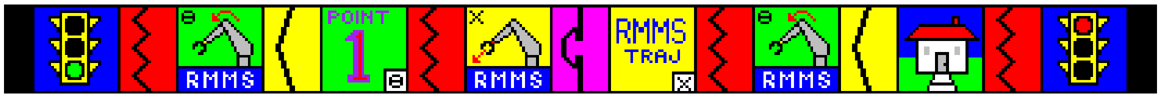


Figure 2-8. An Onika application.

2.3.3 RMMS Control Software

The port-based Chimera objects used to control the RMMS are listed in Table 2-1. The *trjj-gen*, *dls*, and *grav_comp* components require the knowledge of certain configuration dependent parameters of the RMMS, such as the number of degrees-of-freedom, and the Denavit-

name	**function**	**description**
rmms	RMMS interface	Initializes RMMS; Computes and sends raw joint positions to the joint modules; Receives and converts raw sensor readings.
grav_comp	gravity compensation	Computes the gravity compensation torques based on the kinematic and dynamic manipulator models.
fkjac	forward kinematics and Jacobian	Computes the forward kinematics and Jacobian, based on the kinematic manipulator model.
dls	damped least-squares kinematic controller	Computes the desired joint space position according to the damped least-squares kinematic control algorithm.
trjjgen	joint trajectory generator	Generates a 5th order polynomial interpolated joint space trajectory.
trjcgen	Cartesian trajectory generator	Generates a 5th order polynomial interpolated Cartesian space trajectory.

Table 2-1. Chimera software module description.

Hartenberg parameters. During the initialization phase, the RMMS interface establishes contact with each of the hardware modules to determine automatically which modules are being used and in which order and orientation they have been assembled. For each module, a data file with a parametric model is read. By combining this information for all the modules, kinematic and dynamic models of the entire manipulator are built.

After the initialization, the rmms object operates in a distributed control mode in which the microcontrollers of each of the RMMS modules perform PID control locally at 1900Hz. The communication between the modules and the host interface is at 400Hz, which can differ from the cycle frequency of the *rmms* Chimera object. Since we use a triple buffer mechanism (Stewart 1994) for the communication through the dual-port RAM on the ARMbus

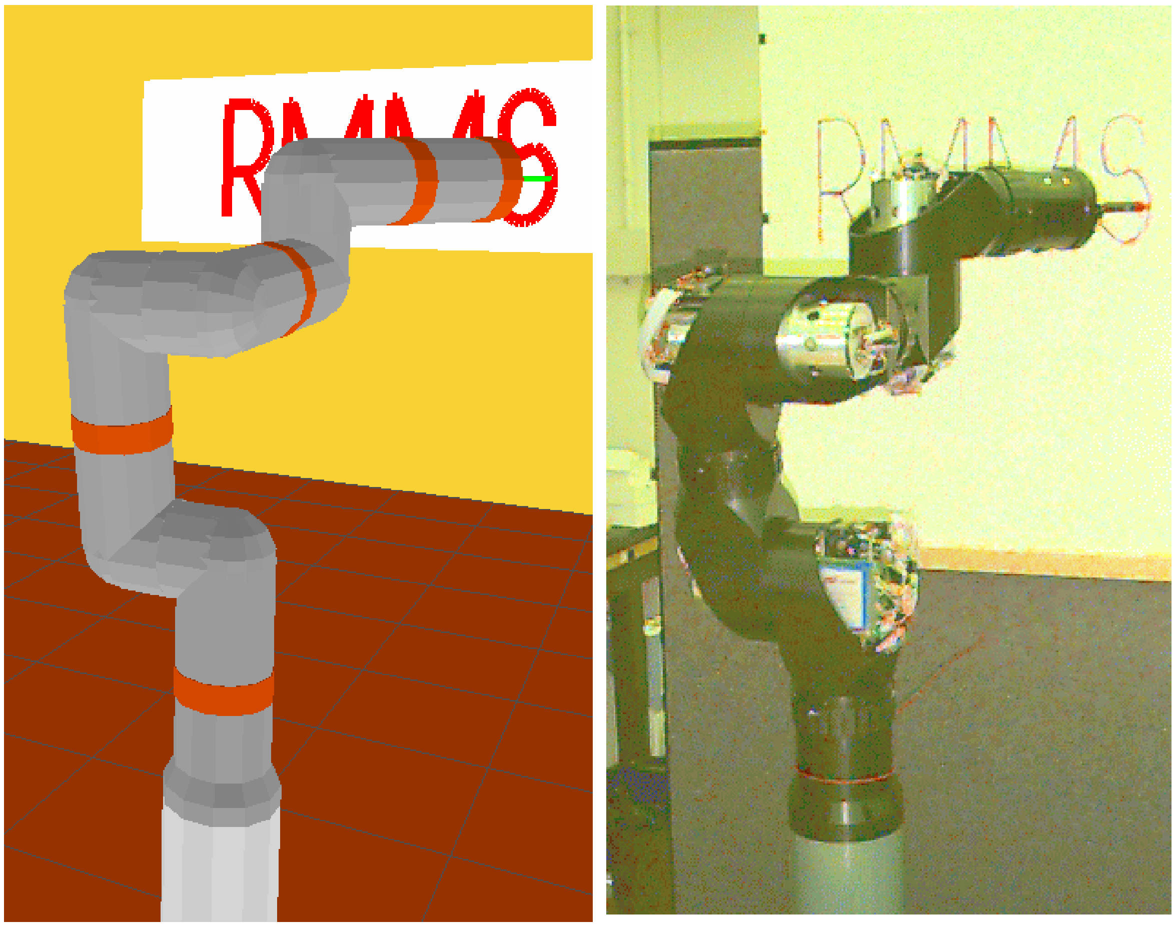


Figure 2-9. An RMMS configuration: simulation and hardware

host interface, no synchronization or handshaking is necessary.

Because closed form inverse kinematics do not exist for all possible RMMS configurations, we have to use iterative inverse kinematics. We have implemented a damped least-squares kinematic controller that can automatically adjust itself to sudden joint immobilization. This is important for fault recovery of fault tolerant manipulators. A more detailed description of the fault tolerant kinematic controller can be found in Chapter 4, Section 4.6.

2.3.4 Seamless Integration of Simulation

To assist the user in evaluating whether an RMMS configuration can successfully complete a given task, we have built a simulator. The simulator is based on the TeleGrip robot simulation software from Deneb Inc., and runs on an SGI Crimson which is connected with the real-time processing unit through a Bit3 VME-to-VME adaptor, as is shown in Figure 2-5. A graphical user interface allows the user to assemble simulated RMMS configurations very much like assembling the real hardware. Completed configurations can be tested and programmed using the TeleGrip functions for robot devices. The configurations can also be interfaced with the Chimera real-time software running on the same RTPUs used to control the actual hardware. As a result, it is possible to evaluate not only the movements of the manipulator but also the real-time CPU usage. Figure 2-9 shows an RMMS simulation compared with the actual task execution.

2.4 Experiments

In this section, we compare two different RMMS configurations executing the same task to demonstrate the configuration independence of the RMMS controller and the Onika high level programming language. The task consists of writing “RMMS” on a white board, as is shown in Figure 2-9. The white board is positioned 0.65m from the manipulator base and the word “RMMS” measures 0.2m by 0.553m (see Figure 2-11). The whole path is 3.166m long and is traversed in 35.62 seconds.

The first of the two manipulator configurations used in this experiment is shown in Figure 2-9. It has 3 rotational degrees-of-freedom. The first two rotation axes are horizontal and parallel to the writing surface. The third rotation axis is perpendicular to the first two. The second configuration depicted in Figure 2-10 has also three rotational degrees-of-freedom. The first rotation axis is again horizontal but this time perpendicular to the writing surface. The second and third rotation axes are parallel to each other and perpendicular to the first axis.

These configurations are only two of the many configurations that one can build with the five RMMS modules currently available. Taking into account the axial symmetry of the link and rotate joint module, and the functional equivalence of the three pivot modules, one can build 3,520 different 3-DOF freedom manipulators and 12,288 different 4-DOF manipula-

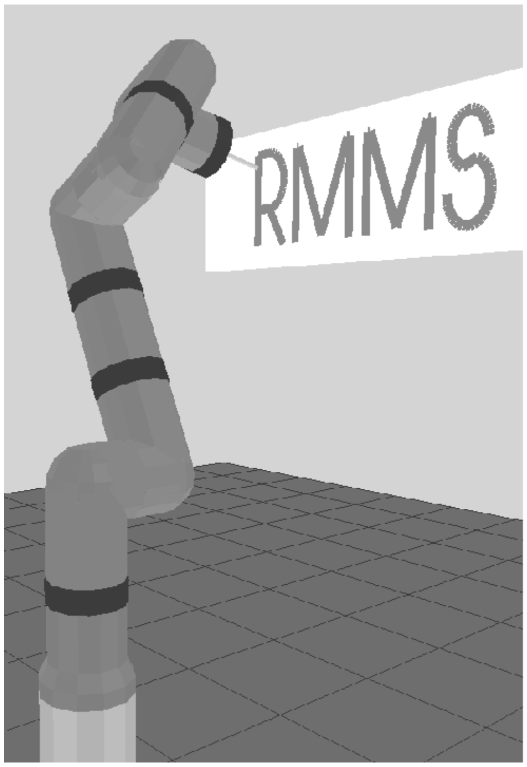


Figure 2-10. The second configuration used in the experiments.

tors. In Chapter 6, we will apply the agent-based design framework, developed in Chapter 5, to the same task that we consider in this section, and determine which of the 3,520 3-DOF manipulators is optimally suited to perform this task.

This section focuses on the actual task execution. Even though the two manipulator configurations are totally different in their kinematic and dynamic structure, they can both execute the task with the exact same Chimera and Onika programs.

The Onika program shown in Figure 2-8, consists of three sub-tasks: move under joint space control to the starting point; then, switch to Cartesian control and follow the Cartesian path defined in the file “RMMS.traj”; finally, switch back to joint space control and move to the home position. At the beginning of the task execution, Onika spawns all the Chimera modules listed in Table 2-1, but commands only the *rmms*, *grav_comp*, and *trjjgen* modules to start cycling. These three modules constitute the joint space control configuration. At the end of the joint space controlled sub-task, the *trjjgen* module turns itself off, and Onika switches to Cartesian control by commanding the *fkjac*, *dls*, and *trjcgen* modules to start cycling. Since the *rmms* and *grav_comp* modules appear in both the joint space control configuration and Cartesian control configuration, they continue uninterruptedly. Similarly, at the end of the “RMMS” Cartesian trajectory, the *trjcgen* module turns itself off, after which Onika turns the *fkjac* and *dls* modules also off, and the *trjjgen* module back on for the next sub-task.

Figure 2-11 illustrates the performance of the damped least squares kinematic controller in combination with the distributed PID controller. The end point error is similar for both configurations—less than 2.5mm at any time. There are three sources of error: the damped least-squares kinematic controller, the distributed PID controller, and the incorrectly calibrated kinematic parameters. The error shown in Figure 2-11b is based on the measured joint angles, so that it includes the controller errors but not the calibration errors. As a result, the actual end-point error will be slightly larger.

2.5 Summary

This chapter provided an overview of the development of the RMMS hardware and control software. The fully self-contained RMMS modules owe their rapid reconfigurability to two important innovations: the quick-coupling connectors, and the ARMbus modular communication and control system. The control software has also been implemented in a modular and reconfigurable framework, using the Chimera real-time operating system and the Onika visual programming language. This control implementation allows for manipulator configuration independent programming and control, as has been illustrated with a final example.

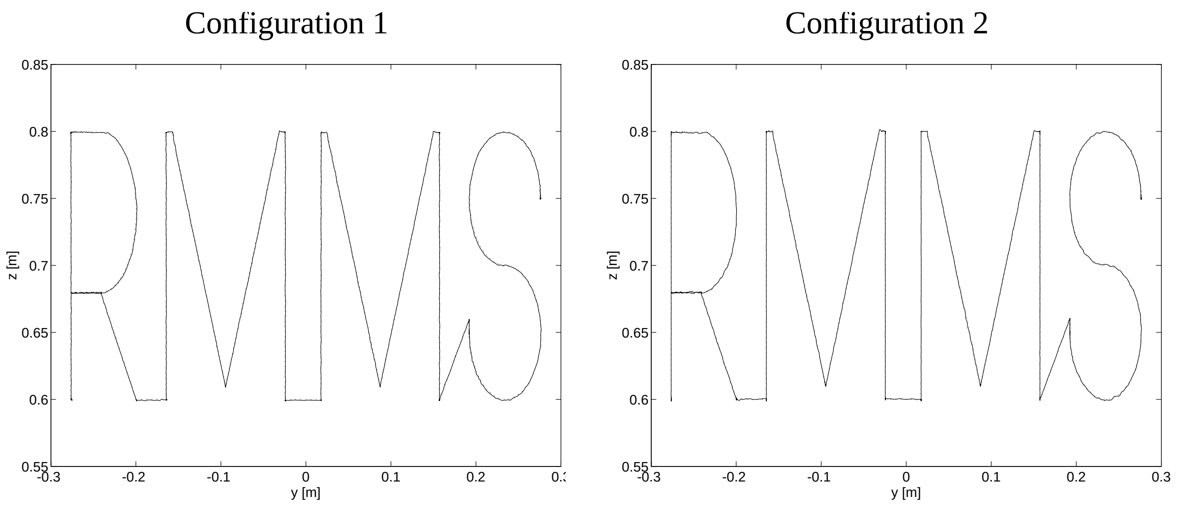


Figure 2-11a. measured path (solid) and desired path (dotted).

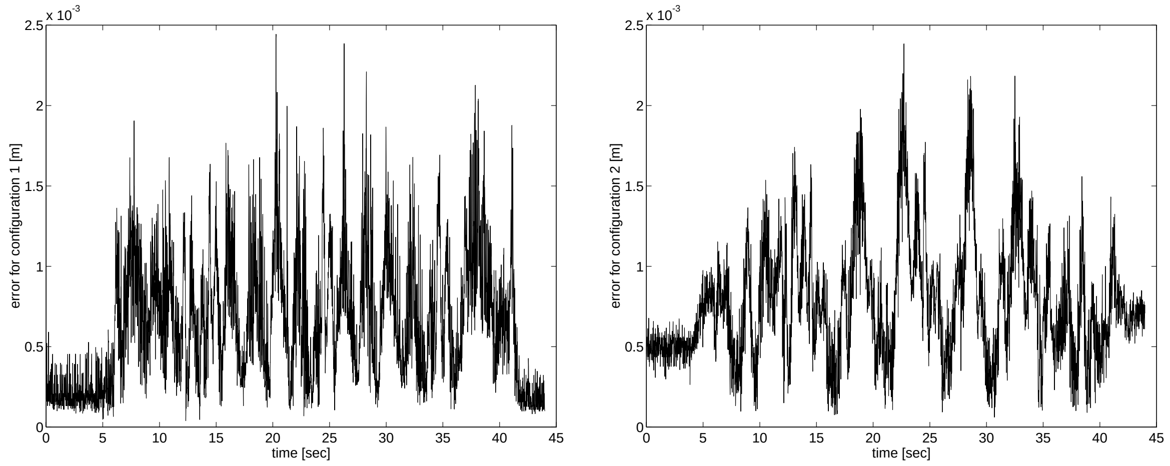


Figure 2-11b. End point position error.

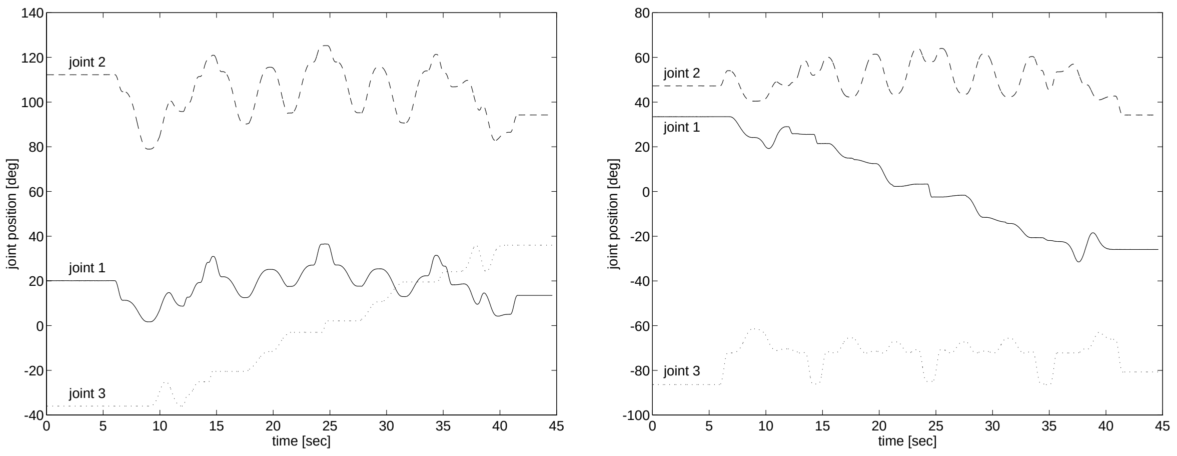


Figure 2-11c. Joint angles.

Figure 2-11. Comparison of the same task executed by two different manipulator configurations.

Chapter 3

Fault Tolerant Manipulators

3.1 Introduction

As we mentioned in Chapter 1, an important requirement for rapidly deployable manipulators in hazardous and remote environments is that they be extremely reliable. We propose to achieve a high level of reliability through fault tolerance, that is, by including redundancy to compensate for possible failures of components. Yet, one can add redundancy to a manipulator in many different ways, not all of which are equally effective at increasing the system's reliability. In this chapter, we present a simple but comprehensive scenario for fault tolerance and describe how our approach relates to the past work on manipulator fault tolerance. We then focus on an important question with respect to the design of fault tolerant manipulators: how many degrees-of-redundancy are necessary and sufficient to achieve fault tolerance? The answer to this question depends on the assumptions that are made about the task and about the manipulator's redundancy resolution algorithm. We consider two cases: *general purpose fault tolerance* and *task specific fault tolerance*.

3.2 Approach

Over the past decade, a lot of research has been done in fault tolerance for computer systems (refer to Johnson (1989) for an overview), but only recently has the concept been applied in robotics. Most of the work in fault tolerant robotics is directly based on the results from computer science, and can be classified in three categories:

1. Design of fault tolerant robots,
2. Fault detection and identification (FDI),
3. Fault recovery and intelligent control.

When designing a fault tolerant manipulator, one should decide where to include redundancy so that the overall reliability is maximum. One should distinguish between hardware, software, analytical, information, and time redundancy (Johnson 1989). Our focus will be on hardware redundancy, which consists of actuation, sensor, communication and computing redundancy. Each of these types of redundancies can still be implemented at different levels. In Sreevijayan (1992), for instance, a four-level subsumptive architecture for actuation redundancy is proposed:

1. Dual actuators—extra actuators per joint,
2. Parallel structures—extra joints per DOF,
3. Redundant manipulators—extra DOFs per manipulator arm,
4. Multiple arms—extra arms per manipulator system.

A system can possibly be designed with redundancies at all four levels, resulting in the ability to sustain multiple simultaneous faults.

An example of a fault tolerant design for the space shuttle manipulator is described in Wu et al. (1995). Fault tolerance is here guaranteed by using a differential gear train with dual input actuators for every DOF—an implementation of the first level of the four-level subsumptive architecture. In this thesis, we are more interested in achieving fault tolerance using redundant DOFs (Level 3), and, accordingly, propose an alternative space shuttle manipulator design in Chapter 7. We envision the following scenario for level three fault tolerance:

> *A fault detection and identification algorithm monitors the proper functioning of each DOF of a redundant manipulator. As soon as it detects a failure of a subcomponent, an intelligent controller immobilizes the corresponding DOF by activating its fail-safe brake. Automatically, the controller also adapts the joint trajectory to the new manipulator structure, so that the task can be continued without interruption.*

The strength of this scenario resides in the fact that it is simple and, yet, can handle a large variety of possible faults, ranging from sensor failures to transmission and actuation failures. All these failures can be treated in the same manner, namely, by eliminating the whole DOF through immobilization.

English and Maciejewski (1996) consider a different scenario, namely, *free-swinging failures*, which corresponds to a hardware or software failure that causes the loss of torque (or force) on a joint. They define several criteria that can be used for null-space optimization of a redundant manipulator to reduce the effect of an anticipated free swinging failure. The criteria are designed to minimize torque, acceleration, or swing angle after failure. In this thesis, we do not consider this free-swinging scenario as a separate case, because the failures that cause free-swinging can be handled equally well with the joint locking scenario if one turns on the brakes of failing joints.

Although fault detection and identification (FDI) is an important part of our scenario for fault tolerance, we will not cover this subject in this thesis. Instead we refer to the following references: Chow and Willsky (1984), Stengel (1988), Ting, Tosunoglu, and Tesar (1993), and Visinsky, Walker, and Cavallaro (1993 and 1994). Visinsky, Walker, and Cavallaro (1993) present an FDI algorithm along the lines of Chow and Willsky (1984)—based on the concept of analytical redundancy. The result is a set of four simple equations which test for consistency between the measured position and velocity and the expected acceleration and jerk. This FDI algorithm fits into a three-layer intelligent control framework, consisting of a servo layer, and interface layer and a supervisory layer. The main problem presented to the intelligent controller is to distinguish between failures, disturbances and modeling errors, and to respond to each in the proper way. An overview of intelligent fault tolerant control is given by Stengel (1988). He reported a range of approaches beginning with robust control, progressing through parallel and analytical redundancy, and ending with rule-based systems and artificial neural networks. The task of the robotics researcher is to apply and modify these approaches to the highly non-linear dynamics of robot manipulators.

After a fault has been detected, the failing DOF is immobilized by activating its brake. In Pradeep et al. (1988), the authors analyze the effect of the immobilization of one of the DOFs of three commercial manipulators. They conclude that the robots with decoupled DOFs are more severely “crippled” by the loss of a joint than the ones with strongly coupled DOFs. This can be translated into the guideline that, for the design of fault tolerant manipulators, strong coupling between the DOFs is highly desirable. The results presented by Roberts and Maciejewski (1996) and Lewis and Maciejewski (1994a) can be interpreted in a similar way. A kinematic fault tolerance measure is defined as the minimum kinematic dexterity after joint failure. The maximum kinematic fault tolerance is achieved in a manipulator posture in which each joint contributes equally to the null-space motion—a posture with strong coupling between the DOFs. For a manipulator with at least one decoupled DOF, the kinematic fault tolerance measure is always minimal, that is, zero. The same measure can be used as a criterion for the redundancy resolution of the fault-free manipulator. It is shown that the chances for task completion, after immobilization of one joint due to failure, are much better than when traditional pseudo-inverse control is used. However, due to the local nature of the fault tolerance measure, completion of the task cannot be guaranteed on a global scale (Lewis and Maciejewski 1994b).

An important conclusion is that the ability to recover from a fault depends strongly on the joint trajectory followed by the fault-free manipulator system. This conclusion led us to explore two approaches to the problem of manipulator fault tolerance. The two approaches differ in the assumptions that are made with regard to the task and with regard to the choice of redundancy resolution algorithm. In a first approach, the goal is to design a *general pur-*

pose fault tolerant manipulator. We assume that the task is only characterized by the size and position of the *task space* which is the portion of the Cartesian output space in which the task will take place. No assumptions are made about the path that needs to be followed within the task space or about the redundancy resolution algorithm used to execute the task. Such a general purpose fault tolerant manipulator can fault tolerantly execute any task of which the task space lies inside the fault tolerant workspace of the manipulator. This approach to fault tolerance is further explored in Section 3.4.

In a second approach, the goal is to design a *task specific fault tolerant manipulator.* In this approach, we assume that the Cartesian path to be followed is known a priori and that the corresponding set of possible joint trajectories can be limited by an appropriate choice of a redundancy resolution algorithm. In Section 3.5 and in more detail in the next chapter, we show how these additional assumptions allow us to design a fault tolerant manipulator with fewer DOFs than a general purpose fault tolerant manipulator.

For both approaches, we will answer the question: How many degrees-of-redundancy are necessary and sufficient for fault tolerance? However, before we address this design problem, we want to make sure that fault tolerance is indeed a good mechanism for achieving high reliability. In the next section, we investigate under which conditions fault tolerance improves the overall system reliability the most.

3.3 Fault Tolerance and Reliability

The basic idea presented in this chapter is to use a manipulator's redundant DOFs to compensate for a possible failure of one of the joints. The underlying assumption is that a manipulator that can sustain a joint failure is more reliable than one that cannot. The question is: "Does fault tolerance always result in an increase in reliability?" The answer is given by reliability theory (Johnson 1989).

The reliability, $R(t)$, of a component or a system is the conditional probability that the component operates correctly throughout the interval $[t_0, t]$, given that it was operating correctly at the time . For non-fault-tolerant serial link manipulators, the system fails when any single subsystem—a DOF or joint module for modular manipulators—fails. The system reliability can then be computed as the product of the module reliabilities, $R_i(t)$:

$$R_s(t) = R_1(t)R_2(t)\ldots R_n(t). \tag{3-1}$$

Or, in the case that every module is equally reliable with reliability $R_{mod}(t)$:

$$R_s(t) = R^n_{mod}(t). \tag{3-2}$$

If there are n modules and only m of those are required for the system to function properly—the system can tolerate $(n-m)$ module failures—then the system reliability is the sum of the reliabilities of all systems with $(n-m)$ or fewer faults. Since there are $\binom{n}{i}$ different systems with i faults, the system reliability of a fault tolerant system with equal module reliabilities can be written concisely as

$$R_s(t) = \sum_{i=0}^{n-m} \binom{n}{i} R^{n-i}_{mod}(t)(1-R_{mod}(t))^i. \tag{3-3}$$

We can apply this formula to the example of an 8-DOF fault tolerant manipulator, which needs only seven DOFs to function properly. The system reliability of the fault tolerant system is:

$$^fR_s(t) = R^8_{mod}(t) + 8R^7_{mod}(t)(1-R_{mod}(t)) = R^7_{mod}(t)(8-7R_{mod}(t)), \tag{3-4}$$

compared to $R_s(t) = R^6_{mod}(t)$ for an equivalent 6-DOF non-fault-tolerant system. Both reliabilities are plotted as a function of the module reliability in Figure 3-1, while Figure 3-2 shows the *relative* system reliability $^fR_s/R_s$:

$$\frac{^fR_s}{R_s} = R_{mod}(8-7R_{mod}), \tag{3-5}$$

which equals 1 for $R_{mod} = 1$ and $R_{mod} = 1/7$. These graphs should be interpreted as fol-

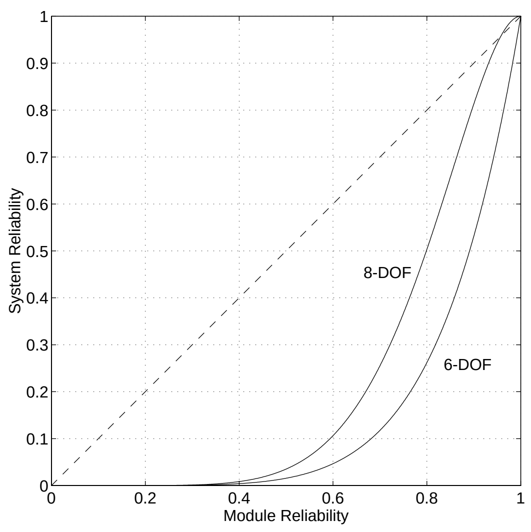


Figure 3-1. System reliability of an 8-DOF fault tolerant manipulator and a 6-DOF non-fault-tolerant manipulator.

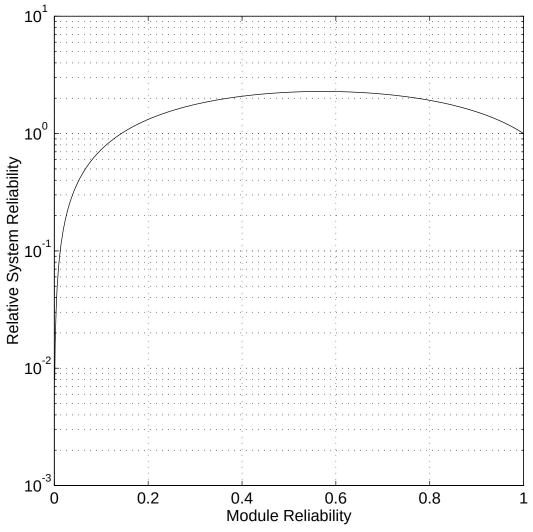


Figure 3-2. Relative system reliability of an 8-DOF fault tolerant manipulator versus a 6-DOF non-fault-tolerant manipulator.

lows:

- when $R_{mod}(t) = 0$, then ${}^fR_s = R_s = 0$. The system reliability is zero in both cases, i.e., both systems are guaranteed to fail.
- when $R_{mod}(t) = 1$, then ${}^fR_s = R_s = 1$. That is, both systems are 100% reliable.
- when $\frac{1}{7} < R_{mod}(t) < 1$, then ${}^fR_s > R_s$, meaning that the fault tolerant system is more reliable than the non-fault-tolerant one.
- when $R_{mod}(t) < \frac{1}{7}$, then ${}^fR_s < R_s$. The modules are so unreliable that the added complexity of the fault tolerant system reduces the overall performance.

Preferably, one would like to operate a system at a reliability close to one, for which the fault tolerant system is the more reliable. To compare both alternatives for modules with a high reliability, it is more instructive to rewrite Equation (3-4) as an expression for the system's *unreliability*, $Q(t) = 1 - R(t)$:

$$ {}^fQ_s(t) \approx 28Q^2_{mod}(t) - 112Q^3_{mod}(t), \tag{3-6}$$

when $Q_{mod} \to 0$. The unreliability for the non-fault-tolerant system is

$$Q_s(t) \approx 6Q_{mod}(t) - 15Q_{mod}^2(t). \qquad (3\text{-}7)$$

In general, the unreliability of a k-fault tolerant system—one that can sustain k faults—is of the order $O(Q_{mod}^{k+1})$. This means that the reliability of a fault tolerant system increases more significantly when the reliability of the individual modules is high. Best results are thus obtained when fault tolerance is combined with high component reliability, or fault intolerance.

3.4 General Purpose Fault Tolerant Manipulators(1)

In this section, we discuss the kinematic design of a general purpose fault tolerant manipulator without joint limits. Just like for non-fault-tolerant manipulators, the kinematic capabilities are mainly characterized by the shape and size of the workspace or rather the fault tolerant workspace (FTWS) in this case. We identify several properties of general purpose fault tolerant manipulators and their workspaces, and propose an 8-DOF design template.

To set the stage for our development, we define the following concepts relating to general purpose fault tolerant manipulators:

- **General Purpose Fault Tolerant Manipulator**: A manipulator that will still be able to meet the task specifications, even if any one or more of its joints fail and are frozen at any arbitrary joint angles.
- ***k*-Reduced Order Derivative (*k*-ROD)**: When k joints of an n-DOF manipulator fail, the effective number of joints is $(n-k)$. The resulting faulty manipulator is called a k-reduced order derivative.
- **Order of Fault Tolerance**: A manipulator is fault tolerant of the k-th order, if and only if all possible k-reduced order derivatives can still perform the speci-

(1) This section is based on Paredis and Khosla (1994).

fied task. We call the manipulator k-fault tolerant.

- **Fault Tolerant Workspace (FTWS)**: The fault tolerant workspace of a k-fault tolerant manipulator is the set of points reachable by all possible k-reduced order derivatives.

Notice that our definition of a general purpose fault tolerant manipulator reflects the assumption that the redundancy resolution algorithm is not known a priori: a joint failure can occur at an arbitrary angle.

In the remainder of this section, if no specific task is mentioned, it is assumed that the task is to reach a nonzero volume of points. That is, the task space is an m-dimensional manifold in the m-dimensional output space of the manipulator. A manipulator with a FTWS of dimension less than m is considered not to be fault tolerant.

3.4.1 Properties of General Purpose Fault Tolerant Manipulators

3.4.1.1 Existence

A general purpose manipulator has six DOFs which allow it to position its end effector in an arbitrary position and orientation anywhere in its workspace. An obvious way to make this manipulator fault tolerant is to design every joint with a redundant actuator. If one of the actuators of the resulting $2n$-DOF fault tolerant manipulator were to fail, the redundant actuator could take over and the manipulator would still be functional. Similarly, a k-fault tolerant manipulator can be constructed by duplicating every DOF k times, resulting in a $(k+1)n$-DOF manipulator. This argument illustrates that $(k+1)n$ DOFs are sufficient for k-th order fault tolerance. In the remainder of this section, we determine the number of DOFs *necessary* to achieve general purpose fault tolerance.

3.4.1.2 Boundary of the Fault Tolerant Workspace

In this section, we show that a boundary point of the FTWS is a critical value.(2) Consider a k-fault tolerant planar manipulator, $\mathtt{M}$. A boundary point, $\boldsymbol{p}_b$, of the FTWS has to be an

(2) A critical value is an end-effector position that can be reached in a singular configuration, i.e., that is the image of a critical point (Burdick 1988).

element of the boundary of the workspace of at least one ROD, $\mathtt{M}*$, obtained by freezing k joints of $\mathtt{M}$. Indeed, if $\boldsymbol{p}_b$ were an interior point of the workspaces of all RODs, then it would by definition be an interior point of the FTWS and not a boundary point. The Jacobian of $\mathtt{M}*$, J_{M*}, can be obtained from the Jacobian of $\mathtt{M}$, J_M, by deleting the columns corresponding to the frozen DOFs. Because $\boldsymbol{p}_b$ is a boundary point of the workspace of $\mathtt{M}*$, the Jacobian of $\mathtt{M}*$ at $\boldsymbol{p}_b$ is singular. We prove now that J_M is singular too. Assume that J_M were non-singular, then at least one of the columns corresponding to a frozen DOF would be outside the column space of the singular matrix, J_{M*}. Physically this means that a small change in the angle of that frozen DOF would cause the end effector of $\mathtt{M}$ to move in a direction with a component perpendicular to the boundary of the workspace of the ROD, $\mathtt{M}*$, as illustrated in Figure 3-3. The ROD with this new frozen angle would be unable to reach the point, $\boldsymbol{p}_b$. As a result, $\boldsymbol{p}_b$ would be outside the FTWS, contradicting the fact that $\boldsymbol{p}_b$ is a boundary point of the FTWS. Thus, J_M is singular and $\boldsymbol{p}_b$ is a critical value.

Consequently, the FTWS is bounded by critical value manifolds. For planar positional manipulators, the critical value manifolds are concentric circles, and the FTWS is an annulus with inner radius $R_{\min}^{\text{FTWS}}$ and outer radius $R_{\max}^{\text{FTWS}}$.

3.4.1.3 Required Degree of Redundancy

In Section 3.4.1.1, it is shown that, in general, kn redundant DOFs—i.e. $(k+1)n$ DOFs in total—are sufficient to achieve k-th order fault tolerance. For planar positional manipula-

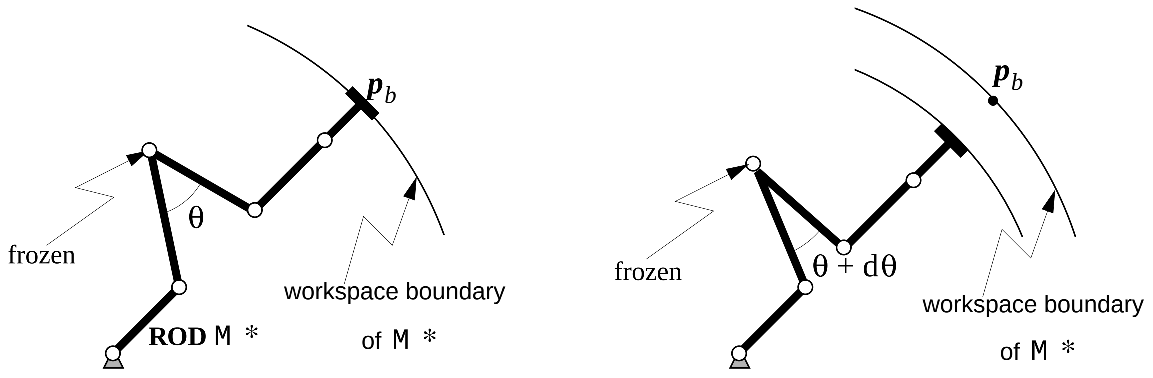


Figure 3-3. A ROD unable to reach a point outside the FTWS.

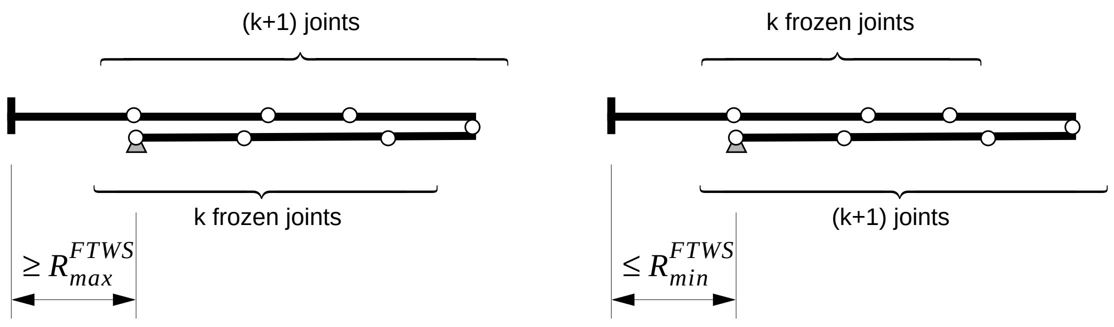


Figure 3-4. An upper bound for $R_{\max}^{\text{FTWS}}$ and a lower bound for $R_{\min}^{\text{FTWS}}$

tors, however, we prove that $2k$ redundant DOFs are necessary and sufficient for k-th order fault tolerance.

Necessary: The proof shows that $(2k+1)$ DOFs (or $2k-1$ redundant DOFs) are insufficient, by finding a lower bound for $R_{\min}^{\text{FTWS}}$ and an upper bound for $R_{\max}^{\text{FTWS}}$. First consider the ROD obtained by freezing the first k joints at 0 radians, as illustrated in Figure 3-4. The maximum reach in the opposite direction is an upper bound for $R_{\max}^{\text{FTWS}}$:

$$R_{\max}^{\text{FTWS}} \leq -\sum_{i=1}^{k} l_i + l_{k+1} + \sum_{i=k+2}^{2k+1} l_i, \tag{3-8}$$

where l_i is the length of the i-th link. In order for $R_{\max}^{\text{FTWS}}$ to be positive, we must have that:

$$\sum_{i=1}^{k} l_i \leq l_{k+1} + \sum_{i=k+2}^{2k+1} l_i. \tag{3-9}$$

Making this assumption, we find that $R_{\min}^{\text{FTWS}}$ is bounded below by the inner radius of the workspace of the ROD obtained by freezing the k last joints at 0 radians, as illustrated in Figure 3-4:

$$R_{\min}^{\text{FTWS}} \geq \sum_{i=k+2}^{2k+1} l_i + l_{k+1} - \sum_{i=1}^{k} l_i. \tag{3-10}$$

From Equation (3-8) and Equation (3-10), it follows that at best

$$R_{\max}^{\text{FTWS}} = R_{\min}^{\text{FTWS}}, \tag{3-11}$$

resulting in a one-dimensional FTWS. Therefore, a $(2k+1)$-DOF planar manipulator cannot be k-th order fault tolerant.

Sufficient: The proof shows that there exists a $(2k+2)$-DOF manipulator template that is k-fault tolerant. Consider a manipulator with $(2k+2)$ links of length l. Because all the links have the same length, it is possible to compensate for a fault in a DOF by choosing a neighboring DOF to be at π radians; that is, folded back onto the failing DOF. Even when consecutive DOFs fail, this trace-back-mechanism can be used to compensate for failures. The result is that, by sacrificing one DOF to compensate for every fault, the $(2k+2)$-DOF manipulator with k faults is equivalent to a faultless 2-DOF manipulator. The FTWS of the $(2k+2)$ DOF manipulator is then the workspace of the equivalent 2-DOF manipulator, that is,

$$FTWS = \{(x, y) | \sqrt{x^2 + y^2} \le 2l\}. \tag{3-12}$$

3.4.1.4 Including Orientation

Thus far, we have only considered planar positional manipulators. The results can be easily extended to the case in which orientation is considered also, by converting the orientational problem into an equivalent positional problem:

An n-DOF manipulator, $\mathtt{M}$, is k-fault tolerant with respect to a set of points, $W = \{(x_i, y_i, \varphi_i)\}$, if and only if:

1. The positional manipulator, $\mathtt{M}'$, obtained from $\mathtt{M}$ by deleting its last link, l_n, is k-fault tolerant with respect to the set of points $W' = \{(x_i - l_n\cos\varphi_i, y_i - l_n\sin\varphi_i)\}$,
2. $\mathtt{M}'$ is $(k-1)$-fault tolerant while reaching the points in W' in any direction.

Necessary: The positional manipulator, $\mathtt{M}'$, needs at least $(2k+2)$ DOFs to be k-th order fault tolerant with respect to W'; therefore, the manipulator $\mathtt{M}$ needs at least $(2k+3)$ DOFs.

Sufficient: Again, we show that there exists a $(2k+3)$-DOF manipulator template that is k-fault tolerant. Consider a template of which the first $(2k+2)$ links have length l and the last link has length zero; it is the template described in the previous section with a zero-length link added at the end. For this template, one can again use the trace-back-mechanism to show that it is equivalent to a faultless 3-DOF manipulator with link lengths l, l, and 0. The FTWS is thus:

$$FTWS = \left\{ (x, y, \varphi) \mid \sqrt{x^2 + y^2} \le 2l \text{ and } \varphi \in [0, 2\pi) \right\} \tag{3-13}$$

This result for planar manipulators with orientation and the result obtained in Section 3.4.1.3 can be summarized in the following theorem:

Theorem:

> *For planar manipulators without joint limits, $2k$ degrees-of-redundancy are necessary and sufficient for k-th order general purpose fault tolerance.*

3.4.2 Spatial Fault Tolerant Manipulators

For planar fault tolerant manipulators, we were able to prove that $2k$ is the required degree of redundancy. The proof was based on geometric workspace analysis. For spatial manipulators, however, the geometric analysis becomes too complex. Therefore, we will demonstrate some properties of spatial general purpose fault tolerant manipulators using two examples.

As a first example, consider a 5-DOF spatial positional manipulator. Its Denavit-Hartenberg (D-H) parameters are listed in Table 3-1. This manipulator is first order fault tolerant, and because of its simple kinematic structure, an analytic expression for the boundary of the FTWS can be derived. The FTWS is symmetric with respect to the first axis. A cross-section (the X-Z plane), as shown in Figure 3-5, can be described by two segments of a circle with radius 2 and center at $(x = 1, z = 0)$, and a straight line from $(x = 2, z = \sqrt{3})$ to

DOF i	d_i	a_i	α_i
1	0	1	90°
2	a	1	0°
3	-a	1	90°
4	b	1	0°
5	-b	1	—

Table 3-1. D-H parameters of a 5-DOF first order fault tolerant spatial manipulator, without orientation.

DOF i	d_i	a_i	α_i
1	0	1	90°
2	a	1	0°
3	-a	1	90°
4	b	1	0°
5	-b	0	90°
6	1	0	90°
7	0	0	90°
8	0	0	—

Table 3-2. D-H parameters of an 8-DOF first order fault tolerant spatial manipulator.

$(x = 2, z = -\sqrt{3})$. An important property of this FTWS is that it does not have any holes or a central void, so that the FTWS of the same manipulator scaled by any factor $\lambda > 1$ contains the original FTWS. As a result, this fault tolerant manipulator can be used as a *design template*. Any task space can be enclosed in the FTWS of a scaled version of the design template.

In Section 3.4.1.2, it is shown that the boundary of the FTWS of a planar manipulator coincides with its critical value manifolds. Figure 3-5 demonstrates that this property also holds for the 5-DOF spatial manipulator considered in this example. The critical value manifolds are computed using the algorithm described in Burdick (1992) and are depicted in a solid line. The bold part of the critical value manifolds is the boundary of the FTWS. The property that the FTWS is bounded by critical value manifolds can be effectively used for the determination of the FTWS. Testing whether a point is an element of the FTWS is a complicated procedure. One has to verify whether that point is reachable for all possible RODs, i.e., for all manipulator structures resulting from a joint failure of every possible joint at every possible joint angle. To find a good approximation of the FTWS, one would have to execute this test for a large number of points. This would be prohibitively slow. However, to improve the efficiency, one can compute the critical value manifolds of the manipulator first. These manifolds partition the Cartesian output space of the manipulator in sectors that are

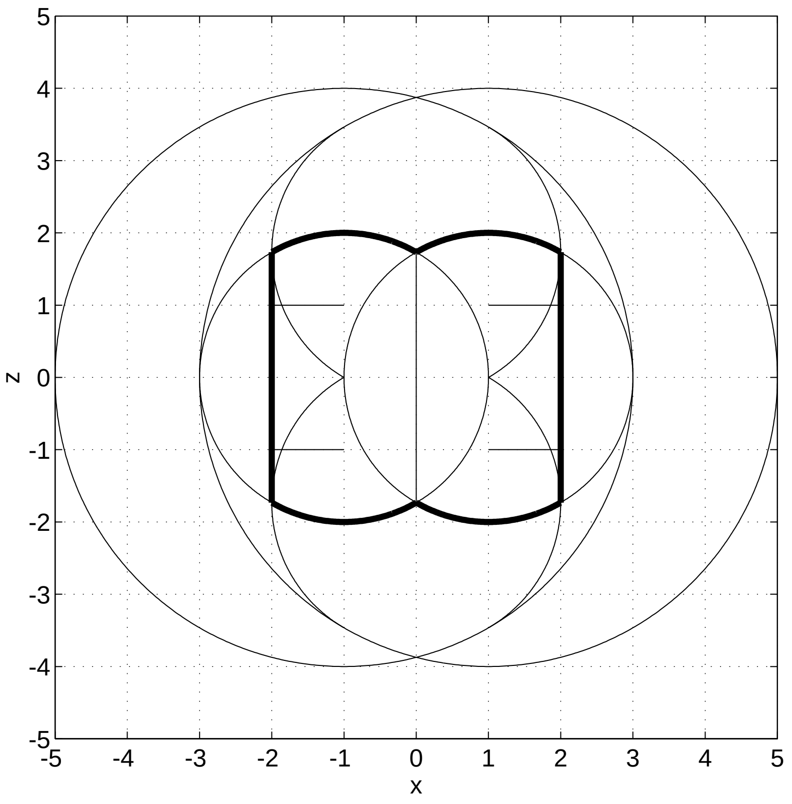


Figure 3-5. A cross-section of the boundary of the FTWS of a 5-DOF spatial manipulator (bold) as part of its critical value manifolds.

either entirely inside the FTWS or entirely outside the FTWS. Thus, by checking whether *one point* of a sector is an element of the FTWS, one can check whether the *whole sector* is in the FTWS. The number of FTWS-membership tests is so reduced to the number of sectors in the partition of the output space.

As a second example of a spatial general purpose fault tolerant manipulator consider the 8-DOF manipulator depicted in Figure 3-6 with D-H parameters listed in Table 3-2. It is the same manipulator as in the previous example, with a zero-length 3-roll-wrist added at the end. Using a Monte Carlo method, it has been determined that this manipulator is first order fault tolerant while reaching all the points, in the FTWS of example one, *in any direction*. This property can be demonstrated with the following arguments. When one of the first five DOFs fails, the manipulator can still reach any position in the FTWS (because the 5-DOF positional manipulator is FT) and can take any orientation at this position using the intact 3-

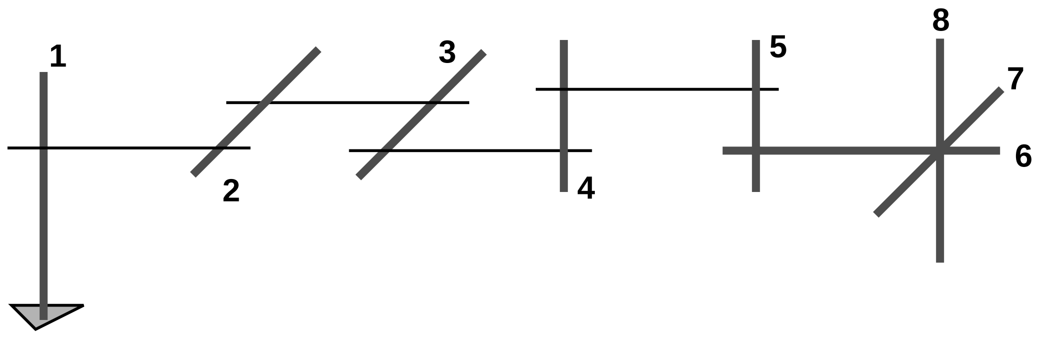


Figure 3-6. The kinematic structure of a 8-DOF fault tolerant manipulator template.

roll-wrist. When one of the DOFs in the wrist fails, we are left with a 7-DOF manipulator which has enough orientational capabilities to reach any point in the FTWS in any orientation. Consequently, one could call this the *dextrous* FTWS. Since there are again no holes or voids in the FTWS, this manipulator can also be used as a design template.

Finally, one should notice that both examples have only two redundant DOFs, which indicates that two degrees-of-redundancy are also sufficient for 1-fault tolerance of spatial manipulators. Whether the theorem in Section 3.4.1.4 also holds for higher orders of fault tolerance of spatial manipulators requires further research.

3.5 Task Specific Fault Tolerant Manipulators

In the previous section, we considered the design of fault tolerant manipulators for general use. We proved that two redundant DOFs are necessary and sufficient for first order fault tolerance. However, as we will show in this section, a simpler kinematic structure is often sufficient when a specific task is considered.

The disadvantage of this approach is that a task specific fault tolerant manipulator is only suited for a very limited set of similar end-effector paths. For each new Cartesian path, one has to plan a fault tolerant joint space trajectory to be followed by the manipulator *before* failure in order to guarantee fault tolerance. Moreover, unlike general purpose fault tolerance, task specific fault tolerance might require a different manipulator structure for every task.

However, these disadvantages can be overcome, by integrating task specific fault tolerance into our framework for rapidly deployable manipulators. To achieve fault tolerance, one can then plan an appropriate fault tolerant joint space trajectory with the algorithm developed in Chapter 4, and one can easily choose a manipulator custom-tailored for the specific task using the Task Based Design software that is described in Chapter 5. Once a manipulator has been chosen, one can assemble the RMMS modules and execute the task fault tolerantly with the resulting manipulator.

In this section, we prove that task specific fault tolerance can indeed be achieved with fewer degrees-of-redundancy than required for a general purpose fault tolerant manipulator.

Consider the task of reaching all the points in an ε-neighborhood, $B(p, \varepsilon)$, of the point $p \in \Re^m$. Suppose that p can be reached by an n-DOF manipulator in a posture, $\theta \in T^n$. If the posture, θ, is non-singular, then there exists an $\varepsilon > 0$, such that the manipulator can reach any point in $B(p, \varepsilon)$ when all its joints are functioning. However, for k-fault-tolerance, any point in $B(p, \varepsilon)$ needs to be reachable even when k of the joints of the manipulator are frozen. This is possible if and only if the Jacobians of all k-RODs in the posture θ are non-singular, i.e., have at least rank m. We call such a posture, θ, *locally fault tolerant.*

The Jacobian of a k-ROD, J_{ROD}, can be obtained by deleting the columns of the fault-free Jacobian that correspond to the frozen DOFs; the dimensions of J_{ROD} are $m \times (n-k)$. A necessary condition for J_{ROD} to be of rank m is that n has to be larger than or equal to $(m+k)$. Indeed, the manipulator needs to have at least m functional DOFs, even after a failure of k of them. That means that k degrees-of-redundancy are necessary for local fault tolerance.

Are k degrees-of-redundancy also *sufficient* for local fault tolerance? Consider a manipulator with $n = m + k$ DOFs; the Jacobian of a k-ROD, J_{ROD}, is then a square $m \times m$ matrix. A posture, $\theta \in \Re^n$, is locally fault tolerant if the Jacobians of all m RODs are full rank. When the rank of any J_{ROD} is less than m, the robot is in an *internal singularity.* The differ-

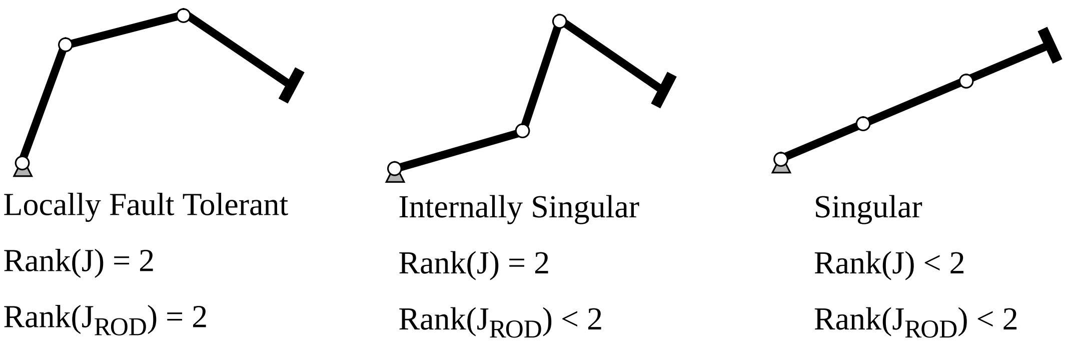


Figure 3-7. Examples of locally fault tolerant, internally singular, and singular postures of a 3-DOF planar manipulator.

ence between singular, locally fault tolerant and internally singular postures is illustrated in Figure 3-7. The locus of internal singularities is a set of $(m + k - 1)$-dimensional surfaces in T^n; or $(n - 1)$-dimensional surfaces, when $n = m + k$. Thus, nearly all postures of a manipulator with k degrees-of-redundancy are locally k-fault tolerant, so that k degrees-of-redundancy are indeed sufficient. This can be summarized in the following theorem.

Theorem:

k degrees-of-redundancy are necessary and sufficient for k-th order local fault tolerance.

The fact that a posture is locally k-fault tolerant guarantees that the manipulator can reach every point in a neighborhood of the end effector position, even after failure of k DOFs. However, this neighborhood can be small, for instance, when the posture is close to an internal singularity.

To extend this result to larger trajectories, we have to formulate a *global* fault tolerance condition. This is the topic of the next chapter, in which we develop a global fault tolerant trajectory planning algorithm.

3.6 Summary

In this chapter, we have shown that making a manipulator fault tolerant by adding redundant DOFs is an effective way to increase the reliability of a manipulator. However, not every redundant manipulator is fault tolerant. Thus, an important problem for the design of fault tolerant manipulators is: How many DOFs are necessary and sufficient for fault tolerance and how should these DOFs be distributed along the length of the manipulator? We have shown that, depending on the assumptions that are made about the task, the answer to this question varies.

For general purpose fault tolerant manipulators without joint limits, two degrees-of-redundancy are necessary and sufficient to sustain one fault. This conclusion was illustrated with two spatial general purpose fault tolerant manipulator designs: a 5-DOF positional manipulator and an 8-DOF positional and orientational manipulator. Both manipulators have a fault tolerant workspace without any holes or voids so that one can scale the designs to fit any task.

For task specific fault tolerant manipulators, only one degree-of-redundancy is necessary and sufficient for 1-fault tolerance. However, one might have to use a different manipulator and recompute a fault tolerant joint space trajectory, for every task. This drawback can be partially overcome by using a modular manipulator system that can be quickly reconfigured to suit a particular task. Computing a fault tolerant joint space trajectory, and determining a task specific fault tolerant manipulator configuration are the two problem addressed in the next two chapters.

Chapter 4

Global Fault Tolerant Trajectory Planning

4.1 Introduction

As we have shown in the previous chapter, whether a task can be completed after a joint failure depends not only on the structure of the manipulator, but, for task specific fault tolerant manipulators, also on the specific joint angle at which the failure occurs. In general, failures at a fully extended or folded back position of a joint are most detrimental to the remaining capabilities of the manipulator.

The basic idea that we exploit in this chapter is to achieve fault tolerance by avoiding unfavorable joint positions before failure. This idea was first proposed by Lewis and Maciejewski (1994a) where the null-space component of a redundant manipulator was used to locally minimize the kinematic fault tolerance measure (kfm). The authors showed that, for a particular test path, a manipulator with kfm minimization is more likely to be fault tolerant than a manipulator with traditional pseudoinverse control. However, due to the local nature of the kfm, fault tolerance could not be guaranteed on a global scale (Lewis and Maciejewski 1994b).

In this chapter, we present a trajectory planning algorithm that *does* guarantee fault tolerance on a global scale, while, in addition, avoiding any violations of secondary kinematic requirements such as joint limits and obstacles. To achieve this global result, we have to consider the topology of the self-motion manifolds, as has been previously suggested by Lück and Lee (1994).

4.2 Definitions

In this section, we introduce several concepts that are essential for the development of the algorithm presented in the next section. We start by giving an exact definition of the problem.

Definition 1: Fault Tolerant Trajectory Planning Problem

Given: – a manipulator defined by its geometry, joint limits, and redundancy resolution algorithm.

– a task description consisting of a Cartesian path, $p(t) \in \Re^m$, and the geometry of the obstacles.

Find: – a fault tolerant trajectory in joint space(1): $\theta(t) \in T^n$.

A fault tolerant trajectory is defined as follows:

Definition 2: Fault Tolerant Trajectory

A trajectory, $\theta(t) \in T^n$, is 1-fault tolerant with respect to the task of following the Cartesian path $p(t) \in \Re^m$, if for every DOF, $j = 1 \ldots n$, and for every instant, t^*, there exists an alternate trajectory, $\theta(t, j, t^*)$, for which:

1) $\theta(t, j, t^*)$ maps onto $p(t)$ under the forward kinematics
2) $\theta(t^*) = \theta(t^*, j, t^*)$
3) $\theta_j(t, j, t^*) = \theta_j(t^*), \quad \forall t > t^*$

(1) We assume that the manipulator has only revolute joints. The joint space is therefore the n-dimensional torus T^n.

4) $\theta(t, j, t^*)$ does not violate any secondary task requirements such as joint limits, obstacles collision, or self collision.

This definition corresponds to our scenario for fault tolerance as described in the previous chapter. Before any failures occur, the manipulator follows the fault tolerant joint trajectory $\theta(t)$. After a failure in joint j at time t^*, joint j is immobilized and the joint trajectory is adapted to keep tracking the path $p(t)$. The new trajectory, $\theta(t, j, t^*)$, is equal to $\theta(t)$ at the instant of failure and maintains a constant joint angle, $\theta_j(t^*)$, for the frozen joint j after the failure.

There are an infinite number of alternate trajectories, $\theta(t, j, t^*)$, one for every possible combination of a failing DOF and an instant of failure. This poses practical problems. While one can explicitly store a discretized version of $\theta(t)$, explicit storage of all $\theta(t, j, t^*)$ is impossible. Therefore, we assume that the alternate trajectories are stored *implicitly* in a redundancy resolution algorithm that computes $\theta(t, j, t^*)$ at run time once a failure has taken place. We also assume that this redundancy resolution algorithm unambiguously determines $\theta(t, j, t^*)$, given t^*, $\theta(t^*)$, and j; that is, we only consider redundancy resolution algorithms that determine the next joint vector based on the current joint vector, and not on past joint vectors. This assumption is satisfied for commonly used Jacobian-based algorithms of the form:

$$\dot{\theta} = J^{\dagger}\dot{x} + (I - J^{\dagger}J)\dot{\Phi}. \tag{4-1}$$

Readers unfamiliar with this kind of redundancy resolution algorithms are referred to (Nenchev 1989) for a detailed overview; a detailed example is also given in Section 4.6 of this chapter.

Because the choice of the redundancy resolution algorithm fully determines the alternate trajectories, it also influences the solution of the trajectory planning problem. In this thesis, we assume the redundancy resolution algorithm to be a given of the problem, i.e., a part of the manipulator definition. Consequently, for a failure of joint j at posture $\theta(t^*)$, there exists a unique alternate trajectory $\theta(t, j, t^*)$.

In the fourth point of the definition of a fault tolerant trajectory, we refer to “secondary task requirements.” The primary requirement is to follow the path $p(t)$. In the problem definition, we included joint limits, obstacle collision, and self collision as secondary requirements, but one can include any other kinematic requirement that only depends on the current posture. For instance, all the dexterity measures enumerated by Kim and Khosla (1991) depend only on the current joint position and could thus be included as secondary requirements. We call the set of postures that satisfy all the secondary task requirements the set $\mathtt{S}$.

At each instant, t, the manipulator postures $\theta(t)$ and $\theta(t, j, t^*)$ map onto the path $p(t)$ under the forward kinematics of the manipulator, $p = f(\theta)$. Consequently, we say that these postures are elements of the preimage of p.

Definition 3: Preimage of a point p

The preimage of a point, p, is the set $\Sigma(p) = \{\theta \in T^n | f(\theta) = p\}$, where f is the forward kinematic mapping of the manipulator.

This preimage is a set of r-dimensional manifolds,(2) where $r = n - m$ is the degree-of-redundancy of the manipulator.

Assume now that joint j fails. We call the resulting manipulator, with joint j immobilized, a reduced order derivative (ROD).

Definition 4: k-Reduced Order Derivative

A manipulator with $(n - k)$ DOFs, obtained by immobilizing k of the joints of an n-DOF manipulator, is called a k-reduced order derivative.

Whether this ROD is able to complete the task, as is required for fault tolerance, depends on the posture $\theta(t^*) \in \Sigma(p(t^*))$ at which the failure occurred. For certain $\theta(t^*)$, the path $p(t)$ might pass outside the workspace of the ROD, the redundancy resolution algorithm might get stuck at a singularity, or the alternate trajectory $\theta(t, j, t^*)$ might violate the joint limits or cause a collision with an obstacle or another part of the manipulator. In all of these cases, the task cannot be completed. We call the corresponding posture $\theta(t^*)$ intolerant to a failure

(2) An exception is the preimage of a critical value, which is not a manifold but a bouquet of tori (Burdick 1988).

of DOF j.

Definition 5: Posture Tolerant to a Failure of DOF j

> A posture $\theta \in \Sigma(p(t^*))$ is tolerant to a failure of DOF j if and only if the alternate trajectory $\theta(t, j, t^*)$, as determined by the redundancy resolution algorithm, satisfies all the task requirements.

We call the set of postures $\theta \in \Sigma(p(t^*))$ that are tolerant to a failure of DOF j the set $\mathsf{F}_j^{t^*} \subset \Sigma(p(t^*))$.

Based on the definition of a fault tolerant trajectory, we conclude that a posture is an acceptable posture for a fault tolerant trajectory if it is tolerant to failures of each of the DOFs. The set of acceptable postures $\theta \in \Sigma(p(t^*))$ is given by the equation:

$$\mathsf{A}^{t^*} = \bigcap_{j=1}^{n} \mathsf{F}_j^{t^*}. \tag{4-2}$$

4.3 The Algorithm

The algorithm to determine a fault tolerant trajectory consists of two parts. In the first part, we determine for each instant, t^*, the set of acceptable postures, A^{t^*}, as defined in the previous section. In the second part, we create a connectivity graph for the acceptable postures and search this graph to determine a fault tolerant trajectory.

A key observation for the development of our algorithm is that whether a posture, $\theta(t^*)$, is acceptable depends only on the future course of the path $p(t)$; it is independent of $p(t)$ for $t < t^*$. For example, if a failure occurs at the last point, p_{last}, of the path, the task can always be completed, regardless of which course the path followed previously and regardless of the posture in which the manipulator reaches this last point. We conclude that

$$\mathsf{A}^{t_{last}} = \mathsf{F}_1^{t_{last}} = \dots = \mathsf{F}_n^{t_{last}} = \Sigma(p_{last}) \cap \mathsf{S}. \tag{4-3}$$

This conclusion forms the basis for the algorithm's initialization.

The main iteration of our algorithm is based on a second important observation. Consider a candidate fault tolerant trajectory, $\theta_1(t)$. At time t_1, joint j fails and the alternate trajectory $\theta_1(t, j, t_1)$ is followed, as is illustrated in Figure 4-1. Consider also a second candidate fault tolerant trajectory, $\theta_2(t)$, which intersects with $\theta(t, j, t_1)$ at time t_2, so that $\theta_1(t_2, j, t_1) = \theta_2(t_2)$. If a failure of joint j were to occur at time t_2, the alternate trajectory $\theta_2(t, j, t_2)$ would be followed. Because the joint velocity, $\dot{\theta}$, in the redundancy resolution algorithm, depends only on j, $p(t)$, and the current joint vector, the two alternate trajectories $\theta_1(t, j, t_1)$ and $\theta_2(t, j, t_2)$ are equal to each other for $t > t_2$. A Corollary of this observation is that a posture is tolerant to a fault in joint j if and only if all the postures along the corresponding alternate trajectory are also tolerant to faults in joint j:

$$\begin{gathered}\theta(t_1) \in \mathsf{F}_j^{t_1} \quad \Leftrightarrow \quad \theta(t^*, j, t_1) \in \mathsf{F}_j^{t^*}, \\ \forall t^* > t_1 \text{ and } \theta(t_1) \in \mathsf{S}\end{gathered} \tag{4-4}$$

which is equivalent to the expression:

$$\begin{gathered}\theta(t_1) \in \mathsf{F}_j^{t_1} \quad \Leftrightarrow \quad \theta(t^*, j, t_1) \in \mathsf{F}_j^{t^*}, \\ \forall t^* \in (t_1, t_1 + \Delta t] \text{ and } \theta(t_1) \in \mathsf{S}, \quad \Delta t > 0\end{gathered} \tag{4-5}$$

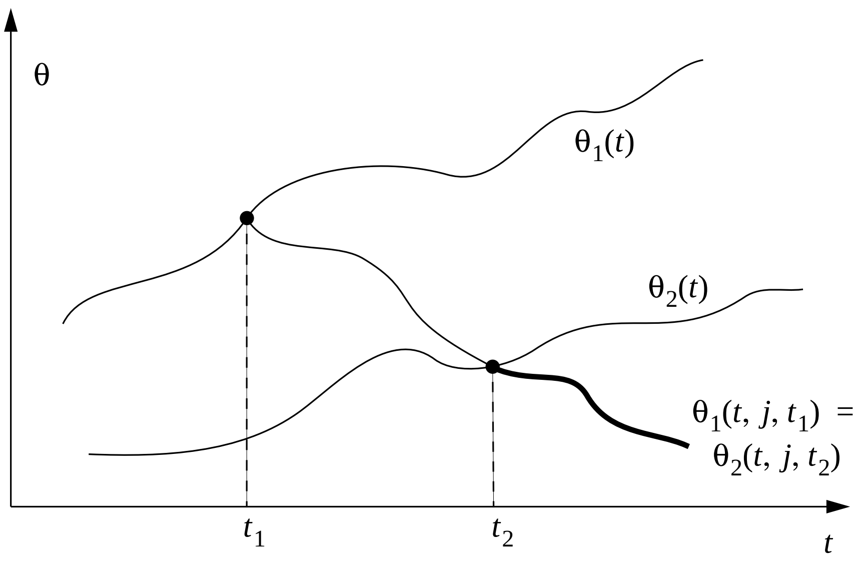


Figure 4-1. Two possible failures resulting in the same alternate trajectory.

Equation (4-5) means that we can determine whether a posture, $\theta(t_1)$, is tolerant to a fault of DOF j by tracing the alternate trajectory up to $t_1 + \Delta t$ rather than up to t_{last}. This property is used in the main iteration of the first part of our algorithm.

Part 1: determination of the acceptable postures

```
• discretize the path: $p_k = p(t_k) = p(k\Delta t)$
• compute the preimage of the last point: $\Sigma(p_{last})$
• compute the acceptable postures for the last point:
                                   $A^{t_{last}} = F_j^{t_{last}} = \Sigma(p_{last}) \cap S$
• for k=last-1 to first do
    • compute the preimage: $\Sigma(p_k)$
    • for j=1 to n do
        • for every posture $\theta \in \Sigma(p_k)$ do
            • compute $\theta(t_{k+1}, j, t_k)$ using the redundancy
                                              resolution algorithm
            • if $\theta \in S$ and $\theta(t_{k+1}, j, t_k) \in F_j^{t_{k+1}}$ then $\theta \in F_j^{t_k}$
        • next θ
    • next j
    • compute the set of acceptable postures: $A^{t_k} = \bigcap_{j=1}^{n} F_j^{t_k}$
• next k
```

Once the sets of acceptable postures have been computed, a fault tolerant trajectory is chosen in the second part of our algorithm. A fault tolerant trajectory consists of a sequence, $\{\theta(t_k)\}$, of acceptable postures—one posture $\theta(t_k) \in A^{t_k}$ for each instant t_k. However, one cannot pick the postures $\theta(t_k)$ at random from A^{t_k}. For a valid fault tolerant sequence, there should exist a continuous trajectory of acceptable postures connecting each pair of postures $\theta(t_k)$ and $\theta(t_{k+1})$. Moreover, the sequence $\{\theta(t_k)\}$ should preferably vary smoothly and stay away from the boundaries of A^{t_k}. To simplify the search for such a sequence, we first group the postures of A^{t_k} that are connected to each other.

In general, a set, A^{t_k}, may consist of several disjoint regions, $\mathsf{R}_i^{t_k}$, of acceptable postures:

$$\mathsf{A}^{t_k} = \bigcup \mathsf{R}_i^{t_k} \quad \text{and} \quad \mathsf{R}_i^{t_k} \cap \mathsf{R}_j^{t_k} = \varnothing, \ \forall i \neq j . \tag{4-6}$$

The postures in each region $\mathsf{R}_i^{t_k}$ are connected to each other in the sense that there exists a continuous trajectory of acceptable postures, $\theta \in \mathsf{A}^{t_k}$, connecting any two postures in $\mathsf{R}_i^{t_k}$. On the other hand, by definition, there does not exist any combination of two postures, one from $\mathsf{R}_\mathtt{i}^{t_k}$ and one from $\mathsf{R}_\mathtt{j}^{t_k}$, for which such a continuous trajectory can be found. Similarly, we call two regions $\mathsf{R}_i^{t_k}$ and $\mathsf{R}_j^{t_{k+1}}$ connected if there exists a continuous trajectory of acceptable postures, $\theta(t)$, with $t \in [t_k, t_{k+1}]$, connecting any two postures $\theta_i \in \mathsf{R}_i^{t_k}$ and $\theta_j \in \mathsf{R}_j^{t_{k+1}}$. As a result, a fault tolerant trajectory exists if and only if there exists a sequence of connected regions, $\{\mathsf{R}_i^{t_1}, \ldots, \mathsf{R}_j^{t_{last}}\}$. This result is used in the second part of our algorithm, in which we build a connectivity graph representing the connections between the regions $\mathsf{R}_i^{t_k}$. The structure of this graph is in general very simple due to the limited number of disjoint regions in each A^{t_k}, and due to the limited number of connections between regions at time t_k and regions at time t_{k+1}. It is possible that there exists no fault tolerant sequence of connected regions. To achieve fault tolerance in this case, the manipulator itself needs to be adapted by changing its structure, joint limits, or redundancy resolution algorithm.

In the final step of the algorithm, a fault tolerant trajectory is determined from the sequence of connected regions. In general, there are an infinite number of possible fault tolerant trajectories. However, a good trajectory should vary smoothly and stay away from the boundaries of the regions $\mathsf{R}_j^{t_k}$. The choice of one specific fault tolerant trajectory can be further limited by imposing additional task requirements or objectives.

Part 2: search for a fault tolerant trajectory

```
• for k=last to first do
    • group the acceptable postures, $A^{t_k}$, in disjoint
                                                  regions $R_i^{t_k}$
    • for each region $R_i^{t_k}$, determine the connections with
                                            the regions for k+1
    • store in connectivity graph
• next k
• search the connectivity graph to determine a fault
                        tolerant sequence of the regions $R_i^{t_k}$
• select a fault tolerant trajectory
```

4.4 Implementational issues

4.4.1 Computation of the Preimage

Although most of the steps of the algorithm, presented in the previous section, can be easily implemented, the computation of the preimage, $\Sigma(p)$, requires some further explanation. As mentioned before, for an n-DOF manipulator, the preimage of a point, $p \in \Re^m$, is a set of r-dimensional manifolds in the n-dimensional torus T^n, where $r = n - m$ is the degree-of-redundancy of the manipulator. The preimage is defined implicitly by the forward kinematics function $f(\theta) = p$. The computation of the preimage involves translating this implicit representation into an explicit one, for example, a random sampling of the preimage stored as a finite set of postures $\theta \in T^n$. However, this particular representation is insufficient for our algorithm because it does not capture the topology of the preimage. Topological information is needed in three steps of the algorithm: first, where the sets $F_j^{t_k}$ are computed; second, where the intersection of these sets is taken to obtain A^{t_k}; and third,

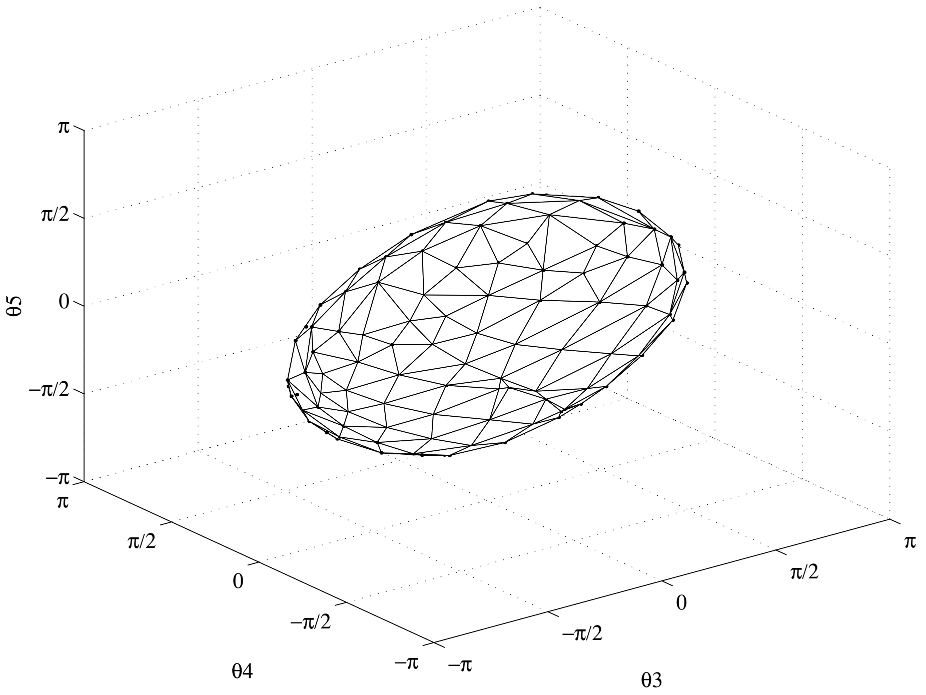


Figure 4-2. The projection onto the $(\theta_3, \theta_4, \theta_5)$-space of a polygonal approximation of a 2-dimensional preimage for a 5-DOF manipulator.

where the acceptable postures are grouped into disjoint regions $\mathrm{R}_j^{t_k}$. It is important to notice that in all three instances only the local topology matters. Locally, an r-dimensional manifold is diffeomorphic to $\Re^r$, and can thus be approximated by an r-dimensional hyperplane. Therefore, we have chosen to represent the preimage $\Sigma(p)$ by a polygonal approximation consisting of line segments when $r = 1$, or triangular patches when $r = 2$, as is illustrated in Figure 4-2.

In the next chapter on Task Based Design, the global fault tolerant trajectory planning algorithm is integrated into the function that evaluates whether a candidate manipulator design meets all the task requirements. During the design process, this function is executed many times and for a large variety of different manipulators, some of which may be degenerate (all axes are parallel, axes coincide etc.). This requires that the global fault tolerant trajectory planning algorithm be *fast* and very *robust*. We have not yet been able to achieve both of these attributes simultaneously for manipulators with more than one degree-of-redundancy. Therefore, we will limit ourselves in the next chapter to the design of fault tolerant manipulators with at most one degree-of-redundancy.

4.4.2 Computational Complexity

Let S be the number of postures $\theta \in T^n$ used to approximate $\Sigma(p)$, then S increases as the accuracy of the approximation increases. S also depends on the dimensionality, r, of the preimage; this dependency is *exponential*. The algorithm also requires the Cartesian path $p(t)$ to be approximated by a sequence $\{p_k\}$ with $p_k = p(k\Delta t)$. Let P be the number of points in the sequence $\{p_k\}$. Just like S, P depends on the accuracy of the approximation. In this case, the dependency is always linear because the Cartesian path is 1-dimensional.

The complexity of the algorithm is mainly determined by the nested loop of the first part of the algorithm. Assuming that the complexity of the redundancy resolution algorithm is linear in n, the complexity of our trajectory planning algorithm can be expressed as:

$$P \cdot n \cdot S \cdot O(n) \quad \text{or} \quad P \cdot e^r \cdot O(n^2). \tag{4-7}$$

Because of the exponential dependency on r, the algorithm is only practical for $r = 1$ or $r = 2$.

Because the fault tolerant trajectory planning algorithm is included in the evaluation function of the task based design framework in the next chapter, it is very important that the trajectory planning algorithm be implemented efficiently. An important property, in this respect, is the progressive nature of the computation. Starting at the end of the specified Cartesian path, we work our way forward computing only those parts of the preimage manifolds that are likely to contain postures tolerant to failures (the sets $\mathsf{F}_j^{t_k}$). When for a certain t_k no acceptable postures remain, i.e., $\mathsf{A}^{t_k} = \varnothing$, one is guaranteed that a fault tolerant trajectory does not exist and the computations can be interrupted. As a result, relatively little computational effort is wasted trying to determine a fault tolerant trajectory for a “bad” manipulator configuration.

4.4.3 Selection of Trajectory

In the last step of part two of the fault tolerant trajectory planning algorithm, a joint space trajectory is selected from a sequence of connected regions, $\{\mathsf{R}_i^{t_1}, \ldots, \mathsf{R}_j^{t_{last}}\}$. There exist an infinite number of trajectories that lie within this sequence, and one can impose additional criteria that have to be met by the trajectory. In our implementation, we have opted for a combination of two criteria: robustness and minimal length. Through local optimization with the BFGS method (Fletcher 1987), we determine a trajectory that minimize a cost function that combines the length of the trajectory and its distance to the boundary of the connected regions. The further the trajectory stays from the boundaries of the connected regions, the more robust it is in maintaining the property of fault tolerance, despite small deviations from the fault tolerant trajectory. Minimizing the length of the trajectory has the additional benefit of yielding a smooth trajectory with small joint velocities.

4.5 Illustrative Example

In this section, we illustrate the use of the fault tolerant trajectory planning algorithm with an example of a 3-DOF planar manipulator. This simple example enables us to describe graphically how a fault tolerant trajectory is selected.

The 3-DOF manipulator has 3 links of length 1; the joint limits are $\pm 100°$ for θ_1 and $\pm 150°$ for θ_2 and θ_3; no redundancy resolution algorithm is specified because the 2-DOF reduced order derivatives are non-redundant. The task is to follow the trajectory shown in Figure 4-3 at constant speed in a total time of 10 seconds; a circular obstacle is centered at $(0.7, 0)$ and has a radius of 0.2.

Because the manipulator in this example has one degree-of-redundancy, $r = 1$, the preimage of every point, $p(t)$, is a one dimensional subset of T^3 and can be parametrized as $\Sigma(p(t)) = g(p(t), \alpha)$ with $\alpha \in T^1$. The function g describes a 2-dimensional surface in T^3, as illustrated in Figure 4-4. The surface can be parametrized by two parameters: the Cartesian position p (or time t), and the preimage parameter α. The postures labeled *a*, *d*, *e*, and *f*, are all part of the preimage of the same point, $p(0)$ (or $p(10)$). The postures labeled *a*, *b*, and *c*, on the other hand, have the same value for the preimage parameter, $\alpha = 0$ (or $\alpha = 2\pi$), but map onto different points, p, along the path. One can also unwrap this surface and represent it in a planar coordinate system with the time in abscissa and the preim-

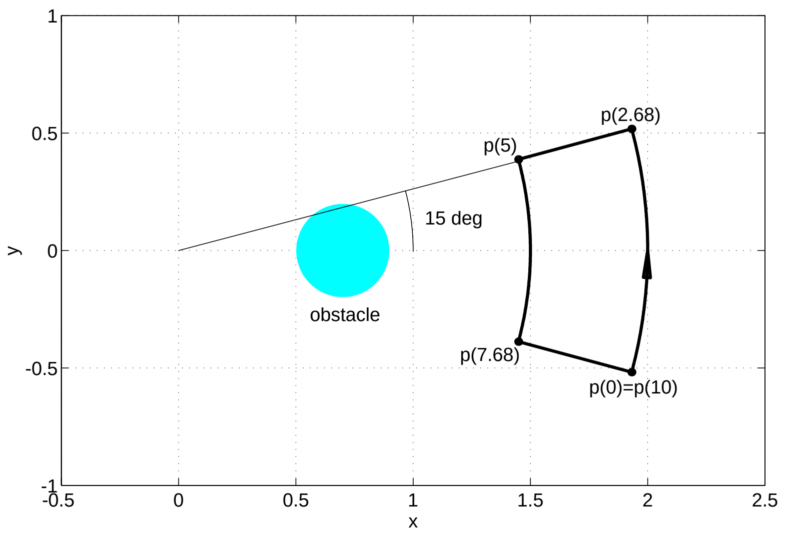


Figure 4-3. The Cartesian path and obstacle position.

age parameter α in ordinate(3); this representation is used in Figures 4-5a through 4-5d. Table 4-1 gives the coordinates of the six postures, *a* through *f*, for both the three-dimensional and the two-dimensional representations of Figures 4-4 and 4-5, respectively.

In the first part of the algorithm, the sets $\mathsf{F}_j^{t_k}$ are determined. They are depicted in Figures 4-5a through 4-5c as the white areas. The dark gray areas are postures that do not satisfy the secondary task requirements; i.e., they are the sets $\Sigma(p(t)) - \mathsf{S}$. These postures would be unacceptable for a joint trajectory even if fault tolerance were not required. The light gray area is the set of postures for which the alternate trajectories do not meet the task requirements. The alternate trajectories $\theta(t, j, t_k)$ for this example are totally determined by keeping the joint angle θ_j constant, and are represented by the black curves. Notice that, for the postures in the light gray area, the alternate trajectories either pass through a posture that violates a secondary task requirement, or get stuck at a singularity and do not reach the end of the path. In either case, the requirement for fault tolerance is violated. The sets of acceptable postures A^{t_k} are indicated in white in Figure 4-5d. This white area is the intersection of

(3) Keep in mind that the planar representation does not capture the exact topology of the 2-dimensional surface, because the preimage parameter α is an element of T^1 and the path $p(t)$ is closed, $p(0) = p(10)$.

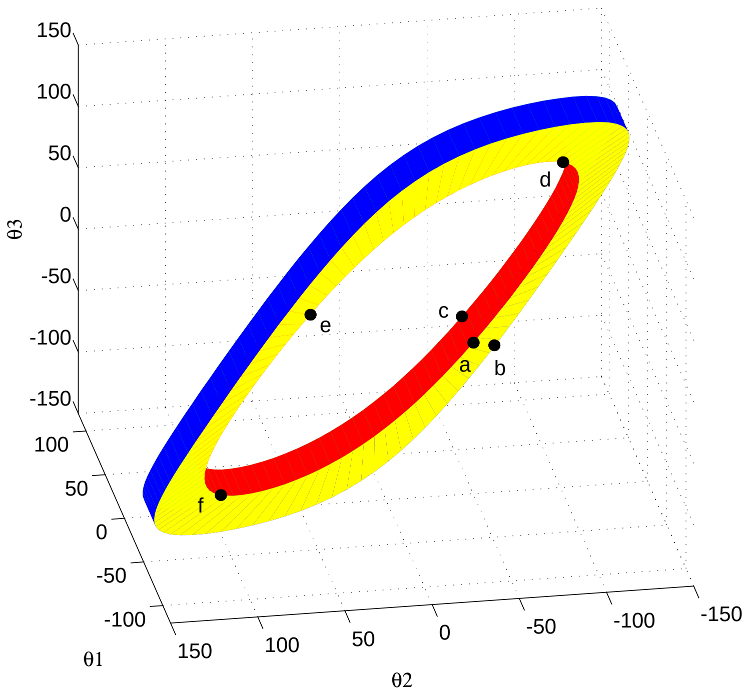


Figure 4-4. The preimage of the trajectory.

posture	$\theta 1$ [deg]	$\theta 2$ [deg]	$\theta 3$ [deg]	*t* [sec]	α [rad]
a	45.00	-60.00	-60.00	0 and 10	0 or 2π
b	75.00	-60.00	-60.00	2.68	0 or 2π
c	61.52	-75.52	-75.52	7.68	0 or 2π
d	13.95	-104.48	104.48	0 and 10	$\pi/2$
e	-75.00	60.00	60.00	0 and 10	π
f	-43.95	104.48	-104.48	0 and 10	$3\pi/2$

Table 4-1. The coordinates for each of the postures labeled a–f in Figure 4-4.

Joint angles $\theta 1$, $\theta 2$, and $\theta 3$:	the coordinates for the 3D representation of the preimage (Figure 4-4)
time *t* and preimage parameter α:	the coordinates for the 2D representation of the preimage (Figure 4-5)

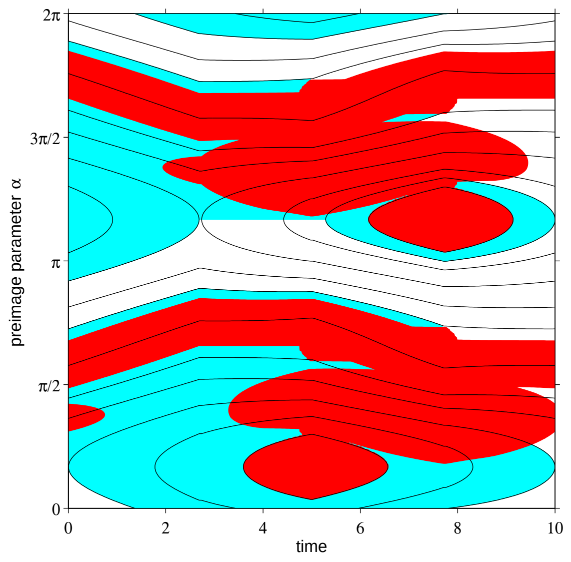


Figure 4-5a. The set of postures tolerant to a fault in joint 1 (in white).

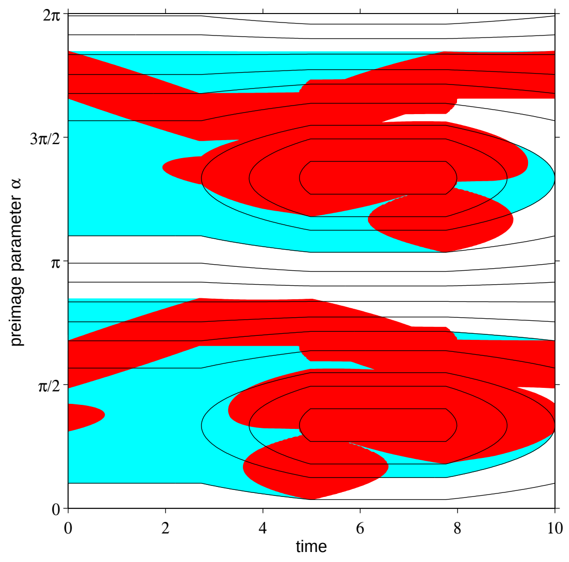


Figure 4-5b. The set of postures tolerant to a fault in joint 2 (in white).

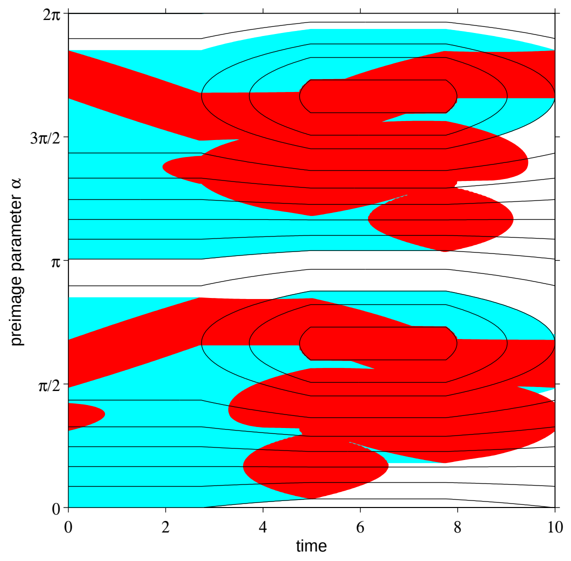


Figure 4-5c. The set of postures tolerant to a fault in joint 3 (in white).

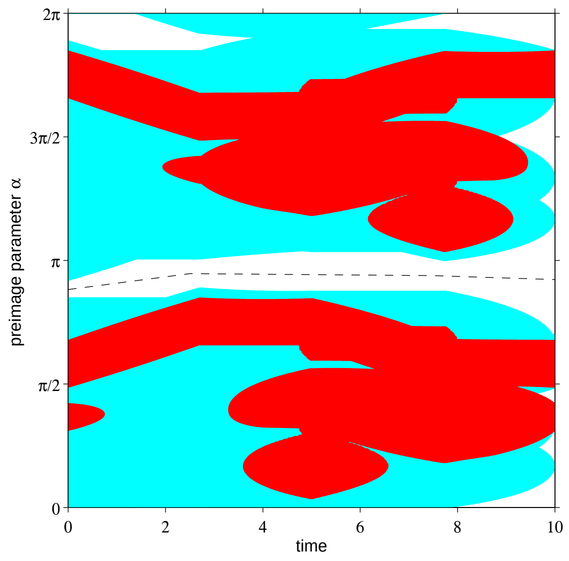


Figure 4-5d. A possible fault tolerant trajectory (dashed line). The white areas are the sets of acceptable postures.

Figure 4-5. Planning a fault tolerant trajectory: an illustrative example
The dark gray areas are postures that violate the secondary task requirements (joint limits and obstacles). The light gray areas are postures for which the alternate trajectory violates the task requirements. The black curves are alternate trajectories, determined by keeping the angle of the failing joint constant.

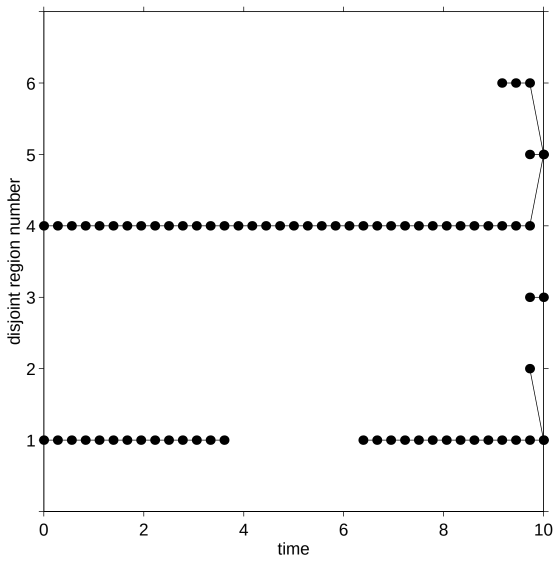


Figure 4-6. Connectivity graph for the disjoint regions of acceptable postures.

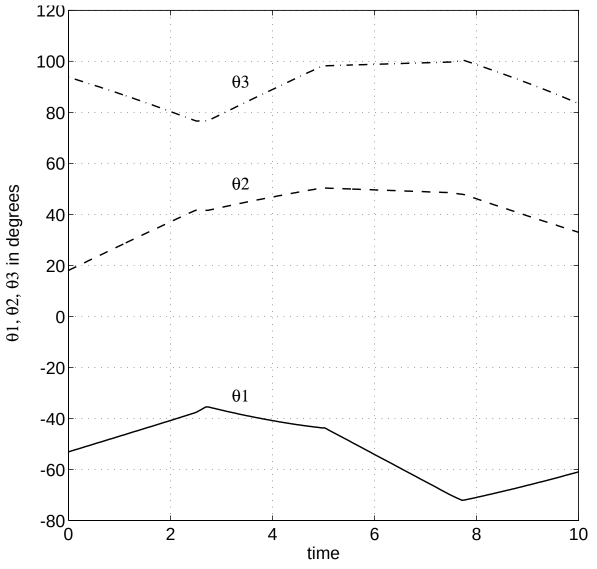


Figure 4-7. Fault tolerant trajectories for the individual joints.

the white areas in Figures 4-5a, 4-5b and 4-5c, in accordance with Equation (4-2).

In the second part of our algorithm, the acceptable postures are first grouped into disjoint regions. Such a region, $R_i^{t_k}$, corresponds to a vertical white line segment with abscissa t_k in Figure 4-5d. The number of disjoint regions is usually small—a maximum of six for this example. Once the regions $R_i^{t_k}$ have been determined, the connections with the disjoint regions at time t_{k+1} are stored in a connectivity graph, which is shown in Figure 4-6. As mentioned before, the graph is very simple in general. For this example, there exists only one fault tolerant sequence of connected regions. A possible fault tolerant trajectory for this sequence is shown as a dashed line in Figure 4-5d. The corresponding individual joint trajectories are depicted in Figure 4-7.

4.6 Comprehensive Example

In this section, we describe an example of fault tolerant task execution by the 4-DOF spatial manipulator shown in Figure 4-8. The manipulator is a configuration of the Reconfigurable Modular Manipulator System (RMMS), and consists of the manipulator base, three pivot

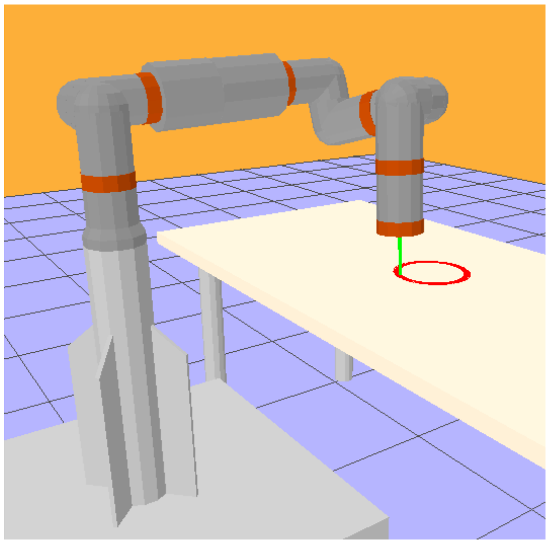


Figure 4-8. A simulation of the Reconfigurable Modular Manipulator System executing a fault tolerant trajectory.

dof	link offset	link length	twist angle
1	-0.1373	0.0	π/2
2	0.7344	0.0	π/2
3	-0.1373	0.3270	-π/2
4	-0.1373	0.4705	0.0

joint modules, one rotate joint module, and a link module. The modules are assembled such that the resulting manipulator has the Denavit-Hartenberg parameters listed in Table 4-1. The joint limits for each of the joint modules are $\pm 165°$. The task is to follow a circular path on a table while avoiding collisions with the table—even after one of the joint modules has failed and is immobilized.

The simulation uses the same control software running on the same hardware as would be used to control the actual manipulator system. The only difference is that the robot interface is replaced by an interface to the TeleGrip simulation software package (by Deneb Inc.). TeleGrip runs on a SGI Crimson which is connected to the VME-based control hardware through a VME-to-VME-adaptor. We use a damped least-squares kinematic controller with null-space optimization:

$$\dot{\theta} = J^*(\dot{p} + \alpha(p - x)) + \beta(I - J^*J)(\theta_{ft} - \theta), \tag{4-8}$$

where J^* is the singularity robust Jacobian inverse (Kelmar and Khosla 1990), $p(t)$ is the desired Cartesian path, and $\theta_{ft}(t)$ is the fault tolerant trajectory. The null-space optimization component of the controller ensures that the manipulator follows the desired fault tolerant

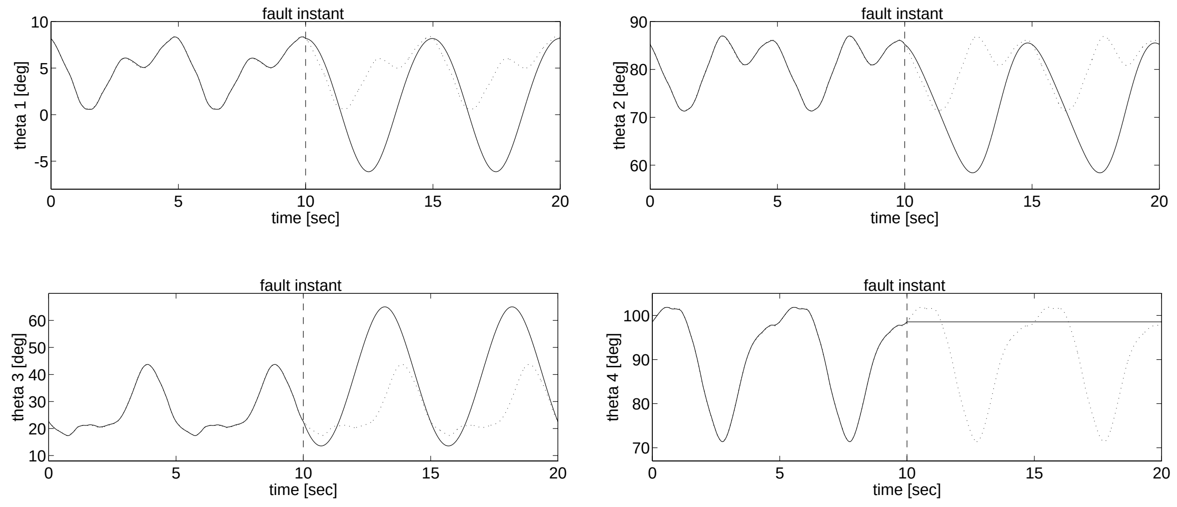


Figure 4-9. Joint trajectories theta 1 through theta 4.
Before failure (0–10sec), the manipulator follows the fault tolerant trajectory. At the tenth second, joint 4 fails and is immobilized. The manipulator then deviates from the fault tolerant trajectory (dotted line) and follows the alternate trajectory (solid line) as determined by the redundancy resolution algorithm.

joint space trajectory closely before a failure occurs, as is shown in Figure 4-9. After failure, the same controller is used with the column of the Jacobian, which corresponds to the frozen joint, set to zero. The trajectories for the remaining joints deviate from the fault tolerant trajectory to ensure that the end effector continues to track the desired path.

Instead of using the kinematic controller given by Equation (4-8), one could linearly interpolate the fault tolerant trajectory before failure and only switch to Cartesian control after failure. Besides having to switch controllers at the instant of failure, this has the disadvantages that a dense sampling of the fault tolerant trajectory is required to achieve good end effector position accuracy. Furthermore, Equation (4-8) allows us to move back smoothly to the fault tolerant trajectory if the failure was temporary.

Figure 4-10 shows the position error of the end effector. The error does not increase after failure and remains an order of magnitude smaller than the distance, s, traveled by the end effector during one sample period:

$$s = \frac{2\pi \cdot 0.1\text{m}}{5\text{sec} \cdot 150\text{Hz}} = 8.38\times10^{-4}\text{m} \gg \text{error} \qquad (4\text{-}9)$$

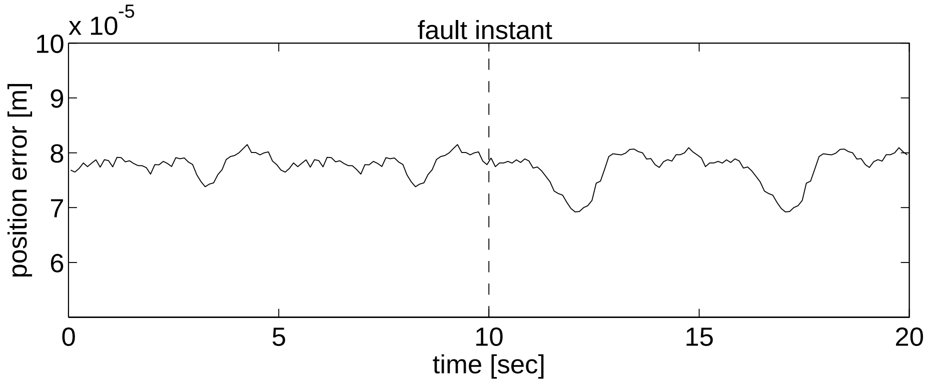


Figure 4-10. The end effector position error does not increase after failure.

The error could be further reduced by increasing the cycle frequency of the damped least-squares controller.

Many simulations with a wide variety of failure times and failing joints resulted in a maximum positional error of $8.67\times10^{-5}\text{m}$. This confirms the fact that the trajectory determined by our algorithm is indeed fault tolerant with respect to this task. Unfortunately, we cannot demonstrate this graphically as we did for the previous example. Because the topology of the preimage changes at the points where the path crosses the critical value manifolds of the manipulator (Burdick 1988), the preimage of the path cannot be unfolded into a 2-dimensional graph such as Figure 4-5.

When executing the task with the real hardware, instead of in simulation, one can notice a small jerking motion at the instant of failure. This motion was not visible in the simulation because the simulation is purely kinematic and the jerking motion results from dynamic effects. More specifically, at the instant of failure, the velocity of the failing joint drops almost instantaneously to zero (the brakes can exert a very large torque). In order to maintain a constant end-effector velocity, the velocity of the other joints needs to change instantaneously too. However, this cannot be accomplished with the limited torque of the motors. This can be expressed mathematically as follows. Assume that joint k fails at instant t. Before as well as after the failure, the kinematic relationship has to be satisfied, while the end-effector velocity remains constant:

$$J(t)\dot{\theta}(t^-) = \dot{x}(t^-) = \dot{x}(t^+) = J(t)\dot{\theta}(t^+), \qquad (4\text{-}10)$$

or

$$\sum_{i=1}^{n} J_i(t)(\dot{\theta}_i(t^-) - \dot{\theta}_i(t^+)) = 0, \tag{4-11}$$

Where J_i is the i-th column of the Jacobian, and $\dot{\theta}_i$ is the velocity of joint i. After joint k has failed, the joint velocity $\dot{\theta}_k(t^+)$ equals zero, so that Equation (4-11) becomes:

$$\sum_{\substack{i=1 \\ i \neq k}}^{n} J_i(t)(\dot{\theta}_i(t^-) - \dot{\theta}_i(t^+)) = J_k(t)\dot{\theta}_k(t^-). \tag{4-12}$$

This equation can be interpreted as follows: unless the joint velocity of joint k was zero or did not contribute to the end-effector motion right before the instant of failure, a discontinues change in joint velocity has to occur in some or all of the other joints in order to maintain a constant end-effector velocity. As a result, a small jerking motion of manipulator at the instant of failure cannot be avoided, but it can be reduced by decreasing the end-effector velocity and hence decreasing the joint velocities.

4.7 Comparison

In this section, we compare our algorithm for global fault tolerant trajectory planning with the approaches for fault tolerant task execution described in (Lewis and Maciejewski 1994a), (Lewis and Maciejewski 1994b), and (Paredis and Khosla 1994).

Lewis and Maciejewski (1994a) propose a local redundancy resolution algorithm which maximizes the kinematic fault tolerance measure, *kfm*(4). The *kfm* is defined as the minimum remaining dexterity of the manipulator after joint failure:

$$kfm = \min_{f = 1..n} \sigma_m({}^{f}J), \tag{4-13}$$

(4) The authors also define a dynamic fault tolerance measure, *dfm*, which we do not consider in this thesis.

where $\sigma_m({}^f J)$ is the smallest singular value of the Jacobian of the original manipulator with the f-th column removed. One can think of the dexterity, σ_m, as the ease with which the end-effector of the manipulator can be moved in the least suitable direction. To achieve fault tolerance, it is important that, after a joint fails and is immobilized, all end-effector movements remain feasible, that is, σ_m remains nonzero. This idea is realized practically by using a redundancy resolution algorithm with null-space maximization of the *kfm*. Due to the local nature of the kinematic fault tolerance measure, however, this method cannot guarantee fault tolerance on a global sale. Moreover, it does not take secondary requirements such as joint limits and obstacles into consideration. Nevertheless, the method is important in case the desired end-effector path is unknown a priori; global off-line path planning is impossible in this case, so that local optimization of the *kfm* combined with joint limit and obstacle avoidance, is probably the best one can do. A drawback of the method is that it requires the computation of the gradient of the kfm, which in turn requires the computation of the full singular value decomposition of ${}^f J$; the computational complexity for the singular value decomposition of an $m \times n$ matrix is approximately (Golub and Van Loan 1989)

$$4m^2n + 8mn^2 + 9n^3. \tag{4-14}$$

Although this is quite computationally intensive for an on-line algorithm, it does not suffer from exponential complexity in the degree-of-redundancy, r, so that this method could be used to achieve local fault tolerance even in highly redundant manipulators.

Lewis and Maciejewski (1994b) acknowledge the local nature of the *kfm* approach and propose a global method for fault tolerant task execution. For every “critical task point” (a point that needs to be reachable after joint failure), a bounding box of the preimage manifold is computed. Every critical task point is reachable after joint failure, if the failure occurs at a posture inside the intersection of the bounding boxes of the preimage manifolds. Global fault tolerance is achieved by using a redundancy resolution algorithm that ensures that the joint space trajectory remains inside the intersection of the bounding boxes. This method is similar to our global fault tolerant trajectory planning algorithm to the extent that it uses the pre-computation of the preimage manifolds to achieve global fault tolerance. However, since it only uses the *bounding boxes* of the preimage manifolds, it cannot guarantee path

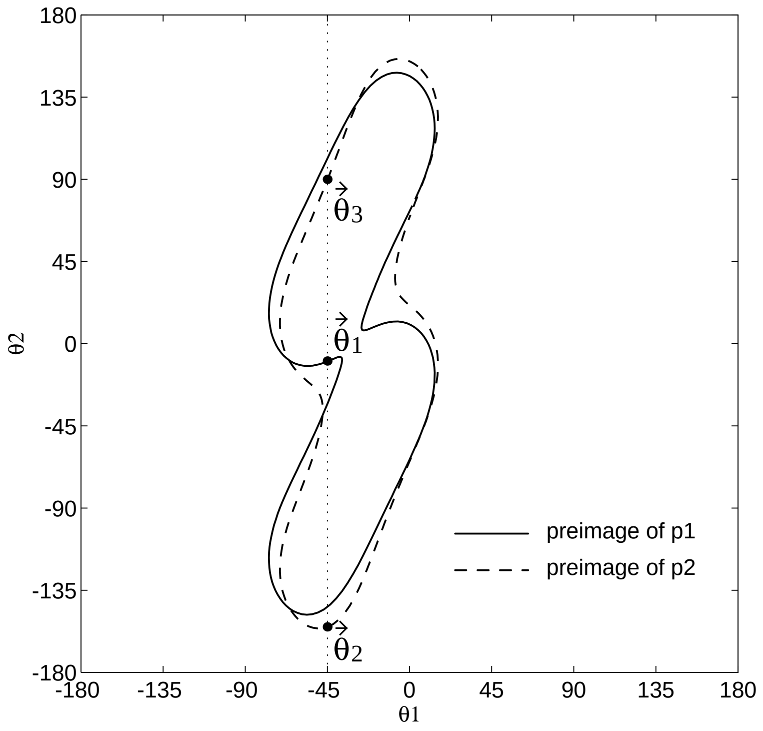


Figure 4-11. A case in which the method described in (Lewis and Maciejewski 1994b) cannot guarantee fault tolerant path following.

following. We can illustrate this drawback with the comprehensive example of Section 4.6. The desired Cartesian path passes through the two points, $p_1 = (0.763, 0.393, 0.783)$ and $p_2 = (0.895, 0.331, 0.783)$. Assume that the point p_1 is reached in the posture $\vec{\theta}_1$, which is inside the bounding boxes of the preimages of p_1 and p_2, as is shown in Figure 4-11. A failure occurs in joint 1, locking it at $\theta 1 = -45°$. To reach the point p_2, the manipulator would have to move to either $\vec{\theta}_2$ or $\vec{\theta}_3$, both of which would require a deviation from the desired path. On the contrary, in our global trajectory planning algorithm, $\vec{\theta}_1$ is not an acceptable posture because a failure of joint one would result in the redundancy resolution algorithm getting stuck at a singularity. Another disadvantage of the method described in (Lewis and Maciejewski 1994b) is that, just like in (Lewis and Maciejewski 1994a), secondary requirements are not taken into account. Also, the algorithm requires the computation of the preimage manifolds which is, as we have shown before, exponential in the degree-of-redundancy, r; this limits the usefulness of the algorithm to manipulators with $r = 1$ or 2.

A third approach to fault tolerant task execution is the one described in Chapter 3. This approach is different because it does not make any assumptions about the redundancy resolution algorithm used at run-time. Instead, fault tolerance is achieved at the design stage. A manipulator is designed which is able to reach every task point even when an arbitrary joint fails at an arbitrary angle. As a result of the assumption that a joint can fail at an *arbitrary* angle, at least two degrees-of-redundancy, instead of one, are necessary for 1-fault tolerance. An additional drawback is that the design approach does not consider obstacles—it does take joint limits into account. Moreover, it only guarantees reachability of task points, not path following.

In conclusion, the two main qualities that distinguish our fault tolerant trajectory planning algorithm from the other approaches to fault tolerant task execution are its ability to guarantee fault tolerance for path following, and its consideration of secondary requirements.

4.8 Summary

In this chapter, we have presented a trajectory planning algorithm for fault tolerant task execution. This algorithm guarantees fault tolerance on a global scale, while also satisfying secondary kinematic task requirements such as joint limits, obstacles collision, and self collision. The algorithm consists of two main parts. In the first part, the postures acceptable for a fault tolerant trajectory are determined. The computations are based on the topology of the preimages of the Cartesian path, and on the characteristics of the redundancy resolution algorithm which is used after a failure has occurred. In the second part of the algorithm, a connectivity graph is constructed, representing the topological structure of the set of acceptable postures. By searching this graph, a global fault tolerant trajectory is found. A simple example for a 3-DOF planar manipulator is used to explain the development of the algorithm graphically. A second more comprehensive example further illustrates some of the kinematic control issues of fault tolerant task execution. Compared to other approaches for fault tolerant task execution, our method has the advantage that it guarantees fault tolerance for path following, and that it takes secondary requirements such as joint limits and obstacles into consideration.

Chapter 5

Task Based Design: An Agent-Based Approach

5.1 Introduction

This chapter is the most important part the thesis. It ties together all the building blocks presented in the previous chapters—reconfigurable hardware, control software, fault tolerance, and trajectory planning—and turns them into a unified rapidly deployable fault tolerant manipulator system. Indeed, to be able to rapidly deploy a reconfigurable modular manipulator system, one needs to solve the *Task Based Design* (TBD) problem—that is, one needs to create a framework for answering the question: which modular configuration should be used to perform the given task? This question cannot be answered correctly without taking into account the complete manipulator system, including control and trajectory planning software. This is especially true for the design of task specific *fault tolerant* manipulators, because the guarantee for task specific fault tolerance depends critically on the fault tolerant trajectory and redundancy resolution algorithm.

Unfortunately, TBD is a very difficult problem. The size of the design space grows exponentially with the number of modules in the inventory; the constraints and optimization criteria are highly coupled and non-linear; and evaluating whether the constraints are satisfied and to which extent the optimality criteria are achieved is very computationally expensive.

To overcome this complexity, we introduce a novel agent-based approach to TBD which is based on genetic algorithms. All of the agents contain problem specific knowledge to reduce the size of the search space and to reduce the cost of evaluation whether the constraints are satisfied. This results in a more effective search strategy than a standard genetic algorithm. Moreover, our agent-based framework can be easily executed in parallel on a distributed network of workstations. This combination of an effective search strategy with the high computing performance of a system of networked workstations results in a very powerful new approach which allows us to solve complicated TBD problems within a reasonable amount of time.

5.2 The Task Based Design Problem

The goal of TBD is to design a manipulator that is optimally suited to perform a given task. More specifically, as is illustrated in Figure 5-1, the class of design problems considered in TBD is defined as follows:

Given:

- a description of the task
- an inventory of available RMMS modules

Find:

- a manipulator configuration of RMMS modules and its base position/orientation
- a desired joint space trajectory

5.2.1 Task Definition

The first input component of the TBD problem is a low-level task description consisting of a timed Cartesian trajectory and obstacles models. There are many different levels of abstraction at which a manipulation task can be defined. For instance, consider the assembly of an electric motor. At the highest level, this task could simply be described as: “assemble electric motor.” At an intermediate level, the task is split into subtasks such as: locate rotor, pick up rotor, insert rotor into stator, etc. At the lowest level, all the details are included: move to point A, exert a force of x Newtons in the Z-direction, etc. Assuming that the task is feasible, one can translate high-level task descriptions into low-level task descriptions. Carriker (1995) addressed this task level planning problem for assembly tasks. He presents a framework for an assembly planning and execution system (CAPEK) which takes high level assembly plans generated by an assembly sequence planner, and generates low level sequences of robotic skills that can be used to execute the assembly task automatically with a manipulator. Although this is a very interesting problem which still remains to be fully resolved, it is not the focus of this thesis. Instead, we assume that a low-level task description is provided. Specifically, the task description consists of a timed Cartesian path to be followed by the end-effector, and a description of the obstacles in the manipulator work-

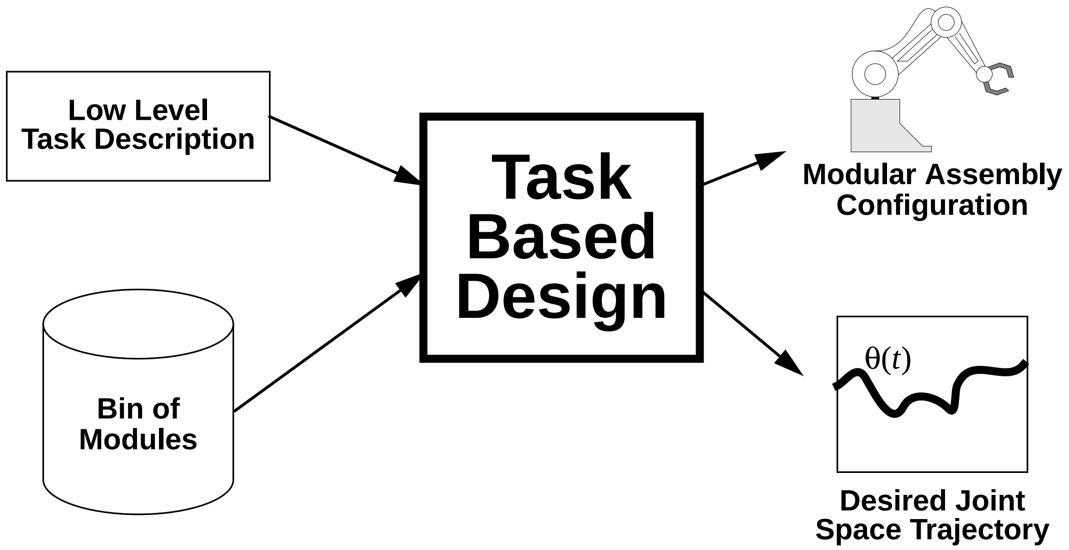

Low Level Task Description
Bin of Modules
Task Based Design
Modular Assembly Configuration
θ(t)
Desired Joint Space Trajectory

Figure 5-1. Task Based Design problem definition.

space. The user also needs to indicate whether the end-effector orientation should be taken into account and whether that task should be executed fault tolerantly.

In many respects, this is a more complete task description than has been used in earlier research on TBD (Chedmail and Ramstein 1996, Chen 1994, Kim 1992, Murthy 1992, Paredis 1990). In all five references, a task is defined as a sequence of points to be reached by the end-effector—no time information and no specification of the behavior in-between points is given. Without time information, it is impossible to consider dynamic task requirement. Previously, only Murthy (1992) considered dynamic task requirements. He specified desired velocities and accelerations at each of the given task points, but failed to guarantee that these velocities could be reached during the time in-between task points. Besides the desired Cartesian path, obstacles constitute an important constraint on the kinematic structure of task specific manipulators. Yet they are only considered as part of the task description by Chedmail and Ramstein (1996) and by Paredis (1990). Finally, this is the first approach to TBD that considers fault tolerance as a criterion.

Although manipulability and other dexterity measures have been often studied as a task criterion in TBD (Chen 1994; Kim 1992; Murthy 1992; Paredis 1990), we do not consider them in this thesis. Manipulability is a measure for the manipulator's ability to arbitrarily change the position and orientation of its end-effector (Yoshikawa 1985). It is an important measure for the design of general purpose manipulators for which arbitrary changes in end-effector position and orientation are desirable. However, in TBD, the task is known a priori and, therefore, it is not important whether an arbitrary change in end-effector position/orientation can be achieved rather than the change in position/orientation required by the task.

The main objective of TBD is to find a *feasible* solution, that is, a manipulator that is able to execute the task successfully—without violating any task constraints. In addition to task constraints, one could also consider *optimality criteria*. In our current implementation, we use energy consumption as an optimality criterion.

5.2.2 Inventory of Modules

Besides a task specification, TBD requires and inventory of modules as an input. The inventory of modules specifies the set of modules from which candidate manipulators design can be built. Each of the modules in the inventory is described by a module definition file. The same module definition files are also used by the hardware controller, and the TeleGrip simulation software. An example of such a file can be found in Appendix A. The descriptions provided in the files include kinematic parameters, dynamic parameters, CAD-models for collision detection, and sensor calibration parameters. Any module with only one input connector and one output connector can be completely defined in this format, regardless of the joint type (prismatic or revolute) and regardless of the number of DOFs.

Limitations on module parameters are a source for additional design constraints: joint angle limitations, joint velocity limitations, and joint torque limitations. Furthermore, the `MOD_TYPE` parameter limits the number of feasible assembly configurations; base-type modules can only appear as the first module of a manipulator configuration, end-effector-type modules can only be mounted last, while all the other modules can be used anywhere except first or last.

5.2.3 Manipulator Configurations

The first output component of the TBD problem is a manipulator configuration. A modular manipulator configuration consists of the position and orientation, (x, y, φ), of the base module, and an ordered list of modules with their relative assembly orientations. For the RMMS hardware, a module can be assembled in eight different orientations (at 45 degree intervals) with respect to the previous module.

Besides the constraints arising from the module type, the user can define additional constraints limiting the acceptable manipulator configurations. One can restrict the base position of the RMMS to remain within rectangular bounds. Furthermore, one can specify bounds on the number of DOFs. This is important, especially from the perspective of the search algorithm used to solve the TBD problem. By providing a tight estimate for the num-

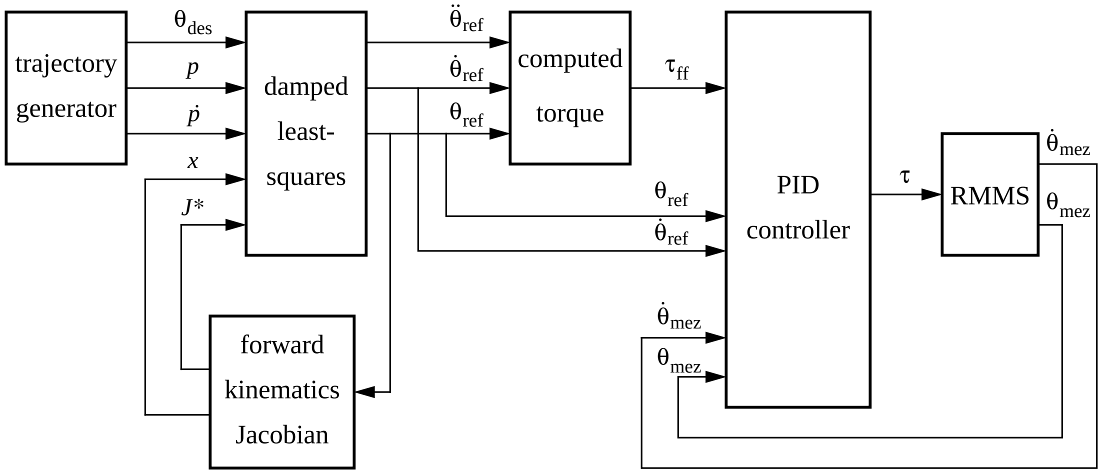


Figure 5-2. The control structure of the RMMS

ber of DOFs, one can limit the size of the design space which improves the convergence of the search algorithm.

5.2.4 Desired Joint Space Trajectory

In addition to the manipulator configuration, the output of the TBD problem includes a desired joint space trajectory. It is typical for a wide range of engineering design problems that the design task consists of determining not only an artifact, but also the artifact's behavior or function. For a manipulator, the behavior is determined by the controller and the path planner. We have limited ourselves to the control structure illustrated in Figure 5-2. The damped least-squares kinematic controller has the following structure:

$$\dot{\theta}_{ref} = J^*(\dot{p} + \alpha(p - x)) + \beta(I - J^*J)(\theta_{des} - \theta_{ref}) \tag{5-1}$$

where J^* is the singularity robust Jacobian inverse (Nakamura and Hanafusa 1986; Kelmar and Khosla 1990). The objective of this controller is to follow the desired Cartesian path, p, while staying as close as possible to the desired joint space trajectory, θ_{des}, through null-space optimization. The advantage of using this control structure rather than executing the desired joint space trajectory directly with a joint space controller, is that one can achieve good end-effector accuracy with only a limited number of via points in joint space. In other words, when designing the behavior of the manipulator, one needs to provide only a *coarse sampling* of the desired joint space trajectory to guide the redundancy resolution, while the damped least-squares controller maintains good end-effector accuracy.

Providing the desired joint space trajectory as an output of the TBD problem, is particularly important when the successful completion of a task depends on the choice of redundancy resolution, as for example in the case of fault tolerance or obstacle avoidance. When the manipulator is non-redundant, on the other hand, the null-space of the Jacobian is empty, and the second term in Equation (5-1) reduces to zero. In this case, the behavior of the manipulator is independent of the desired joint space trajectory. Yet, even the inverse kinematics of non-redundant manipulators have multiple solutions—at most 16 solutions for a 6R serial link manipulator (Raghavan and Roth 1993). Therefore, one does have to include as a design variable the posture of the manipulator at the start of the Cartesian path.

5.3 Problem Characteristics

In order to pick the best solution approach for the TBD problem, it is important to understand the problem characteristics. Tong and Sriram (1992) classify engineering design problems according to the following criteria:

- available methods and knowledge;
- amount of unspecified (physical) structure;
- complexity of interactions between subproblems;
- amount and type of knowledge the user can provide.

Available methods and knowledge. TBD can be classified as an *innovative* design task. Tong and Sriram (1992) distinguish between three levels of complexity: routine, innovative, and creative. If an appropriate method and/or sufficient knowledge is available to translate the functional design requirements into an acceptable design without any search, the design task is called a *routine* task. If the available knowledge by itself provides unacceptable designs, but an acceptable solution can be generated through a limited search, we call the design process *innovative*. Finally, if the construction of the search space itself requires problem-solving, or if the best known design method is an unguided search through a very large space, we have a *creative* design task. According to those definitions, Task-Based-Design of the RMMS qualifies as innovative: the design space is well known, but there is insufficient design knowledge available so that a search is required.

The knowledge that is available about TBD of manipulators can be divided into two categories: knowledge about the design of the structure and knowledge about the design of the behavior. It might come as a surprise to the reader that we consider knowledge about the behavior of the manipulator, i.e. planning and control, as *design* knowledge. Remember, however, that trajectory planning plays a critical role in the design of task specific fault tolerant manipulators. Therefore, one could look at the determination of the desired joint space trajectory, θ_{des}, as a subproblem of the overall TBD problem. As we have shown in Chapter 4, there is considerable knowledge available about the fault tolerant trajectory planning subproblem, to the extent even that one can consider this subproblem a routine design task—there exists an algorithmic solution to the problem. Even though the fault tolerant trajectory planning algorithm is computationally expensive, it is much desirable over *searching* for a trajectory.

As for the sub-task of designing the manipulator structure itself, very little knowledge exists—most of the manipulator design knowledge pertains to the design of *general purpose* manipulators and is not applicable to TBD. The first structured investigations of manipulator design started in the early eighties. Vertut and Liégois (1981) listed a set of general design criteria for manipulators, and provided a software for analyzing these criteria, given a specified manipulator. The optimization of the design parameters was left to the designer. Gupta and Roth (1982) focussed on kinematic design criteria, and proved that the hand size of a manipulator should be as small as possible in order to maximize the dexterity; they did not investigate the positional structure. Lin and Freudenstein (1986) developed a systematic search procedure to maximize the workspace and the workspace-to-void ratio of a 3R regional structure. The procedure was mainly based on workspace analysis combined with an exhaustive search of a limited number of parametrized designs. A more rigorous approach is described by Vijaykumar, Waldron, and Tsai (1986). The conclusion, based on geometric reasoning, is that an elbow manipulator with zero link offsets is optimal with respect to working volume and dexterity. The same conclusion was obtained by Tsai and Soni (1984) and was proved mathematically by Paden and Sastry (1988). Although the design knowledge described in the above mentioned articles is important for the design of general purpose manipulators, it is of limited use for Task Based Design. For instance, the criterion of workspace volume is important when one does not know in advance for which task the manipulator will be used; by maximizing the workspace, one increases the chance that the manipulator will be able to execute the still unknown task. However, if the task *is* known a priori, and one can design a manipulator specifically for that task, then workspace volume is relatively unimportant, as long as the manipulator is able to reach all the points required for the execution of the task. Furthermore, the papers, cited above, consider none or only a few design constraints such as joint limits, obstacles in the workspace, torque limits etc. The knowledge regarding optimal design of manipulators without constraints, does, in general, not apply to the highly constrained TBD problem. For instance, an elbow manipulator configuration, considered to be optimal in the above papers, becomes useless for a task constrained by two obstacles forming a narrow horizontal gap; the manipulator cannot move into the gap without striking the obstacles with the elbow (Paredis and Khosla 1993).

The first attempts to include task specificity at the design stage were based on an exhaustive search of a limited number of manipulator configurations. Tsai and Morgan (1985) made the assumption that an elbow manipulator is optimal for any task; only the size of the manipulator and its base position are considered as design parameters. A mechanism-design-based approach is presented by Manoochehri and Seireg (1990). The authors divide the manipulator design problem into two subproblems: form synthesis and dimensional optimization. First, a list of possible topologies is generated based on a user provided set of basic elements (single links and dyads). For each of those topologies, the dimensions are optimized with respect to the given task. Finally, the best optimized design is retained. Although this procedure probably yields good results for simple tasks, it is severely limited by the fact that an exhaustive search of the possible topologies is performed.

To avoid an exhaustive search, other researchers have used guided global search algorithms such as simulated annealing (Paredis 1990) and genetic algorithms (Chedmail and Ramstein 1996; Chen and Burdick 1995; Farritor et al. 1996; Kim 1992; Murthy 1992). Most of these papers include design knowledge implicitly by penalizing infeasible designs during the evaluation stage of the search algorithm. Only Murthy (1992) explicitly includes knowledge about the design. This knowledge is represented in the form of heuristics such as “move base to fix reach violation.” For the design of planar manipulators, as considered by Murthy (1992), these explicit design heuristics can be easily derived from kinematic and dynamic analysis. However, due to the highly nonlinear nature of the kinematics and dynamics of spatial manipulators, these heuristics cannot be easily extended to spatial manipulators, especially when considering such global criteria as fault tolerance or energy consumption.

In conclusion, the literature reveals that there is very little explicit knowledge available about task specific design of robot manipulators. The most promising results so far, have been obtained by using guided global search algorithms such as simulated annealing or genetic algorithms in which design knowledge is included implicitly in the evaluation of candidate solutions.

Amount of unspecified (physical) structure: TBD is a *structure configuration* design task. A design process involves creating a *complete* (physical) structure. However, in most engineering design tasks only a limited part of the structure still needs to be identified. In TBD of modular manipulators, the complete physical structure of each of the modules is specified a priori. The remaining unidentified part of the design is the base position/orientation, and the order and relative orientation of the manipulator modules. This kind of design task is typically called a *structure configuration task*: the unspecified structure is a configuration of parts of pre-determined type. The advantage to this kind of problems is that the design space is usually well known and easily parametrizable. Nevertheless, structure configuration tasks can be very challenging depending on the size of the design space. The number of different assembly configurations of the RMMS grows exponentially with the number of available modules (approximately as $n!8^n$, where the number 8 corresponds to the number of possible relative orientations between two successive modules). In addition, for every assembly configuration, one has to consider all possible base positions/orientations (continuously varying).

Complexity of interactions between subproblems. TBD is complicated by a strong interaction between its subproblems and by the global nature of its constraints and optimality criteria. In general, complexity of design problems increases when the number of interactions between subproblems increases and when the type of interaction becomes more complex. As mentioned earlier, one can think of TBD as consisting of two subproblems: (1) the design of the manipulator structure and (2) the planning of the trajectory. This is a convenient division of the problem because the trajectory planning subproblem can solved algorithmically. However, the two subproblems strongly interact. A trajectory planned for one manipulator structure is meaningless for another manipulator structure.

A fundamental principle in engineering design is “the division of tasks” (Pahl and Beitz 1996) or “independence of functional requirements” (Suh 1988). The idea is to divide the design problem in subproblems that each address an independent functional requirement. For instance, in the design of a general purpose manipulators, the functional requirement is to be able to position the end-effector in an arbitrary position and orientation. This requirement is commonly divided into the requirements to position the end-effector and to orient the end-effector. For each of these sub-tasks a different structure of the manipulator is responsible: the positional structure and the wrist, respectively (There is a weak interaction when the hand length, the distance between the wrist center and the end-effector, is non-zero). In Cartesian manipulators, the positional structure is further divided into substructures that are responsible for positioning the wrist in each of the three independent coordinate directions. Although the principle of the division of tasks is a very valuable concept in general, it does not apply to the TBD of *fault tolerant* robot manipulator. As we indicated in Section 3.1 of Chapter 3, it is important for the design of fault tolerant manipulators that there be strong coupling between the manipulator DOFs, so that a failure in one of the DOFs can be compensated for by the remaining DOFs. For instance, when a failure occurs in one of the DOFs of a Cartesian robot, with fully decoupled DOFs, all capability to move in that particular direction is lost. This strong interaction between the subcomponents of a fault tolerant manipulator configuration complicates the TBD process.

In addition to the subcomponent interactions introduced by the fault tolerance requirement, other strong interactions arise from the global nature of the constraints and optimality criteria. Constraints such as torque limits, or optimality criteria such as power consumption depend on all the parameters of a manipulator—kinematic, dynamic, and even electric. Furthermore, this dependence is highly non-linear due to the nature of manipulator kinematics and dynamics. To solve the design problem, one needs to determine the *inverse* of this highly non-linear mapping, i.e., the mapping from task requirements to design variables. This is currently beyond the capability of modern mathematics—except for a few toy problems (Paredis and Khosla 1993).

Amount and type of knowledge a system user can provide. One can consider the human users of a design system as knowledge sources. For instance, if the user can provide assistance to the design system when it lacks the knowledge to continue, the design task can still be considered “routine.” It is my experience, however, that for TBD of fault tolerant manipulators, the human lacks the knowledge and intuition to assist the design system. It seems that our knowledge of manipulator design is mainly based on our experience with a rather limited set of common manipulator configurations. Most of these common configurations are simple structures with parallel or perpendicular motion axes. These designs make good general purpose manipulators, but are usually not optimal (or even feasible) for specific tasks, as we will show in Chapters 6 and 7. Furthermore, satisfying the fault tolerance requirement requires the simultaneous consideration of all possible failures in each of the DOFs at any time during the task execution. This amounts to such an enormous number of possible failure scenarios that it becomes impossible for a human to consider them all at once.

In conclusion, TBD is an innovative design task that can be split into two subproblems: the design of the manipulator structure, for which very little design knowledge exists, and the design of the manipulator’s behavior, which is a routine design task (solved algorithmically). One can consider TBD to be a structure configuration task for which the design space is well known and easy to represent, yet very large. The complexity of the TBD problem arises mainly from the degree of interaction between the subproblems, and from the global nature of the constraints and optimality criteria. Even for humans, these interactions are very difficult to understand or predict intuitively.

5.4 Previous Solution Approaches

The initial approach to task based design (Krishnan 1989; Paredis 1990) has been to divide the design problem into several stages as is shown in Figure 5-3. First, we design the kinematic structure; second, the dynamic structure. We then iterate between these two design processes until a solution is found, and end by designing a controller and planning a trajectory. While this approach was a reasonable first attempt, it had the disadvantage that it was slow. Since the kinematic parameters depend on design choices made during the dynamic design phase, several iterations of the design process were needed. The search for a kinematic design was performed by simulated annealing (Kirkpatrick, Gelatt and Vecchi 1983). This is a global search algorithm that, under certain assumptions, is guaranteed to converge to the global minimum with a probability of one. However, it has the disadvantage that it is

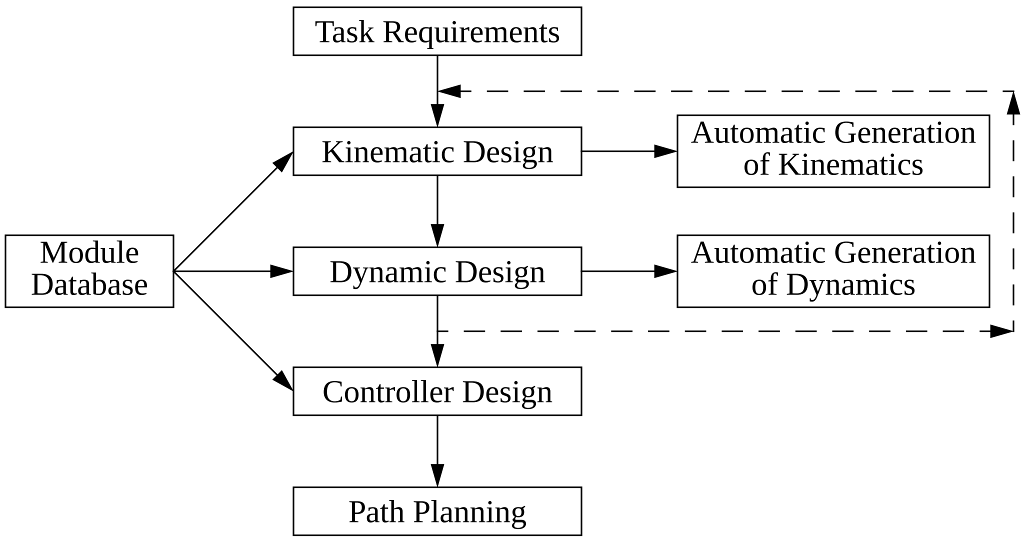


Figure 5-3. Initial design approach for the RMMS.

rather slow and that it cannot easily be parallelized. The objective function of the simulated annealing algorithm was a penalty function in which penalties for violating task constraints were combined in a weighted sum—no optimality criteria were considered.

Kim (1992) focussed also on the kinematic design of manipulators and used a Multi-Population Genetic Algorithm (MPGA) to search for a feasible solution. The idea was to have one GA per task point to search for locally optimal designs, and then progressively increase the coupling between the individual GAs to arrive at one single globally optimal design for all the task points. Kim also introduced the concept of Progressive Design—a coarse-fine approach in which the number of task points is progressively increased while the number of design variables is progressively decreased.

Murthy, Khosla, and Talukdar (1993) studied the task specific design of planar modular manipulators. They used an asynchronous team (A-team) approach to search the design space. An A-team is an organization of autonomous agents, that work in a cyclic, iterative, asynchronous fashion on common shared memories (Talukdar, de Souza, and Murthy 1993). In addition to mutation operators as in GAs, Murthy uses a large set of simple heuristic modification operators. Each of these heuristics is unable to generate good or even feasible designs; yet, by combining them into a team of agents, the population of candidate designs have been shown to converge to good solutions quickly and robustly. Unfortunately, the heuristics proposed by Murthy rely extensively on kinematic and dynamic analysis of *planar* manipulators and do not translate or scale well to the design of three-dimensional manipulators.

Chen (1994) addresses TBD of three-dimensional modular manipulators. He uses a genetic algorithm to optimize the Assembly Configuration Evaluation Function, which consists of two parts: structural evaluation and task evaluation; that is, in addition to evaluating how well a candidate manipulator design can execute the given task, Chen also evaluates the manipulator structure itself to discourage manipulators with degenerate sub-structures (for instance, coinciding rotation axes). This is especially important when the design of the manipulator modules being used is such that there is a high probability for degeneration.

Chedmail and Ramstein (1996) consider only kinematic constraints (reachability and obstacle avoidance) and build a cost function based on the length of the desired trajectory that can be followed by the manipulator without colliding with any of the obstacles. They also use a standard genetic algorithm to solve the global search for a feasible manipulator design.

Related to the TBD of robot manipulators is the design of modular mobile robots from task specifications. We found the following papers in the literature on this subject: Farritor et al. (1996), Roston (1994), Rutman (1995), and Sims (1994). In all four papers, the solution approach is based on genetic algorithms (Holland 1975) or genetic programming (Koza 1992).

Farritor et al. (1996) and Rutman (1995) consider the problem of automated design of modular field robots. Similar to the concept of the RMMS, they envision an inventory of robot modules (bodies, joints, wheels, etc.) with which one can build a mobile robot suited for a particular task. To deal with the combinatorial explosion of the size of the search space, they propose a hierarchical selection procedure combined with a genetic algorithm to determine the robot structure. In a second stage, genetic programming is used to generate motion plans built from elementary "action modules" (Cole 1995).

Roston (1994) presents a genetic methodology for configuration design and, as an example, applies this "Genetic Design" methodology to the design of a frame walking robot or stepping stone walker. Instead of building the robot from modules, Roston uses formal grammar to describe and represent candidate designs. The design process itself is based on genetic programming.

Sims (1994) also uses genetic algorithms to create mobile robots or "virtual creatures." these creatures consist of blocks that are connected by rotational joints. The design representation is very general; it accommodates a large variety of structural topologies and also includes a control structure which determines the joint torques applied to the blocks. Both the physical structure and the control structure evolve genetically during the design process. With a simple, yet computationally expensive, fitness function, Sims is able to generate creature with interesting behaviors such as swimming, walking, and jumping.

5.5 An Integrated Solution Approach

Based on the literature review on Task Based Design in the previous section, one can distinguish a spectrum of approaches between two extremes: sequential and fully integrated design. In a first approach, illustrated in Figure 5-3, the design problem is split into subproblems: kinematic design, dynamic design, trajectory planning, and control. Each of these subproblems is tackled individually and sequentially. But because the subproblems are not fully decoupled, an optimal solution at one stage might not be optimal anymore after the next stage. In other words, the globally optimal solution might never be found, because of its possible suboptimality at an intermediate stage. The second extreme, as implemented by Sims (1994), integrates all the design criteria (kinematics, dynamics, planning, and control) into one large global search problem. As a result, the global optimum is never pruned from the search space prematurely, but the search space is so large and the evaluation of all the design criteria is so computationally expensive that finding the global optimum takes a very long time.

We have chosen to take an integrated solution approach to the TBD problem, because the coupling between the different subproblems is significant (especially when considering fault tolerance), and the chance of obtaining sub-optimal or even infeasible solutions is considerable. To make an integrated approach computationally feasible, we have focussed on eliminating the disadvantages of such an approach, namely, a large search space and a computationally expensive evaluation. The two main concepts that we have applied to achieve this goal are to include *problem specific design knowledge* in the search, and to implement the search in an easily *parallelizable agent-based paradigm*.

The basis for our design system is a genetic algorithm. Genetic algorithms were first introduced by Holland in 1975. Holland showed how the evolutionary process of survival of the fittest can be applied to artificial systems, and he provided a strong mathematical framework for analyzing such adaptive systems: *the schemata theory.* Yet, it was not until 1989, when Goldberg (1989) provided a more practical guide to genetic algorithms, that they became widely used. Genetic algorithms are a class of algorithms that transform a set or population of mathematical objects, each with an associated fitness value, into a new population. The transformation process is based on the natural processes of sexual recombination and asexual mutation, combined with the principle of survival of the fittest. Although the original algorithm was formulated for objects represented by a fixed length bit-string, many variations now exist representing individual objects by variable length structures, graphs, and trees.

5.6 Problem Specific Design Knowledge and Genetic Algorithms

In the literature, one can find numerous examples of task specific manipulator or robot design using variations of simulated annealing and genetic algorithms (Chedmail and Ramstein 1996; Chen and Burdick 1995; Farritor et al. 1996; Kim and Khosla 1993; Murthy, Khosla, and Talukdar 1993; Paredis and Khosla 1993; Roston 1994; Sims 1994). Both simulated annealing and genetic algorithms are global search methods that are very robust, but also rather slow.

The performance of these search algorithms can be drastically improved by including problem specific knowledge. This concept was proposed by Michalewicz (1994) who includes problem specific knowledge into genetic algorithms; he calls the new algorithms *evolution programs*. He shows that by using problem specific data structures (instead of binary strings) and problem specific "genetic" operators (generalizations of the generic mutation and cross-over operators), the performance of the search algorithm increases dramatically. The more specific the evolution program, the higher the performance can be, as is illustrated in Figure 5-4. The drawback is that the more specific the algorithm, the narrower the range of problems it can solve. Ideally, one would like to include just enough problem specific knowledge to make the algorithm fast and robust for the class of problems in which one is interested.

There are two major mechanisms through which problem specific knowledge can improve the efficiency of the search algorithm:

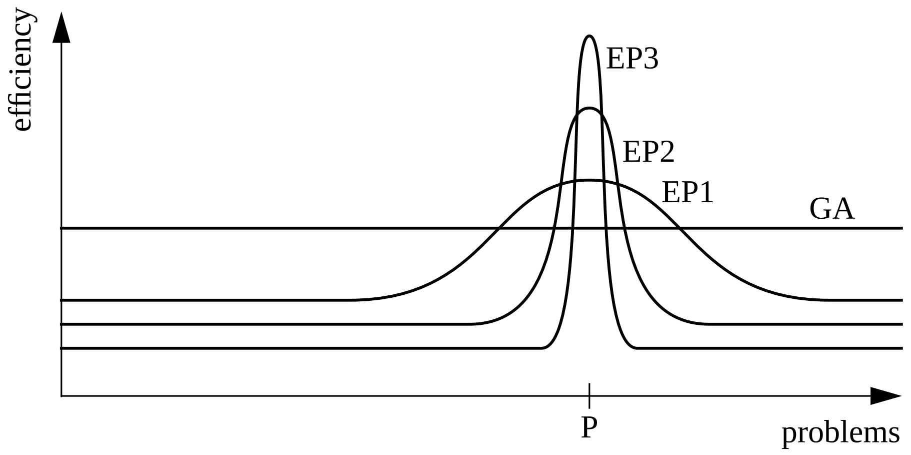


Figure 5-4. Efficiency vs. problem spectrum for Evolution Programs. From Michalewicz (1994).

- reduction of the search space size
- reduction of the evaluation cost of a candidate solution

5.6.1 Search Space Reduction

In the context of TBD, avery important reduction in the size of the search space is obtained by applying the global fault tolerant trajectory planning algorithm, developed in Chapter 4, to the trajectory planning and control subproblem of TBD. Instead of searching the entire joint space (i.e., T^n for an n-DOF revolute manipulator), the fault tolerant trajectory planning algorithm reduces the search space in several steps to a simple connectivity graph of connected regions, $R_j^{t_k}$. First, by taking the manipulator kinematics into account, the search space is reduced to the preimage of the Cartesian path. Second, the postures that do not satisfy the secondary requirements (joint limits, obstacle collision, and self collision) are excluded. Third, the search space is further reduced to the sets of postures that are acceptable for fault tolerant trajectories, and, finally, to a sequence of connected regions. By using a local optimization algorithm to find the shortest trajectory through this set of connected regions, we have totally eliminated the need to *search* for a desired trajectory, and have replaced the search with an algorithmic solution. This amounts to an elimination of $P \cdot n$

continuous design variables, where P is the number of postures in the discretized desired trajectory, and n is the number of DOFs of the manipulator.

The fact that the trajectory planning subproblem can be solved algorithmically does not contradict the fact that there is a strong interaction between it and the structural design problem. For every candidate manipulator configuration, one still needs to plan a *different* desired trajectory; only, this desired trajectory can be determined algorithmically instead of through search.

A second important reduction in search space size, is achieved by including problem specific design knowledge in the generation of candidate design solutions:

- Because base and end-effector modules only have one female or male connector, respectively, they can only be used as the first or the last module of a manipulator configuration. The other modules, with two connectors, can be used at any position except the first and the last.
- We know that at least six DOFs are necessary to complete a task which involves positioning and orienting the end-effector along a specified trajectory (except for some degenerate cases). Therefore, candidate solutions with fewer than six DOFs are immediately rejected—without being evaluated. Similarly, designs with more DOFs than the user-specified maximum are rejected.
- Some of the modules are axially symmetric; it does not matter in which orientation they are mounted with respect to the previous module. Therefore, we arbitrarily set the corresponding relative orientation equal to zero.

The combination of these three constraints result again in a very significant reduction of the search space size. An example illustrates this best. Consider the case of the existing RMMS hardware being used to construct a 3-DOF manipulator. There are seven RMMS modules: a base, an end-effector, a link, and four joint modules. The modules can be connected with each other in eight different relative orientations. The initial search space consists of an ordered list of modules and their relative orientations. The size, N, of the search space equals:

$$N = \sum_{i=1}^{7} \frac{7!}{(7-i)!} 8^i \approx 1.198\times10^{10}. \tag{5-2}$$

After taking into account that an acceptable configuration has to start and end with a base and end-effector module, respectively, the size reduces to:

$$N = 8^2 \sum_{i=0}^{5} \frac{5!}{(5-i)!} 8^i \approx 2.852\times10^{8} \tag{5-3}$$

In the next step, the search space is further limited to configurations with three DOFs:

$$N = 8^2 \binom{4}{3} (4!8^4 + 3!8^3) \approx 2.595\times10^{7} \tag{5-4}$$

Finally, we incorporate the knowledge that the base, the end-effector, the link module, and one of the four joint modules are axially symmetric:

$$N = (4! + 3!)(8^3 + 3 \cdot 8^2) = 2.112\times10^{4}, \tag{5-5}$$

which is six orders of magnitude smaller than the initial size of the search space.

5.6.2 Progressive Evaluation

The second mechanism to improve the efficiency of the search algorithm, is to reduce the cost of evaluating a candidate solution. This is particularly important for our integrated approach to TBD, where simultaneous consideration of all the task specifications can lead to very long evaluation times.

We call the approach we have implemented “progressive evaluation,” because it progressively uses more and more complicated tests to evaluate designs. Very often it is possible to devise a simple test for a *necessary condition*—if a design fails the test one can guarantee that the design does not meet the task requirements. When a design does not pass the test for a necessary condition, an estimate for the fitness of the design is generated and the evaluation is terminated. This approach is based on the observation that it is very often possible to recognize a bad design quickly using a simple test. Instead of completing the full evaluation for such a design, it is sufficient to make a rough estimate of the fitness and interrupt the evaluation to reduce the computation time. Here is a successive list of tests that are implemented in the TBD evaluation function:

- If fault tolerance is required, test whether the manipulator is redundant.
- Test whether the manipulator can reach the initial point of the Cartesian path.
- If the manipulator is redundant, test whether a desired joint space trajectory exists.
- The trajectory planning algorithm itself is also implemented in a progressive manner: the computations stop as soon as a Cartesian point is reached for which no more acceptable postures can be found.
- Only when all these tests are satisfied does the simulation of the task execution start.

5.7 An Agent-Based Design Framework

An approach to TBD that integrates kinematics, dynamics, planning, and control, has the disadvantage that the search for a globally optimal solution can take a long time. In the previous section, we addressed this problem by including problem specific design knowledge to reduce the size of the search space and to reduce the computation cost for evaluating a candidate design. In this section, we take a different but complementary approach to speed up the search, namely, by increasing the computational resources.

The computational resources can be increased by developing an agent-based implementation which runs on a distributed network of workstations. When Sims (1994) evolved creatures using an integrated approach, he resorted to a 32 processor CM5 to overcome the problem of computational power. He implemented a modified genetic algorithm to run in parallel according to a master/slave message passing model. The approach we have taken, is to modify the genetic algorithm according to a multi-agent paradigm. The resulting algorithm can be executed in a distributed fashion on any MIMD computer, including shared memory multi-processors and, in particular, networks of computers used for concurrent computation.

Due to the centralized control of the standard genetic algorithm, it is not well suited for distributed implementation. For every new generation in a genetic algorithm, one needs to modify and evaluate each of the individuals. These modification and evaluation operations can be parallelized. However, a straightforward parallel implementation of the standard genetic algorithm would cause load balancing problems and inefficient processor usage due to the synchronization requirements of the centrally controlled algorithm—selection of the parents for the next generation of individuals depends on the *normalized* fitness and therefore cannot be executed until *all* the individuals have been evaluated. In the current literature on parallel genetic algorithms, one can find three main adaptations of the standard genetic algorithm that avoid this centralized control. In a first approach, new parents are selected through “tournament selection”; this eliminates the need for normalization of the fitness value (Goldberg, Deb, and Korb 1991). A second approach is to run many standard genetic algorithms in parallel (one sub-population per processor), and to have individuals migrate from one sub-population to another every so often (Tanese 1989). In this way, the synchronization problem is not totally eliminated, but it is reduced significantly. A last approach is to have one single population in which each individual has a dedicated processor (Mühlenbein 1992). The individuals select a mating partner not from the global population, but only from their immediate neighbors. This happens asynchronously without the intervention of a centralized controller.

Synchronization problems can also be avoided through an agent-based implementation, where each agent corresponds to an *operator* rather than an individual. This approach, illustrated in Figure 5-5, combines four types of agents: creation agents, modification agents, evaluation agents, and destroyer agents. The agents do not communicate with each other directly, but only indirectly through a shared memory which contains the population of candidate solutions. A population manager manages the storage and retrieval of individuals of the population, but does not control the execution of the agents. The agents are fully autono-

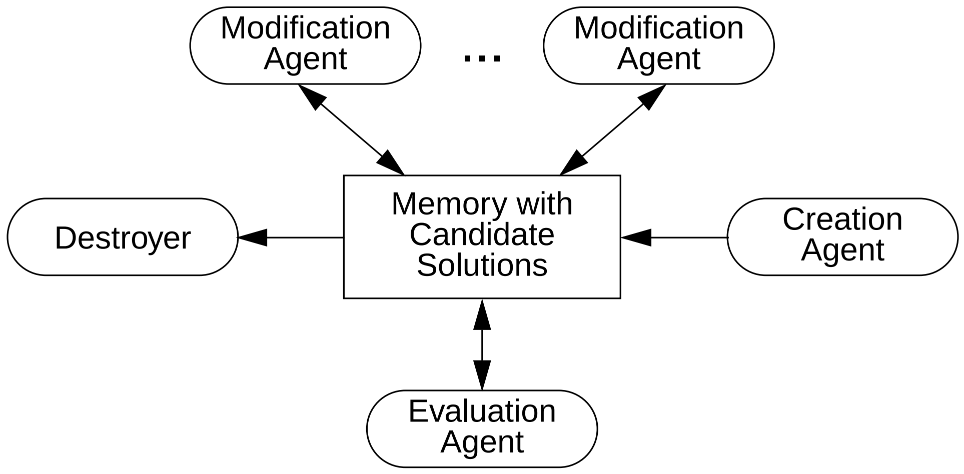


Figure 5-5. Layout of a generic agent-based design framework.

mous and execute asynchronously; they decide when to act, and which individual to act on, without being controlled by a centralized controller. Their functions correspond roughly to the functional entities of the genetic algorithm. At start-up, a *creation agent* generates initial candidate solutions and places them in the population (creation of the initial population). One or more *evaluation agents* then retrieve the unevaluated solutions, evaluate them and return them to the population (determination of the fitness values). Based on the evaluations, *modification agents* select and modify candidate solutions and return them to the population (selection, mutation, and cross-over in genetic algorithms). One major difference between this agent-based approach and the standard genetic algorithm is the way the population is managed. Rather than creating a new population each generation based on the population in the previous generation, the algorithm does not consider distinct generations at all. Instead, the off-spring is simply *added* to the current population. To avoid an ever growing population, individuals with a low fitness value are destroyed by *destroyer agents;* this is similar to the mechanism used in $(\mu + \lambda)$ -Evolution Strategies described by Schwefel (1981). Theoretical models of genetic algorithms show the importance of selection as a converging force within a population (Holland 1975). In genetic algorithms, convergence is achieved by preferably selecting the best individuals as parents for the next generation. A similar converging force is achieved in our approach by preferably destroying the worst individuals. As a result, the framework, shown in Figure 5-5, maintains the desired convergence characteristics of genetic algorithms, while, at the same time, avoiding the need for centralized control—a critical criterion for a distributed implementation.

The main advantage of an agent-based framework is *performance*. Current day computing facilities consist of a highly networked group of powerful workstations. Many of these workstations are idle for large amounts of time. These idle cycles can be used at no extra cost to run the agents in our distributed framework. Since we are dealing with a small data structure to represent candidate solutions, the time needed to exchange data between the agents and the population manager is negligible compared to the computing time required for modification and evaluation of candidate solutions. The bottleneck of our implementation is thus evaluation and modification and not communication with the shared resource. Therefore, one can expect a speedup which is almost linear in the number of processors. This is confirmed by a simple test of our current implementation which revealed that congestion of the population manager starts to occur around 20 solution insertions per second. The average insertion rate for the example in Chapter 7 is approximately one insertion every two seconds—far below the congestion limit.

An additional advantage of this agent-based design framework is its *modularity.* We have divided the monolithic standard genetic algorithm into functional entities that, by removing the centralized control, have become independent modules or autonomous agents. These agents are separate processes that interact only with the population manager, so that the addition of new agents does not affect the other agents. Modularity makes the design system also reconfigurable or *composable*. For instance, in the examples in Chapter 6, the base position is fixed. Therefore, one can compose a design system without the MutateBasePosition agent. In Chapter 7, on the other hand, we do consider the base position as a design variable so that the MutateBasePosition agent is included. This can be achieved without having to change any code.

5.8 Implementation

5.8.1 The Asynchronous Team Toolkit

The agent-based design framework described in the previous section has been implemented using the Asynchronous Team Toolkit developed by Talukdar et al. (1996). This toolkit provides object oriented support software to implement a team of asynchronous autonomous agents (Talukdar, deSouza, and Murthy 1993). At a low level, the toolkit provides utilities to write and read candidate solutions to and from a shared memory managed by a population managing process. Furthermore, the toolkit contains a query mechanism to search and select for specific candidate solutions that are stored in the population. At an intermediate level, agent templates aide the user in creating agents for a specific search problem. There are generic templates for destroyer, creation, modification, and evaluation agents. At the highest level, the A-teams toolkit provides a graphical user interface, illustrated in Figure 5-6, with which one can first connect agents and memories into a team and then spawn and monitor them on a network of computers.

To support a distributed implementation across a wide range of platforms, the toolkit is built on top of PVM (Geist et al. 1994). PVM (Parallel Virtual Machine) is a software package that enables the computer user to define a networked heterogeneous collection of serial, parallel, and vector computers to function as one large computer. It can be used as stand-alone software or as a foundation for other heterogeneous network software such as the A-teams toolkit.

The software agents that constitute our design framework all have the same internal structure. They consist of a scheduler, a searcher, a selector, and an operator.

The scheduler. The scheduler decides when the agent should be active. For instance, a creation agent or seeder might only be active at start-up to create the initial population. We have implemented a relatively simple scheduler with which one can limit the number of times the agent executes (especially useful for creation agents), and with which one can specify the time an agents sleeps in-between executions, both under normal circumstances and when an execution error occurs.

The searcher. The searcher determines which individuals from the population are eligible to be acted upon. For instance, the searcher of an evaluation agent will query the population manager for candidate solutions that have not yet been evaluated. Other agents can have more sophisticated searchers. Assume, for instance, that a modification operation is only useful for manipulator designs that do not meet the kinematic constraints, then the searcher

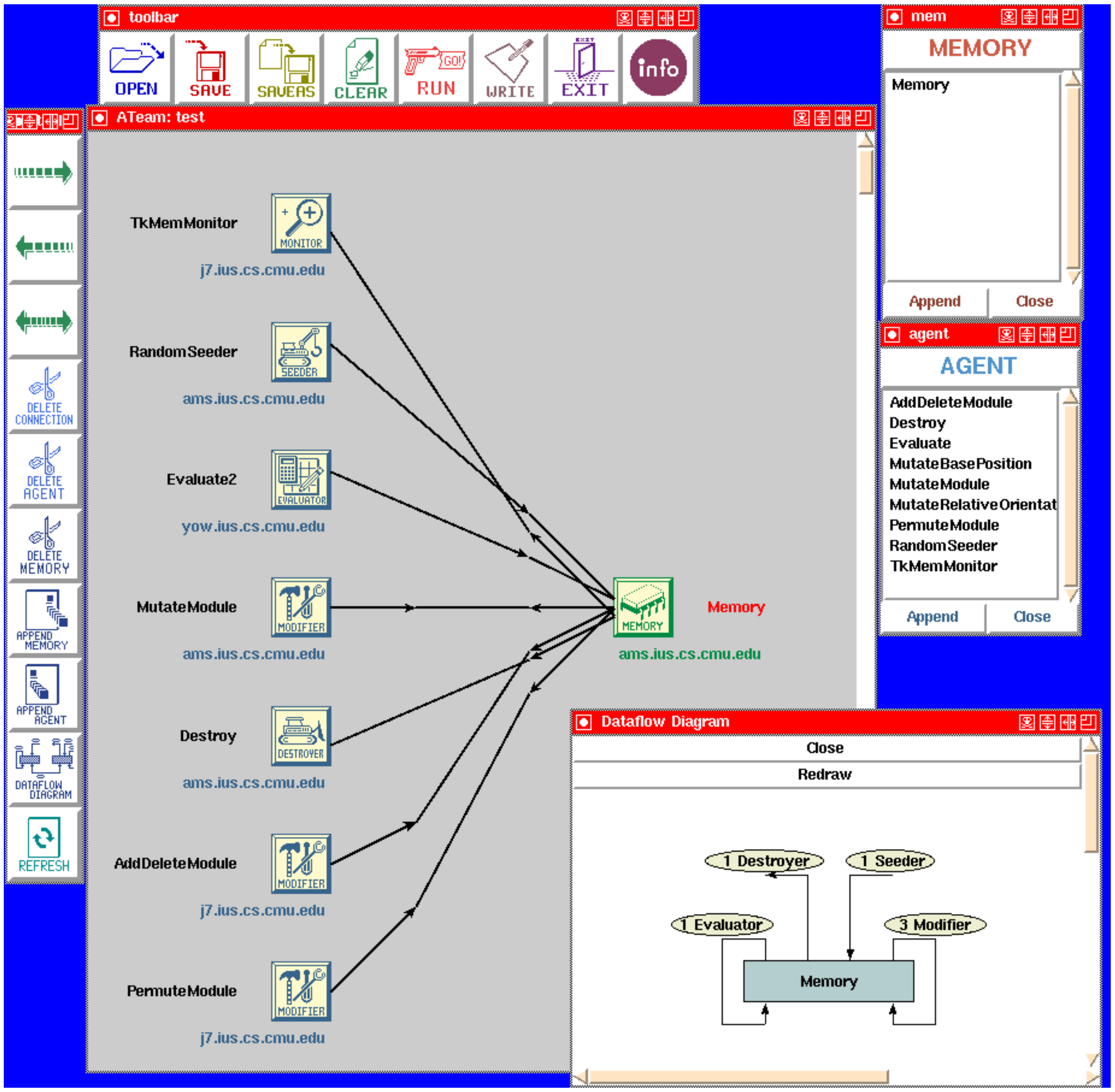


Figure 5-6. An overview of the Ateams toolkit GUI.

can query the population manager for designs that contain a non-zero kinematic penalty. The population manager replies to agent queries by sending them a list of designs that match the query.

The selector. Since most agents act on only one or two solutions at a time, they have to make a selection from the search list—the list of solutions returned by the population manager upon a search query. This is performed by the selector. A mutation agent, for instance, might perform tournament selection, that is, it compares the fitness value for two or more individuals randomly chosen from the search list and it retains the fittest. If the best individual from k randomly chosen individuals is retained, then the probability that the individual with rank i is selected from a population of n individuals ranked by fitness from 1 (best) to n (worst) equals

$$P(\text{individual } i \text{ is selected}) = \frac{k(n-i+1)^{(k-1)}}{n^k}. \tag{5-6}$$

As the tournament size k increases, the selective pressure increases too. All the agents that we have implemented so far use tournament selection with $k = 2$, which corresponds to a linear probability distribution.

The operator. Once the agent has selected one or more individuals, the actual operation is performed—creation, modification, destruction, or evaluation. The operator functions are explained in more detail in the next two sections.

The internal structure of agents outlined above is general enough to accommodate a wide variety of operators, ranging from the standard genetic operators to operators that contain explicit knowledge about the design problem.

5.8.2 The Evaluation Function

The evaluation function is the function that determines the fitness value for a given candidate design. In the case of TBD, the fitness value is a combination of the extent to which the constraints are violated and the extent to which the optimality criterion is achieved. Specifically, the constraints include:

- **Reachability:** is the manipulator able to follow the specified trajectory?
- **Joint limits:** do the joint angles remain within their limits?
- **Joint velocities:** do the joint velocities remain within their limits?
- **Singularity avoidance:** does the manipulator stay away from singular configurations?
- **Torque limits:** do the joint torques remain within the motor limits?
- **Collisions with obstacles:** does any part of the manipulator collide with the specified obstacles?
- **Self-collision:** does any part of the manipulator collide with any other part of the manipulator?

The optimality criterion that we consider is:

- **Energy consumption**.

The evaluation process occurs in two stages. In a first stage, the *desired behavior* of the manipulator is determined. As we explained in Section 5.2.4, the behavior of the manipulator is determined by the trajectory planner and the controller. For the RMMS, the controller is the damped least-squares controller defined by:

$$\dot{\theta}_{ref} = J^*(\dot{p} + \alpha(p - x)) + \beta(I - J^*J)(\theta_{des} - \theta_{ref}) \tag{5-7}$$

This controller completely determines the behavior of the manipulator when given the initial posture, θ_{ref} at time zero, the desired trajectory, $\theta_{des}(t)$, and the parameters α and β. The parameters α and β are constant; they are set by the user in the problem definition file. The desired trajectory, $\theta_{des}(t)$, is determined using the global trajectory planning algorithm described in Chapter 4. The trajectory planner guarantees that the specified Cartesian path can be followed even when one of the joints fails and is immobilized, i.e., fault tolerantly. Secondary requirements (joint limits, obstacle collision, and self-collision) are also taken into account. With a small change, the same algorithm can also be used to plan *non*-fault tolerant trajectories that satisfy the secondary requirements. As a result, the behavior of any redundant manipulator can be determined algorithmically using the global trajectory planning algorithm of Chapter 4.

The evaluation function including the global fault tolerant trajectory planning algorithm is executed many times and for a large variety of different manipulators, some of which may be degenerate (all axes are parallel, axes coincide etc.). This requires that the global fault tolerant trajectory planning algorithm be *fast* and very *robust*. We have not yet been able to achieve both of these attributes simultaneously for manipulators with more than one degree-of-redundancy. Therefore, we will limit ourselves to the design of fault tolerant manipulators with at most one degree-of-redundancy.

For non-redundant manipulators, the null-space of the Jacobian is empty and the second term of Equation (5-7) reduced to zero. The behavior is independent of the desired trajectory, $\theta_{des}(t)$, and fully determined by the initial posture, $\theta_{ref}(0)$, which is computed by solving the inverse kinematics for the initial point of the Cartesian path.

If no acceptable behavior can be found, that is, no global trajectory can be planned for redundant manipulators or no initial posture can be found for non-redundant manipulators, the evaluation routine returns a fitness value that reflects how close the first stage came to being successful. For non-redundant manipulators this fitness value is based on how close in Cartesian space the manipulator was able to approach the initial point of the Cartesian path; for redundant manipulators, the fitness value is based on the total size of the sets of acceptable postures.

In the second stage of the evaluation, the task execution is simulated with the given manipulator configuration and the behavior determined in stage one. The simulation engine uses the same control structure as is used for the actual RMMS hardware. The only difference being that the PID controller is eliminate and the feed forward torque, τ_{ff} in Figure 5-2, is considered to be the actual torque required to accurately follow the trajectory $\theta_{ref}(t)$. For every time step of the simulation ($\Delta t = 0.01\mathrm{s}$), the evaluation routine checks whether any constraints have been violated. For every constraint, it keeps track of the number of time steps during which a violation occurs. At the end, a fitness value is computed based on the total number of constraint violations.

The joint velocity and singularity avoidance constraints are not checked explicitly. They are implicitly included in the reachability constraint. Thanks to the singularity robust Jacobian (Nakamura and Hanafusa 1986) in the damped least-squares kinematic controller, the controller remains stable when approaching a singularity, but will deviate from the desired Cartesian end-effector path. Moreover, the kinematic controller limits the joint velocities (through clipping), before the velocities are integrated to obtain the joint trajectory, $\theta_{\text{ref}}(t)$. Therefore, both joint velocity and singularity avoidance violations result in a deviation from the desired Cartesian path and are thus implicitly included in the reachability constraint.

To check for obstacle collision and self-collision constraint violations, we have included a public domain software package called RAPID (Gottschalk, Lin, and Manocha 1996). RAPID is a *Rapid and Accurate Polygon Interference Detection* library for large environments composed of unstructured models. The models of the obstacles and the manipulator may consist of sets of triangles without adjacency information and without any topological constraints (the models may contain cracks, holes, self-intersections, degenerate polygons, and non-generic configurations). Collision detection is performed hierarchically using Oriented Bounding Box trees (OBB-trees). These trees are pre-computed for each of the RMMS modules during the problem initialization, so that there is no computation overhead during the evaluation of a specific manipulator configuration. An example of a manipulator model is shown in Figure 5-7. Notice that the models for the RMMS modules have been defined a little larger than the actual modules, so that one can guarantee collision free task execution without having to model every little detail of the modules.

The evaluation routine also integrates the total energy being consumed by the manipulator. The energy consumption consists of three parts: energy consumed by the electronics, by the amplifier and brake, and by the motors. Therefore, link modules consume less energy than joint modules, even when the joint is not generating any torque. According to our measurements on the actual RMMS hardware, the electronics dissipate approximately 16.2W and

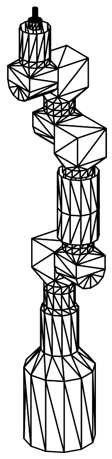


Figure 5-7. A CAD-model of the manipulator used for collision detection

the amplifier and brake dissipate 15.8W. The power required by one of the motors is time dependent and equals

$$P_{motor} = Ri^2(t) + \tau(t)\dot{\theta}(t) = \frac{R}{K^2}\tau^2(t) + \tau(t)\dot{\theta}(t) , \tag{5-8}$$

where K is the torque constant of the motor, $\dot{\theta}$ is the joint velocity and, τ is the joint torque as computed by the computed torque algorithm. For a manipulator with n DOFs and m modules, the total power consumption equals

$$P_{tot} = m16.2 + n15.8 + \sum_{j=1}^{n} \frac{R}{K^2}\tau_j^2(t) + \tau_j(t)\dot{\theta}_j(t) \tag{5-9}$$

Note that Equation (5-9) remains valid even when the motor power becomes negative, because the PWM amplifiers regenerate the mechanical power and redistribute it over the other motors and brakes.

The total fitness value for a manipulator design can range from -200,000 to zero. This value is used by the selection operators of the agents to select and individual design from the current population. All the agents that we have implemented use tournament selection. Thus, an individual is selected based on its fitness value *relative* to another individual's fitness value—the *absolute* value of the fitness function is irrelevant. This gives us the freedom to design the evaluation function in such a way that the fitness value indicates at which stage of the progressive evaluation procedure the evaluation was terminated:

- $f = -200{,}000$: insufficient number of DOFs to achieve fault tolerance.
- $-200{,}000 < f < -100{,}000$: the design is unable to reach the initial point of the Cartesian path.
- $-100{,}000 \leq f < -80{,}000$: the fault tolerant trajectory planning algorithm failed to return an acceptable solution.
- $-80{,}000 \leq f < -20{,}000$: certain task requirements were violated during the simulation.
- $-20{,}000 \leq f$: no violations; the fitness value equals minus the power consumption.

5.8.3 The Modification Operators

For the TBD problem, we have defined six different modification operators: five mutation operators and one cross-over operator. Each of the five mutation operators modifies a candidate solution in a different way:

- **MutateBasePosition** makes a small random change to the base position and orientation.
- **MutateRelativeOrientation** randomly selects a module and changes its relative orientation with respect to the previous module. The change in relative orientation is at most 90 degrees.
- **MutateModule** randomly selects a module and replaces it by a different module retrieved from the inventory.
- **PermuteModule** swaps two randomly chosen modules both from the current manipulator configuration.
- **AddDeleteModule** adds or deletes a module at a randomly selected position in the manipulator configuration.

In Section 5.6, we formulated three constraints that have to be satisfied by a candidate solution: start and end with a base and end-effector module respectively, stay within the specified range of DOFs, and have a relative orientation of zero for axially symmetric modules. The mutation operators have been implemented in such a way that these constraints are always satisfied; if they cannot find a candidate solution satisfying these constraints, they return an error-code.

For the cross-over operator, one has to consider the additional constraint that each RMMS module can only appear once in a legal candidate solution. Direct application of the standard genetic cross-over operator could result in illegal solutions. For instance, consider the following two sequences of module numbers, with a cross-over point after the second module:

$$\langle 0, 7, \mid 3, 6, 2, 8 \rangle$$
$$\langle 0, 2, \mid 1, 9, 7, 8 \rangle$$

The off-spring resulting from these two parents are:

$$\langle 0, 7, \mid 1, 9, 7, 8 \rangle$$
$$\langle 0, 2, \mid 3, 6, 2, 8 \rangle$$

Both children are illegal because they contain a duplicates of the same module. Moreover, there is no guarantee that the children satisfy the imposed constraint on the number of DOFs. To overcome these problems we propose a variation of OX (order cross-over) as introduced by Davis (1985) for the traveling salesman problem. The idea is to maintain the *order* of the modules, omitting the ones already in use. The algorithm is illustrated with the same two parents used above.

1. Select a random cross-over point in the first parent: $\langle 0, 7, \mid 3, 6, 2, 8 \rangle$
2. Copy the first part of the parent to the child: $\langle 0, 7, \mid x, x, x, x \rangle$
3. Remove the modules that are already in use from the second parent: $\langle 2, 1, 9, 8 \rangle$
4. Select a random cross-over point in this reduced parent: $\langle 2, \mid 1, 9, 8 \rangle$
5. Complete the child, and check whether it satisfies the conditions for legal candidate solutions (Section 5.6); if not, go back to step 4: $\langle 0, 7, \mid 1, 9, 8 \rangle$

Note that, if both parents are legal, there will always be enough DOFs left in the reduced parent to create a legal child; only modules that are already part of the child are deleted from the second parent, so that in the worst case all the DOFs of parent two are needed to complete a legal child.

5.8.4 The Agents

When composing an agent-based design system for a particular design task, one needs to decide which agents one should use (based on the problem statement) and how many instances of each agent one should spawn. The number of agents in a design system depends on the available computing facilities; one would like to spawn at least one agent on each available processor. For TBD, the fitness evaluation is the computational bottleneck, so that it is important to distribute the evaluation effort as evenly as possible over the computational resources. The design modification operators, on the other hand, perform only very little computation. Yet, every design modification needs to be followed by an evaluation. Therefore, there needs to be a well balanced proportion between the number of evaluation and modification operators. To avoid having to fine tune this proportion for every new design task (the fitness evaluation time varies from one task to another), we decided to combine the modification and evaluation operators into one single agent. Similarly, creation operators are paired with evaluators in one agent. The resulting design system has a very simple topology, shown in Figure 5-8, and consists of only three types of agents: creation/evaluation agents, modification/evaluation agents, and a destroyer agent. Internally, these agents are structured as follows:

Creation/Evaluation agents:

- *scheduler:* the agent is active until the number of individuals in the population reaches the desired population size.
- *operator:* the agent contains a creation operator that generates a (legal) random candidate solution, and an evaluation operator that evaluates the solution before it is stored in the population.
- a creation agent does not have a *searcher* and *selector* because it does not

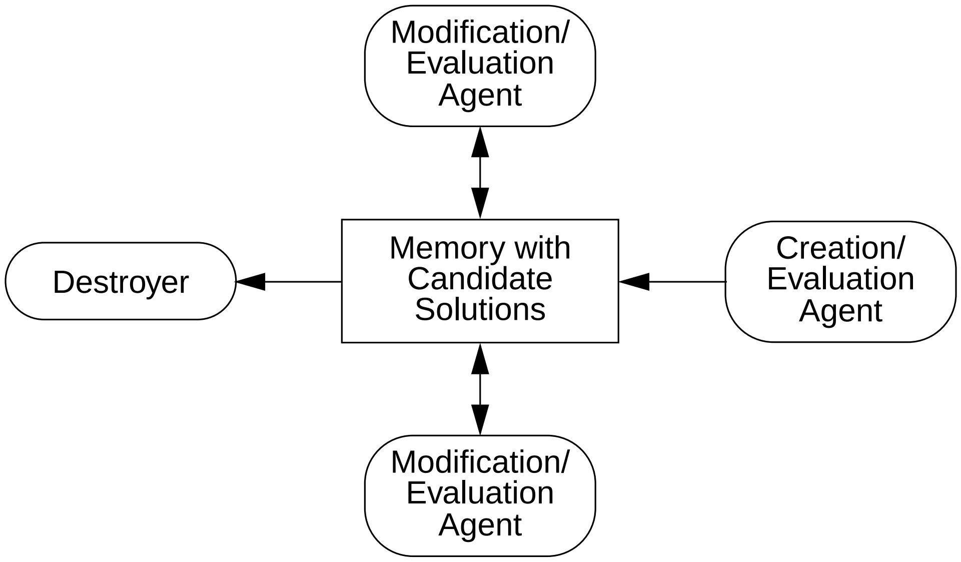

Modification/ Evaluation Agent
Destroyer
Memory with Candidate Solutions
Creation/ Evaluation Agent
Modification/ Evaluation Agent

Figure 5-8. Topology of the agent-based design system.

retrieve candidate solutions from the population.

Modification/Evaluation agents:

- *scheduler:* the modification agents are always active. However, if an execution error occurs, for instance, due to the fact that the search query returns an empty list, the agent sleeps between zero and ten seconds. This is to avoid that the agents submerge the population manager with search queries when no “interesting” candidate solutions are found in the population. The sleep time is randomly chosen between zero and ten seconds with a uniform probability distribution. A randomly chosen sleep time is used to avoid any periodicity in the schedule.
- *searcher:* the modification agents search for evaluated candidate solutions. However, if the number of evaluated candidate solutions in the population is smaller than desired population size, the searcher returns an error, causing the agent to sleep for an average of five seconds.
- *selector:* from the list of candidate solution returned by the population manager, the selector picks one candidate solution (two for cross-over) using tournament selection: it randomly picks two candidate solution from the list, compares their

fitness values and retains the fittest candidate.

- *operator:* for each of the six modification operators described in the previous section (five mutation operators and one cross-over operator), there exists a separate agent. The agent performs the mutation or cross-over on the selected candidate solution(s) and evaluates the new solution before returning it to the population.

Destroyer:

- *scheduler:* same scheduler as modification/evaluation agents.
- *searcher:* the destroyer agent also searches for evaluated solutions, while requiring that the number of evaluated solutions be larger than or equal to the desired population size.
- *selector:* the destroyer agent also uses tournament selection. However, the tournament is set up such that the worst individual wins. As a result, the destroyer preferably destroys bad solutions.
- *operator:* the operation performed by a destroyer is to remove the selected candidate solution from the population.

There is one additional agent, namely a display agent. Periodically, this agent retrieves the best solution so far from the population and stores it in a file. Furthermore, it prints some simple statistics about the current population (minimum, maximum, and average fitness value) together with a time stamp, so that the user can examine the state of the design process and decide whether an acceptable solution has been reached and whether the design process can be terminated.

5.8.5 Emergent Behavior

Based on the behaviors of the individual agents, as specified above, one can distinguish two phases in the emergent group behavior. At start-up, the population manager is spawned first, so that it is ready to respond to the agents's queries once they have been created. Once the population manager is running, we start the creation/evaluation agents to seed the initial population (there is only one type of creation agent of which we spawn multiple copies).

Due to a limitation of the current implementation, the agents have to be spawned sequentially. As a result, the rate at which new candidate designs are introduced into the population is small in the beginning and increases slowly as more creation agents are spawned. While the creation agents are filling up the population, the destroyer agent and all the modification agents are spawned. During this initialization phase, the modification agents remain mostly dormant; they wait for the number of candidate designs in the population to reach the desired population size. Every five seconds they inquire about the current number of designs in the population. When the desired population size is reached, a transition occurs. The modification agents wake up and start working non-stop, while the creation agents, having completed their job, exit. The destroyer agent also becomes active and periodically removes the excess candidate designs from the population. In this second phase of operation, the quality of the designs in the population gradually improves, because the modification agents select preferably the highest ranked candidate solutions, while the destroyer agent deletes preferably the lowest ranked individuals.

5.9 Summary

This chapter presented a framework for solving the Task Based Design problem. This framework is the linchpin of the rapidly deployable fault tolerant manipulator system. It ties together all the building blocks: reconfigurable hardware, control software, fault tolerance, and trajectory planning.

TBD is a very complicated problem because (1) there is only very little explicit knowledge available about structural design of task specific manipulators, (2) the design space consisting of modular manipulator configurations is very large, and (3) the relation between the design variables and the constraints and optimality criteria is highly coupled and nonlinear.

Unlike most previous TBD approaches, we presented an integrated approach to the TBD problem to deal with the tight coupling between the design subproblems caused by the fault tolerance requirement. However, this resulted in a very challenging, computationally expensive problem. We proposed two complementary approaches to cope with this challenge: reduce the computational complexity by including problem specific knowledge, and increase the computational resources through a distributed implementation.

The agent-based design framework that is the main focus of this chapter combines these two approaches. It is based on genetic algorithms, but avoids the central control of the standard genetic algorithm by implementing the modification and evaluation operations as autonomous asynchronous agents. These agents include problem specific knowledge which results in a significant reduction of the size of the search space and a reduction of the cost of evaluating a candidate design solution. Furthermore, thanks to their autonomous and asynchronous nature, these agents can be easily executed on a network of workstations, drastically increasing the computation power available to solve the TBD problem.

In conclusion, the flexibility and performance of the agent-based design implementation combined with the problem specific knowledge included in the modifiers and evaluators results in a powerful new approach to the task based design of rapidly deployable fault tolerant manipulators. In the next chapters, we will test the performance of this agent based design system, first with two relatively simple tasks, and, in Chapter 7, with a very challenging comprehensive task for a satellite docking operation aboard the space shuttle.

Chapter 6

Analysis of the Task Based Design Problem

6.1 Introduction

The goal of this chapter is threefold: (1) to provide insight into the nature of the TBD problem; (2) to characterize the complexity of the TBD problem; (3) to compare the performance of the agent based design framework with such benchmark algorithms as random search and multiple restart statistical hill-climbing.

These goals are achieved through an empirical approach. Two different examples are considered: a non-redundant non-fault-tolerant task, and a redundant fault tolerant task. For both examples, a complete analysis is performed based on an exhaustive search, random search, multiple restart statistical hill-climbing, and the agent-based design framework presented in the previous chapter.

Whether a solution to a specific TBD problem can be found depends both on the characteristics of the problem and on the power of the algorithm. Both of these aspects are investigated for the two examples in this chapter. In this respect, the exhaustive search offers some insights into the nature of the TBD problem, and it provides the baseline against which the statistical search algorithms can be compared.

6.2 Set-up of the Experiments

The analysis of the TBD problem is based on the results from two experiments. The problem definitions, algorithms, and facilities that are used for these experiments are described in this section.

6.2.1 Test Problems

In this chapter, we consider two problems with the following characteristics:

- The desired Cartesian path traces the letters “RMMS” on a white board (the same trajectory as is used in Chapter 2).
- The module inventory consists of the seven RMMS modules that have been physically realized so far: one base module, one end-effector module (a marker and holder), one link module, three pivot joint modules, and one rotate joint module.
- Only end-point position is considered—not orientation.
- In addition to self-collisions, collisions with the white board should be avoided.
- Three DOFs for example one; four DOFs for example two.
- Non-fault tolerant for example one; fault tolerant for example two.

We have chosen to limit the inventory to the seven RMMS modules, so that design solutions can be tested and demonstrated with the existing hardware. Moreover, the search space for an inventory of seven modules is relatively small, which allows us to perform an exhaustive search. Even though exhaustive search is unacceptably slow for a practical solution method, it is important for the performance analysis because it gives us ground truth about the complexity of the problem and about the absolute quality of the solutions found by statistical search algorithms.

6.2.2 Search Algorithms

Besides exhaustive search, three statistical search algorithms are used to solve the TBD problems defined above: random search, multiple restart statistical hill-climbing, and the agent-based genetic algorithm developed in the previous chapter.

In random search, the simplest of the three statistical search algorithms, candidate solutions are selected at random, while keeping track of the best solution so far. Random search is of little practical importance as a solution method because it is too slow. However, for problems for which an exhaustive search is impossible due to the size of the search space, random search can give an indication of the complexity of the problem. Indeed, an important problem characteristic is the fraction of acceptably good solutions in the search space; this fraction can be approximated through random search.

Multiple restart statistical hill-climbing (MRSH) has been suggested in the literature as an adequate baseline method for evaluating the performance of genetic algorithms (Juels and Wattenberg 1994; Baluja 1995). The method consists of a statistical hill-climbing algorithm that is started multiple times at randomly chosen initial points in the search space. In each of the statistical hill-climbing runs one keeps track of the best solution so far. This solution is randomly perturbed to create new candidate solutions that are only accepted when they are better than the current best solution—similar to simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983) but without the provision for occasional descent to avoid local maxima. Since statistical hill-climbing cannot avoid local maxima, an individual run might get stuck relatively quickly. Both Juels and Wattenberg (1994) and Baluja (1995) indicate that the performance of statistical hill-climbing improves dramatically when multiple starting points are chosen and the length of each run is reduced accordingly. At the end, one compares the best solutions found by the individual statistical hill-climbing runs, retaining the overall best one as the solution for the MRSH algorithm.

It is important to notice that, whether MRSH is successful at solving the TBD problem, depends strongly on the choice of the perturbation operations and on the choice of the fitness heuristic. In the case of TBD, the perturbations are generated through the five mutation operators defined in Section 5.8.3 of Chapter 5. Because both the mutation operators and the fitness evaluation function contain a significant amount of problem specific knowledge, one can expect the simple MRSH algorithm to perform relatively well. Note also that in the current implementation the multiple runs of the statistical hill-climbing algorithm are performed in parallel, rather than sequentially. Each of the 24 Sparc workstations in the experiment runs one instance of the statistical hill-climbing algorithm. At one second intervals, the total number of evaluations and the best solutions so far from each of the runs are recorded.

The third statistical search algorithm is the agent-based genetic algorithm described in the previous chapter. The algorithm used in the experiments consists of 32 agents and one population manager. Since fitness evaluation is the most computationally expensive aspect of the genetic algorithm, each of the following 24 modification/evaluation agents runs on a separate workstation (remember that modification agents also evaluate the newly created candidate designs before returning them to the population):

- 12 copies of the MutateRelativeOrientation agents
- 3 copies of the MutateModule agents
- 3 copies of the PermuteModule agents
- 3 copies of the AddDeleteModule agents
- 3 copies of the CrossOver agents

The choice of a large number of MutateRelativeOrientation agents is based on the analysis of the exhaustive search in Sections 6.4.2 and 6.5.2, which indicates the importance of changing the orientation of the modules once a promising module order has been determined.

There are six creation agents. They are only active at start-up so that they can share a workstation with one of the modification agents without contending for the same CPU-time. In addition to the 24 workstations used for the modification agents, one extra workstation hosts the memory manager, the destroyer agent, and a displayer agent that outputs the best solution so far and some simple statistics about the current population of candidate solutions.

The only algorithm parameter that we varied throughout the experiments is the population size. The population size determines the selective pressure for the agent-based genetic algorithm. At the extremes of the population size, the agent-based genetic algorithm behaves similarly to other algorithms considered. For very large populations, the algorithm behaves almost like a random search. It takes a long time to seed the population and much longer for the population to converge to a solution. As a result, the genetic algorithm with a large population size is very robust but also very slow. On the other hand, a genetic algorithm with a population of only one individual reduces to a single start statistical hill-climbing algorithm; it improves quite rapidly but is very likely to get stuck in a local maximum. Depending on the problem being solved the optimal size of the population may vary. We have not explored this dependency in detail, but have chosen conservatively high estimates to achieve robustness rather than fast convergence.

6.2.3 Computing Resources

One of the characteristics that complicates the TBD problem is that evaluating the fitness heuristic is very computationally expensive. To complete the computations within a reasonable amount of time, we used parallel implementations for all three search methods and executed them on 24 networked Sun Sparc workstations (ten Sparc 5/110 workstations, eight Sparc 20/61 workstations, and six Sparc 5/85 workstations). The combined computing power of these 24 workstations is 548 Mflops (Mega floating point operations per second).

6.3 Analysis Tools

The data from the experiments performed in the next two sections provide information about two different aspects of task based design: it defines the performance of the solution approaches but it also characterizes the problems themselves. This section describes the criteria that are used for this analysis.

6.3.1 Problem Characterization

The statistical search algorithms presented in the previous section not only provide a means to solve the TBD problem, but they also are a source of information about the character of the problem itself. In the previous chapter, we transformed the TBD problem into a search problem with a heuristic fitness function to guide the search towards the best designs. Section 5.3 lists several characteristics of the TBD problem from an engineering design perspective. However, after having transformed the design task into a search problem, one should use additional measures that better characterize the complexity of the problem, namely:

- Computation cost of the fitness evaluation
- Dimensionality of the search space
- Fraction of acceptably good designs in the search space
- Region of attraction of these designs

By comparing these characteristics for different TBD problems, it is possible to estimate the relative problem complexity which in turn can be used to estimate the required computation time.

Computation cost of the fitness evaluation. It is clear that problems become more complex as the evaluation cost of the fitness function increases. Within a given time interval, one is able to perform fewer fitness evaluations when the computation cost is high, which reduces the chance that a good solution will be found.

In the case of TBD, the issue is a bit more complicated because the cost for evaluating the fitness function is not constant due to the *progressive evaluation* of the fitness heuristic (weed out the bad designs quickly by testing for simple necessary conditions, while spending more time distinguishing subtle difference in good designs only). The effect of progressive evaluation on the average computation cost for fitness evaluation is illustrated in Figure 6-1. For random search, the average time required for a fitness evaluation is constant. At any time, the population from which a design is selected is the total population of designs consisting mainly of relatively bad designs which require little time to evaluate. Initially, multiple restart statistical hill-climbing also starts by selecting from the total population of designs. However, the algorithm then limits itself to the population of direct "neighbors" of

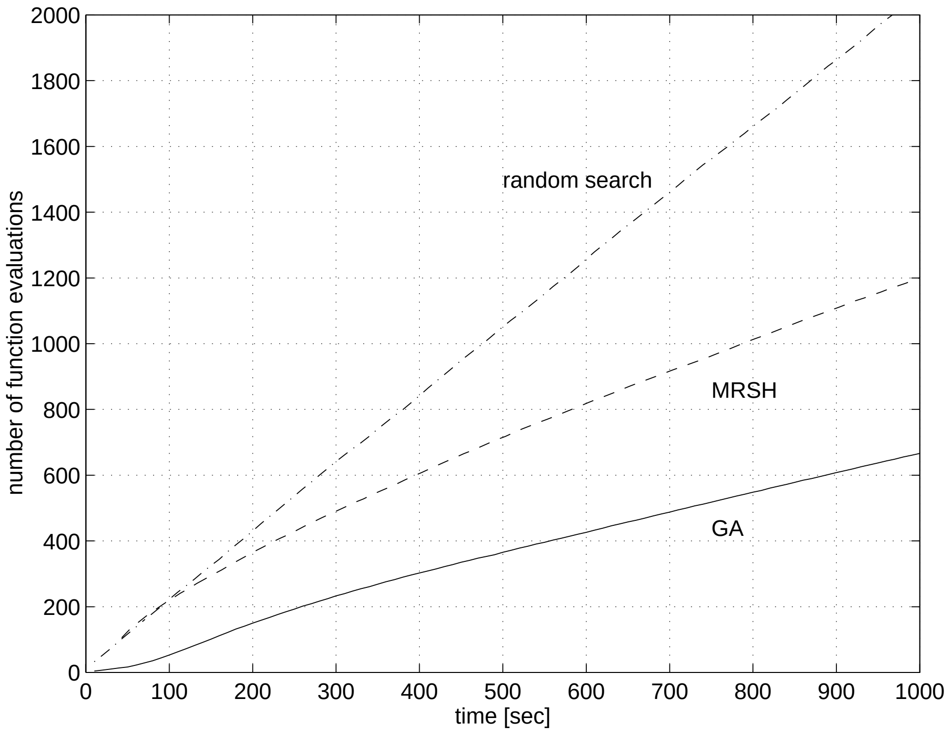


Figure 6-1. The effect of progressive evaluation on the fitness evaluation time

the best solution found so far (a neighbor is a design obtained through a single mutation operation). Because neighbors of good solutions are more likely to be good solutions also, the computation time per function evaluation increases as the algorithm progresses to better solutions. The same slow-down effect occurs for the agent-based genetic algorithm. The large initial difference between GA and MRSH arises from the implementation of the agent-based software; the agents are spawned sequentially, so that it takes about 150 seconds before all agents are active and the full computing power is being used.

Dimensionality of the search space. This characteristic needs to be approached carefully: a TBD problem with a large search space is not necessarily difficult to solve. When the size of the search space is so small that an exhaustive search is feasible, the search problem is very simple. On the other hand, when the search space is very large, the search problem tends to be difficult. However, this is not *always* the case. For instance, a linear programming problem with one hundred continuous variables has a huge search space, yet is relatively easy to solve.

Fraction of acceptably good designs in the search space. Problems for which only a small fraction of the search space corresponds to acceptably good solutions tend to be more difficult.(1) Even when the size of the search space is large, if the fraction of the search space yielding acceptably good solutions is large, then it is relatively easy to find such a solution. On the other hand, if this fraction is very small the problem tends to be difficult. However, if a good fitness function exists that leads the algorithm to these acceptably good solutions quickly, the search problem is still easy. Therefore, a small fraction of acceptable solutions *may* indicate a difficult problem, but one should investigate the characteristics of the fitness function before drawing a final conclusion. Note also that, if an exhaustive search is not feasible, the fraction of acceptably good designs can be estimated by randomly sampling the search space, i.e., random search. The fraction of good designs in the sample is an unbiased estimate of the fraction of good designs in the population.

Region of attraction of the acceptably good designs. When the region of attraction of the acceptably good solutions is small, it is difficult for hill-climbing algorithms to find such a solution. In global optimization, the region of attraction of a global maximum, f^*, in a space A, is defined as the set of points $attr(f^*) \subset A$ such that for any starting point $x \in attr(f^*)$ the infinitely small step steepest ascent algorithm will converge on f^* (Törn and Zilinskas 1989). Based on this definition, one can conclude that the probability that a single run of the infinitely small step steepest ascent algorithm converges on f^* equals:

$$P(f^*) = \frac{\mu[attr(f^*)]}{\mu[A]}, \tag{6-1}$$

where $\mu[Q]$ is the higher dimensional volume of the set Q. The probability $P(f^*)$ is an important characteristic of a global optimization problem. If $P(f^*) = 1$, then the function is either unimodal or all the global maxima have the same value, f^*. In both cases, the problem can be easily solved with local optimization methods. On the other hand, if $P(f^*)$ is close to zero, the problem will be very hard to solve.

(1) The definition of an "acceptably good solution" should be provided by the user; sometimes one is satisfied with a feasible solution, other times one requires optimality. In general, one can define a fitness value, f_a, above which the solutions are considered to be acceptable.

This conclusion can be extended to the case of discrete search spaces and statistical search algorithms. Assume that one uses the (single start) statistical hill-climbing algorithm described in Section 6.2.2. One can start this algorithm at randomly chosen starting points and record how many times the algorithm reaches an acceptably good solution. The fraction of the runs that converges to a good solution is an unbiased estimate of the probability, $P(f \geq f_a)$, that a statistical hill-climbing run converges to an acceptably good solution. Again, this probability characterizes the complexity of the search problem. If $P(f \geq f_a)$ is close to one, the problem can be solved with one or two runs of the statistical hill-climbing algorithm—regardless of how small the fraction of acceptably good solution might be. If $P(f \geq f_a)$ is close to zero, the problem is very difficult and one may have to repeat many runs of the statistical hill-climbing algorithm to find a good solution. In this case, it may be better to use a search method that can overcome local maxima, such as genetic algorithms.

From the probability, $P(f \geq f_a)$, that a single statistical hill-climbing run reaches an acceptably good solution, one can derive the probability that a run of the MRSH algorithm finds a good solution. Assume that the MRSH algorithm consists of n single start statistical hill-climbing runs, then the probability that MRSH arrives at an acceptably good solution equals:

$$P_{MRSH} = 1 - (1 - P(f \geq f_a))^n \tag{6-2}$$

6.3.2 Performance Criterion

In addition to characterizing the problem, the data from the experiments also indicate how well the agent-based genetic algorithm performs compared to random search and multiple restart statistical hill-climbing.

The performance of an optimization algorithm should be expressed by the fitness value as a function of time. In general, it is desirable for optimization algorithms to find *good* solutions *quickly.* Thus, the two main attributes that define the performance of an algorithm are: time and fitness value (or cost function value, search heuristic value, etc.). Sometimes, it is desirable to find a reasonably good solution quickly, while other times, it is preferred to find a very good solution even when finding it requires a long time. By characterizing the performance in terms of the fitness value as a function of time, one can choose the best available algorithm given the time and computer resources that one has available.

It is common in the optimization and search literature to express the performance of an algorithm with respect to the *number of function evaluations*, instead of with respect to time. Indeed, time itself is not an objective quantity for the performance evaluation of an algorithm. The processing power of computers varies widely so that within the same amount of time different computers can perform a different amount of computation. Moreover, the agent-based design framework is motivated in part by the use of free idle-cycles on networked workstations. That implies that even for one particular computer the computation power that is available to us may vary depending on the usage by other users. These dependencies can be avoided by using the *number of function evaluations* as a hardware independent measure of required computation time. However, this practice implicitly assumes that one function evaluation requires a fixed amount of computation.

As is illustrated in Figure 6-1, in the case of TBD the computation cost of the fitness evaluation varies from one solution candidate to another, due to *progressive evaluation* of the fitness heuristic. If one were to use the number of function evaluations as a measure for computing time, one would systematically undervalue the performance of random search and multiple restart statistical hill-climbing with respect to the agent-based genetic algorithm. Choosing the lesser of two evils, we will express the performance of the algorithms as a function of computing time. By using the same set of computers for all our experiments, any systematic biases are excluded (assuming that the usage of the workstations by other users is randomly distributed and uncorrelated, so that it does not skew the results of the statistical analysis).

The performance comparison of *statistical* search algorithms calls for a *statistical* evaluation method. Genetic algorithms, random search, and multiple-restart statistical hill-climbing are all three statistical search methods. How well a particular run of a statistical algorithm performs, depends literally on the luck of the draw—the performance criterion is itself a statistic with a certain probability density function. This further complicates the performance comparison. For instance, it is possible (even common) that algorithm A finds a better solution than algorithm B for a particular run, while on average B does much better than A. This problem can be overcome by comparing a large number of runs for each of the algorithms. If for a given run algorithm A finds a larger fitness value than algorithm B, algorithm A "wins." If both algorithms perform equally well, then each will win 50% of the time. Conversely, if algorithm A is better than algorithm B, A will win more than 50% of the time. This approach is formalized mathematically in the Fisher sign test, which also determines a criterion for statistical significance. The Fisher sign test is the subject of the next section.

In addition to the *relative* comparison between two statistical algorithms, one can also estimate each individual algorithm's ability to find the *absolute* global optimum. Of course, this is only possible when this global optimum is known a priori or has been determined through an exhaustive search.

6.3.3 Fisher Sign Test(2)

Assume that one wants to determine which of two search methods, A or B, yields the largest fitness value, f, after k function evaluations. Both A and B are statistical search methods and f_A and f_B are random variables with a certain probability density function. These probability density functions are unknown, and most likely *not* normal (Gaussian). Therefore, one cannot compare the performance of the algorithms using any of the standard statistical methods, such as the t-test, that are based on the normal assumption. Instead, it is better to use nonparametric statistics, i.e., methods that hold under relatively mild assumptions regarding the underlying populations. In particular, the Fisher test is well suited for our analysis.

Assume that, to compare the performance of two statistical search algorithms, one executes n runs of each algorithm and records the fitness values, f_A and f_B, after k function evalu-

(2) This section borrows heavily from Chapter 3, Section 4 of Hollander and Wolfe (1973).

ations:

run number	f_A	f_B
1	X_1	Y_1
2	X_2	Y_2
.	.	.
.	.	.
.	.	.
n	X_n	Y_n

The sets X_i and Y_i are then samples of the populations f_A and f_B, respectively. One can model the difference in fitness between method A and B as

$$Z_i = Y_i - X_i = \theta + e_i, \qquad i = 1, \ldots, n, \tag{6-3}$$

where the e's are unobservable random variables and θ, is the difference in performance that we are trying to measure. Assume also that the e's are mutually independent (which is true given that we use good random number generator), and that each e comes from a continuous population that has a median of zero, so that

$$P(e_i < 0) = P(e_i > 0) = \frac{1}{2}, \qquad i = 1, \ldots, n, \tag{6-4}$$

which can always be satisfied by choosing θ to be the median of the population $\{Z_i\}$. Although the e's do not always come from a continuous population, Hollander and Wolfe (1973) indicate that the Fisher test, described below, remains valid as long as the zero values among the Z's are discarded and the variable n be redefined as the number of nonzero Z's.

The Fisher test proceeds by defining the indicator variables

$$\varphi_i = \begin{cases} 1 & \text{if } Z_i > 0 \\ 0 & \text{if } Z_i < 0 \end{cases}, \tag{6-5}$$

and the statistic B as the number of positive Z's:

$$B = \sum_{i=1}^{n} \varphi_i \tag{6-6}$$

For a one-sided test of the hypothesis H_0: $\theta = 0$ versus the alternative $\theta > 0$, at the α level of significance, the Fisher test is:

$$\begin{aligned} &\text{reject } H_0 \quad \text{if} \quad B \geq b(\alpha, n, \tfrac{1}{2}) \\ &\text{accept } H_0 \quad \text{if} \quad B < b(\alpha, n, \tfrac{1}{2}) \end{aligned} \tag{6-7}$$

where the constant $b(\alpha, n, 1/2)$ is the upper α percentile point of the binomial distribution with sample size n and $p = 1/2$, that is, $b(\alpha, n, 1/2)$ can be determined from

$$\alpha = \left(\frac{1}{2}\right)^n \sum_{i=B}^{n} \frac{n!}{(n-i)!\,i!} . \tag{6-8}$$

Justification for the Fisher test comes from the fact that, if both algorithms were equally good, the Z's would be positive half of the time on average, and the statistic B would be distributed according to a binomial distribution with $p = 1/2$, which peaks at $n/2$. Equation (6-7) reflects the fact that θ is more and more unlikely to equal zero when the statistic B becomes larger than $n/2$.

It is important to pause here and carefully consider what the null hypothesis, H_0: $\theta = 0$ means. Given that, by definition, θ is the median of the population $\{Z_i\}$, one can conclude that $P(X_i > Y_i) = P(X_i < Y_i) = 1/2$, that is, it is equally likely that algorithm A yields a larger fitness value that algorithm B, as it is for algorithm B to yield a larger fitness value than algorithm A. However, one cannot make any conclusions about the size of the differences, $|Z_i| = |Y_i - X_i|$. It may well be that the probability density function of $\{Z_i\}$ is skewed, which could mean, for instance, that algorithm A tends to perform very poorly when it is outperformed by B, while B is only marginally worse when it is outperformed by

A. The Fisher sign test does not consider the size of the difference and concludes that, as long as the median of the distribution equals zero, the algorithms perform equally well.

With the presentation of the Fisher sign test, we completed the list of all the analysis tools that we will apply in the next sections to two TBD problems: a non-redundant non-fault tolerant task, and a redundant fault tolerant task.

6.4 Non-Redundant Manipulator Design

6.4.1 Problem Description

The goal is to design a 3-DOF manipulator that can perform the task of writing “RMMS” on a white board without colliding with itself or with the white board, and without violating any of the kinematic and dynamic constraints imposed by hardware limitations of the individual modules. We do not consider the orientation of the end-effector, so that three DOFs are sufficient to perform this task. For a detailed description of the trajectory refer to Section 2.4 in Chapter 2.

The modules with which the candidate design solutions can be built are the seven RMMS modules that are currently available: a base module, an end-effector module, three pivot joint modules, one rotate joint module, and one link module, as illustrated in Figure 6-2.

6.4.2 Problem Characterization

To characterize this instance of the TBD problem and to establish ground truth for the performance analysis of the three statistical search methods, an exhaustive search of the design space is performed. This is possible because the set of 3-DOF manipulators that can be constructed from the seven modules in the inventory is relatively small. When taking into account that a legal manipulator configuration has to start with base and end with an end-effector module, and that the relative orientation of axially symmetric modules can be arbitrarily chosen, one can construct 21,120 different 3-DOF manipulators. These are all the manipulators considered by the three statistical methods considered. However, for the exhaustive search, we reduced the search space further by including the knowledge that the three pivot joint modules are functionally equivalent. This results in a set of 3,520 different manipulators, which one can group into twenty cases to be evaluated in parallel on twenty workstations. Each of the twenty cases corresponds to one possible ordering of the manipulator modules, as is shown in Figure 6-3. The complete set of 3-DOF configurations can be generated by cycling through all possible combinations of relative orientations: 512 for cases one through five, 64 for all the other cases (512*5 + 64*15 = 3520).

The fitness evaluations for all manipulator configurations are depicted per case in Figure 6-4. This data provides some interesting insights:

- Cases 16 through 20 have only fitness values of -100,000 or less (the higher the fitness value the better). A careful look at these designs shows that the last joint module is a rotate module which cannot change the position of the end-effector. With only two useful DOFs, this manipulator configuration cannot possibly follow a three dimensional trajectory.
- For case seven, all 64 manipulators yield a feasible solution. The design has a spherical shoulder followed by an arm consisting of two links of equal length (The Puma’s elbow configuration belongs to this case). The data indicate that

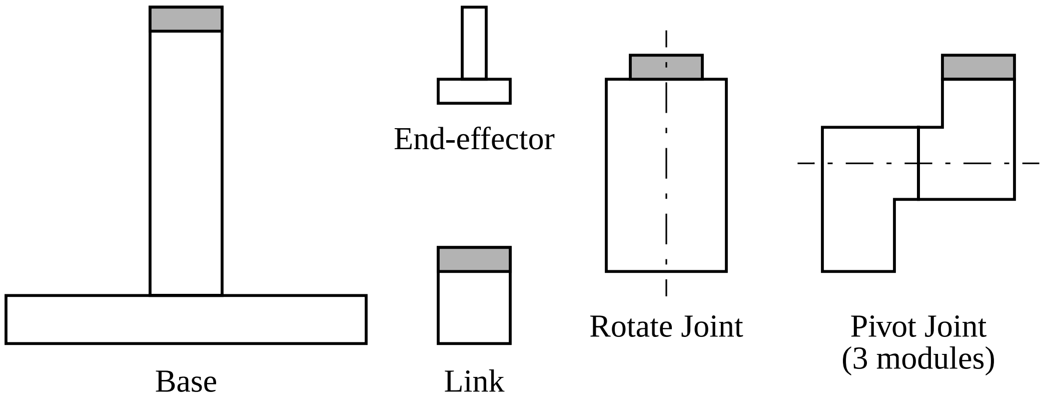


Figure 6-2. Inventory of modules

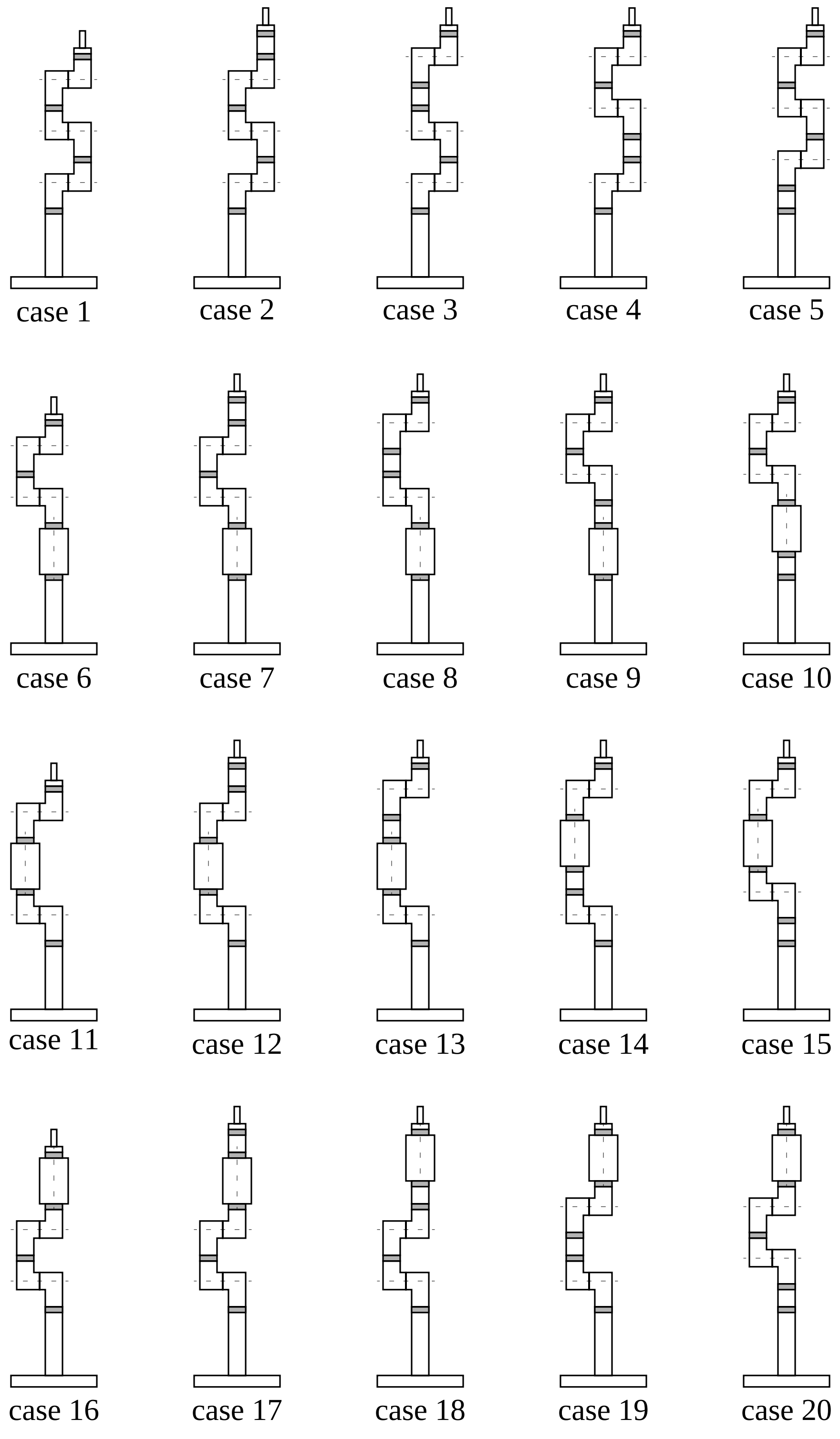


Figure 6-3. Different module orders for 3-DOF manipulators. Each of these cases represents a group of designs obtained by cycling through all the possible combinations of relative orientations. (512 combinations for cases 1–4; 64 combinations for the other cases)

this manipulator can perform the task for all possible twist angles between the second and the third joint.

- Case eight is similar to case seven but with unequal link lengths. The result is that for a few twist angles the manipulator design fails to satisfy all the constraints.
- All of the near optimal designs appear in case 1. The energy consumption for this group of manipulator designs is considerably less than that for the other feasible designs. The reason is that the designs in case 1 only have five modules while the other feasible designs have six modules. Remember that each module uses 16.2 W to power the on-board electronics, so that manipulator configurations with fewer modules consume less energy.
- Out of 3,520 candidate design solutions, 374 designs can perform the task without violating any constraints—a fraction of 10.6%.

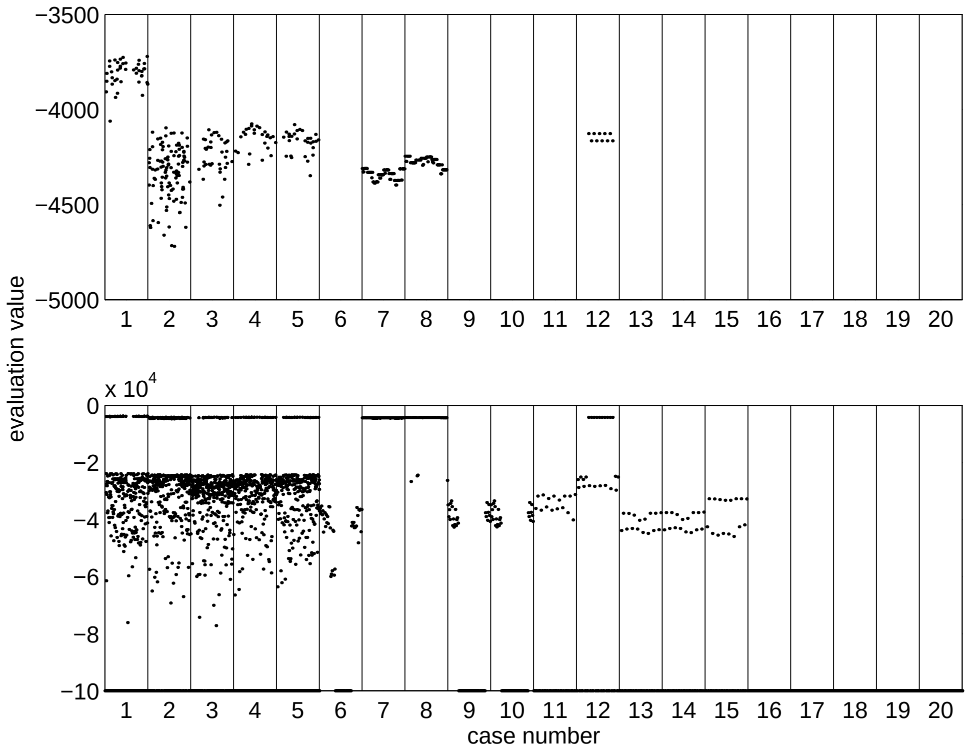
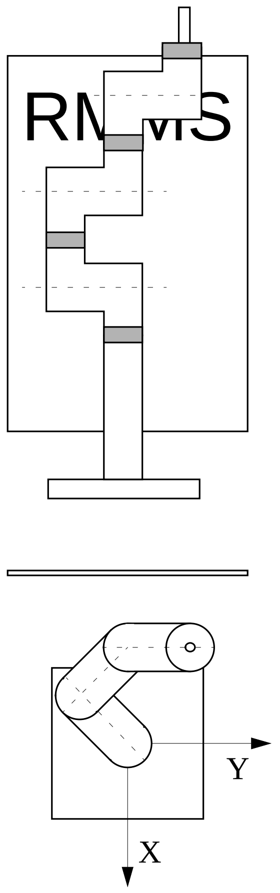


Figure 6-4. Fitness evaluations for all 3-DOF manipulators.



module number	module	relative orientation
1	base	0
2	pivot joint	225
3	pivot joint	270
4	pivot joint	315
5	end-effector	0

Base Position = (0, 0, 0)

Figure 6-5. The optimal solution.

The optimal solution, with an energy consumption of only 3,721 Joules (and a fitness value of -3,721), is shown in Figure 6-5. Unlike most traditional manipulators, its first rotation axis is horizontal and at a 45 degree angle with respect to the white board on which "RMMS" is written. Beyond the fact that this manipulator configuration has only five modules, it is not intuitively clear why this design performs better than other designs. It seems that we are not sufficiently familiar with manipulators that have 45 degree twist angles to gain intuitive understanding of their performance.

Based on the exhaustive search, one can evaluate the four problem characteristics presented in Section 6.3.1. (Only the optimal configuration is considered "acceptably good.")

- *The computation cost of a fitness evaluation:* 6.8 seconds on average.
- *The size of the search space:* 3,520 designs (21,120 considered by the search

algorithms)

- *The fraction of acceptably good designs:* 1/3,520 = 0.028%.
- *The fraction of statistical hill-climbing runs reaching an acceptably good design:* 2.1%

The conclusion is that this TBD problem is relatively simple because the search space is small, and the cost of evaluating a candidate solution is low. Yet the heuristic function is rather poor (contains many local maxima) because only 2.1% of the statistical hill-climbing runs reach the global maximum.

6.4.3 Performance Evaluation

Besides the exhaustive search, we used three statistical search algorithms to solve this TBD problem: random search, multiple-restart statistical hill-climbing, and an agent-based genetic algorithm. To be able to make statistically significant conclusions, we performed thirty runs for each of the three algorithms, and recorded for each run the maximum fitness value obtained as a function of the computing time—up to 2,000 seconds—on the fixed set of 24 Sparc workstations.

Figure 6-6 shows that a genetic algorithm with a population of 200 individuals performs the best in the long term; the other algorithms are either too slow, or lack the robustness. As one might expect, random search is unacceptably slow. However, even though it is inefficient, it is a reliable method because for an infinite number of function evaluations, random search will find the global optimum with probability one. The same cannot be said about multiple-restart statistical hill-climbing (labeled MRSH). Since it is a pure hill-climbing algorithm it tends to get stuck in local maxima; after 500 seconds no additional runs reached the global optimum. No matter how many more function evaluations one performs, the MRSH algorithm cannot escape from the local maxima anymore (assuming that one does not start additional hill-climbing runs). In the case of a genetic algorithm the performance depends on the size of the population. The trade-off between robustness and convergence speed, predicted in Section 6.2.2, is confirmed by the data depicted in Figure 6-6. The genetic algorithm with a population of 50 individuals (labeled GA-50) converges much faster than the genetic algorithm with a population of 200, but gets stuck in local maxima more easily.

When the global optimum is unknown, one can still compare the performance of the agent-based genetic algorithm relative to benchmark statistical algorithms such as random search and multiple restart statistical hill-climbing. Figure 6-7 shows the probability that an agent-based genetic algorithm with a population of 200 individuals (labeled GA-200) outperforms (i.e., finds a design with a higher fitness value) a random search, multiple restart statistical hill-climbing, and a genetic algorithm with a population of 50 individuals.(3) After a slow start, GA-200 easily outperforms random search; in seventy to eighty percent of the runs, GA-200 was able to find a design with a fitness value higher than the best design found by random search. Likewise, multiple restart statistical hill-climbing performs better than GA-200 in the beginning, but it is well outperformed by GA-200 after 1,000 seconds or more. The genetic algorithm with a population of fifty individuals (GA-50) is also more successful than GA-200 for short runs, but GA-200 catches up and seems to perform slightly better

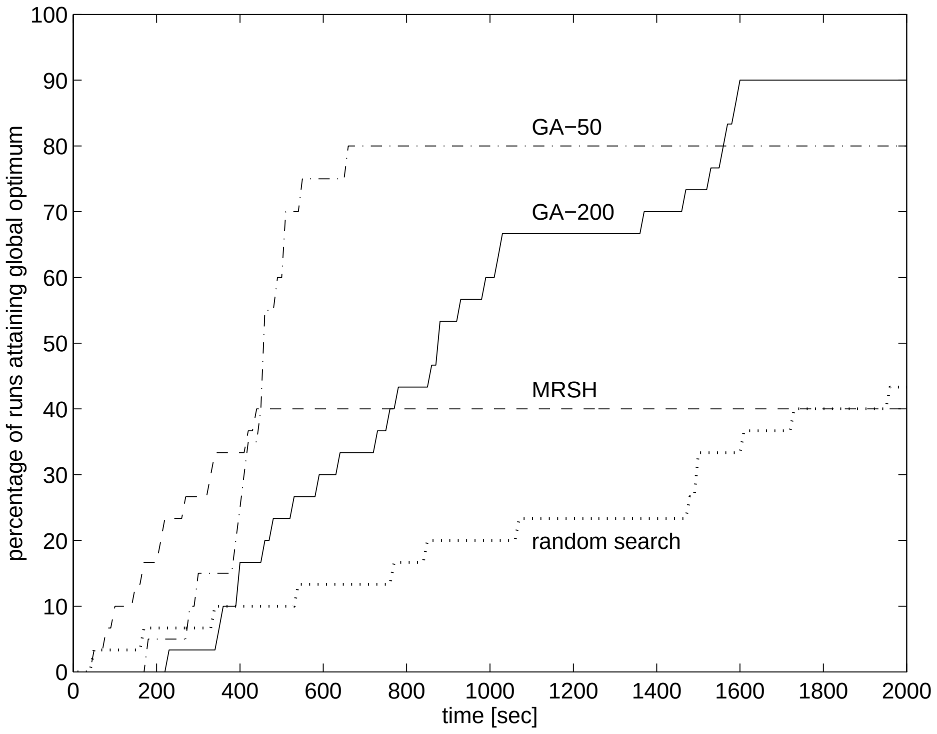


Figure 6-6. Percentage of runs that attained the global optimum.

(3) Figure 6-7 is based on the same data used to generate Figure 6-6.

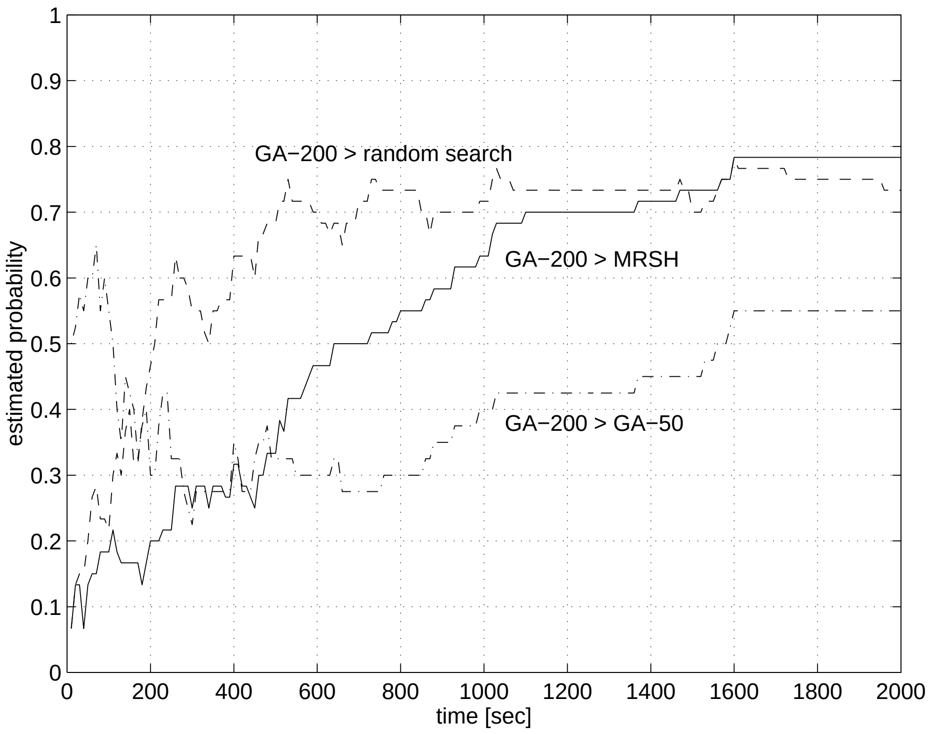
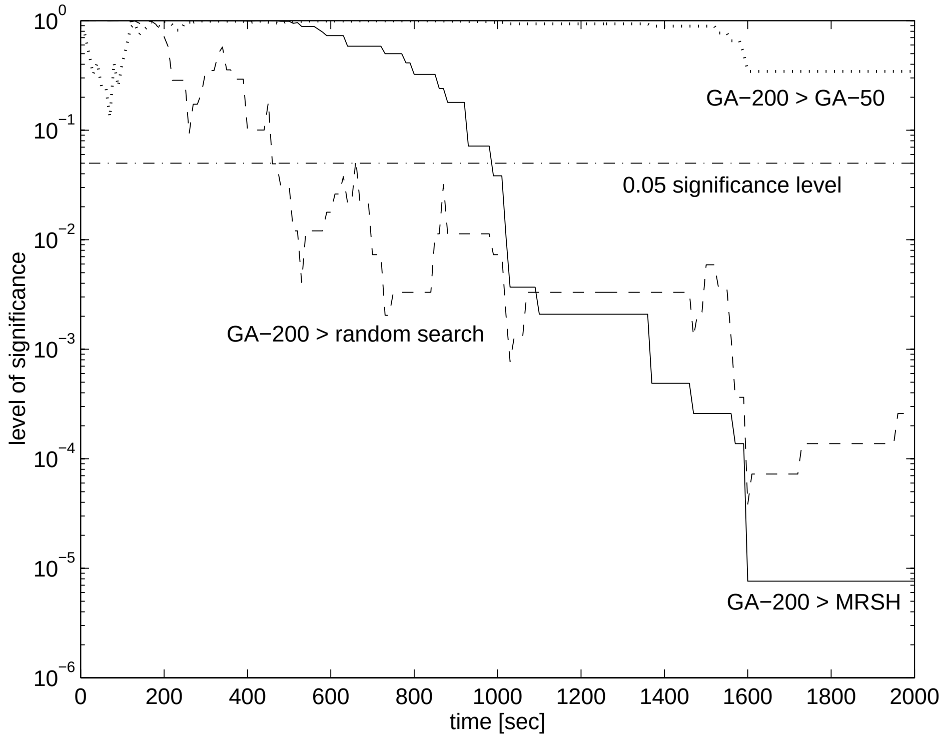


Figure 6-7. The probability that the agent-based genetic algorithm outperforms other statistical algorithms.

after 1,600 seconds.

The Fisher sign test confirms the statistical significance of the performance differences reported in the previous paragraph. Remember that the null hypothesis for the Fisher sign test is that, for a given number of function evaluations, both algorithms have an equal probability of finding a fitness value larger than then the fitness value obtained by the other algorithm. Figure 6-8 indicates the largest level of significance at which one can reject this null hypothesis in favor of the hypothesis that one algorithm is better than the other, that is, has a higher probability of achieving the larger fitness value. It is common to reject the null hypothesis if the level of significance is below 0.05. According to that criterion, one can conclude that both random search and MRSH do better that GA-200 initially, but that after 1,000 seconds the roles are reversed and both are outperformed by GA-200. With respect to the comparison of GA-50 and GA-200, it would be better to perform more experiments to obtain statistically significant results, even though Figure 6-6 seemed to indicate that GA-200 performed better after 2,000 seconds.



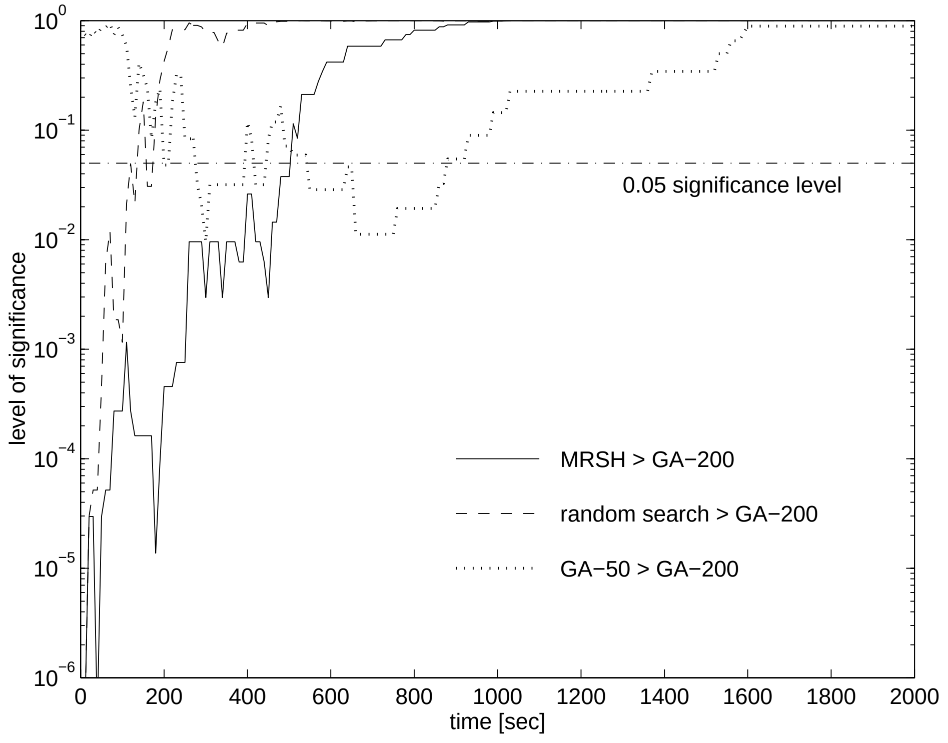


Figure 6-8. Results of the Fisher sign test.

In conclusion, the TBD problem of finding a non-redundant manipulator to write “RMMS” on a white board was a relatively simple problem. However, the heuristic fitness function contained a large number of local maxima, so that MRSH was easily outperformed by the more robust agent-based genetic algorithm.

6.5 Fault Tolerant Manipulator Design

6.5.1 Problem Description

In this section, we explore the performance of the agent-based TBD software for the design of a *fault tolerant* manipulator. The task is the same as the one considered in Section 6.4: trace the letters “RMMS” on a white board. Because fault tolerance requires redundancy, we only consider the 4-DOF manipulators that can be built with the seven existing RMMS modules.

6.5.2 Problem Characterization

The number of different 4-DOF manipulators that can be constructed with the seven RMMS modules is 12,288 (the search space of the statistical algorithms contains 73,728 individuals, because the algorithms do not recognize the functional equivalence of the three pivot modules). These possible designs can be divided into 24 cases, one for each possible ordering of the manipulator modules, as illustrated in Figure 6-9. Each case contains 512 candidate designs, obtained by cycling through all possible combinations of relative orientations for the three non-axially symmetric modules (512 = 8*8*8).

The fitness evaluations for all 12,288 manipulators, sorted by case, are depicted in Figure 6-10. These results support the following interpretation:

- The manipulator designs in cases 19 through 24 all have a fitness value of -100,000 or less. Because the last DOF is aligned with the end-effector, it loses the ability to change the end-effector position. Therefore, the effective number of DOFs reduces to three, which is insufficient for fault tolerant task execution.

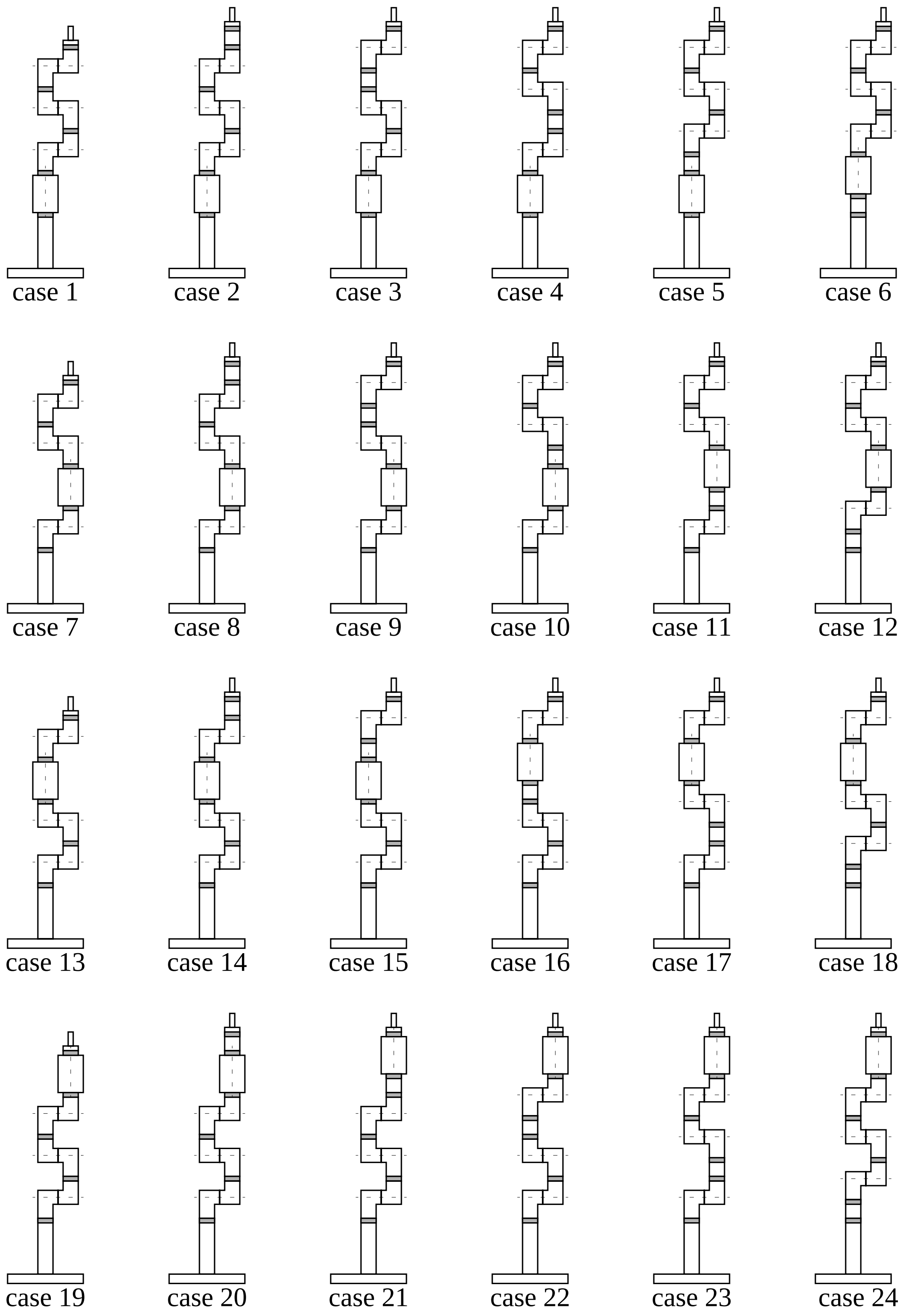


Figure 6-9. The different module orders for 4-DOF manipulators

Each of these cases represents a group of 512 designs obtained by cycling through all the possible combinations of relative orientations.

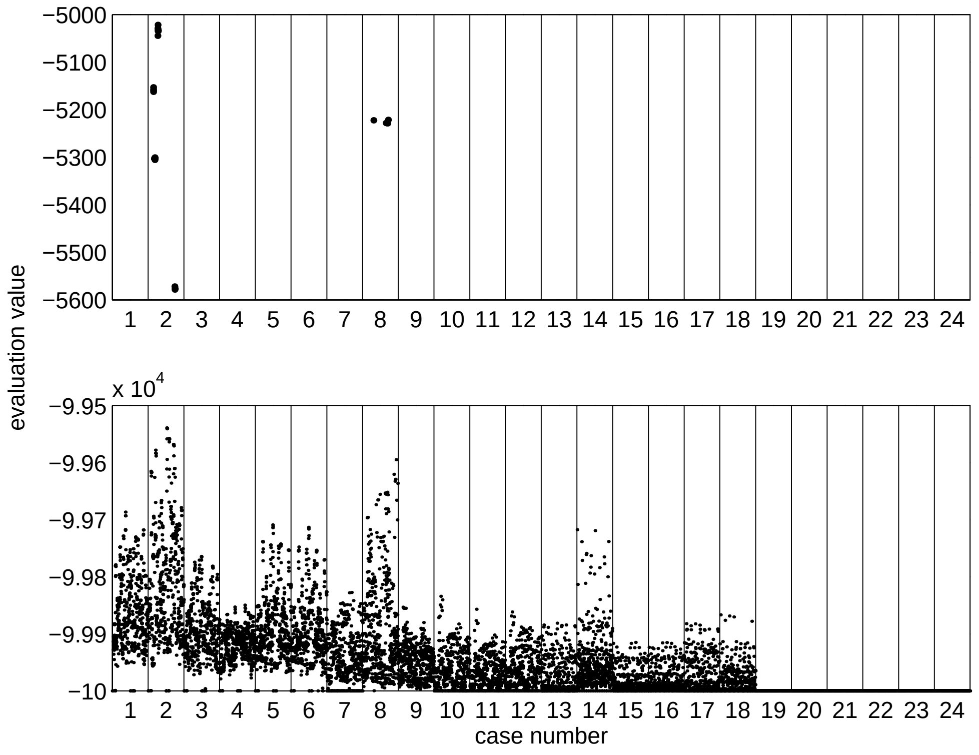


Figure 6-10. Fitness evaluations for all 4-DOF manipulators.

- There are only 22 different feasible solutions—designs that do not violate any task constraints. Their fitness values range from -5,020 to -5,578 (corresponding to energy consumption levels of 5,020 to 5,578 Joules).
- Consider a group of eight designs that differ only in the relative orientation of the pivot joint that follows the rotate joint module. If the rotate joint did not have any joint limits, all eight designs would be perfectly equivalent, because a rotation of the rotate module by some multiple of 45 degrees would result in the exact same physical posture for all eight designs. Based on this fact, the 22 feasible solutions can be further reduced to a set of only six truly different designs, which are listed in Table 6-1 and depicted in Figure 6-11. Notice that, because the rotate joint does have joint limits, some of the eight otherwise equivalent designs become infeasible due to joint limit violations.
- Within each of the six groups of feasible solutions, the fitness value differs slightly even though the designs are almost perfectly equivalent. Part of that

module number	module type	relative orientation				module type	relative orientation	
	case 2	**1**	**2**	**3**	**4**	case 8	**5**	**6**
1	base	0	0	0	0	base	0	0
2	rotate joint	0	0	0	0	pivot joint	270	270
3	pivot joint	45/ 90/ 135	not 90/ 135	not 225/ 270	135/ 225/ 270	rotate joint	0	0
4	pivot joint	45	135	45	315	pivot joint	45/ 180/ 225	135
5	pivot joint	45	45	90	225	pivot joint	225	90
6	link	0	0	0	0	link	0	0
7	end-effector	0	0	0	0	end-effector	0	0

Table 6-1. The six different 4-DOF fault tolerant manipulator designs.

variation is due to a random element in the fitness evaluation function. The initial postures used to generate the polygonal approximation of the self-motion manifolds are determined randomly to increase the robustness of the algorithm. This small variation results in a slightly different desired trajectory, which in turn results in a different energy consumption.

- Except for one solution, all the feasible solutions contain twist angles of 45 degrees. This again makes this design task difficult to solve for the human, who tends to lack intuitive knowledge about the kinematics of manipulators with 45 degree twist angles. Moreover, it is our impression that people also lack good intuition with respect to the fault tolerance requirement.
- There is a correlation between the fitness value and the case number; candidate solutions within the same case attain a similar level of performance. For instance, the two cases containing feasible solutions (case 2 and case 8) contain also many other solutions that satisfy almost all the constraints, and very few

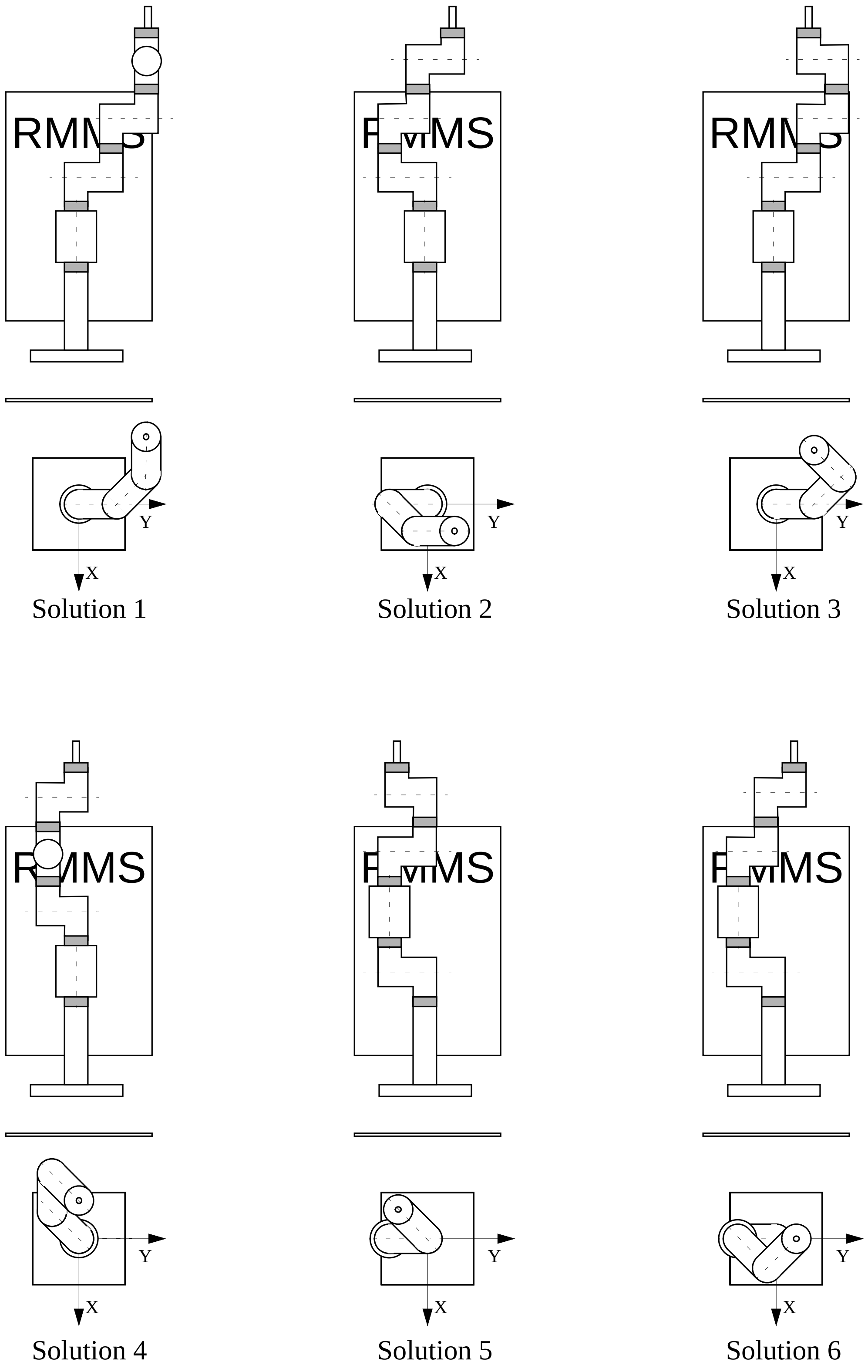


Figure 6-11. The six types of fault tolerant solutions.

solutions with a fitness value of -100,000 or less. One can conclude that it is important to include a relatively large number of MutateRelativeOrientation agents, to explore the other good designs within the same case.

- The highest fitness value (-5020) is obtained by solution 3, described in Figure 6-11 and Table 6-1.

In addition to the above observations, it is interesting to consider the four problem characteristics presented in Section 6.3.1. As mentioned earlier, the optimal solution to this problem is a set of six solutions that have indistinguishable fitness values due to the random element in the fitness evaluation. (To take this random element into account, an “acceptably good” solution has a fitness value larger than –5,030). The four problem characteristics are:

- *The computation cost of a fitness evaluation:* 11.5 seconds on average.
- *The size of the search space:* 12,288 designs (73,728 designs considered by the search algorithms)
- *The fraction of acceptably good designs:* 6/12,288 = 0.049%.
- *The fraction of statistical hill-climbing runs reaching an acceptably good design:* 11.8%

The conclusion is that, even though this TBD problem has a larger search space and a higher evaluation cost than the non-fault-tolerant TBD problem, it is easier to solve because the fraction of acceptably good designs is larger and especially because the fraction of good statistical hill-climbing runs is very high.

The fact that the fraction of good statistical hill-climbing runs is so high indicates that the fitness function does an excellent job of leading the search algorithm towards designs that satisfy the fault tolerance requirement. Once a design is fault tolerant, it is very likely to be a member of “case 2”, so that after a few mutations of the relative orientations, an optimal design is reached.

After having characterized the TBD problem itself, the next section proceeds by analyzing the performance of the three statistical search algorithms considered.

6.5.3 Performance Analysis

In addition to the exhaustive search, we performed 30 runs for each of the three statistical search algorithms: random search, multiple restart statistical hill-climbing, and the agent-based genetic algorithm. In Figure 6-12, the results from these experiments are compared to the results from the exhaustive search. More specifically, for each of the statistical algorithms, the percentage of runs attaining the global optimum, found in the exhaustive search, is plotted as a function of computing time. The agent-based genetic algorithm with a population of 50 individuals clearly performs better than multiple restart statistical hill-climbing and random search (at least after it has overcome the slow start caused by the software implementation).

The comparison between Figure 6-12 and the equivalent graph of the first example (Figure 6-6), confirms that the convergence properties of the algorithms depend not only on the evaluation cost and the fraction of optimal designs, but also on the nature of the fitness heuristic as characterized by the fraction of statistical hill-climbing runs attaining the global maximum. First, notice that the performance of random search is better for example one. Even

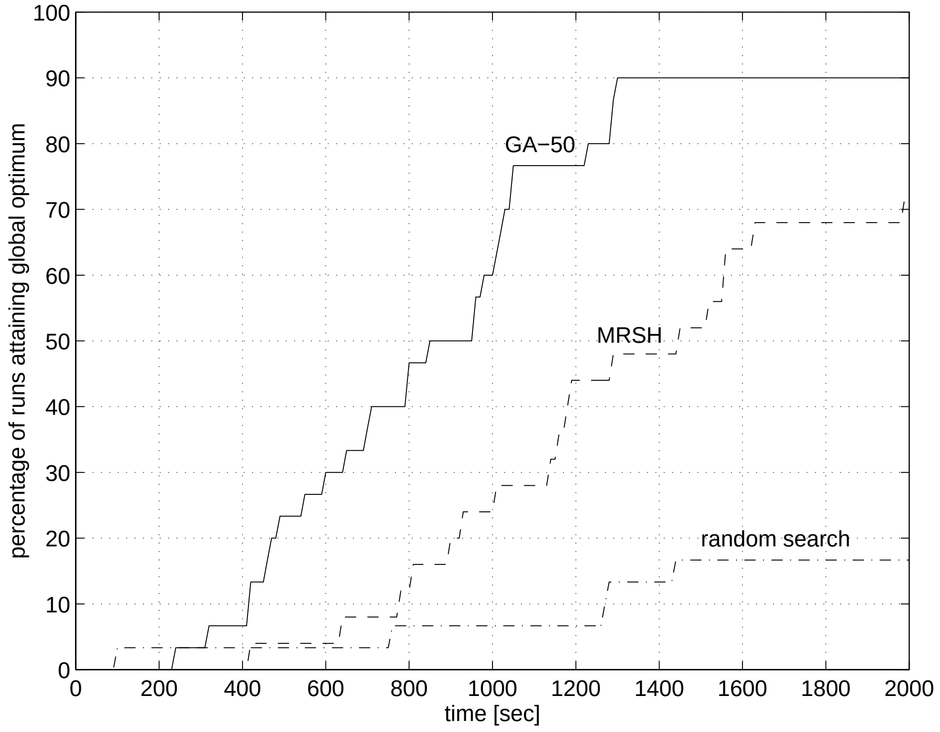


Figure 6-12. Percentage of runs attaining the global optimum.

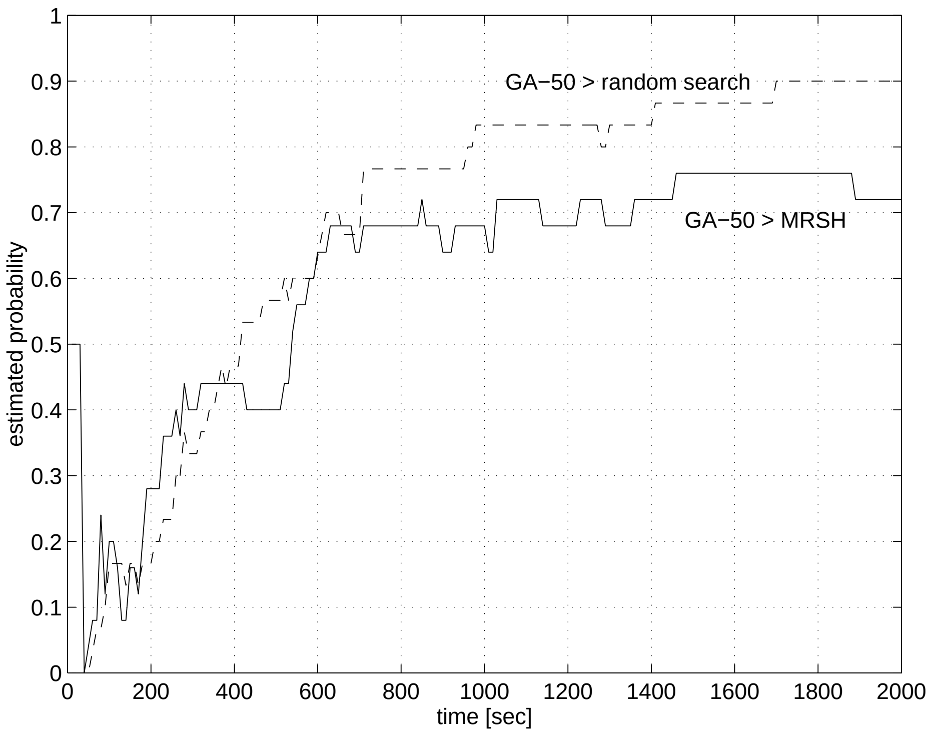


Figure 6-13. Relative comparison between statistical algorithms.

though the fraction of optimal solutions is larger for the fault tolerant example, the time required to evaluate a solution is so much smaller for the first example that random search performs better. Similarly, MRSH and GA-50 converge faster initially for the first example because less time is required to evaluate the designs; yet, both algorithms get stuck more easily at suboptimal solutions for the non-redundant example, so that after 2,000 seconds a higher percentage of optimal solutions has been achieved in the fault tolerant example. Indeed, the fitness function for the fault tolerant example guides the search algorithms more directly towards the optimal solution than in the non-redundant example, without encountering too many local maxima along the way.

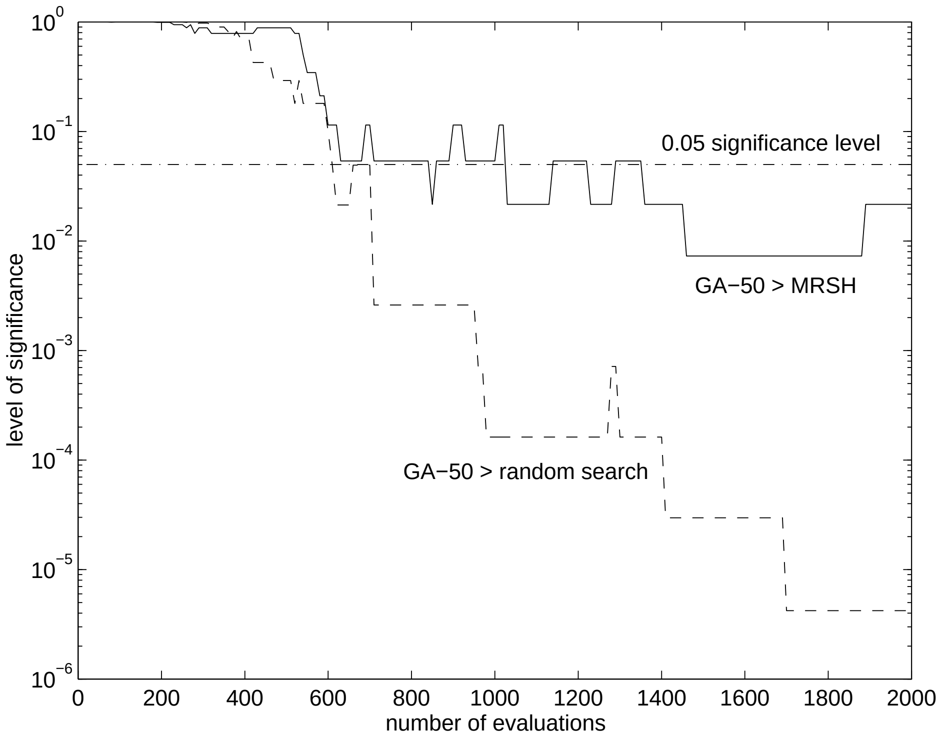
The last part of the performance analysis for the fault tolerant example focuses again on the relative comparison between the statistical algorithms using the Fisher sign test. Figure 6-13 shows the estimated probability that the agent-based genetic algorithm with a population of 50 individuals reaches a higher fitness value than random search, or multiple restart statistical hill-climbing, respectively. Although the graph is based on the same data used to generate Figure 6-12, it does not unambiguously favor GA-50. Unlike Figure 6-12, it clearly shows that GA-50 is initially outperformed by both MRSH and random search (again, this is due to the sequential spawning of the agents causing a slow start of the agent-based genetic algorithm). This initial weakness of GA-50 was impossible to discern in Figure 6-12 because none of the three algorithms had been very successful at reaching the global optimum. Both graphs unambiguously indicate that GA-50 has caught up with the other two algorithms after 700 seconds, and from then on continues to outperform them. These conclusions are supported by the results of the Fisher sign test shown in Figure 6-14. Nevertheless, it would be good to gather some additional data on the comparison between GA-50 and MRSH.

6.6 Summary

This chapter presented an analysis of the TBD problem itself and of the three statistical search algorithms used to solve the TBD problem: random search, multiple restart statistical hill-climbing, and the agent-based genetic algorithm introduced in the previous chapter.

The analyses were performed for two relatively simple TBD problems for which the design space was small enough to make an exhaustive search feasible. For both problems, the exhaustive search provided some interesting insights, the most important of which are that there is a correlation between the module order and the fitness value (designs within the same case, have similar fitness values), and that the optimal designs are counter intuitive to the human designer.

We further characterized each of the two problems based on four problem characteristics: the cost of a fitness evaluation, the size of the search space, the fraction of acceptably good solutions, and the fraction of single start statistical hill-climbing runs to reach an acceptably good solution. Especially this last characteristic is a strong indicator of the problem's complexity. If few statistical hill-climbing runs reach acceptably good solutions, then the fitness heuristic tends to contain many local maxima. In that case, it is best to use an algorithm that



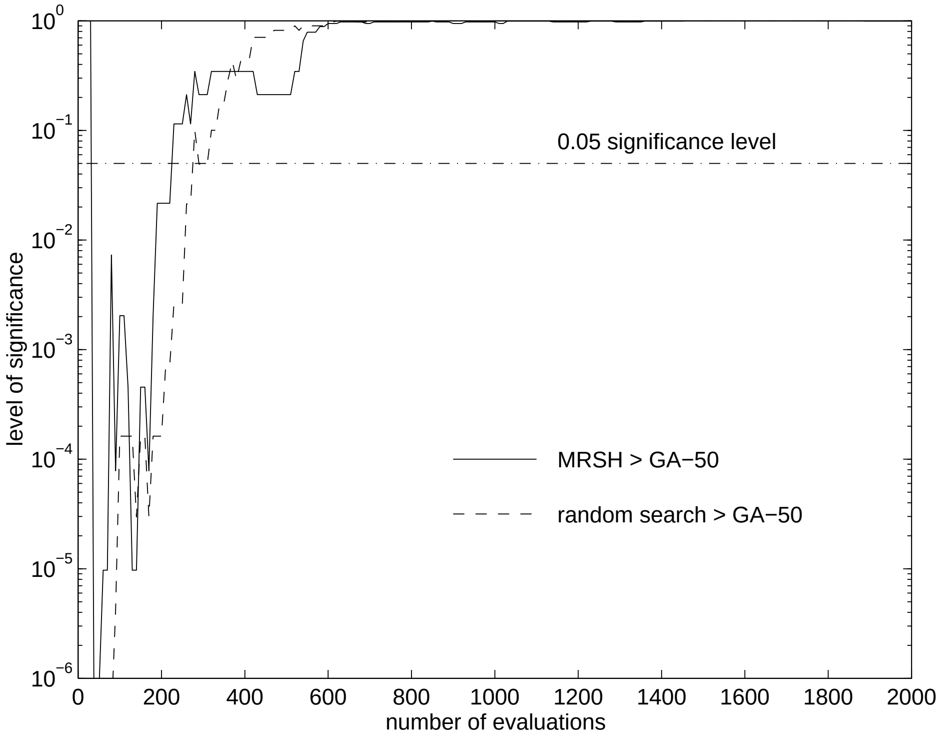


Figure 6-14. Results of the Fisher sign test.

can avoid local maxima, such as the agent based genetic algorithm introduced in the previous chapter.

This conclusion was confirmed by the empirical performance analysis of the three statistical search algorithms. The genetic algorithm performed significantly better than the multiple restart statistical hill-climbing algorithm, especially in the first example for which the fraction of statistical hill-climbing runs that reached an acceptably good solution was small. The random search algorithm performed unacceptably poorly in both examples.

Chapter 7

A Fault Tolerant Manipulator for a Satellite Docking Operation

7.1 Introduction

In the previous chapter, we analyzed the performance of the agent-based design framework for two relatively simple design tasks. This allowed us to perform an exhaustive search of the design space, and to establish a ground truth against which to compare the solutions found by the design system. In this chapter, we give up the luxury of knowing the optimal solution in order to really test the performance limits of the agent-based genetic algorithm. We formulate a highly constrained comprehensive task to be executed with a manipulator built from a large inventory of modules. The result is a design problem with a very large design space and very small set of feasible solutions.

7.2 Problem Description

The setting for the manipulation task that we consider in the chapter is aboard the Space Shuttle Endeavor. The space shuttle has been sent into space to retrieve a faulty satellite, and bring it back to earth. Luckily, the satellite is equipped with an external handle that is well suited for grasping by a robot manipulator, so that the NASA engineering staff has decided to use a manipulator arm for the docking operation. However, there is one big concern, namely, the reliability of the manipulator. If the manipulator were to fail while grasping the satellite, or while pulling it into the cargo bay, both the satellite and the shuttle could be damaged. Or, even without causing any collision damage, it would be possible for the manipulator to get stuck without being able to continue its task successfully, or being able to abort the task and release the satellite. To avoid these complicated and possibly dangerous scenarios, NASA decides to use a redundant manipulator with fault tolerant capabilities, and, to speed up the design and deployment process, the manipulator will be built from reconfigurable manipulator modules. A reconfigurable and modular manipulator system has the additional advantage of being easily repairable in space. For instance, if a joint module fails due to the vibrations during take-off, it could be replaced in space by one of the spare modules on board.

The plan for the retrieval mission is as follows: On earth, a redundant 7-DOF manipulator will be assembled from reconfigurable manipulator modules. This arm will be stowed in the cargo bay of the space shuttle, in its pre-assembled configuration. Once in orbit, the captain of the shuttle will maneuver to a position close to the satellite that is being retrieved; but far enough away to avoid damaging the space shuttle. After making sure that all manipulator modules are still operative, the RMMS will be deployed into the initial posture for the docking task, with the end-effector 0.5m in front of the satellite. From this point on the task becomes critical and can no longer be aborted. The manipulator is switched into fault tolerant mode, which guarantees that it will be able to complete the docking operation even if one of its joints were to fail during the task. The manipulator continues its task by moving forward and grabbing the satellite by its special purpose external handle, and pulling it into the storage unit in the cargo bay of the space shuttle.

In order to execute this task fault tolerantly, the operation needs to be carefully planned beforehand. We will use the agent-based design framework, presented in Chapter 5, not only to design a fault tolerant manipulator, but also to determine the appropriate position and orientation of the space shuttle relative to the satellite, and to plan the corresponding fault tolerant joint space trajectory to be followed by the manipulator before any joint failures occur.

The satellite is cylindrical in shape, with a diameter of 1m and a height of 1.5m. It weighs 1000kg. The handle by which it will be attached to the robot manipulator is located 0.75m from the bottom. The final position of the satellite is in the middle of the cargo bay, 9m back from the cockpit wall. The manipulator base is also positioned in the middle of the cargo bay, but only 2m behind the cockpit wall or 7m in front of the satellite storage unit. To insert the satellite into its storage unit successfully, it is important that the final descent be straight down. Therefore, there is an approach point included in the trajectory, which is 0.75m above the satellite storage unit. Thus, the complete manipulator trajectory is defined by four points (and the corresponding times), as shown in Figure 7-1:

- *Start point (time=0sec):* 0.5m directly in front of the satellite; the orientation aligned up with the orientation of the satellite handle.
- *Connection point (time=10sec):* at the satellite handle.
- *Approach point (time=40sec):* 0.75m directly above the satellite storage unit; the orientation aligned with the orientation of the final position.
- *Storage point (time=50sec):* 7m behind the manipulator base, in the middle of the cargo bay.

Note that the start point and the connection point are defined relative to the initial position of the satellite (which is also the world origin), while the approach point and the storage point are defined relative to the manipulator base. Because the position and orientation of the manipulator base (that is, the space shuttle) are design variables, the Cartesian trajectory varies from one design to another. It is part of the design task to determine the optimal position and orientation of the space shuttle with respect to the satellite. We have restricted the

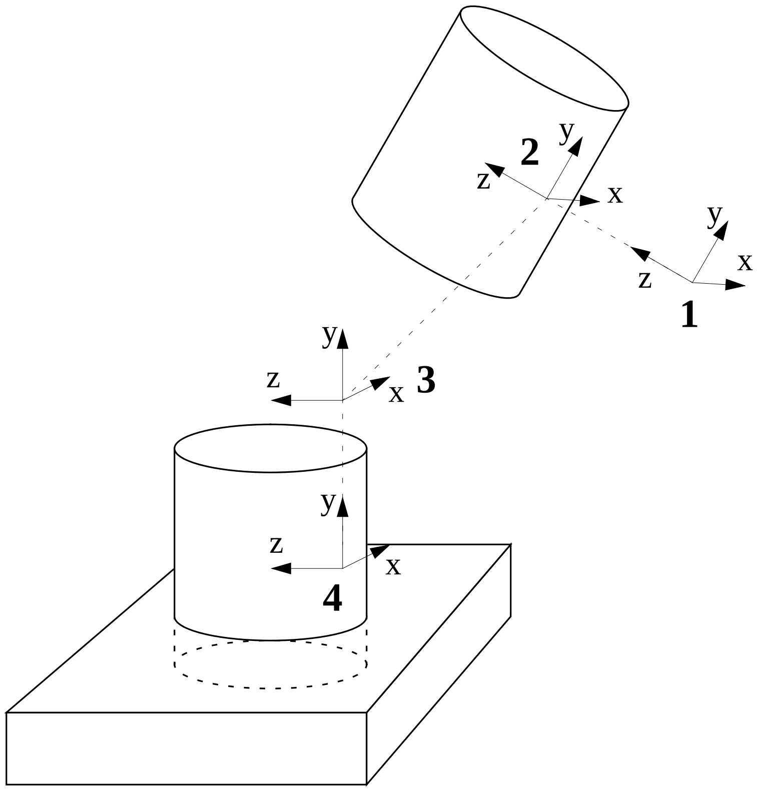


Figure 7-1. The Cartesian trajectory.

The coordinate frames indicated the desired position and orientation of the end-effector of the manipulator.

base position to be within a cube of $24 \times 24 \times 24$ meters centered around the connection point, while there are no restrictions on the orientation of the space shuttle. This ensures that the design system does not waste too much time exploring designs for which the satellite is positioned totally out of the range of the manipulator. On the other hand, we have also restricted the position of the space shuttle to assure that the satellite is not too close to the shuttle, which would make the initial approach maneuver too dangerous. We require that the center of the satellite be more than 3m removed from any part of the space shuttle.

Unlike the two tasks we considered in the previous chapter, this task prescribes the desired position *and orientation* of the manipulator's end-effector. Therefore, to perform this task fault tolerantly, the manipulator needs *seven* DOFs. We have included two 3-roll wrists in the inventory of manipulator modules to provide the necessary orientational capabilities. In total, the inventory contains 23 modules:(1)

- 4 pivot joint modules with varying torque characteristics,
- 4 rotate joint modules with varying torque characteristics,
- 10 link modules with varying lengths,
- 2 corner link modules (with a 90 degree bend) with varying lengths,
- 2 wrist modules (each with three DOFs) with varying lengths,
- 1 base module mounted in the space shuttle cargo bay.

7.3 Results and Interpretation

7.3.1 Problem Characterization

The most important difference between this design task and the tasks considered in the previous chapter is the size of the search space. The search spaces for the examples in Chapter 6 contained 21,120 and 73,728 candidate manipulator designs, respectively. This relatively small size allowed the performance of an exhaustive search, and therefore a more complete analysis. In this example, we consider an inventory of 23 modules, yielding a total of

$$N = \sum_{i=1}^{23} \frac{23!}{(23-i)!} 8^i \approx 1.7 \times 10^{43} \tag{7-1}$$

possible configurations. Even when including the knowledge that a legal configuration has to start with a manipulator base and end with an end-effector, that it should have seven DOFs, and that axially symmetric modules have a zero relative orientation, the number of configurations is still extremely large:(2)

$$N \approx 3 \times 10^{20} . \tag{7-2}$$

(1) Complete specifications of all the modules can be found in Appendix A.
(2) This is the number of configuration considered by all the solution methods.

In addition to the size of this discrete search space, one has to consider the six continuously varying parameters defining the position and orientation of the space shuttle with respect to the satellite.

In addition to the large search space, this TBD problem is complex due to the high computation cost of a function evaluation. As is shown in Figure 7-2, the average computation cost for the fitness evaluation of a random design is relatively small. The random search algorithm performed approximately 12,000 fitness evaluations per hour on 24 Sparc workstations, which corresponds to 7.2 seconds per fitness evaluation on one Sparc workstation. However, as is illustrated in the next paragraph, the vast majority of the randomly chosen designs are very poor, so that the progressive fitness evaluation requires very little time. On the other hand, the average computation cost for the designs considered by the agent-based genetic algorithm is approximately 62 seconds on one Sparc workstation (8340 evaluations in 6 hours on 24 Sparc workstations). Moreover, towards the end of the algorithm's execu-

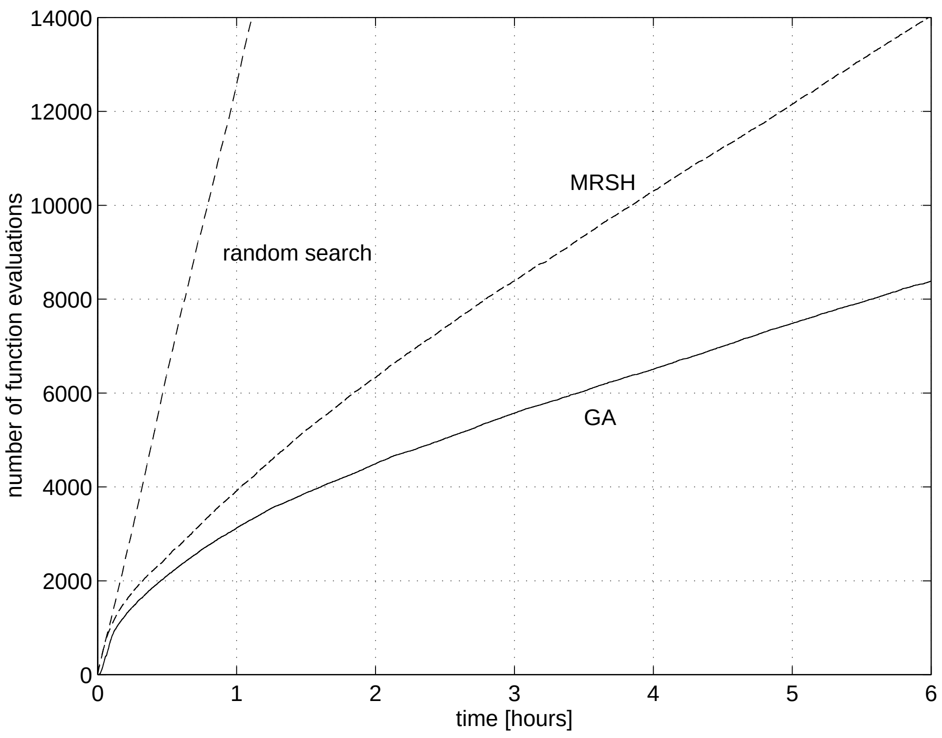


Figure 7-2. The number of fitness evaluations as a function of time.

tion, when the better designs are being evaluated, the average evaluation cost increases to 100 seconds.

To determine the fraction of the design space containing feasible solutions, a random search has been executed; an exhaustive search is impossible due to the size of the search space. Even though the search space is very large, it could be that a large percentage of all candidate design are acceptably good solutions (for this example, a feasible solution, which does not violate any task constraints, is considered to be acceptably good). If this were the case, a random search would find one of those feasible solutions quickly. However, for the satellite docking operation, the random search found only **one** feasible solution after 750,000 function evaluations (42 hours on 24 Sparc workstations). The probability density function of the fitness values is shown in Figure 7-3. About 81% of the solution candidates are unable to reach the end point of the desired Cartesian trajectory, that is, the storage point or point 4 in Figure 7-1. Remember that the determination of the fault tolerant trajectory starts at the *end*

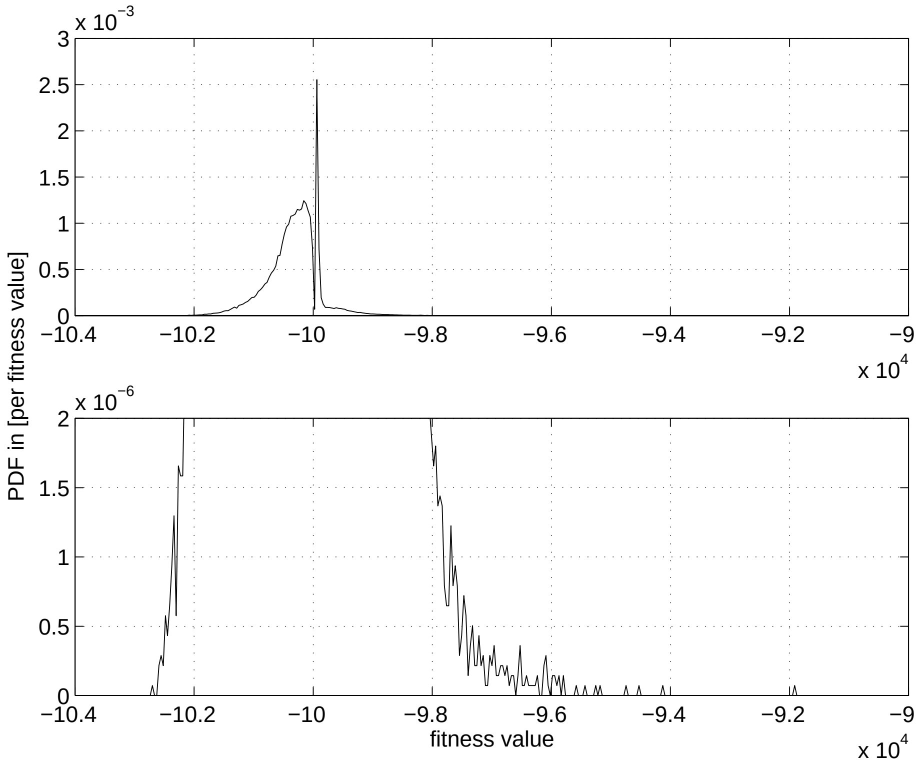


Figure 7-3. Probability density function of the fitness value.

point of the Cartesian path and works its way forward to the start of the path. If a manipulator cannot reach the final point of the path, the fault tolerant trajectory planning algorithm exits immediately, resulting in a fitness value of less than $-100{,}000$. This group of designs is followed by the designs that *can* reach the storage point, but only in postures that violate the secondary requirements. This is a fairly large group of designs that all have the same fitness value of just over –100,000, resulting in a tall spike in the probability density function. The next group of designs, with fitness values between -100,000 and -90,000, are designs for which the set of acceptable postures for a fault tolerant trajectory reduces to the empty set before the initial point of the Cartesian path has been reached. The closer the fault tolerant trajectory algorithm gets to the initial point, the higher the fitness value. According to the probability density function, the probability for a fitness value larger than -95,000 is extremely small (8 solutions out of 750,000). There are two more groups of designs left: the fault tolerant designs that cause a torque limit violation during the simulation (no solutions found in this group), and finally, the feasible designs (one solution out of 750,000).

The next step in the characterization of this TBD problem is to determine the probability that a single start statistical hill-climbing algorithm reaches a feasible solution. Even though only a very small fraction of the design space contains feasible designs, it could still be possible that the TBD problem is relatively simple, namely, if a heuristic function existed that would lead the hill-climbing algorithm directly to the region containing the feasible designs. One can check whether the fitness function for the satellite docking problem exhibits this property by performing a large number of statistical hill-climbing runs. In the experiments for this problem, only 8 out of 480 single start statistical hill-climbing runs converged to a feasible solution—that is approximately 1.7%. All the other runs got stuck in an infeasible local maximum. Although 1.7% is not extremely small, it does indicate that it is best to use an algorithm that can avoid local maxima—for instance, the agent-based genetic algorithm.

In conclusion, the TBD problem for the satellite docking operation is characterized by:

- a very large search space
- a high computation cost for evaluating the fitness of a candidate design
- a very small fraction of feasible designs

- a small probability of reaching these feasible designs through statistical hill-climbing.

The combination of these characteristics result in a very challenging design problem, for which one can expect a high computation cost for finding a feasible solution. This will be further investigated in the next section in which the performance of MRSH is compared with that of the agent-based genetic algorithm.

7.3.2 Performance Analysis of the Statistical Search Algorithms

Unlike the performance analysis in the previous chapter, this section compares the performance of multiple restart statistical hill-climbing and the agent-based genetic algorithm only; random search is not included because it performs so poorly. The MRSH algorithm is the same as the one in the previous chapter: 24 single start statistical hill-climbing algorithms running in parallel on 24 Sun Sparcs. The agent-based genetic algorithm is slightly different from the one in the previous chapter; the group of agents has been recomposed to include the MutateBasePosition agent. In total there are now 46 agents: 20 creation agents, 3 crossover agents, 3 MutateModule agents, 3 AddDeleteModule agents, 3 PermuteModule agents, 3 MutateBasePosition agents, 9 MutateRelativeOrientation agents, one destroyer agent, and one displayer agent. These agents run on the same set of 24 Sun Sparcs—one modification agent per workstation.

Because this satellite docking example is so complicated, the solution process requires a large amount of computation. A single run of the MRSH or the GA lasts 6 hours on 24 Sparc workstations. If one were to use one single Sparc 20, the experiment would last 6 days, which would be absolutely unacceptable. This clearly illustrates the need for a distributed implementation of the design system.

The analysis consists of two components: comparison based on an *absolute* and a *relative* performance criterion. The absolute performance criterion, illustrated in Figure 7-4, is the ability of the algorithms to reach a feasible solution. Because the design space for this example is too large for an exhaustive search, it is impossible to determine the absolute global optimum. However, since the differences in power consumption between the feasible solutions are relatively small, it makes sense to consider any feasible solutions to be acceptably good. This is also convenient because, as we pointed out in Chapter 5, a feasible solution can be recognized by its fitness value which is always larger than –20,000.

As Figure 7-4 indicates, the agent-based genetic algorithm is much more likely than the multiple restart statistical hill-climbing algorithm to achieve a feasible solution. The genetic algorithm failed to reach a feasible solution only in five out of twenty runs. In four of those five runs, it was successful at planning a fault tolerant trajectory, but the execution of this trajectory resulted in a torque limit violation, so that the design did not meet all the task requirements. The MRSH algorithm reached a feasible solution only in seven out of twenty runs, and failed to even plan a fault tolerant trajectory in the remaining thirteen runs. This 35% success rate corresponds to the rate predicted by the results from the single start statistical hill-climbing. The single start statistical hill-climbing algorithm reached a feasible solution in 8 out of 480 runs, that is, in 1.7% of the runs. For a MRSH algorithm starting at

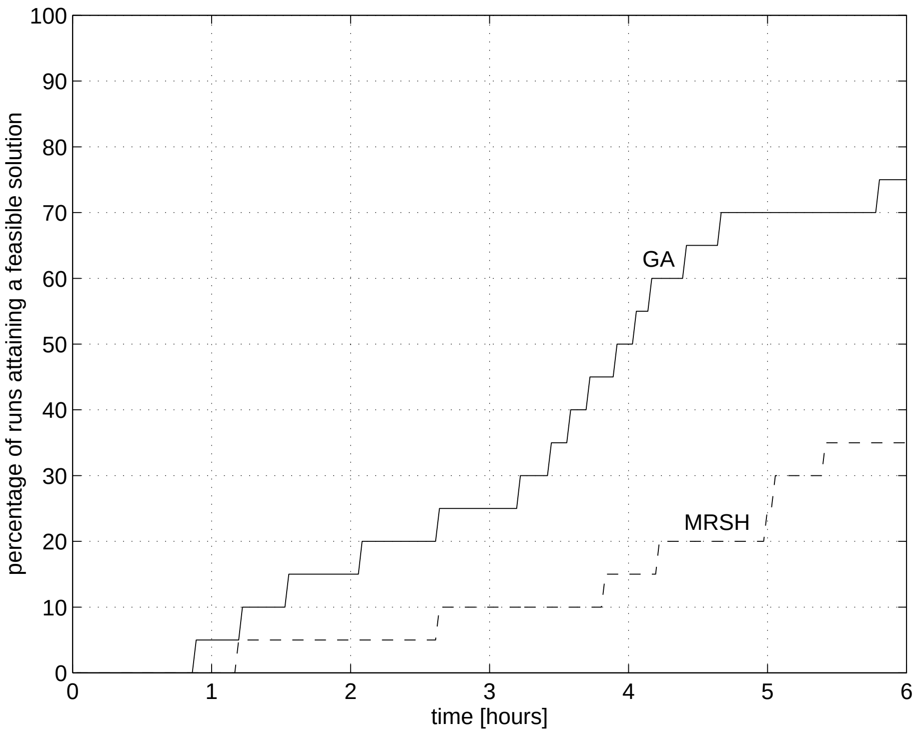


Figure 7-4. Percentage of the runs attaining a feasible solution.

24 random points, the success rate should be:

$$P(success) = 1-(1-0.017)^{24} \approx 0.34\,, \tag{7-3}$$

which corresponds closely to the 35% obtained in the experiment.

According to the relative performance criterion, the agent-based genetic algorithm also outperforms the MRSH algorithm. For the relative performance criterion, the two algorithms are compared in a tournament, run by run. The algorithm that achieves the highest fitness value in a particular run wins. Figure 7-5 depicts, as a function of time, the probability that the agent-based genetic algorithm achieves a higher fitness value than the MRSH algorithm. Initially, the two algorithms perform almost equally well—that is, their fitness value increases at an almost equal pace. Yet, as Figure 7-4 indicates, it is very unlikely that either algorithm reaches a feasible solution at this stage. It is only after about three hours, that the genetic algorithm starts to perform better than MRSH. That is, the probability that it finds a candidate design with a fitness value larger than the one found by MRSH is between 80%

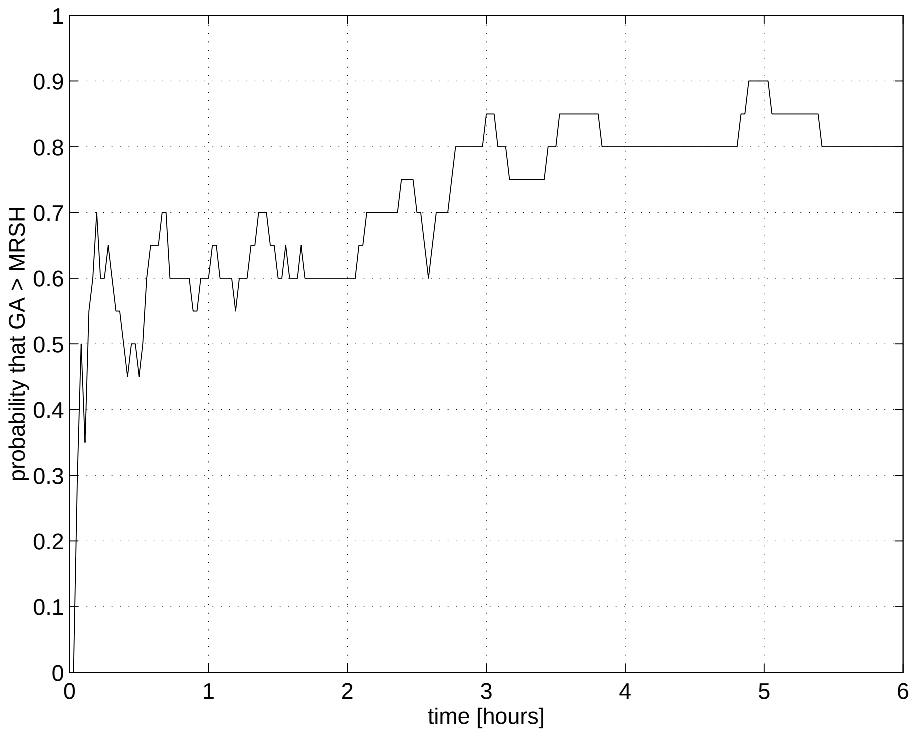


Figure 7-5. Relative comparison between GA and MRSH.

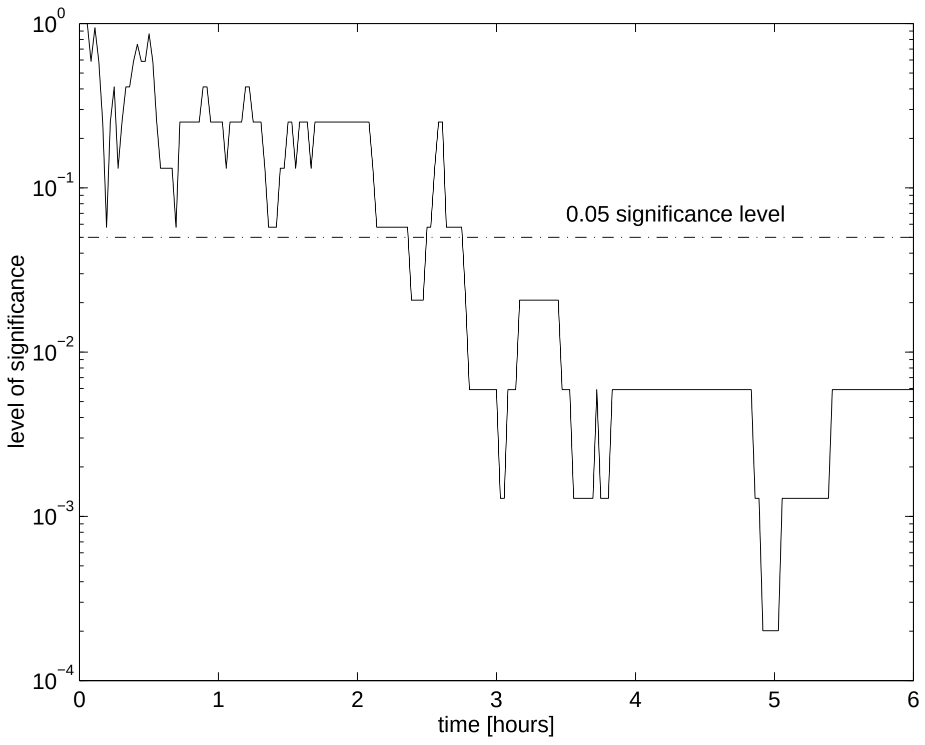


Figure 7-6. Results of the Fisher sign test.

and 90%. Figure 7-6 confirms that this difference in performance is statistically significant according to the Fischer sign test.

In conclusion, three hours into the experiment, the agent-based genetic algorithm performs significantly better than the MRSH algorithm: the chance that the GA achieves a higher fitness value than MRSH is more that 80%. Moreover, after six hours, the agent-based genetic algorithm finds a feasible solution with a probability of 75%.

7.3.3 Interpretation of the Optimal Design

So far the focus has been on the solution process. In this section, we take a closer look at the result of this process, namely, the optimal design found by the agent-based genetic algorithm.

The optimal manipulator configuration is depicted in Figure 7-7. It consists of only eight modules, as listed in Table 7-1. All the other designs found by the agent based design system had more than eight modules. Since the energy consumption (which is the optimality crite-

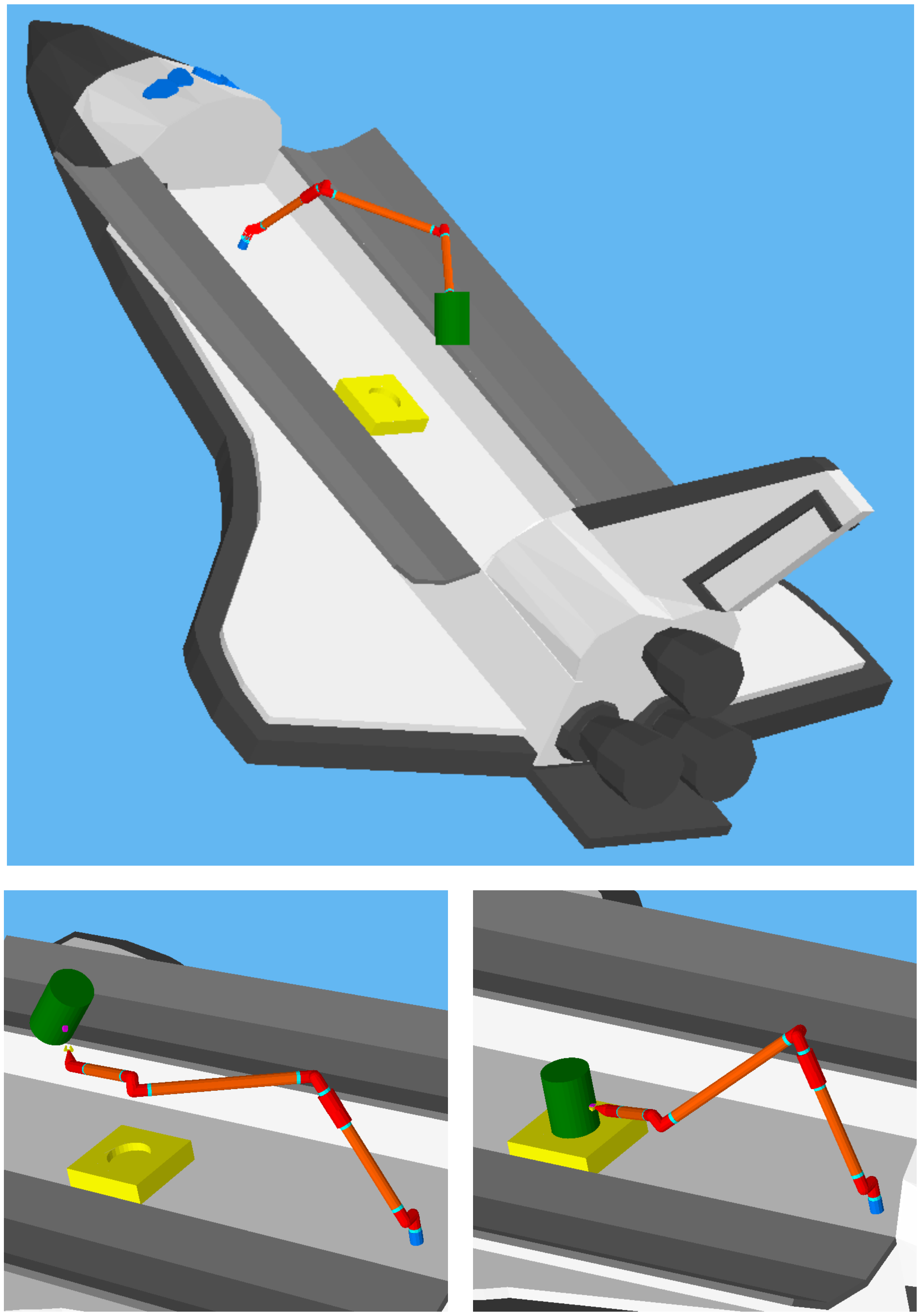


Figure 7-7. The optimal design found by the agent-based design system.

The two figures at the bottom are taken by a camera which is positioned to the right of the cockpit. They show the manipulator in the initial and final posture.

rion) depends on the number of modules (energy consumed by the module electronics), designs with fewer modules are preferred over the others. An additional benefit is that designs with fewer modules tend to have a simpler structure.

Given that the design system does not include any explicit guidelines as to what a good manipulator should look like, it is surprising how much sense this manipulator design makes from the perspective of a human designer:

- The four positional DOFs of the manipulator are very equally distributed over the length of the manipulator (the last link only looks shorter in Figure 7-7 because of perspective distortion). The Denavit Hartenberg parameters shown in Table 7-2 indicate that the three links are almost equal in length: 3.469m, 3.654m, and 3.289m. This is very important from a fault tolerant perspective. If, for instance, the first link were much longer than the others, a failure in the first DOF would severely restrict the positioning capabilities of the manipulator.
- The total length of the manipulator is about 11m. This allows the manipulator to

module number	serial number	relative orientation	joint type	comment
1	sn50	270	pivot joint	Tmax = 400Nm
2	sn76	0	link	length = 2m
3	sn61	0	rotate joint	Tmax = 600Nm
4	sn52	90	pivot joint	Tmax = 600Nm
5	sn77	0	link	length = 3m
6	sn51	270	pivot joint	Tmax = 600Nm
7	sn75	0	link	length = 2m
8	sn91	0	3-roll wrist	length = 1.5m

Table 7-1. The module configuration of the optimal design.

DOF i	d_i	a_i	α_i
1	–0.137	0	$\pi/2$
2	–3.469	0	$\pi/2$
3	–0.275	3.654	$-\pi/2$
4	–0.275	0	$\pi/2$
5	3.289	0	$\pi/2$
6	0	0	$\pi/2$
7	0	0	0

Table 7-2. Denavit Hartenberg parameters of the optimal manipulator design.

reach the initial and final position of the satellite without being fully stretched out or folded back—both of which would be detrimental to the fault tolerant capabilities of the manipulator.

- The DOFs are strongly coupled, as is required for fault tolerance. The twist angles between the pivot joints are all 90 degree angles, and by including a rotate joint between the first and the second pivot modules, the second half of the manipulator can be pointed in any direction. (One could consider the rotate joint followed by a pivot joint to be a 2-DOF spherical joint.)
- The first rotation axis is perpendicular to the axis of the space shuttle, allowing the first link to move along a plane in the middle of the cargo bay. This makes sense because both the initial and final position of the satellite are in the middle of the cargo bay.
- The base position of the manipulator, i.e., the position and orientation of the space shuttle with respect to the satellite, is chosen very carefully. The satellite is positioned almost exactly in the middle of the cargo bay (y-coordinate = –0.18m), 8.36m behind the manipulator base, and 3.85m above the space shuttle. This means that the satellite is as close to its final position as it is allowed to be (3m removed from any part of the space shuttle). Furthermore its orientation

is such that only a small rotation is required to move it to its final position in the storage unit.

7.4 Summary

In this chapter, we analyzed a comprehensive TBD example: a manipulator designed for a satellite docking operation aboard the space shuttle. This TBD problem was extremely challenging due to the large design space, the high cost of evaluating a candidate design, and the small fraction of feasible designs in the design space. Moreover, the MRSH algorithm performed poorly because there are a relatively large number of local maxima in the fitness heuristic. Nevertheless, in almost 80% of the runs, the agent-based genetic algorithm was able to find a feasible solution.

The resulting optimal solution consists of eight manipulator modules and incorporates many of the general criteria for fault tolerant design described in Chapter 3, even though none of these criteria was explicitly included in the design system.

Chapter 8

Conclusions

In this thesis, we have developed all the main components of a rapidly deployable fault tolerant manipulator system: the reconfigurable and modular hardware, the control software, the global fault tolerant trajectory planning algorithm, and the agent-based design framework which is the linchpin of the system. In this chapter, we summarize the main contributions and outline the directions for future work in this area.

8.1 Contributions

To achieve the goal of a rapidly deployable fault tolerant manipulator system, we made the following contributions:

Rapidly Deployable Systems:

- We developed the hardware of a reconfigurable modular manipulator system: the RMMS.

- We implemented and tested a distributed reconfigurable control system that automatically adapts itself to the current manipulator configuration by building configuration independent kinematic and dynamic manipulator models.
- We seamlessly integrated RMMS simulation software with the real-time control software and hardware.

Manipulator Fault Tolerance:

- We formulated a simple yet comprehensive scenario for fault tolerance; it can handle a large variety of faults with one single recovery mechanism: immobilize the failing degree-of-freedom (DOF) by enabling its brake, and continue the task with the remaining DOFs.
- We proved that two degrees-of-redundancy are necessary and sufficient for general purpose fault tolerance.
- We provided an 8-DOF template for general purpose fault tolerant manipulator.
- We proved that, under certain conditions, one degree-of-redundancy is necessary and sufficient for task specific fault tolerance.
- Based on the idea that one can achieve fault tolerance by avoiding unfavorable joint angles *before* failure, we developed a global fault tolerant trajectory planning algorithm.
- We developed an efficient implementation of the global fault tolerant trajectory planning algorithm, which is used to evaluate the fault tolerant properties of candidate manipulator designs in an agent-based design framework.
- We implemented a fault tolerant recovery mechanism that allows a manipulator to continue its task uninterruptedly when a simulated joint failure occurs. We demonstrated this controller on the RMMS.

Task Based Design:

- We considered a very complete definition of the TBD problem, including energy consumption as an optimality criterion, and all of the following task constraints: trajectory reachability, joint position limits, joint velocity limits, joint torque limits, singularity avoidance, obstacle collision, and self-collision.

- We formulated an integrated solution approach to the TBD problem, based on Genetic Algorithms. This approach considers simultaneously the manipulator kinematics and dynamics, trajectory planning, and control.
- We included problem specific knowledge in the genetic algorithm to reduce the size of the search space.
- We introduced the concept of “progressive evaluation,” which drastically reduces the average computation cost of fitness evaluations.
- We introduced an agent-based implementation of the genetic algorithm, which increases the computational power through distributed computing, and provides a modular composable framework for adapting the design system to the design task at hand.
- We performed a detailed performance analysis of the agent-based design framework, by comparing it to exhaustive search, random search, and multiple restart statistical hill-climbing.
- We proposed the Fisher sign test, to compare the performance of statistical search algorithms.
- We solved a comprehensive TBD problem, which consists of designing a modular fault tolerant manipulator for a satellite docking operation with the space shuttle. The design task includes the determination of the optimal position and orientation of the space shuttle with respect to the satellite, and the determination of the corresponding fault tolerant trajectory.

8.2 Future Directions

8.2.1 The RMMS Hardware and Controller

- **A Computed Torque Controller:** At this point, the dynamic model of the RMMS is only used to compute a gravity compensation feed forward torque. Better results could be obtained by implementing a complete computed torque scheme, including inertial, centrifugal, Coriolis, and even friction torques. In addition to changing the control structure at each of the joint modules, the implementation a computed torque controller would require accurate identification of all the inertial and friction parameters. Because the friction parameters for each individual joint tend to depend on the manipulator configuration and the end-effector load, it may even be necessary to identify these parameters on-line, and include them in a simple adaptive controller. In collaboration with the students in Dr. Yangsheng Xu's Robot Control class, we have already obtained some promising preliminary results with this new controller.

- **A Reconfigurable Distributed Control Structure:** The RMMS controller is implemented in a distributed fashion: the PID control for each individual joint is performed on the local microprocessor. The current implementation allows the user to change the PID gains of each controller, but the control structure itself is fixed—it is stored on an EPROM. If one would like to change the control structure, for instance to sliding mode control or force control, one would have to reprogram the EPROMs, which is very time consuming. To further increase the reconfigurability of the RMMS, it would be good to develop a small kernel that allows the user to download a new control program over the LAN connecting all the RMMS modules.
- **Additional RMMS Hardware:** The seven modules that have been built so far allow us to demonstrate the elementary capabilities of the RMMS. However, one important capability is missing: controlling the orientation of the end-effector. The usefulness of the RMMS would be greatly increased with the addition of a 3-DOF wrist module.

8.2.2 Fault Tolerance

- **Extending the Fault Tolerant Trajectory Planning Algorithm:** A key component of our approach to the design of fault tolerant manipulators, is the fault tolerant trajectory planning algorithm presented in Chapter 4. Incorporating this algorithm in the task based design framework, requires a very robust implementation (the algorithm should also work for degenerate candidate designs generated by the creation and modification agents). This has not yet been accomplished for manipulators with two degrees-of-redundancy. Moreover, as we indicated in Chapter 4, the algorithm is limited to two degrees-of-redundancy because of its computational complexity. It is an interesting challenge to develop fault tolerant trajectory planning algorithms for manipulators with more than two degrees-of-redundancy, and in particular, for hyper-redundant manipulators. Due to the large number of DOFs of hyper-redundant manipulators, the crippling effect of one single joint failure can be expected to be rather small. Consequently, it may be possible to achieve a guarantee for fault tolerance based only on a local rather than a global planning strategy.

- **Fault Detection and Identification and Supervisory Control:** In this thesis, we have limited ourselves to fault recovery and design of fault tolerant manipulators. We assumed that the controller is notified when a failure and subsequent immobilization has occurred in one of the joints. However, thus far, we have no means of detecting faults, nor do we have means of assessing the impact of subcomponent failure on the healthiness of the overall system. Two things are missing: Fault Detection and Identification (FDI) algorithms that monitor the health of subcomponents, and an intelligent supervisory controller that controls the fault recovery mechanism and can decide to shut down the system when all redundancy provisions have been exhausted.
- **Distributed Fault Tolerant Control of the RMMS:** The current fault recovery mechanism of the RMMS is based on a centralized implementation. The core of the algorithm resides on the Chimera real-time processors, so that a failure in the communication between one of the modules and the Chimera hardware is catastrophic. However, the distributed computing facilities of the RMMS are well suited for a hierarchical fault tolerant supervisory controller, which would contain faults locally and only send error signals to higher levels when all the low level redundancy provisions have been exhausted.
- **Fault Tolerance of Mechatronic Systems:** Research on fault tolerance has been mainly limited to computing systems. This thesis demonstrates that some

of the same concepts can be applied to electro-mechanical systems. A promising example could be an agile manufacturing system. Currently, if one of the stations of an assembly line fails, the whole line stops as soon as the buffers are depleted (fault intolerance). Even though the mean-time-between-failures statistics for manufacturing equipment in advanced serial-process factories remain quite good, downtime still occurs during feeder jams, part exhausts, and machine malfunctions. By including extra flexibility in the assembly stations, and by providing parallel transportation segments, it would be possible for the assembly line to reconfigure itself when one of the assembly stations fails. One such fault tolerant assembly line is the “Fusion Factory” at Motorola (Strobel and Johnson 1993). This prototype demonstrates that fault tolerance of large mechatronic systems is feasible and even cost effective. Further research could focus on optimal assembly line layout and distribution of the functionality over the assembly stations to reduce the redundancy to a minimum while still maintaining fault tolerance.

8.2.3 Task Based Design

- **High Level Task Descriptions:** The current definition of the TBD problem requires a very low level description of the task. Often, it would be more convenient for the user to provide a high level task description, such as “assemble this motor.” In order to create a suitable manipulator for such a high level task, one would first have to translate this high level task description into low level primitives. This can be achieved for instance with CASPER (Carnegie Mellon ASsembly Planner and ExecutioneR). However, to create a low level assembly description, CASPER uses the kinematic structure of the manipulator as a constraint, so that there exists a strong coupling between the task level planner and the TBD problem. Therefore, it may be necessary to integrate both problems into one framework. Further research is required to investigate this problem.
- **Determination of the Parameters of the Agent-Based Design Framework:** The agent-based genetic algorithm that we presented in Chapter 5 contains certain parameters that need to be chosen by the user, namely, the population size

and the composition of the group of agents. The choice of these parameters will affect the performance of the algorithm. A carefully executed sensitivity analysis is needed to aid the user in determining the optimal parameters for a given problem. In this thesis, we did not conduct any sensitivity analysis, because of the enormous time and computation requirements needed to perform this analysis in a statistically sound manner. Instead we made a conservative choice, preferring robustness over convergence speed.

- **Design of Complex Electro-Mechanical Systems:** Even though the agent-based design framework, was developed to solve the TBD problem, it is based on general principles that will allow it to be applied in other design areas. In particular, the design of complex electro-mechanical systems displays some striking parallels: it requires an integrated design approach that simultaneously considers electrical and mechanical aspects of the design; the design space is also very large, and the design process will most like require a very large amount of computation. Nevertheless certain key differences will have to be addressed. For instance, the design space is not as well defined as for TBD (no longer structure configuration task). Also, for electro-mechanical design, there does exist a large amount of design knowledge, so that one will have to provide a framework for including this knowledge into the design system.
- **A Framework for Including Problem Specific Knowledge in Agent-Based Genetic Algorithms:** In Chapter 5, we indicated how the inclusion of problem specific knowledge can drastically improve the performance of the agent-based genetic algorithm. We included problem specific knowledge to reduce the size of the search space, and to reduce the cost of the fitness evaluation. Yet, this knowledge was included in a rather ad hoc manner. There does not exist a framework for including knowledge systematically, or for including new additional knowledge easily. Further research is required to provide simpler mechanisms to translate existing human knowledge into new agents.
- **Including Learning Strategies in Agents:** Instead of creating agents based on existing human knowledge, it would be an interesting extension to have the

agent acquire new knowledge themselves. By observing the design process—guided by the interactions of all the agents as a group—the individual agents could learn certain general characteristics of the problem and use this knowledge to improve their actions and guide the search more quickly towards promising areas of the search space.

8.3 Conclusions

In this thesis, we have developed a rapidly deployable fault tolerant manipulator system. Such a system combines reconfigurable hardware with support software to enable the user to rapidly configure and deploy a fault tolerant manipulator that is optimally suited for a given task.

The central building block of a rapidly deployable system is the reconfigurable modular hardware: the RMMS. We have built seven RMMS modules that can be easily configured into a large number of different manipulators. The control software that we have developed supports configuration independent programming and control of the RMMS by automatically adapting itself to the current manipulator configuration.

For the deployment of manipulator systems in critical task, for instance in hazardous or remote environments, we proposed to include fault tolerance into the manipulator system. We proved that, to achieve fault tolerance with a minimum number of redundant, it is important to consider trajectory planning and redundancy resolution at the design stage. To accomplish this, we developed a global fault tolerant trajectory planning algorithm with which it is possible to guarantee that a task can be completed even if one of the DOFs of the manipulator system fails and is immobilized.

The most important part of this thesis is the development of the agent-based design framework for Task Based Design. In TBD all the components of a rapidly deployable fault tolerant manipulator system are tied together: the reconfigurable hardware, the control software, fault tolerance, and trajectory planning. To solve this design problem, we have formulated an integrated solution approach implemented in an agent-based genetic algorithm. Our agent-based design framework, combines an effective search strategy with the high computing performance of distributed networked workstations. The effectiveness of the search was improved by including problem specific knowledge in the agents, reducing the size of the search space and reducing the cost of evaluating the fitness heuristic.

We have performed an extensive analysis of the TBD problem based on two relatively simple tasks which allowed for an exhaustive search. The data from this exhaustive search provided additional insight into the TBD problem and served as a baseline for the further performance analysis. We formulated four problem characteristics that allow one to compare the complexity of different TBD problems. We further compared the performance of the agent-based genetic algorithm with two other statistical search algorithms: random search and multiple restart statistical hill-climbing. The conclusion is that the agent-based genetic algorithm outperforms both other algorithms, especially when the fraction of statistical hill-combing runs that reach the global optimum is small.

Finally, we used the agent-based design framework to solve a comprehensive TBD problem for a satellite docking operation with a manipulator mounted inside the space shuttle cargo bay. Even though this is a very challenging problem, the agent-based genetic algorithm was able to find a feasible solution consistently.

Appendix A: Module Description Files

A.1 The RMMS modules

The following module description files, describe the seven RMMS modules that have been built so far: a base module, an end-effector module, three pivot modules, a rotate module, and a link module. The same module description files are used in the TeleGrip simulation software, and in the task based design examples in Chapter 6.

File sn00.cfig:

```
# This cfig file describes an RMMS base module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn00 | base         | 0 | hight 0
SERIAL_NR       sn00
MOD_TYPE        base
AXIAL_SYM       yes
MOD_NDOF        0

# end with the last transformation matrix
MASS            500.0                                       # in kg
CENTER_MASS     0.0     0.0     0.0                         # in m
INERTIA_MOMENT  0       0       0       0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0  0.0  0.0 0.0  1.0  0.0 0.0 0.0
CAD_MODEL       sn00.0
EOF
```

File sn01.cfig:

```
# This cfig file describes an RMMS pivot joint module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn01 | pivot  joint | 1 | Tmax = 270 Nm;  Qmin/max = -/+ 185 deg
SERIAL_NR       sn01
MOD_TYPE        module
AXIAL_SYM       no
MOD_NDOF        1

# for every DOF include the following data:
MASS            5.36                                        # in kg
CENTER_MASS     0.039   0.0     0.1625                      # in m
INERTIA_MOMENT  0.02    0.02    0.01    0       0       0       # in kgm2
T_MATRIX        0  0  1  0  1 0  -1  0  0  0.06865  0  0.1825
CAD_MODEL       sn01.0
JOINT_TYPE      revolute              # revolute or prismatic
Q_MIN           -2.8797933            # in radians (+/- 165 deg)
Q_MAX           2.8797933             # in radians
QD_MIN          -0.9                  # in rad/sec
QD_MAX          0.9                   # in rad/sec
T_MIN           -270.0                # in Nm
T_MAX           270.0                 # in Nm
COULOMB         8.0                   # in Nm
VISCOUS         32.0                  # in Nmsec/rad
I_ARMATURE      8.0                   # in kgm2
POWER_PAR       0.001                 # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.5922903e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1406336
TACHO_OFFSET    -0.9400418

# end with the last transformation matrix
MASS            5.36                                        # in kg
CENTER_MASS     0.02    0.0     -0.03                       # in m
INERTIA_MOMENT  0.01    0.01    0.02    0       0       0       # in kgm2
T_MATRIX        0  0  -1  0  1  0  1 0  0  0.1445 0 -0.06865
CAD_MODEL       sn01.1
EOF
```

File sn02.cfig:

```
# This cfig file describes an RMMS pivot joint module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn02 | pivot  joint | 1 | Tmax = 270 Nm;  Qmin/max = -/+ 185 deg
SERIAL_NR       sn02
MOD_TYPE        module
```

```
AXIAL_SYM       no
MOD_NDOF        1

# for every DOF include the following data:
MASS            5.36                                        # in kg
CENTER_MASS     0.039   0.0     0.1625                      # in m
INERTIA_MOMENT  0.02    0.02    0.01    0       0       0       # in kgm2
T_MATRIX        0  0  1  0  1 0  -1  0  0  0.06865  0  0.1825
CAD_MODEL       sn02.0
JOINT_TYPE      revolute                # revolute or prismatic
Q_MIN           -2.8797933              # in radians (+/- 165)
Q_MAX           2.8797933               # in radians
QD_MIN          -0.9                    # in rad/sec
QD_MAX          0.9                     # in rad/sec
T_MIN           -270.0                  # in Nm
T_MAX           270.0                   # in Nm
COULOMB         8.0                     # in Nm
VISCOUS         32.0                    # in Nmsec/rad
I_ARMATURE      8.0                     # in kgm2
POWER_PAR       0.001                   # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.6259e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1379491
TACHO_OFFSET    -0.9483085

# end with the last transformation matrix
MASS            5.36                                        # in kg
CENTER_MASS     0.02    0.0     -0.03                       # in m
INERTIA_MOMENT  0.01    0.01    0.02    0       0       0       # in kgm2
T_MATRIX        0  0  -1  0  1  0  1 0  0  0.1445 0 -0.06865
CAD_MODEL       sn02.1
EOF
```

File sn03.cfig:

```
# This cfig file describes an RMMS link module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn03 | link           | 0 | Length = 163mm
SERIAL_NR       sn03
MOD_TYPE        module
AXIAL_SYM       yes
MOD_NDOF        0

# end with the last transformation matrix
MASS            2.13                                        # in kg
CENTER_MASS     0.0     0.0     0.081                       # in m
INERTIA_MOMENT  0.007   0.007   0.005   0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0 0.0  0.0  0.0  1.0  0.0  0.0  0.163
CAD_MODEL       sn03.0
EOF
```

File sn04.cfig:

```
# This cfig file describes an RMMS pivot joint module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn04 | pivot  joint | 1 | Tmax = 270 Nm;  Qmin/max = -/+ 185 deg
SERIAL_NR       sn04
MOD_TYPE        module
AXIAL_SYM       no
MOD_NDOF        1

# for every DOF include the following data:
MASS            5.36                                        # in kg
CENTER_MASS     0.039   0.0     0.1625                      # in m
INERTIA_MOMENT  0.02    0.02    0.01    0       0       0   # in kgm2
T_MATRIX        0  0  1  0  1 0  -1  0  0  0.06865  0  0.1825
CAD_MODEL       sn04.0
JOINT_TYPE      revolute                # revolute or prismatic
Q_MIN           -2.8797933              # in radians (+/- 165 deg)
Q_MAX           2.8797933               # in radians
QD_MIN          -0.9                    # in rad/sec
QD_MAX          0.9                     # in rad/sec
T_MIN           -270.0                  # in Nm
T_MAX           270.0                   # in Nm
COULOMB         8.0                     # in Nm
VISCOUS         32.0                    # in Nmsec/rad
I_ARMATURE      8.0                     # in kgm2
POWER_PAR       0.001                   # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.5861927e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1298002
TACHO_OFFSET    -0.938335

# end with the last transformation matrix
MASS            5.36                                        # in kg
CENTER_MASS     0.02    0.0     -0.03                       # in m
INERTIA_MOMENT  0.01    0.01    0.02    0       0       0   # in kgm2
T_MATRIX        0  0  -1  0  1  0  1 0  0  0.1445 0 -0.06865
CAD_MODEL       sn04.1
EOF
```

File sn05.cfig:

```
# This cfig file describes an RMMS rotate joint module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn05 | rotate joint | 1 | Tmax = 280 Nm;  Qmin/max = -/+ 165 deg
SERIAL_NR       sn05
MOD_TYPE        module
```

```
AXIAL_SYM       yes
MOD_NDOF        1

# for every DOF include the following data:
MASS            5.8                                         # in kg
CENTER_MASS     0.0     0.0     0.14                        # in m
INERTIA_MOMENT  0.04    0.04    0.006   0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  -1.0 0.0  0.0  0.0  -1.0  0.0  0.0
0.2037
CAD_MODEL       sn05.0
JOINT_TYPE      revolute                # revolute or prismatic
Q_MIN           -2.8797933              # in radians (+/- 165 deg)
Q_MAX           2.8797933               # in radians
QD_MIN          -0.9                    # in rad/sec
QD_MAX          0.9                     # in rad/sec
T_MIN           -280.0                  # in Nm
T_MAX           280.0                   # in Nm
COULOMB         8.0                     # in Nm
VISCOUS         32.0                    # in Nmsec/rad
I_ARMATURE      8.0                     # in kgm2
POWER_PAR       0.001                   # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.642445e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1172407
TACHO_OFFSET    -0.94938

# end with the last transformation matrix
MASS            5.8                                         # in kg
CENTER_MASS     0.0     0.0     -0.0563                     # in m
INERTIA_MOMENT  0.04    0.04    0.006   0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  -1.0  0.0  0.0 0.0  -1.0  0.0 0.0 -
0.2037
CAD_MODEL       sn05.1
EOF
```

File sn08.cfig:

```
# This cfig file describes an RMMS pointer module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#=================================================================
# sn08 | pointer      | 0 | length = 163 mm
SERIAL_NR       sn08
MOD_TYPE        end-effector
AXIAL_SYM       yes
MOD_NDOF        0

# end with the last transformation matrix
MASS            .2                                          # in kg
CENTER_MASS     0.0     0.0     0.02                        # in m
INERTIA_MOMENT  0       0       0       0       0       0       # in kgm2
```

```
T_MATRIX        1.0  0.0  0.0  0.0  1.0  0.0  0.0 0.0  1.0  0.0 0.0 0.163
CAD_MODEL       sn08.0
EOF
```

A.2 The Space Shuttle Modules

The next set of modules are used for the satellite docking example in Chapter 7.

File sn50.cfig

```
# This cfig file describes a Space Shuttle pivot joint module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#=============================================================
# sn50 | pivot  joint | 1 | Tmax = 400 Nm;  Qmin/max = -/+ 165 deg
SERIAL_NR       sn50
MOD_TYPE        module
AXIAL_SYM       no
MOD_NDOF        1

# for every DOF include the following data:
MASS            6.0                                         # in kg
CENTER_MASS     0.078   0.0     0.325                       # in m
INERTIA_MOMENT  0.08    0.08    0.04    0       0       0       # in kgm2
T_MATRIX        0  0  1  0  1 0  -1  0  0  0.1373  0  0.365
CAD_MODEL       sn50.0
JOINT_TYPE      revolute              # revolute or prismatic
Q_MIN           -2.8797933            # in radians (+/- 165 deg)
Q_MAX           2.8797933             # in radians
QD_MIN          -0.9                  # in rad/sec
QD_MAX          0.9                   # in rad/sec
T_MIN           -400.0                # in Nm
T_MAX           400.0                 # in Nm
COULOMB         8.0                   # in Nm
VISCOUS         32.0                  # in Nmsec/rad
I_ARMATURE      8.0                   # in kgm2
POWER_PAR       0.001                 # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.5922903e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1406336
TACHO_OFFSET    -0.9400418

# end with the last transformation matrix
MASS            6.0                                         # in kg
CENTER_MASS     0.04    0.0     -0.06                       # in m
INERTIA_MOMENT  0.04    0.04    0.08    0       0       0       # in kgm2
T_MATRIX        0  0  -1  0  1  0  1 0  0  0.289 0 -0.1373
CAD_MODEL       sn50.1
EOF
```

File sn51.cfig:

```
# This cfig file describes a Space Shuttle pivot joint module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn51 | pivot  joint | 1 | Tmax = 600 Nm;  Qmin/max = -/+ 165 deg
SERIAL_NR       sn51
MOD_TYPE        module
AXIAL_SYM       no
MOD_NDOF        1

# for every DOF include the following data:
MASS            8.0                                          # in kg
CENTER_MASS     0.078   0.0     0.325                        # in m
INERTIA_MOMENT  0.1     0.1     0.05    0       0       0    # in kgm2
T_MATRIX        0  0  1  0  1 0  -1  0  0  0.1373  0  0.365
CAD_MODEL       sn51.0
JOINT_TYPE      revolute                # revolute or prismatic
Q_MIN           -2.8797933              # in radians (+/- 165 deg)
Q_MAX           2.8797933               # in radians
QD_MIN          -0.9                    # in rad/sec
QD_MAX          0.9                     # in rad/sec
T_MIN           -600.0                  # in Nm
T_MAX           600.0                   # in Nm
COULOMB         8.0                     # in Nm
VISCOUS         32.0                    # in Nmsec/rad
I_ARMATURE      10.0                     # in kgm2
POWER_PAR       0.001                   # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.5922903e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1406336
TACHO_OFFSET    -0.9400418

# end with the last transformation matrix
MASS            8.0                                          # in kg
CENTER_MASS     0.04    0.0     -0.06                        # in m
INERTIA_MOMENT  0.05    0.05    0.1     0       0       0    # in kgm2
T_MATRIX        0  0  -1  0  1  0  1 0  0  0.289 0 -0.1373
CAD_MODEL       sn51.1
EOF
```

File sn52.cfig:

```
# This cfig file describes a Space Shuttle pivot joint module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn52 | pivot  joint | 1 | Tmax = 600 Nm;  Qmin/max = -/+ 165 deg
SERIAL_NR       sn52
MOD_TYPE        module
```

```
AXIAL_SYM       no
MOD_NDOF        1

# for every DOF include the following data:
MASS            8.0                                          # in kg
CENTER_MASS     0.078   0.0     0.325                        # in m
INERTIA_MOMENT  0.1     0.1     0.05    0       0       0    # in kgm2
T_MATRIX        0  0  1  0  1 0  -1  0  0  0.1373  0  0.365
CAD_MODEL       sn52.0
JOINT_TYPE      revolute                # revolute or prismatic
Q_MIN           -2.8797933              # in radians (+/- 165 deg)
Q_MAX           2.8797933               # in radians
QD_MIN          -0.9                    # in rad/sec
QD_MAX          0.9                     # in rad/sec
T_MIN           -600.0                  # in Nm
T_MAX           600.0                   # in Nm
COULOMB         8.0                     # in Nm
VISCOUS         32.0                    # in Nmsec/rad
I_ARMATURE      10.0                     # in kgm2
POWER_PAR       0.001                   # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.5922903e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1406336
TACHO_OFFSET    -0.9400418

# end with the last transformation matrix
MASS            8.0                                          # in kg
CENTER_MASS     0.04    0.0     -0.06                        # in m
INERTIA_MOMENT  0.05    0.05    0.1     0       0       0    # in kgm2
T_MATRIX        0  0  -1  0  1  0  1 0  0  0.289 0 -0.1373
CAD_MODEL       sn52.1
EOF
```

File sn53.cfig:

```
# This cfig file describes a Space Shuttle pivot joint module
# we need the following 3 line as a description for TELEGRIP
# name   | type        |ndf| info
#==============================================================
# sn53 | pivot  joint | 1 | Tmax = 800 Nm;  Qmin/max = -/+ 165 deg
SERIAL_NR       sn53
MOD_TYPE        module
AXIAL_SYM       no
MOD_NDOF        1

# for every DOF include the following data:
MASS            10.0                                         # in kg
CENTER_MASS     0.078   0.0     0.325                        # in m
INERTIA_MOMENT  0.12    0.12    0.07    0       0       0    # in kgm2
T_MATRIX        0  0  1  0  1 0  -1  0  0  0.1373  0  0.365
CAD_MODEL       sn53.0
JOINT_TYPE      revolute                # revolute or prismatic
```

```
Q_MIN           -2.8797933              # in radians (+/- 165 deg)
Q_MAX           2.8797933               # in radians
QD_MIN          -0.9                    # in rad/sec
QD_MAX          0.9                     # in rad/sec
T_MIN           -800.0                  # in Nm
T_MAX           800.0                   # in Nm
COULOMB         8.0                     # in Nm
VISCOUS         32.0                    # in Nmsec/rad
I_ARMATURE      12.0                     # in kgm2
POWER_PAR       0.001                   # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.5922903e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1406336
TACHO_OFFSET    -0.9400418

# end with the last transformation matrix
MASS            10.0                                        # in kg
CENTER_MASS     0.04    0.0     -0.06                       # in m
INERTIA_MOMENT  0.07    0.07    0.12    0       0       0       # in kgm2
T_MATRIX        0  0  -1  0  1  0  1 0  0  0.289 0 -0.1373
CAD_MODEL       sn53.1
EOF
```

File sn60.cfig:

```
# This cfig file describes a Space Shuttle rotate joint module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#===================================================================
# sn60 | rotate joint | 1 | Tmax = 400 Nm;  Qmin/max = -/+ 165 deg
SERIAL_NR       sn60
MOD_TYPE        module
AXIAL_SYM       yes
MOD_NDOF        1

# for every DOF include the following data:
MASS            6                                           # in kg
CENTER_MASS     0.0     0.0     0.28                        # in m
INERTIA_MOMENT  0.16    0.16    0.01    0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  -1.0 0.0  0.0  0.0  -1.0  0.0  0.0
0.4074
CAD_MODEL       sn60.0
JOINT_TYPE      revolute                # revolute or prismatic
Q_MIN           -2.8797933              # in radians (+/- 165 deg)
Q_MAX           2.8797933               # in radians
QD_MIN          -0.9                    # in rad/sec
QD_MAX          0.9                     # in rad/sec
T_MIN           -400.0                  # in Nm
T_MAX           400.0                   # in Nm
COULOMB         8.0                     # in Nm
VISCOUS         32.0                    # in Nmsec/rad
I_ARMATURE      8.0                     # in kgm2
```

```
POWER_PAR       0.001                   # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.642445e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1172407
TACHO_OFFSET    -0.94938

# end with the last transformation matrix
MASS            6.0                                             # in kg
CENTER_MASS     0.0     0.0     -0.1126                         # in m
INERTIA_MOMENT  0.16    0.16    0.024   0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  -1.0  0.0  0.0 0.0  -1.0  0.0 0.0 -
0.4074
CAD_MODEL       sn60.1
EOF
```

File sn61.cfig:

```
# This cfig file describes a Space Shuttle rotate joint module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn61 | rotate joint | 1 | Tmax = 600 Nm;  Qmin/max = -/+ 165 deg
SERIAL_NR       sn61
MOD_TYPE        module
AXIAL_SYM       yes
MOD_NDOF        1

# for every DOF include the following data:
MASS            8                                               # in kg
CENTER_MASS     0.0     0.0     0.28                            # in m
INERTIA_MOMENT  0.2     0.2     0.015   0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  -1.0 0.0  0.0  0.0  -1.0  0.0  0.0
0.4074
CAD_MODEL       sn61.0
JOINT_TYPE      revolute                # revolute or prismatic
Q_MIN           -2.8797933              # in radians (+/- 165 deg)
Q_MAX           2.8797933               # in radians
QD_MIN          -0.9                    # in rad/sec
QD_MAX          0.9                     # in rad/sec
T_MIN           -600.0                  # in Nm
T_MAX           600.0                   # in Nm
COULOMB         8.0                     # in Nm
VISCOUS         32.0                    # in Nmsec/rad
I_ARMATURE      10.0                     # in kgm2
POWER_PAR       0.001                   # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.642445e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1172407
TACHO_OFFSET    -0.94938
```

```
# end with the last transformation matrix
MASS            8.0                                         # in kg
CENTER_MASS     0.0     0.0     -0.1126                     # in m
INERTIA_MOMENT  0.2     0.2     0.03    0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  -1.0  0.0  0.0 0.0  -1.0  0.0 0.0 -
0.4074
CAD_MODEL       sn61.1
EOF
```

File sn62.cfig:

```
# This cfig file describes a Space Shuttle rotate joint module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn62 | rotate joint | 1 | Tmax = 600 Nm;  Qmin/max = -/+ 165 deg
SERIAL_NR       sn62
MOD_TYPE        module
AXIAL_SYM       yes
MOD_NDOF        1

# for every DOF include the following data:
MASS            8                                           # in kg
CENTER_MASS     0.0     0.0     0.28                        # in m
INERTIA_MOMENT  0.2     0.2     0.015   0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  -1.0 0.0  0.0  0.0  -1.0  0.0  0.0
0.4074
CAD_MODEL       sn62.0
JOINT_TYPE      revolute                # revolute or prismatic
Q_MIN           -2.8797933              # in radians (+/- 165 deg)
Q_MAX           2.8797933               # in radians
QD_MIN          -0.9                    # in rad/sec
QD_MAX          0.9                     # in rad/sec
T_MIN           -600.0                  # in Nm
T_MAX           600.0                   # in Nm
COULOMB         8.0                     # in Nm
VISCOUS         32.0                    # in Nmsec/rad
I_ARMATURE      10.0                     # in kgm2
POWER_PAR       0.001                   # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.642445e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1172407
TACHO_OFFSET    -0.94938

# end with the last transformation matrix
MASS            8.0                                         # in kg
CENTER_MASS     0.0     0.0     -0.1126                     # in m
INERTIA_MOMENT  0.2     0.2     0.03    0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  -1.0  0.0  0.0 0.0  -1.0  0.0 0.0 -
0.4074
CAD_MODEL       sn62.1
EOF
```

File sn63.cfig:

```
# This cfig file describes a Space Shuttle rotate joint module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#===================================================================
# sn63 | rotate joint | 1 | Tmax = 800 Nm;  Qmin/max = -/+ 165 deg
SERIAL_NR       sn63
MOD_TYPE        module
AXIAL_SYM       yes
MOD_NDOF        1

# for every DOF include the following data:
MASS            10                                           # in kg
CENTER_MASS     0.0     0.0     0.28                         # in m
INERTIA_MOMENT  0.25    0.25    0.02    0       0       0    # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  -1.0 0.0  0.0  0.0  -1.0  0.0  0.0
0.4074
CAD_MODEL       sn63.0
JOINT_TYPE      revolute                # revolute or prismatic
Q_MIN           -2.8797933              # in radians (+/- 165 deg)
Q_MAX           2.8797933               # in radians
QD_MIN          -0.9                    # in rad/sec
QD_MAX          0.9                     # in rad/sec
T_MIN           -800.0                  # in Nm
T_MAX           800.0                   # in Nm
COULOMB         8.0                     # in Nm
VISCOUS         32.0                    # in Nmsec/rad
I_ARMATURE      12.0                     # in kgm2
POWER_PAR       0.001                   # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.642445e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1172407
TACHO_OFFSET    -0.94938

# end with the last transformation matrix
MASS            10.0                                         # in kg
CENTER_MASS     0.0     0.0     -0.1126                      # in m
INERTIA_MOMENT  0.25    0.25    0.036   0       0       0    # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  -1.0  0.0  0.0 0.0  -1.0  0.0 0.0 -
0.4074
CAD_MODEL       sn63.1
EOF
```

File sn70.cfig:

```
# This cfig file describes a Space Shuttle link module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#===================================================================
# sn70 | link           | 0 | Length = 0.25m
```

```
SERIAL_NR       sn70
MOD_TYPE        module
AXIAL_SYM       yes
MOD_NDOF        0

# end with the last transformation matrix
MASS            2.5                                         # in kg
CENTER_MASS     0.0     0.0     0.125                       # in m
INERTIA_MOMENT  0.23    0.23    0.05    0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0 0.0  0.0  0.0  1.0  0.0  0.0  0.25
CAD_MODEL       sn70.0
EOF
```

File sn71.cfig:

```
# This cfig file describes a Space Shuttle link module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn71 | link           | 0 | Length = 0.25m
SERIAL_NR       sn71
MOD_TYPE        module
AXIAL_SYM       yes
MOD_NDOF        0

# end with the last transformation matrix
MASS            2.5                                         # in kg
CENTER_MASS     0.0     0.0     0.125                       # in m
INERTIA_MOMENT  0.23    0.23    0.05    0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0 0.0  0.0  0.0  1.0  0.0  0.0  0.25
CAD_MODEL       sn71.0
EOF
```

File sn72.cfig:

```
# This cfig file describes a Space Shuttle link module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn72 | link           | 0 | Length = 0.5m
SERIAL_NR       sn72
MOD_TYPE        module
AXIAL_SYM       yes
MOD_NDOF        0

# end with the last transformation matrix
MASS            3.75                                        # in kg
CENTER_MASS     0.0     0.0     0.25                        # in m
INERTIA_MOMENT  0.7     0.7     0.075   0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0 0.0  0.0  0.0  1.0  0.0  0.0  0.5
```

```
CAD_MODEL       sn72.0
EOF
```

File sn73.cfig:

```
# This cfig file describes a Space Shuttle link module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#===================================================================
# sn73 | link           | 0 | Length = 1m
SERIAL_NR       sn73
MOD_TYPE        module
AXIAL_SYM       yes
MOD_NDOF        0

# end with the last transformation matrix
MASS            5.0                                         # in kg
CENTER_MASS     0.0     0.0     0.5                         # in m
INERTIA_MOMENT  2.8     2.8     0.1     0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0 0.0  0.0  0.0  1.0  0.0  0.0  1.0
CAD_MODEL       sn73.0
EOF
```

File sn74.cfig:

```
# This cfig file describes a Space Shuttle link module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#===================================================================
# sn74 | link           | 0 | Length = 2m
SERIAL_NR       sn74
MOD_TYPE        module
AXIAL_SYM       yes
MOD_NDOF        0

# end with the last transformation matrix
MASS            10.0                                        # in kg
CENTER_MASS     0.0     0.0     1.0                         # in m
INERTIA_MOMENT  20.0    20.0    0.2     0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0 0.0  0.0  0.0  1.0  0.0  0.0  2.0
CAD_MODEL       sn74.0
EOF
```

File sn75.cfig:

```
# This cfig file describes a Space Shuttle link module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn75 | link           | 0 | Length = 2m
SERIAL_NR       sn75
MOD_TYPE        module
AXIAL_SYM       yes
MOD_NDOF        0

# end with the last transformation matrix
MASS            10.0                                        # in kg
CENTER_MASS     0.0     0.0     1.0                         # in m
INERTIA_MOMENT  20.0    20.0    0.2     0       0       0   # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0 0.0  0.0  0.0  1.0  0.0  0.0  2.0
CAD_MODEL       sn75.0
EOF
```

File sn76.cfig:

```
# This cfig file describes a Space Shuttle link module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn76 | link           | 0 | Length = 2m
SERIAL_NR       sn76
MOD_TYPE        module
AXIAL_SYM       yes
MOD_NDOF        0

# end with the last transformation matrix
MASS            10.0                                        # in kg
CENTER_MASS     0.0     0.0     1.0                         # in m
INERTIA_MOMENT  20.0    20.0    0.2     0       0       0   # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0 0.0  0.0  0.0  1.0  0.0  0.0  2.0
CAD_MODEL       sn76.0
EOF
```

File sn77.cfig:

```
# This cfig file describes a Space Shuttle link module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn77 | link           | 0 | Length = 3m
SERIAL_NR       sn77
MOD_TYPE        module
AXIAL_SYM       yes
MOD_NDOF        0

# end with the last transformation matrix
```

```
MASS            15.0                                        # in kg
CENTER_MASS     0.0     0.0     1.5                         # in m
INERTIA_MOMENT  65.0    65.0    0.3     0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0 0.0  0.0  0.0  1.0  0.0  0.0  3.0
CAD_MODEL       sn77.0
EOF
```

File sn78.cfig:

```
# This cfig file describes a Space Shuttle link module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#===================================================================
# sn78 | link           | 0 | Length = 3m
SERIAL_NR       sn78
MOD_TYPE        module
AXIAL_SYM       yes
MOD_NDOF        0

# end with the last transformation matrix
MASS            15.0                                        # in kg
CENTER_MASS     0.0     0.0     1.5                         # in m
INERTIA_MOMENT  65.0    65.0    0.3     0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0 0.0  0.0  0.0  1.0  0.0  0.0  3.0
CAD_MODEL       sn78.0
EOF
```

File sn79.cfig:

```
# This cfig file describes a Space Shuttle link module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#===================================================================
# sn79 | link           | 0 | Length = 3m
SERIAL_NR       sn79
MOD_TYPE        module
AXIAL_SYM       yes
MOD_NDOF        0

# end with the last transformation matrix
MASS            15.0                                        # in kg
CENTER_MASS     0.0     0.0     1.5                         # in m
INERTIA_MOMENT  65.0    65.0    0.3     0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0 0.0  0.0  0.0  1.0  0.0  0.0  3.0
CAD_MODEL       sn79.0
EOF
```

File sn80.cfig:

```
# This cfig file describes a Space Shuttle corner link module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn80 | link 90deg     | 0 | L=200mm
SERIAL_NR       sn80
MOD_TYPE        module
AXIAL_SYM       no
MOD_NDOF        0

# end with the last transformation matrix
MASS            3.5                                          # in kg
CENTER_MASS     0.03    0.0     0.17                         # in m
INERTIA_MOMENT  0.03    0.04    0.03    0       0       0    # in kgm2
T_MATRIX        0.0  0.0  -1.0  0.0  1.0 0.0  1.0  0.0  0.0  0.2  0.0  0.2
CAD_MODEL       sn80.0
EOF
```

File sn81.cfig:

```
# This cfig file describes a Space Shuttle corner link module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn81 | link 90deg     | 0 | L=300mm
SERIAL_NR       sn81
MOD_TYPE        module
AXIAL_SYM       no
MOD_NDOF        0

# end with the last transformation matrix
MASS            4.5                                          # in kg
CENTER_MASS     0.06    0.0     0.24                         # in m
INERTIA_MOMENT  0.06    0.1     0.06    0       0       0    # in kgm2
T_MATRIX        0.0  0.0  -1.0  0.0  1.0 0.0  1.0  0.0  0.0  0.3  0.0  0.3
CAD_MODEL       sn81.0
EOF
```

File sn90.cfig:

```
# This cfig file describes a Space Shuttle wrist module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#==================================================================
# sn90 | wrist module | 3 | 3-roll wrist with parallel jaw gripper
SERIAL_NR       sn90
MOD_TYPE        end-effector
```

```
AXIAL_SYM       no
MOD_NDOF        3

# for every DOF include the following data:
# for DOF 1:
MASS            12.0
CENTER_MASS     0.0     0.0     0.35                           # in m
INERTIA_MOMENT  0.5     0.5     0.02    0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0 0.0  0.0  0.0  1.0  0.0  0.0  0.5
CAD_MODEL       sn90.0
JOINT_TYPE      revolute                # revolute or prismatic
Q_MIN           -2.8797933              # in radians (+/- 165 deg)
Q_MAX           2.8797933               # in radians
QD_MIN          -0.9                    # in rad/sec
QD_MAX          0.9                     # in rad/sec
T_MIN           -150.0                  # in Nm
T_MAX           150.0                   # in Nm
COULOMB         7.0                     # in Nm
VISCOUS         25.0                    # in Nmsec/rad
I_ARMATURE      7.0                     # in kgm2
POWER_PAR       0.001                   # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.5922903e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1406336
TACHO_OFFSET    -0.9400418

# for DOF 2:
MASS            2.0                                             # in kg
CENTER_MASS     0.0     0.0     0.0                             # in m
INERTIA_MOMENT  0.02    0.02    0.01    0       0       0       # in kgm2
T_MATRIX        0.0  0.0  1.0  0.0  1.0 0.0  -1.0  0.0  0.0  0.0  0.0  0.0
CAD_MODEL       sn90.1
JOINT_TYPE      revolute                # revolute or prismatic
Q_MIN           -1.7453293              # in radians
Q_MAX           1.7453293               # in radians
QD_MIN          -0.9                    # in rad/sec
QD_MAX          0.9                     # in rad/sec
T_MIN           -150.0                  # in Nm
T_MAX           150.0                   # in Nm
COULOMB         7.0                     # in Nm
VISCOUS         25.0                    # in Nmsec/rad
I_ARMATURE      7.0                     # in kgm2
POWER_PAR       0.001                   # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.5922903e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1406336
TACHO_OFFSET    -0.9400418

# for DOF 3:
MASS            2.0                                             # in kg
CENTER_MASS     0.0     0.0     0.0                             # in m
INERTIA_MOMENT  0.02    0.02    0.01    0       0       0       # in kgm2
T_MATRIX        0.0  0.0  -1.0  0.0  1.0 0.0  1.0  0.0  0.0  0.0  0.0  0.0
CAD_MODEL       sn90.2
```

```
JOINT_TYPE      revolute                # revolute or prismatic
Q_MIN           -4.6425758              # in radians
Q_MAX           4.6425758               # in radians
QD_MIN          -0.9                    # in rad/sec
QD_MAX          0.9                     # in rad/sec
T_MIN           -150.0                  # in Nm
T_MAX           150.0                   # in Nm
COULOMB         7.0                     # in Nm
VISCOUS         25.0                    # in Nmsec/rad
I_ARMATURE      7.0                     # in kgm2
POWER_PAR       0.001                   # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.5922903e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1406336
TACHO_OFFSET    -0.9400418

# end with the last transformation matrix
MASS            4.0                                             # in kg
CENTER_MASS     0.0     0.0     0.15                            # in m
INERTIA_MOMENT  0.04    0.04    0.01    0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0  0.0  0.0 0.0  1.0  0.0 0.0 0.5
CAD_MODEL       sn90.3
EOF
```

File sn91.cfig:

```
# This cfig file describes a Space Shuttle wrist module
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#===================================================================
# sn91 | wrist module | 3 | 3-roll wrist with parallel jaw gripper
SERIAL_NR       sn91
MOD_TYPE        end-effector
AXIAL_SYM       no
MOD_NDOF        3

# for every DOF include the following data:
# for DOF 1:
MASS            14.5
CENTER_MASS     0.0     0.0     0.70                            # in m
INERTIA_MOMENT  0.5     0.5     0.02    0       0       0       # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0 0.0  0.0  0.0  1.0  0.0  0.0  1.0
CAD_MODEL       sn91.0
JOINT_TYPE      revolute                # revolute or prismatic
Q_MIN           -2.8797933              # in radians (+/- 165 deg)
Q_MAX           2.8797933               # in radians
QD_MIN          -0.9                    # in rad/sec
QD_MAX          0.9                     # in rad/sec
T_MIN           -150.0                  # in Nm
T_MAX           150.0                   # in Nm
COULOMB         7.0                     # in Nm
VISCOUS         25.0                    # in Nmsec/rad
```

```
I_ARMATURE      7.0                     # in kgm2
POWER_PAR       0.001                   # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.5922903e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1406336
TACHO_OFFSET    -0.9400418

# for DOF 2:
MASS            2.0                                             # in kg
CENTER_MASS     0.0     0.0     0.0                             # in m
INERTIA_MOMENT  0.02    0.02    0.01    0       0       0       # in kgm2
T_MATRIX        0.0  0.0  1.0  0.0  1.0 0.0  -1.0  0.0  0.0  0.0  0.0  0.0
CAD_MODEL       sn91.1
JOINT_TYPE      revolute                # revolute or prismatic
Q_MIN           -1.7453293              # in radians
Q_MAX           1.7453293               # in radians
QD_MIN          -0.9                    # in rad/sec
QD_MAX          0.9                     # in rad/sec
T_MIN           -150.0                  # in Nm
T_MAX           150.0                   # in Nm
COULOMB         7.0                     # in Nm
VISCOUS         25.0                    # in Nmsec/rad
I_ARMATURE      7.0                     # in kgm2
POWER_PAR       0.001                   # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.5922903e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1406336
TACHO_OFFSET    -0.9400418

# for DOF 3:
MASS            2.0                                             # in kg
CENTER_MASS     0.0     0.0     0.0                             # in m
INERTIA_MOMENT  0.02    0.02    0.01    0       0       0       # in kgm2
T_MATRIX        0.0  0.0  -1.0  0.0  1.0 0.0  1.0  0.0  0.0  0.0  0.0  0.0
CAD_MODEL       sn91.2
JOINT_TYPE      revolute                # revolute or prismatic
Q_MIN           -4.6425758              # in radians
Q_MAX           4.6425758               # in radians
QD_MIN          -0.9                    # in rad/sec
QD_MAX          0.9                     # in rad/sec
T_MIN           -150.0                  # in Nm
T_MAX           150.0                   # in Nm
COULOMB         7.0                     # in Nm
VISCOUS         25.0                    # in Nmsec/rad
I_ARMATURE      7.0                     # in kgm2
POWER_PAR       0.001                   # in sec/(kgm2)
MOTOR_SCALE     0.03448
RESOLVER_SCALE  9.587379e-5
TACHO_SCALE     4.5922903e-4
MOTOR_OFFSET    -1130.0
RESOLVER_OFFSET -3.1406336
TACHO_OFFSET    -0.9400418

# end with the last transformation matrix
```

```
MASS            4.0                                         # in kg
CENTER_MASS     0.0     0.0     0.15                        # in m
INERTIA_MOMENT  0.04    0.04    0.01    0       0       0   # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0  0.0  0.0 0.0  1.0  0.0 0.0 0.5
CAD_MODEL       sn91.3
EOF
```

File sn98.cfig:

```
# This cfig file describes the Space Shuttle as a manipulator base
# we need the following 3 line as a description for TELEGRIP
# name   | type         |ndf| info
#===================================================================
# sn98 | base         | 0 | space shuttle
SERIAL_NR       sn98
MOD_TYPE        base
AXIAL_SYM       no
MOD_NDOF        0

# end with the last transformation matrix
MASS            50000.0                                     # in kg
CENTER_MASS     0.0     0.0     0.0                         # in m
INERTIA_MOMENT  0       0       0       0       0       0   # in kgm2
T_MATRIX        1.0  0.0  0.0  0.0  1.0  0.0  0.0 0.0  1.0  0.0 0.0 0.0
CAD_MODEL       sn98.0
```

Bibliography

[1] ARCNET Trade Association. 1992. *ANSI/ATA 878.1—Local Area Network: Token Bus (version 1.10*)

[2] Baluja, S. 1995. An Empirical Comparison of Seven Iterative and Evolutionary Function Optimization Heuristics. Technical Report CMU–CS–95–193. Computer Science Department, Carnegie Mellon University.

[3] Burdick, J. W. 1988. Kinematic Analysis and Design of Redundant Robot Manipulators. Stanford Computer Science Report no. STAN-CS-88-1207.

[4] Burdick, J. W. 1992 (May 12–14, Nice, France). A recursive method for finding revolute-jointed manipulator singularities. *Proceedings of the 1992 IEEE International Conference on Robotics and Automation.* Los Alamitos, CA: IEEE, pp. 448–453.

[5] Benhabib, B., and Dai, M. Q. 1991. Mechanical Design of a Modular Robot for Industrial Applications. *Journal of Manufacturing Systems*. Vol. 10. No. 4. pp. 297–306.

[6] Carriker, W. F. 1995. A Rapid Prototyping System for Flexible Assembly. Ph.D. Thesis. Department of Electrical and Computer Engineering. Carnegie Mellon University.

[7] Chedmail, P., and Ramstein, E. 1996 (April, Minneapolis, MN). Robot Mechanism Synthesis and Genetic Algorithms. *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*. Vol. 4. Los Alamitos, CA: IEEE, pp. 3466–3471.

[8] Chen, I-M., and Burdick, J. W. 1995 (May 21–27, Nagoya, Japan). Determining Task Optimal Modular Robot Assembly Configurations. *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, Vol. 1. Los Alamitos, CA: IEEE, pp. 132–137.

[9] Chen, I-M. 1994. Theory and Application of Modular Reconfigurable Robotic Systems. Ph.D. thesis. Department of Mechanical Engineering. California Institute of Technology.

[10] Chow, E. Y., and Willsky, A. S. 1984. Analytical redundancy and the design of robust failure detection systems. *IEEE Transactions on Automation and Control.* Vol. 29. No. 7. pp. 603–614.

[11] Cohen, R. et al. 1992. Conceptual Design of a Modular Robot. *Transactions of the ASME: Journal of Mechanical Design*. Vol. 114. pp. 117–125.

[12] Cole, J. R. 1995. Rapid Generation of Motion Plans for Modular Robotic Systems. M.S Thesis. Department of Mechanical Engineering, Massachusetts Institute of Technology.

[13] Davis, L. 1985. Applying Adaptive Algorithms to Epistatic Domains. *Proceedings of the International Joint Conference on Artificial Intelligence.* pp. 162–164.

[14] English, J. D., and Maciejewski, A. A. 1996 (April 22–28, Minneapolis, MN). Fault Tolerance for Kinematically Redundant Manipulators: Anticipating Free-Swinging Joint Failures. *Proceedings of the 1996 IEEE International Conference on Robotics and Automation.* Vol. 1. Los Alamitos, CA: IEEE, pp. 460–467.

[15] Farritor, S., Dubowsky, S., Rutman, N., and Cole, J. 1996 (April, Minneapolis, MN). A System-Level Modular Design Approach to Field Robotics. *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*. Vol. 4. Los Alamitos, CA: IEEE, pp. 2890–2895.

[16] Fletcher, R. 1987. *Practical Methods of Optimization (Second Edition).* New York, NY: John Wiley & Sons.

[17] Fukuda, T., et al. 1992. Concept of cellular robotic system (CEBOT) and basic strategies for its realization. *Computers and Electrical Engineering.* Vol. 18. No 1. pp. 11–39.

[18] Geist, A. et al. 1994. *PVM (Parallel Virtual Machine): A Users' Guide and Tutorial for Network Parallel Computing*. Scientific and Engineering Computation Series. Cambridge, MA: The MIT Press.

[19] Gertz, M. W., and Khosla, P. K. 1994 (June, New Orleans, LA). Onika: A Multilevel Human-Machine Interface for Real-Time Sensor-Based Robotic Systems. *Proceedings of the 1994 Annual Meeting of the American Nuclear Society*.

[20] Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley.

[21] Goldberg, D. E., Deb, K., and Korb, B. 1991. Do not Worry, Be Messy. *Proceedings of the Fourth International Conference on Genetic Algorithms.* Eds: Belew and Booker. Los Altos, Ca: Morgan Kaufmann Publishers, pp. 24–30.

[22] Golub, G. H., and Van Loan, C. F. 1989. *Matrix Computations (second edition),* Baltimore: The Johns Hopkins University Press.

[23] Gottschalk, S., Lin, M. C., and Manocha, D. 1996 (August 4–9, New Orleans, LA). OBB-Tree: A Hierarchical Structure for Rapid Interference Detection. *to appear in Proceedings of ACM SIGGRAPH '96.*

[24] Gupta, K. C., and Roth, B. 1982. Design Considerations for Manipulator Workspace. *Transactions of the ASME, Journal of Mechanical Design*. Vol. 104. pp. 704–711.

[25] Holland, J. H., 1975. *Adaptation in Natural and Artificial Systems,* Ann Arbor, MI: University of Michigan Press.

[26] Hollander, M. and Wolfe, D. A. 1973. *Nonparametric Statistical Methods.* Wiley Series in Probability and Mathematical Statistics. New York, NY: John Wiley & Sons.

[27] Hui, R. et al. 1993 (May 2–6, Atlanta, GA). Design of the IRIS Facility—a Modular, Reconfigurable and Expandable Robot Test Bed. *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*. Los Alamitos, CA: IEEE, pp. 155–160.

[28] Johnson, B.W. 1989. *Design and analysis of fault-tolerant digital systems*. Reading, Mass.: Addison-Wesley.

[29] Juels, A., and Wattenberg, M. 1994. Stochastic Hillclimbing as a Baseline Method for Evaluating Genetic Algorithms. Technical Report CSD–94–834. Computers Science Department, University of California at Berkeley.

[30] Kelmar, L., and Khosla, P. K. 1990. Automatic Generation of Forward and Inverse Kinematics for a Reconfigurable Modular Manipulator System. *Journal of Robotic Systems*. Vol. 7. No. 4. pp. 599–619.

[31] Kim, J.-O. 1992. Task Based Kinematic Design of Robot Manipulators. Ph.D. Thesis. The Robotics Institute, Carnegie Mellon University.

[32] Kim, J.-O., and Khosla, P. K. 1991. Dexterity Measures for Design and Control of Manipulators. *Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems (IROS'91*). pp. 758–763.

[33] Kim, J.-O., and Khosla, P. K. 1992 (July 7–10, Raleigh, NC). A Multi-Population Genetic Algorithm and Its Application to Design of Manipulators. *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'92).*

[34] Kim, J.-O., and Khosla, P. K. 1993 (May, Atlanta, GA). Design of Space Shuttle Tile Servicing Robot: an Application of Task Based Kinematic Design. *Proceedings of the 1993 IEEE International Conference on Robotics and Automation,* Vol. 3, pp. 867–874.

[35] Kirkpatrick, S., Gelatt Jr., C. D., Vecchi, M. P. 1983. Optimization by Simulated Annealing. *Science.* Vol 220. No. 4598, pp 671–680.

[36] Kotosaka, S. et al. 1992 (September 21–22, Saitama, Japan). Development of a Functionally Adaptive and Robust Manipulator. *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems.* pp. 85–90.

[37] Koza, J. R. 1992. *Genetic Programming: on the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press.

[38] Krishnan, A. 1989. A Methodology to Determine the Dynamic Configuration of a Reconfigurable Manipulator. M.S. Thesis. Electrical and Computer Engineering Department, Carnegie Mellon University.

[39] Lewis, C. L., and Maciejewski, A. A. 1994a. Dexterity Optimization of Kinematically Redundant Manipulators in the Presence of Joint Failures. *Computers and Electrical Engineering*. Vol. 20. No. 3. pp. 273–288.

[40] Lewis, C. L., and Maciejewski, A. A. 1994b (May, San Diego, CA). An Example of Failure Tolerant Operation of a Kinematically Redundant Manipulator. *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, Los Alamitos, CA: IEEE, pp 1380–1387.

[41] Lin, C.-C. D., and Freudenstein, F. 1986. Optimization of the Workspace of a three-Link Turning-Pair Connected Robot Arm. *International Journal of Robotics Research*. Vol. 5. No. 2, pp. 104–111.

[42] Lück, C. L., and Lee, S. 1994 (May, San Diego, CA). Global Path Planning of Redundant Manipulators Based on Self-Motion Topology. *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. Los Alamitos, CA: IEEE, pp. 372–377.

[43] Manoochehri, S. and Seireg, A. A. 1990. A Computer-Based Methodology for the Form Synthesis and Optimal Design of Robot Manipulators. *Transactions of the ASME, Journal of Mechanical Design*. Vol. 112, pp. 501–508.

[44] Matsumaru, T. 1995 (May 21–27, Nagoya, Japan). Design and Control of the Modular Robot System: TOMMS. *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*. Los Alamitos, CA: IEEE, pp. 2125–2131.

[45] Michalewicz, Z. 1994. *Genetic Algorithms + Data Structures = Evolution Programs (second edition),* New York, NY: Springer Verlag.

[46] Morrow, J. D., and Khosla, P. K. 1995 (May 21–27, Nagoya, Japan). Sensorimotor Primitives for Robotic Assembly skills. *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, Los Alamitos, CA: IEEE.

[47] Morrow, J. D., Nelson, B. J., and Khosla, P. K. 1995 (August 5–9, Pittsburgh, PA). Vision and Force Driven Sensorimotor Primitives for Robotic Assembly Skills. *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS95)*. Vol. 2. Los Alamitos, CA: IEEE, pp. 90–95.

[48] Mühlenbein, H. 1992. Parallel Genetic Algorithm in Combinatorial Optimization. *Computer Science and Operations Research*. Eds.: O. Balcki, R. Shandra, and S. Zenios. New York, NY: Pergamon Press, pp. 441–456.

[49] Murthy, S. S. 1992. Synergy in Cooperating Agents: Designing Manipulators from Task Specifications. Ph.D. Thesis. Department of Electrical and Computer Engineering, Carnegie Mellon University.

[50] Murthy, S. S., Khosla, P. K., and Talukdar, S. N. 1993 (Nagoya, Japan). Designing Manipulators from Task Requirements: An Asynchronous Team Approach. *Proceedings of the 1st WWW Workshop on Multiple Distributed Robotic Systems*.

[51] Nakamura, Y., and Hanafusa, H. 1986. Inverse Kinematic Solutions with Singularity Robustness for Robot Manipulator Control. *Journal of Dynamic Systems, Measurement, and Control.* Vol. 108. pp. 163–171.

[52] Nenchev, D. N. 1989. Redundancy Resolution through Local Optimization: A Review. *Journal of Robotic Systems*. Vol. 6. No. 6. pp. 769–798.

[53] Paden, B., and Sastry, S. 1988. Optimal Kinematic Design of 6R Manipulators. *International Journal of Robotics Research*, Vol. 7. No. 2. pp. 43–61.

[54] Pahl, G., and Beitz, W. 1996. *Engineering Design: A Systematic Approach (second edition)*. London, UK: Springer-Verlag.

[55] Paredis, C. J. J. 1990. An Approach for Mapping Kinematic Task Specifications into a Manipulator Design. M.S. Thesis. Department of Electrical and Computer Engineering, Carnegie Mellon University.

[56] Paredis, C. J. J., and Khosla, P. K. 1993. Kinematic Design of Serial Link Manipulators From Task Specifications. *International Journal of Robotics Research*. Vol. 12. No. 3. pp. 274–286.

[57] Paredis, C. J. J., and Khosla, P. K. 1994 (May 8–13, San Diego, CA). Mapping Tasks into Fault Tolerant Manipulators. *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. Los Alamitos, CA: IEEE, pp. 696–703.

[58] Paredis, C. J. J., and Khosla, P. K. 1996. Fault Tolerant Task Execution through Global Trajectory Planning. *Reliability Engineering and System Safety (Special Issue on Safety of Robotic Systems)*. In Press.

[59] Paredis, C. J. J., Brown, H. B., and Khosla, P. K. 1996 (April 22–28, Minneapolis, MN). A Rapidly Deployable Manipulator System. *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*. Vol. 2. Los Alamitos, CA: IEEE, pp. 1434–1439.

[60] Pradeep, A. K., et al. 1988. Crippled motion in robots. *IEEE Transactions on Aerospace and Electronic Systems.* Vol. 24. No. 1. pp. 2–13.

[61] Raghavan, M., and Roth, B. 1993. Inverse Kinematics of the General 6R Manipulator and Related Linkages. *Transactions of the ASME. Journal of Mechanical Design*. Vol. 115. No. 3. pp. 502–508.

[62] Roberts, R. G., and Maciejewski, A. A. 1996. A Local Measure of Fault Tolerance for Kinematically Redundant Manipulators. *IEEE Transactions on Robotics and Automation.* Vol. 12. No. 4. pp. 543–552.

[63] Roston, G. P. 1994. A Genetic Methodology for Configuration Design. Ph.D. Thesis. Department of Mechanical Engineering and The Robotics Institute, Carnegie Mellon University. (Also published as Technical Report CMU-RI-TR-94-42).

[64] Rutman, N. 1995. Automated Design of Modular Field Robots. M.S. Thesis. Department of Mechanical Engineering, Massachusetts Institute of Technology.

[65] Schwefel, H.-P. 1981. *Numerical Optimization for Computer Models*. Chichester, UK: John Wiley.

[66] Sims, K. 1994 (July 24–29, Orlando, FL). Evolving virtual creatures. Proceedings of 21st International SIGGRAPH Conference. New York, NY: ACM, pp. 15–22.

[67] Sreevijayan, D. 1992. On the Design of Fault-Tolerant Robotic Manipulator Systems, M.S. Thesis. Mechanical Engineering Department, The University of Texas at Austin.

[68] Stengel, R. F. 1988. Intelligent failure-tolerant control. *IEEE Control Systems Magazine.* Vol. 11. No. 4. pp. 2–13.

[69] Stewart, D. B., and Khosla, P. K. 1995 (August 5–9, Pittsburgh, PA). Rapid Development of Robotic Applications using Component-Based Real-Time Software. *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'95)*. Vol. 1. Los Alamitos, CA: IEEE, pp. 465–470.

[70] Stewart, D. B. 1994. *Real-Time Software Design and Analysis of Reconfigurable Multi-Sensor Based Systems*. Ph.D. Dissertation. Carnegie Mellon University, Department of Electrical and Computer Engineering.

[71] Strobel, R., and Johnson, A. 1993. Pocket Pagers in Lots of One. *IEEE Spectrum*. Vol. 30, No. 9, pp. 29–32.

[72] Suh, N. P. 1988. *The Principles of Design*. Oxford, UK: Oxford University Press.

[73] Talukdar et al. 1996. *Asynchronous Team Toolkit User's Guide.* Internal document. Carnegie Mellon University.

[74] Talukdar, S. N., de Souza, P. S., Murthy, S. S. 1993. Organizations for Computer-Based Agents. *International Journal of Engineering Intelligent Systems for Electrical Engineering and Communications*. Vol. 1, No. 2. pp. 75–87.

[75] Tanese, R. 1989. Distributed Genetic Algorithm. *Proceedings of the Third International Conference on Genetic Algorithms*. ed.: H. Schaffer, Morgan-Kaufmann, pp 434–440.

[76] Tesar, D., and Butler, M. S. 1989. A Generalized Modular Architecture for Robot Structures. *Manufacturing Review*. Vol 2. No. 2. pp. 91–118.

[77] Ting, Y., Tosunoglu, S., and Tesar, D. 1993 (May 2–6, Atlanta). A control structure for fault-tolerant operation of robotic manipulators. *Proc. 1993 IEEE Int. Conf. Robot. Autom.* Loa Alamitos, CA: IEEE, pp. 684–690.

[78] Tong, C., and Sriram, D. 1992. *Artificial Intelligence in Engineering Design, Volume 1: Design Representation and Models of Routine Design,* San Diego, CA: Academic Press.

[79] Törn, A., and Zilinskas, A. 1989. *Global Optimization.* Lecture Notes in Computer Science Vol. 350. Eds: Goos and Hartmanis. New York, NY: Springer Verlag.

[80] Tsai, L.-W., and Morgan, A. 1985. Solving the Kinematics of the Most General Six- and Five-Degree-of-Freedom Manipulators by Continuation Methods. *Transactions of the ASME, Journal of Mechanism, Transmissions, and Automation in Design*. Vol. 107, pp. 189–200.

[81] Tsai, Y.-C., and Soni, A. H. 1984. The Effect of Link Parameter on the Working Space of General 3R Robot Arms. *Mechanism and Machine Theory*. Vol. 19, No. 1, pp. 9–16.

[82] Vertut, J., and Liégois, A. 1981. General Design Criteria for Manipulators. *Mechanism and Machine Theory*. Vol. 16. pp. 65–70.

[83] Vijaykumar, R., Waldron, K. J., Tsai, M. J. 1986. Geometric Optimization of Serial Chain Manipulator Structures for Working Volume and Dexterity. *The International Journal on Robotics Research*. Vol. 5. No. 2. pp. 91–103.

[84] Visinsky, M. L., Walker, I. D., and Cavallaro, J. R. 1993 (May 2–6, Atlanta, GA). Layered dynamic fault detection and tolerance for robots. *Proc. 1993 IEEE Int. Conf. Robot. Autom.* Los Alamitos, Calif: IEEE, pp. 180–187.

[85] Visinsky, M. L., Walker, I. D., and Cavallaro, J. R. 1994 (May 8–13, San Diego, CA). New Dynamic Model-Based Fault Detection Thresholds for Robot Manipulators. *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. 1388-1395.

[86] von Neumann, J. 1956. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies (Annals of mathematics studies, no. 34)*, eds. C. E. Shannon and J. McCarthy. Princeton: Princeton University Press.

[87] Wu, E. C., Hwang, J. C., and Chladek, J. T. 1995. Fault-Tolerant Joint Development for the Space Shuttle Remote Manipulator System: Analysis and Experiment. *IEEE Transactions on Robotics and Automation.* Vol. 9. No. 5. pp. 675–684.

[88] Yoshikawa, T. 1985. Manipulability of Robotic Mechanisms. *The International Journal of Robotics Research.* Vol. 4. No. 2. pp. 3–9.