Agent-Based Design of Fault Tolerant Manipulators for Satellite Docking

Christiaan J.J. Paredis
paredis@cmu.edu
Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213

Pradeep K. Khosla
pkk@cs.cmu.edu
Dept. of Electrical and Computer Engineering,
The Robotics Institute,
Carnegie Mellon University,
Pittsburgh, PA 15213

Abstract:

A rapidly deployable fault tolerant manipulator system consists of modular hardware and support software that allow the user to quickly configure and deploy a fault tolerant manipulator that is custom-tailored for a given task. The main focus of this paper is on the Task Based Design component of such a system; that is, the determination of the optimal manipulator configuration, its base position, and the corresponding joint space trajectory for a given task. We introduce a novel agent-based solution approach to task based design and illustrate it with a fault tolerant manipulator design for a satellite docking operation aboard the space shuttle.

1 Introduction

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.

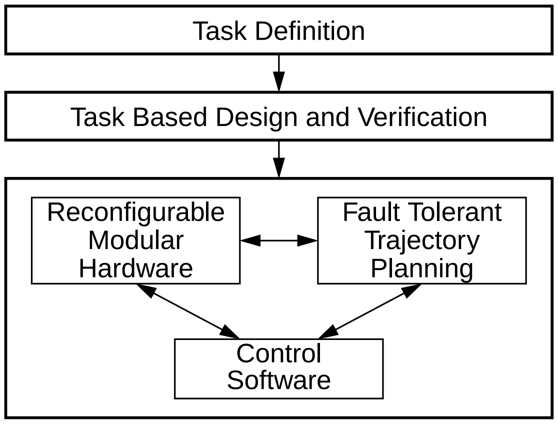
Task Definition
Task Based Design and Verification
Reconfigurable Modular Hardware
Fault Tolerant Trajectory Planning
Control Software

Figure 1: A rapidly deployable fault tolerant system.

To address this need for a more flexible and reliable manipulator, we propose the concept of a rapidly deployable fault tolerant manipulator system [11], as is illustrated in Figure 1. At the base of such a system is the Reconfigurable Modular Manipulator System (RMMS) [12]. The RMMS consists of an inventory of link and joint modules of different sizes and performance specifications. Quick coupling connectors allow the modules to be rapidly configured into a wide variety of manipulator structures; each module can be connected in 8 different orientations (@45deg angles) relative to the previous module. The inventory of modules considered in this paper contains 23 modules of 5 different types as shown in Figure 2.

To achieve rapid deployability, the RMMS is combined with support software for rapid programming, control, and fault tolerant trajectory planning. In this paper, fault tolerance is defined as the ability to continue a given task even when one of the manipulator modules fails and is immobilized. By carefully planning a fault tolerant joint space trajectory to be followed by the manipulator before any failures occur, one can guarantee that a fault tolerant manipulator can continue its task regardless of which degree-of-freedom is immobilized and regardless of the time at which the failure occurred. For more information about manipulator fault tolerance refer to [11, 13, 14].

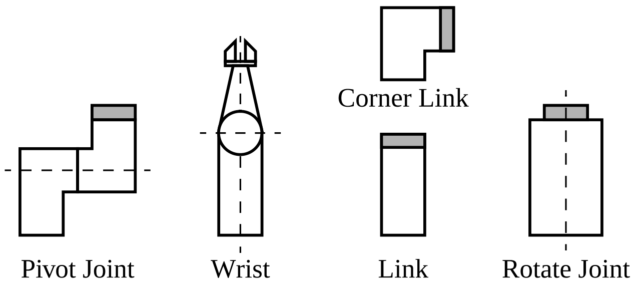

Figure 2: The modules in the inventory.

2 The Task-Based Design Problem

The central component of rapidly deployable fault tolerant manipulator system is the Task Based Design (TBD) software. It answers the question: given a low-level task description and an inventory of manipulator modules, find a manipulator configuration, its base position, and the corresponding fault tolerant joint space trajectory which is optimally suited to perform the given task. Note that TBD involves the complete specification of not only the physical structure of the manipulator (kinematics as well as dynamics), but also its behavior (trajectory planning and control).

We assume that a task is defined as a Cartesian path to be followed by the end-effector of the manipulator without violating any of the following constraints:

- joint position, velocity, and torque limits
- singularity avoidance
- collision avoidance with obstacles and the manipulator itself
- fault tolerance.

The optimality criterion that we consider is energy consumption.

Unfortunately, TBD is a 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 achieved is very computationally expensive.

In the remainder of this paper, we propose a novel agent-based approach to TBD and illustrate the power of this approach with a comprehensive manipulator design problem for a satellite docking operation.

3 An Integrated Solution Approach

In the literature, the TBD problem is usually decoupled into four subproblems: kinematic design, dynamic design, trajectory planning, and control [2, 3, 7, 10, 15]. These subproblems are then solved individually and sequentially. However, this approach assumes a weak interaction between the subproblems, which is often not the case in reality, especially when considering such global design criteria as fault tolerance and energy consumption. In a sequential design approach, strong interactions would result in an unacceptably large number of re-iterations. Therefore, we propose a fully integrated design approach that considers kinematics, dynamics, trajectory planning, and control simultaneously in one large global search problem.

Our approach is based on a genetic algorithm to which the following important modifications have been made: 1) the computational requirements have been reduced by including problem specific design knowledge in the modification and evaluation functions; 2) the computational resources are increased through an agent-based implementation that can be executed on a group of networked workstations.

4 Including Problem Specific Design Knowledge

Michalewicz [8] has shown that the performance of genetic algorithms can be drastically improved by including problem specific knowledge. We implemented two different mechanisms through which problem specific knowledge improves the efficiency of the search algorithm:

- reduction of the search space size
- reduction of the evaluation cost of a candidate solution (i.e. fitness determination)

In the context of TBD, a very important reduction in the size of the search space is obtained by applying the global fault tolerant trajectory planning algorithm [14] to the trajectory planning and control subproblem. Instead of having to *search* for a desired trajectory, one can determine the manipulator's behavior algorithmically, resulting in a very important reduction in the size of the search space. A second important reduction in search space size, is achieved by including problem specific design knowledge in the generation of candidate designs. We require that a candidate design satisfies the following three constraints:

- The first and last module of a manipulator must be a base and end-effector module, respectively.
- A spatial fault tolerant manipulator requires 7 degrees-of-freedom.
- The orientation in which an axially symmetric module is mounted with respect to the previous module can be arbitrarily chosen to be zero degrees.

The combination of these three constraints results again in a very significant reduction of the search space size.

The second important mechanism to improve the efficiency of the search algorithm, is to reduce the cost of evaluating a candidate solution. We call the approach we have implemented "progressive evaluation," because it progressively uses more and more complicated—and time consuming—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. Progressive evaluation is based on the observation that it is very often possible to recognize a bad design quickly using a simple test. 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 fault tolerance is required, test whether a fault tolerant joint space trajectory exists.

- The fault tolerant trajectory planning algorithm itself is also implemented in a progressive manner [14].
- When all tests are satisfied, a simulation of the task execution is started to verify all the other constraints.

5 An Agent-Based Design Framework

To complement the reduction of the computational requirements, it is possible to increase the computational resources through an agent-based implementation of the genetic algorithm.

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: tournament selection [4], parallel subpopulations [17], and a physically distributed population with one processor per individual [9].

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 3, combines three types of agents: creation/evaluation agents, modification/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 autonomous and execute asynchronously.

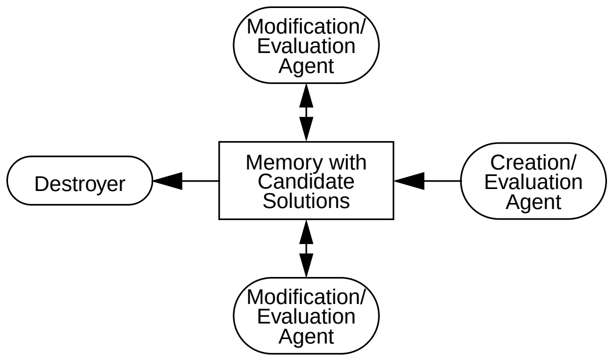
Modification/ Evaluation Agent
Destroyer
Memory with Candidate Solutions
Creation/ Evaluation Agent
Modification/ Evaluation Agent

Figure 3: Layout of the agent-based design framework.

Their functions correspond roughly to the functional entities of the genetic algorithm. At start-up, a *creation/evaluation agent* generates initial candidate solutions, determines their fitness value, and places them in the population (creation of the initial population). Based on the evaluations, *modification/evaluation 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 [16]. Population convergence is achieved in our approach by preferably destroying the worst individuals. As a result, the framework, shown in Figure 3, 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, resulting in a speedup that is almost linear in the number of processors.

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.

6 Task Based Design Example

The TBD problem solved in this example is to design a fault tolerant manipulator for a satellite docking operation aboard the space shuttle. The task is to grab a satellite with a manipulator mounted in the cargo bay, and store it fault tolerantly in a storage unit. The goal of TBD is to determine the optimal manipulator configuration; the base position of the manipulator, i.e., the position and orientation of the space shuttle with respect to the satellite; and the corresponding fault tolerant joint space trajectory. To create the manipulator structure, an inventory of 23 modules is available:[1]

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

As is illustrated in Figure 4, the satellite is cylindrical in shape and weighs 1000kg. The fault tolerant task starts with manipulator's end-effector 0.5m in front of the satellite's grasp handle. The final position of the satellite is in the middle of the cargo bay, 7m behind the manipulator base. 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. The complete manipulator trajectory is defined by four points (and the corresponding times), as shown in Figure 4. Note that the first two points are defined relative to the initial position of the satellite, while the last two points are defined relative to the manipulator base. Because the position and orientation of the manipulator base (i.e., 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 base position to be within a cube

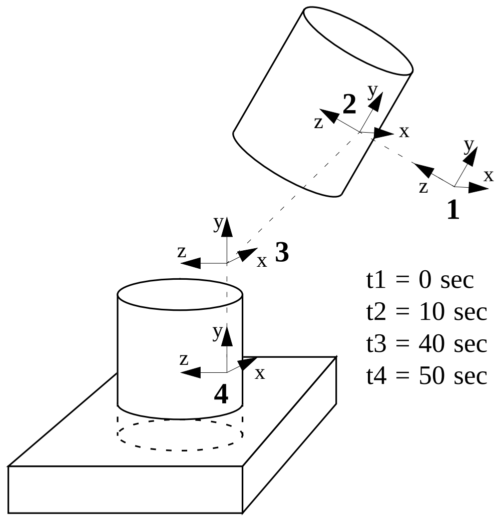

Figure 4: The Cartesian trajectory. The coordinate frames indicated the desired position and orientation of the end-effector of the manipulator.

[1] Complete specifications of all the modules can be found in [11]

of $24 \times 24 \times 24$ meters centered around the initial satellite position. This ensures that the design system does not waste too much time exploring designs for which the satellite is positioned completely 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.

Finally, because the position and orientation of the end-effector are considered, the manipulator needs *seven* DOFs to perform the task fault tolerantly [13].

7 Problem Characterization

Before we evaluate the performance of the agent-based design system, it is important to get an idea of the complexity of the design problem. In this section, we characterize the TBD problem using four characteristics:

- size of the design space
- cost of evaluating a candidate design
- fraction of the design space containing feasible solutions
- quality of the fitness heuristic

The 23 modules in the inventory can be assembled into an extremely large number of configurations:

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

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 very large:

$$N \approx 3 \times 10^{20}. \qquad (2)$$

Besides the discrete design variables, the size of the search space is further increased by 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 (in addition to the determination of the fault tolerant trajectory a complete kinematic and dynamic simulation of the task execution needs to be performed). As is shown in Figure 5, 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 20 workstation. However, the vast

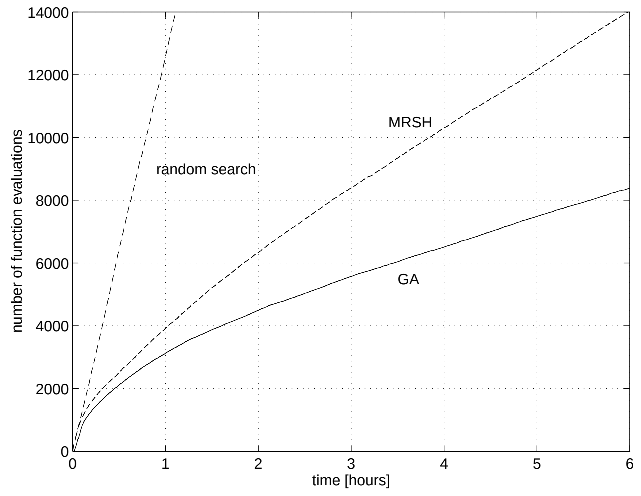

Figure 5: The number of fitness evaluations as a function of time.

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. Moreover, towards the end of the algorithm's execution, 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 possible that a large percentage of all candidate designs 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 in 750,000 function evaluations (64 hours on 24 Sparc workstations).

The final step in characterizing the TBD problem is to examine the quality of the fitness heuristic by determining the probability that a pure 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 the fitness heuristic function leads 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 [6] 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.

8 Performance Analysis

This section compares the performance of the agent-based genetic algorithm with multiple restart statistical hill-climbing (MRSH); MRSH has been suggested as an adequate benchmark for evaluating the performance of genetic algorithms [1, 6]. The MRSH algorithm is implemented as 24 single start statistical hill-climbing algorithms running in parallel on 24 Sparc workstations. On the other hand, the agent-based genetic algorithm a total of 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 Sparc workstations used for MRSH—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 6, 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.

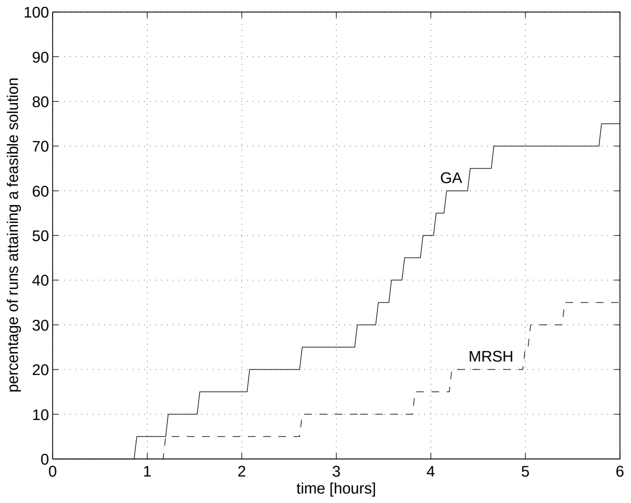

Figure 6: Percentage of the runs attaining a feasible solution.

As Figure 6 indicates, the agent-based genetic algorithm is much more likely than the multiple restart statistical hill-climbing algorithm to find a feasible solution. The genetic algorithm failed to reach a feasible solution in only five out of twenty runs. In those five runs, it was still successful at planning a fault tolerant trajectory, but during the simulation of this trajectory a torque limit violation occurred, 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 all but two of the remaining runs.

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 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 6 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 significantly better than MRSH; the probability that it finds a candidate design with a fitness value larger than the one found by MRSH is between 80% and 90%. Figure 8 confirms that this difference in performance is statistically significant as indicated by the non-parametric Fisher sign test [5].

In conclusion, three hours into the experiment, the agent-based genetic algorithm performs significantly better than the MRSH algorithm: there is a chance of more

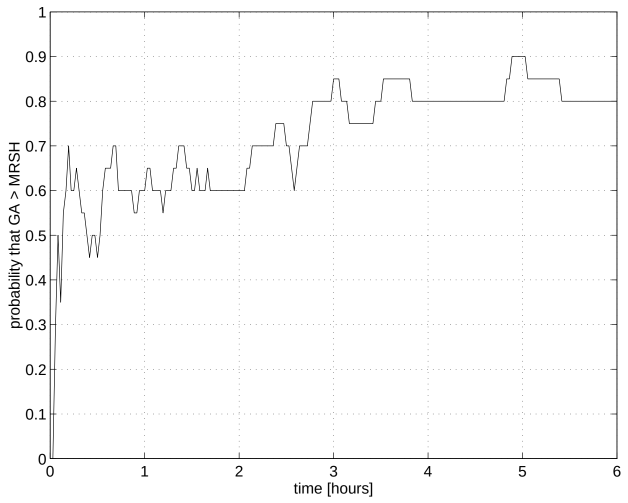

Figure 7: Relative comparison between GA and MRSH.

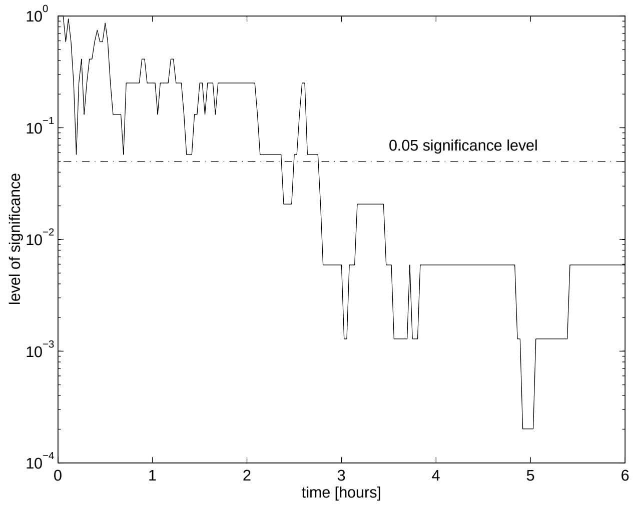

Figure 8: Results of the Fisher sign test.

than 80% that the GA achieves a higher fitness value than MRSH. Moreover, after six hours, the agent-based genetic algorithm finds a feasible solution with a probability of about 75%.

9 Interpretation of the Optimal Design

So far the focus has been on the design *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 9. It consists of only eight modules, as listed in Table 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 criterion) depends on the number of modules (energy consumed by the

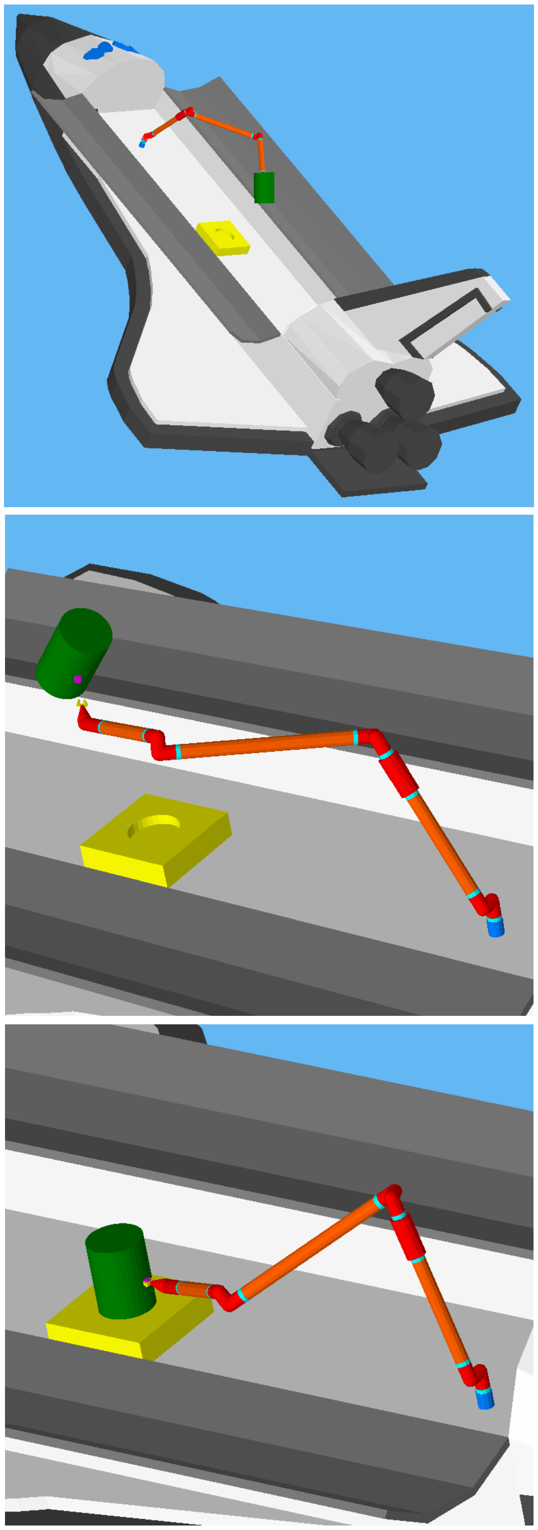
Figure 9: The optimal design found by the agent-based design system. The two figures at the bottom are taken by a camera which is positioned to the right of the cockpit. They show the manipulator in the initial and final posture.

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

Table 1: The module configuration of the optimal design.

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 equally distributed over the length of the manipulator (the last link only looks shorter in Figure 9 because of perspective distortion). The Denavit Hartenberg parameters shown in Table 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 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 [13]. The twist angles between the pivot

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 2: Denavit Hartenberg parameters of the optimal manipulator design.

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.

10 Summary

This paper introduced a novel agent-based design system for task-based design of modular reconfigurable manipulators. This system combines the flexibility and performance of an agent-based implementation of a genetic algorithm with an effective search based on problem specific design knowledge.

The performance of the design system is evaluated for a very challenging problem: the design of a fault tolerant manipulator for a satellite docking operation aboard the space shuttle. The agent-based genetic algorithm easily outperforms a multiple restart statistical hill-climbing algorithm that is used as a benchmark.

Acknowledgment

This research was funded in part by the DOE 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, Carnegie Mellon University.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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).*

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

[9] 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.

[10] 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.

[11] Paredis, C. J. J. 1996. An Agent-Based Approach to the Design of Rapidly Deployable Fault Tolerant Manipulators. (http://www.cs.cmu.edu/~paredis/pubs/thesis.html) Ph.D. thesis. Department of Electrical and Computer Engineering. Carnegie Mellon University.

[12] 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.

[13] Paredis, C. J. J., and Khosla, P. K. 1996. Designing Fault Tolerant Manipulators: How Many Degrees of Freedom? *The International Journal of Robotics Research*. Vol. 15. No. 6.

[14] 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.

[15] 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.

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

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