Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. (IROS'95).
Pittsburgh, Pennsylvania, August 5–9, 1995. Volume 2, pp. 428–434.
IEEE Computer Society Press. Los Alamitos, California.

Global Trajectory Planning for Fault Tolerant Manipulators

Christiaan J. J. Paredis   Pradeep K. Khosla

Department of Electrical and Computer Engineering and
The Robotics Institute,
Carnegie Mellon University,
Pittsburgh, Pennsylvania 15213.

Abstract

Whether a task can be completed after a failure of one of the degrees-of-freedom of a redundant manipulator depends on the joint angle at which the failure takes place. It is possible to achieve fault tolerance by globally planning a trajectory that avoids unfavorable joint positions before a failure occurs. In this paper, we present a trajectory planning algorithm that guarantees fault tolerance while simultaneously satisfying joint limit and obstacle avoidance requirements.

1 Introduction

Reliability is becoming an essential attribute of robot manipulators in a growing range of applications such as space missions, nuclear waste retrieval, and medical robotics. This trend has spawned a research effort in fault tolerant robotics, covering topics ranging from fault detection and identification [11] to design [7, 9], control [10], and redundancy resolution [3] of fault tolerant manipulators.

A manipulator is 1-fault tolerant if it can complete its task even if one of its joints fails and is immobilized [7]. This definition is based on the following scenario. A fault detection and identification algorithm monitors the proper functioning of each degree-of-freedom (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 brake, and automatically adapts the joint trajectory to the new manipulator structure; the task is continued without interruption. According to this scenario, a large variety of possible faults, ranging from sensor faults to transmission and actuation faults, can be treated in exactly the same manner, namely, by eliminating the whole DOF through immobilization.

Whether a task can be completed after a joint failure, depends not only on the structure of the manipulator [8], but also on the specific joint angle at which the failure occurred. 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 paper is to achieve fault tolerance by avoiding unfavorable joint positions *before* failure. This idea was first proposed in [3] where the null-space component of a redundant manipulator was used to locally minimize the kinematic fault tolerance measure (kfm)[1]. 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 globally [4].

In this paper, we present a trajectory planning algorithm that guarantees fault tolerance on a global scale, while 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 in [4, 5].

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[2]: $\theta(t) \in T^n$.

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

1. The kinematic fault tolerance measure, or kfm, is defined as the minimum dexterity after joint failure.
2. We assume that the manipulator has only revolute joints. The joint space is therefore the n-dimensional torus T^n.

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) $\theta_j(t, j, t^*) = \theta_j(t^*), \quad \forall t > t^*$
4) $\theta(t, j, t^*)$ does not violate any secondary task requirements such as joint limits or obstacles.

This definition corresponds to our scenario for fault tolerance as described in the introduction. 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 on 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 algorithms of the form [6]:

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

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 paper, 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 and obstacles 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 in [2] 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 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, $p = f(\theta)$. This implies 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 $\Sigma(p) = \{\theta \in T^n \mid f(\theta) = p\}$, where f is the forward kinematics mapping of the manipulator.

This preimage is a set of r-dimensional manifolds,[3] 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. In all of these cases, the task cannot be completed. We call the corresponding posture $\theta(t^*)$ intolerant to a failure 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 $F_j^{t^*} \subset \Sigma(p(t^*))$.

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:

$$A^{t^*} = \bigcap_{j=1}^{n} F_j^{t^*}. \tag{2}$$

3. An exception is the preimage of a critical value, which is not a manifold but a bouquet of tori [1].

3 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

$$A^{t_{last}} = F_1^{t_{last}} = \dots = F_n^{t_{last}} = \Sigma(p_{last}) \cap S. \quad (3)$$

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 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:

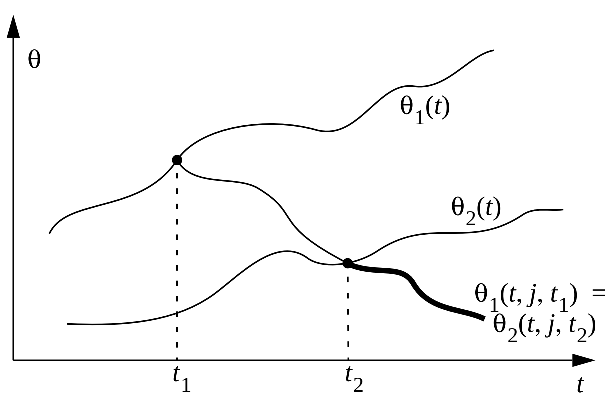

Figure 1: Two possible failures resulting in the same alternate trajectory.

$$\theta(t_1) \in F_j^{t_1} \Leftrightarrow \theta(t^*, j, t_1) \in F_j^{t^*}, \quad \forall t^* > t_1 \text{ and } \theta(t_1) \in S \quad (4)$$

which is also equivalent to the expression:

$$\theta(t_1) \in F_j^{t_1} \Leftrightarrow \theta(t^*, j, t_1) \in F_j^{t^*}, \quad \forall t^* \in (t_1, t_1 + \Delta t] \text{ and } \theta(t_1) \in S \quad (5)$$

Equation (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, $R_i^{t_k}$, of acceptable postures:

$$A^{t_k} = \cup R_i^{t_k} \quad \text{and} \quad R_i^{t_k} \cap R_j^{t_k} = \emptyset, \ \forall i \neq j. \qquad (6)$$

The postures in each region $R_i^{t_k}$ are connected to each other in the sense that there exists a continuous trajectory of acceptable postures, $\theta \in A^{t_k}$, connecting any two postures in $R_i^{t_k}$. On the other hand, by definition, there does not exist any combination of two postures, one from $R_i^{t_k}$ and one from $R_j^{t_k}$, for which such a continuous trajectory can be found. Similarly, we call two regions $R_i^{t_k}$ and $R_j^{t_{k+1}}$ connected if there exists a continuous trajectory of acceptable postures, $\theta(t)$, with $t \in [t_k, t_{k+1}]$, connecting any two postures $\theta_i \in R_i^{t_k}$ and $\theta_j \in R_j^{t_{k+1}}$. As a result, a fault tolerant trajectory exists if and only if there exists a sequence of connected regions, $\{R_i^{t_1}, \ldots, R_j^{t_{last}}\}$. This result is used in the second part of our algorithm, in which we build a connectivity graph representing the connections between the regions $R_i^{t_k}$. 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 $R_j^{t_k}$. 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 Implementational issues

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, where the acceptable postures are grouped into disjoint regions $R_j^{t_k}$. 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 2. 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*.

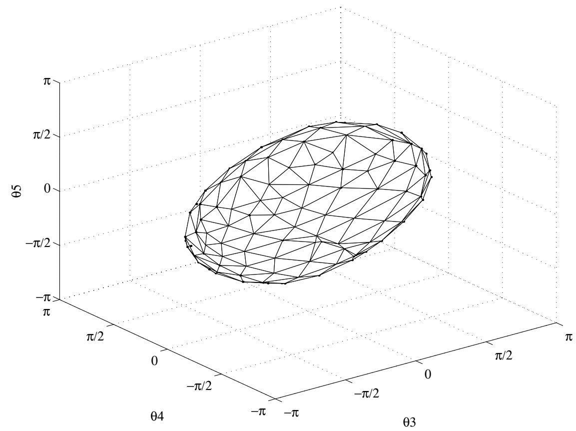

Figure 2: The projection onto the $(\theta_3, \theta_4, \theta_5)$-space of a polygonal approximation of a 2-dimensional preimage for a 5-DOF manipulator.

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. Assum-

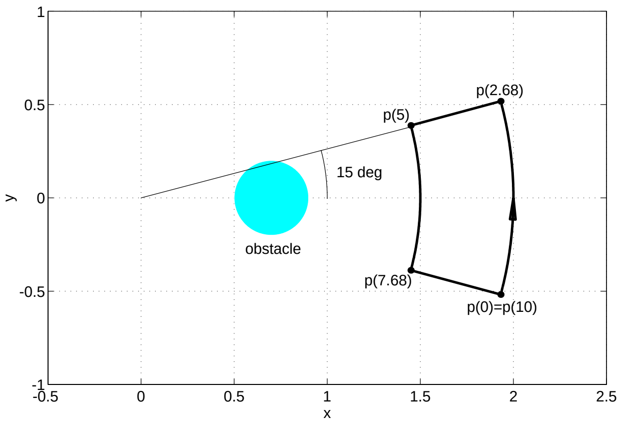

Figure 3: The Cartesian path and obstacle position.

ing that the complexity of the redundancy resolution algorithm is linear in n, the complexity of our trajectory planning algorithm can be expressed as:

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

Because of the exponential dependency on r, the algorithm is only practical for $r = 1$ or $r = 2$. Fortunately, two degrees-of-redundancy are sufficient to achieve fault tolerance in most practical applications [7].

5 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 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. The two parameters are the time and the preimage parameter α. One can also unwrap this surface and represent it in a planar coordinate system with the time in abscissa and the preimage parameter α in ordinate[4]; this representation is used in Figures 5 through 8. A fault tolerant trajectory, $\theta(t) = g(p(t), \alpha(t))$, can then be specified by a function $\alpha(t)$.

In the first part of the algorithm, the sets $F_j^{t_k}$ are determined. They are depicted in Figures 5 through 7 as the white areas. The dark gray areas are postures that do not satisfy the secondary task requirements; i.e., they are the sets $\Sigma(p(t)) - S$. These postures would be unacceptable for a joint trajectory even in case 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 8. This white area is the intersection of the white areas in Figures 5, 6 and 7, in accordance with Equation (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 8. The number of disjoint regions is usually small—maximum 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 9. 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 in dashed line in Figure 8. The corresponding individual joint trajectories are depicted in Figure 10.

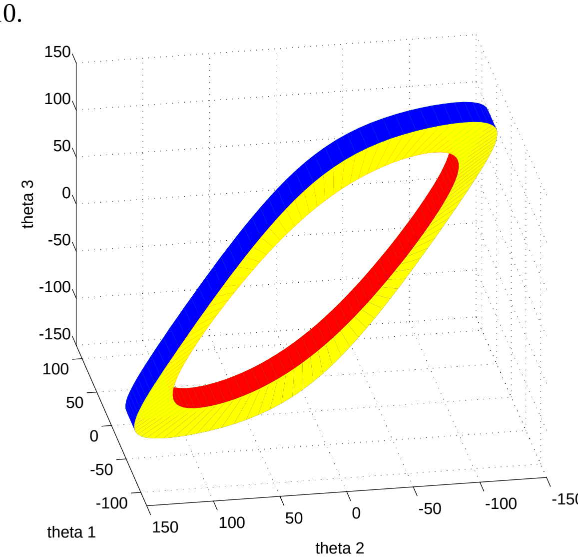

Figure 4: The preimage of the trajectory.

4. 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, $p(0) = p(10)$.

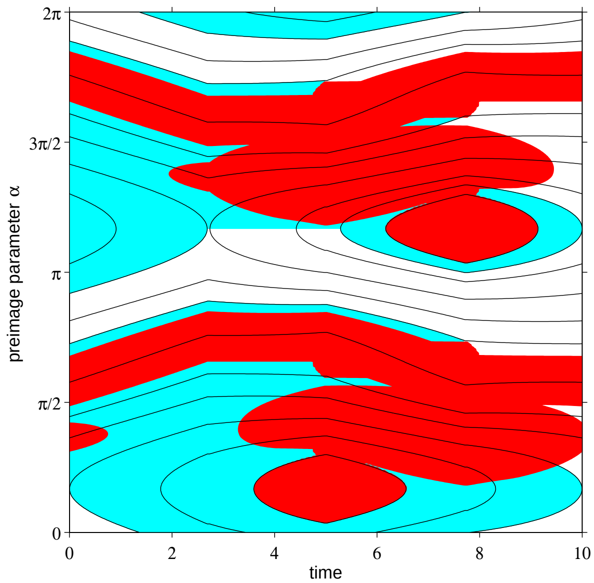

Figure 5: The set of postures tolerant to a fault in joint 1 (in white).

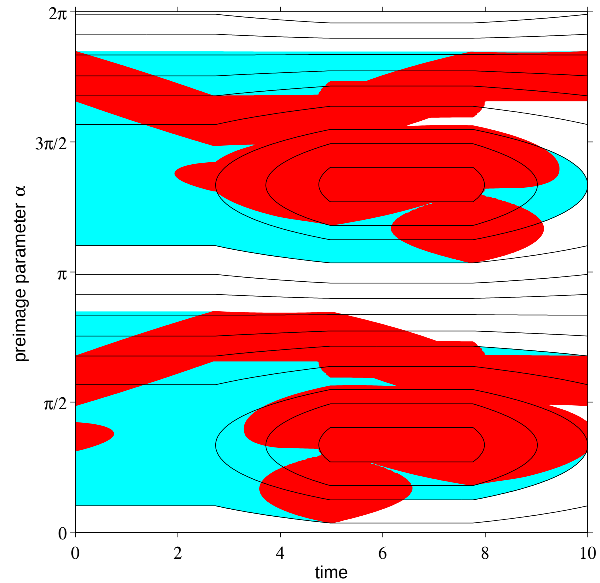

Figure 6: The set of postures tolerant to a fault in joint 2 (in white).

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.

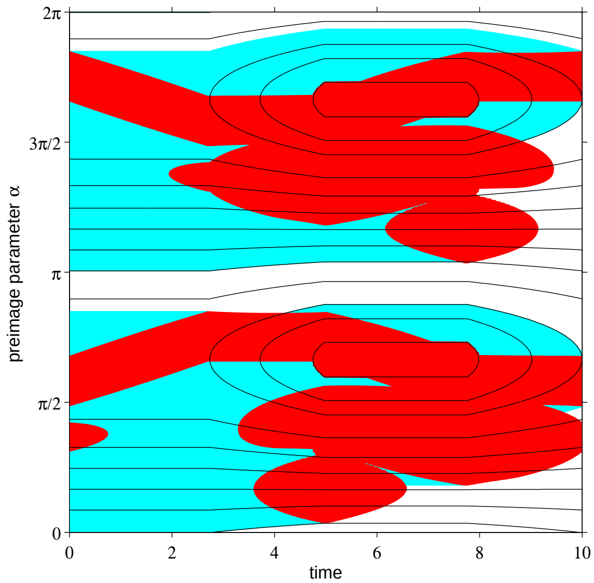

Figure 7: The set of postures tolerant to a fault in joint 3 (in white).

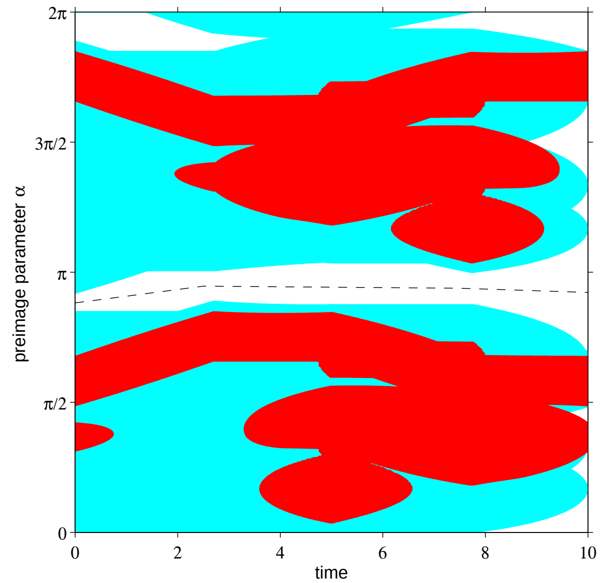

Figure 8: A possible fault tolerant trajectory (dashed line). The white areas are the sets of acceptable postures.

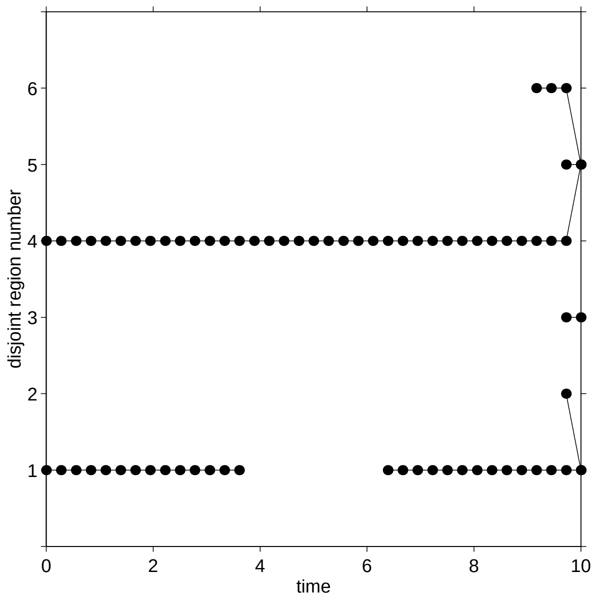

Figure 9: Connectivity graph for the disjoint regions of acceptable postures.

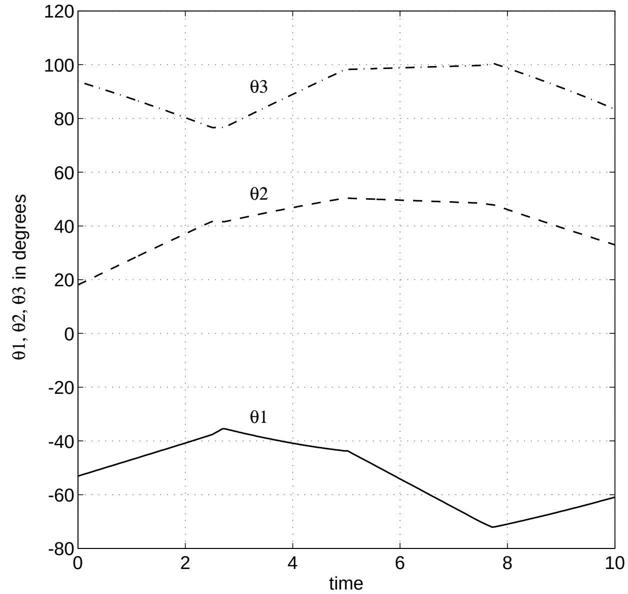

Figure 10: Fault tolerant trajectories for the individual joints.

6 Summary

We have presented a fault tolerant trajectory planning algorithm. This algorithm guarantees fault tolerance on a global scale, while also satisfying secondary kinematic task requirements such as joint limits and obstacles. 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 topological information 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 developed, 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.

Acknowledgment

This research was funded in part by DOE under grant DE-F902-89ER14042, by Sandia under contract AL–3020, by the Department of Electrical and Computer Engineering, and by The Robotics Institute, Carnegie Mellon University.

References

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

[2] J.-O. Kim and P. K. Khosla, “Dexterity Measures for Design and Control of Manipulators,” in *Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems IROS’91*, pp. 758–763, Osaka, Japan, November 1991.

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

[4] C. L. Lewis and A. A. Maciejewski, “An Example of Failure Tolerant Operation of a Kinematically Redundant Manipulator,” in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pp 1380–1387, San Diego, California, May 1994.

[5] C. L. Lück and S. Lee, “Global Path Planning of Redundant Manipulators Based on Self-Motion Topology,” in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pp. 372–377, San Diego, California, May 1994.

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

[7] C. J. J. Paredis and P. K. Khosla, “Mapping Tasks into Fault Tolerant Manipulators,” in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pp. 696–703, San Diego, California, May 1994.

[8] C. J. J. Paredis and P. K. Khosla, “Design of Modular Fault Tolerant Manipulators,” to appear in *The Algorithmic Foundations of Robotics*, eds. K. Goldberg et al., Boston, Massachusetts: A.K. Peeters, 1995.

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

[10] Y. Ting, S. Tosunoglu, and D. Tesar, “Control Algorithms for Fault-Tolerant Robots,” in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pp. 910–915, San Diego, California, May 1994.

[11] M. L. Visinsky, I. D. Walker, and J. R. Cavallaro, “New Dynamic Model-Based Fault Detection Thresholds for Robot Manipulators,” in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, San Diego, California, pp. 1388-1395, May 1994.