A Computational Feasibility Study of Failure-Tolerant Path Planning

Rodrigo S. Jamisola, Jr.[†], Anthony A. Maciejewski[†], & Rodney G. Roberts[‡]
[†]Colorado State University, Ft. Collins, Colorado 80523-1373, USA
[‡]Florida A&M-Florida State University, Tallahassee, Florida 32310-6046, USA
E-mails: r.jamisola@colostate.edu, aam@colostate.edu, & rroberts@wombat.eng.fsu.edu

***Abstract*— This work considers the computational costs associated with the implementation of a failure-tolerant path planning algorithm proposed in [1]. The algorithm makes the following assumptions: a manipulator is redundant relative to its task, only a single joint failure occurs at any given time, the manipulator is capable of detecting a joint failure and immediately locks the failed joint, and the environment is static and known. The algorithm is evaluated on a three degree-of-freedom planar manipulator for a total of eleven thousand different scenarios, randomly varying the robot's start and goal positions and the number and locations of obstacles in the environment. Statistical data are presented related to the computation time required by the different steps of the algorithm as a function of the complexity of the environment.**

I. Introduction

Kinematic failure tolerance gives a robot the ability to gracefully recover from joint failures. Such a control strategy is most useful for robots performing tasks in hazardous or remote environments, such as in space exploration [2], [3], underwater exploration [4], and nuclear waste disposal [5], [6], [7]. It allows a robot to immediately complete the task at hand without unnecessary delays due to robot repair and also avoid the potentially significant danger associated with a robot failure during task execution amongst hazardous materials.

A worst-case failure-tolerance measure was first introduced in [8] using the minimum singular value of the robot Jacobian matrix. The nature of joint failures that have been studied include locked-joint [9], [10], [11] and free-swinging [12] joint failures. A real-time implementation is given in [11]. A number of failure-tolerant control strategies have been proposed, including adaptive control [13], [14], [15], reflex control [16], and least squares approaches [17]. Generally, obstacles are not considered in most kinematic failure tolerance studies. One of the earliest works in kinematic failure tolerance that does consider obstacles in the workspace is [18]. However, the method introduced extensively checks every possibility of failure at every instance in time as the robot plans to move from a start to a goal workspace location. In more recent work [1], a method was introduced that guarantees task completion for any single locked-joint failures despite obstacles in the workspace without extensively checking every possibility of failure at every instance in time. The approach searches for a continuous obstacle-free monotonic surface in the configuration space that guarantees the existence of a solution.

The goal of this work is to quantify the computational feasibility of implementing the kinematic failure tolerance algorithm presented in [1]. We first provide a review of this approach. This is followed by a set of simulation experiments consisting of eleven thousand scenarios, where a scenario is defined as a workspace start location, a workspace goal location, and a set of obstacles. The resulting data is then analyzed to determine the types of scenarios where the implementation of a failure-tolerant path planning strategy may be feasible.

II. Definition of Terms

For an n degree-of-freedom (DOF) kinematically redundant robot operating in an m DOF workspace, the degree-of-redundancy (DOR) is defined as $r = n - m$. The set of configurations in the configuration space (C-space) that result in the same end-effector workspace position/orientation $\mathbf{x}$ is called the pre-image of $\mathbf{x}$, denoted $f^{-1}(\mathbf{x})$. The pre-image can be written as a union of disjoint connected sets

$$f^{-1}(\mathbf{x}) = \bigcup_{i=1}^{n_m} \boldsymbol{\mathcal{M}}_i \tag{1}$$

where $\boldsymbol{\mathcal{M}}_i$ is the i-th r-dimensional self-motion manifold in the inverse kinematic pre-image such that $\boldsymbol{\mathcal{M}}_i \cap \boldsymbol{\mathcal{M}}_j = \emptyset$ when $i \neq j$, and n_m is the number of self-motion manifolds [19]. Fig. 1 shows a set of single dimensional start and goal self-motion manifolds for a robot with $r = 1$. The dark portions of the self-motion manifolds denote configurations of the robot that are in contact with obstacles. A continuous obstacle-free portion of the goal self-motion manifold is denoted as $\boldsymbol{\gamma}_g$ while an obstacle-free start configuration is denoted as $\boldsymbol{\theta}_s$.

For a single locked-joint failure, the resulting C-space is an $(n-1)$-dimensional hyperplane called a *failure hyperplane*. For an obstacle-free start configuration $\boldsymbol{\theta}_s$,

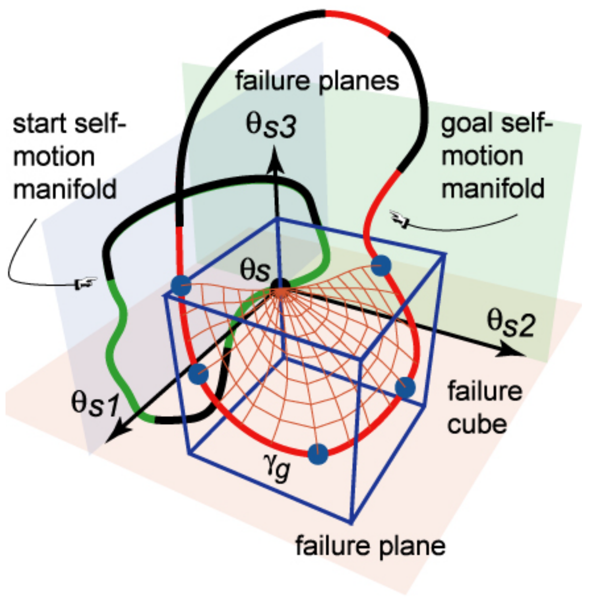

Fig. 1. The configuration space for a single degree of redundancy robot showing a start and goal self-motion manifold. All the failure planes corresponding to an obstacle-free start configuration $\boldsymbol{\theta}_s$ intersect an obstacle-free portion of the goal self-motion manifold $\boldsymbol{\gamma}_g$. The failure cube contains an obstacle-free start configuration $\boldsymbol{\theta}_s$ and an obstacle-free portion of the goal self-motion manifold $\boldsymbol{\gamma}_g$. The failure surface, shown as a web-like network of paths, corresponding to an obstacle-free start configuration $\boldsymbol{\theta}_s$ is identified by generating monotonic paths within the failure cube and connecting the obstacle-free start configuration $\boldsymbol{\theta}_s$ to points on the obstacle-free portion of the goal self-motion manifold $\boldsymbol{\gamma}_g$. Each node along the obstacle-free portion of the goal self-motion manifold $\boldsymbol{\gamma}_g$ defines an intersection with either a failure plane or a face of the failure cube.

a failure hyperplane $\mathbf{H}_i$ associated with $\boldsymbol{\theta}_s$ is given by

$$\mathbf{H}_i(\boldsymbol{\theta}_s) = \{\boldsymbol{\theta} \mid \theta_i = \theta_{si}\} \tag{2}$$

where θ_i denotes the i-th component of $\boldsymbol{\theta}$ in a failure-induced C-space, θ_{si} is the fixed value of the locked i-th component of start configuration $\boldsymbol{\theta}_s$, and $i = 1, \ldots, n$. An obstacle-free start configuration $\boldsymbol{\theta}_s$ is shown with its corresponding failure planes in Fig. 1.

A *failure hypercube* is a hypervolume in C-space that contains an obstacle-free start configuration $\boldsymbol{\theta}_s$ and an obstacle-free portion of the goal self-motion manifold $\boldsymbol{\gamma}_g$ such that all the failure hyperplanes corresponding to the obstacle-free start configuration $\boldsymbol{\theta}_s$ intersect an obstacle-free portion of the goal self-motion manifold $\boldsymbol{\gamma}_g$. A failure hypercube associated with $\boldsymbol{\theta}_s$ and $\boldsymbol{\gamma}_g$ has the form

$$\mathbf{V} = \{\boldsymbol{\theta} \mid \theta_{\mathbf{H}_i^l} \le \theta_i \le \theta_{\mathbf{H}_i^u} \text{ for } i = 1, \ldots, n\} \tag{3}$$

where $\mathbf{H}_i^l$ is the lower bounding hyperplane,

$$\mathbf{H}_i^l = \{\boldsymbol{\theta} \mid \theta_i = \theta_{\mathbf{H}_i^l}\}, \tag{4}$$

and $\mathbf{H}_i^u$ is the upper bounding hyperplane,

$$\mathbf{H}_i^u = \{\boldsymbol{\theta} \mid \theta_i = \theta_{\mathbf{H}_i^u}\}, \tag{5}$$

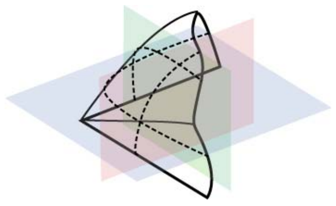

Fig. 2. A monotonic surface is defined as a surface where there are no closed contours along the intersection with any plane parallel to its failure planes. A monotonic surface has no local minima or maxima.

for $i = 1, \ldots, n$. The values of $\theta_{\mathbf{H}_i^l}$ and $\theta_{\mathbf{H}_i^u}$ define the location of their corresponding hyperplanes. Typically, some of the faces of a failure hypercube will lie in the failure hyperplanes associated with $\boldsymbol{\theta}_s$. The rest of the faces of the failure hypercube are defined by the extremal points of $\boldsymbol{\gamma}_g$. Note that a failure hypercube is not typically obstacle free.

A *failure surface* $\boldsymbol{\mathcal{S}}$ is a continuous obstacle-free monotonic surface within the failure hypercube that contains an obstacle-free start configuration $\boldsymbol{\theta}_s$ and is bounded by three curves: an obstacle-free portion of the goal self-motion manifold $\boldsymbol{\gamma}_g$, an obstacle-free monotonic curve lying in the first failure hyperplane that intersects $\boldsymbol{\gamma}_g$, and an obstacle-free monotonic curve lying in the last failure hyperplane that intersects $\boldsymbol{\gamma}_g$. A monotonic surface has the property that its intersection with every plane parallel to a failure hyperplane is a non-closed curve (see Fig. 2). Thus a monotonic surface does not have any local internal minima or maxima. Fig. 1 shows a web-like network of paths that represent a failure surface $\boldsymbol{\mathcal{S}}$ within a failure cube. The outermost paths from $\boldsymbol{\theta}_s$ to $\boldsymbol{\gamma}_g$ lie on their respective failure planes.

III. Guaranteeing a Failure-Tolerant Path

A. A Necessary Condition and a Sufficient Condition

Two conditions are derived that determine the existence of a solution. A necessary condition is used to identify feasible obstacle-free start configurations $\boldsymbol{\theta}_s$ and a sufficient condition is used to determine the existence of a solution from all the feasible obstacle-free start configurations $\boldsymbol{\theta}_s$.

Necessary Condition. A given obstacle-free configuration $\boldsymbol{\theta}_s$ is called a feasible configuration if all the corresponding failure hyperplanes associated with $\boldsymbol{\theta}_s$ intersect a portion of the obstacle-free goal self-motion manifold $\boldsymbol{\gamma}_g$, that is,

$$\mathbf{H}_i(\boldsymbol{\theta}_s) \cap \boldsymbol{\gamma}_g \neq \emptyset, \text{ for all } i = 1, \ldots, n \tag{6}$$

where $\mathbf{H}_i$ is the failure hyperplane at $\boldsymbol{\theta}_s$ corresponding to joint i. This ensures a possibility of reaching the obstacle-free portion of the goal self-motion manifold $\boldsymbol{\gamma}_g$ despite a joint failure at the start configuration $\boldsymbol{\theta}_s$. Note that this is equivalent to the goal position being in the fault-tolerant workspace [10].

Sufficient Condition. Consider a given failure hypercube $\mathbf{V}$ containing an obstacle-free start configuration $\boldsymbol{\theta}_s$ and an obstacle-free portion of the goal self-motion manifold $\boldsymbol{\gamma}_g$. If a failure surface $\boldsymbol{\mathcal{S}}$ which is a continuous obstacle-free monotonic surface in the failure hypercube $\mathbf{V}$ exists such that the two conditions

$$\boldsymbol{\theta}_s \in \boldsymbol{\mathcal{S}} \text{ and } \boldsymbol{\gamma}_g \subset \boldsymbol{\mathcal{S}} \tag{7}$$

hold, then an obstacle-free path to the goal is guaranteed for any single locked-joint failure at any given time despite the presence of obstacles in the workspace.

B. Algorithm

The procedure for our proposed method is enumerated in the following.

1. Determine the start self-motion manifold $\boldsymbol{\mathcal{M}}_s$ and goal self-motion manifold $\boldsymbol{\mathcal{M}}_g$, from the given start, $\mathbf{x}_s$, and goal, $\mathbf{x}_g$, workspace position/orientation, respectively.
2. Identify an obstacle-free start configuration $\boldsymbol{\theta}_s$ and an obstacle-free portion of the goal self-motion manifold $\boldsymbol{\gamma}_g$.
3. Check for intersections of the failure hyperplanes $\mathbf{H}_i(\boldsymbol{\theta}_s)$ with $\boldsymbol{\gamma}_g$ for $i = 1, \ldots, n$. (Note that this step uses the necessary condition in Section III-A).
4. Check for the existence of a failure surface $\boldsymbol{\mathcal{S}}$. This is done by generating monotonic paths from $\boldsymbol{\theta}_s$ to points in $\boldsymbol{\gamma}_g$ and checking for intersections with obstacles. These paths are limited to straight lines or the set of quadratic curves that are monotonic. Straight line connections between paths are used to check for the continuity of obstacle-free space between paths. (This step uses the sufficient condition in Section III-A).

Given that a failure surface $\boldsymbol{\mathcal{S}}$ exists, to move from the start position/orientation, $\mathbf{x}_s$, towards the goal position/orientation, $\mathbf{x}_g$, the manipulator configuration traverses from the obstacle-free start configuration $\boldsymbol{\theta}_s$ along the continuous web of paths that represents the failure surface $\boldsymbol{\mathcal{S}}$ toward the obstacle-free portion of the goal self-motion manifold $\boldsymbol{\gamma}_g$. Because $\boldsymbol{\mathcal{S}}$ is known to be collision free, as long as the manipulator configuration remains on the surface, the manipulator would be free from collision and at the same time it can reach the goal for any single locked-joint failure at any time. If no failure surface $\boldsymbol{\mathcal{S}}$ exists, then it is not guaranteed that the robot can successfully complete its task for any single locked-joint failure with the given obstacles in the environment.

The computational complexity of the proposed algorithm is highly dependent on the method used for computing the start and goal self-motion manifolds, and the method used for collision detection. For a single DOR, the computational complexity is $O(mn^2) + O(mnp)$ where p is the number of obstacles in the workspace. The first term is the contribution for the computation of the self-motion manifolds, while the second term is the contribution due to collision detection.

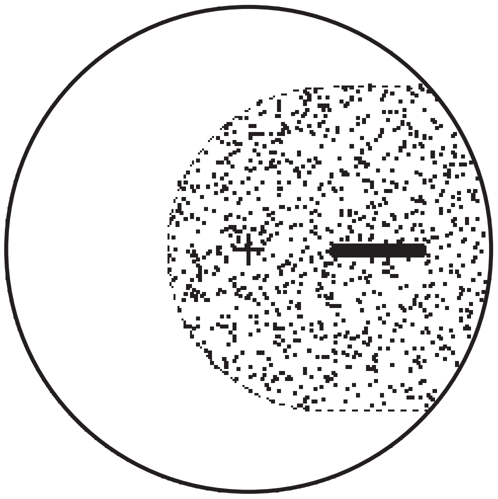
Fig. 3. The workspace of a 3-DOF planar manipulator used for the simulation experiments (all three link lengths are 100 units long). The one thousand start locations, $\mathbf{x}_s$, are randomly generated within the range $[100, 200]$ units along the x-axis (shown as a thick bold line). The one thousand goal workspace locations, $\mathbf{x}_g$, (shown as dots) are randomly generated to be within the range $[0, 200]$ units away from their corresponding start location (but inside the reachable workspace). The center of the workspace is marked with a bold cross, the workspace boundary with a solid line, and the boundary of the goal locations with a dashed line.

IV. Feasibility Study: 3-Link Planar Robot

A 3-DOF planar robot with equal link lengths of 100 units was used for this feasibility study. The workspace contained a number of circular obstacles, each of a diameter of 40 units, where the number of obstacles was varied from zero to twenty in two obstacle increments. For each workspace with a given number of obstacles, simulation experiments were performed for one thousand randomly generated scenarios, where a scenario consists of a given start workspace location, a given goal workspace location, and the specified locations of the obstacles. In each scenario, the locations for the corresponding number of obstacles are randomly selected from a uniform distribution throughout the entire robot workspace. The start workspace location $\mathbf{x}_s$ is randomly selected from a uniform distribution of $[100, 200]$ along the x-axis, that is, at a distance that corresponds to between one and two link lengths away from the base. The goal workspace location $\mathbf{x}_g$ is randomly selected to be within a range of $[0, 200]$ units from the corresponding start workspace location (while restricting the goal to be within the reachable workspace of the manipulator). Fig. 3 presents an example of the start and goal locations generated for one thousand scenarios. A total of eleven thousand scenarios

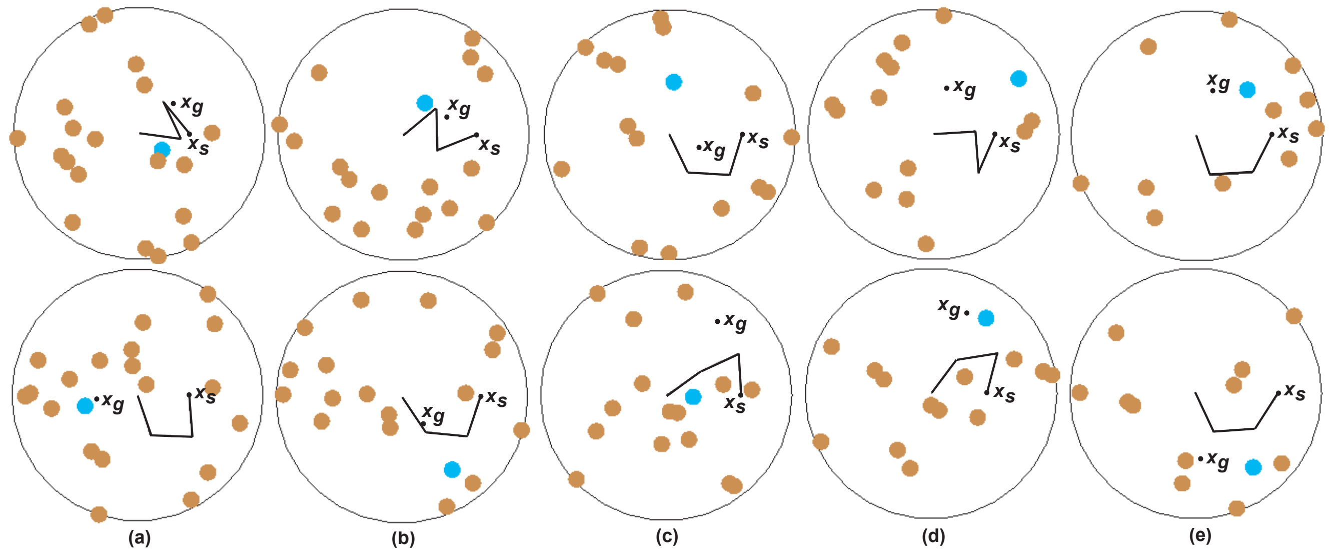

Fig. 4. Several examples selected from the eleven thousand scenarios where the fault-tolerant path planning algorithm was applied to 3-DOF planar manipulator. The subfigures denoted (a) through (e) correspond to workspaces containing from twenty to twelve obstacles, respectively. In each case, the top scenario represents one in which the algorithm successfully identified a failure surface $\mathcal{S}$, whereas the bottom scenario represents a case where such a surface could not be found. In all cases the start configuration shown for the manipulator is one that satisfies the necessary condition for a surface to exist.

were generated, i.e., one thousand for each workspace containing a given number of obstacles.

Ten examples from this set of eleven thousand scenarios are presented in Fig. 4, where the number of obstacles is varied from twenty to twelve in two obstacle increments. The top row of this figure illustrates cases where the fault tolerant path planning algorithm was able to identify a failure surface, with the bottom row showing cases where it could not. The necessary condition for a surface to exist was satisfied in all cases, with the manipulator configuration illustrating one such feasible starting configuration. In the following subsections, data gathered from these eleven thousand simulation experiments will be presented according to the order in which these steps are performed in the algorithm described in Sec. III-B. In all cases the algorithm was executed on a computer with dual Intel Xeon processors running at 2.4 GHz.

A. Computation of Self-Motion Manifolds

The first step in the failure-tolerant path planning algorithm is to compute the self-motion manifold(s) for both the start and goal workspace locations. These are computed by identifying a single configuration on each disjoint manifold and then stepping along the manifold (in two degree increments) by computing the null vector of the manipulator Jacobian matrix (which corresponds to the tangent of the self-motion manifold). The average lengths for the sum of all manifolds corresponding to a workspace location are given in Table I. The length for the start manifolds are larger than those for the goal manifold because the start workspace locations were

TABLE I

COMPUTATION OF SELF-MOTION MANIFOLDS

	Ave.	Std. Dev.
Length of $\mathcal{M}_s$ (deg)	1043.95	4.97
Length of $\mathcal{M}_g$ (deg)	883.71	8.52
Time to compute $\mathcal{M}_s$, $\mathcal{M}_g$ (ms)	41.90	5.88

intentionally restricted to a region of the workspace where these manifolds are larger, and thus, the locations are more failure tolerant [10]. (A distance of one link length from the base is optimally failure tolerant, i.e., all joints span their entire range of motion, while the location on the workspace boundary is most failure intolerant, i.e., its self-motion manifold consists of a single point.) The average computation time over all manifolds was 41.9 ms.

B. Identifying Obstacle-Free Portions of $\mathcal{M}_s$ and $\mathcal{M}_g$

After the start self-motion manifold $\mathcal{M}_s$ and goal self-motion manifold $\mathcal{M}_g$ are computed, the next step is to determine the obstacle-free portions of the manifolds. This identifies candidate feasible start configurations $\boldsymbol{\theta}_s$ and continuous obstacle-free portion of the goal self-motion manifold γ_g that can possibly satisfy the necessary condition.

Fig. 5 shows the percentage of the start self-motion manifold $\mathcal{M}_s$ and goal self-motion manifold $\mathcal{M}_g$ that are obstacle free as a function of the number of obstacles in the workspace. As expected, the percentage of the obstacle-free self-motion manifold decreases as the

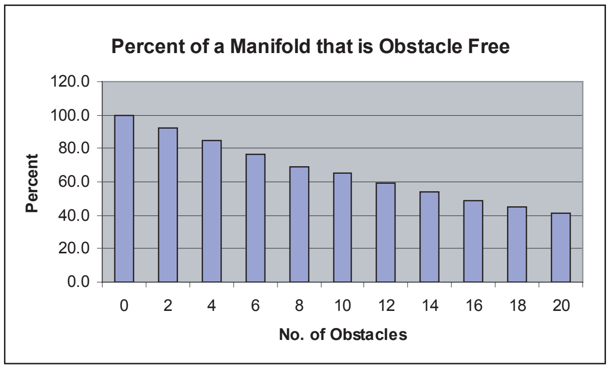

Fig. 5. Percent of the self-motion manifolds that are obstacle free.

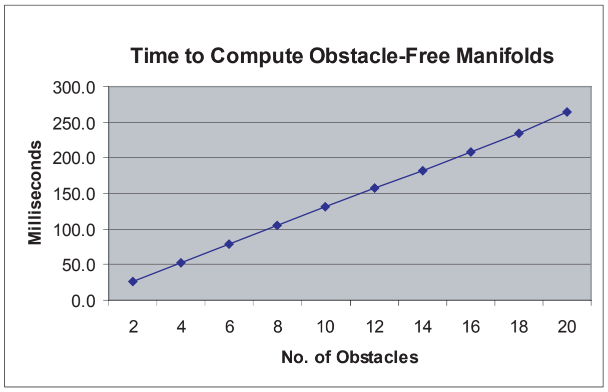

Fig. 6. Time needed to determine the obstacle-free portion of the start self-motion manifold $\mathcal{M}_s$ and the goal self-motion manifold $\mathcal{M}_g$ as a function of the number of workspace obstacles.

number of workspace obstacles increases. Fig. 6 shows the time, in milliseconds, required to determine the obstacle-free portion of the self-motion manifolds. As expected, the time required to determine which portions of a manifold are obstacle free is a linear function of the number of obstacles.

In many cases, one could avoid most of the computation time associated with checking the entire manifold for collisions with obstacles by performing the necessary condition check first, and then verifying that the start configuration and corresponding portion of the goal self-motion manifold are collision free. (This check is the same as determining if the goal location $\mathbf{x}_g$ is in the failure-tolerant workspace of the start location $\mathbf{x}_s$ as first considered in [10].)

C. Checking the Necessary Condition

The next step in the failure-tolerant path planning algorithm is to apply the necessary condition to those portions of the start and goal manifolds that are collision free. This is done by selecting a start configuration $\boldsymbol{\theta}_s$ from the obstacle-free portion of the start manifold and then checking for intersections of all failure planes with each γ_g. If a γ_g intersects all failure planes then the necessary condition is satisfied and this step terminates. If no γ_g satisfies the intersection tests then a new $\boldsymbol{\theta}_s$ is selected and the tests repeated until either the necessary condition is satisfied or all $\boldsymbol{\theta}_s$ have been tested. Thus if this test fails it is guaranteed that no failure-tolerant obstacle-free path exists between the start and goal locations.

Table II presents the computational data associated with checking the necessary condition. Note that even without any obstacles, the necessary condition can only be satisfied in 69% of the scenarios tested. This is essentially an indication of the difficulty of guaranteeing that a goal location will be in the failure-tolerant workspace of the start configuration. This is clearly very dependent on both the starting location and the distance between the start location and the goal location [10]. As the number of obstacles in the workspace increases, the percentage of cases that satisfy the necessary condition decreases, reaching a minimum of 25% for the case with twenty obstacles. For those cases where a $\boldsymbol{\theta}_s$ satisfied the necessary condition, we further processed the start manifold to see what percentage of the start manifold would be able to satisfy the necessary condition. (This is not required by the algorithm and the time required to perform this computation was not included in the overall execution time data presented.) Table II shows that this is also a monotonically decreasing function of the number of obstacles in the workspace. Thus, it becomes increasingly more time consuming to identify a $\boldsymbol{\theta}_s$ that satisfies the necessary condition as the number of obstacles in the workspace increases. This is illustrated by the fact that the computation time is a monotonically increasing function of the number of obstacles. This is true despite the fact that there is less and less of the start manifold that is obstacle free (see Fig. 5) because an increasingly smaller percentage of the obstacle-free manifold is able to satisfy the necessary condition. In contrast, the time to compute that the necessary condition is not satisfied is relatively independent of the number of obstacles. This at first appears anomalous because one would expect this to be a monotonically decreasing function due to a smaller percentage of the start manifold needing to be checked (because less of it is obstacle free). However, this is offset by the fact that larger and larger manifolds are now failing the necessary condition, thus keeping the computation time relatively constant.

Fig. 7 shows the average length of the γ_g from a $(\boldsymbol{\theta}_s, \gamma_g)$ pair that satisfies the necessary condition. This generally decreases as the number of obstacles increases due to the fact that it is more difficult to have large γ_g because they are required to be obstacle free. The size of a γ_g that satisfies the necessary condition is correlated to the distance between the start and goal workspace locations so

TABLE II

COMPUTATION TO CHECK THE NECESSARY CONDITION

No. of Obs.	Time N.C. Sat. (ms)	Time N.C. Not Sat. (ms)	% Cases N.C. Sat.	% $\mathcal{M}_s$ N.C. Sat.
0	0.73	4.39	69.3	25.4
2	0.84	4.59	64.8	20.5
4	1.13	5.03	57.9	16.5
6	1.13	4.86	52.2	15.4
8	1.05	4.57	48.9	12.4
10	1.33	5.13	45.8	10.0
12	1.48	5.35	38.2	7.6
14	1.50	5.43	36.7	6.3
16	1.84	6.15	30.8	5.9
18	1.90	5.56	27.6	4.1
20	2.92	5.53	24.8	3.7

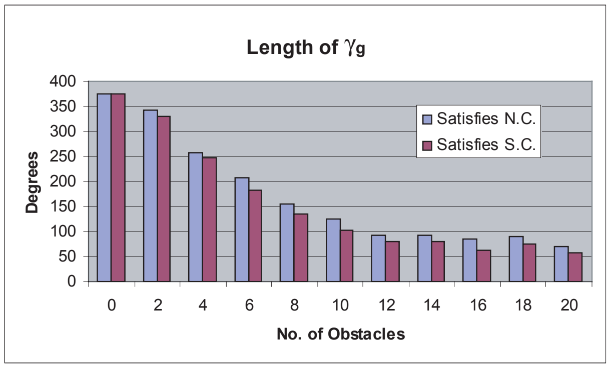

Fig. 7. Average length of a γ_g that satisfies the necessary condition and that also satisfies the sufficient condition as a function of the number of obstacles in the workspace.

that, as expected, start and goal locations must be closer together as more and more obstacles are added to the workspace.

D. Computing a Failure Surface

The final step in the algorithm is to check the sufficient condition by attempting to compute a failure surface $\mathcal{S}$ that guarantees the existence of a solution to the failure-tolerant path planning problem. Once a feasible start configuration $\boldsymbol{\theta}_s$ is found from the previous step, the search for a failure surface $\mathcal{S}$ begins. A failure surface $\mathcal{S}$ is identified by generating monotonic paths that connect a feasible start configuration $\boldsymbol{\theta}_s$ to points on its corresponding γ_g that satisfies the necessary condition. (The γ_g curve is discretized at a resolution of two degrees.) For each point on γ_g the algorithm first attempts to use a straight-line path. If this path is not obstacle free, then it attempts to find a monotonic quadratic path that is obstacle free. If no such path can be found then the algorithm discards this $(\boldsymbol{\theta}_s, \gamma_g)$ pair and uses the next $(\boldsymbol{\theta}_s, \gamma_g)$ pair that satisfies the necessary condition. If all such pairs are exhausted

TABLE III

COMPUTATION TO CHECK THE SUFFICIENT CONDITION (ONCE THE NECESSARY CONDITION IS SATISFIED)

	Ave.	Std. Dev
% of times $\mathcal{S}$ found	84.23	2.58
% linear paths on $\mathcal{S}$	99.63	0.19
Time to compute $\mathcal{S}$(s)	1.16	0.23
Time to not find $\mathcal{S}$(s)	1.80	0.37

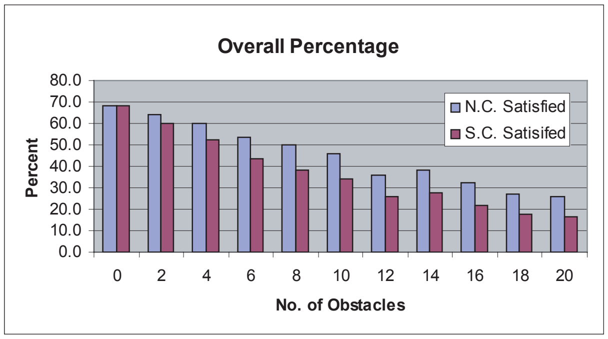

Fig. 8. Percent of total cases where the necessary condition is satisfied and percent of total cases where the sufficient condition is satisfied, i.e., where a failure surface $\mathcal{S}$ is found.

without completing a failure surface then the algorithm terminates with a message that is was unsuccessful.

Table III presents data on the computation of failure surfaces. It is interesting to note that once the necessary condition is satisfied, it is highly likely that a failure surface will be found, i.e., this occurs 84% of the time. In addition, this percentage is relatively independent of the number of obstacles that are in the workspace as illustrated in Fig. 8 (except, of course, for the case of no obstacles). This is fortuitous, because the construction of failure surfaces is by far the most time consuming portion of the algorithm. (The average time to compute a failure surface is 1.16 s, and it takes 1.80 s, on average, to exhaust all possible candidates in cases where a surface cannot be found.) Thus the time to evaluate surfaces is seldom wasted, with the majority of the cases where no solution exists being identified in a matter of milliseconds by the necessary condition test. It is also interesting to note that nearly all (99.6%) of the paths in failure surfaces are linear. However, it is important to point out that even if only one path in a failure surface is quadratic, then that implies that that surface would not have satisfied the sufficient condition if only straight-line paths were allowed.

V. SUMMARY AND CONCLUSION

This paper has presented and analyzed the behavior and computational cost of a failure-tolerant path planning strategy originally proposed in [1]. This algorithm consists of four steps: (1) construction of self-motion manifolds, (2) collision detection for those manifolds, (3) testing of a necessary condition for a solution to the problem, and (4) determining if a sufficient condition was satisfied. The probability of identifying a failure-tolerant path is strongly dependent on several factors, i.e., the start workspace position, the goal workspace position, and the number and location of the obstacles. It was tested on 11,000 scenarios for a planar 3-DOF manipulator with all of these factors randomly varied. Even with no obstacles, failure-tolerant paths only existed for 69% of the cases tested. For environments with 20 obstacles (the largest number of obstacles tested) this number dropped to 15%. However, one advantage of the algorithm is that for most of the cases where a path does not exist it is identified by the relatively computationally inexpensive test of the necessary condition. In all cases, the algorithm was computationally feasible, i.e., to determine that no possible path exists took an average of 0.2 s, to compute a failure-tolerant surface took an average of 1.2 s, and if the algorithm could not determine whether a surface exists took an average of 2.0 s.

VI. References

[1] R. S. Jamisola, Jr., A. A. Maciejewski, and R. G. Roberts, “A path planning strategy for kinematically redundant manipulators anticipating joint failures in the presence of obstacles,” in *Proc. Int. Conf. Intell. Robots Syst.*, Las Vegas, NV, Oct. 27-31 2003, pp. 142–148.

[2] E. C. Wu, J. C. Hwang, and J. T. Chladek, “Fault-tolerant joint development for the space shuttle remote manipulator system: Analysis and experiment,” *IEEE Trans. Robot. Automat.*, vol. 9, no. 5, pp. 675–684, Oct. 1993.

[3] G. Visentin and F. Didot, “Testing space robotics on the Japanese ETS-VII satellite,” ESA Bulletin-European Space Agency, pp. 61–65, Sept. 1999.

[4] P. S. Babcock and J. J. Zinchuk, “Fault-tolerant design optimization: Application to an autonomous underwater vehicle navigation system,” in *Proc. 1990 Symp. Autonom. Underwater Vehicle Technol.*, Washington, D.C., June 5-6 1990, pp. 34–43.

[5] R. Colbaugh and M. Jamshidi, “Robot manipulator control for hazardous waste-handling applications,” *J. Robot. Syst.*, vol. 9, no. 2, pp. 215–250, 1992.

[6] W. H. McCulloch, “Safety analysis requirements for robotic systems in DOE nuclear facilities,” in *Proc. 2nd Specialty Conf. Robot. Challenging Environ.*, Albuquerque, NM, June 1-6 1996, pp. 235–240.

[7] M. L. Leuschen, I. D. Walker, and J. R. Cavallaro, “Investigation of reliability of hydraulic robots for hazardous environment using analytic redundancy,” in *Proc. Annual Rel. Maintain. Symp.*, Washington, D.C., Jan. 18-21 1999, pp. 122–128.

[8] A. A. Maciejewski, “Fault tolerant properties of kinematically redundant manipulators,” in *Proc. IEEE Int. Conf. Robot. Automat.*, Cincinnati, OH, May 13-18 1990, pp. 638–642.

[9] R. G. Roberts and A. A. Maciejewski, “A local measure of fault tolerance for kinematically redundant manipulators,” *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 543–552, Aug. 1996.

[10] C. L. Lewis and A. A. Maciejewski, “Fault tolerant operation of kinematically redundant manipulators for locked joint failures,” *IEEE Trans. Robot. Automat.*, vol. 13, no. 4, pp. 622–629, Aug. 1997.

[11] K. N. Groom, A. A. Maciejewski, and V. Balakrishnan, “Real-time failure-tolerant control of kinematically redundant manipulators,” *IEEE Trans. Robot. Automat.*, vol. 15, no. 6, pp. 1109–1116, Dec. 1999.

[12] J. D. English and A. A. Maciejewski, “Fault tolerance for kinematically redundant manipulators: Anticipating free-swinging joint failures,” *IEEE Trans. Robot. Automat.*, vol. 14, no. 4, pp. 566–575, Aug. 1998.

[13] Y. Ting, S. Tosunoglu, and D. Tesar, “A control structure for fault-tolerant operation of robotic manipulators,” in *Proc. IEEE Int. Conf. Robot. Automat.*, Atlanta, GA, May 2-6 1993, pp. 684–690.

[14] K. S. Tso, M. Hecht, and N. I. Marzwell, “Fault-tolerant robotic system for critical applications,” in *Proc. IEEE Int. Conf. Robot. Automat.*, Atlanta, GA, May 2-6 1993, pp. 691–696.

[15] Y. Ting, S. Tosunoglu, and R. Freeman, “Actuator saturation avoidance for fault-tolerant robots,” in *Proc. 32nd Conf. Decision Contr.*, San Antonio, TX, Dec. 1993, pp. 2125–2130.

[16] T. S. Wikman, M. S. Branicky, and W. S. Newman, “Reflexive collision avoidance: A generalized approach,” in *Proc. IEEE Int. Conf. Robot. Automat.*, Atlanta, GA, May 2-6 1993, pp. 31–36.

[17] J. Park, W.-K. Chung, and Y. Youm, “Failure recovery by exploiting kinematic redundancy,” in *5th Int. Workshop Robot Human Commun.*, Tsukuba, Japan, Nov. 11-14 1996, pp. 298–305.

[18] C. J. J. Paredis and P. K. Khosla, “Fault tolerant task execution through global trajectory planning,” *Rel. Eng. Syst. Safety*, vol. 53, pp. 225–235, 1996.

[19] J. W. Burdick, “On the inverse kinematics of redundant manipulators: Characterization of the self-motion manifolds,” in *Proc. IEEE Int. Conf. Robot. Automat.*, Scottsdale, AZ, May 14-19 1989, pp. 264–270.