Computing Fault Tolerant Motions for a Robot Manipulator*

Scott K. Ralph Dinesh K. Pai
Department of Computer Science
University of British Columbia
Vancouver, Canada
{ralph|pai}@cs.ubc.ca

Abstract

We introduce a method of planning fault tolerant trajectories based on the *least constraint* (LC) framework presented in [12]. Fault tolerance is achieved in two ways: exploiting properties of LC itself, and using a performance measure which assess the fault tolerant potential of a given configuration. LC encourages designs which are based solely on salient constraints of the task, allowing the inherent redundancy of the robot to be used to maintain a safe configuration.

We compute the effects of faults on the topology of the configuration space and construct optimal recovery motions for a set of faults. We describe an efficient algorithm for computing the optimal recovery motions for a large number of faults over the entire configuration space simultaneously. A performance measure, called *longevity*, quantifies the ability of the recovery motions to complete the task. From the performance measure fault tolerant paths are constructed.

We look at the simple task of positioning the end effector of a Puma 560 at a given point in the workspace. We compare the fault tolerant trajectory to an equivalent joint interpolated motion and show that our method produces trajectories which are significantly more fault tolerant.

1 Introduction

This paper investigates the problem of taking a task specification and generating a robot program which achieves the task with a maximum degree of fault tolerance. To maximize the fault tolerance of the robot, we must first ensure that the robot is redundant with respect to the task, and secondly we must ensure the robot avoids configurations which, in the event of a fault, leave the robot unable to complete the task.

First we introduce a method for specifying the task, called *Least Constraint (LC)*, which encourages the designer to specify the task using only the salient constraints of the task. Additional degrees of freedom of the robot with respect to the task can be used to maintain configurations which are fault tolerant. Also, since the robot need only be redundant with respect to the constraints, the inherent redundancy of the robot can be utilized, even in cases where the manipulator is not redundant for an arbitrary task.

An important feature of LC is that it allows for a natural means of incorporating dynamic knowledge, such as the addition of a fault, by inclusion of additional constraints to the task. The generality of the mechanism for including dynamic information allows virtually any type of fault to be modeled easily. In addition to the straightforward actuator faults which leave the robot with a frozen joint, we can also consider the presence of an unmodeled object as a fault.

The second aspect of achieving fault tolerance is to utilize configurations which, in the event of an actuator fault, permit the robot to complete the task. Using the *LC* specification we compute a performance measure which examines the topological properties of the configuration space, and the changes to these properties when an actuator fault is introduced. We define a utility measure for trajectories and compute optimal recovery motions for a set of faults. The recovery motions can be stored for their timely use in the event of a fault. The performance measure, called *longevity*, uses the utility of the optimal recovery motion for a fault as a measure the risk for the configuration. Using the worst-case fault for a configuration, the performance measure gives a lower bound of the utility of a configuration in the event of a single fault.

Consider the task of moving the end effector of a Puma 560 to a goal position in its workspace as illustrated in Fig. 1. The goal position is given by the small cube. (a) and (b) are the same distance from the goal in joint space, but have very different fault tolerant capabilities. Taking the worst-case fault for (a) and (b), resulting in a frozen actuator, we compute the recovery motions which minimize the distances to the goal. The endpoints for the recovery motions of (a) and (b) are given by (c) and (d) respectively. We see that the recovery motion of (a) is able to get much

*Supported in part by grants from NSERC and IRIS.

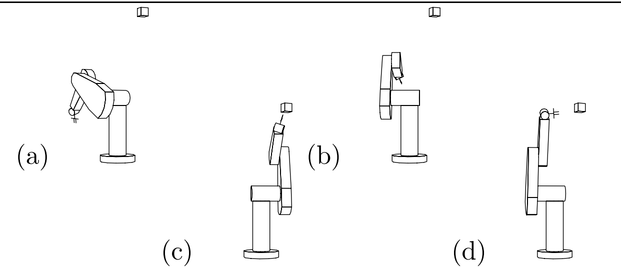

Figure 1: Two competing configurations, (a) and (b) are two configurations at the same distance in joint space from the goal. Freezing the most critical actuator for each we compute the optimal recovery motion to the goal (shown as the small cube). The endpoints for the recovery motions for (a) and (b) are given by (c) and (d) respectively.

closer to the the goal position as compared (b).

This illustrates our goal of constructing paths which, in the presence of a fault, are more likely to complete the goal. Ideally we would like the performance of the robot to degrade gradually with the addition of a fault so that, even in the cases where the robot is unable to completely satisfy the goal, it still approximates the goal to the best of its ability.

An additional benefit of computing the performance measure is that the optimal recovery motions for future faults are computed ahead of time (off line), allowing their expedient use in the event of a fault.

The paper is laid out as follows: In §2 we review previous work in fault tolerant robotics and related work. In §3 we introduce LC specification and describe fault modeling. Next we define a measure of satisfiability and utility of a trajectory in §3.2. In §3.3 we define the optimal recovery motion for a fault, and and develop a performance measure, quantifying the degree of fault tolerance of a configuration. In §4 we describe an efficient algorithm for computing recovery motions. In §5 we look at a simple task of positioning the end effector of a Puma 560 at a point in Cartesian space and show the effectiveness of the recovery motions. We compare the fault tolerant paths to the shortest joint interpolated motion for the same task.

2 Previous Work

Least constraint was introduced in [8] as method for programming large DOF robots. The use of LC to specify fault tolerant trajectories was presented in [12] which looked at the computation of a fault tolerant gait for a legged robot. This paper builds on [12] by using an algorithm for computing the performance measure and fault tolerant paths which is much more computationally efficient. This efficiency allows us to compute trajectories for a 5 DOF configuration space compared to a 3 DOF problem of [12], and allows for a much finer decomposition of the configuration space. This work also illustrates the use of the techniques for problems involving kinematically redundant manipulators.

Lewis and Maciejewski [6] have investigated the fault tolerant properties of kinematically redundant manipulators. By examining the self-motion manifolds of such manipulators, portions of the workspace which are inherently suitable for critical tasks can be identified.

Paredis and Khosla [11] constructed globally fault tolerant trajectories for redundant manipulators. A trajectory is said to be fault tolerant if at every point there exists an alternative trajectory to the goal which satisfied the task constraints. The alternative trajectories are not explicitly computed, but are chosen at run time by the redundancy management algorithm. The set of postures which are tolerant to a fault are computed by examining the kinematic mapping of the manipulator at discrete points along the path, and grouping these postures into regions. A path through these regions is constructed to find the final path.

Paredis and Khosla [10] have looked at the problem of designing a manipulator to be fault tolerant for point-to-point tasks. Like this work, contingent reachability of points in the workspace were characterized. LC allows us to specify a family of solutions, thus providing the needed flexibility for fault tolerance, while still satisfying the salient features of the task.

In [5] the task was defined using the manipulator velocity profile $\dot{x}(t)$. A measure of dexterity is defined by the smallest singular value of the Jacobian. Other task specific performance measures have been developed (see [13] for summary and example). The longevity performance measure differs in that it examines the global nature of the task.

Motion planning with uncertainty can be viewed as a form of fault tolerance (see [4] for survey). Donald considered the problem of error detection and recovery for manipulation tasks [2]. Manipulation strategies are produced that are guaranteed to succeed, or fail noticeably, under the specific model of dynamics.

Finally, a large body of work has been devoted to the problem of fault detection and identification (see [3] for a survey). We are concerned with the limits on fault tolerant operation of a robot, and not the methods for detecting the faults.

3 Task Specification

The task is defined using a set of algebraic inequalities describing the set of valid configurations over time. Let $\hat{C} = \mathbf{C} \times \mathbb{R}^+$, with $\mathbf{C}$ the configuration space of the robot, and $\mathbb{R}^+$ parameterizing time; $t = 0$ corresponds to the initial state of the robot. The set of feasible configurations over time, written $\mathcal{FCT} \subseteq \hat{C}$, is defined by a set of constraints as follows. The set of constraint functions, $h_{i,j} : \hat{C} \rightarrow \mathbb{R}$, and the corresponding set of predicates, $g_{i,j} : (h_{i,j} \leq 0)$, define the task specification $G = \{g\}$. The set of valid points is expressed in conjunctive normal form as:

$$G(\hat{q}) \overset{\text{def}}{=} \left\{ \hat{q} \in \hat{C} \,\middle|\, \bigvee_i \bigwedge_j g_{i,j}(\hat{q}) \right\}, \tag{1}$$

$$\mathcal{FCT} \overset{\text{def}}{=} \left\{ \hat{q} \in \hat{C} | G\left(\hat{q}\right) \right\}. \tag{2}$$

Using algebraic inequalities to define the task allows us to easily express static constraints, such as joint angle limits or configuration space obstacles. It is important to note that the trajectory itself is specified via dynamic constraints which drive the robot towards the goal. The designer must ensure that the solutions to Eq. 2 converge to the desired goal over time.

LC specification is particularly useful for describing systems such as an autonomous robot where one may not have an explicit task, but rather a set of safety constraints which must be satisfied throughout the entire trajectory, and also ensure the completion of the task. LC easily permits the inclusion of task constraints which are dynamic, such as obstacle constraints. Deadlines and time ordering of task constraints are easily specified since there is an explicit measure of progress, namely time. Such a specification frees the designer from explicitly constructing a trajectory, and allows the robot the freedom to choose the safest trajectory.

3.1 Representing Faults

Faults are modeled as the addition of further constraints to the specification G. As a result we have a means of computing the effects of a fault at a given configuration. Let $\Omega = \{f_i\}$ be a set of faults we are considering. Elements of Ω are quite general, and can be any algebraic inequality; however, later we will make some simplifying assumptions in order to make computation over Ω more tractable. Each fault, $f_i \in \Omega$, has an associated constraint function, $\alpha_i : \hat{C} \rightarrow \mathbb{R}$, and associated predicate, $\omega_i \equiv (\alpha_i \leq 0)$, which describes the fault constraint. A configuration $\hat{q}$ is feasible given a fault ω if $\hat{q} \in \mathcal{FCT}_{|_\omega}$, where $\mathcal{FCT}_{|_\omega}$ is the *reduced feasible configuration space* under the fault, ω, defined as:

$$\mathcal{FCT}_{|_\omega} = \{\hat{q} \in \mathcal{FCT} | \, \omega(\hat{q})\}. \tag{3}$$

An important aspect of the LC specification is its ability to describe the valid trajectories of the robot, as well as various fault scenarios, using the same formalism. The simplicity permits efficient computation of recovery motions for a large number of fault scenarios.

3.2 Satisfiability and Utility

Given a task described as Eq. 2, describing the static and dynamic constraints on the task, we seek a trajectory T which satisfies the specification. A trajectory, $T : \mathbb{R}^+ \rightarrow \hat{C}$, **satisfies** a feasibility set, F, for a time $t_{\max}$, denoted $\text{sat}(T, F, t_{\max})$, if the trajectory remains within the set F for all points in time up to $t_{\max}$:

$$\text{sat}(T, F, t_{\max}) \Leftrightarrow \forall t \in [0, t_{\max}] \; (T(t), t) \in F. \tag{4}$$

We can therefore define satisfiability of the task as $\text{sat}(T, \mathcal{FCT}, t_{\max})$, as well as satisfiability subject to a fault ω as $\text{sat}(T, \mathcal{FCT}_{|_\omega}, t_{\max})$. The key to effectively computing fault tolerant paths is the efficient computation of the topological properties of $\mathcal{FCT}_{|_\omega}$ over all faults, w (see §3.3 for details).

Definition 1 (Utility of a trajectory)
The utility *of a trajectory T, subject to the feasibility set F, is the maximum time the trajectory satisfies F:*

$$\mathit{util}(T, F) = \sup\{t \,|\, \mathit{sat}(T, F, t)\} \tag{5}$$

This definition of utility is based on the fact that our goal is to satisfy the specification for as long as possible into the future. Since convergence to the goal is assumed to have been verified by the designer beforehand, we may take time as a parameterization of the progress towards the goal.

Our definition of utility allows us to express tasks that are to be repeated ad infinitum. The utility, therefore, is a measure of how far into the future the configuration will satisfy the constraints. A trajectory, T, which is completely fault tolerant with respect to a fault, ω, has $\text{util}(T, \mathcal{FCT}_{|_\omega}) = \infty$.

The utility of a configuration $\hat{q_0} = (\mathbf{q_0}, t_0)$ is:

$$\text{util}(\hat{q_0}) = \max_{T \in \mathcal{T}, T(t_0) = \mathbf{q_0}} \text{util}(T, \mathcal{FCT}), \tag{6}$$

where $\mathcal{T}$ is the set of all trajectories.

3.3 Recovery Motions

Given a fault described by ω, at a given configuration $\hat{q}_0 = (q_0, t_0)$, we must compute the best recovery action. This is most concisely phrased as an optimal control problem where we seek the best trajectory, T, which maximizes the utility measure, $\text{util}(T, \mathcal{FCT}_{|_\omega})$, subject to the initial condition $T(t_0) = q_0$.

Definition 2 (Recovery Motion)
Given a fault described by ω, and an initial configuration, q_0, at time t_0, let $Rec((q_0, t_0), \omega)$ be the trajectory computed from the optimal control problem:

$$\begin{aligned} \mathit{Rec}(\hat{q_0}, \omega) &= \arg\max_{T \in \mathcal{T}, T(t_0) = q_0} \mathit{util}(T, \mathcal{FCT}_{|_\omega}), \\ \hat{q_0} &= (\mathbf{q_0}, t_0). \end{aligned}$$

The longevity measure maximizes the utility over the set of possible recovery actions for a given fault, and hence provides a natural measure of fault tolerance of a given configuration. Configurations with a large longevity measure have recovery motions which are effective in continuing to complete the task given the fault. In addition to giving us a method of computing the optimal recovery motion, we may also use the utility of the recovery motion as a measure of the robot's ability to sustain the fault w.

Definition 3 (Performance Measure, $L(\hat{q}, \omega)$) *The* longevity *of a configuration* $(\hat{q}) = (q_0, t_0)$*, given a fault described by,* ω *is the utility of the optimal recovery motion constructed from* $\hat{q}$*:*

$$L(\hat{q}, \omega) \quad = \quad util(Rec(\hat{q}, \omega)). \tag{7}$$

Hence the longevity is the maximum utility obtainable given a fault. Longevity is a meaningful performance measure for fault tolerance for the configuration because it evaluates the effectiveness of the optimal recovery action in the face of a potential fault. The worst-case longevity of a given configuration, denoted by L_W *is defined as:*

$$L_W(\hat{q}) \quad \stackrel{\text{def}}{=} \quad \min_{i \ s.t.\ \omega_i(\hat{q}) \leq 0} L(\hat{q}, \omega_i). \tag{8}$$

which describes the worst-case failure mode of the configuration $\hat{q}$*. The worst-case longevity, therefore, gives a lower bound on the utility of paths from* $\hat{q}$ *given a single fault in the future.*

While the longevity measure is relatively computationally expensive to compute, it has the benefit of providing the recovery actions for a given fault, which may be used at run-time. To compute the recovery motions efficiently we use a discrete form of the problem, and make use of dynamic programming techniques.

4 Computation of Rec$(\hat{q}, \omega)$

In order to compute the optimal recovery motions more efficiently we exploit features of the problem as well as making some simplifying assumptions. First we will use a discrete form of the configuration space to make the construction of paths through the space more tractable. Because optimal recovery actions of nearby configurations are likely to follow similar paths, we use a dynamic programming approach for computing the recovery actions. Secondly, we shall restrict the set of faults under consideration to make it more manageable while still being general enough to describe most common faults.

4.1 Decomposition of $\mathcal{FCT}$

While there are many approaches for decomposing the valid configuration space $\mathcal{FCT}$, we have chosen a simple uniform decomposition. Given n_j joints, and assuming an open kinematic chain, $\hat{C} \subseteq \mathbb{R}^{n_j} \times \mathbb{R}^+$. Dividing each of the (n_j+1) axes into n_d subintervals, we get a total of $N_c = {n_d}^{n_j+1}$, (n_j+1)-dimensional hypercubes, or *cells*, each $360/n_d$ degrees on a side.

Let $V = \{v_i\}$ be the set of vertices representing the (n_j+1)-dimensional cells. For each vertex v_i let $X(v_i) = (x_i^1, x_i^2, \cdots, x_i^{n_j}), x_i^j \in (1, \cdots, n_d)$ be the position of the cell in the lattice. The decomposition requires us to test whether the cell lies entirely inside $\mathcal{FCT}$, which involves a constrained optimization problem on the task constraint functions. We say that $Cell(v_i)$ is contained in $\mathcal{FCT}$ if the interior lies entirely inside $\mathcal{FCT}$. We use an iterative method using the Jacobian of the manipulator. The small size of the cells means that differential motions within the cell are quite accurately approximated by the Jacobian. As a result the performance of the constrained optimization behaves, in practice, as a well behaved linear optimization. Indeed for almost all cells which do contain the goal, g_p, the extremum occurred on the boundary of the cell.

Let $E = \{e_{ij}\}$ be the set of edges connecting the vertices. The adjacency relationships between cells, which is dependent on dynamic constraints on the robot, as well as the structure of the cells, are encoded by E. Additionally, the edges must ensure that paths which cross them are time-monotonic. A path is denoted by the ordered list of vertices through which the path passes.

The utility of a vertex, denoted util(v_i), is taken as the minimum over configurations interior to the cell. We will take the most conservative approach and take:

$$\text{util}(v_i) = \min_{\hat{q} \in Cell(v_i)} \text{util}(\hat{q}, \mathcal{FCT}).$$

4.2 Fault Model

Thus far we have placed few limitations on the types of faults which are considered using our methods; any fault which can be represented as a set of algebraic inequalities on $\hat{C}$ can be modeled. For the present we limit our scope to a simple fault model, leaving more complicated fault models for a future date. In accordance with the definition "1-fault tolerance" given in [10], we consider only faults which result in the actuator being frozen at the position at which the fault was detected. Using this limited fault model, a parameterized set of faults is given as:

$$\Omega \quad \stackrel{\text{def}}{=} \quad \{1, \cdots, n_j\} \times \mathbb{R}, \tag{9}$$

where $(i, r) \in \Omega$ indicates that the i-th actuator is frozen at position r. For the sake of simplicity assume that a sensor or actuator failure results in the joint being frozen at the point of failure, and that the position is known accurately. In general each actuator will have an associated sensor. If either fail the actuator should be frozen since it can not be controlled [5]. We do not address the problem of detecting and identifying the fault, but instead assume it has been performed for us prior to recovery action; it does not effect motion planning for optimal trajectories.

4.3 Recovery Motion Algorithm

The optimal recovery motion is the shortest motion in the reduced configuration space which has the largest utility measure. The computation of the optimal recovery motion for the set of vertices V, using edges E, is relatively easy given our fault assumption. A fault in joint j, while at node v_i, constrains the motion to a n_j dimensional hyper-plane of $\hat{C}$. The subspace of the robot with the frozen actuator, j, sometimes called a reduced order derivative [10], is the set of vertices:

$$rod(v_i, j) \quad = \quad \left\{ v_k \left| x_k^j = x_i^j \right. \right\}, \tag{10}$$

where the position of the j-th joint, x_i^j, is fixed. To compute the set of all recovery motions, we must consider each of the n_j actuators, at each of the n_d positions. For each fault we must compute the path through $rod(v_i, j)$ which maximizes the utility. The following algorithm describes how to compute the optimal recovery motion, given a fault, for all vertices simultaneously. We can represent the recovery motion as an array:

$$\text{rec}[1 \cdots N_c][1 \cdots n_j] \in V.$$

The recovery motions are stored as a linked list where rec$[i][j]$ gives the recovery motion from v_i for a fault in actuator j at position X_i^j. The recovery motion is constructed by following the linked-list until termination. We let rec$[i][j] = \emptyset$ when v_i has the optimal utility, hence the optimal recovery motion is null. Thus the array rec$[i][j]$ contains a compact representation of all of the recovery motions of the robot.

```
RecoveryMotions ($j \in [1, n_j], posn \in [1, n_d]$)
/* Given a fault in joint $j$ at position $posn$
 * compute the recovery motions of all cells in
 * $rod(v_i, j)$ where $x_i^j = posn$.*/
1. **For each** $v_k \in rod(v_i, j)$, compute connected
   components using Union_Find operation.
2. **For each** connected component $C$ Do
3.       **Let** $v_m$ have the maximum utility in $C$
4.       Perform BFS from $v_m$ to each vertex in $C$
   **endFor**
```

For the purposes of computing the topological properties of the reduced configuration space, we use a graph in which each cell is connected to the $(2n_j)$ which share a face. The number of vertices in $rod(v_i, j)$, is ${n_d}^{n_j}$, hence the time complexity of line 1 is $O({n_d}^{n_j})$ since Union_Find takes $O(n)$ for practical values of n. The breadth first search at line 4 takes $O(n_j {n_d}^{n_j})$, since at each step of the search we must examine each neighbor. Hence the overall time complexity is $O(n_j {n_d}^{n_j})$, which for practical problems has $n_j \ll n_d$, the time complexity becomes $O(N_c)$.

We can then define the longevity of a vertex v_i as:

$$L(i,j) \quad = \quad \begin{cases} \text{util}(v_i) & \text{rec}[i][j] = \emptyset \\ L(\text{rec}[i][j]) & \text{Otherwise} \end{cases} \tag{11}$$
$$i \in [1, N_c] \qquad j \in [1, n_j]$$

This simply involves following the linked list of the recovery motion to the largest utility value.

4.4 Computation of Paths

Thus far we have described the computation of the recovery motions and the related performance measure, longevity. What remains is the computation of the path from the initial configuration to the goal which maximizes this performance measure along the path. The algorithm for computing the paths is similar to Dijkstra's edge relaxation algorithm [1] for computing minimum-cost paths from a single source, however rather than using a simple weight to express the cost of a given edge, we use a the *sorted minimum performance* [9], defined below, which is better suited our problem. Additionally we use the set of nodes with the largest performance measure, $V_0 \subseteq V = \{v_i \,|\text{perf}(v_i) = \max_j \text{perf}(v_j)\}$, as the source node in Dijkstra's algorithm. To modify Dijkstra's algorithm for multiple source nodes involves adding a single vertex v_0, adjacent to each of the source nodes, connected with a zero weight edge.

The computational complexity is $O(V^2)$, however for sparse graphs we can use *modified Dijkstra's algorithm* to achieve $O((V+E)\log_2 V)$ or $O(E \log_2 V)$ (see [1, p. 527–531] for details). With a modification of the edge-relaxation procedure to use the sorted-minimum performance, the algorithm updates the paths so as to keep the safest paths. The sorted-minimum performance is a conservative estimate of the safety of a configuration and is described below.

4.5 Sorted Minimum Performance

Given a performance measure perf$(\cdot)$ to be maximized along a path, a seemingly natural metric for comparing paths is the average performance along the path. However, there are some difficulties difficulties with this metric. First, there is no penalty for allowing extremely large paths which stay in regions with good performance and never attaining the goal. Also from a practical point of view, we want a metric which penalizes the use of nodes with bad performance, but do not overly penalize their use if it is inevitable. A metric for paths which captures the above is the sorted minimum.

Given two paths $p = \{v_1, \cdots, v_n\}$, and $p' = \{v'_1, \cdots, v'_m\}$, let $z = \{z_i\}$ and $z' = \{z'_i\}$ be the respective lists of perf(v_i) and perf(v'_i), sorted in increasing order. We write $p \overset{\diamond}{>} p'$ if, when comparing in a lexicographical manner:

$$\exists j < \min(m,n), \left(\bigwedge_i^{j-1} z_i = z'_i\right) \wedge z_j > z'_j.$$

If z or z' are prefixes of the other, then the shorter path is ranked higher.

The dominant cost of the computation of the recovery motions is the cost of comparing the sorted minimum performances of two paths. For each path we store the minimum performance encountered along the path. In cases where the minimum performances are unique, the comparison takes constant time. Otherwise the paths must be sorted in ascending order, and then compared. If we assume that each vertex has degree δ, and the path lengths are bounded by p_{len}, an upper bound on the floating point comparisons is:

$$\delta N_c p_{\text{len}} \left(\log_2 N_c\right) \left(\log_2 p_{\text{len}}\right). \tag{12}$$

5 Results

Consider the task of placing the end effector of a Puma 560 at a given position in the workspace, with no constraints on the orientation. Since the last joint does not contribute to the positioning of the object, we consider only the first 5 actuators. The positioning task requires only 3 DOF, so we have 2 redundant DOF's. The DH parameters of the Puma 560 are simplified according to [7, p. 218-219]:

Link	Angle θ_n	Displacement d_n	Length l_n	Twist α_n	Range (°)
1	θ_1	660.4	0	+90°	−160, 160
2	θ_2	149.5	432	0°	−225, 45
3	θ_3	0	0	−90°	−45, 225
4	θ_4	432	0	+90°	−110, 170
5	θ_5	0	0	−90°	−100, 100
6	θ_6	56.5	0	0°	−266, 266

We define a goal position, g_p, in the robot's workspace where the end effector is to be positioned. To simplify the specification of the task, as well as permitting a simple interpretation of the results, we define the proximity of the end effector as:

$$\text{prox}(\mathbf{q}) = \frac{d_{\max} - ||p(\mathbf{q}) - g_p||}{d_{\max}}, \tag{13}$$

where $p(\mathbf{q})$ is the position of the end effector, and $d_{\max}$ is the maximum distance of any point of the workspace to the goal point. The range of prox$(\mathbf{q})$ is the unit interval, and is at a maximum when the end effector is at the goal. We then define the task as a simple relation on the proximity:

$$G(\hat{q}) = \{\hat{q} = (\mathbf{q}, t) | \text{prox}(\mathbf{q}) \geq t \wedge JL(\mathbf{q})\}. \tag{14}$$

where $JL(\mathbf{q})$ are the joint angle constraints. Thus we require that the robot reach the goal in one time unit. We can interpret the longevity $L(\hat{q}, \omega)$ as the proximity of the closest possible configuration attainable after the fault occurred, and therefore can be interpreted as a dimensionless distance.

Also, since we are not imposing any time constraints throughout the motion of the manipulator, we can omit the time dimension of the decomposition, and let $\hat{q} = \mathbf{q}$, and reduce dimension of the space from 6 to 5.

We use the worst-case longevity, $L_{\text{W}}(\mathbf{q})$ in constructing the performance measure of a configuration.

$$\text{perf}(\mathbf{q}) = \begin{cases} L_{\text{W}}(\mathbf{q}) & \text{if } L_{\text{W}}(\mathbf{q}) < 1 \\ 1 + \text{util}(\mathbf{q}) & \text{if } L_{\text{W}}(\mathbf{q}) = 1 \end{cases} \tag{15}$$

This ensures that we rank configurations first according to the ability to complete the task given a failure, and secondly according to the utility of the configuration. Paths which maximize this performance measure ensure that much of the path lies in regions where we are guaranteed to be able to position the end effector at the goal position exactly, (*i.e.* perf$(\mathbf{q}) \geq 1$).

Taking the goal position $g_p = (55, -430, 1472)$, and taking $n_d = 20$, resulted in a total of 470,400 cells within the joint angle limits, each 18° on a side. Each cell is connected to all valid cells that share a face, thus each vertex has a minimum degree of 5, and a maximum degree of 10. Paths were constructed using the center points of each of the cells through which the path passed. The initial configuration was taken to be $\mathbf{q} = (171°, -171°, 27°, 153°, -27°)^T$, corresponding to a manipulator position of $(101.6, 155.7, 216.0)^T$ measured in mm from the center of the base.

To quantify the degree of fault tolerance of the path generated by the longevity performance measure, we computed a joint interpolated motion from the initial configuration to the closest configuration satisfying the goal. The shortest-distance joint interpolated motion thus served as a baseline for our algorithm. Fig. 2 gives the trajectories of the Puma 560 for both the fault tolerant (FT) path generated with the longevity performance measure, and the joint interpolated motion (JI). Both paths were 20 steps in length. The optimal recovery motions for the worst-case faults were computed at each step for both the FT and JI paths and are shown in Fig. 3.

Fig. 4 gives the utility and worst-case longevity measures for each step of the FT and JI paths. Re-

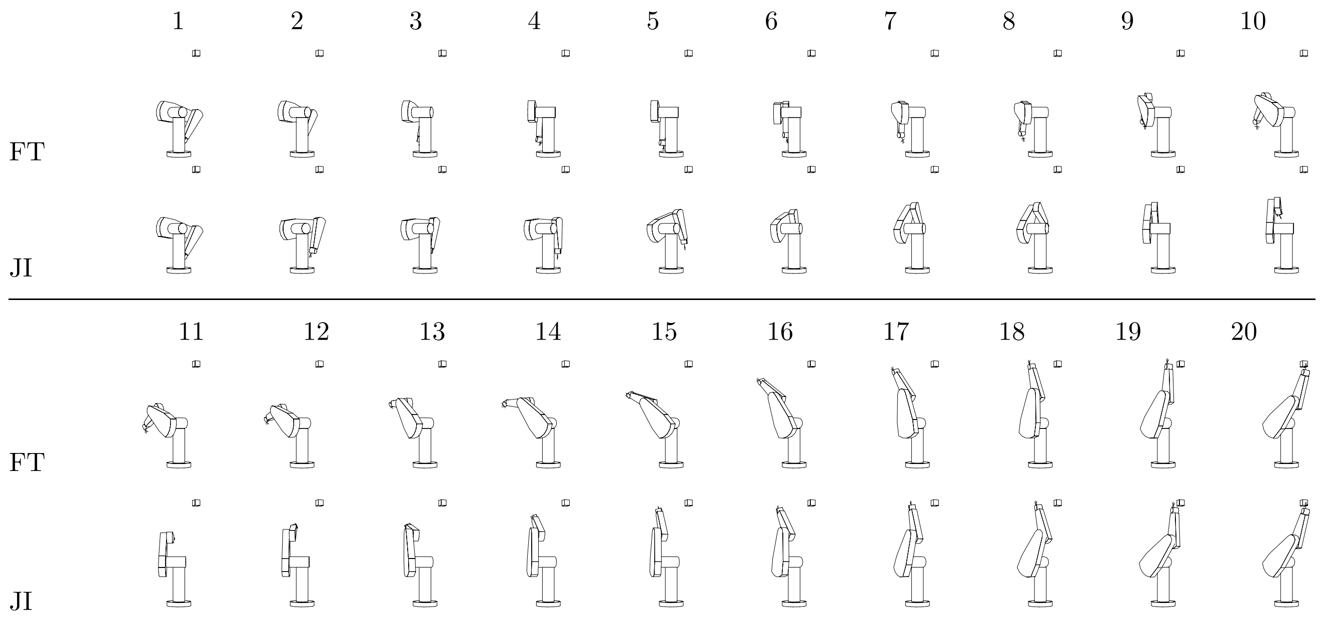

Figure 2: Trajectories fault tolerant path (FT) generated with $L_{\mathrm{W}}()$ measure, and joint interpolated motion (JI).

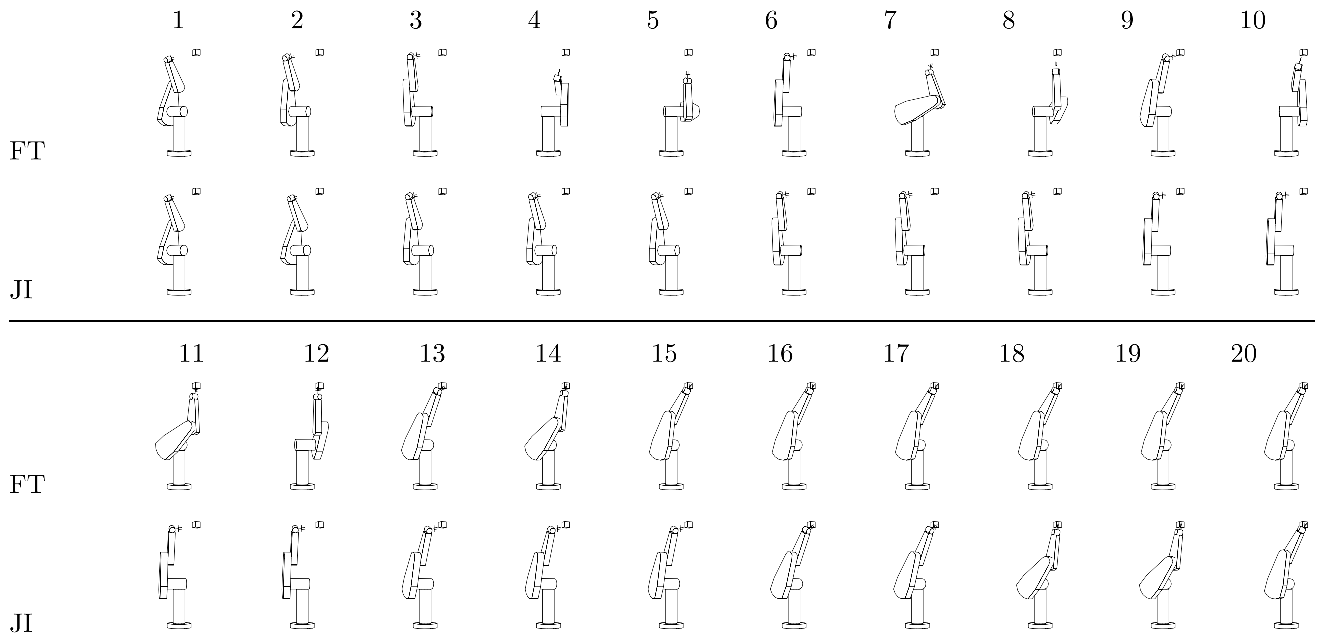

Figure 3: Endpoint configurations for optimal recovery motions for the worst-case faults for both the fault tolerant path (FT) and the joint interpolated path (JI).

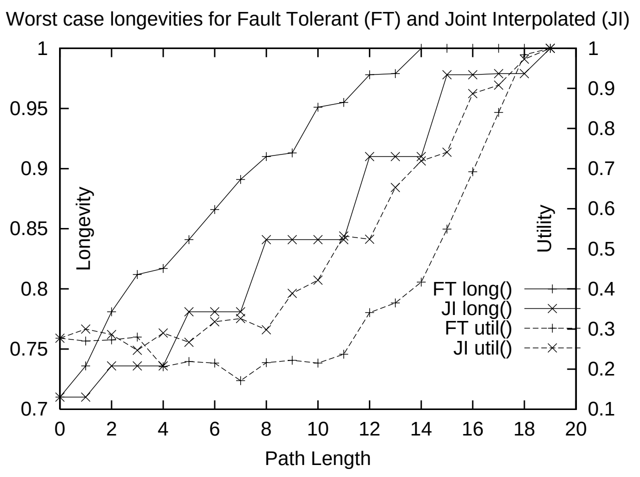

Figure 4: Longevity and utility vs. path length for optimal and straight-line motion paths.

call that the longevity can be interpreted as the proximity of the endpoint of the optimal recovery motion for the worst-case fault. We can see that the joint interpolated motion has a much closer proximity, especially through steps 1–14. We can compute the actual distance knowing $d_{\max} = 1826mm$, so $d(\mathbf{q}) = (1 - \text{prox}(\mathbf{q}))d_{\max}$. The proximity value (utility) at step 14 is only 0.365 (1160mm) for the FT path while the JI path is 0.653 (634mm). While the proximity values of the JI path are almost monotonically increasing, the values for the FT path remain almost constant at 0.725 (502mm) for steps 1-11.

Looking at the longevity values for both paths we see that the FT path is able to make considerable gains over the interpolated motion. The mean longevity value for the FT path is 0.907, and for JI motion it is 0.849 for a difference of 0.058 (106mm). This means that given a fault in both FT and JI paths, the FT path will, on average, get 106mm closer to the goal. The maximum difference in the longevity values occurred at step 11 where the FT path had a longevity value which exceeded the JI path by 0.114. A significant feature of the longevity plots is that the longevity values remain at the optimal value of unity from step 14 to the end of the motion for the FT path. The JI path on the other hand does not reach a longevity value of unity until step 19. This means that, even though the proximity at step 14 is only 0.365 at it is guaranteed to reach the goal under any 1-fault scenario. From the plots of utility and longevity it is clear that the FT path is optimizing the longevity measure by choosing configurations which are not closer to the goal, but rather are safer.

6 Conclusions

We have demonstrated an effective algorithm for computing fault tolerant motions for a 5 DOF manipulator. We have further developed the LC framework and related performance measures as an effective means of computing fault tolerant trajectories for robots performing a specific task. The framework allows for a unified method for expressing trajectories and recovery motions of the robot, as well as a method for quantifying the risk of a given configuration to the completion of the task. The fast computation of recovery motions for a large set of faults has been developed which can be stored efficiently to expedite their use in the event of a fault. The performance measure *longevity* has been shown to be an effective measure of the fault tolerance of a configuration since it considers the topological properties of the configuration space, and the effects of the faults on the available recovery motions.

References

[1] Thomas H. Corman, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms (McGraw-Hill edition).* McGraw-Hill, 1990.

[2] B. R. Donald. *Error Detection and Recovery for Robot Planning with Uncertainty.* PhD thesis, MIT Department of Electrical Engineering and Computer Science, 1987.

[3] Paul M. Frank. Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy – a survey and some new results. *Automatica*, 26(3):459–474, 1990.

[4] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA., 1991.

[5] Christopher L. Lewis and Anthony A. Maciejewski. Dexterity optimization of kinematically redundant manipulators in the presence of failures. *Computers and Electrical Engineering*, 20(3):273–288, 1994.

[6] Christopher L. Lewis and Anthony A. Maciejewski. An example of failure tolerant operation of a kinematically redundant manipulator. In *International Conference on Robotics and Automation*, pages 1380–1387, 1994.

[7] Phillip John McKerrow. *Introduction to Robotics.* Addison-Wesley Publishing Co., Sydney, 1991.

[8] Dinesh K. Pai. Least constraint: A framework for the control of complex mechanical systems. In *Proceedings of the American Control Conference*, pages 1615–1621, 1991.

[9] Dinesh K. Pai and L. M. Reissell. Multiresolution rough terrain motion planning. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, volume 2, Pittsburgh, PA., 1995.

[10] C. J. J. Paredis and P. K. Khosla. Kinematic design of fault tolerant manipulators. *Computers and Electrical Enginerring*, 20(3), 1994.

[11] Christiaan J. J. Paredis and Pradeep K. Khosla. Global trajectory planning for fault tolerant manipulators. In *1995 IEEE/RSJ International Conference on Intelligent Robotis and Systems*, volume 2, 1995.

[12] Scott K. Ralph and Dinesh K. Pai. Fault tolerant locomotion for walking robots. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 130–137, Monterey, CA, July 10–11 1997.

[13] Kees van den Doel and Dinesh K. Pai. Constructing performance measures for robot manipulators. In *International Conference on Robotics and Automation*, pages 1601–1607, 1994.