Fault Tolerant Locomotion for Walking Robots

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 general method of planning fault tolerant motion for a robotic task based on the **least constraint** (LC) framework, which uses a set of constraints on the robot's configuration over time. A performance measure called **longevity** is defined which, for a given configuration and type of fault, describes the potential for future progress towards the goal. This measure examines the connectivity of the configuration space given failure. An algorithm for computing longevity based on dynamic programming is described. Using the longevity computed at discrete points, a path is computed which is optimally fault tolerant. The set of paths which maximize longevity form a contingency plan for faults occurring at each point.

Using LC we specify a gait for a four-legged walking robot called a "Platonic Beast". A prototypical step is produced using the longevity measure, which we compare to a straight-line motion implementation. The optimal longevity paths are shown to be significantly more fault tolerant than the straight-line motion.

1 Introduction

Before describing the methods of constructing fault tolerant behavior, we must clearly define what we mean by fault tolerance. We assume we are given a specification for a robotic task that is unambiguous and complete: every state of the system is classified as valid or invalid [1]. The process of entering an invalid state is called a **failure** which is irrecoverable. **Faults** may cause erroneous states in the system which are still valid. **Fault tolerance** is the capacity of the system to detect and correct erroneous states before they lead to system failures.

To successfully tolerate a fault we must be have some degree of redundancy in our sensors and actuators. Much of the previous work in fault tolerant robotics has focused on kinematically redundant robots executing motions which have been explicitly prescribed. For example we may specify a set of via-points in the robot's workspace [12, 13], or a velocity profile of the robot [6]. Provided we stay within the kinematically redundant workspace of the robot, we can sustain a fault while continuing to follow the commanded motion. The choice of the inverse kinematics solution can be made according to dexterity or other performance measures to maximize the post-failure capabilities of the robot [7].

Instead of requiring the robot to be redundant with respect to all tasks, we have proposed to change the task specification to one based on constraints [9]. We believe that this enables us to exploit the full potential for fault tolerance of the robot with respect to a task. Extra degrees of freedom of the robot with respect to the task can be used to choose configurations which are maximally fault tolerant.

The specification method, called **least constraint** (LC) [9], encourages the user to capture the salient features of the task, and allows incremental inclusion of further constraints into the system when new knowledge is acquired such as new sensor information or the detection of a fault. This permits a straightforward way of computing the set of reachable configurations given a particular fault. Example 1 describes a planar walking robot with three 2 DOF legs.

This paper makes use of the LC framework to develop a performance measure, called **longevity**, which quantifies the amount of fault tolerance at a particular configuration. A fault may alter the topology of the feasible configuration space, changing the feasible regions which are reachable from a given configuration. Given a fault, the longevity measure examines the topological properties of the reduced set of feasible configurations, and constructs a recovery action which maximizes the progress measure. Thus the longevity measure quantifies the degree to which a robot with a fault can continue to perform the task. The longevity measure is then used to produce a trajectory which is optimally fault tolerant with respect to the task.

Example 1 (Walking in 2D)
Fig. 1 shows a three legged walking robot, a planar version of the 4-beast depicted in Fig. 3, and described in [10]. Assume only workspace and static stability constraints are present, and we are to move from right to left towards the goal region. Stability is specified by requiring the center of mass to satisfy inequality constraints g_1 and g_2 shown in the figure. Horizontal motion is achieved by a moving constraint $g_4(t)$. If a fault is introduced in the present configuration rendering the distal right joint immovable, the body position is constrained to lie on circular arc (shown as a dotted arc). By not prescribing a specific trajectory, the robot is still able to reach the goal using the reduced workspace.

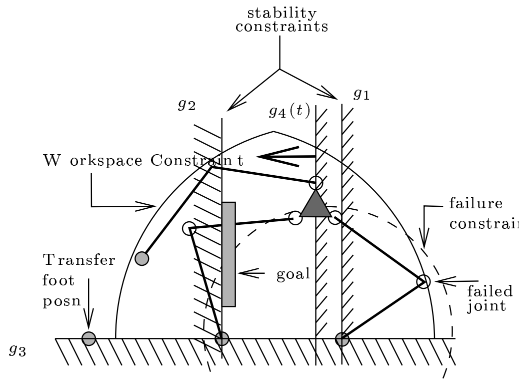

Figure 1: 2-D Walking Robot.

We examine legged locomotion as a specific example of a task which may be loosely defined via a set of constraints. The example is based on a four-legged spherically symmetric walking machine called a "Platonic Beast" [10]. Next we will construct a motion which maximizes the likelihood of continued service in the event of a sensor or actuator fault during execution.

Previous work in the generation of gaits for walking robots have looked at various criteria for their selection including: minimizing total energy consumption, maximizing stability with respect to disturbance forces, and many others. To the best of our knowledge this is the first attempt of producing a gait which maximizes fault tolerance with respect to actuator/sensor failures. This property is important for safety-critical or mission-critical applications.

This work differs from existing work on fault tolerant robotics in that it provides an explicit method for specifying a robot's behavior based on a set of task constraints. Using this specification, a performance measure is computed which measures the effects of possible faults on the robot's ability to complete the task. The task specification is completely general and can be applied to any robot task.

Section 2 summarizes previous work in fault tolerant robotics and related matters. In section 3 we start by giving a specification for a single step of the robot. We assume that the robot is slow moving, and thus requires static stability during the entire motion of the robot. In section 4 we develop a performance measure called longevity, a global measure on the feasible configuration space; experimental results are analyzed in section 5.

1.1 Benefits of the Approach

The most important benefit of using the LC approach is that it is applicable to a much larger class of task/robot combinations. Since our methods of describing fault tolerant behavior are defined with respect to a set of task constraints, the methods for producing fault tolerant behavior are still meaningful for non-redundant manipulators.

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 achieved 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.

Our formulation represents a fault as an additional task constraint, allowing simple computation of the effects of a fault on the set of feasible configurations. In contrast [12] refer to a manipulator with a frozen joint as a "reduced order derivative" which is modeled as a new manipulator with one fewer actuator.

We argue that the explicit inclusion of time in the task specification provides us with a natural utility of a given path. The longevity measure maximizes this 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.

The longevity differs from other performance measures, such as proximity to the goal, in that it ranks a configuration according to the utility of the optimal recovery action, rather than using local properties of the configuration. Longevity is particularly meaningful for tasks in which there is no specific goal, such as those described as a set of constraints.

Because the longevity measure is dependent only on the configuration, it can be re-used for computing many trajectories, each of which may differ in the initial or final configurations, or for differing fault probabilities.

The computation of the longevity explicitly computes the most effective recovery action which can be used later on-line. This differs from [13] where the recovery actions are stored implicitly in the redundancy resolution algorithm. By explicitly computing the recovery actions we ensure that we are not confronted with problems faced by redundancy resolutions algorithms: becoming stuck at a singularity, or choosing a trajectory that violates joint angle or obstacle constraints.

2 Previous Work

Lewis and Maciejewski [7] 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 [13] 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 [12] have looked at the problem of designing a manipulator to be fault tolerant for point-to-point tasks. Like the work presented here, contingent reachability of points in the workspace are characterized. LC allows us to specify a family of solutions, thus providing the needed flexibility for for fault tolerance, while still satisfying the salient features of the task.

Alternatively we can define the manipulator's task by specifying the manipulator velocity profile $\dot{x}(t)$ [6]. The measure of dexterity is defined by the smallest singular value of the Jacobian. Other task specific performance measures have been developed (see [14] 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 [5] for survey). Donald considered the problem of error detection and recovery for manipulation tasks [3]. 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 [4] for a survey). We are concerned with the theoretical limits on fault tolerant operation of a robot, and not the methods for detecting the faults.

3 Task Specification

The task is defined as a set of feasible 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; $t = 0$ is take to be 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, $h_{i,j} : \hat{C} \to \mathbb{R}$, 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:

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

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

When explicit time constraints are not present, it is still useful to use time, or some other single parameter, to characterize the progress towards the goal.

In example 1, the specification consists of a set of four constraints: $g_1(t)$ and g_2 ensure the body remains over the support interval; g_3 ensures that the body remains above the ground; and the motion from right to left is achieved with the time dependent constraint $g_4(t)$.

3.1 Specification of a Tumble Step

We will look at the problem of constructing a gait for a walking robot from a set of task constraints with configuration space, $\mathbf{C} = \mathbb{R}^3$. Figure 2 gives the initial placement of the 4-beast. Leg link lengths, as well as the length of each of the tetrahedron edges is unity. For the purposes of the motion in Fig. 2 we will label the feet as L, R, T and B meaning the "left", "right", "top" and "back" feet respectively (See Fig. 2).

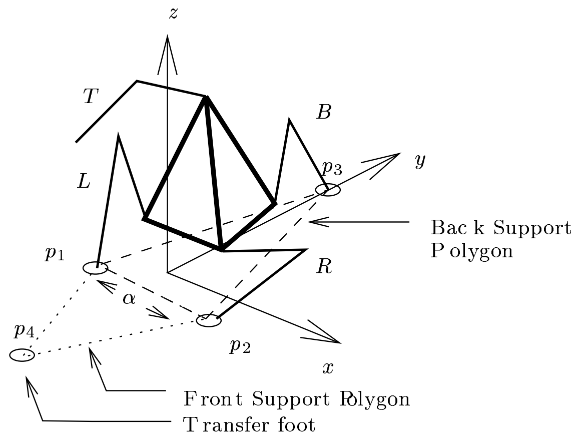

Figure 2: Starting configuration for 4-beast.

The goal is to produce a "tumble step" [10] in which the body translates in the $(-y)$-direction, placing the top leg in front, and removing the back foot. Fig 3 shows a sequence of images of the platonic beast taking a tumble-step up a 20 degree slope (see [10]). The foot positions p_1, p_2, p_3, and p_4 are chosen to lie in an equilateral triangle with edge length α. This maximizes the size of the feasible workspace, $\mathcal{FCT}$, as well as giving a symmetry to the kinematics of the step.

Let $p_{cm} = (x, y, z)$ denote the center of mass of the beast. To simplify the analysis, we only consider translations of the beast, and not rotations. Each step consists of a translation from one support triangle to the next, with a rotation at the end to reorient for the next step.

For static stability, we require that the projection of p_{cm} into the xy-plane lie within one of the two support triangles, $\triangle p_1p_2p_3$ or $\triangle p_1p_2p_4$. The static stability constraint for $\triangle p_1p_2p_3$ is:

$$\begin{aligned} h_{p_1p_2p_3}(p_{cm}, t) &= \bigwedge_{i=1}^{3} (h_{1,i}(p_{cm}, t) \leq 0), \quad (3) \\ \text{where} \quad h_{1,1}(p_{cm}, t) &= p_{cm} \times (p_2 - p_1) \cdot \hat{k} \\ h_{1,2}(p_{cm}, t) &= p_{cm} \times (p_3 - p_2) \cdot \hat{k} \\ h_{1,3}(p_{cm}, t) &= p_{cm} \times (p_1 - p_3) \cdot \hat{k}, \end{aligned}$$

with $\hat{k} = (0, 0, 1)^T$. A similar constraint, $h_{p_1p_2p_4}$, can be written for the other support triangle. The kinematic constraints relate to the relative position of the feet to the respective vertices of the tetrahedron. Reachability of point p_1 by the left leg, for example, is given as:

$$h_{p_1} = \left\| p_1 - p_{cm} - (\alpha - 1) \begin{pmatrix} \frac{-1}{2} \\ \frac{-1}{2\sqrt{3}} \\ \frac{-1}{2\sqrt{6}} \end{pmatrix} \right\| - 2, \quad (4)$$

similarly we can define h_{p_2}, h_{p_3} and h_{p_4}. The constant 2 arises from the radius of the spherical workspace of each leg. We introduce a single "driving constraint", $g_y \equiv (h_y \leq 0)$, which forces the robot to move forward:

$$h_y(p_{cm}, t) = y_0 + y + t, \qquad y_0 = 0.2165. \quad (5)$$

The choice for y_0 arises due to our choice to discretize the feasible configuration space (see §5.1). The specification for the feasible configurations is:

$$\begin{aligned} G \;=\; & (g_y \wedge \\ & ((g_{p_1p_2p_3} \wedge g_{p_1} \wedge g_{p_2} \wedge g_{p_3}) \vee \quad (6) \\ & (g_{p_1p_2p_4} \wedge g_{p_1} \wedge g_{p_2} \wedge g_{p_4}))) . \end{aligned}$$

which simply states that we must move forward, and remain in a reachable position above one of two support regions.

4 Performance Measures

Most previous examples of performance measures have exploited only local information about the manipulator's configuration [6] or the robot's task [14]. We propose a fault dependent measure called longevity which examines the connectivity of the configuration space given a fault.

We will now define our fault model, as well as the corresponding fault constraints which further constrain the specification.

4.1 Fault Model

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[6]. We do not address the problem of detecting and identifying the fault, but instead that it has been performed for us prior to recovery action; it does not effect motion planning for optimal trajectories.

Faults will be modeled as the addition of further constraints to the specification G. As a result we have

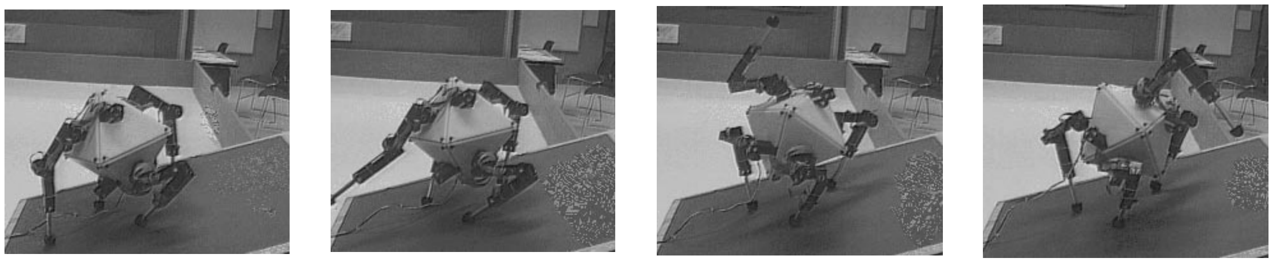

Figure 3: Canonical tumble with 4-beast prototype up a 20 degree slope (taken from [10]).

a simple means of computing the effects of the introduction of a fault into the system at a given configuration.

Let $\Omega = \{1, \cdots, d\} \times \mathbb{R}$ be the set of possible faults where $(i, r) \in \Omega$ indicates that the i-th actuator is affected and was at position r when the fault occurred. We could extend this to include other fault models in the future. 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.

Longevity is a mapping, $\hat{C} \times \Omega \rightarrow \mathbb{R}$, which takes a fault, presumed to occur at that configuration, and produces a scalar quantity. This quantity, defined formally below, is the latest time that a given point can remain in $\mathcal{FCT}$ given the fault.

Definition 1 (reachability set)
The set of points in $\hat{C}$ which are reachable from a point $\hat{q} \in \hat{C}$ given a fault described by ω, is the largest path connected subset of $\mathcal{FCT} \cap \left\{c \in \hat{C} | \omega(c)\right\}$ containing $\hat{q}$. Let $R(\hat{q}, \omega)$ denote this set. If $\hat{q}$ and ω are not consistent, (i.e. $\neg\omega(\hat{q})$), then we set $R(\hat{q}, \omega) = \perp$.

From this we define the longevity of a point $\hat{q}$ with a fault ω as the latest time that $\hat{q}$ can remain in $R(\hat{q}, \omega)$.

Definition 2 (longevity)
For a given failure described by ω, the longevity of a point $\hat{q}$ is:

$$L(\hat{q}, \omega) \overset{\text{def}}{=} \max_{(q,t) \in R(\hat{q}, \omega)} t, \tag{7}$$

the largest value of t for which there exists a point in the reachable set.

We can then define the worst case and average case longevities as:

$$L_{\text{worst}}(\hat{q}) \overset{\text{def}}{=} \min_{f_i \in \Omega \ s.t. \ \omega_i(\hat{q})} L(\hat{q}, \omega_i), \text{ and} \tag{8}$$

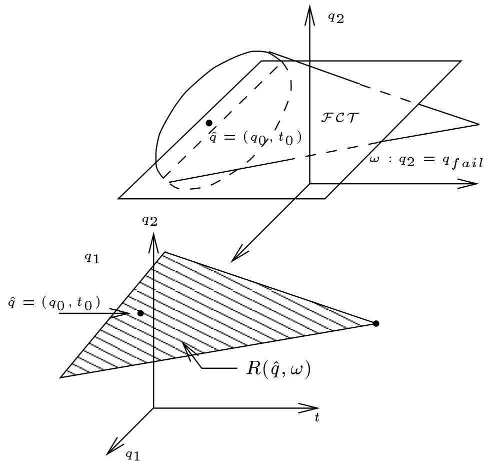

Figure 4: Computing the longevity of a point $\hat{q}$.

$$L_{\text{avg}}(\hat{q}) \overset{\text{def}}{=} \frac{\sum\limits_{f_i \in \Omega \ s.t. \ \omega_i(\hat{q})} L(\hat{q}, \omega_i)}{|\{f_i \in \Omega \,|\omega_i(\hat{q})\}|}. \tag{9}$$

The longevity of a point is a meaningful measure of potential for fault tolerant operation because it gives an upper bound on the length of time that we may continue to satisfy the task specification.

To illustrate the process of computing the longevity of a point, $\hat{q} = (q_0, t_0)$, consider the example given in Fig. 4 with a conical $\mathcal{FCT}$, and two actuators, q_1 and q_2. A failure in q_2 at time t_0 results in the actuator being frozen at $q_2 = q_{fail}$. The additional constraint, ω, results in the reduced configuration space $R(\hat{q}, \omega)$.

Computing the longevity requires finding the point within this region with maximum time, t_l. This is a constrained optimization problem, which at present, is implemented using gradient descent. Section 6 describes a more efficient algorithm which we are currently in the process of implementing.

5 Experiments

The process of constructing paths in the feasible configuration space first involves the decomposing the feasible space into a set of discrete nodes. The choice for the path performance measure is explained below. Paths are constructed through the center point of each cell using the performance at this point. The search was given an initial position for the start of the search, but no endpoint was specified. Instead we only constrained the the point to lie within G. Finally ,the discrete path was smoothed using a cubic spline.

5.1 Decomposition of $\mathcal{FCT}$

To construct paths in the feasible configuration space, the space was first decomposed into an oct-tree with cells marked as either `VALID`, `INVALID`, or `MIXED` [8]. The decomposition was performed to a depth of 3 and all `VALID` cells collected, yielding 56 valid cells. This depth was sufficient for capturing most of the detail of the configuration space, while still permitting manageable path searching times.

5.2 Path Performance

There are two natural candidates for the performance of a cell, L_{worst} and L_{avg}. Additionally we may choose to take the average, or the "sorted minimum" [11] performance along the path. The sorted minimum performance criterion sorts paths increasing order of performance, and then compares them in a pair-wise lexicographic manner. Using the average performance along a path has the effect of producing paths which meander in regions of high performance while not proceeding towards the goal. Using the minimum performance removes this problem, but introduces a potential for "ravine effects" in which two paths which contain a common region of poor performance will not be compared meaningfully using the minimum performance value. The poor behavior of the average path performance metric was exhibited with the longevity paths, producing paths which orbited regions with large z-values while not proceeding towards the goal. Unless otherwise specified we will use the minimum path performance criteria.

5.3 Path Analysis

Next we shall attempt to quantify the degree of the fault tolerance of each of the paths. As a benchmark for performance we will compare the trajectories to a straight-line motion. The straight line motion in parametric form is:

$$p_0(t) = \begin{pmatrix} 0 \\ 0.2165 \\ 0.9375 \end{pmatrix} + t \begin{pmatrix} 0 \\ -1 \\ -0.325 \end{pmatrix}. \quad (10)$$

This path is chosen as the shortest straight-line path passing through the centers of starting cell to the smallest attainable y-value. Fig. 5 shows the paths generated with L_{avg} and L_{worst} as well as the straight-line motion implementation of the same task. The total path lengths for the straight-line motion and L_{avg} and L_{worst} paths is 0.607, 1.58 and 1.60 units respectively, indicating longevity paths are approximately 2.5 times as long as the straight-line motion.

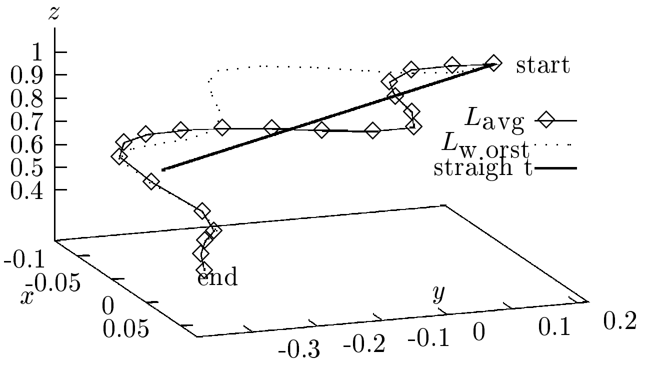

Figure 5: Longevity paths generated with $\alpha = 1.0$ using L_{avg} and L_{avg} and straight-line motion.

Since at the point of maximum longevity, $\hat{y} = y_0 + y + t$, we can compute the value of y which corresponds to the longevity value allowing us to interpret longevities as displacement.

$$\hat{y}_{\text{avg}}(\hat{q}) \overset{\text{def}}{=} -y_0 - L_{\text{avg}}(\hat{q}) \quad (11)$$

$$\hat{y}_{\text{worst}}(\hat{q}) \overset{\text{def}}{=} -y_0 - L_{\text{worst}}(\hat{q}). \quad (12)$$

To compare each path, 20 samples were taken along the trajectory such that the arc length between samples was equal. For each sample $\hat{y}_{\text{avg}}$ and $\hat{y}_{\text{worst}}$ was computed, as well as the y-positions for the straight line motion, denoted $\hat{y}_{\text{avg},S}$ and $\hat{y}_{\text{worst},S}$. The results are depicted in Fig. 6.

Comparing the average-case fault behaviors of $\hat{y}_{\text{avg}}$ against the corresponding $\hat{y}_{\text{avg},S}$ we see that the longevity path consistently performs much better. If we were to arbitrarily define success as reaching a y value of say, -0.35, we would see that over half of the longevity path would be fault tolerant on average.

Comparing the worst-case fault behaviors of $\hat{y}_{\text{avg}}$ against $\hat{y}_{\text{worst},S}$ we see that the longevity path consistently performs significantly better. The larger relative improvement is likely due in part to the fact

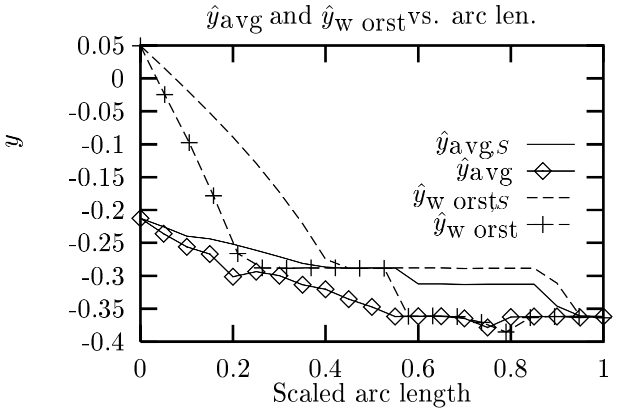

Figure 6: Final y-positions, $\hat{y}_{\rm avg}, \hat{y}_{\rm worst}$ and end positions $\hat{y}_{{\rm avg},S}$ and $\hat{y}_{{\rm worst},S}$.

that the minimum path performance criteria was used which is better suited for computing worst-case paths than for average-case paths.

6 Computing Longevity

When computing the longevity for a large number of points there is a great deal of wasted computation since for nearby points the paths through $\mathcal{FCT}$ which realize the optimal longevity will often be very similar.

We can avoid this wasted computation by using dynamic programming to compute the longevities; however it requires that we use a discrete form of the problem [2]. Since the results of partial paths are kept during the computation, we can compute the longevities of an entire set of discrete cells at the same time.

Computing longevity is essentially the same problem as constructing an optimal control policy for a system with a discrete sets of states and control inputs, with the difference that we are attempting to construct optimal trajectories subject to the introduction of the additional fault constraints. The collection of paths which maximizes the longevities for varying configurations thus comprises a contingency plan for each possible fault along the path.

In the discrete form of the problem $\mathcal{FCT}$ is decomposed into a cells labeled Q_i, $\mathbf{Q} = \{Q_i\}$, such that each cell lies entirely within $\mathcal{FCT}$. The interior of a cell is denoted $Cell(Q_i)$. The adjacency relationships between cells, which is dependent both on dynamic constraints on the robot, as well as the structure of the cells, is encoded in the edges $e_{i,j} \in E$.

We will let $L^*(Q_i)$ be the longevity of all points in

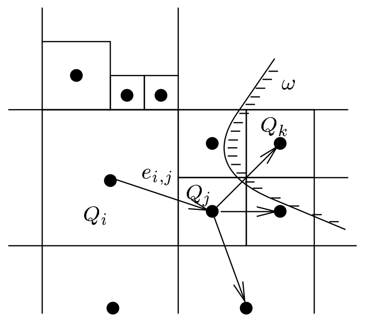

Figure 7: Discrete longevity computation.

$Cell(Q_i)$ with no fault. Computing the longevity at node Q_i subject to failure constraint ω requires that we search the graph for nodes that are path-connected to Q_i, and do not violate ω. More specifically, let $\tau(Q) : \mathbf{Q} \to 2^\Omega$ be the set of faults that are "tolerable" at Q, that is:

$$\tau(Q) \overset{\rm def}{=} \{f_i | \exists \hat{q} \in Cell(Q), \omega_i(\hat{q})\}. \qquad (13)$$

For example, in Fig. 7, $\omega \in \tau(Q_i), \omega \in \tau(Q_j)$, but $\omega \notin \tau(Q_k)$. Likewise we can define a set of faults tolerable over a path as:

$$\tau(Q_1, \cdots, Q_n) \overset{\rm def}{=} \bigcap_{i=1}^{n} \tau(Q_i). \qquad (14)$$

Key to the efficient computation of the longevity is the simplicity of τ. In general $\tau(Q)$ will be a small set of intervals:

$$\tau(Q_j) = \{(i, (r_{\min}, r_{\max}))\}, \qquad (15)$$

meaning all faults of the i-th actuator with position in the range $(r_{\min}, r_{\max})$ are tolerable. Computing this set of intervals involves solving a small constrained optimization problem for each cell. This optimization problem will generally be much easier to solve than the corresponding longevity constrained optimization problem.

Given τ we can now compute the longevities $L(Q_i)$ iteratively. We will let

$$\begin{aligned} \tau_k^0(Q_i) &= \{(r_1, r_2, L^*(Q_i)) | \\ &\quad (k, (r_1, r_2)) \in \tau(Q_i)\} \end{aligned} \qquad (16)$$

which is a set of 3-tuples in which r_1 and r_2 give the range over which we can tolerate a fault in actuator k, and $L^*(Q_i)$ is a conservative estimate for the longevity of points in $Cell(Q_i)$. For example we may take:

$$L^*(Q_i) = \min_{(q,t) \in Cell(Q_i)} t. \qquad (17)$$

$\tau_k^n(Q_i)$ gives the longevity of Q_i using paths consisting of n or fewer edges. We may define $L_k^{n+1}(Q_i)$ recursively as:

$$\begin{aligned}\tau_k^{n+1}(Q_i) \;=\; & \tau_k^n(Q_i) \cup \{(a_1,b_1)\cap(a_2,b_2), \max(l_1,l_2) \mid \\ & (a_1,b_1,l_1) \in \tau_k^n(Q_i), \\ & (a_2,b_2,l_2) \in \tau_k^n(Q_j), \\ & e_{i,j} \in E\}. \end{aligned} \tag{18}$$

We then compute $\tau_k^m(Q_i)$ for some minimum value $m < N$ for which $\tau_k^m(Q_i)$ converges, (i.e. $\forall Q_i, \tau_k^m(Q_i) = \tau_k^{m+1}(Q_i)$), where N is the total number of cells. The longevity of a cell Q_i given a fault f_j in actuator k at position r is:

$$L(Q_i,k,r) = \left\{ \begin{array}{c} \max\limits_{(r_1,r_2,l)\in\tau_k^m(Q_i), r_1<r<r_2} \quad l \\ \perp \quad \text{If } f_j \text{ not tolerable at } Q_i \end{array} \right\}. \tag{19}$$

For efficiency, when implementing Eq. 18 we would perform the set union, but would also suppress intervals for which the longevity was not a maximum.

We have implemented the dynamic programming algorithm and tested it on a 6-DOF platonic-beast problem in which $\hat{q} \in \mathbb{R}^3 \times SO(3)$. The longevity measures and corresponding optimal recovery actions were computed for each of the $50,249$ valid cells for each of the 12 types of failures. The computation required approximately one hour of CPU time running on a 200 MHz Silicon Graphics Indigo. The time requirements are not prohibitive since they need only be computed once for each task specification.

7 Conclusions & Future Work

We have developed a performance measure, called longevity, which quantifies the potential for fault tolerant operation of a robot given a set of potential faults. This measure can be applied to any task that is described as a set of constraints on the configuration over time.

This measure has been applied to produce optimal trajectories for a four legged robot. The trajectories were compared to a straight-line motion implementation for the same task, with respect to the worst and average case failure modes of points along the respective trajectories. The longevity paths were shown to obtain much better progress towards the goal for both average and worst case failures along the path.

A dynamic programming algorithm for the computation of the longevities for a set of points has been given. As a byproduct of the computation, a complete contingency plan can be generated which gives the optimal recovery action for each point along the path.

In future work, we plan to develop stochastic models fault process. This would allow the construction of expected performance along a path, which would be more relevant to planning of real-world tasks, as well as being easier to interpret.

References

[1] T. Anderson and P. A. Lee. Fault tolerance, principles and practice. Prentice Hall, Englewood Cliffs, NJ., 1981.

[2] R. Boudarel, J. Delmas, and P. Guichet. Dynamic Programming and its Application to Optimal Control. Academic Press, New York, 1971.

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

[4] 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.

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

[6] 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.

[7] 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.

[8] D Meagher. Geometric modeling using octree encoding. Computer Graphics and Image Processing, 19(2):129–147, June 1982.

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

[10] Dinesh K. Pai, Roderick A. Barman, and Scott K. Ralph. Platonic beasts: Spherically symmetric multilimbed robots. Autonomous Robots, 3(2):191–202, 1995.

[11] 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.

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

[13] 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.

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