

Real-Time Collision Avoidance for Dexterous 7-DOF Arms

Homayoun Seraji, Bruce Bon
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

Abstract

*A new approach to real-time collision avoidance for dexterous 7-DOF arms and supportive simulation and **experimental** results are presented. The collision avoidance problem **is** formulated and solved **as** a force control problem. Virtual forces opposing intrusion of the arm into the obstacle safety zone are computed **in real time**. These forces are then nullified **by** employing **an** outer feedback loop which perturbs the arm Cartesian commands for the inner position control system. Graphical simulation results are presented to demonstrate the application of the collision avoidance approach to the dexterous 7-DOF arms of the NASA-Ranger **Telerobotic Flight Experiment**. The approach **is** also implemented and tested on a 7-DOF RRC arm and a set of experiments are conducted **in** the laboratory. These experiments demonstrate perturbations of the **end-effector** position and orientation, as well as the arm posture, **in** order to avoid impending collisions. The proposed approach is simple, **computationally** fast, requires minimal modification to the arm control system, and applies to whole-arm collision avoidance.*

1 Introduction

The need for human-equivalent manipulative capabilities has motivated the development of *dexterous robotic arms* over the past decade. These robotic arms are cinematically similar to the human arm and have 7 joints (instead of the conventional 6 joints), which makes them cinematically redundant. This redundancy is the basis for the arm dexterity, and implies that there are infinite distinct arm postures which yield the same end-effector position and orientation. Since 1985, the Robotics Research Corporation (RRC) has been manufacturing a family of commercially available 7-DOF arms. Similarly, robots planned for space operations,

including the NASA-Ranger Telerobot and the Space Station Dexterous Robotic System, have 7-DOF arms.

While motion control of dexterous 7-DOF arms in an obstacle-free workspace has been the subject of considerable research in recent years, the more realistic problem of collision-free motion in an obstacle field has not been investigated extensively. Maciejewski and Klein[1] describe a method for collision avoidance using the Jacobian pseudoinverse approach for redundant arm control. Khatib[2] suggests a method for real-time collision avoidance in operational space using the gradients of artificial potential fields. Wikman and Newman[3] describe a reflex control approach for on-line collision avoidance. Boddy and Taylor[4] develop a whole-arm reactive collision avoidance scheme using the configuration control methodology. Glass et al[5] describe a real-time configuration controller for a 7-DOF arm utilizing collision avoidance as the additional task. Finally, Seraji, Steele, and Ivlev[6] develop a method for sensor-based collision avoidance for a position-controlled dexterous arm based on perturbations of the end-effector position coordinates.

In this paper, we present a new methodology and a set of supportive simulation and experimental results on collision avoidance of dexterous 7-DOF arms. This methodology applies to *whole-arm* collision avoidance by perturbing both the end-effector position and orientation as well as the arm posture. The underlying concept is to represent intrusion of the arm into the obstacle safety zone by a *virtual force*, and nullifying this force by perturbing the nominal arm motion trajectory. This approach is simple and **computationally** efficient, suitable for real-time implementation, and requires minimal modification to the arm control system.

The paper is organized as follows. Section 2 describes the collision avoidance strategy. The model-based collision detection method is presented in Section 3. Section 4 describes a set of graphical simulations of the Ranger dexterous arms demonstrating the collision avoidance capability. Six experimental case studies highlighting different types of collision avoidance using RRC arms are discussed in Section 5 and experimental results are presented. Section 6 presents conclusions drawn from this work.

2 Collision Avoidance Strategy

Robotic arms are basically positioning devices which can carry a payload from an initial position and orientation to a target destination along a prescribed Cartesian trajectory. This arm motion is accomplished by mapping the desired Cartesian path into joint angle trajectories which are then tracked using joint servo control loops. For a cinematically redundant 7-DOF arm, such as the RRC arm, we assume that the end-effector position and orientation and the arm angle¹ (which defines the arm posture) are under the control of a configuration control scheme, as described in [7,8]. In this approach, we use the configuration-controlled

¹The arm angle is defined to be the angle between the arm plane passing through the upper-arm and forearm and a reference plane passing through the shoulder-wrist axis.

arm as the baseline system and make the necessary enhancements to this system to provide the collision avoidance capability.

2.1 Arm Segmentation

For the development of the collision avoidance **strategy**, it is convenient to segment the 7-DOF arm into three links or arm segments as shown in Figure 1a for the RRC arm and in Figure 1b for the Ranger arms. These segments are the *tool-link* TW , the *forearm link* WE , and the *upper-arm link* ES , where T , W , E , and S refer to the tool-tip, wrist, elbow, and shoulder, respectively. Three classes of obstacles are now defined as illustrated in Figures 1a-1b: *tool-tip obstacle*, *wrist obstacle*, and *elbow obstacle*. A tool-tip obstacle is one whose nearest point on the arm is on the tool-link closer to the tool-tip than a user-specified distance D . A wrist obstacle is one whose nearest point on the arm is on the tool-link further away from the tool-tip than D . An elbow obstacle is the one whose nearest point on the arm is located either on the upper-arm or on the forearm. Notice that an extended obstacle can be described as a combination of a tool-tip obstacle, a wrist obstacle, and an elbow obstacle. In our control strategy, collision with a tool-tip obstacle is avoided by perturbing the three end-effector position coordinates. Collision avoidance with a wrist obstacle is achieved with perturbations of the three end-effector orientation coordinates. Finally, an elbow obstacle is avoided by perturbing the arm angle, i.e. rotating the elbow E about the shoulder-wrist axis SW without disturbing the tool frame (i.e., arm self-motion). Notice that the obstacle detection software provides data on the single nearest obstacle in each of the three zones; thus limiting perturbation computations to no more than three obstacles during any iteration. This separation of influence of the obstacles is adopted to avoid unnecessary trajectory perturbations to both the end-effector position and orientation, as well as to the arm angle.

2.2 Virtual Spring and Damper Forces

For every reachable object in the workspace, the user defines a *safety zone*, which is displaced from the object surface by a user-specified *stand-off distance* d_r . Inside the safety zone, there are *fictional* springs with natural length d_r and user-defined stiffness k_i , and dampers with user-specified damping coefficient k_p occupying the space between the object surface and the safety zone boundary; a typical example is shown in Figure 2. The proximity of the arm to each object, d_m , is computed continuously by the obstacle detection software described in Section 3 or measured by arm-mounted proximity sensors[6]. When any point on the arm enters the safety zone of an object as determined by the detection system ($d_m < d_r$), a *virtual intrusion force* is generated in the control software and is exerted on the arm at the intrusion point, see Figure 2. The magnitude of this force is related directly to the extent and rate of the intrusion into the safety zone, and the direction is opposing the intrusion. The intrusion force is computed from

$$F = k_i e + k_p \frac{de}{dt} \quad (1)$$

where $e = d_r - d_m$ denotes the extent of intrusion in the Cartesian x direction (for instance), i.e. the compression of the spring-plus-damper. In equation (1), the term $k_i e$ represents the compressive force due to the spring, while the term $k_p \frac{de}{dt}$ is the resistive force due to the damper. Note that the virtual intrusion force is always along the line of shortest length PQ connecting the closest points on the obstacle P and on the arm Q , where $d_m = |PQ|$.

2.3 End-Effector Position Perturbations

When the end-effector is approaching an obstacle, the virtual intrusion force is applied at or near the tool-tip. End-effector collision avoidance is accomplished by automatically modifying the operator-commanded end-effector position trajectory so as to nullify the intrusion force.

The three-dimensional end-effector virtual intrusion force vector is first decomposed into three components along the x , y , and z axes of a fixed world frame-of-reference attached to the arm base. We shall now describe the collision avoidance system along the x -axis in detail; avoidance along y and z axes are accomplished in a similar manner. Figure 3 shows the block diagram of the end-effector collision avoidance system along the x -axis. The goal of the collision avoidance system is to perturb the nominal operator-commanded motion trajectory x_r in order to nullify the intrusion force F . This force is driven to zero by “pushing” the arm out of the safety zone. This goal is accomplished by employing an external force control loop around the internal position control system as shown in Figure 3. The virtual force, F representing the intrusion along the x -axis is compared with the force setpoint $F_r = 0$. When $e > 0$ indicating intrusion, the virtual force F is driven to zero by an integral controller with gain k which produces the appropriate trajectory perturbation x_f that modifies the nominal trajectory x_r . This perturbation is given by

$$\begin{aligned} x_f &= k \int [F - F_r] dt \\ &= k \int \left[k_i e + k_p \frac{de}{dt} \right] dt \\ &= k k_p e + k k_i \int e dt \end{aligned} \quad (2)$$

The spring and damper produced perturbation components are, respectively

$$x_{fs} = k k_i \int e dt \quad ; \quad x_{fd} = k k_p e$$

Equation (2) indicates that the trajectory perturbation x_f is generated by a **proportional-plus-integral (PI)** controller acting on the intrusion amount e . Observe that when the arm does

not intrude into any safety zone ($e \leq 0$), no corrective action is necessary ($F = 0$, $x_f = 0$), and the nominal trajectory x_r is executed without perturbation.

Now, if a constant integral gain k is used, the position perturbation x_f can cause instability problems at the boundary of the safety zone. This is caused by the abrupt zeroing of the perturbation when the arm exits the safety zone, resulting in discontinuity in both velocity and position. To avoid this instability, a nonlinear gain k is introduced to “smooth out” this transition as follows:

$$k = \begin{cases} 0 & \text{if } e \leq 0 \\ e/d_k & \text{if } 0 < e < d_k \\ 1 & \text{if } e \geq d_k \end{cases} \quad (\text{i.e., arm outside safety zone}) \quad (3)$$

where d_k is the value of e at which the full value of the perturbation is applied. Multiplying the integrator output by the nonlinear gain k ensures that the perturbation x_f will not change to zero abruptly, and hence prevents a discontinuity in commanded position that would otherwise occur when the arm exits the safety zone. The variation of k versus e is depicted in Figure 4.

2.4 End-Effector Orientation Perturbations

When the wrist is approaching an obstacle, the virtual intrusion force F is applied at or near the wrist center. Collision avoidance is accomplished by modifying the end-effector orientation trajectory so as to nullify F .

Figure 5 shows one viable approach to perturbing orientation of the tool-link TW, where T is the tool-tip and W is the wrist center. Let P be the closest point on an obstacle to TW and Q be the closest point on TW to the obstacle. The objective is to leave the position of T unperturbed, but to rotate TW about T such that the point Q will move away from the obstacle to zero out the virtual force \vec{F} . Following the derivation in Section 2.3, the desired displacement of point Q is \vec{Q}_f :

$$\vec{Q}_f = k[k_p \vec{e} + k_i / \vec{e} dt] \quad (4)$$

where k is defined by equation (3) and \vec{e} is the intrusion vector into the safety zone.

Let $\vec{r} = \vec{Q} - \vec{T}$ be the vector from T to Q, where \vec{Q} and \vec{T} are the position vectors of points Q and T, respectively. Then let \hat{r} be the unit vector of \vec{r} and r be the length of \vec{r} . If the geometry is known to be planar with \vec{F} perpendicular to TW (as shown), then the desired orientation perturbation will be a rotation about T by the angle α , where:

$$\alpha = |\vec{Q}_f| / r \quad (5)$$

In order to express this perturbation as a rotation about an axis through T, we define the angular rotation perturbation vector \vec{R}_f to be a vector along the desired axis of rotation whose magnitude is the desired angle of rotation. Then \vec{R}_f is given by the cross-product:

$$\vec{R}_f = \hat{r} \times (\vec{Q}_f / r) \quad (6)$$

or

$$\vec{R}_f = \frac{r}{r^2} \times \vec{Q}_f \quad (7)$$

Substituting equation (4) into equation (7), we obtain:

$$\vec{R}_f = \frac{k\vec{r}}{r^2} \times [k_p \vec{e} + k_i \int \vec{e} dt] \quad (8)$$

The vector \vec{R}_f represents the necessary rotation of the tool-link about T to accomplish the collision avoidance; i.e., rotate TW about the vector \vec{R}_f perpendicular to the (F,@) plane by the angle $|\vec{R}_f| = \alpha$. Now, to find the corresponding change in the end-effector orientation, we map the 3x1 rotation vector \vec{R}_f to a change ΔR in the 3x3 end-effector rotation matrix using the equivalent angle-axis representation [9]. The perturbed end-effector orientation R_c is found as

$$R_c = R_r \cdot \Delta R \quad (9)$$

where R_r is the nominal end-effector rotation matrix.

2.5 Arm Angle Perturbations

Figure 6 illustrates the basic geometry involved, where the points S, E and W refer to the shoulder, elbow and wrist centers, respectively, and \hat{v} is a user-defined vector (often the vertical vector through S) which, together with the line SW, defines the reference plane. The arm angle ϕ is measured from this reference plane to the arm plane containing S, E and W. Potential collision with the upper-arm and forearm links are avoided by perturbing the nominal arm angle, namely, by rotating the elbow E about the SW axis (i.e., arm self-motion) without affecting the end-effector position and orientation.

Let the point P represent the nearest point on the surface of an obstacle to the surface of the upper-arm link SE or the forearm link EW. Let the point Q be the point on SE or EW that is closest to P. The error \vec{e} is the vector whose direction is from P to Q and whose magnitude is the amount of intrusion of the point P into the safety zone, where this zone is defined by the arm-link radius d_t and the stand-off distance d_r . Then:

$$|\vec{e}| = (d_t + d_r) - IQ - PI \quad (10)$$

and \vec{e} is in the direction of (Q - P), i.e. from P to Q.²

²For ease of visualization, the safety zone in Figure 6 is shown surrounding the arm link rather than the obstacle – this is an equivalent model to the usual one, where the safety zone surrounds each obstacle.

The signed magnitude of \vec{e} , denoted by e_m , is the basis for computing the arm angle perturbation. The sign of e_m is chosen to perturb Q away from the obstacle. The scalar virtual force for collision avoidance is given by

$$F_f = k_i e_m + k_p \frac{de_m}{dt} \quad (11)$$

where the signs are positive to indicate a force in the direction opposing the intrusion. The virtual force is always applied in the direction perpendicular to the SEW plane.

Let r be the distance of the point Q to the line SW, i.e. $r = QH$ where H is the projection of Q on SW. Then, r will be the radius of revolution of the point Q around SW to perturb the arm angle. We need to find the signed scalar angular perturbation ϕ_f to the nominal arm angle ϕ_r which will nullify the safety zone intrusion force. To cause a displacement AQ in the position of the point Q, the arm angle must change by the amount

$$\phi_f = \frac{1}{r} \Delta Q \quad (12)$$

Since $AQ = k k_p e_m + k k_i \int e_m dt$, we obtain

$$\phi_f = \frac{k k_p}{r} e_m + \frac{k k_i}{r} \int e_m dt \quad (13)$$

where k is defined by equation (3) and serves, as discussed in Section 2.2, to prevent discontinuities as the arm moves out of the safety zone.

2.6 Whole-Arm Collision Avoidance

The results of Sections 2.3-2.5 on end-effector position and orientation perturbations and arm angle perturbation are now combined to obtain a *whole-arm* collision avoidance system. Figure 7 depicts the block diagram of a 7-DOF arm with a configuration controller in the inner loop and a collision avoidance controller in the outer loop. In this Figure, K , K_p , and K_i are 7×7 diagonal matrices, where the first six elements of each matrix are related to the end-effector position and orientation and the seventh element is related to the arm angle. This control system ensures that the end-effector coordinates and the arm posture respond in real time to avoid impending collisions.

2.7 Stability Analysis of Collision Avoidance System

Consider the collision avoidance system shown in Figure 8 with the nonlinear gain k defined by equation (3). Because of the nonlinear nature of k , the stability analysis of the collision avoidance system is non-trivial. This subject is studied in this section.

For a robotic arm with a position controller, the motion of the arm in each Cartesian direction (such as x) can be adequately modeled by a second-order transfer-function as [see, e.g., 10]

$$G(s) = \frac{x(s)}{x_r(s)} = \frac{\omega^2}{s^2 + 2\xi\omega s + \omega^2} = \frac{b}{s^2 + as + b} \quad (14)$$

where ξ and ω denote, respectively, the damping ratio and natural frequency of the inner robot system (arm-plus-controller), $a = 2\xi\omega$ and $b = \omega^2$. During collision avoidance, the outer force control loop perturbs the Cartesian setpoint for the inner position control loop. In the proposed collision avoidance scheme, the outer loop employs the PI controller

$$K(s) = k_p + \frac{k_i}{s} \quad (15)$$

in cascade with a nonlinear gain k , where k_p and k_i are the constant positive Proportional and integral gains, respectively.

To investigate the absolute stability of the closed-loop collision avoidance system, we combine the linear components (14) and (15) as

$$w(s) = G(s)K(s) = s \left(\frac{b(k_p s + k_i)}{s^2 + as + b} \right) \quad (16)$$

which is a third-order transfer-function, and separate out the nonlinear element which is the gain k . We can now apply the Popov Stability Criterion [11] to the system by examining the Popov plot of $W(j\omega)$, which is the plot of $\mathcal{Re}W(j\omega)$ versus $\omega\mathcal{Im}W(j\omega)$, with ω as a parameter and \mathcal{Re} and \mathcal{Im} refer to the real and imaginary parts, respectively. This plot reveals the range of values that the nonlinear gain k can assume while retaining closed-loop stability. The Popov Criterion states that:

“A sufficient condition for the closed-loop system to be absolutely stable for all nonlinear gains in the range $(0, k_{max})$ is that the Popov plot of $W(j\omega)$ lies entirely to the right of a straight-line passing through the point $-\frac{1}{k_{max}} + j0$.”

In order to apply the Popov Criterion to the collision avoidance system, we need to compute the crossing of the Popov plot of $W(j\omega)$ with the real axis. In this case, from equation (16), we obtain

$$\mathcal{Re}W(j\omega) = a^2\omega^2 + \frac{-b}{(b - \omega^2)^2} [\omega^2 k_p - (ak_i + bk_p)] \quad (17)$$

$$\omega\mathcal{Im}W(j\omega) = \frac{-b}{a^2\omega^2 + (b - \omega^2)^2} [\omega^2(ak_p - k_i) + bk_i] \quad (18)$$

Two distinct cases are now possible depending on the relative values of k_i and k_p .

2.7.1 Case One: $k_i \leq ak_p$

In this case, $\omega\mathcal{Im}W(j\omega)$ is *always* negative for all ω , that is, the Popov plot of $W(j\omega)$ remains entirely in the third and fourth quadrants and does *not* cross the real axis. This implies that we can construct a straight-line passing through the origin such that the Popov

plot is entirely to the right of this line. Therefore, according to the Popov Criterion, the range of the allowable nonlinear gain k is $(0, \infty)$.

2.7.2 Case Two: $k_i > ak_p$

In this case, the Popov plot of $W(j\omega)$ crosses the real axis. The crossover frequency ω_o is found by solving $\omega \text{Im}W(j\omega) = 0$ to yield

$$\omega_o^2 = \frac{bk_i}{k_i - ak_p} \quad (19)$$

The value of $W(j\omega)$ at the crossover is then obtained as

$$\text{Re}W(j\omega_o) = \frac{ak_p - k_i}{a} \quad (20)$$

Therefore, the maximum allowable gain is

$$k_{max} = \frac{1}{\text{Re}W(j\omega_o)} = k_i - \frac{a}{ak_p} \quad (21)$$

We can now construct a straight-line passing through the point $-0.2 + j0$ such that the Popov plot of $W(j\omega)$ is entirely to the right of this line. Thus the range of the allowable nonlinear gain k is $(0, k_{max})$.

Observe that the distinction between the above two cases is on the *relative* values of the proportional and integral gains k_p and k_i in the PI controller. Notice that a reasonable estimate of the attenuation factor a can readily be obtained experimentally from the open-loop response of the Cartesian coordinate x to the step reference command x_r . Specifically, the step response has the settling time of $t_s = \frac{5}{\xi\omega} = \frac{10}{a}$ to reach within the $\pm 1\%$ tolerance band of the final value.

For the sake of illustration, computer simulations of a position-controlled arm with a nonlinear PI collision avoidance controller are obtained. Given $G(s) = \frac{25}{s^2 + 10s + 25}$ and $K(s) = k_p + \frac{2}{s}$, the Popov plots of $W(s) = G(s)K(s)$ for the two values of $k_p = 2$ and $k_p = 0$ are shown in Figures 9a–9b. For $k_p = 2$, it is seen from Figure 9a that the Popov plot of $W(j\omega)$ does not cross the real axis as expected; hence the allowable range of the nonlinear gain k is $(0, \infty)$. In contrast, when k_p is reduced to zero, Figure 9b reveals that the Popov plot of $W(j\omega)$ crosses the real axis at -0.2 , hence the allowable range of k is now reduced to $(0, 5)$. We conclude that reducing k_p has a destabilizing effect and decreases the range of the allowable nonlinear gain k to maintain closed-loop stability.

3 Model-Based Obstacle Detection

Obstacle detection can either be *sensor-based*, utilizing proximity sensors or machine vision to identify obstacles, or *model-based*, utilizing geometric computations on a database that

includes locations and geometries for manipulator arms and all potential obstacles. Model-based detection has usually been used off-line as a component of path planning and simulation systems and, as such, has not had the requirement for real-time performance. These detection methods are capable of modeling complex manipulators and obstacles and can exhaustively search for nearest obstacles (see, e.g., 12-13). Oftentimes, designs for real-time collision avoidance have used sensor-based obstacle detection (see, e.g., 6, 14-15), but robotic manipulators planned for space operations are not equipped with sensors capable of detecting obstacles in real time. Usage of on-line detection data, however, will be similar to usage of real-time sensor data; the model-based detection software may be considered to be a *software sensor suite*, as opposed to hardware sensors, instrumenting the entire workspace and manipulators. The goal of our obstacle detection effort, therefore, is to provide a model-based obstacle detection capability which can detect nearest obstacles in real time. This has necessitated a minimalist approach, with simple object models and distance computation algorithms, together with some database sophistication to eliminate unnecessary computations.

For the sake of illustration and ease of presentation, we shall consider the NASA-Ranger Telerobotic Flight Experiment to exemplify the approach. Figure 10 shows the major components of the Ranger flight vehicle, consisting of a *propulsion module* with an octagonal cross-section, two essentially planar *solar arrays*, a tapered *electronics module* with a square cross-section, and a 'cubicle *manipulator module*. Four robotic manipulators will be attached to the manipulator module: two 7-DOF *dexterous manipulators*, attached to the left and right sides of the module; a *camera manipulator* attached to the top of the module; and a *grapple manipulator* attached to the bottom. Figure 1b shows the top view of the current design of the Ranger dexterous manipulators that are mounted on the manipulator module.

In order to detect impending collisions, the fundamental requirement is to find the shortest distance between all parts of the active manipulator and all potential obstacles, including other manipulators (which may have moved since the previous distance computation) and other components of the Ranger flight vehicle. All the components represented in Figures 1b and 10 can be modeled rather simply with cylindrical or polyhedral shapes. Furthermore, the Ranger vehicle and manipulators need not be modeled with high fidelity, but computational speed is critical because obstacle detection must be done at a high rate on-board the Ranger to meet safety requirements. In order to meet the real-time Ranger performance requirements, we have developed an approach which emphasizes simple object models, direct geometric computation of distances, and avoidance of any unnecessary distance computations.

The collision detection software assumes that only one manipulator is moved at a time. For each manipulator link that has moved, the shortest distance from the link to all objects in its *obstacle list* (see description below) is measured. The collision detection function returns the nearest object, its distance, and the link and collision object nearest points for three categories: manipulator-to-manipulator, manipulator-to-vehicle, and **manipulator-to-bounding box**. The *bounding box* is defined to be a hollow virtual rectangular box enclosing the arm and centered on the Ranger manipulator module. The three dimensions of the box in x , y , and z directions are specified by the user. When a manipulator approaches any side

of the bounding box, an obstacle is detected.

The collision detection database (CDDDB) contains geometrical data, about the Ranger flight vehicle and its manipulators for use by the collision detection software. The potential obstacles in the CDDDB are *faces*, *edges* and manipulator *links*. The CDDDB identifies relationships between objects and contains lists of point locations, which may be either link or edge end-points, as well as a list of pointers to potential obstacles for each link (the *obstacle M*). The obstacle list³ typically has a relatively small number of entries, because most of the objects in the CDDDB are not within the reachable space of the link and are thus not candidates for collision. By only checking the cases described above, and by only checking feasible obstacles, collision detection computation is minimized.

All coordinates in the CDDDB are expressed in the fixed *manipulator reference frame*, a right-handed frame-of-reference whose origin is at the center of the manipulator module. For each link of each arm, the CDDDB contains the proximal and distal end-points of a line segment through the axis of the link, and the radius for the link. Because there are only 5 link end-points per dexterous arm, a list of link end-point locations is maintained. These locations are updated due to changes in joint angles, using forward kinematics computations.

For the vehicle, the CDDDB contains a list of vehicle vertex points, whose coordinate values are constant in the manipulator reference frame. Each edge contains pointers to two of the vertex points. Each face contains pointers to vertex points surrounding the face in a counter-clockwise fashion. These data structures also contain additional, redundant data computed at initialization time to make distance computations fast.

For link-to-link distances, the shortest distance between the link axis line segments is computed and the radii of both links are subtracted from this distance in order to get the shortest distance between link surfaces. The geometric link surface that this computation models is a cylinder of uniform radius with hemispherical end-caps of the same radius.

For link-to-edge distances, the shortest distance between the link axis line segment and the edge line segment is computed and the link radius is subtracted in order to get the shortest link-to-edge distance. The nearest points are the points on the link axis line segment and the edge line segment that are closest to each other.

For link-to-face distances, the distance between the distal end-point of the link and the plane of the face is computed, as well as the projection of the end-point in the plane. If the projection of the distal end-point is not within the face, then this end-point-to-face computation is discarded. Otherwise, the projection point is the face nearest point, the link distal end-point is the link nearest point, and the link-to-face distance is the distance between the nearest points less the radius of the link.

For link-to-bounding box distances, the distance between both end-points of each link and the maximum and minimum values in each of the three reference frame axis directions

³The obstacle list structure is designed to allow dynamic addition and deletion of pointers to objects as well as dynamic reordering of each list to reflect priority based on changing obstacle distances. In the current implementation, the obstacle lists are static and are manually constructed based on simple inspection of the Ranger geometry.

is computed with a simple subtraction per end-point to bounding box limit pair. The least of these distances is the minimum bounding box distance. The link nearest point is simply the end-point used for the minimum distance, and the bounding box nearest point has the same coordinate value as the bounding box limit in the axis corresponding to the bounding box limit (e.g. if the closest limit is in the +x direction, the x coordinate will have the +x limit value) and the same values as the link nearest point for the other two coordinates.

3.1 Line Segment to Line Segment Distance Computation

In order to determine the shortest distance between two arm links or between an arm link and the edge of a polygonal face, it is necessary to compute the shortest distance between one line segment (the arm link axis) and another line segment (another axis or the edge) in the three-dimensional space. Figure 11 illustrates the problem. Line segment 1 is defined by end-points P_1 and P'_1 , where:

$$\begin{aligned} P_1 &\equiv (x_1, y_1, z_1) \\ P'_1 &\equiv (x'_1, y'_1, z'_1) \\ l_1 &\equiv \text{length of line segment 1} = |P'_1 - P_1| \\ \hat{A}_1 &\equiv \text{direction cosines of line 1} = (a, b, c) \end{aligned}$$

and the coordinates are in the fixed manipulator frame-of-reference. The parametric equation for the infinite line through line segment 1 is:

$$\mathbf{X} = P_1 + t_1 \hat{A}_1 \quad (22)$$

where the 3x1 vector \mathbf{X} represents the coordinates of any point along the line and t_1 is a scalar parameter. Line segment 2 has parameters and an equation analogous to that for line segment 1; namely

$$\mathbf{X} = P_2 + t_2 \hat{A}_2 \quad (23)$$

We now wish to find the shortest distance d_m between line 1 and line 2. We also want to find the points M_1 on line 1 and M_2 on line 2 corresponding to this shortest distance.

Because the vector $(M_2 - M_1)$ must be perpendicular to both \hat{A}_1 and \hat{A}_2 :

$$(M_2 - M_1) \cdot \hat{A}_1 = 0 \quad (24)$$

$$(M_2 - M_1) \cdot \hat{A}_2 = 0 \quad (25)$$

Thus

$$M_2 - M_1 = \vec{D}_{12} + t_2 \hat{A}_2 - t_1 \hat{A}_1 \quad (26)$$

where $\vec{D}_{12} \equiv P_2 - P_1$. Substituting this result into equations (24) and (25) and solving for t_1 and t_2 yields:

$$t_1 = a + t_2 b \quad (27)$$

$$t_2 = \frac{a h - c}{1 - b^2} \quad (28)$$

assuming $b \neq \pm 1$, where $a \equiv \vec{D}_{12} \cdot \hat{A}_1$, $b \equiv A_1 \cdot \hat{A}_2$, and $c \equiv \vec{D}_{12} \cdot A_2$. The shortest distance between the lines, d_m , is simply the distance between the points M_1 and M_2 corresponding to the parameters t_1 and t_2 given by equations (27) and (28). This algorithm will give correct results for all non-parallel lines, including intersecting lines. If $b = \pm 1$, then the lines are parallel or **colinear**. This special case is handled in a similar fashion with straightforward geometric computations [16].

Whether or not the line segments are parallel, the nearest points and shortest distances for infinite lines may not correspond to the correct results for finite-length line segments. In order to find the correct nearest points and shortest distances for non-parallel line segments, the infinite-line nearest points are first tested to find out whether or not they are both within their respective line segments – if so, the infinite-line results are correct. If only one of the infinite-line nearest points is within its line segment, then the corrected nearest point on the second line segment will be the end-point which is nearest to the infinite-line nearest point on the first line segment. Then the corrected nearest point on the first line segment will be the point on the first line segment which is closest to the corrected nearest point on the second line segment. If neither of the infinite-line nearest points is within its line segment, then a two-stage correction is necessary. In the first stage, each line segment nearest point is taken to be the end-point which is closest to the infinite-line nearest point for that line segment. One of these first stage nearest points is guaranteed to be the correct nearest point, but the other is not. In the second stage correction, for each first stage nearest point, the point-to-line-segment distance and nearest point on the opposing line segment are calculated. Then the pair of nearest points with the smaller distance is selected as the final, correct set.

3.2 Point to Polygonal Face Distance Computation

This section presents the mathematical details of the algorithm used for computing the shortest distance between a point and a polygonal face in the three-dimensional space. The algorithm presented here is valid under the following two assumptions: (i) all faces are planar, convex polygons, and (ii) all objects are convex polyhedra, i.e. with no concave angles between faces.

Given a sequence of vertex locations ordered in a counter-clockwise direction around a face and an arbitrary point in space, we wish to find the projection of the point into the plane of the face, determine whether or not the projection of the point lies within the face, and find the distance from the point to the plane of the face.

Figures 12a–12b illustrate the problem. A face is represented by a sequence of vertex points ordered in a counter-clockwise direction around the face **relative to** a point of view which is outside of the object of which the face is a part. Point P is an arbitrary point (for Ranger, the distal end-point of a manipulator link) which may be near to collision with the face. We need to determine whether or not Q, the projection of P, falls within the face, since P is not considered a collision hazard if it does not. If Q does fall within the face, we also need the location of Q and the distance from P to the face.

The set of unit direction vectors, \hat{A}_i , as shown in Figure 12a, is computed from the vertex locations, and the face normal \hat{n} is computed as the average of all vertex face normals, $\hat{n}_i = \hat{A}_i \times \hat{A}_{(i+1) \bmod n}$. The face position vector \vec{p}_f is computed as the average of the vertex locations. The plane of the face is then defined by the face normal and the face position vector.

The distance d_{of} from the origin of the coordinate frame to the face plane is the dot-product of the face plane perpendicular \hat{n} with the face position vector \vec{p}_f (see Figure 12b):

$$d_{of} = \hat{n} \cdot \vec{p}_f \quad (29)$$

The shortest distance d_j from the plane to an arbitrary point P is the same as the distance between the plane and a parallel plane through the point P. Therefore, the shortest distance is given by

$$d_f = \hat{n} \cdot \vec{p}_p - d_{of} \quad (30)$$

Since the perpendicular to the face plane is computed from edge direction vectors that are directed clockwise around the face as viewed from outside of the solid, the perpendicular is directed outward from the face. Then the signed value d_j computed for the distance between the point P and the face plane is greater than zero for a point outside of the solid (“above” the face plane) and less than zero for a point that may be inside the solid.

The intersection point Q of a perpendicular from an arbitrary point P to the plane is the location vector of the point less the perpendicular unit vector \hat{n} times the shortest distance to the plane:

$$Q \equiv \vec{p}_q = \vec{p}_p - d_f \hat{n} \quad (31)$$

Finally, we need to find out whether or not the point Q lies within the **face**. Remember that \hat{A}_i is the direction vector from vertex i to vertex $i + 1$ of the face. If \vec{B}_i is a vector from vertex i to Q, then the intersection point is interior to the face if and only if

$$\hat{n} \cdot (\hat{A}_i \times \vec{B}_i) > 0 \quad (32)$$

for $i = 0, \dots, (n-1)$.

3.3 Computation Times

Line segment-to-line segment distance computation time, averaged over hundreds of different line segments in different configurations, is measured to be about 16 μsec on a MIPS R4600PC

processor running at 100 MHz. Point-to-face distance computation time is even faster, but has not been measured.

Total obstacle detection computation time, including computation of distances between every link of the active arm and every obstacle in the link's obstacle list, is measured to be about 1.23 msec on the same MIPS processor. The Ranger flight computer is expected to be a MIPS R4600 processor, but running at a slower clock rate, resulting in an estimated computation time for obstacle detection of about 2.5 msec, which is well within the allowable range for real-time arm control computations.

4 Ranger Simulation Results

The NASA-Ranger Telerobotic Flight Experiment[17-18], led by the University of Maryland, is aimed at the development and demonstration of robotics technologies for executing manipulation tasks in space. Ranger incorporates two 7-DOF dexterous manipulation arms mounted on a free-flying base, in addition to grapple and camera arms. The two dexterous arms will be used, both individually and cooperatively, to perform a variety of manipulation experiments and servicing operations in space, as shown in Figure 13. The Ranger dexterous arms will be controlled using the configuration control approach in which the arm angle is controlled in addition to end-effector position and orientation.

The software packages for obstacle detection and collision avoidance are implemented in C on an SGI Indy workstation under IRIX, but are designed to be portable for integration into the Ranger flight software running on a MIPS R4600 processor under VxWorks. A graphical user interface (GUI) program is developed to drive a 3-D graphical simulation of the Ranger provided by the University of Maryland. The GUI is implemented in an object-oriented interpretive language called Python [19-20], controlling widgets provided by Tk [21].

Figure 14 is a screen snapshot of the GUI, showing the Ranger 3-D graphics animation in the upper left, the main control panel along the bottom of the screen, the test and viewpoint control panel just above the main control panel, and the arm control panel in the upper right. The 3-D graphics displays the Ranger Neutral Buoyancy Vehicle (NBV) and the two dexterous arms. The nearest obstacle to the currently active arm is identified by a colored line connecting the obstacle and the arm link that is closest to it. If the obstacle is within the stand-off distance specified by the user, the nearest arm link changes color to red and the connecting line changes from yellow to red. When collision avoidance is not active, commanding an arm into a collision with an object will result in the arm moving *inside* the obstacle. When collision avoidance is active, the arm will move as commanded until it hits the invisible safety zone of the object, and will then slide along the safety zone boundary until it is as close as possible to the target state.

The GUI provides the main control panel for operation of the Ranger graphical simulation environment. The left-most column of the main control panel has buttons to select joint or Cartesian arm control, to turn obstacle detection and collision avoidance on or off, to bring

up a button panel for selecting test cases and viewpoints, and to terminate execution. The second column has buttons which are used to select the active arm for control. The third column displays obstacle detection data, including the arm link and obstacle nearest points and the distance between them, and allows specification of the detection threshold.⁴The background of the distance panel is green for no obstacle within the detection threshold distance, yellow for an obstacle within the distance, and red if the arm has collided with the obstacle. The fourth column provides specification of the bounding box limits. The GUI panel in the lower right corner displays current avoidance data. The top-most label widget simply indicates whether or not avoidance perturbations are currently required. The middle widget displays the reference and perturbed Cartesian coordinates and arm angle (aka *SEW roll angle*). The slider along the bottom provides user setting of the stand-off distance.

The arm motion control panel is in the upper right of the screen image. In Cartesian mode (as shown in Figure 14), it provides operator control of the position and orientation of the end-effector of the currently active arm, as well as the arm angle. In joint mode, the desired target values of the seven joint angles are specified.

The Test Panel, immediately above the main control panel, is comprised of buttons that allow the user to select from ten vista points for viewing the simulation, as well as to set up and execute any of ten “canned” test and demonstration cases. Each Setup button puts the dexterous arms into the starting pose for one of the ten test and demonstration cases, selects the active arm for that case, brings up the arm control panel and turns obstacle detection on with an appropriate detection threshold. Each Test or Demo button commands the active arm to execute a trapezoidal trajectory in one or more of the six end-effector coordinates or the arm angle. The same tests may be executed with or without collision avoidance turned on, to demonstrate the behavior of the active arm with **and** without this capability.

Several computer simulation tests are conducted on collision avoidance of the Ranger dexterous arms, and a typical set of results are presented here. The obstacles used in these tests are bounding box surfaces. When a manipulator approaches any side of the bounding box, a virtual wall is detected as an obstacle.

Two simulation case studies are now presented:

4.1 **Simulation 1: Hand Perturbations**

Figure 15a shows the results of a simulation run using the Ranger test program and the collision avoidance software to demonstrate hand position perturbation for collision avoidance. For this run, the x component of the end-effector position is commanded toward a bounding box surface at $x = 107$ cm. The safety zone boundary, taking into account the radius of the link, is at $x = 99.65$ cm, and the *reference* trajectory is the trajectory which would have been followed if collision avoidance were not active. As shown in the Figure, the *perturbed* trajectory enters the safety zone by less than 0.2 cm before settling down to a safety zone

⁴This is a user-set threshold within which impending collisions are reported.

intrusion of less than 0.1 cm. This is in contrast to an intrusion of 3.15 cm into the safety zone when the collision avoidance is deactivated.

Figure 15b shows the perturbations used to modify the trajectory in Figure 15a. As shown, the spring perturbation continues to increase over time, while the damper perturbation decreases toward zero. Because the intrusion is always less than the parameter d_k , the nonlinear gain k reduces the effect of the spring perturbation. The net result is a total perturbation which approaches the value corresponding to the maximum move allowed per iteration, or 0.04 cm in this case. A simulation run demonstrating hand orientation perturbation for collision avoidance is shown in Figure 15c. The perturbation profile is similar to that shown in Figure 15b.

4.2 Simulation 2: Arm Posture Perturbation

Figure 16 shows the results of a simulation run to demonstrate arm angle perturbation for collision avoidance. The corresponding perturbation follows a similar pattern to that shown in Figure 15b. The maximum intrusion into the safety zone is approximately 0.28 cm. The safety zone is such that ϕ is limited to 101.6° . The reference trajectory is the trajectory that ϕ would have followed had avoidance been turned off. When the reference trajectory ϕ_r moves out of the safety zone, the perturbed trajectory again coincides with the reference trajectory. Generally, the avoidance results for this run are very similar to the end-effector test run results discussed before.

Perturbation computation times per iteration, not including the time required for obstacle detection or forward and inverse kinematics computations, are measured to be less than 0.2 msec on a MIPS R4600PC processor running at 100 MHz. Total computation time, including obstacle detection and forward and inverse kinematics computations, are measured to be about 2.5 msec. The Ranger flight computer is expected to be a MIPS R4600 processor, which should have a similar performance for these computations.

5 RRC Experimental Results

In this section, we present a set of laboratory experiments conducted at JPL on real-time collision avoidance of an RRC arm.

The experimental setup[22] consists of two RRC model K12077-DOF arms and control units, a VME chassis, two hand controllers, a SUN Ultra 1 workstation, and a one-third scale partial mockup of the truss structure of the International Space Station, see Figure 17. The VME chassis serves as the real-time controller for the arm under study, and uses three Motorola MC68040 processors along with shared memory card and various data acquisition and communication cards. The VME-based real-time controller interfaces directly with the Multibus-based arm control unit supplied by RRC. This interface is through a two-card VME-to-Multibus adaptor set from the BIT3 Corporation. This allows a high speed bi-directional shared memory interface between the real-time controller and the arm control

unit. The Servo-Level Interface software supplied by RRC enables low-level communication between the VME controller and the arm servo control loops at the rate of 400 Hz. The VME controller is also linked via socket communication to the SUN workstation, where the user interface software resides.

The arm is controlled by a configuration controller [22] which runs on a CPU with the computational time of about 1 msec. This controller ensures that the six end-effector position and orientation coordinates, as well as the arm angle, track user-specified trajectories. The user commands seven Cartesian trajectories for the arm task coordinates using either the trajectory generation software or the hand controllers. The configuration controller causes the arm to execute the commanded motion for the end-effector and the elbow in the absence of workspace obstacles. During arm motion, the obstacle detection software running on another CPU in the VME chassis continuously computes the distances between the arm links and the workspace obstacles. The collision avoidance strategy described in Section 2 and shown in Figure 7 is implemented in the VME controller so that the arm coordinates are perturbed from their commanded trajectories as soon as an impending collision is detected.

Six experimental case studies on real-time collision avoidance are now described. In all experiments, $k_i = 0.2$, $k_p = 0.2$, and the units of length and angle are meters and radians, respectively. The numerical values of k_i and k_p are found empirically after a few trial-and-error runs.

5.1 Experiment 1: Position perturbation

The goal of this experiment is to demonstrate perturbation of the end-effector position in order to avoid an impending collision. A rectangular “window” representing an opening in the Space Station truss structure is placed in the arm workspace parallel to the world frame y-z plane, and the four sides of the opening are defined as obstacles in the collision detection database. The end-effector of the RRC arm is commanded to move to the center of the opening initially and then execute a diamond-shaped path inside the opening. Figure 18 (dashed line) shows the Cartesian path traversed by the end-effector when the obstacle avoidance capability is disabled. The path is designed so that the end-effector is moved close to the four sides of the opening with a clearance of 1 to 2 cm and then returns to the center. The experiment is then repeated with the collision avoidance enabled. The end-effector is commanded to traverse the *same* path as before. However, when the end-effector is now closer to a side than the user-specified stand-off distance $d_r = 10$ cm, the collision avoidance software inhibits motion toward the side. The path traversed by the end-effector in this case is shown by the solid line in Figure 18. Observe that while the end-effector motion in the z (or y) direction is inhibited to avoid collision, the commanded motion in the remaining direction (y or z) is faithfully executed. Notice that collision avoidance is accomplished by truncating the peaks of the diamond automatically to maintain the specified minimum distance to the sides, thus turning the quadrilateral diamond into an octagon.

5.2 Experiment 2: Orientation perturbation

In this experiment, we demonstrate perturbation of the end-effector orientation to avoid collision. A large box is attached to the truss structure mockup in the laboratory. The end-effector is initially positioned somewhat above the box, with the forearm pointing diagonally upward. The end-effector is then commanded to traverse a trapezoidal path that will move it diagonally toward the top of the box, then parallel to the edge, and finally diagonally away from the box. Figure 19 shows the experimental results obtained in this case. With collision avoidance disabled, the wrist comes very close to the edge of the box. When the collision avoidance is enabled, the wrist will automatically rotate away from the edge of the box in order to maintain the specified clearance. This behavior is shown in Figure 19, indicating a rotation about the world frame x-axis, where the commanded rotation about the x-axis is 0.6 radians.

5.3 Experiment 3: Arm angle perturbation

The purpose of this experiment is to demonstrate perturbation of the arm posture, i.e. change in the arm angle, in order to avoid an impending collision. The second RRC arm is positioned in the upright configuration to pose as a vertical obstacle during the first arm motion, and the data representing the second arm configuration is inputted to the obstacle detection software. The first arm is then commanded to execute a trajectory that brings its elbow very close to the second arm, and the arm motion is then reversed. The dashed line in Figure 20 shows the constant commanded value of 0.493 radians for the arm angle. With collision avoidance disabled, the constant commanded arm angle is maintained and the elbow almost touches the second arm. The collision avoidance is then enabled and the same arm motion is commanded. The experimental results are depicted by the solid line in Figure 20. The results clearly demonstrate that the elbow now rotates away from the second arm in order to maintain a clearance of $d_r = 20$ cm specified by the user.

5.4 Experiment 4: Arm-to-base collision avoidance

The goal of this experiment is to demonstrate arm-to-base collision avoidance. The second RRC arm is configured to a specific set of joint angles and base position, and this information is inputted to the collision detection database for the first arm. The end-effector of the first arm is commanded to move close to the second arm base pedestal. Figure 21 (dashed line) depicts the top-down view (x-y plane projection) of the end-effector motion with the collision avoidance disabled. It is observed that the end-effector comes to within about 2 cm of the base pedestal. With the collision avoidance enabled, the end-effector is commanded to execute the *same* motion, and the results are shown by a solid line in Figure 21. Observe that the end-effector now follows a smooth trajectory to avoid the second arm base pedestal and maintain the stand-off distance of $d_r = 10$ cm to the base.

5.5 Experiment 5: Self-arm collision avoidance

In this experiment, we demonstrate self-collision avoidance for the RRC arm. The arm is positioned initially “curled in” on itself, with the end-effector and wrist pointed in opposite directions and the end-effector pointed toward the upper-arm link away from the truss structure. The end-effector is commanded to move toward the truss. The top-down view (x-y projection) of the path traversed by the end-effector is shown by a dashed line in Figure 22. With collision avoidance disabled, the end-effector comes very close to the upper-arm. The collision avoidance is now enabled and the *same* end-effector motion is commanded. The experimental results are shown by a solid line in Figure 22. The end-effector deviates from the nominal path and makes a “detour” to stay clear of the upper-arm and maintain the stand-off distance of $d_r = 10$ cm.

5.6 Experiment 6: Teleoperation

The goal of this experiment is to demonstrate the collision avoidance capability when the arm is under teleoperation. In this experiment, commands for the end-effector position and orientation and the arm angle are issued by the operator acting on two hand controllers. This is in contrast to previous experiments where the commanded motions are produced by the trajectory generation software. Experiment 1 is repeated with the user teleoperating the end-effector within the fake truss opening using the hand controller. While the end-effector is away from the sides of the opening, the teleoperated commands are executed. As soon as the end-effector is commanded to move close to a side, a counter-command is generated by the collision avoidance software that nullifies the teleoperated command and inhibits motion toward the side. Thus erroneous teleoperated commands that would otherwise cause collision are corrected on-line and in real time. This is an important augmentation to **teleoperation**, particularly for operation in partially or completely occluded regions of the workspace.

6 Conclusions

A new approach for real-time collision avoidance of dexterous 7-DOF arms is developed and demonstrated in this paper. This approach is based on representing the proximity of the arm to workspace obstacles by virtual forces, and servoing these forces to zero by employing an outer feedback loop around the inner arm position control system.

The approach presented in this paper is equally applicable to both model-based and sensor-based collision avoidance. For the sensor-based case, the on-line distance computations are replaced by the readings provided by the proximity sensors mounted on the arm[6]. Furthermore, the proposed approach applies to both stationary and moving obstacles (as in Experiment 5), since the distance computations (or measurements) are updated continuously in real time. Finally, the approach is pragmatic because: it is simple and **computationally**

very fast, it requires minimal modification to the arm positioning system, and it applies to *whale-arm*, not just the end-effector, collision avoidance.

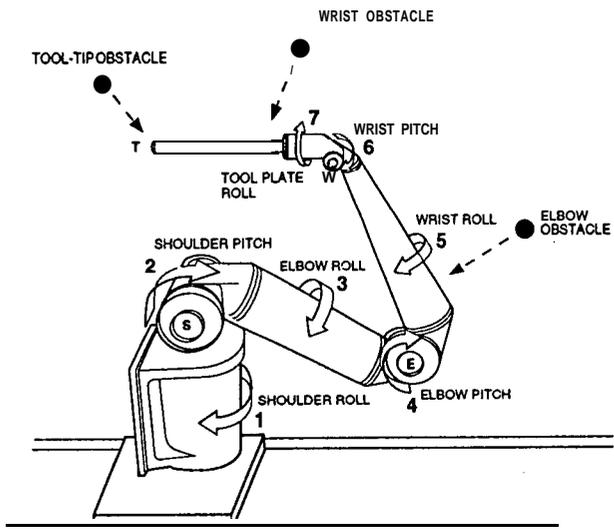
7 Acknowledgments

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration (Code SM Telerobotics Program). The 3-D graphical simulation of the Ranger was provided by the University of Maryland, and is gratefully acknowledged. Thanks are also due to Robert Steele of JPL for integration of the algorithms into the real-time arm control software used in Section 5.

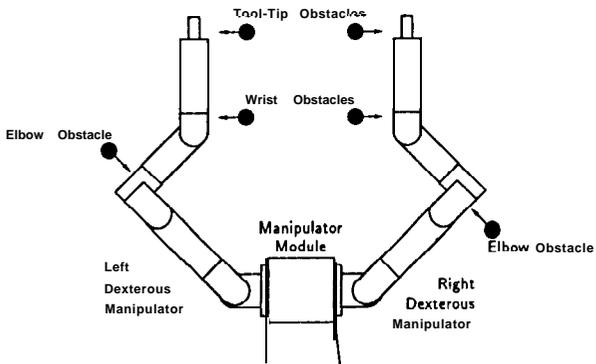
8 References

1. A. Maciejewski and C. Klein: "Obstacle avoidance for cinematically redundant manipulators in dynamically varying environments", Intern. Journ. of Robotics Research, vol. 4, no. 3, pp. 109-117, 1985.
2. O. Khatib: "Real-time obstacle avoidance for manipulators and mobile robots", Intern. Journ. of Robotics Research, pp. 90-98, 1986.
3. T.S. Wikman and W.S. Newman: "A fast, on-line collision avoidance method for a cinematically redundant manipulator based on reflex control", Case Western Reserve University, CAISR Technical Report 91-160, 1991.
4. C.L. Boddy and J.D. Taylor: "Whole-arm reactive collision avoidance control of kinematically redundant manipulators", Proc. IEEE Intern. Conf. on Robotics and Automation, vol. 3, pp. 382-387, Atlanta, 1993.
5. K. Glass, R. Colbaugh, D. Lim, and H. Seraji: "Real-time collision avoidance for redundant manipulators", IEEE Trans. on Robotics and Automation, vol. 11, no. 3, pp. 448-457, 1995.
6. H. Seraji, R. Steele, and R. Ivlev: "Sensor-based collision avoidance: Theory and experiments", Journal of Robotic Systems, vol. 13, no. 9, pp. 571-586, 1996.
7. H. Seraji: "Configuration control of redundant manipulators: Theory and implementation," IEEE Trans. on Robotics and Automation, Vol. 5, No. 4, pp. 472-490, 1989.
8. H. Seraji, M.K. Long, and T.S. Lee: "Motion control of 7-DOF arms: The configuration control approach", IEEE Trans. on Robotics and Automation, vol. 9, no. 2, pp. 125-139, 1993.

9. J.J. Craig: *Robotics: Mechanics and Control*, Addison-Wesley Publishing Company, New York, 1989.
10. J. De Schutter: "A study of active compliant motion control methods for rigid manipulators based on a generic scheme", *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 2, pp. 1060-1065, Raleigh, 1987.
11. K.S. Narendra and J.H. Talor: *Frequency Domain Criteria for Absolute Stability*, Academic Press Publishers, New York, 1973.
12. H. Ding and W. Schiehlen: "On controlling robots with redundancy in an environment with obstacles," *Proc. Symp. on Robot Control*, pp. 771-776, Capri, Italy, 1994.
13. J. Cohen, M. Lin, D. Manocha and K. Ponamgi: "I-COLLIDE: an interactive and exact collision detection system for large-scaled environments," *Proc. ACM Int. 3D Graphics Conference*, pp. 189-196, 1995.
14. E. Cheung and V. Lumelsky: "Proximity sensing in robot manipulator motion planning: system and implementation issues," *IEEE Trans. on Robotics and Automation*, pp. 740-751, 1989.
15. J. Feddema and J. Novak: "Whole arm obstacle avoidance for teleoperated robots," *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 3303-3309, San Diego, 1994.
16. B. Bon: "Collision detection algorithms for Ranger," Internal JPL Document, 1996.
17. D. Akin and R. Howard: "The roll of free-flight in space telerobotic operations," *International Conference on Automation and Robotics*, 1993.
18. D. Akin and P. Churchill: "Robotics architectures for telerobotics flight operations," *AIAA Space Programs and Technologies Conference*, Huntsville, 1993.
19. G. van Rossum and J. de Boer: "Interactively testing remote servers using the Python programming language," *CWI Quarterly*, Vol. 4, Issue 4, pp. 283-303, Amsterdam, Holland, 1991.
20. G. van Rossum: "An introduction to Python for Unix/C programmers," *Proc. NLUUG najaarsconferentie (Dutch UNIX users group meeting)*, 1993.
21. J. K. Ousterhout: *Tcl and the Tk Toolkit*, Addison-Wesley Publishing Company, New York, 1994.
22. D. Lim, T.S. Lee, and H. Seraji: "A real-time control system for a mobile dexterous 7-DOF arm", *Proc. IEEE Intern. Conf. on Robotics and Automation*, vol. 2, pp. 1188-1195, San Diego, 1994.



(a) Dexterous 7-DOF RRC arm



(b) Ranger dexterous arms

Figure 1: Examples of dexterous robotic arms

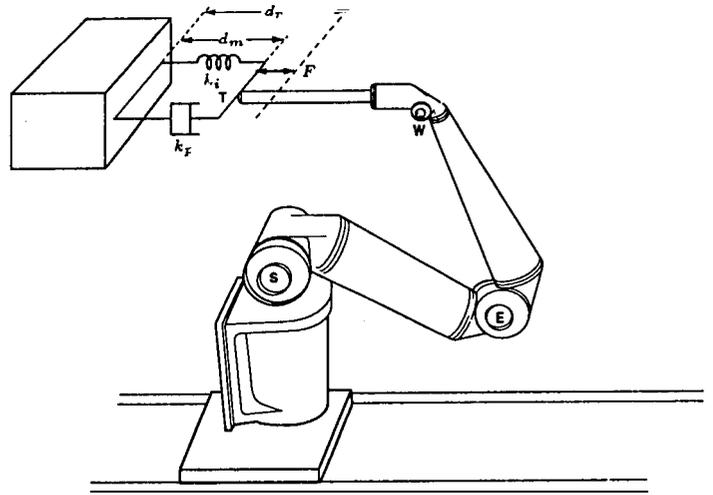


Figure 2: Spring-plus-damper model for intrusion force generation

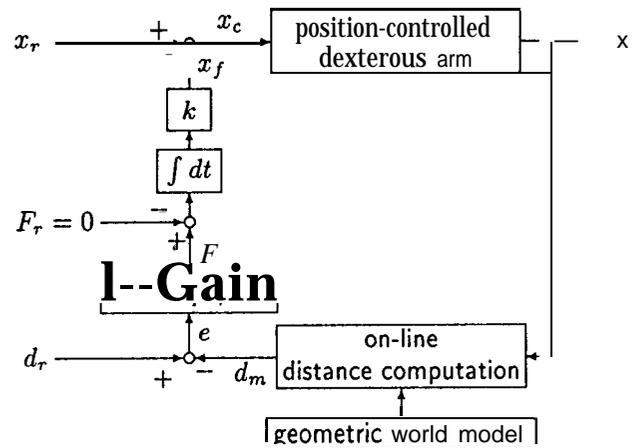


Figure 3: Virtual force control approach to collision avoidance

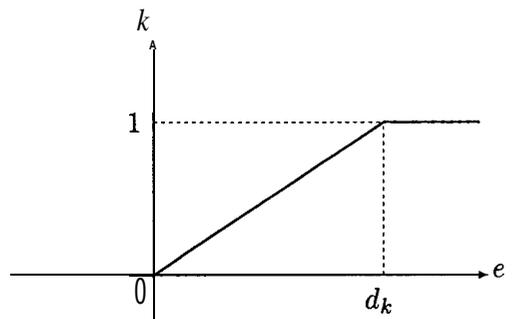


Figure 4: Nonlinear gain characteristics

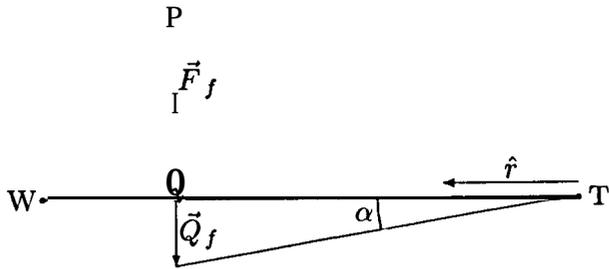


Figure 5: Geometry for orientation perturbation

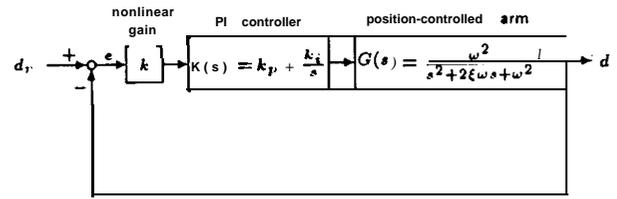


Figure 8: Block diagram of collision avoidance system

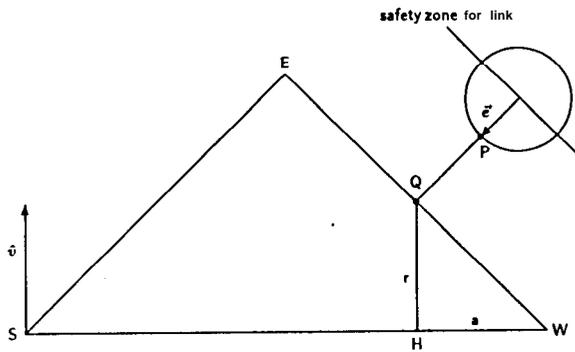
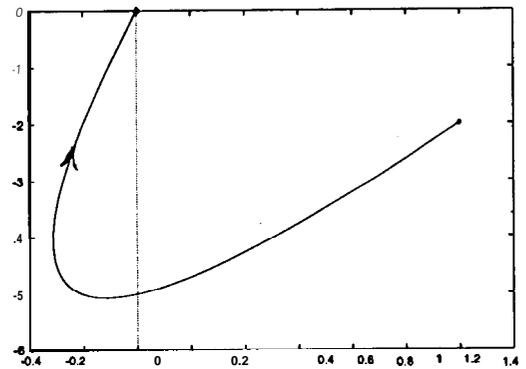


Figure 6: Geometry for arm angle perturbation



(a) Popov plot for $k_p = 2$

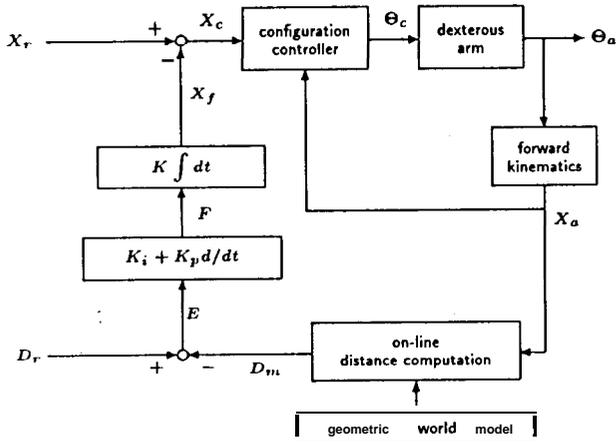
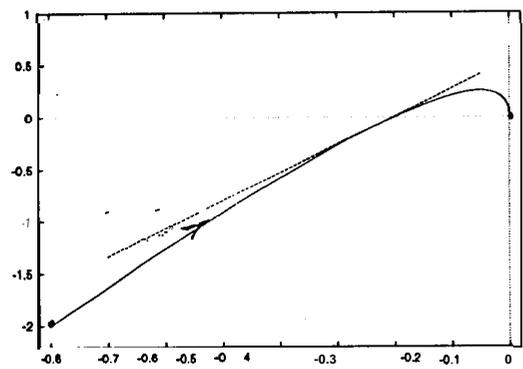


Figure 7: Control diagram for whole-arm collision avoidance



(b) Popov plot for $k_p = 0$

Figure 9: Popov plots

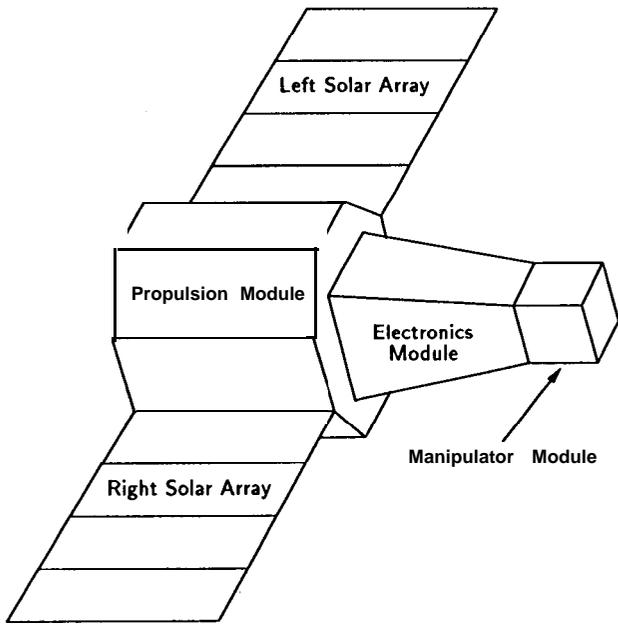
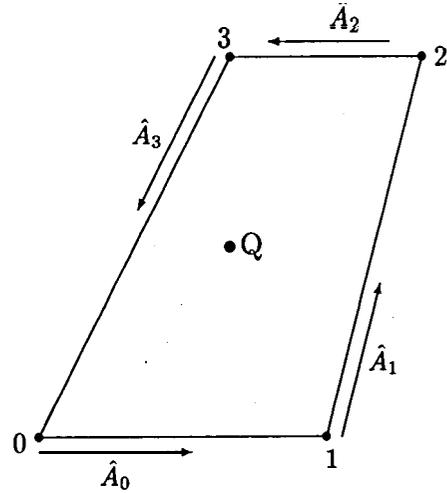


Figure 10: Ranger flight vehicle geometry (arms not shown)



(a) Polygonal face viewed from above

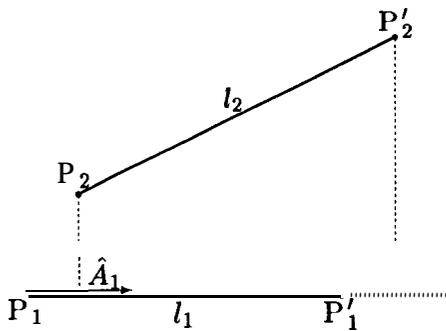
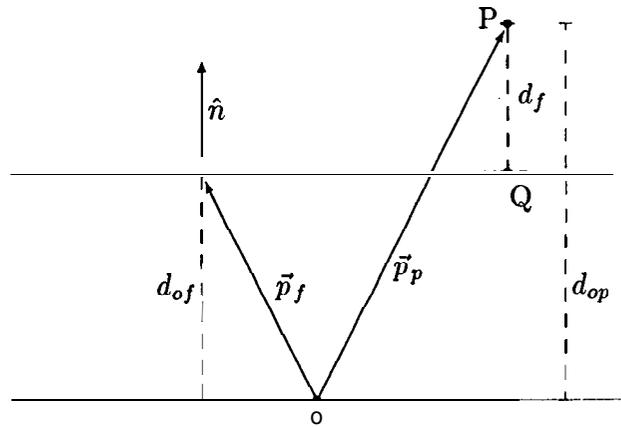


Figure 11: Two finite-length line segments



(b) Face geometry viewed edge-on

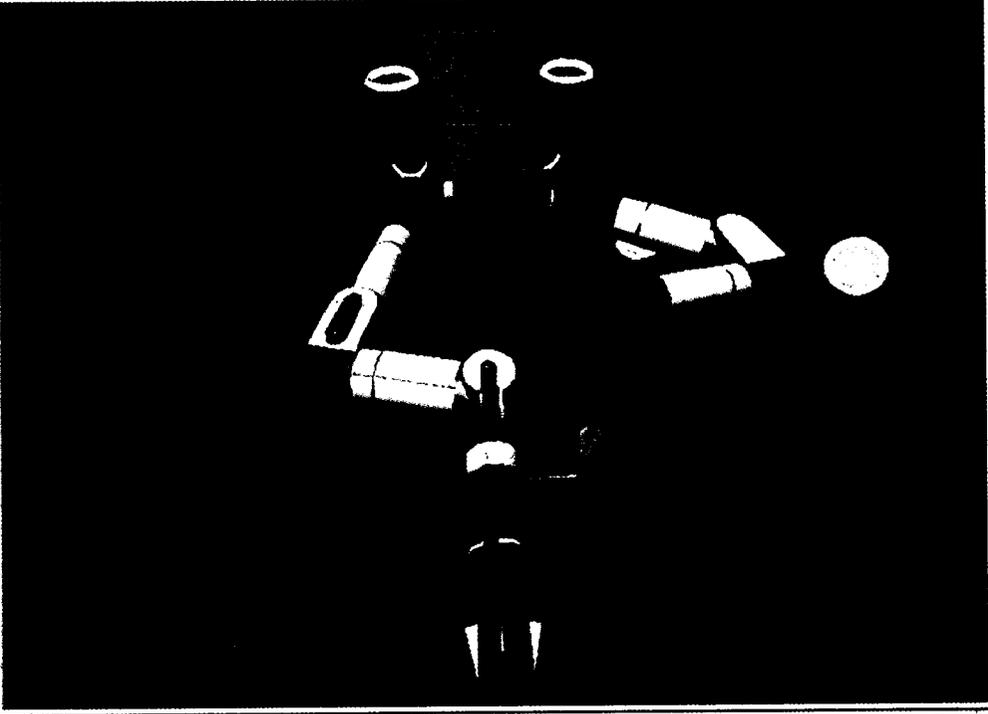
Figure 12: Polygonal face geometry for distance computation



Figure 13: Ranger performing an on-orbit experiment

Figure 14: The JPL Ranger graphical user interface

ssltank regular
□



Tool tip X position wrt vehicle Frame, cm

-150	-100	-50	0	50	100	150
------	------	-----	---	----	-----	-----

Tool tip Y position wrt vehicle Frame, cm

-150	-100	-50	0	50	100	150
------	------	-----	---	----	-----	-----

Tool tip Z position wrt vehicle Frame, cm

-150	-100	-50	0	50	100	150
------	------	-----	---	----	-----	-----

Tool tip roll wrt vehicle Frame, degrees

-88	-68	-44	0	44	68	88
-----	-----	-----	---	----	----	----

Tool tip pitch wrt vehicle Frame, degrees

-188	-98	0	98	188		
------	-----	---	----	-----	--	--

Tool tip yaw wrt vehicle Frame, degrees

-188	-98	0	98	188		
------	-----	---	----	-----	--	--

SEW roll angle, degrees

-188	-98	0	98	188		
------	-----	---	----	-----	--	--

ShowCart
Home
Reset
EXIT

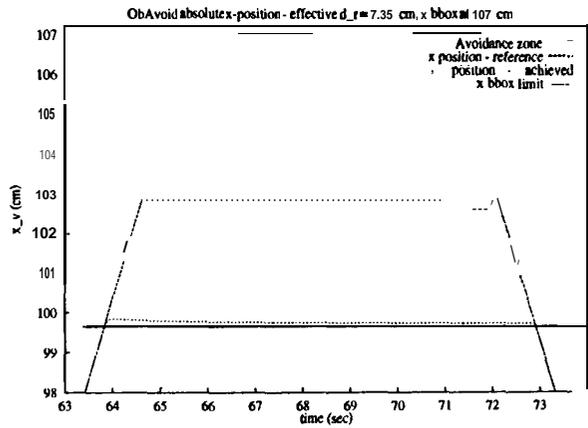
Test Panel
□

Setup 1	Setup 2	Setup 3	Setup 4	Setup 5	Setup 6	Setup 7	Setup 8	Setup 9	Setup 10
Test 1	Test 2	Test 3	Test 4	Test 5	Demo 6	Demo 7	Demo 8	Demo 9	Demo 10
View 1	View 2	View 3	View 4	View 5	View 6	View 7	View 8	View 9	View 10

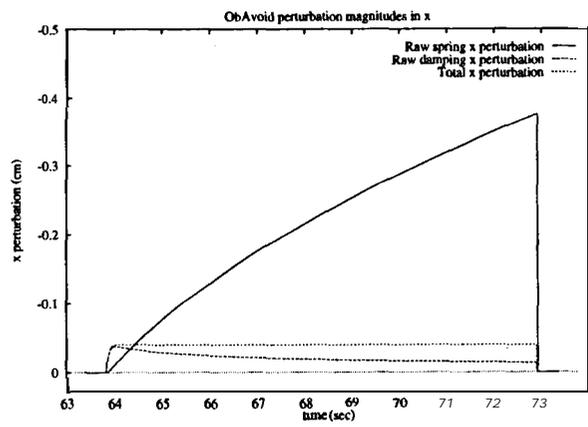
r_gui
□

Joint Control		Minimum distance: 5.83 cm	Bounding box limit in
Cartesian Control	Left Dexterous Arm	Objects nearest to colliding: (Link) Left tool-link (95.85, -8.23, 0.00) cm (Link) Right tool-link (95.28, 10.29, 0.00) cm	155
Collision Detection	Right Dexterous Arm		Bounding box limit in
Collision Avoidance		Collision detection threshold, cm	185
Test Panel		6	155

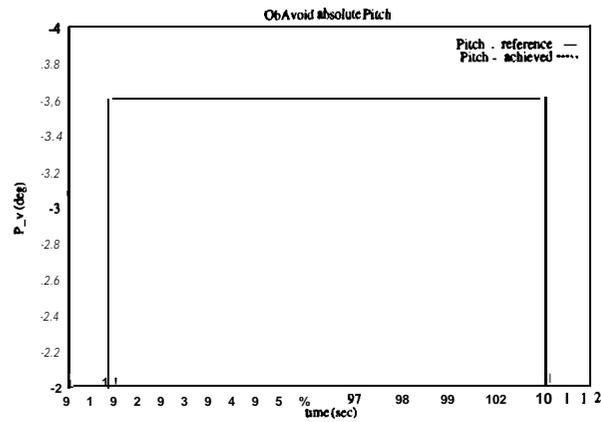
Avoidance perturbations required
Reference Cartesian position and orientation: 95.8, -8.8, 0.0 cm; -88.0, -98.0, 98.0 deg 0.0 degrees SEW roll angle Achieved Cartesian position and orientation: 95.8, -8.2, 0.0 cm; -88.0, -98.0, 98.0 deg 0.0 degrees SEW roll angle
Collision avoidance stand-off distance, cm
6



(a) Reference and achieved x trajectories



(b) Collision avoidance position perturbations



(c) Reference and achieved pitch trajectories

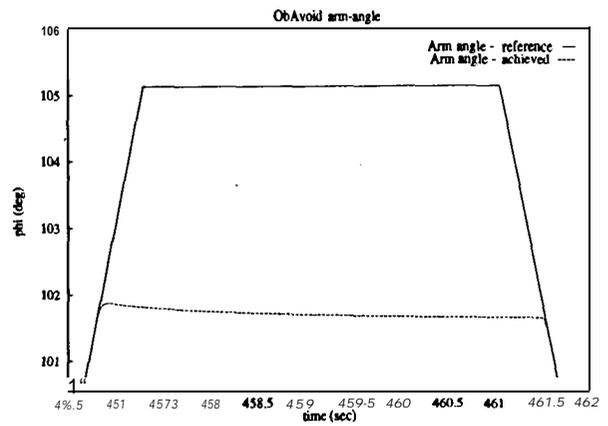


Figure 16: Arm angle perturbation example: reference and achieved ϕ trajectories

Figure 15: End-effector perturbation examples

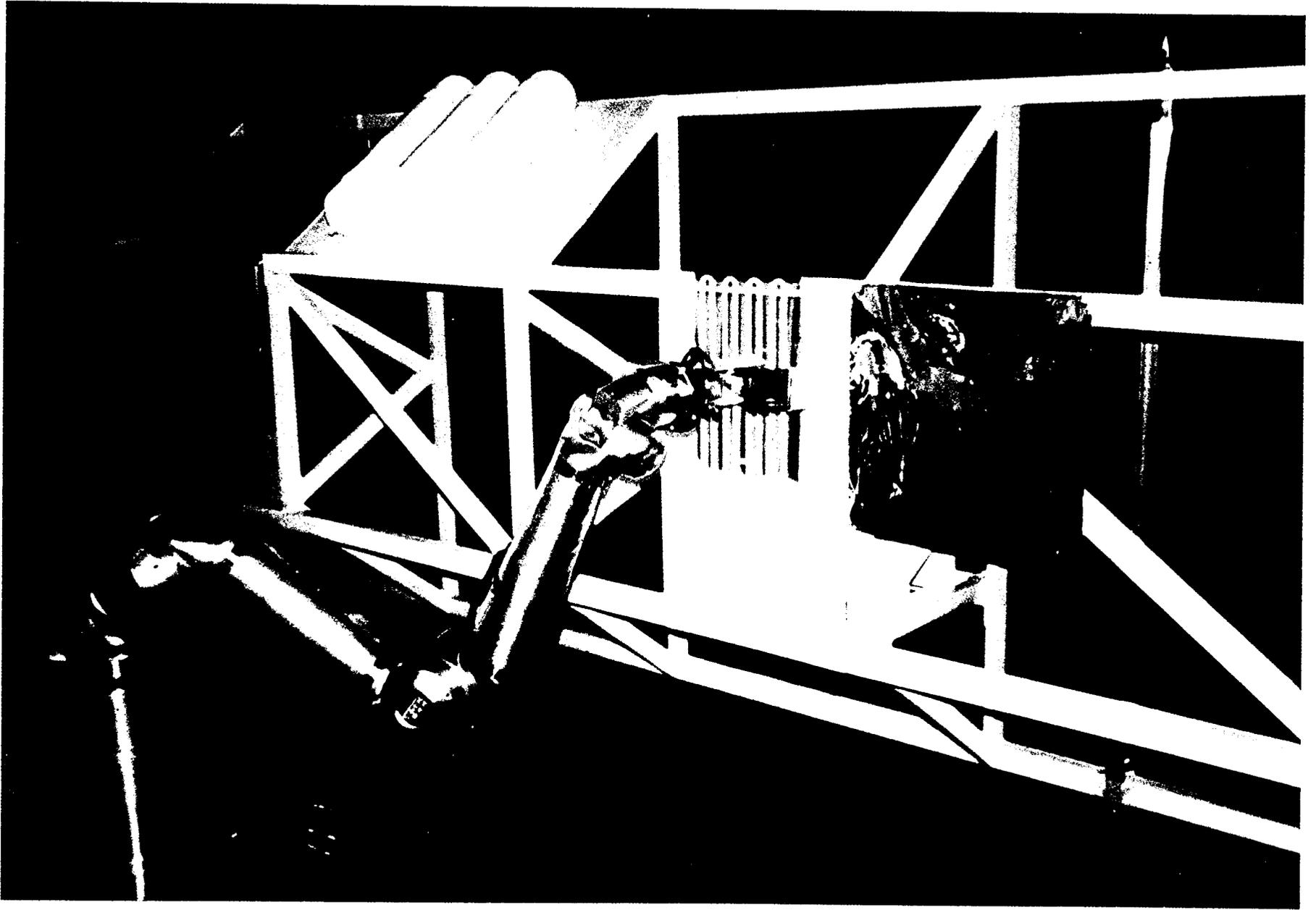


Figure 17: JPL Remote Surface Inspection Laboratory

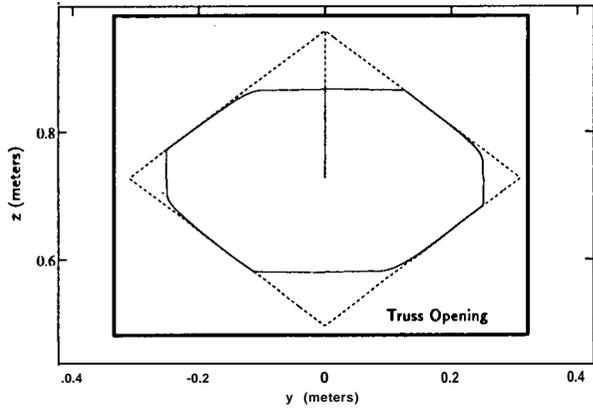


Figure 18: Position perturbation for collision avoidance

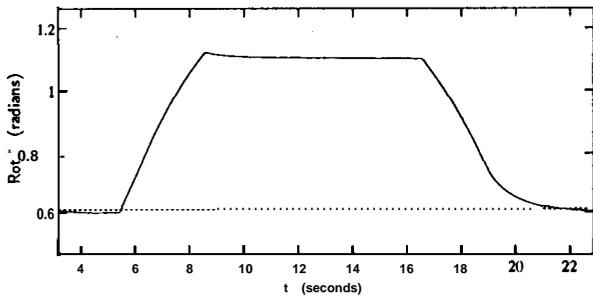


Figure 19: Orientation perturbation for collision avoidance

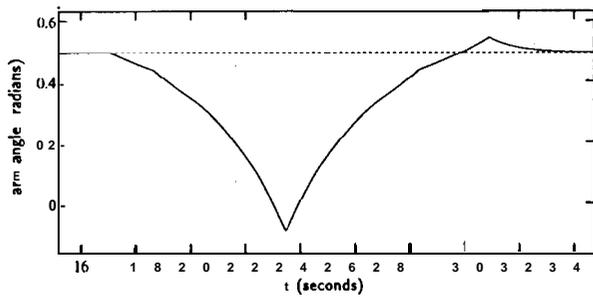


Figure 20: Arm angle perturbation for collision avoidance

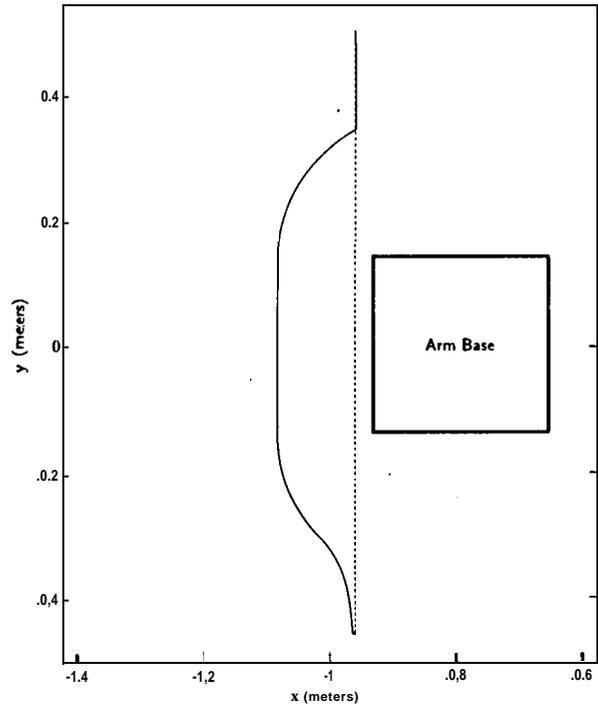


Figure 21: Arm-to-base collision avoidance

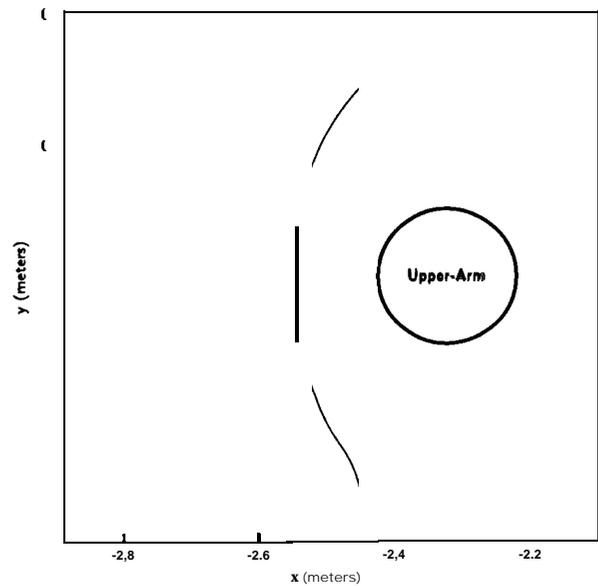


Figure 22: Self-arm collision avoidance