Real-Time Model-Based Obstacle Detection for the NASA Ranger Telerobot

Bruce Bon **Homayoun Seraji**

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109, USA

Abstract

This paper describes the approach and algorithms developed for real-time model-based obstacle detection and distance computation for the NASA Ranger Telerobotic Flight Experiment. Objects of interest, such as manipulator arms or the Ranger vehicle and solar arrays, are modeled using a small set of component types: edges, polygonal faces and cylindrical links. Link positions are computed using standard forward kinematics, and distances between object components are computed directly using equations derived from geometry. Prioritized lists of potential obstacles for each manipulator link eliminate needless distance computations and assure that the most likely obstacles are checked, even if the computation is terminated early due to real-time constraints. A test program, utilizing a 3-D graphical simulation and providing a graphical user interface for operator control, has been developed and used to test and demonstrate obstacle detection. An earlier paper [1] described how the obstacle detection results are utilized for collision avoidance.

1 Introduction

The NASA Ranger Telerobotic Flight Experiment [2, 3, 4, 5], 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 dexterous seven degree-of-freedom manipulator arms mounted on a cubical base, in addition to grapple and vision arms. The two dexterous arms will be used, both individually and cooperatively, to perform a variety of manipulation experiments and servicing operations (Figure 1). JPL has developed model-based on-line obstacle detection and collision avoidance capabilities for the Ranger project. These capabilities do not currently exist in the Ranger baseline control system, and erroneous operator commands can cause collision between the dexterous arms and the camera and grapple arms, the base, or the task board. On-line obstacle detection and avoidance will enable collision-free motions of the arms throughout the workspace. It will also cause a reduction in the Ranger operation time, since possible motions with potential collisions will not be executed. This capability will increase the safety of the Ranger during the operation of the arms, a feature which is vital to the success of the Ranger mission.

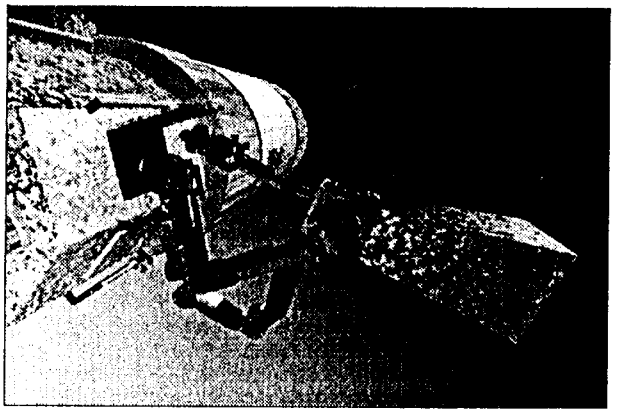

Figure 1: Ranger performing an on-orbit experiment

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 (e.g. [6, 7]). Oftentimes, designs for real-time collision avoidance have used sensor-based obstacle detection (e.g. [8, 9, 10, 11]), but the Ranger spacecraft

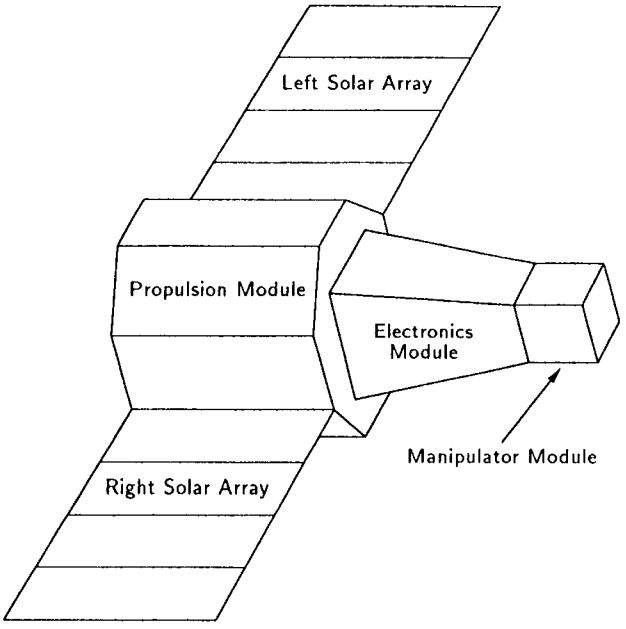

Figure 2: Ranger flight vehicle geometry, as modeled (arms not shown)

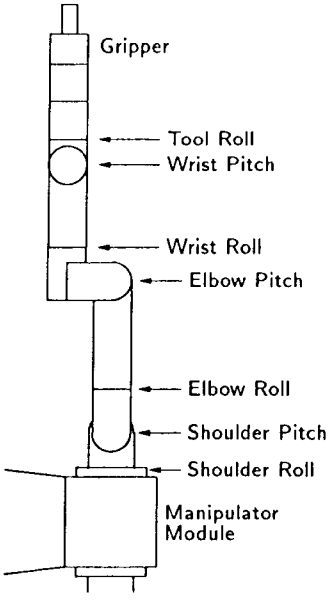

Figure 3: Top view of Ranger manipulator module and left dexterous manipulator with all zero joint angles

does not include 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 virtual sensor suite instrumenting the entire spacecraft and manipulators. The goal of our obstacle detection effort, therefore, is to provide a real-time model-based obstacle detection capability which is tailored to Ranger requirements and which can detect the 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.

Figure 2 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 cubical *manipulator module*. Four robotic manipulators will be attached to the manipulator module: two 7-degree-of-freedom (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 3 shows a top view of the current design of the Ranger dexterous manipulator that is mounted on the left side of the manipulator module; an identical manipulator is mounted on the right side of the manipulator module.

In order to warn of impending collisions, the fundamental requirement is to find the minimum 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 2 and 3 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 spacecraft in order to meet safety requirements.

2 Approach

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 obstacle detection software assumes that only one manipulator is moved at a time, providing the nearest obstacle data for one manipulator at a time.[1] For each manipulator link that has moved, the minimum distance from the link to all objects in its *obstacle list* (see description below) is measured. The obstacle detection function returns the nearest object, its distance, and the link and object nearest points for three categories: manipulator-to-manipulator, manipulator-to-

[1] If more than one manipulator is moving, the software may be used on each manipulator in turn.

vehicle and manipulator-to-bounding box. The *bounding box* is defined to be a virtual rectangular box enclosing the arms and centered on the Ranger manipulator module. When a manipulator approaches any side of the bounding box, an obstacle is detected.

When a manipulator link is approaching collision with another manipulator, it is sufficient to test for line-segment to line-segment distances, taking into account the radii of the links. When a manipulator link is approaching collision with a vehicle face, then the nearest point to the link on the face will either be (1) the nearest point to the link along one of the edges of the face, or (2) the nearest point of the distal end-point of the link to the face. When the distal end-point is the nearest point to the face, then the projection of the distal end-point into the face plane[2] will be within the boundaries of the face. Thus, if we can detect that this projection point is not within the face, then we do not need to find the actual nearest point on the face; otherwise the desired nearest point in the face plane is the projection point.

The obstacle detection database (ODDB) contains geometrical data about the Ranger spacecraft and its manipulators for use by the obstacle detection and collision avoidance software. The potential obstacles in the ODDB are *faces*, *edges*, and manipulator *links*. The ODDB 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 list*). The obstacle list[3] typically has a relatively small number of entries, because most of the objects in the ODDB are not within the reach space of the link and are thus not candidates for collision. By only checking the cases described above, and by only checking feasible obstacles, obstacle detection computation is minimized.

All coordinates in the ODDB are expressed in the *manipulator frame*, a right-handed reference frame whose origin is at the center of the Manipulator Module. For each link of each arm, the ODDB contains the proximal and distal end-points of a line segment through the axis of the link, and a radius for the link. Because there are only 5 points per dexterous arm, corresponding to end-points for all Ranger arm links, 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 ODDB contains a list of vehicle vertex points, whose coordinate values are constant in the manipulator frame. Each edge contains pointers to two of the vertex points. Each face contains pointers to vertex points surrounding the face in a counterclockwise fashion. These data structures also contain additional redundant data, computed at initialization time, to make distance computations fast.

For link-to-link distances, the minimum 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 minimum distance between link surfaces. The geometric link surface that this models is a cylinder of uniform radius with hemispherical ends of the same radius.

For link-to-edge distances, the minimum 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 minimum 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 measurement 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 manipulator-frame axis directions 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.

[2]The *projection* of the distal end-point is the intersection with the face plane of the perpendicular to the face plane which passes through the distal end-point.

[3]The 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.

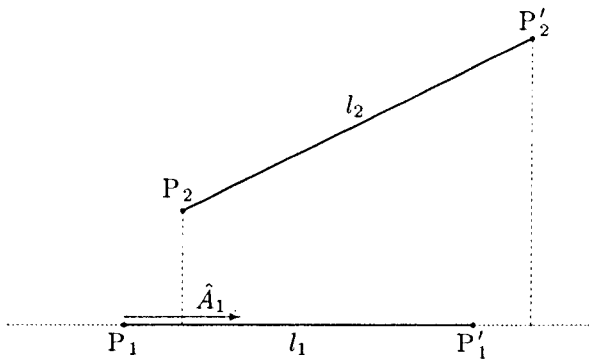

Figure 4: Two finite-length line segments

3 Line Segment to Line Segment Distance Computation

In order to determine the minimum distance between two arm links or between an arm link and the edge of a polygonal face, it is necessary to compute the minimum distance between one line segment (the arm link axis) and another line segment (another axis or the edge). Figure 4 illustrates the problem. Line segment 1 is defined by the end-points P_1 and P'_1, where:

$$\begin{aligned} P_1 &\equiv (x_1, y_1, z_1) \\ P'_1 &\equiv (x'_1, y'_1, z'_1) \\ l_1 &\equiv \textit{length of line segment } 1 = |P'_1 - P_1| \\ \hat{A}_1 &\equiv \textit{direction cosines of line } 1 = (a_1, b_1, c_1) \end{aligned}$$

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

$$X = P_1 + t_1 \hat{A}_1 \qquad (1)$$

where 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.

3.1 Derivation of the minimum distance between two infinite non-parallel lines

We wish to find the minimum 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 minimum distance.

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

$$(M_2 - M_1) \cdot \hat{A}_1 = 0 \qquad (2)$$
$$(M_2 - M_1) \cdot \hat{A}_2 = 0 \qquad (3)$$

Then:

$$M_2 - M_1 = \vec{D}_{12} + t_2 \hat{A}_2 - t_1 \hat{A}_1 \qquad (4)$$

where $\vec{D}_{12} \equiv P_2 - P_1$. Substituting this result into equations (2) and (3), and solving for t_1 and t_2:

$$t_1 = a + t_2 b \qquad (5)$$

$$t_2 = \frac{a\,b - c}{1 - b^2} \qquad (6)$$

assuming $b \neq \pm 1$, where $a \equiv \vec{D}_{12} \cdot \hat{A}_1$, $b \equiv \hat{A}_1 \cdot \hat{A}_2$, and $c \equiv \vec{D}_{12} \cdot \hat{A}_2$. The minimum distance between 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 (5) and (6).

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 [15].

3.2 Correcting the nearest points for two non-parallel finite line segments

Whether or not the line segments are parallel, the nearest points and minimum distances for infinite lines may not correspond to the correct values for finite-length line segments. In order to find the correct nearest points and minimum 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 values 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 lies 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 the 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.

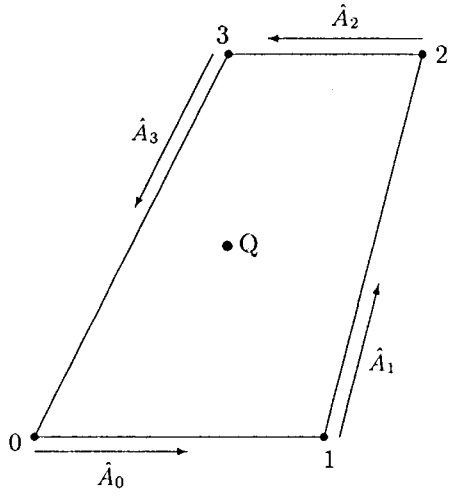

Figure 5: A polygonal face viewed from above

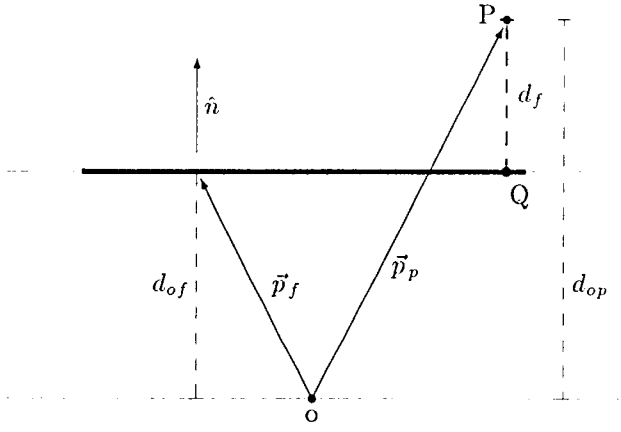

Figure 6: Face geometry viewed edge-on

4 Point to Polygonal Face Distance Algorithm

This section presents the mathematical details of the algorithm used for computing the minimum distance between a point and a polygonal face in 3-dimensional space. The algorithm presented below is valid under the following assumptions:

1. All faces are planar, convex polygons.
2. 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 5 and 6 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 5, 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, $\vec{p}_f$ (Figure 6):

$$d_{of} = \hat{n} \cdot \vec{p}_f \tag{7}$$

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

$$d_f = \hat{n} \cdot \vec{p}_p - d_{of} \tag{8}$$

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_f, computed for the distance between 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 position vector of the point less the perpendicular unit vector, $\hat{n}$, times the minimum distance to the plane:

$$Q \equiv \vec{p}_q = \vec{p}_p - d_f \hat{n} \tag{9}$$

Finally, we need to test to see 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:

$$\hat{n} \cdot (\hat{A}_i \times \vec{B}_i) > 0 \tag{10}$$

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

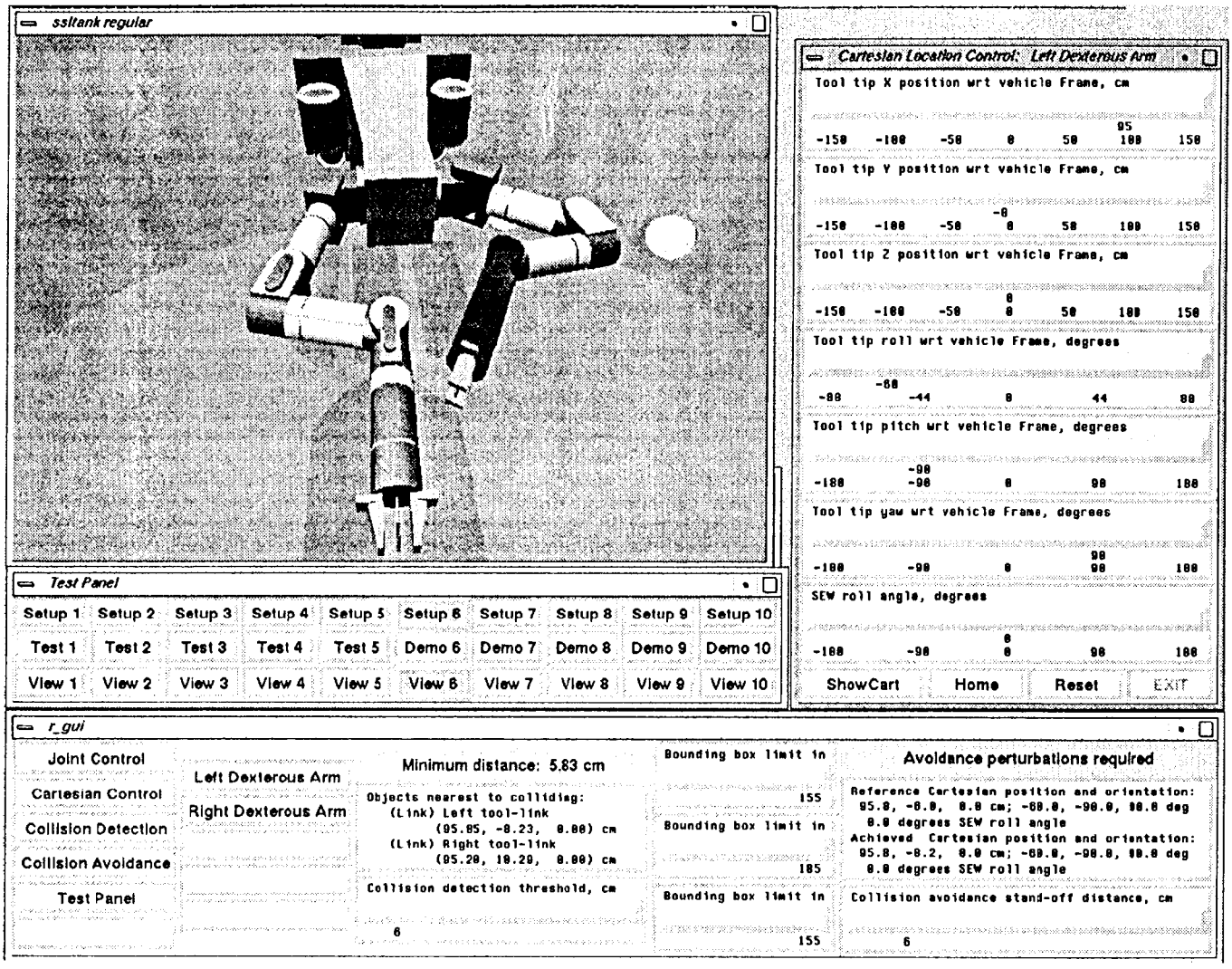

Figure 7: The JPL Ranger graphical user interface

5 Computation Times

Line-segment-to-line-segment 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 measurement computation time should be 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 links' obstacle lists, 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.

6 Graphical User Interface

The software package for obstacle detection and collision avoidance is implemented in C on an SGI Indy under IRIX, but is 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 drives a 3-D graphical simulation provided by the Ranger project. The GUI is implemented in an object-oriented interpretive language called Python [12, 13], controlling widgets provided by Tk [14].

Figure 7 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 Cartesian control panel in the

upper right.

The 3-D graphics displays the Ranger Neutral Buoyancy Vehicle (NBV) rather than the space vehicle and only the dexterous arms are modeled, but the geometry and kinematics of the NBV dexterous arms closely match those of the space vehicle dexterous arms. The nearest obstacle to the currently active arm is identified by a colored line in the 3-D scene connecting the obstacle and the arm link that is closest to it. If the obstacle is within the detection threshold specified by the user, the nearest arm link changes color to red and the connecting line changes from yellow to red.

The left-most column of the main control panel has buttons to select joint or Cartesian control, to turn obstacle detection and collision avoidance on or off, to bring up a button panel for controlling tests 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. The background of the distance panel is green for no obstacle within the detection threshold distance, yellow for an obstacle within the threshold distance, and red if the arm has collided with the obstacle. The fourth column provides specification of the bounding box limits. The right-most column of the main control panel displays collision avoidance data and allows the operator to set the stand-off distance.

The motion control panel is in the upper right of the screen image. In Cartesian mode (as shown in Figure 7), 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 7 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 the 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 the case, brings up the motion control panel, and turns obstacle detection on with an appropriate 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.

7 Conclusions

A real-time model-based obstacle detection software package for the Ranger Telerobotic Flight Experiment is presented. All algorithms described are documented in detail in [15] and have been implemented and extensively tested in a simulation environment, using geometric models corresponding to the latest design of the flight hardware. Extensive tests have shown the software to be efficient and to provide all data needed for collision avoidance. Computation is fast enough that it can easily run within a real-time loop in the flight computer to provide continuous obstacle detection for Ranger operations. Experimental results from laboratory testing using 7-DOF RRC arms are described in a companion paper [16].

In the current implementation, obstacle lists are static and the distance to every potential obstacle on the list for each link of the active arm is evaluated at every iteration. As described in the Approach, the obstacle list structure is designed to allow dynamic addition and deletion of pointers to objects as well as dynamic reordering of each list. Dynamic manipulation of obstacle lists may be used to improve performance by establishing a priority based on changing obstacle distances and on time since last distance computation for each obstacle. Then only the distances to the higher priority obstacles need to be computed on each iteration. Dynamic manipulation of obstacle lists, automated generation of obstacle lists based on an object database, and limiting distance evaluations based on priority and available computation time are areas for future research and development.

Acknowledgements

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 (the Code SM Telerobotics Program). The 3-D graphical simulation program was provided by the Space Systems Laboratory of the University of Maryland.

References

[1] B. Bon and H. Seraji: "On-line collision avoidance for the Ranger Telerobotic Flight Experiment," Proc. IEEE Intern. Conf. on Robotics and Automation, vol. 3, pp. 2041-2048, 1996.

[2] D. Akin and R. Howard: "The role of free-flight in space telerobotic operations," Proc. 1993 Intern. Conf. on Advanced Robotics, Tokyo, 1993.

[3] D. Akin and P. Churchill: "Robotics architectures for telerobotics flight operations," AIAA

Space Programs and Technologies Conference, Huntsville, 1993.

[4] C. Carignan: "A decoupling inverse kinematics algorithm for a seven joint manipulator," SSL publication, University of Maryland, 1992.

[5] J. Graves: "Ranger Telerobotic Flight Experiment program update," Proc. Conference on Space Manufacturing, Princeton, 1995.

[6] 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.

[7] 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.

[8] 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.

[9] J. Feddema and J. Novak: "Whole arm obstacle avoidance for teleoperated robots," Proc. IEEE Intern. Conf. on Robotics and Automation, pp. 3303-3309, San Diego, 1994.

[10] H. Seraji and B. Bon: "Real-time collision avoidance for dexterous 7-DOF arms," submitted for publication, 1996.

[11] 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.

[12] 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, 1991.

[13] G. van Rossum: "An introduction to Python for Unix/C programmers," Proc. NLUUG najaarsconferentie (Dutch UNIX users group meeting), 1993.

[14] J. K. Ousterhout: **Tcl and the Tk Toolkit**, Addison-Wesley, 1994.

[15] B. Bon: "Obstacle detection algorithms for Ranger," internal JPL document, 1996.

[16] H. Seraji, B. Bon and R. Steele: "Experiments in real-time collision avoidance for dexterous 7-DOF arms," Proc. IEEE Intern. Conf. on Robotics and Automation, Albuquerque, 1997.