Teleoperation with virtual force feedback

Robert J. Anderson

Sandia National Laboratories,
Intelligent Machine Systems Division I: 1661,
P.O.Box 5800, Albuquerque, NM 87185[1]

ABSTRACT

In this paper we describe an algorithm for generating virtual forces in a bilateral teleoperator system. The virtual forces are generated from a world model and are used to provide real-time obstacle avoidance and guidance capabilities. The algorithm requires that the slave's tool and every object in the environment be decomposed into convex polyhedral primitives. Intrusion distance and extraction vectors are then derived at every time step by applying Gilbert's polyhedra distance algorithm, which has been adapted for the task. This information is then used to determine the compression and location of nonlinear virtual spring-dampers whose total force is summed and applied to the manipulator/teleoperator system. Experimental results validate the whole approach, showing that it is possible to compute the algorithm and generate realistic, useful psuedo forces for a bilateral teleoperator system using standard VME bus hardware.

1. INTRODUCTION

Sandia National Laboratories has been developing telerobotic technology for environmental restoration, waste management and agile manufacturing. Our goal is to achieve the optimal blend of human and robot behaviors for a given task. One of the prerequisites for intelligent teleoperation is the ability to convey world model information to the operator. This is useful for preventing misguided human operator commands (e.g., knocking out the windows in a glove box, or breaching a waste tank container's walls) for intelligently guiding redundant robots through cluttered environments (e.g., for decommissioning nuclear facilities) or for creating entirely virtual worlds for operator training and task planning. In this paper we describe the development of a real-time, high-resolution 6 DOF, obstacle avoidance representation and avoidance capability for telerobotic systems, using a priori world model information to generate force barriers. The algorithm not only prevents inadvertent collisions, but generates intuitive contact information which the operator can feel using a force-reflecting master.

Sandia's search for flexible, modular teleoperator control systems has lead to the development of SMART (Sequential Modular Architecture for Robotics and Teleoperation) which is currently being used at multiple sites inside and outside of Sandia [1]. SMART is a real-time telerobotic control architecture which allows the user to construct a telerobot system using independent modules describing input devices (e.g., space ball, force reflecting masters), manipulators (e.g., PUMA, Schilling Titan) sensors (proximity sensors, force sensors). Each module represents a one-port or a two-port network element which can either perturb the force or the velocity of the teleoperator system. This paper describes the development of a module which will generate large forces whenever the manipulator tool comes into close proximity to a modeled object in the environment. Additional modules can be combined with this module to achieve force reflection, force compliance or obstacle avoidance behaviors for the manipulator. By attaching a force-reflecting master, the operator can feel virtual forces [2].

---

[1] This work was performed at Sandia National Laboratories and supported by the U.S. Department of Energy under contract DE-AC04-76DP00789.

2. APPROACH

Using the network based passivity philosophy of SMART we decided to generate boundary functions around obstacles using non-linear springs and dampers, where the spring/damper combination would provide zero force disturbance outside a given threshold region, and would ramp up to a large force at the surface of the object. Figure 1 shows a diagram to illustrate this approach. As the gripper approaches the corner workpiece, computer generated spring-dampers push away the gripper from points of nearest contact. The method of using nonlinear springs is similar in concept to the potential field approach [3], but substantially different in implimentation. Here, only object geometry is embedded and no attraction functions exist. The springs provide a barrier function over a very limited region, and serve only to repel. The influence function is zero over the vast majority of the workspace.

In order to calculate the forces from a number of nonlinear springs we developed an algorithm which would compute the distances and the points of nearest contact in real-time. The first step in the development of this algorithm is data representation. We considered three candidates, memory map, C-space and convex polyhedra primitives. Memory map techniques represent a number of methods by which the Cartesian space of the manipulator has been segmented into a 3-D array, and geometric data (e.g. occupancy or object potential) is stored at each location. These methods, although offering great speed, do not readily deal with the tool's geometry, and cannot be readily queried for surface normal information. C-space methods [4], we felt could deal with tool geometry, but would be difficult to apply to systems having 5 or more DOF. Both C-space and Cartesian memory-map approaches we felt would have resolution problems and would not be easily derivible from our polyhedra based graphical representation. Thus, by default, we decided to decompose the environment into convex polyhedra.

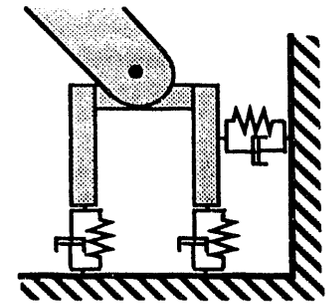

Figure 1: Use of spring-dampers for simulating contact.

Consider the gripper shown in Figure 2a. It can be decomposed into convex polyhedra in many ways. We prefer to choose a decomposition consisting of the minimum number of sections using the maximum amount of overlap. We want the minimum number because it reduces the computation required, and we want maximal overlap to reduce the likelihood of driving the system to a zero-potential at the interface of two polyhedra. Such a decomposition is shown in Fig 2b. The overlap areas are shown in grey.

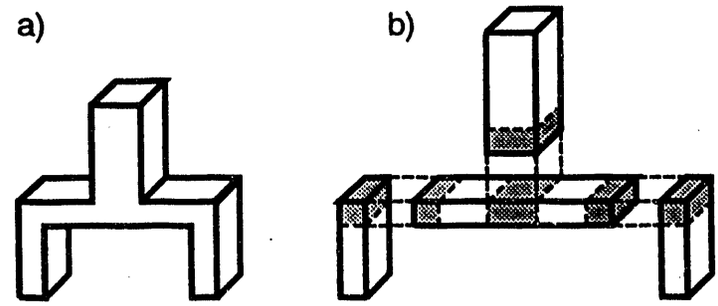

Figure 2: Convex object decomposed into primitives. a) original object; b) decomposed object with shading showing the overlap.

Once every object of interest in the workspace has been decomposed into convex polyhedra, and the manipulator tool has been similarly decomposed, we need to determine when object pairs (i.e., one object from the set of gripper polyhedra, and one object from the set of environment polyhedra) come into close proximity. Furthermore, this needs to be done at a high sampling rate in order to achieve realistic stiffness and a minimal region of perturbation. After studying various possible approaches [5, 6], we decided to implement an algorithm based on Gilbert's algorithm [7] using a few enhancements. The basic algorithm computes the unique distance and the not necessarily unique vector between the convex hulls of two sets of points in Cartesian space. The convex hull of a set of points generates a polyhedron. The algorithm has two main drawbacks. It won't return anything if the two obstacles are in collision, and if two objects have parallel edges or sides the algorithm will only return a single vector with no information about the type of contact. The next two subsections explains how we deal with these drawbacks.

2.1 Dealing with object collisions

The first problem with Gilbert's algorithm is that it gives no informative answer when objects are in collision. It comes back immediately telling when two polytopes overlap, but does not tell which direction the polytopes should move to best extricate themselves. For our real time architecture this was unacceptable. Because our real-time system can only generate a boundary function spring with a finite maximum stiffness (infinite proportional gain is difficult to implement in a sampled data system), we cannot guarantee a priori, that a barrier won't be breached. Thus actual collision of polytopes is always possible. In some cases collision is desirable as we try to simulate membranes and other soft tissue which can naturally be penetrated.

The simplest solution is to drive the objects apart in the opposite direction from the initial collision direction. This solution is unacceptable for sustained collision however, since movement along some other direction might provide quicker disentanglement. To explain this scenario, consider Figure 3, which shows a small rectangle as it moves into a box from the left side. In Fig 3a the polytopes first collide and the extraction vector should oppose the motion. In Fig 3b however the rectangle has intruded substantially into the rectangle, and the extraction vector should be at some angle. Finally, by Fig 3c the rectangle should be extracted by directing a force upwards.

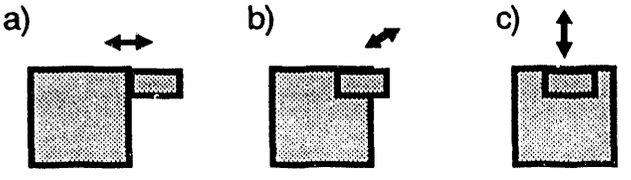

Figure 3: Determining extraction vectors for objects in collision a) Objects just collided; b) Objects after substantial intrusion; c) Objects after full intrusion.

Our solution to this problem is to use reduced primitives. For each convex object primitive we generate a linked list of sub-polytopes such that each sub-polytope is fully contained and centered inside a delta window of its parent. Each sub-polytope has its own embedded sub-polytope until finally the original polytope is reduced to a point. Figure 4 gives several examples. For example a cube can be reduced to any number of smaller cubes, and finally to a point. A long square box can be reduced to a line then to a point. An arbitrary box can be reduced to a plane, then a line, and then finally to a point. A cylinder can be reduced to a line and then to a point.

Using the reduced polytope concept, if two convex objects are found to be in collision then the same test is made using smaller and smaller sub-polytopes until no collision condition exists. The nearest distance vector between the sub-polytopes is used as the extraction vector for the parent polytopes, and the "distance" is derived from the computed sub-polytope distance and the combined delta functions. The beauty of this approach is that the same algorithm can be used, and that time spent in computing overlapping polytopes can be recovered since the sub-polytopes typically contain fewer vertices and are thus faster to compute.

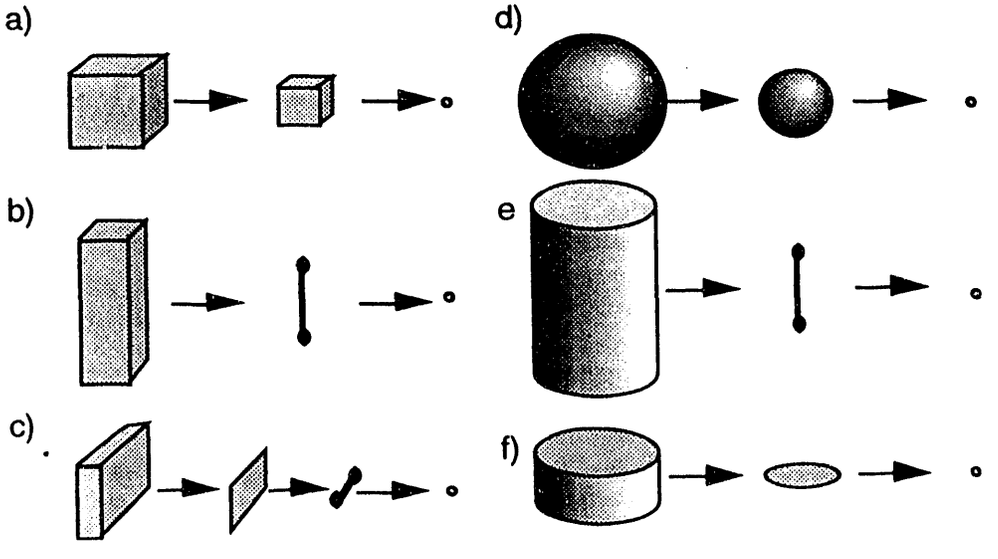

Figure 4: Convex object reductions: a) cube; b) long square box; c) generic box; d) sphere; e) cylinder; f) disk.

Furthermore, the distance function between objects can be made continuous and monotonically non-increasing (although not monotonically decreasing) as a function of object position. This is important since the total spring force is a direct function of position and should not jump in magnitude for an infinitesimally small change of the object's position. The following two-dimensional example clarifies the issue. Consider the triangle moving toward the rectangle in Fig. 5 below. The thick lines show the polytope used for making the distance and extraction vector calculations. In samples a) through d) the extraction vector will point to the right. In sample e) it will point straight up, and in samples f) thru h) it will point to the left. The grey area on the triangle shows the "equidistance" region on the triangle. When the nearest distance vector points to this region on the triangle, the distance function stays constant. This is illustrated in Figure 6, which gives the distance function for the example shown in Figure 5. The sub-polytopes method allows us to use Gilbert's algorithm without significant modification and allows us to achieve a suitable extraction vector whenever two objects collide.

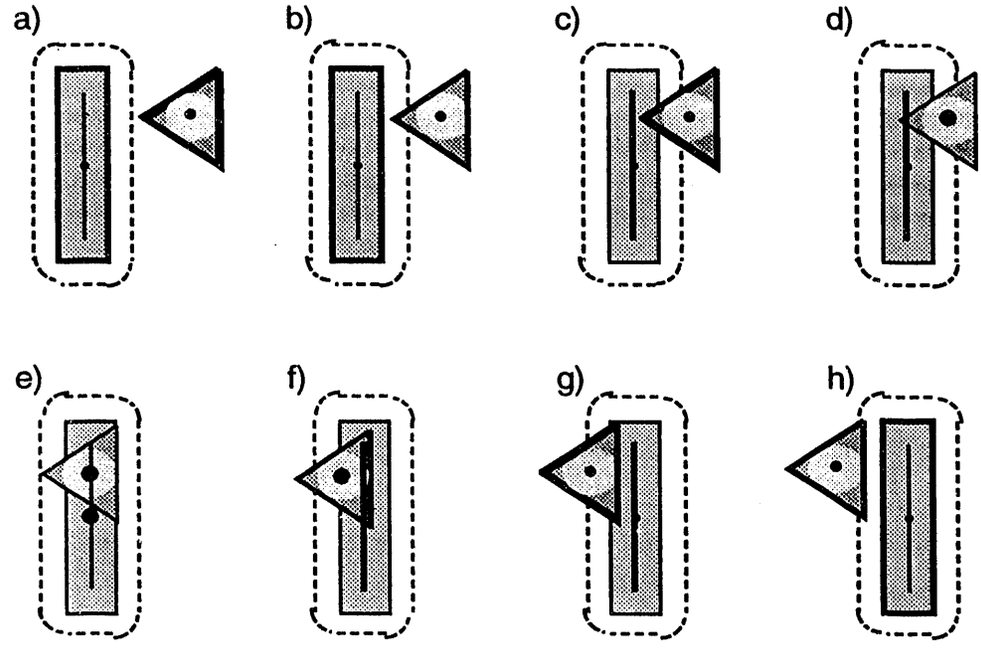

Figure 5: Example showing use of sub-polytopes to determine extrication vectors and distances:a) no collision; b) within delta window; c) Using rectangle line sub-polytope; d) using triangle point sub-polytope; e) Using both point sub-polytopes; f) Using line sub-polytope; g) Using-full triangle polytope; h) Using both full polytopes.

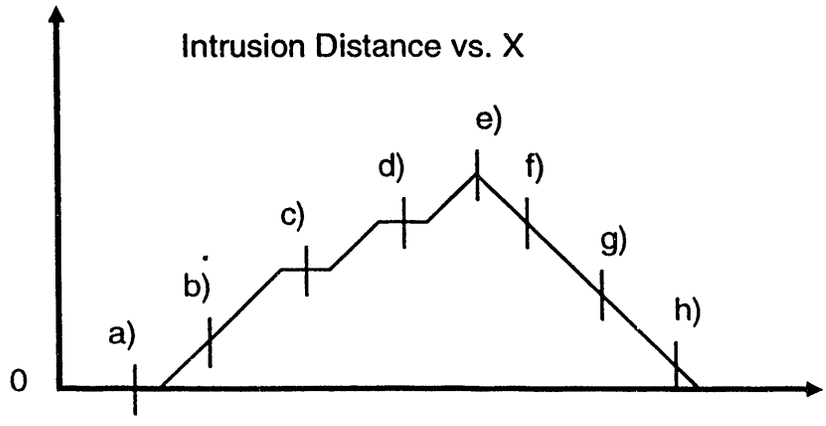

Figure 6: Distance function for rectangle/triangle collision path example of Figure 5.

2.2 Determining the type of contact

Rigid three dimensional convex polytopes can interact in a finite number of ways. Corners can collide with surfaces, edges or other corners. Edges can align in parallel or skewed. A summary of edge contact conditions is given in [8, 9], and numerous examples are shown in Figure 7.

Gilbert's algorithm does not distinguish between the type of contact, it just returns a unique nearest distance, and a not necessarily unique direction vector and interaction point. Thus instantaneously there is no way to distinguish between point-to-point, point-to-edge, edge-to-edge, point-to-surface, edge-to-surface and surface-to-surface contacts. Furthermore, if surfaces and edges are parallel the algorithm typically returns a vertex rather than a midpoint.

This originally caused concern since we thought an interaction point at the middle of a parallel surface made the most sense for representing parallel contact. Later we conjectured that the instantaneous determination of contact type and the exact location of nearest contact vectors was inconsequential. What mattered was how the system reacts to instantaneous rotations. If small shifts in orientation are opposed in two directions there is surface contact, if opposed in one there is edge contact, if opposed in no directions there is point contact. In essence, the type of contact did not need to be determined instantaneously, but could be determined over time by gauging the net effect of reaction forces.

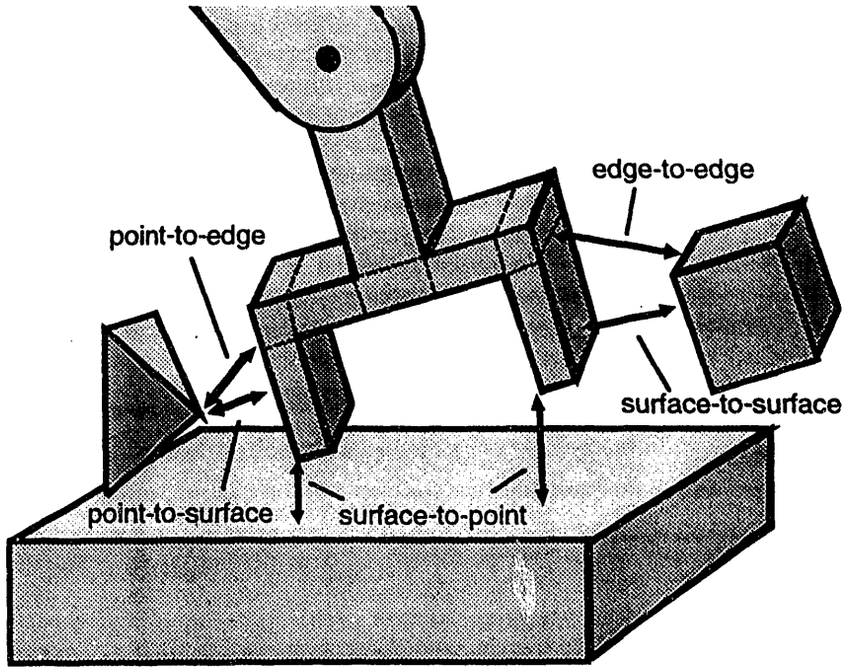

Figure 7: Surface Contact types.

Consider the parallel plate interaction shown in Figure 8. The distance algorithm might return any of the four points (A, B, C or D) as being the nearest point. The algorithm presented in this paper will generate a virtual force at this point, which due to the impedance algorithm implemented for the manipulator will result in that vertex rotating away from the surface. This will cause another vertex to become the "nearest" vertex, and the algorithm will next rotate that corner away. This will continue indefinitely. The extrusion vector will race all across the surface of the tool opposing any infinitesimal infraction, and the plates should remain in parallel

The question to be answered through experimental work was would it work? Could an algorithm which utilized rapid switching of contact points achieve smooth tool-environment interactions through some type of averaging phenomena, or would objects chatter wildly whenever brought near parallel? Could the algorithm be computed fast enough using commercial hardware to make it viable for building virtual force fields in a telerobotic system? Was determination of contact type unneccesary for developing obstacle avoidance algorithms?

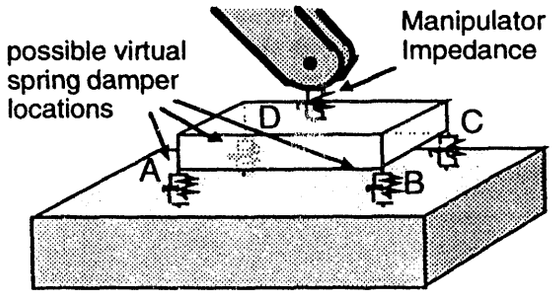

Figure 8: Parallel plate interaction.

3. ALGORITHM SUMMARY

Before we describe the results of the experiments, we summarize the algorithm used for generating virtual forces below:

Virtual Force Algorithm

I. Decompose tool and the environment into overlapping convex object primitives.

A. Divide environment and tool into convex objects, using maximal overlap.

B. Approximate convex objects by polytopes.

C. For each convex object primitive determine a list of embedded sub-polytopes ending in a point.

II. Determine object interactions.

A. Use bounding boxes to eliminate object pairs which are definitely outside a delta region.

B. Apply Gilbert algorithm (using the last computed distance vectors as a starting point) to all remaining pairs of objects.

C. For any objects pairs found to be in collision use the reduced polyhedra until an intrusion distance, an interaction point, and an extrusion vector can be determined.

III. Determine force on tool.

A. For each pair within a delta distance, compute the force (f_i) as a function of intrusion distance based on non-linear spring/damper (spring constant changes from 0 at object interaction boundary to maximum value at contact point).

B. Apply repulsive force element along the extrusion vector at the interaction point.

C. Determine net force (f_{TOOL}) and moment (N_{TOOL}) on tool by summing up repulsive forces (f_i), and computing sum of cross products of the distances to the point of nearest contact (d_i) .

$$f_{TOOL} = \sum_{\substack{\text{Pairs in} \\ \text{proximity}}} f_i \qquad N_{TOOL} = \sum_{\substack{\text{Pairs in} \\ \text{proximity}}} d_i \times f_i$$

IV. Compute motion of manipulator based on impedance law.

A. Add force from world model interaction to forces from other inputs and sensors.

B. Compute delta change of position based on local dynamics and impedance parameters.

C. Go back to II.

4. EXPERIMENTAL SETUP

The algorithm was implemented in C using Gilbert's code as a starting point. Gilbert's algorithm utilizes varying amounts of iteration, requiring irregular amounts of computation time. For instance it typically takes more time to compute extraction vectors for objects near parallel. Because each iteration of the algorithm had to be computed in a finite time step the following approach was taken. Every time stop the Gilbert algorithm was run on as many convex object pairs as could be computed. If all pairs couldn't be computed in the time step then old vector/distance data would be used for that pair during that time sample.

The module representing obstacle avoidance was incorporated into the SMART architecture as the OBSTACLE module using a multi-processor VME-Bus environment running under VxWorks. The OBSTACLE module itself was run on a Mercury MC860 Intel I860 based attached processor. Update rates of 400 Hz were achieved for a tool consisting of 4 convex objects, and an environment consisting of 8 convex objects.

The SMART module connection diagram for this experiment is shown below (Fig. 5), and uses the following modules: A torque arm for 1 DOF force reflection, a MULTIPLEX module for coupling a 1 DOF device to 6 DOF and scaling forces and velocities, a SPACEBALL module for enabling 6 DOF unilateral motion from a Dimension 6 Force Ball, a RELIEF module to enable compliance along arbitrary DOF, a KBB2 module to provide filtering and reduce wave reflections, the OBSTACLE module implementing the algorithm described here, a PUMA_KIN module for mapping the PUMA's kinematics from world space to joint space, a LIMITS module for imposing joint limit restrictions, a VISUAL module for displaying the system in a 3-D modeling environment (Deneb's IGRIP) and a PUMA_JOINTS module to connect to the PUMA 560 hardware.

The system was run using the two input devices and with various compliance modes. The simulated gripper was driven into environment objects at various angles. At all times the resulting motion of the gripper was smooth and intuitive. No chattering was ever observed for any type of contact. In addition the virtual forces provided more than obstacle avoidance capabilities, they

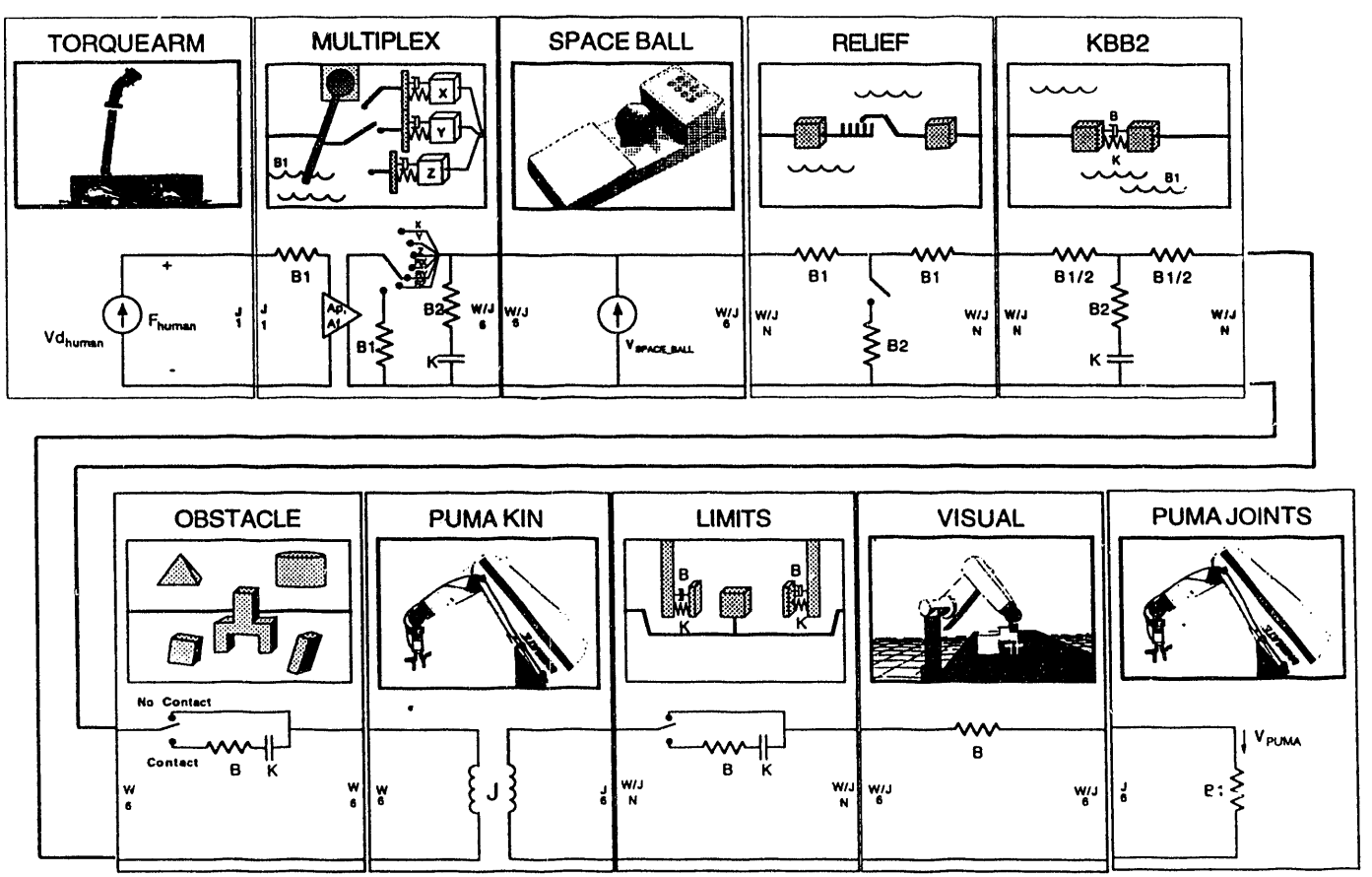

Fig. 9: SMART Modules used for experimental validation.

generated constraint forces which could be utilized to achieve task objectives. For example, tools could be aligned with virtual surfaces and slid into virtual corners.

4. Conclusions.

The experiments described in this work demonstrated that it is possible to use a very simple algorithm with no knowledge of interaction types, to achieve smooth, intuitive, object interactions for a complex gripper tool working in a complex environment. Although requiring substantial computation every sample instant, the computation can be achieved using relatively inexpensive off the shelf computer hardware.

The SMART module deriving from this work (i.e., the OBSTACLE module) is now a core module of the SMART architecture and virtual forces generated from this module can be used in addition to the existing sensor feedback and simulated dynamics, to control any type of robot. In particular, it is being applied to obstacle avoidance in our underground storage tank laboratory and for task planning in our waste processing operations lab, and is providing virtual force information for a kinesthetic virtual reality laboratory.

Currently, we are extending the virtual force concept to the entire arm, rather than just the tool. In the case of the PUMA this requires modelling the arm and forearm as two addition rigid body polyhedra moving with respect to the same set of fixed polyhedra in the environment. The net virtual force signal will then be mapped back into joint space using the appropriate Jacobians to achieve the desired behavior.

5. References.

1. R. J. Anderson, "SMART: A Modular Architecture for Robotics and Teleoperation",*Proceedings of the Fourth ISRAM*, 1992, Santa Fe, NM, pp 467-474.

2. W. S. Kim, "Developments of New Force Reflecting Control Schemes and an Application to a Teleoperation Training Simulator," *1992 IEEE Int. Conf. on R&A*, pp 261-266, Nice.

3. O., Khatib, , "Real-time Obstacle Avoidance for Manipulators and Mobile Robots", *The International Journal of Robotics Research*, Vol 5, No 1, Spring 1986.

4. T. Wikman, and W. S. Newman, "A Fast On-line Collision Avoidance Method for a Kinematically Redundant Manipulator Based on Reflex Control", *1992 IEEE Int. Conf. on R&A*, pp 1412-1419, Nice.

5. J. E. Bobrow, "A Direct Minimization Approach for Obtaining the Distance Convex Polyhedra", *The International Journal of Robotics Research*, Vol. 8, No. 3, 1989, 65-76.

6. M. C. Lin, and Canny J. F., "A Fast Algorithm for Incremental Distance Calculation", *Proc. IEEE Int. Conf. on Robotics and Automation*, 1991, pp. 1008-1014.

7. E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space," *IEEE Journal of R&A*, vol. 4, no. 2, pp193-203, April 1988.

8. S. Hirai and K. Iwata, "Recognition of Contact State Based on Geometric Model", *1992 IEEE Int. Conf. on R&A*, pp 1507-1518, Nice.

9. M.T. Mason, "Compliance and Force Control for Computer Controlled Manipulators", *IEEE Trans on Systems, Man, and Cybernetics*, vol. 11, pp. 418-432, 1981.

END

DATE FILMED

10/22/93