April 1990

Teleoperation in the Presence of Communication Delays

Janez Funda
University of Pennsylvania

Thierry Simeon
Laboratorie d'Automatique et d'Analyse des Systemes (LAAS)

Richard P. Paul
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-90-27.

Teleoperation in the Presence of Communication Delays

Abstract

Modern industrial processes, public service needs, and research interests have established a clear need to perform work remotely [12][4]. Teleoperators were developed with the advent of nuclear industry in the mid 1940's and have been since used extensively to perform work in hazardous environments (nuclear, chemical), undersea (resource exploration, waste management, pollution monitoring), and in the outer space (sample acquisition, satellite deployment/repair). Sophisticated systems have been designed and built to meet these needs, providing the human operator with high bandwidth and high fidelity visual and kinesthetic feedback information about the task in progress [22] [32] [16] [6].

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-90-27.

Teleoperation in the Presence of Communication Delays

MS-CIS-90-27
GRASP LAB 213

Janez Funda
Thierry Simeon
Richard P. Paul

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104

April 1990

Teleoperation in the Presence of Communication Delays

Janez Funda †

Thierry Simeon ‡

Richard P. Paul †

† Grasp Laboratory, University of Pennsylvania, Philadelphia, U.S.A.

‡ Laboratorie d'Automatique et d'Analyse des Systemes (LAAS), Toulouse, FRANCE

Contents

1 Introduction 1

2 Outline of the proposed solution 3

2.1 Modeling the environment 3

2.2 Controlling the motion of the slave 4

2.3 Generating kinesthetic feedback 5

2.4 Aiding the operator 5

2.5 Generating remote slave motion commands 6

2.6 Using task information 7

2.7 Error handling and model consistency 8

2.8 Applications 9

3 The graphical simulator 10

3.1 The model 10

3.2 The simulation technique 10

4 Distance computation and collision detection 12

4.1 Distance computation 13

4.2 Collision avoidance 15

4.3 Contact type determination 15

4.4 Constraint information 16

5 Operator's motion analysis 18

5.1 Classification of allowable motions 18

5.2 Free space motion 19

5.3 Contact motion 20

5.3.1 Types of contact 20

5.3.2 Constraint normals 22

5.3.3 Kinesthetic feedback 23

5.3.4 Types of contact motions — overview 23

5.3.5 'Freeze' mode 24

5.3.6 'Slide' mode — single contact 25

5.3.7 'Slide' mode — multiple contacts 26

5.3.8 'Pivot mode — single contact 28
5.3.9 'Pivot mode — multiple contacts 33
5.4 Pushing 36
5.4.1 Single-contact pushing 36
5.4.2 Multiple-contact pushing 38

6 Filtering of operator's motions **38**

7 Generation of symbolic slave commands **41**
7.1 Types of motions 42
7.2 Task frame specification 43
7.3 Motions to keep contact 43
7.4 Motions to change contact 45
7.4.1 Sliding case 45
7.4.2 Pivoting case 45

8 The experimental hardware/software testbed **48**

9 Preliminary results and discussion **51**

A Appendix **i**
A.1 Notation i
A.2 Coordinate frames and rotational matrices i
A.3 Mapping rotations between frames ii
A.4 Displacement of a point due to motion of the frame ii

List of Figures

1 Interpretation of the force/torque sensor readings. 4
2 Overview of the proposed solution. 11
3 Types of polyhedral contacts. 21
4 Constraint normals for the three types of polyhedral features. 23
5 Single-contact sliding. 25
6 Multiple-contact sliding. 26
7 Tangential and contact frames. 29
8 Computing $\Delta\theta_y$ of the contact frame. 30
9 Single-contact pivoting — line contact. 32
10 Single-contact pivoting — plane contact. 33
11 Multiple-contact pivoting. 34
12 Single-contact pushing. 37
13 Trajectory filter — the "closeness test". 40
14 Changes of contact during a sliding motion. 45
15 Transition between two *vertex/face* contacts. 46
16 The hardware architecture of the experimental testbed. 48
17 The operator's station. 49

1 Introduction

Modern industrial processes, public service needs, and research interests have established a clear need to perform work remotely [12][4]. Teleoperators were developed with the advent of nuclear industry in the mid 1940's and have been since used extensively to perform work in hazardous environments (nuclear, chemical), undersea (resource exploration, waste management, pollution monitoring), and in the outer space (sample acquisition, satellite deployment/repair). Sophisticated systems have been designed and built to meet these needs, providing the human operator with high bandwidth and high fidelity visual and kinesthetic feedback information about the task in progress [22] [32] [16] [6].

A problem occurs, however, when teleoperation is attempted in a situation where the response from the slave manipulator is delayed (with respect to the command issued from the master) due to a large physical separation and/or insufficient communication link capacity [8][10]. Given that neurological control of normal human movements operates at approximately 5 Hz [34], communication delays between the master and the slave in excess of one second greatly impair the operator's ability to perform work [8][13]. It has been shown, in fact, that supplying the operator with delayed force feedback can be counter-productive, as it does not correspond to the current situation and thus does not provide the feedback that the operator expects. This problem is particularly severe when the operator wishes to be in contact with the environment and expects to rely on the kinesthetic feedback information to adjust her motions.

Overcoming communication delays has been recognized as one of the central areas of research in telerobotics for some time [34]. Among the proposed approaches to solve the problem are

- slowing down the motion so as to minimize the effect of the delay [9]
- adopting a "move-and-wait" strategy whereby the operator performs a small incremental motion and then waits for the delayed remote-site feedback to confirm the success of the motion before proceeding [9]
- strengthening the slave arm and the objects which it manipulates in order to avoid damage (*e.g.*, underwater "remotely operated vehicles", ROV's, for

off-shore oil exploration)

- formally modelling up-link and down-link delays by augmenting the dynamic state-space model of the system (environment + slave) – delays are modelled as delay lines on the output and introduce (a potentially large number of) additional states [16]

- using predictive visual (graphical) displays to allow the operator to "be ahead" of the slave [25][2]

- relying on autonomous, sensor-driven, preprogrammed contact motion primitives at the slave site [32]

None of the above approaches by itself, however, has proven to be entirely satisfactory. Of course, a totally autonomous manipulative capability would solve the problem, but its realization is beyond the state of the art in robotics.

We believe that a predictive visual display of the remote environment is an essential component of a successful man-machine system for teleoperation in the presence of delays. However, we also recognize the necessity of providing the operator with some sense of real-time "kinesthetic feel" for slave-environment interactions. We thus propose to combine a graphical simulation of the slave world with real-time *estimated* kinesthetic feedback which is computed on the basis of a geometrical analysis of polyhedral object interactions in the simulated remote world. We then propose to analyze (on-line) the observed (operator supplied) motion trajectories of the simulated slave and extract a stream of task-oriented motion primitives (*e.g.*, guarded and compliant moves in the *task space*) to be sent to the slave. These elementary motions are then executed at the slave site under the supervision of the local sensory and control modules and the status of their execution is reported back to the operator's station.

Therefore, the essential paradigm of the proposed solution is *teleprogramming* the remote slave, as illustrated in Figure 2. Section 2 below introduces the essential modules of the proposed conceptual architecture, whereas Sections 3—7 address separate components in more detail. The graphical simulation model and simulation technique are described in Section 3. Section 4 offers some detail on the manner in which distances between objects in the simulated (slave) environment are monitored

and collisions, as well as contact types, are identified. In Section 5 we introduce the envisioned operator-machine interface and describe the manner in which we compute a real-time approximation to the (delayed) actual kinesthetic feedback. Section 6 proposes a simple 6-d.o.f. filter to smooth the operator supplied motion trajectories, whereas Section 7 describes the methodology for partitioning the task in progress and generating symbolic command strings for the remote slave. Section 8 describes the hardware and software experimental testbed which is being used to test these ideas. Finally, Section 9 presents a discussion of some preliminary results and future directions.

2 Outline of the proposed solution

2.1 Modeling the environment

We will assume in this work that we are manipulating in an apriori unknown environment. The initial description of the environment is obtained through the use of sensors, such as vision or dense range data, producing a high-level three-dimensional scene description consisting of object features such as planes, edges, vertices, *etc.* The process of extracting this information is within the state of the art of computer vision [3][15][33]. Moreover, it has been demonstrated that such descriptions can be converted to polyhedral CAD-type models [14][25]. We propose to display such a CAD image of the environment (including the slave manipulator) and interface a 6 d.o.f. input device (master) to the simulator, such that the images of the slave manipulator and any objects that it may be manipulating could be moved under the control of the operator.

Because the environment is assumed unstructured and we must rely on an idealized and simplified approximation of the actual environment, we can not predict all work situations (due to model incompleteness), nor can we predict the outcome of a particular action exactly (due to model inaccuracy). Therefore, we are unable to construct detailed, robust and reliable plans of action ahead of time. Instead, we propose to keep the operator in the control loop at all times, and let her define the plan incrementally as she interactively programs the slave robot actions by moving the master.

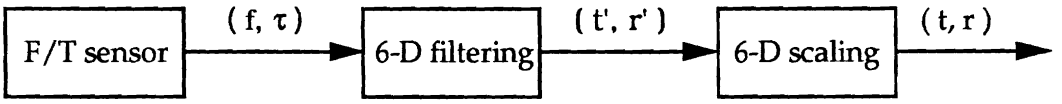

Figure 1: Interpretation of the force/torque sensor readings.

2.2 Controlling the motion of the slave

Incremental positional/orientational information can be specified to the (simulated) slave manipulator in a variety of ways. We propose the use of a general 6 d.o.f. force/torque sensor mounted at the tip of the master manipulator, whose force and torque readings are (through a series of filters and amplifiers) interpreted as positional and orientational information, respectively (Figure 1). The pair $(\mathbf{f}, \tau)$ is the 6-vector of raw forces and torques as read from the sensor. This information is then filtered/smoothed and appropriately scaled to become the positional/orientational displacement of the master (operator's hand). The rotation $\mathbf{r}$ is interpreted as roll/pitch/yaw (RPY) parameters.

The so obtained incremental displacement[1]

$$\Delta \mathbf{d} = (\mathbf{t}, \mathbf{r}) \tag{1}$$

is interpreted as *master handle* (sensor-based frame) displacement. The motion of the master manipulator is then computed by mapping this handle displacement into the master's end-effector frame ($T6_m$) and using it as an incremental Cartesian positional displacement with respect to the end-effector frame.

The motion of the slave simulator is coupled to the motion of the master by establishing a *correspondence of motion* between the *master's handle frame* and the *slave's end-effector frame* ($T6_s$). In general, due to the fact that the master and slave manipulators will be kinematically dissimilar (and will therefore have different workspace volumes), this correspondence will not be a straight-forward one-to-one positional/orientational equivalence of motion. Instead, another level of scaling for translational motions will be needed to account for the workspace volume differences.

[1]We use the term *incremental displacement* instead of *differential displacement*, since we deal with discrete rather than instantaneous changes in displacement.

2.3 Generating kinesthetic feedback

Having obtained an initial graphical description of the world, we then monitor the position of the slave arm (and any object it may be carrying) for contacts with the environment. This collision checking must be performed in real time and is used to prevent interpenetration of colliding objects (their motion is stopped on contact), thus modifying the intended motion of the (simulated) slave manipulator. In order for the system to feel natural to the operator, the positional/orientational correspondence between the master device and the slave must be preserved at all times, including on contact with the environment as well as while one or more contacts persist. We therefore need an input device, that is itself movable in space and backdrivable, such as a specially designed teleoperator master arm or a backdrivable general purpose robot manipulator. Such a device enables us to not only specify the desired positional/orientational displacement to the slave arm, but also gives the operator a sense of three-dimensional manipulation as it follows the operator's hand through space. More importantly, however, backdriving the master arm to correspond to the state of the simulated slave arm provides the operator with the ability to explicitly feel the constrained d.o.f. of the motion of the slave (and thus master) and therefore allows the operator to kinesthetically "feel" contacts between objects, examine shapes of objects, follow their contours, *etc.* This capability of combining graphical object interference detection with backdriving the master device represents a crucial feature of the proposed system. It provides the operator with a strong sense of *telepresence* (*i.e.*, a simulated sense of force reflection in real time), despite the communication delays, which cause the actual feedback to be delayed and therefore not usable for direct reflection to the operator. This is significant since kinesthetic feedback has been shown to be essential in any teleoperation activity [12].

2.4 Aiding the operator

The operator can now move the slave manipulator in the simulated world, come into contact with the environment and "feel" in a very natural way any constraints that the geometry of the task world may be imposing onto pthe motion of the slave. Moreover, we propose that the system provide a set of elementary classes of motion,

which are natural, convenient and easy to perform, yet powerful enough to allow the operator sufficient flexibility in performing tasks. This is particularly cruicial during contact motion, when the operator may wish to concentrate on a certain subset of motion parameters (*e.g.*, sliding, reorienting), and be aided by the system in keeping other parameters constant. The system can also assist the operator by biasing the interpretation of her motions towards preserving achieved contacts (for instance, to aid in feature tracking), while still allowing arbitrary changes of or departures from the current contact state. We will address these issues in more detail in Section 5.

2.5 Generating remote slave motion commands

We next attempt to interpret the positional and force information accumulated by the simulator to extract a stream of elementary motion commands that are to be commanded to the slave robot. In view of this, we first filter the gathered information and eliminate the (presumably unintended) noise in the data. We then analyze this filtered information of positional/orientational parameters, contact state changes, and forces/torques exerted onto the environment to produce a sequence of symbolic instructions to the slave. Again, as our model of the slave world is only approximate, the nature of these instructions must reflect and accommodate possible discrepancies between the model and the actual world. While this is not critical during free space motion, it is vitally important when attempting to establish or maintain contact with the environment. Consequently, for the case of contact motion, we propose to generate instructions of the type "move along a given direction until contact" (guarded motion), or "move along a given feature while maintaining contact in some direction" (compliant motion). There will be also a class of motions (such as tight tolerance part mating, fine precision adjusting motions) which may be difficult to perform using an incomplete model and approximate kinesthetic feedback. Such motions are therefore best executed by the slave autonomously, under local sensor supervision and local high-bandwidth feedback processes. We will have more to say about symbolic command string generation in Section 7.

2.6 Using task information

The process of interpreting the operator's actions in the simulated world can be a difficult one in the absence of any other information about the nature of the task in progress. For instance, a sequence of rapid contact changes may be interpreted either as noisy data or a purposeful action, such as tapping, scraping, or rocking. Similarly, a highly irregular path of an object during a sliding motion could be taken as unintended (and therefore would be filtered out or smoothed) or it could correspond to a motion such as polishing or sanding (in which case it should be kept intact). In order to disambiguate between such interpretations, the system needs additional information about the task, such as a description of the type of expected primitive motions (*e.g.*, pick and place, polishing, pounding). Moreover, the graphical simulator should be supplied with some information as to which objects are expected to come into contact during a given task to avoid having to monitor every pair of objects for a possible collision.

These are but a few examples of why high-level task information may be essential for correct interpretation of operator's intent and efficient internal computations. We feel that the design of the structure, organization, and content of such a task-level database is a significant research problem in itself. Consequently, we may not be able to address this aspect of the proposal fully in the preliminary stages of the project. However, we envision the task related information being gathered in the following manner

- by loading and using a preexisting task database
- by querrying the user (operator) prior to the manipulation to extract the essential features of the task to be performed
- by maintaining an on-line dialogue with the operator to allow her to augment and modify the current task information while the task is in progress, as well as to allow the command stream generator to request additional information from the operator when her intent is still unclear

This would allow on-line refinement of the task description and should greatly expand the repertoire of tasks that the system could interpret correctly and thus issue appropriate motion commands to the remote slave.

2.7 Error handling and model consistency

We now have a system, where a human operator can essentially *teleprogram* a remote slave robot, overcoming the communication delay problem by using real-time simulated visual and kinesthetic feedback. Of course, while all is well in the simulated world, various things may go wrong in the actual work environment. The slave can detect such error conditions by not reaching an expected motion-terminating condition, by hitting an obstacle, by seeing excessive or premature motor torques, *etc.* Upon detecting such a condition, the slave can signal the occurrence of an error state to the operator's station, which in turn can alert the user through a variety of visual or audio means (*e.g.*, flashing the display, synthesized voice warnings, *etc.*). It is then up to the operator to plan corrective actions. First, the operator's station based model of the world must be updated to properly reflect the current situation. This can be done through gathering and reconciling information from a variety of remote site based sensors (*e.g.*, video cameras, range finders, *etc.*) and/or purposeful exploratory motions on the part of the operator (if this is possible) to find or correct certain model parameters. Then, the operator can attempt to correct the problem and proceed with the task.

It is important to note that discrepancies between the model and the world can also arise due to effects of external environmental agents, *i.e.*, other than slave's actions. Such changes may not be discovered through the actions of the slave, but may cause problems at a later stage in the manipulation. What is needed, therefore, is a rather sophisticated *environment updating mechanism*, which continuously (in reasonable intervals) checks at least the local portions of the environment model (*i.e.*, in the immediate work area), but can also be brought into action by request from the operator under her control. The latter facility is important not only for situations when the slave has entered an error state, but also when the operator wishes to verify poorly recovered or uncertain features of the workspace.

We believe that the problem of ensuring consistency between the model and the world is a very critical one to successful operation of the proposed system and again represents a challenging research topic in its own right. We will in this work restrict ourselves to some general comments on how this problem may be solved and will not attempt to provide a detailed solution.

Another feature of the system, thus, is that by keeping the operator in the control loop, she can take on the responsibility of handling error conditions. This is significant, since anticipating various possible error states and planning for their recovery by introducing various exception handling routines plagues conventional robot programming. Clearly, not all possible error states can be anticipated, especially in a situation where the environment is unstructured and we only have an approximate model of it. Moreover, programming exception handlers can easily become a self-defeating enterprise as corrective actions for every error may themselves involve errors. This system therefore eliminates the need to write elaborate robot manipulator programs taking on the impossible task of accounting for all possible errors.

2.8 Applications

We believe that a system, such as the one outlined above, will facilitate teleoperation with time delay allowing a very natural interaction between the operator and an image of the task involving both visual and kinesthetic feedback. The system will also allow for considerable time delays limited only by the extent that the operator is allowed to move ahead of actual execution.

Application of such technology to undersea manipulation would free us from the need to maintain wide bandwidth communications between an operator and the vehicle. While it appears possible to eliminate vehicle tethers based on energy considerations [24], it is still impossible to eliminate the tether based on manipulation control considerations due to the delays in bringing acoustic signals to the surface. Operators must either be in the vehicle or in a surface ship at the end of a tether. With the proposed technology it would be possible to drop a submersible from a plane together with an acoustic relay buoy and then to control operations at the ocean bottom remotely over a radio link from either the plane or the shore. The principal cost saving is, of course, the elimination of the need for a surface ship maintaining station during the entire underwater operation. Secondary cost savings relate to the elimination of the tether and the possibility of working in environments in which the tether might become tangled, as well as the possibility of using more than one submersible in the same working area when the control of tethers becomes

impossible.

Cost justification for work in shallow space relate to the possibility of eliminating the need for an astronaut on EVA to perform the task, vastly reducing the cost involved.

3 The graphical simulator

3.1 The model

We propose to adopt a polyhedral, boundary-representation based graphical model of the world. While other representations are clearly possible (*e.g.*, CSG, generalized cylinders), polyhedral models are widely used and consequently a variety of algorithms exist for polyhedral analysis. Perhaps the most important advantage, however, is the convenience of polyhedral models for contact analysis, which is a central requirement and feature of this work.

An important component of the graphical simulator is an exact kinematic model of the slave manipulator (and any attached equipment). This simulated slave robot must accurately reflect the kinematic limitations of the actual slave (*i.e.*, joint range limitations) and the simulator software must ensure this. Moreover, there should be no need for the slave and the master manipulator to bear any structural or kinematic resemblance to each other. While this significantly complicates the control of the system (space transformations, two sets of singular configurations, reindexing), it is an important feature of a general purpose teleprogramming software system.

3.2 The simulation technique

A key decision in this work has been to use a *kinematic* simulation of the motion of the slave and the manipulated objects. The simulation therefore does not account for the dynamic effects of either the slave robot or the environment. Moreover, the slave (plus any held object) are the only moving parts in the environment during each simulation time slice. Consequently, dynamic changes in the environment, other than the slave's state, must be related to the operator's station through the environment updating mechanism (Section 2.7), rather than direct simulation. This applies to the dynamic changes caused by the slave (*i.e.*, dropping or tipping an

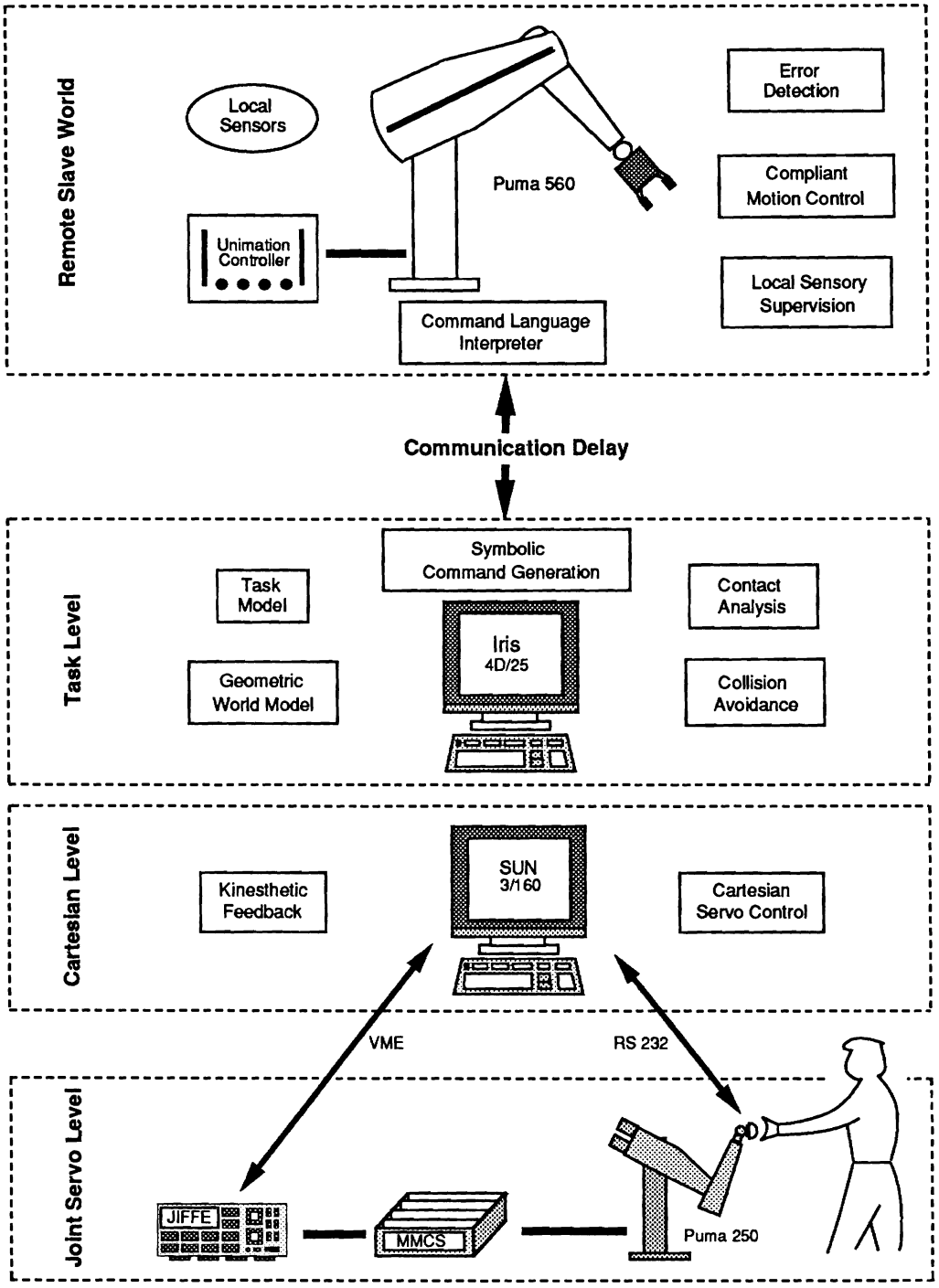

Figure 2: Overview of the proposed solution.

object), as well as those produced by external environmental agents (*i.e.*, winds, water currents). While the choice of a kinematic simulation may seem restrictive, we feel that it is the most practical approach for the following reasons

- since only approximate information about the world is available, we can not expect to have complete information about the masses, centers of mass, inertias, frictional parameters, *etc.* about the objects in the environment; yet, these are essential parameters for a dynamic simulation
- in many environments and situations a rigid-body dynamic model may not be adequate; we may be manipulating on a soft ocean bottom, or we may have erroneous confidence in the hardness of the objects in the slave world
- a dynamic simulation of both the robot and the environment represents a significant computational burden; in all but the simplest cases it in fact may not be computable in real time
- due to model uncertainty, only rough predictions based on dynamic computations are possible; such approximate, unreliable results do not justify the time spent in computation
- unmodelable and unpredictable external agents (water turbulence, buoyancy effects) may contribute to the dynamic state of the world, further diminishing the utility of a costly dynamic simulation

Clearly, a kinematic simulation leaves much to be desired, but under the circumstances we feel that it is a more reasonable and more practical choice than a full dynamic simulation of both the slave manipulator and the environment.

4 Distance computation and collision detection

The kinesthetic feedback described in Section 5 relies heavily on the detection and analysis of the contacts which arise during the motion of the slave in the simulated environment. Expected contacts will normally occur between the slave's end-effector, tool, or an object it is currently holding, and some part of the slave world involved in the execution of the task. We will hereafter refer to the former as the

movable object and will abbreviate it as MO. Moreover, the graphical simulator must also provide an aid to the operator by checking that undesired collisions between the slave arm and the environment do not occur during the motion.

Both cases can be solved by monitoring the distances between pairs of objects. While the former requires precise models of the objects, simpler, approximate, yet conservative models suffice for the latter. Simplified models are preferred, whenever possible, in order to limit the computational cost of the collision checking module.

During the execution of a task, may pairs of objects may need to be monitored for contact at each step of the simulation. Consequently, there is a definite need for an efficient distance computation algorithm.

4.1 Distance computation

Several methods exist to compute distance[2] between polyhedral objects. Because of its efficiency we chose to implement the distance algorithm between convex sets of points described in [11]. The aim of this section is to summarize the main features of this algorithm. For a more detailed description, the reader is referred to [11].

Let A and B denote the two polyhedral objects, whose distance (from each other) we are seeking. For the purposes of the algorithm the two objects need to be represented simply as the respective sets of vertices $S(A)$ and $S(B)$. The algorithm uses the following property of distance between the two sets

$$dist(A, B) = dist(\phi, C) \tag{2}$$

where ϕ denotes the origin of the space and $C = B \ominus A$ represents *Minkowsky's difference* between the sets A and B. Instead of first computing C[3], the algorithm is based on an iterative procedure which generates sequences of elementary sets C_k containing 1 to 4 vertices of $S(C)$. These C_k are such that their distance to the origin converges to the desired distance between the objects A and B.

An efficient procedure is used to compute the closest point $\mathbf{u}_k$ of the convex hull of these simple sets of points C_k (line segments, triangular faces, tetrahedrons) to

[2] *Distance* between two objects is defined as the smallest translation which will put them into contact.

[3] If A and B have n_A and n_B vertices, respectively, then C can have up to $n_A \cdot n_B$ vertices.

the origin of the space. $\mathbf{u}_k$ is obtained from the computation of the coefficients λ_i of the set's barycentric representation, *i.e.*,

$$\mathbf{u}_k = \sum \lambda_i \cdot \mathbf{x}_i \qquad \text{with } \lambda_i \geq 0 \ , \ \sum \lambda_i = 1, \text{ and } \mathbf{x}_i \in S(C_k) \tag{3}$$

The points $\mathbf{x}_i$ of C_k whose $\lambda_i > 0$ define a $C_k^* \subset C_k$ containing $\mathbf{u}_k$ (for example, if C_k is a triangular face defined by three vertices, then C_k^* can be either one of the three line segments, or one of the three vertices of the face, depending of the number of positive λ_i computed). The sequence of $\mathbf{u}_k$ generated is such that $\|\mathbf{u}_{k+1}\| \leq \|\mathbf{u}_k\|$ and the norms converge to $dist(A, B)$.

The generation of the next C_{k+1} from the current C_k and $\mathbf{u}_k$ is based on the notion of a support function. The support function of a set of points X is defined by[4] $h_X(\mathbf{n}) = \max_{\mathbf{x}_i \in S(X)} \{\mathbf{n} \cdot \mathbf{x}_i\}$ and we will use $s_X(\mathbf{n})$ to denote one of the $\mathbf{x}_i$ which verifies this maximum.

It is shown in [11] that if $\|\mathbf{u}_k\| + h_C(-\mathbf{u}_k) = 0$, then $dist(A, B) = \|\mathbf{u}_k\|$. Otherwise, the C_{k+1} to be checked at the next iteration is obtained from the set of vertices $S(C_k^*) \cup \{s_C(-\mathbf{u}_k)\}$. The interest of using this support function for the generation of the vertices of C comes from the fact that $s_C(\mathbf{n})$ and $h_C(\mathbf{n})$ can both be computed in $O(n_A + n_B)$ time, *i.e.*,

$$\begin{aligned} h_C(\mathbf{n}) &= h_B(\mathbf{n}) + h_A(\mathbf{n}) \\ s_C(\mathbf{n}) &= s_B(\mathbf{n}) - s_A(\mathbf{n}) \end{aligned} \tag{4}$$

Each iteration is therefore performed in linear time in the total number of vertices and as only a few iterations are needed for the convergence, the distance algorithm is *quasi-linear* in the total number of vertices.

The overall structure of the algorithm also plays a important role in its efficiency:

- The algorithm relies exclusively on simple computations (dot products and vector additions). Moreover, the procedure used for the computation of $\mathbf{u}_k$ reuses many of the values already computed during the previous step. These values are stored and each iteration needs to perform only a few additional computations.

[4] In fact, this function defines for a given direction $\mathbf{n}$ a plane $\mathbf{x} \cdot \mathbf{n} = h_X(\mathbf{n})$, such that all the points of X lie on the same side of this plane.

- An extra speedup is obtained by providing an initial estimation of $S(C_0)$ to the algorithm. This feature turns out to be particularly interesting when only small positional changes occur between two successive distance computations. In this case, the set $S(C_k)$ computed at the last iteration of the previous distance computation can be used for this initial estimation. While the closest point of C to the origin stays inside the convex hull of this set, only one iteration will be needed to compute the new distance. Whenever changes occur, a couple of iterations will be generally sufficient to update the new sets of points and compute the distance.

4.2 Collision avoidance

Let $\mathbf{x}_A$ and $\mathbf{x}_B$ denote the closest points between two convex objects A and B. Their distance is then given by $d = \|\mathbf{x}_B - \mathbf{x}_A\|$. If an incremental displacement $(\Delta\mathbf{p}, \Delta\mathbf{r})$ is applied to A, it can be shown [7] that the distance variation Δd can be expressed as

$$\Delta d = -\mathbf{n} \cdot \Delta\mathbf{x}_A \tag{5}$$

where $\mathbf{n} = (\mathbf{x}_B - \mathbf{x}_A)/d$ and $\Delta\mathbf{x}_A$ is the positional displacement of the point $\mathbf{x}_A$ due to the displacement $(\Delta\mathbf{p}, \Delta\mathbf{r})$.

Clearly, a positive Δd indicates that the motion causes the objects to be separated further apart. However, even when Δd is negative, there is no danger of collision as long as $|\Delta d| < d$. Otherwise, the *penetration factor* has to be computed and only the corresponding fraction of the offending displacement is applied in order to stop the motion in the (non-penetrating) contact configuration.

In fact, as this distance variation computation is only valid for *strictly convex* sets of points[5], special steps are needed to handle changes of contact types for other convex polyhedral sets.

4.3 Contact type determination

As mentioned before, detailed contact monitoring must be performed between pairs of objects involving the movable object (*i.e.*, slave's end-effector or a manipulated

[5]Strictly convex sets exhibit a continuous tangent along the surface.

object) and the part of the environment with which the slave is in contact. Both objects of a such pair are declared to be in contact while the distance between them remains zero. In this case a postprocessing step (following distance computation) is performed to extract the *features* of the polyhedral models of both objects, which are actually in contact (*i.e.*, $face_i$ of $Object_1$ against $edge_j$ of $Object_2$). It is these features that define the constraint on the motion due to the contact and therefore must be known for the contact analysis (Section 5). Likewise, a *contact feature centroid* (*e.g.*, edge or face center) is associated with each constraint for later reference.

4.4 Constraint information

As already mentioned, two types of collisions can occur in the system – *wanted* and *unwanted* collisions. Wanted collisions are those that the operator intended to achieve and will normally involve a part of the environment and the movable object. Unwanted collisions, on the other hand, are all other collisions. Because the slave (plus the manipulated object, if any) is the only moving object in the environment, these collisions will normally involve a part of the slave robot accidentally coming into contact with some part of the environment (obstacle).

Corresponding to the two types of collisions we will define two lists of object pairs (wanted and unwanted *collision list*). As we saw in Section 2.6, this information must be supplied to the system either by the user or a task description module prior to the execution of the task. At each simulation step, while the task is in progress, the collision detection module then checks both lists for possible new or persistent contacts. In the case of an unwanted collision, the system refuses to perform the offending motion that would cause the collision and alerts the operator by "freezing" the motion of the master arm and any other means necessary to unambiguously communicate the problem to the operator (*e.g.*, sound, altering display, console messages, *etc.*). The operator can then adjust her intended motion to avoid the collision or adopt a different strategy to accomplish the same task. Note that this feature in a sense offers a rudimentary *collision avoidance* facility, where motion adjustment and/or replanning are left to the operator.

In the case of a wanted collision, the system stops the motion short of causing the collision, *i.e.*, the system allows the two objects to come into contact but not inter-

penetrate (see Section 4.2). Moreover, the system extracts the relevant information about the contact. In particular, it records what type of a geometric constraint this contact imposes on the motion of MO and adds this information to the list of already active constraints. This information is then used to *filter* commanded incremental motions to the master (and thus indirectly to the slave), such that the resulting (filtered) motion doers not violate *any* of the currently active constraints on the motion of MO.

A constraint can be defined as a pair of contacting features (*vertex*, *edge*, *face*), along with a set of parameters that uniquely define the geometry of the given constraint. This information will be needed both in the motion filtering process, where it will be used to define a filtering coordinate frame (Section 5), as well as in the command string generation process, where it will be used to define a task frame (Section 7). As we will see, the following three parameters suffice to uniquely describe the geometry of a constraint in all cases (*i.e.*, regardless of the types of contacting features)

- the vector $\mathbf{p}$ connecting the *slave wrist center* (where the commanded motions are applied) and the *contact point* (feature centroid, associated with the constraint)
- the *constraint normal* $\mathbf{n}$ (see Section 5.3.2 for the definition of constraint normal)
- *edge direction* $\mathbf{e}$, if the contact involves an edge

For convenience, all of the above vector quantities are computed *w.r.t.* the common global reference frame $\mathcal{F}_B$. Therefore, a constraint c_i can be encoded as the quintuple

$$c_i = \{f_1\ ,\ f_2\ ,\ {}^B\mathbf{p}\ ,\ {}^B\mathbf{n}\ ,\ {}^B\mathbf{e}\} \tag{6}$$

where f_1 and f_2 belong to the set $\{vertex, edge, face\}$ and correspond to the contact features of MO and the environment, respectively. The list of all (N) currently active constraints can thus be encoded as

$$\mathcal{C} = \bigcup_{i=1}^{N} c_i \tag{7}$$

Depending on what types of motions the system allows and how the filtering process is carried out, not all of the above information may be needed in all cases. Therefore, for reasons of compactness and efficiency, an actual implementation may condense the information contained in $\mathcal{C}$ to optimize run-time performance.

5 Operator's motion analysis

5.1 Classification of allowable motions

The system operates under the premise that the operator is trying to perform useful work and that her actions are therefore directed and purposeful. Because most useful work is performed while the slave manipulator is in contact with the environment, a teleoperation system must provide a sufficiently wide range of motions both in free space (while approaching/leaving the work area) and in contact with the surroundings (while performing the work). At the same time the allowed motions should be carefully partitioned and restricted to aid the operator in performing the type of motion intended, as well as aid the subsequent automatic analysis (filtering/interpretation) of operator's motions in view of extracting the corresponding symbolic (slave) robot instructions.

A natural way to simplify general motion (both for the operator and for the system) is to separate rotations and translations whenever possible. This is particularly cruicial in contact motion, where the contact point is physically removed from the wrist center, where motion is commanded. This separation gives rise to a remote compliance center and consequently introduces complex and potentially confusing coupling between rotational and translational parameters of the wrist and contact frames. The choice of elementary motions should strive to eliminate such coupling effects without compromising the flexibility and power of the system.

Another important consideration in deciding on the most convenient and effective set of motion modes is the class of tasks that the system is expected to handle. In view of the intended applications of our system (Section 2.8), the operator will need to be able to perform a relatively wide range of tasks. Representative examples are : accurate free-space motion, standard pick and place operations, basic exploratory procedures (*i.e.*, surface or feature following), simple assembly/disassembly tasks,

etc.

Therefore, in view of the above considerations, we propose the following set of elementary classes of motions

1. **Free Space Motion**
    - **general motion** (both rotations and translations)
    - **freeze position** (rotations + fixed position)
    - **freeze orientation** (translations + fixed orientation)
2. **Contact Motion**
    - **freeze** (no motion)
    - **slide** (translation along constraint features, fixed orientation)
    - **pivot** (rotational motion about contact point, fixed position)
3. **Pushing**

Given a set of elementary motion modes, the operator then specifies to the system which mode she currently desires. To minimize the burden on the operator, this motion mode selection information can be supplied to the system via a hand-held push-button device.

In the following sections we elaborate on each type of elementary class of motions.

5.2 Free space motion

In free space we want to give the operator the maximum possible maneuverability. At the same time we want to aid the operator preserve positional/orientational parameters that she wishes to keep constant during a significant portion of a manipulation task. For instance, if the operator has achieved the desired approach orientation, then the system should allow her to *freeze* (lock) it and subsequently concentrate on translational motion of the slave robot (and MO) only. Similarly, situations may arise (*e.g.*, screwing, valve adjusting), where the operator has positioned the slave end-effector and wishes to freeze the position and concentrate on grasping or turning the desired feature. Therefore, we provide three corresponding elementary free space modes of motion. One could proceed further and introduce

single d.o.f. motion modes restricting the operator's motion to translations along a single direction at a time or rotations about a single axis. However, we have decided against such facilities as they increase the burden on the operator of having to mentally keep track of some task-based coordinate frame in which these restrictions would be specified, all at a dubious benefit to the operator's ability to perform tasks more easily or more efficiently.

Therefore we feel that the above free space motions provide a reasonable compromise between convenience (for the operator) and functionality. Finally, in view of Eq.(1), the three motion modes are realized in a straightforward fashion as follows

- general motion: $\Delta\mathbf{d} = (\mathbf{t}, \mathbf{r})$
- freeze position: $\Delta\mathbf{d} = (\mathbf{0}, \mathbf{r})$
- freeze orientation: $\Delta\mathbf{d} = (\mathbf{t}, \mathbf{0})$

5.3 Contact motion

5.3.1 Types of contact

When the movable object is in contact with the (simulated) environment, its motion (and therefore the motion of the slave manipulator) is restricted, depending on the type of contact. Figure 3 lists the types of contacts that we will consider in this work [31]. Let us emphasize again that we are concerned with rigid polyhedral contacts only. A few notes about Figure 3 are in order. It is easy to see that convex *vertex/vertex* and *vertex/edge* contacts are highly transient contact types and will rarely occur in practice . However, as pointed out in [31], the two types of contacts can be significant and persistent when one of the contacting features is concave. Following the work of Sawada et. al. and recognizing that vertices and edges can be either convex or concave, we generalize the contacts involving these two features to include both cases. This is reflected in Figure 3 by juxtaposing the two cases, separating them with a vertical dashed line.

We will in the following sections have the occasion of referring to *adjacent*, as well as *high* or *low* order contacts. All of these terms are to be interpreted in view of Figure 3. We will define an adjacent contact to be one which can be reached in one contact change from the current state. Also, we will say that a contact c_i is

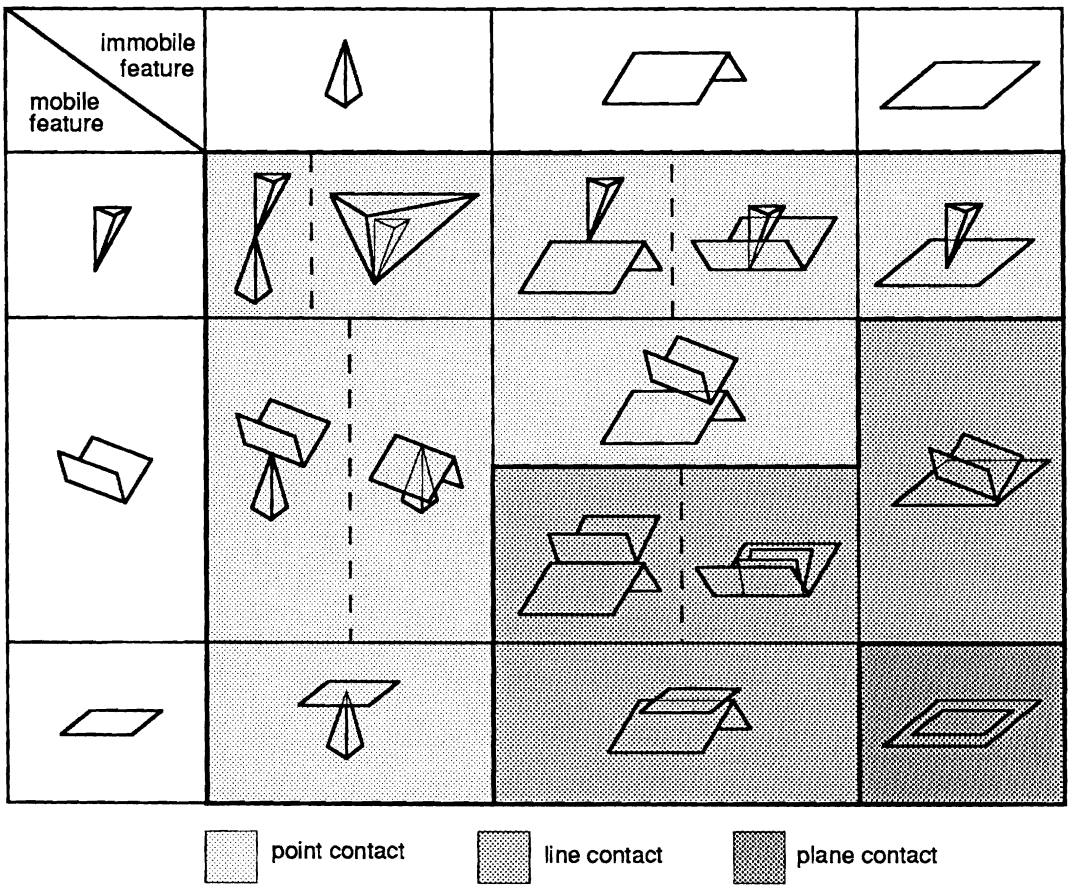

Figure 3: Types of polyhedral contacts.

higher (of higher order) than contact c_j, if c_i offers fewer remaining d.o.f. of motion than c_j.

5.3.2 Constraint normals

In Section 4.4 we discussed the nature of the constraint information maintained by the graphical simulator and passed to the master's Cartesian level servo module. Recall that for each active constraint this information includes an associated *unit normal direction*. We now offer a convention to unambiguously define this constraint normal in each contact type.

We will let the constraint normal in each case be directed away from the environment contact feature and towards the movable object (MO), *i.e.*, the normal specifies the direction *against* which MO can *not* move. Referring to Figure 3, it seems natural to consider the geometry of both contacting features in determining the direction of this normal. Still, different conventions may prove to be equally plausible and practical. We will choose to let the higher-order feature in each case dominate the choice and will break the ties in favor of the environment feature. The only exception to this rule will be the *edge/edge* point contact (see Figure 3), where the normal is most naturally defined by the cross-product of the two edge directions.

In keeping with the above convention, then, the constraint normal direction for a *face/face* planar contact is given by the face normal of the environment plane. Similarly, for the two line contacts involving only edges, as well as for the *vertex/vertex* point contact, the environment feature determines the normal. In all remaining contact types (except the already mentioned *edge/edge* contact), the higher-order feature (regardless of which object it belongs to) determines the axis (but not necessarily the direction) of the constraint normal.

Finally, the normals for each of the three elementary polyhedral features are defined in a straightforward fashion as illustrated in Figure 4.[6] Note that this definition assumes that all face normals in our polyhedral models are outward pointing.

[6]The asterisk (*) in Figure 4 denotes that the corresponding vector is of *unit magnitude*.

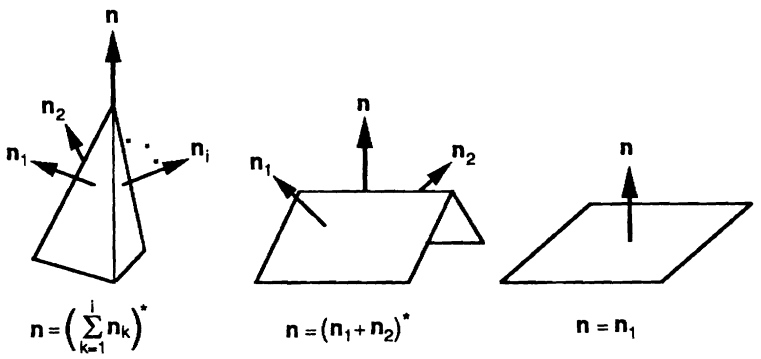

Figure 4: Constraint normals for the three types of polyhedral features.

5.3.3 Kinesthetic feedback

As we saw in Section 3, the graphical simulator maintains the current constraint information on the motion of the movable object. Thus, following the initial motion that caused a particular contact (and cause the new constraint to be reflected in the constraint information) the intended (*i.e.*, operator specified) motion of the movable object (MO) can be checked against the active constraints and appropriately modified (*i.e.*, filtered). Therefore, in the context of a purely kinematic simulation, we propose to provide (simulated) *kinesthetic feedback* to the operator by filtering the intended motion of MO, bringing it into compliance with the existing geometric constraints. By applying this filtered motion to the master manipulator as well (*i.e.*, backdriving the master manipulator appropriately), the operator holding the master *feels* these constraints as resistance to motion.

The filtering must be relatively simple, intuitively natural to the operator, fast to compute and as general as possible, given the above requirements. Simplicity and computational speed are necessitated by the requirement that the kinesthetic feedback be provided to the operator in real time.

5.3.4 Types of contact motions — overview

As indicated in Section 5.1, we propose three types of restrictions on the contact motion. For the case of fine precision motions, where even slight unintended changes in position/orientation of MO caused by an impact (contact with a new constraint)

are unacceptable, we provide the trivial *freeze* mode (no motion at all). In other words, all commanded motion of MO following a new contact is ignored until the operator selects a higher-order contact motion mode. Two such modes are provided.

In *slide* mode, the operator can slide MO along the constraining feature(s) (surfaces, edges) in the permissible directions (*i.e.*, the directions not violating *any* of the constraints). The orientation of MO remains fixed for the duration of motion in this mode. The system attempts to help the operator maintain contact with the environment but will allow the operator to break the contact if she clearly indicates such intent. A cruicial feature of the way we propose to handle contact motion is to require *decisive* actions on the part of the operator to transition to a lower-level contact. This aids the operator in preserving high-order contacts (which are presumed preferred), while still allowing her to transition to an arbitrary adjacent contact. We will analyze this class of motions in the case of a single constraint, as well as in a situation where multiple constraints are acting on MO simultaneously.

Alternatively, the operator can adjust the orientation of MO or transition between adjacent contacts by rotating or pivoting about the contact point (*pivot* mode). In this mode the contact point is not allowed to translate (slide) along or depart from the constraint feature. As the contact type (between MO and the environment) changes, the contact point moves on the surface of MO and with it the pivoting point about which rotational motions are computed. This allows a variety of reorienting and contact changing motions of MO. Again, motion analysis will be performed on the commanded displacements so as to aid the operator perform the desired changes of orientation. We will provide a restricted version of this motion modality to the operator also in situations where multiple constraints are restricting the motion of MO.

In the following sections we detail the proposed approach to contact motion analysis in free space as well as in contact.

5.3.5 'Freeze' mode

This trivial mode ($\Delta\mathbf{d} = (\mathbf{0}, \mathbf{0})$) is included solely to prevent unwanted slippage and twists of MO *w.r.t.* the environment upon the initial (or new) contact. This mode is thus the default contact mode, entered automatically when a new contact

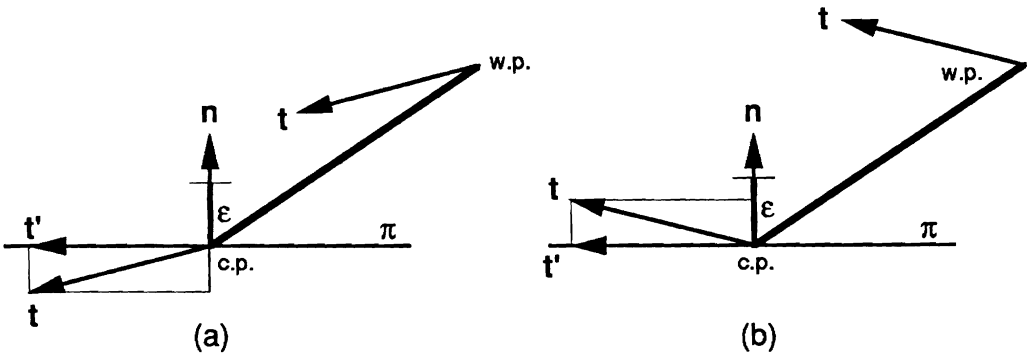

Figure 5: Single-contact sliding.

is detected.

5.3.6 'Slide' mode — single contact

In the case of a single contact, the constraint information, as defined in Section 4.4, specifies the unit constraint normal ${}^B\mathbf{n}$. Given the desired motion of the slave wrist $\left(\Delta^B\mathbf{d} = \left({}^B\mathbf{t}, {}^B\mathbf{r}\right)\right)$, we compute the corresponding allowable subset of translational motion $\Delta^B\mathbf{d}'$ as follows[7]

$$\Delta^B\mathbf{d}' = ({}^B\mathbf{t}' \ , \ \mathbf{0}) \tag{8}$$

where

$$\mathbf{t}' = \begin{cases} \mathbf{t} - (\mathbf{t}\cdot\mathbf{n})\mathbf{n} & , \text{ if } (\mathbf{t}\cdot\mathbf{n}) < \epsilon \\ \mathbf{t} & , \text{ otherwise} \end{cases} \tag{9}$$

Figure 5 illustrates a typical situation for single-contact sliding, where w.p. and c.p. denote the slave wrist center and the contact point, respectively. Note that choosing ϵ a positive value, the operation of Eq.(9) above will filter out not only the component of the commanded translation against the constraint normal $\mathbf{n}_i$, but also the component along $\mathbf{n}_i$ (*i.e.*, away from the contact) if its magnitude is smaller than ϵ (Figure 5). This, in effect, provides an illusion of *contact surface tension*, *i.e.*, with a proper choice of ϵ the operator is forced to exert a decisive, deliberate pull away from the contact in order to break it.[8]

[7]Note that the translational displacement of MO is the same as the commanded translational displacement of the slave wrist, despite the offset between the two.

[8]A reasonable value for ϵ may be half the maximum (positive) incremental displacement expe-

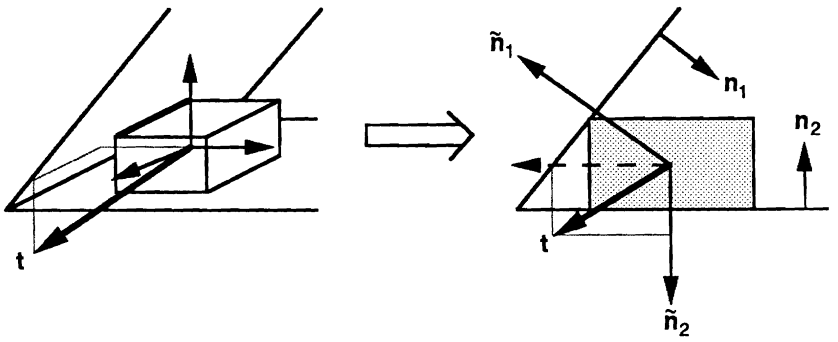

Figure 6: Multiple-contact sliding.

5.3.7 'Slide' mode — multiple contacts

In case of multiple contacts, the constraint information contains a list of constraint normals ${}^B\mathbf{n}_i$, which are currently restricting the motion of the movable object (MO). In general, these constraint normals will *not* be mutually orthogonal and we must approach the filtering process with caution. We will in the following refer to a *constrained direction* as the negative of the corresponding constraint normal $\mathbf{n}_i$, as defined in Figure 4, and denote it as $\tilde{\mathbf{n}}_i$.

Figure 6 illustrates a typical situation, where MO is in contact with two non-orthogonal constraints.[9] In this situation the operator should be able to slide MO along both constraining surfaces, break either contact and slide along the other contact's environment feature (surface), or even break both contacts and transition to free-space motion.

Again we will assume that the commanded incremental slave wrist motion is given as $\Delta^B\mathbf{d} = \left({}^B\mathbf{t}, {}^B\mathbf{r}\right)$. The analysis of the multi-contact case centers on identifying *the primary constrained direction* $\tilde{\mathbf{n}}_p$, *i.e.*, the one which is "closest to" the desired translational motion $\mathbf{t}$. The measure of closeness is the projection of $\mathbf{t}$ along a unit direction $\tilde{\mathbf{n}}_i$. Given this closest $\tilde{\mathbf{n}}_i$ (*i.e.*, $\tilde{\mathbf{n}}_p$), we then construct an orthogonal filtering frame $\mathcal{F}_F$, such that $\tilde{\mathbf{n}}_p$ is one of its axes, and the cross product with

rienced by the system during normal operation.

[9] A two-constraint example has been chosen for illustrative convenience. The discussion and results of this section apply to higher-multiplicity contacts as well.

any other constrained direction $\tilde{\mathbf{n}}_j$ gives its second orthogonal axis. This choice of a filtering coordinate frame is adopted because a commanded translational motion $\mathbf{t}$ in a multi-constraint case will normally give rise to a sliding motion along the constraint feature, whose associated constrained direction is closest to $\mathbf{t}$.

Having constructed the filtering frame, we then express both the commanded motion ${}^B\mathbf{t}$ and the constrained directions ${}^B\tilde{\mathbf{n}}_k$ in this frame (*i.e.*, ${}^F\mathbf{t}, {}^F\tilde{\mathbf{n}}_k$) and filter the commanded slave wrist motion accordingly. The sequence of steps below formalizes the filtering procedure and supplies the necessary details.

1. for all $c_i \in \mathcal{C}$, compute the projections $p_i = \left({}^B\mathbf{t} \cdot {}^B\tilde{\mathbf{n}}_i\right)$
2. let ${}^B\tilde{\mathbf{n}}_p = {}^B\tilde{\mathbf{n}}_i$, for which p_i is most positive over $\mathcal{C}$
3. construct the filtering frame $\mathcal{F}_F$,

$$\mathcal{F}_F \;=\; \left\{\left(\left({}^B\tilde{\mathbf{n}}_p \times {}^B\tilde{\mathbf{n}}_j\right) \times {}^B\tilde{\mathbf{n}}_p\right)^* \;,\; \left({}^B\tilde{\mathbf{n}}_p \times {}^B\tilde{\mathbf{n}}_j\right)^* \;,\; {}^B\tilde{\mathbf{n}}_p\right\}$$

   where $c_j \in \{\mathcal{C} - \{c_p\}\}$, *i.e.*, ${}^B\tilde{\mathbf{n}}_j \neq {}^B\tilde{\mathbf{n}}_p$;
   construct the rotational matrix ${}^B\mathbf{R}_F$ from $\mathcal{F}_F$ (see Section A.2)
4. map ${}^B\mathbf{t}$ into $\mathcal{F}_F$, *i.e.*, ${}^F\mathbf{t} = \left({}^B\mathbf{R}_F\right)^{-1} * {}^B\mathbf{t}$
5. for each $c \in \mathcal{C}$, filter ${}^F\mathbf{t}$ *w.r.t.* c,
   - map ${}^B\tilde{\mathbf{n}}$ into $\mathcal{F}_F$, *i.e.*, ${}^F\tilde{\mathbf{n}} = \left({}^B\mathbf{R}_F\right)^{-1} * {}^B\tilde{\mathbf{n}}$
   - filter each component of ${}^F\mathbf{t}$ in turn, *i.e.*,
     $\Lambda\left({}^F\mathbf{t}, {}^F\tilde{\mathbf{n}}, x\right)$, $\Lambda\left({}^F\mathbf{t}, {}^F\tilde{\mathbf{n}}, y\right)$, $\Lambda\left({}^F\mathbf{t}, {}^F\tilde{\mathbf{n}}, z\right)$
6. map filtered ${}^F\mathbf{t}$ back into $\mathcal{F}_B$, *i.e.*, ${}^B\mathbf{t}' = {}^B\mathbf{R}_F * {}^F\mathbf{t}$

Procedure 1: Multi-constraint sliding motion filter.

The core of the filtering process is Step 5, where each constrained direction $\tilde{\mathbf{n}}$ is in turn rotated into the filtering frame and the components of the commanded motion are filtered according to the Λ operator. This operator is defined as follows

$$\Lambda(\mathbf{t}, \tilde{\mathbf{n}}, x) : t_x = \begin{cases} t_x & , \text{ if } (\tilde{n}_x = 0) \text{ or } (t_x \cdot \text{sgn}(\tilde{n}_x)) \leq -\epsilon \\ 0 & , \text{ otherwise} \end{cases} \tag{10}$$

Therefore, any constrained components of the commanded motion are zeroed. Also, small components away from the constrained orthogonal directions are zeroed as well, providing a sense of surface tension as in the single-contact case above.[10] Having performed the filtering operation on ${}^F\mathbf{t}$, we then rotate the filtered commanded displacement back into the reference frame (Step 6) and assemble the final filtered motion of the slave wrist as $\Delta^B\mathbf{d}' = \left({}^B\mathbf{t}', \mathbf{0}\right)$.

Observe that a filtering frame is constructed even in the case where the original commanded motion does not violate any constraints, *i.e.*, when all p_i in Step 1 are negative. This is done so that the filtering of small components away from the constraint features in Step 5 (which must be done in this case as well) is performed in an orthogonal frame. The requirement that filtering be done only *w.r.t.* orthogonal axes is cruicial.

Finally, for clarity, various optimizations of the above procedure have been omitted (in particular, in Step 5). Any implementation must consider these carefully.

5.3.8 'Pivot mode — single contact

Computing the motion

As mentioned before, in this single contact mode the contact point is stuck in contact and can not be moved (*i.e.*, slid along a contact feature or pulled away from the contact). Only rotations of MO about the contact point are allowed. The class of allowed motions and the nature in which these motions are computed are intended to give the operator the feel of manipulating in a "sticky" environment, as well as allowing the operator to concern herself with the orientational parameters of MO alone, while keeping the contact point position fixed.

The input to the filtering module are the commanded (operator supplied) motion of the slave wrist ($\Delta^B\mathbf{d}$) and the current constraint information $\mathcal{C}$ (Section 4.4). Let the commanded motion be given as a displacement/RPY pair. Our task is to compute the rotational motion of the contact frame (centered at the contact point), based on the supplied slave wrist motion and subject to the above assumptions. Toward this aim we will define two coordinate frames (with the same orientation) as illustrated in Figure 7. In the figure, ${}^B\mathbf{n}$ is the constraint normal, the vector ${}^B\mathbf{p}$

[10] The same ϵ value may be used both in single and multiple-contact situations.

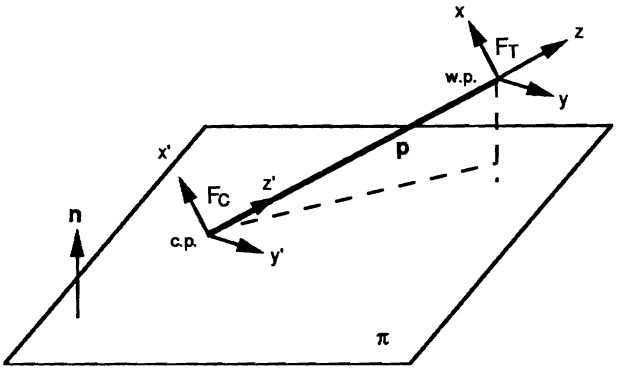

Figure 7: Tangential and contact frames.

denotes the (directed) distance between the slave wrist center point (w.p.) and the contact point (c.p.), and π labels the constraint feature (plane in this case). The first frame $\mathcal{F}_T$ (*tangential frame*) is defined such that its x-y plane is tangential to the surface of the sphere centered at c.p. and having radius $|\mathbf{p}|$. For convenience, we will define a second frame $\mathcal{F}_C$ (*contact frame*) with the same orientation as $\mathcal{F}_T$, but slid along the $\mathbf{p}$ vector, such that its origin coincides with the contact point, *i.e.*,

$$\mathcal{F}_T = \mathcal{F}_C = \left\{ {}^B((\mathbf{p} \times \mathbf{n}) \times \mathbf{p})^* \, , \, {}^B(-\mathbf{p} \times \mathbf{n})^* \, , \, {}^B(-\mathbf{p})^* \right\} \tag{11}$$

The rotational matrix ${}^B\mathbf{R}_T$, specifying the orientation of the frame $\mathcal{F}_T$ *w.r.t.* $\mathcal{F}_B$, is again derived directly from the above definition of the two frames (see Section A.2). In fact, since the orientations of the frames $\mathcal{F}_T$ and $\mathcal{F}_C$ are identical, we have ${}^B\mathbf{R}_T = {}^B\mathbf{R}_C$.

We will describe the (rotational) motion of the contact point in terms of the motion of the contact frame $\mathcal{F}_C$ due to the (operator supplied) motion of the wrist-based tangential frame $\mathcal{F}_T$. In an attempt to kinematically simulate the rotational motion of MO, whose contact point is stuck in contact, and at the same time minimize the complexity of motion analysis, we propose to compute the rotational motion of $\mathcal{F}_C$ as follows

(a) rotational motion of w.p. about the z-axis of $\mathcal{F}_T$ corresponds directly to the rotational motion of c.p. about the z-axis of $\mathcal{F}_C$

(b) translational motion of w.p. (along the x-y plane of $\mathcal{F}_T$) is used to compute

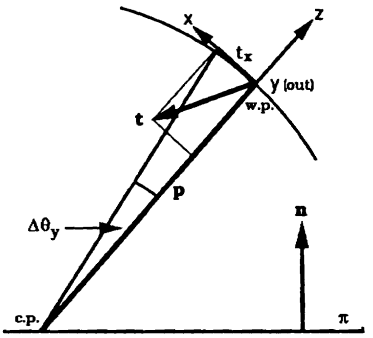

Figure 8: Computing $\Delta\theta_y$ of the contact frame.

the remaining two orthogonal rotational displacements of $\mathcal{F}_C$

In (b), the rotational displacement of $\mathcal{F}_C$ (about its x and y axes) is approximated by considering the components of the commanded translational vector ${}^T\mathbf{t}$ (*i.e.*, ${}^B\mathbf{t}$ rotated into the $\mathcal{F}_T$ frame) projected onto the x, y axes of $\mathcal{F}_T$. For the case of computing the incremental rotation $\Delta\theta_y$ about the y-axis of $\mathcal{F}_C$, we have

$$\Delta\theta_y = \frac{{}^T t_x}{|\mathbf{p}|} \tag{12}$$

Figure 8 illustrates the situation.[11,12]

An important detail that must be noticed here is that the translational vector ${}^B\mathbf{t}$ will only cause pivoting (rotation about c.p.) if it lies below the x-y plane of the *tangential frame* $\mathcal{F}_T$, *i.e.*, if

$$\left({}^B\mathbf{t} \cdot {}^B\mathbf{p}\right) \geq 0 \tag{13}$$

Therefore, the RPY rotation of $\mathcal{F}_C$ due to the (rotational and translational) motion of $\mathcal{F}_T$, under the assumption of stiction, is

$${}^C\mathbf{r}' = {}^T\mathbf{r}' = \left(-\frac{{}^T t_y}{|\mathbf{p}|} , \frac{{}^T t_x}{|\mathbf{p}|} , {}^T r_z \right) \tag{14}$$

[11]The y-axes of both $\mathcal{F}_T$ and $\mathcal{F}_C$ frames are directed out of the page.

[12]Note that the approximation of equating the tangential projections of the displacement vector t with the corresponding great arc segments along the sphere surface is equivalent to assuming that $\sin(\Delta\theta) = \Delta\theta$, as $\sin(\Delta\theta) = \Delta\theta = {}^T t_x/|\mathbf{p}|$ in Figure 8. It is easy to verify that this approximation is quite good for $-\pi/6 < \Delta\theta < \pi/6$, which is more than sufficient for our purposes.

The superscripts on the right hand side of the above equation indicate, that the corresponding displacement and RPY parameters have been rotated into the $\mathcal{F}_T$ coordinates. See Appendix A for details.

The computed rotational motion of the contact point (and thus MO) as given by Eq.(14), is designed to provide a natural feel to the operator, as she is forced to introduce translational motion at the slave wrist to achieve rotational (pivoting) motion at the contact point. In the absence of a full dynamic model, the generated model is only approximate, of course, but nevertheless it has an intuitive basis and should feel natural to the operator.

Filtering

Having computed the rotational motion of the contact point based frame $\mathcal{F}_C$, we now filter this motion on the basis of the contact type. The filtering is done primarily to discard small (presumably unintended) rotational components and has the effect of biasing (the interpretation of) operator's motions towards higher order contact types. In the following paragraphs we will outline the filtering procedure.

In order to filter the rotational motion of $\mathcal{F}_C$, we will first define a contact point based *filtering frame* $\mathcal{F}_F$, which is particularly convenient for the given constraint type. We will then express the intended motion of the contact point in this frame ($\mathcal{F}_F$) and perform the filtering *w.r.t.* its coordinates. In each case the filtering frame will be constructed in terms of the geometric parameters supplied by the constraint information, *i.e.*, the constraint normal (${}^B\mathbf{n}$), wrist-to-contact vector (${}^B\mathbf{p}$), and the edge direction (${}^B\mathbf{e}$) (see Section 4.4). The input motion of the collision point $\Delta^C\mathbf{d}' = (\mathbf{0}, {}^C\mathbf{r}')$ is as computed in Eq.(14) above.

(a) Point Contacts: A filtering frame need not be specified in this case as all three orthogonal rotations are permissible in all point contacts (see Figure 3). Therefore, no filtering is necessary.

(b) Line Contacts: A line contact always involves an edge (at least one, see Figure 3), and it is this edge direction (${}^B\mathbf{e}$), together with the constraint normal (${}^B\mathbf{n}$), that defines the most convenient filtering frame, *i.e.*,

$$\mathcal{F}_F = \left\{ {}^B(\mathbf{e} \times \mathbf{n}) \, , \, {}^B\mathbf{e} \, , \, {}^B\mathbf{n} \right\} \tag{15}$$

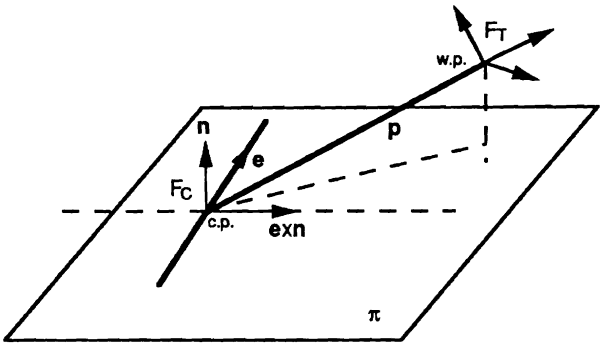

Figure 9: Single-contact pivoting — line contact.

where ${}^{B}\mathbf{e}$ and ${}^{B}\mathbf{n}$ are assumed to be of unit magnitude. The specification of the rotational matrix ${}^{B}\mathbf{R}_{F}$ follows immediately (see Section A.2). Figure 9 illustrates the case of an *edge/face* line contact.

Filtering of the contact point motion $\Delta^{C}\mathbf{d}'$ can now be achieved as a two-stage process:

1. map the motion (RPY rotation) of the contact point from $\mathcal{F}_C$ $({}^{C}\mathbf{r}')$ into $\mathcal{F}_F$ $({}^{F}\mathbf{r}')$, using ${}^{C}\mathbf{R}_{F} = \left({}^{B}\mathbf{R}_{C}\right)^{-1} * {}^{B}\mathbf{R}_{F}$ (see Section A.3)

2. filter out small rotations about ${}^{B}(\mathbf{e} \times \mathbf{n})$ tending to destroy the edge contact, *i.e.*,

$$ {}^{F}\mathbf{r}'' = \left(\Upsilon({}^{F}r'_{x}) , {}^{F}r'_{y} , {}^{F}r'_{z} \right) \tag{16} $$

where the Υ operator is defined as follows[13]

$$ \Upsilon(x) = \begin{cases} 0 & , \text{ if } |x| < \xi \\ x & , \text{ otherwise} \end{cases} \tag{17} $$

(c) Plane Contacts: The only representative of this class of contacts is the *face/face* contact (see Figure 3). Here, the filtering frame can be defined as fol-

[13]The Υ operator is a simple bidirectional threshold filter zeroing out rotations whose magnitude is smaller than ξ ($\xi > 0$). A good candidate value fore ξ may be half of the maximum magnitude of an incremental rotational displacement normally experienced by the system. This forces the operator to indicate a decisive rotation about the edge in order to break the edge contact.

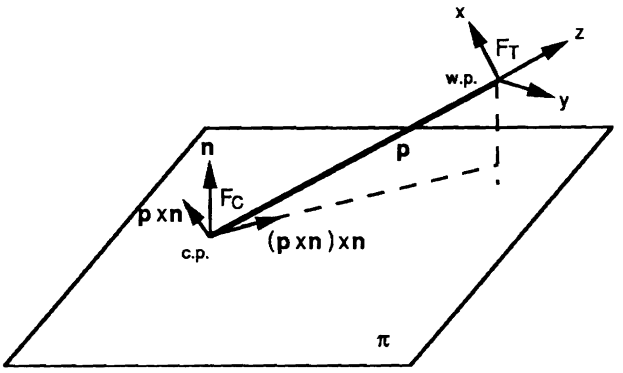

Figure 10: Single-contact pivoting — plane contact.

lows

$$\mathcal{F}_F = \left\{ {}^B((\mathbf{p} \times \mathbf{n}) \times \mathbf{n})^* \, , \, {}^B(\mathbf{p} \times \mathbf{n})^* \, , \, {}^B\mathbf{n} \right\} \tag{18}$$

and the rotational matrix ${}^B\mathbf{R}_F$ can be constructed as before. Figure 10 illustrates the situation.

Again, a two-stage filtering procedure is employed. The given rotational motion of the contact point is mapped from $\mathcal{F}_C$ into $\mathcal{F}_F$ (via the rotational matrix ${}^C\mathbf{R}_F$). The second filtering stage in this case attempts to remove from ${}^F\mathbf{r}'$ small destabilizing rotations about the x and y-axes of the filtering frame, *i.e.*,

$${}^F\mathbf{r}'' = \left(\Upsilon({}^F r'_x) \, , \, \Upsilon({}^F r'_y) \, , \, {}^F r'_z \right) \tag{19}$$

Postprocesing

Having computed the filtered motion of the pivoting contact point, we must now produce the corresponding motion of the slave wrist in the reference ($\mathcal{F}_B$) coordinates, as this is the motion ultimately commanded to the slave manipulator. This is accomplished by mapping the filtered contact point motion $\Delta^F\mathbf{d}'' = (\mathbf{0}, {}^F\mathbf{r}'')$ into $\mathcal{F}_B$ coordinates $\Delta^B\mathbf{d}''$ (see Section A.3) and computing the corresponding $\mathcal{F}_B$ displacement of the slave wrist as described in Section A.4.

5.3.9 'Pivot mode — multiple contacts

In this section we extend the results of Section 5.3.8 to accommodate a restricted, but useful subset of multi-constraint pivoting motions. The restrictions are imposed

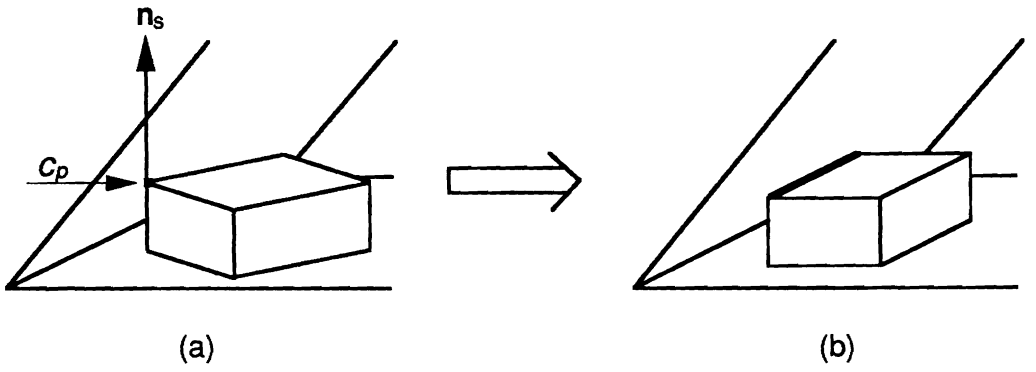

Figure 11: Multiple-contact pivoting.

both to aid the operator perform simple and intuitive multi-contact rotations, as well as to keep the geometrical and numerical complexity of the motion analysis low.

A typical situation that this motion mode is intended to address is one, where the operator has brought the movable object into a multiple contact and wishes to align MO *w.r.t.* the environment so as to obtain a higher order (*i.e.*, more stable) contact type. Figure 11-a illustrates an example, where MO has been slid along a surface (*face/face* contact) against a wall (*vertex/face* contact). This mode will allow the operator to rotate the object into a stable configuration *w.r.t.* the environment (*i.e.*, *edge/face* wall contact, Figure 11-b) and align MO for subsequent sliding along either or both of the constraining surfaces.

It is clear, that in view of the intended applications of this motion mode, the only practical situations will involve two constraints. Also, we will assume that realigning motions either preserve or raise the order of existing contacts. Finally, as any pivoting multi-constraint motion will involve sliding of the moving object along one of the constraints, we will require that one of the contacts be a *face/face* contact.

While the imposed conditions may seem restrictive, the allowed motions still span a sizable set of useful realignment motions that may be needed in a practical application. For instance, most two-constraint situations will arise by sliding the movable object against a second constraint, where the single-constraint sliding motion will be performed in a *face/face* contact state for obvious reasons of con-

venience and stability. Similarly, upon encountering a second constraint, the most likely subsequent motion (if any) is one where the object is pivoted about this new contact into a higher order multiple contact state.

In order to compute the allowed motion of MO in a two-contact situation, we will again make use of the notion of a *primary constraint*, and label the two contacts as *primary* (c_p) and *secondary* (c_s) contact. By convention, we will refer to the mandatory *face/face* contact as the secondary contact. The motion of MO will then be computed as a pure rotation about the contact point associated with the primary contact, and filtered such that it will not violate the secondary constraint. Clearly, if any rotation is to take place, the primary contact must be of a lower order (*e.g.*, *vertex/face*, *edge/face*, *face/edge*, *etc*) than the secondary contact. Moreover, if the primary constraint forms a line contact (see Figure 3), then motion will only be possible if the corresponding edge direction is parallel to the secondary contact normal $\mathbf{n}_s$ (see Figure 11).

Once again, let the original commanded motion of the slave wrist be given by $\Delta^B\mathbf{d} = ({}^B\mathbf{t}, {}^B\mathbf{r})$. Assuming that the above set of conditions is satisfied, we identify the primary constraint c_p and compute rotational motion ${}^C\mathbf{r}'$ about its associated contact point as in Section 5.3.8 (Eq.(14)). This contact-frame based RPY rotation must then be filtered so as to retain only the rotation about the axis parallel to the normal of the secondary constraint. We therefore define a filtering frame $\mathcal{F}_F$, such that one of its axes (*e.g.*, z) coincides with this normal direction, *i.e.*,

$$\mathcal{F}_F = \left\{ {}^B((\mathbf{n}_p \times \mathbf{n}_s) \times \mathbf{n}_s)^* \, , \, {}^B(\mathbf{n}_p \times \mathbf{n}_s)^* \, , \, {}^B\mathbf{n}_s \right\} \tag{20}$$

and map the rotation ${}^C\mathbf{r}'$ into this frame to obtain ${}^F\mathbf{r}'$ (see Section A.3). The filtered rotation is then obtained trivially as

$${}^F\mathbf{r}'' = \left(0, 0, \Upsilon({}^F r_z')\right) \tag{21}$$

The remaining task is to compute the corresponding motion $\Delta^B\mathbf{d}'$ of the slave wrist in the reference frame coordinates. This is accomplished in a straightforward fashion as described at the end of the previous section.

5.4 Pushing

5.4.1 Single-contact pushing

It has been established that pushing motions are difficult to analyze and predict accurately [29][28][20][21][19]. This is due primarily to the fact that the motion of a pushed object depends critically on the complex interaction between the microscopic features of the two sliding surfaces. This in turn accounts for continuously changing frictional properties of the sliding contact, making reliable predictions of the resulting motions impossible without a detailed knowledge of the surface textures and the resulting distribution of the support forces.

In order to facilitate rudimentary pushing operations and yet generate instructions which can be executed successfully and reliably under the slave's local supervision, we provide a simple pushing mode, where the operator can indicate to the system that she wishes to push an object through a certain distance along a *straight-line* trajectory. We naturally require that the object to be pushed be in a *face/face* contact with some supporting surface and that the task information (Section 2.6) indicate that this object is in fact *pushable*. We also require that the slave establish a *face/face* contact with the *pushed object* (PO). The requirements of a straight-line pushing motion and a planar *pushing contact* (between PO and the slave) minimize the possibility of slippage along the pushing contact or unexpected twists of the pushed object in the actual environment.

One more requirement restricting the operator's choice of the pushing contact and aimed at eliminating slipping errors in the remote environment, is that the pushing contact plane have a "reasonable" orientation *w.r.t.* the sliding surface. We quantify this condition by introducing a *pushing frame*

$$\mathcal{F}_P = \left\{ {}^B((\mathbf{n}_p \times \mathbf{n}_s) \times \mathbf{n}_s)^* \, , \, {}^B(\mathbf{n}_p \times \mathbf{n}_s)^* \, , \, {}^B\mathbf{n}_s \right\} \tag{22}$$

centered at the contact point associated with the pushing contact, and requiring that the pushing contact normal $\mathbf{n}_p$ be nearly parallel (within $\alpha_{\max}$)[14] to the sliding direction $\mathbf{d}_s = (\mathbf{n}_p \times \mathbf{n}_s)^*$ (see Figure 12), *i.e.*,

$$(\tilde{\mathbf{n}}_p \cdot \mathbf{d}_s) < \cos \alpha_{\max} \tag{23}$$

[14]The optimal value of $\alpha_{\max}$ will depend on the frictional parameters of a particular pushing contact. A reasonable "generic" value, however, may be around 30°.

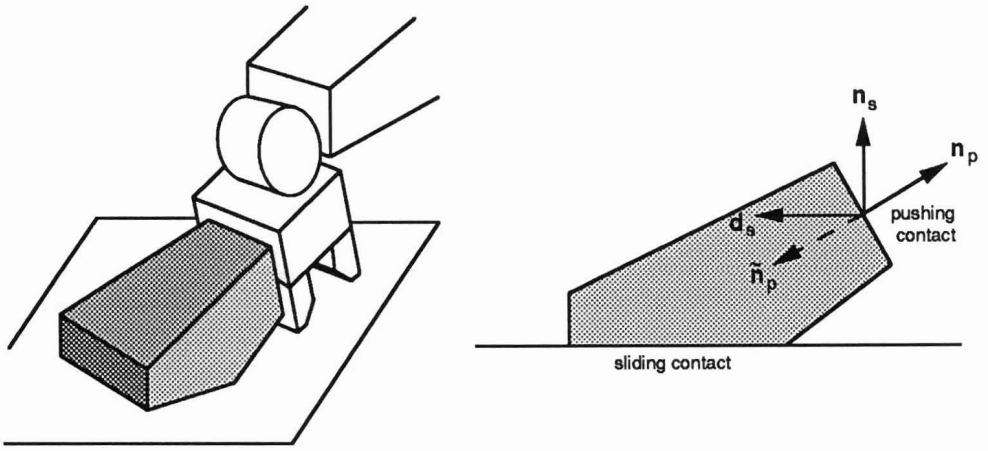

Figure 12: Single-contact pushing.

where $\tilde{\mathbf{n}}_p = -\mathbf{n}_p$.

In order for pushing motion to take place, the operator must first establish a *face/face* contact with some environment object. We propose that the operator signal her intent to push the object by exerting a significant (and therefore easily identifiable) threshold force $\mathbf{f}_t$ against it. If this object is identified as pushable, the system then enters the *pushing mode*. In this mode, the graphical simulator rigidly attaches the pushed object to the slave at the point of pushing contact and filters commanded slave wrist motions so as to move in a straight line along the sliding surface. Given a commanded motion $\Delta^B\mathbf{d}$ of the slave wrist, we therefore compute the translational motion of the pushed object as follows

$$\Delta^B\mathbf{d}' = \left({}^B\mathbf{t}', \mathbf{0}\right) \tag{24}$$

where

$${}^B\mathbf{t}' = \begin{cases} {}^B\mathbf{t} \cdot \mathbf{d}_s & , \text{ if } \left({}^B\mathbf{t} \cdot \mathbf{d}_s\right) > 0 \\ \mathbf{0} & , \text{ otherwise} \end{cases} \tag{25}$$

Note that the pushed object moves only if the commanded incremental translational displacement has a positive component along the sliding direction. Otherwise, zero displacement is applied to the object (and thus the slave), unless this pull away

from the pushing contact exceeds the threshold force $\mathbf{f}_t$, signalling that the operator wishes to exit the pushing mode.

Whereas every precaution has been taken to ensure that pushing motion commands generated at the operator's station are simple and easily executable by the slave, things can still go wrong. In particular, as the operator's station relies on a kinematic simulation of the slave world, error conditions such as the pushed object tipping over in the remote world can not be predicted and detected ahead of time. Avoiding such situations is thus left to the operator who can draw on her approximate knowledge of the relevant dynamic parameters or simply on her intuition in choosing a reasonable pushing contact.

5.4.2 Multiple-contact pushing

In order to enhance the versatility of the system, we again extend the single-constraint pushing motion mode to multi-contact situations. The intended functionality of this mode is essentially identical to multi-constraint pivoting motions (Section 5.3.9). Again we envision this class of motions being used primarily to push and align an object with respect to two simultaneously active environmental constraints. Consequently, the analysis of such aligning pushing motions is therefore analogous to the double-constraint rotational motion case, with the movable object in this case being the pushed object together with the (rigidly attached) slave's end-effector or tool, if any.

6 Filtering of operator's motions

In this section we describe a simple filtering procedure, which is applied to the positional data generated by the graphical simulator. The aim of this filtering stage is to smooth the observed slave trajectories and eliminate the undesired noise in the data.

The input to this module is the motion of the slave as computed in Section 5. As we have seen, various filtering steps have already been applied to the operator-generated motions so as to avoid object penetration and to force the operator to clearly indicate her intent to break (or reduce the order of) an existing contact. We will therefore assume that all the contact changes contained in the incoming data

(*i.e.*, generated by the kinesthetic feedback module) were intended by the operator and that there is no further need to detect and to eliminate transient changes of contact type.

During the same *contact state* (*i.e.*, the same set of elementary contacts), the information available from the graphical simulator is the trajectory of the slave end-effector along the unconstrained degrees of freedom defined by this contact state. This trajectory $\mathcal{T}$ is initially represented by the discrete set $\{\mathbf{p}_i : 0 \leq i \leq n\}$, where $\mathbf{p}_i = (\mathbf{t}_i, \mathbf{r}_i)$ describes the position and orientation of the frame $\mathcal{F}_{SW}$ (the frame attached to the slave wrist) at the i-th step of the simulation, and n is the number of discrete positional data acquired since the generation of the last command stream.[15]

This trajectory needs to be filtered for two reasons:

- The positional data will be inherently noisy due to the way in which this information is acquired, *i.e.*, operator-guided motions of the master. The filtering will eliminate small oscillations and deviations introduced by the operator and the sensor readings.

- More importantly, this trajectory has to be represented in a more compact fashion in order to reduce the number of motion commands to be sent to the remote slave.

Given the set describing $\mathcal{T}$ and two thresholds $\epsilon_{\mathbf{t}}$, $\epsilon_{\mathbf{r}}$, the filtering algorithm produces an approximate trajectory $\mathcal{T}_\epsilon$, composed of straight-line translations and rotations of $\mathcal{F}_{SW}$, such that $\mathcal{T}_\epsilon$ stays inside the *space tunnel* defined by $\mathcal{T}$ and by the radii $\epsilon_{\mathbf{t}}$ and $\epsilon_{\mathbf{r}}$ (for the translational and rotational components, respectively).

The algorithm starts with the simpliest approximation of $\mathcal{T}$, *i.e.*, the straight-line segment between the initial generalized position[16] $\mathbf{p}_0$ and the final one $\mathbf{p}_n$. If this approximation is "close enough" to $\mathcal{T}$, the algorithm simply returns this straight-line motion. Otherwise, an intermediate position $\mathbf{p}_j$ in $\mathcal{T}$ is added to the representation of $\mathcal{T}_\epsilon$ and the two line segments $Seg(\mathbf{p}_0, \mathbf{p}_j)$ and $Seg(\mathbf{p}_j, \mathbf{p}_n)$ are respectively checked against the corresponding portions $\{\mathbf{p}_i : 0 \leq i \leq j\}$ and

[15] Generation and partitioning of the command streams will be adressed in Section 7.

[16] We use the term *generalized position* to denote the 6-vector of positional and orientational parameters.

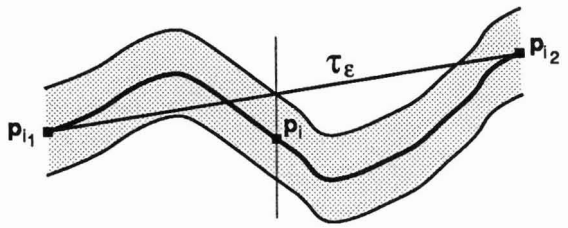

Figure 13: Trajectory filter — the "closeness test".

$\{\mathbf{p}_i : j \leq i \leq n\}$ of the original trajectory $\mathcal{T}$. The same process is iteratively applied to each segment which needs to be refined and the algorithm converges to an approximation of $\mathcal{T}$ by a polygonal path including generally only a few intermediate points. Clearly, the larger the space tunnel defined by the radii $\epsilon_\mathbf{t}$ and $\epsilon_\mathbf{r}$ around $\mathcal{T}$, the fewer intermediate positions will be returned.

A line segment $Seg(\mathbf{p}_{i_1}, \mathbf{p}_{i_2})$ of $\mathcal{T}_\epsilon$ is considered to be a good approximation of the corresponding part of $\mathcal{T}$ defined by the set $\{\mathbf{p}_i : i_1 \leq i \leq i_2\}$, if all the $\mathbf{p}_i$ verify

$$\max\left(\frac{\|\mathbf{t} - \mathbf{t}_i\|}{\epsilon_\mathbf{t}}, \frac{\|\mathbf{r} - \mathbf{r}_i\|}{\epsilon_\mathbf{r}}\right) < 1 \tag{26}$$

where $\mathbf{t}$ (resp. $\mathbf{r}$) denotes the closest point on $Seg(\mathbf{t}_{i_1}, \mathbf{t}_{i_2})$ (resp. $Seg(\mathbf{r}_{i_1}, \mathbf{r}_{i_2})$) to $\mathbf{t}_i \in \mathcal{T}$ (resp. $\mathbf{r}_i$). Figure 13 illustrates the process.

Several approaches can be adopted for the selection of the intermediate position to be introduced after each non-terminal iteration of the algorithm. The point on $\mathcal{T}$ which is farthest from the current approximation $\mathcal{T}_\epsilon$ is in general a good candidate. However, the drawback of this method is that it requires the computation of all distances between the points $\mathbf{p}_i \in \mathcal{T}$ and the line segment $Seg(\mathbf{p}_{i_1}, \mathbf{p}_{i_2})$, $i_1 < i < i_2$.

Consequently, a *binary subdivision* method offers a much more efficient approach: as soon as the algorithm finds a $\mathbf{p}_i$ which does not satisfy the "closeness test" of Eq.(26) for a given line segment of $\mathcal{T}_\epsilon$, it immediately introduces a new generalized position vector $\mathbf{p}_j$, where $j = \max\left(\frac{i_1+i_2}{2}, i\right)$, and cuts this segment into $Seg(\mathbf{p}_{i_1}, \mathbf{p}_j)$ and $Seg(\mathbf{p}_j, \mathbf{p}_{i_2})$.

Clearly, this method will sometimes produce a slightly larger number of intermediate positions than the former approach. Notice, however, that the algorithm will at each step at least halve the complexity of the problem.

This filtering procedure must be applied to all six components of the positional information in the case of a general motion in free space. However, both in the case of free-space motion with frozen orientation (resp. position), as well as in the case of sliding (resp. pivoting) contact motion, only positional (resp. orientational) motion parameters need to be filtered. Moreover, in each motion mode, only the components corresponding to the free degrees of freedom defined by the contact type need this filtering stage. For example, during a sliding motion along a plane whose normal coincides with the $\mathbf{z}$ axis of the reference frame $\mathcal{F}_B$, only the components of translational motion along ${}^B\mathbf{x}$ and ${}^B\mathbf{y}$ will need to be filtered.

7 Generation of symbolic slave commands

In this section we detail our approach to using the sequence of contact state changes (Section 5) and the filtered slave trajectory information within each contact state (Section 6) to extract a stream of symbolic commands to the remote slave.

The commands which will be issued to the slave by the system can be classified into two groups. The first group is composed of *low-level* commands, essentially encompassing *guarded* and *compliant* motions. These commands will be generated to execute simple tasks such as free-space navigation, pick and place operations, motion into contact with the environment, contour following, *etc.*

The *high-level* class of motions, on the other hand, contains more specific special-purpose operations such as tight tolerance part mating, fine-precision motions, *etc.* Even if the operator were able to perform a complex insertion in the simulated world, the observed sequences of contacts clearly would not be reproducible by the slave, due to the environment modeling errors. Therefore, such tasks can not be decomposed into elementary motions and must be executed autonomously by the slave under local sensory supervision. In this case, the graphical simulator need only identify that the operator wishes to perform a high-level operation (either by using the information provided by the task model or by interpreting the operator's motion information directly). The system then gathers the relevant parameters of the task and sends this information to the remote slave, where the information is used to instantiate a local special-purpose procedure.

A new stream of commands is issued after each addition or deletion of a new

contact. However, there is also a maximum time (*e.g.*, on the order of the transmission delay) after which a new stream is automatically generated even if the same contact state persists. This is done to avoid increasing the delay and to prevent accumulation of the positional information to be processed.

In this section, we restrict our analysis to the generation of the low-level commands and discuss the algorithms used to transform the contact-state and positional information provided by the graphical simulator and by the kinesthetic feedback module in order to produce a stream of guarded and compliant motion commands to be executed by the slave.

7.1 Types of motions

An important issue that must be addressed when generating these commands results from the presence of uncertainties in the world model used by the graphical simulator. During free space motion, simple positioning commands will generally be sufficient to be executed safely by the slave. However, as soon as the task involves interactions between the robot and its environment, these discrepencies may cause a failure during the command execution. This problem has been studied extensively during the last decade [18][35] and various methods of using the forces and torques occurring during the contact motion to suitably adapt the robot's trajectory have been proposed. In an hybrid force-position approach [17][27][18] the *free* directions of the motion are controlled in position (or velocity), while the directions *constrained* by the contacts are controlled in force. This hybrid mode of control allows two additional sets of robot commands: *guarded motions* and *compliant motions*. A guarded motion is generally used when approaching a surface to avoid excessive forces after the contact is established. A compliant motion is then required to move along one or more constrained surfaces while maintaining a given force (or torque) constraint in the directions normal to constraining surfaces.

The following section describes how the positioning and contact information provided by the graphical simulator can be translated into a stream of such hybrid control motions.

7.2 Task frame specification

In order to facilitate convenient specification of guarded and compliant motions of the slave manipulator, we will define a *task frame* $\mathcal{F}_T$, such that its position and orientation is closely related to the constraints imposed by the geometry of the current contacts. For each type of elementary contact, the task frame $\mathcal{F}_T = \{\mathbf{p}\,;\, \mathbf{n}_x, \mathbf{n}_y, \mathbf{n}_z\}$ is defined in the following manner:

- Its origin $\mathbf{p}$ coincides with the centroid of the *contact feature*.
- $\mathbf{n}_z$ is aligned with the *constraint normal* (see Section 5.3.2).
- For the three types of contact where an edge is involved (see Figure 3), $\mathbf{n}_y$ is aligned with the direction of this edge. For the other cases, an arbitrary direction lying in the contact plane is chosen.
- $\mathbf{n}_x$ is obtained by $\mathbf{n}_y \times \mathbf{n}_z$.

More work needs to be done to identify the optimal choice of task frame coordinates for the case of multiple-constraint motions!!

Whenever a new task frame needs to be specified, an assignment command is sent to the slave. This command must specify the 3-dimensional vectors $\mathbf{p}, \mathbf{n}_y$ and $\mathbf{n}_z$. In general, this task frame will not have a fixed relation with respect to the reference frame or to the end-effector frame. Depending of the contact type, each of these vectors can be defined either with respect to the base frame $\mathcal{F}_B$ or with respect to the slave's wrist frame $\mathcal{F}_{SW}$.

This assignment of the task frame coordinate frame axes could be specified with the following syntax:

AssignFrame (*orig* $\mathbf{v}_1$: *BaseFrame* ;
diry $\mathbf{v}_2$: *WristFrame* ;
dirz $\mathbf{v}_3$: *BaseFrame*)

7.3 Motions to keep contact

Two types of commands are issued to specify the compliant motions. The first one specifies the directions in which the robot has to *comply* and the forces/torques

to be applied during the motion. The later describes the *positional* displacements along the remaining degrees of freedom. Because the task frame has been chosen to be aligned with the constraints imposed by each contact geometry, the specification of the compliants commands becomes straightforward.

For the case of sliding motions, regardless of the contact type, the translational motion along the x and y directions of $\mathcal{F}_T$, will be position controlled while a force will be specified along the z-axis to maintain the contact.

In point-contact pivoting mode (see Figure 3), any rotational motion around the contact point is allowed and the three axes are therefore position controlled. Line contacts will require that zero torque be maintained about the contact-plane axis perpendicular to the edge direction. Finally, the only allowed rotation in a planar contact is the rotation about the constraint normal direction (task frame z-axis) and zero torques must therefore be commanded about the other two axes. In all cases a force must also be maintained along the z-axis to maintain contact.

The force to be exerted will be specified by a symbolic value in order to indicate what the intended result of this force is (for example $F_{Stiction}$ or $F_{Sliding}$). The actual values of these forces will depend on the physical parameters of the task (*e.g.*, contact surface friction, *etc.*) and will be determined by the slave manipulator control software.

For example, during an *edge/face* contact, the following sequence of commands will be generated to execute a simple translational motion through a distance d in the direction of this edge, followed by a rotational motion through an angle of α around the constraint normal direction:

Comply ($fz\ \mathrm{F}_{Sliding}$; $tx\ 0$)
Move ($y\ d$)
Comply ($fz\ \mathrm{F}_{Stiction}$; $tx\ 0$)
Twist ($rz\ \alpha$)

where both forces are positive and $\mathrm{F}_{Stiction}$ is presumably larger than $\mathrm{F}_{Sliding}$.

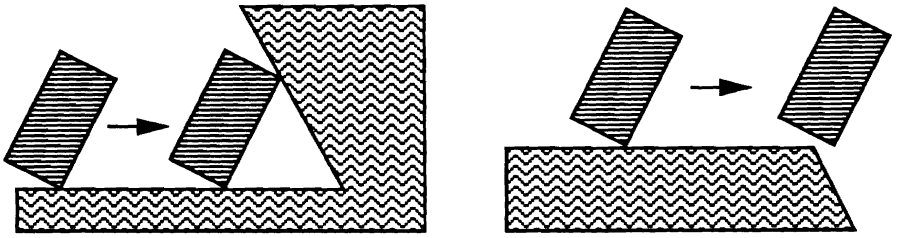

Figure 14: Changes of contact during a sliding motion.

7.4 Motions to change contact

Both sliding and pivoting motions can cause a change of contact. Sliding motions can result only in the introduction of a new contact or deletion of a current one. Pivoting motions, on the other hand, will generally cause a change of the current contact type (for example, a transition from a *vertex/face* to an *edge/face* contact).

Whenever such changes are observed in the simulated world, the command generator must specify one (or more) terminating conditions for each of the corresponding motions. Normally, a motion will be terminated when certain (specified) forces and/or torques exceed their respective thresholds. However, a maximum displacement must also be provided to stop the motion in case the guarded motion does not encounter the expected terminating condition (usually a contact).

7.4.1 Sliding case

When sliding motion along a given direction encounters a new contact (see Figure 14-a), it has to be stopped when a force discontinuity occurs along this direction. Figure 14-b, however, illustrates a situation where a different terminating condition needs to be specified. The mobile object is being slid along a surface and the motion should be stopped when the boundary of the sliding surface is reached. In this and similar situations, termination of the motion corresponds to the occurrence of a positional discontinuity on the axis which was controlled in force during sliding.

7.4.2 Pivoting case

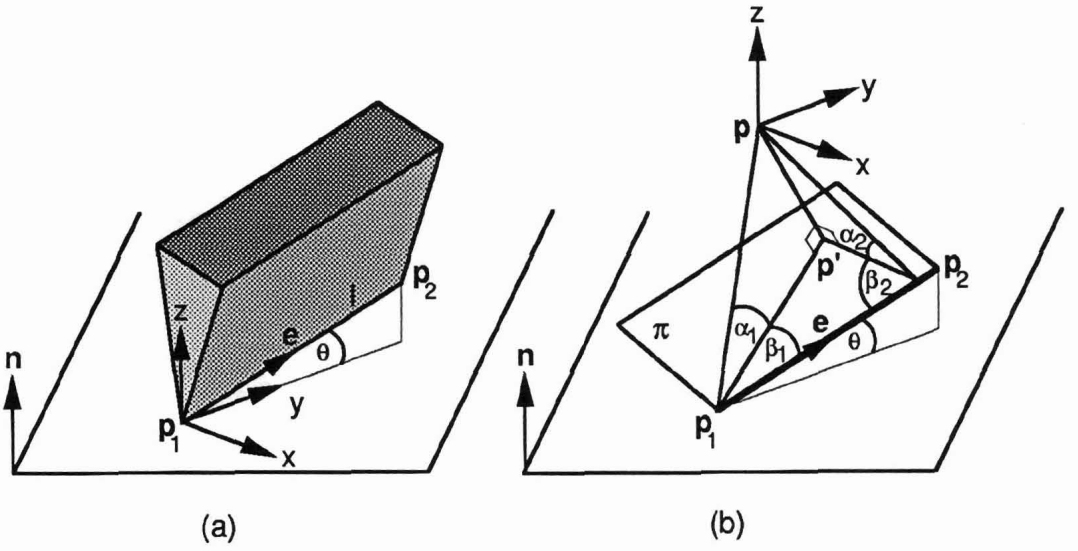

Figure 15: Transition between two *vertex/face* contacts.

When a change of the contact type occurs, this transition can be characterized by a discontinuity of the torques about the contact edges. For example, figure 15-a illustrates a situation where a vertex of the mobile object is in contact with a planar surface of the environment. A rotational motion around the contact point $\mathbf{p}_1$ is then applied to put the edge $\mathbf{e}$ in contact with this face, while exerting a positive force f along $-\mathbf{n}$. In the frame defined by $\{\mathbf{p}_1\,;\,(\mathbf{e}\times\mathbf{n})^*,(\mathbf{e}\times\mathbf{n})^*\times\mathbf{n},\mathbf{n}\}$, the component τ_x of the torque acting on $\mathbf{p}_1$ remains null while this point remains in contact with the surface. However, when the transition occurs and the vertex $\mathbf{p}_2$ comes into contact with the supporting plane, this torque τ_x will suddenly increase to $f \cdot l$ (where l is the length of the edge) and the contact can thus be detected.

In fact, we will show that this variation of torque remains constant, independently of the position of the coordinate frame in which the torques are expressed. This provides an easy way to detect such transitions directly from the torques measured in the frame of the F/T sensor, mounted at the slave manipulator's wrist.

Independence of the torque measurement site:

Figure 15-b illustrates a situation similar to the one shown in Figure 15-a, *i.e.*, an edge $\mathbf{e}$ of the movable object in contact with a plane by one of its vertices. However,

this time we assume that the torques are to be expressed relative to a coordinate frame with the same orientation as before but whose origin has been moved from $\mathbf{p}_1$ to $\mathbf{p}$.

The torque acting at $\mathbf{p}$ due to the reaction force $\mathbf{f} = (0, 0, f)^T$, applied at the point of contact, is expressed in the frame $\{\mathbf{p}\,;\,(\mathbf{e}\times\mathbf{n})^*, (\mathbf{e}\times\mathbf{n})^*\times\mathbf{n}, \mathbf{n}\}$ as follows

$$\tau = \mathbf{r}\times\mathbf{f} = (r_y\cdot f, -r_x\cdot f,\, 0)^T \tag{27}$$

where $\mathbf{r}$ is the vector from $\mathbf{p}$ to the point of contact. During a contact with $\mathbf{p}_1$, the components of this vector $\mathbf{r}_1 = \overline{\mathbf{pp}_1}$ are

$$\mathbf{r}_1 = \begin{pmatrix} l_1\cdot\cos\alpha_1\cdot\sin\beta_1 \\ l_1\cdot(\sin\alpha_1\cdot\sin\theta - \cos\alpha_1\cdot\cos\beta_1\cdot\cos\theta) \\ -l_1\cdot(\cos\alpha_1\cdot\cos\theta + \cos\alpha_1\cdot\cos\beta_1\cdot\sin\theta) \end{pmatrix} \tag{28}$$

where α_1 is the angle between $\overline{\mathbf{p}_1\mathbf{p}}$ and its projection $\overline{\mathbf{p}_1\mathbf{p}'}$ onto the plane π whose normal is obtained by a rotating $\mathbf{n}$ through $Rot(\mathbf{x},\theta)$ (see Figure 15-b). β_1 denotes the angle between $\overline{\mathbf{p}_1\mathbf{p}'}$ and the edge $\mathbf{e}$.

Similarly, during a contact with the point $\mathbf{p}_2$, the vector $\mathbf{r}_2 = \overline{\mathbf{pp}_2}$ is given by

$$\mathbf{r}_2 = \begin{pmatrix} l_2\cdot\cos\alpha_2\cdot\sin\beta_2 \\ l_2\cdot(\sin\alpha_2\cdot\sin\theta + \cos\alpha_2\cdot\cos\beta_2\cdot\cos\theta) \\ -l_2\cdot(-\cos\alpha_2\cdot\cos\theta + \cos\alpha_2\cdot\cos\beta_2\cdot\sin\theta) \end{pmatrix} \tag{29}$$

The transition from contact $\mathbf{p}_1$ to contact $\mathbf{p}_2$ occurs for $\theta = 0$. Computing the values of the two torques just before and after the *edge/face* contact gives

$$\begin{aligned} \tau_1 &= \lim_{\theta\to+0}(\mathbf{r}_1\times\mathbf{f}) = f\cdot(-l_1\cdot\cos\alpha_1\cdot\cos\beta_1, -l_1\cdot\cos\alpha_1\cdot\sin\beta_1, 0)^T \\ \tau_2 &= \lim_{\theta\to-0}(\mathbf{r}_2\times\mathbf{f}) = f\cdot(l_2\cdot\cos\alpha_2\cdot\cos\beta_2, -l_2\cdot\cos\alpha_2\cdot\sin\beta_2, 0)^T \end{aligned} \tag{30}$$

The variation of the torque across the contact then is

$$\Delta\tau = \tau_2 - \tau_1 = f\cdot\begin{pmatrix} l_1\cdot\cos\alpha_1\cdot\cos\beta_1 + l_2\cdot\cos\alpha_2\cdot\cos\beta_2 \\ l_1\cdot\cos\alpha_1\cdot\sin\beta_1 - l_2\cdot\cos\alpha_2\cdot\sin\beta_2 \\ 0 \end{pmatrix} \tag{31}$$

It is easy to see from Figure 15 that

$$\begin{aligned} &l_1\cdot\cos\alpha_1\cdot\cos\beta_1 + l_2\cdot\cos\alpha_2\cdot\cos\beta_2 = l\,, \quad \text{and} \\ &l_1\cdot\cos\alpha_1\cdot\sin\beta_1 = l_2\cdot\cos\alpha_2\cdot\sin\beta_2 \end{aligned} \tag{32}$$

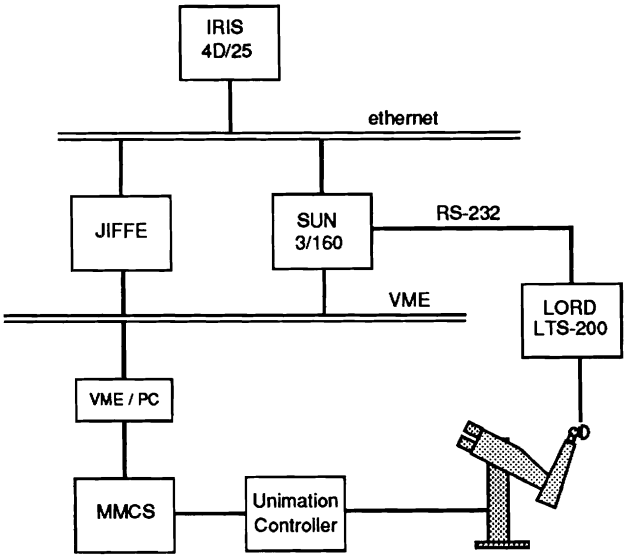

Figure 16: The hardware architecture of the experimental testbed.

and the change of contact therefore introduces a discontinuity on τ_x only. Moreover, the magnitude of this discontinuity is given by $f \cdot l$, regardless of where the torques are measured.

8 The experimental hardware/software testbed

The hardware architecture of our experimental setup is illustrated in Figure 16. The master manipulator in our scenario is a Unimation Puma 250 manipulator. It provides a backdrivable 6 d.o.f. "joystick" with a sufficient operating volume to afford the operator a true sense of spatial positioning and orienting. Digital hardware control for the master is provided by the Modular Motor Control System (MMCS) [5]. This system was designed and built at the laboratory as an experimental PC-bus based general purpose digital motor controller capable of controlling up to 16 independent actuators simultaneously. The MMCS hardware is interfaced to the original (factory-supplied) controller, whose sole remaining function is to provide power and the front panel interface. Finally, a custom-designed PC/VME adaptor connects MMCS's backbone to the VME bus.

Mounted at the wrist of the master is a 6 d.o.f. force/torque sensor (LORD Corp.,

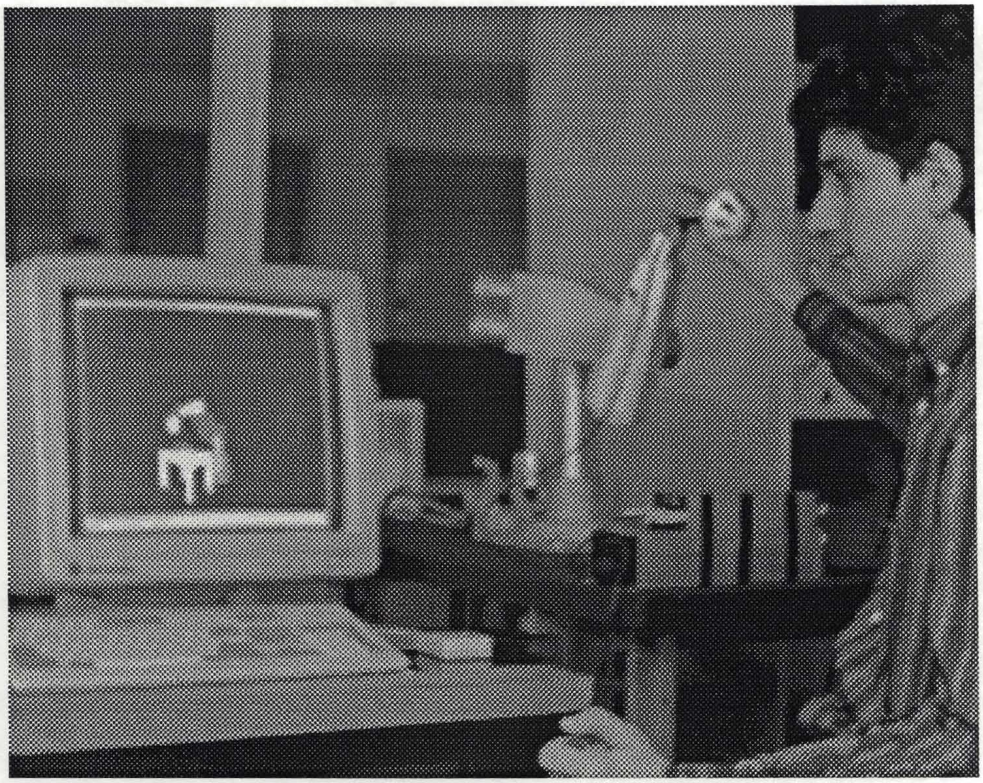

Figure 17: The operator's station.

LTS-200) enclosed within a "whiffle-ball" handle for convenient grasping by the operator (see Figure 17). The sensor is read over a serial line (RS-232) and provides information at a rate of approximately 30 Hz[17]. These readings are interpreted as incremental displacement/RPY Cartesian motion parameters of the sensor/handle assembly, and thus (through a transformation) of the master manipulator.

The computational engine of the system is JIFFE – a very fast, very-long-instruction-word floating point scalar processor delivering 20 real Mflops of computational power [1]. The processor has a standard VME interface and physically resides inside the Sun cage. It is fully C-programmable and supports most of the essential UNIX operating system facilities. JIFFE runs both the low-level joint servo code for the master at 500 Hz (PD control loop + gravity feed-forward), as well as the Cartesian level servo code, which runs at 30 Hz (Cartesian setpoint com-

[17]There is a substantial variation about this nominal bandwidth, largely due to the unpredictable UNIX-incurred delays in servicing the serial port accumulating incoming data.

putation and filtering as described in Section 5)[18]. It communicates with the Sun (model 3/160) via JIFFE-resident shared memory and (via the Sun and ethernet connection) with the Iris graphical workstation. The Sun currently serves mostly as the accumulator and processor of the force/torque information from the sensor and as an intermediary between JIFFE and the Iris. In later stages of the system design and implementation, the Sun will provide a console for an on-line task-level dialogue with the operator (see Section 2.6).

The incremental Cartesian displacements are appropriately scaled into the remote slave's workspace and sent (via ethernet) to the Iris, which tries to realize them in the simulated slave environment. In case of a collision (see Section 4), the offending motion is appropriately modified so as to stop colliding objects in a contact but non-penetrating configuration. The new constraint information is added to the existing set of constraints and communicated back to JIFFE, which in turn filters subsequent operator-supplied motion demands so as to not violate any of the current constraints on the motion of the slave (see Section 5). This filtered motion is then applied both to the graphical model of the slave and the master manipulator, thus providing a sense of kinesthetic feedback to the operator.

The link between JIFFE and the Iris is a bidirectional communication channel conveying filtered incremental Cartesian motions one way and new/updated constraint information the other way. The link is implemented as a standard UNIX socket communication channel (between the Sun and the Iris) and has a round-trip latency of only a few miliseconds. The graphical workstation is a 16 MIPS Personal Iris 4D-25 with a hardware turbo graphics option to boost its drawing speed. Even so, its ability to render shaded graphical images of modest complexity (*e.g.*, the slave manipulator plus an object) lags far behind its scalar number crunching capacity. We are able to obtain refresh rates of about 7 Hz for low complexity environments and only partial shading. However, it is now within the realm of possibility to obtain fully shaded graphic displays of relatively complex scenes at video rates using the latest Silicon Graphics hardware [2].

The software modeling environment for 3-D manipulation of articulated figures was provided by the Computer Graphics Laboratory at the University of Pennsyl-

[18]The Cartesian servo loop bandwidth is limited only by the rate at which force/torque sensor can provide new information, and not by the JIFFE's computational capacity.

vania [30].

9 Preliminary results and discussion

The current implementation of the system allows the operator to move the master and control the motion of the graphical model of the slave. The simulated slave can be brought into contact with the environment and the master is appropriately backdriven to provide a kinesthetic sense of contact to the operator. Recent experiments have shown that purely translational and sliding tasks can be performed with confidence and ease both for single and multiple constraining surfaces. The kinesthetic feedback to the operator feels natural and allows her to easily identify motion constraints and the shape of the constraining surfaces without looking at the display.

We are currently implementing the rotational (pivoting) contact motion mode. This should be completed in the near future and the resulting system should offer a versatile 3-dimensional 6 d.o.f. input device that will allow the operator to perform a variety of probing tasks, exploratory procedures, surface following and identification tasks, *etc.*

An important issue that arose during the preliminary experimentation with the system is that of *reindexing* the master. In our case the problem is perhaps even more acute as a general purpose manipulator is employed as the master device, and as such is not designed to meet the requirements of a versatile master. In particular, we found that due to a large number of kinematic motion singularities, a relatively small workspace volume around any given initial "home" position can be used for maneuvering. We have considered a variety of approaches to solve this problem. Perhaps the simplest solution is to offload the responsibility to reindex to the operator. In this scenario, the operator needs to identify that she is approaching a singular configuration (on the master) and reindex the manipulator accordingly. Reindexing could be done by hitting a switch or pressing a pedal, which in turn would put the arm in a free, gravity compensated mode and allow the operator to reposition the master to an arbitrary new (presumably singularity-free) configuration before resuming position (or velocity) servo mode. Clearly, the drawback of this approach lies in burdening the operator with having to be concerned with the kinematics and

the current state of the master. This is unacceptable as the operator's full attention may be required to control the task in progress.

In an effort to make the details of the implementation of the master device transparent to the operator, we next considered a reindexing scheme with a continuous drift back to the home position. The farther from home the operator has moved the master, the more strongly the master would tend to drift back. We implemented this scheme such that the magnitude of the restoring drift was exponentially related to the distance (for translations) and twist amplitude (for rotations) from the home position. Whereas this eliminated the need for operator's intervention in the reindexing process, it significantly impaired the spatial resolution of the master's motion, which in turn obscured the kinesthetic feedback effects during contact motion.

A third approach that we considered was one where the master would monitor its own motions and alert the operator (by beeping, for instance) when it is approaching a singular configuration. It would then decouple its motions from the simulator display, return to the home position, and alert the operator that she may proceed with the task.

While this third approach has not been implementationally verified, it may well offer the best compromise between the existing requirements and constraints. This may be especially true if this approach is combined with the first method, thus allowing the operator to change the home configuration dynamically (at will) during the execution of the task.

Our current goal is to complete the implementation of the kinesthetic feedback features as described in Section 5, implement a satisfactory reindexing scheme, and concentrate our efforts on the problem of automatically partitioning the task in progress and extracting the relevant parameters to generate a stream of robust elementary task-level instructions to the remote slave.

A Appendix

A.1 Notation

Both 3 and 6-dimensional vector quantities are denoted as boldface (lower-case) characters with an optional preceding superscript indicating the coordinate frame with respect to which they are given, *i.e.*, $\mathbf{a}, {}^{B}\mathbf{n}$, *etc.*

A coordinate frame is specified by a triple of mutually orthogonal unit vectors, with an optional indication of the frame's origin, *i.e.*,

$$\mathcal{F} = \{\mathbf{x}, \mathbf{y}, \mathbf{z}\} \qquad \text{or} \qquad \mathcal{F} = \{\mathbf{p}\,;\, \mathbf{x}, \mathbf{y}, \mathbf{z}\} \tag{1}$$

Rotational matrices are denoted by upper-case boldface letters with optional superscripts and subscripts indicating which two coordinate frames they relate, *e.g.*, the matrix ${}^{B}\mathbf{R}_{F}$ describes the orientation of frame $\mathcal{F}_F$ *w.r.t.* $\mathcal{F}_B$.

Finally, we occasionally use the following non-standard vector notation

$$\begin{aligned} \mathbf{a}^{*} &= \frac{\mathbf{a}}{\|\mathbf{a}\|} \\ \tilde{\mathbf{a}} &= -\mathbf{a} \end{aligned} \tag{2}$$

A.2 Coordinate frames and rotational matrices

Let $\mathcal{F}_A$ be a coordinate frame and let ${}^{A}\mathbf{y}$ and ${}^{A}\mathbf{z}$ be two mutually orthogonal unit vectors, expressed in $\mathcal{F}_A$'s coordinates. Then the two vectors can be thought of as defining a second coordinate frame

$$\mathcal{F}_B = \left\{ \left({}^{A}\mathbf{y} \times {}^{A}\mathbf{z} \right), {}^{A}\mathbf{y}, {}^{A}\mathbf{z} \right\} \tag{3}$$

whose origin is coincident with $\mathcal{F}_A$'s and whose orientation *w.r.t.* $\mathcal{F}_A$ is given by the rotational matrix

$${}^{A}\mathbf{R}_{B} = \begin{bmatrix} | & | & | \\ \left({}^{A}\mathbf{y} \times {}^{A}\mathbf{z} \right) & {}^{A}\mathbf{y} & {}^{A}\mathbf{z} \\ | & | & | \end{bmatrix} \tag{4}$$

Moreover, the rotational matrix ${}^{A}\mathbf{R}_{B}$ can be used to map (rotate) an arbitrary vector ${}^{B}\mathbf{r}$ expressed in $\mathcal{F}_B$'s coordinates into its corresponding description in $\mathcal{F}_A$ coordinates, *i.e.*,

$${}^{A}\mathbf{v} = {}^{A}\mathbf{R}_{B} * {}^{B}\mathbf{v} \tag{5}$$

Likewise,

$$^{B}\mathbf{v} = {}^{B}\mathbf{R}_{A} * {}^{A}\mathbf{v} \tag{6}$$

where ${}^{B}\mathbf{R}_{A} = \left({}^{A}\mathbf{R}_{B}\right)^{-1}$.

A.3 Mapping rotations between frames

Let $\mathcal{F}_A$ and $\mathcal{F}_B$ be two arbitrary coordinate frames and let ${}^{A}\mathbf{r} = \theta \cdot {}^{A}\mathbf{k}^{*}$ denote a rotation expressed in $\mathcal{F}_A$'s coordinates. The same rotation can be expressed in frame $\mathcal{F}_B$ as

$$^{B}\mathbf{r} = \theta \cdot {}^{B}\mathbf{k}^{*} = \theta \cdot \left({}^{B}\mathbf{R}_{A} * {}^{A}\mathbf{k}^{*}\right) = {}^{B}\mathbf{R}_{A} * {}^{A}\mathbf{r} \tag{7}$$

Alternatively, if the rotation ${}^{A}\mathbf{r}$ is expressed as a triple of roll/pitch/yaw parameters, *i.e.*, ${}^{A}\mathbf{r} = (\theta_x, \theta_y, \theta_z)$, the equivalent rotation expressed *w.r.t.* $\mathcal{F}_B$'s coordinates is obtained by

- assembling a rotational matrix representing ${}^{A}\mathbf{r}$

$$^{A}\mathbf{R} = \text{RPYtoM}\left({}^{A}\mathbf{r}\right) \tag{8}$$

- transforming this matrix to $\mathcal{F}_B$'s coordinates

$$^{B}\mathbf{R} = \left({}^{A}\mathbf{R}_{B}\right)^{-1} * {}^{A}\mathbf{R} * {}^{A}\mathbf{R}_{B} \tag{9}$$

- extracting the new triple of RPY parameters

$$^{B}\mathbf{r} = \text{MtoRPY}\left({}^{B}\mathbf{R}\right) \tag{10}$$

See [26] for a detailed discussion of the RPYtoM and MtoRPY conversion operators. For the linear-algebraic basis of these operations, the reader is referred to [23].

A.4 Displacement of a point due to motion of the frame

Let $\mathcal{F}$ be a coordinate frame undergoing a translational and rotational motion $\Delta\mathbf{d}_{\mathcal{F}} = (\mathbf{t}, \mathbf{r})$. Then the resulting displacement of a point located at $\mathbf{p}$ *w.r.t.* the origin of $\mathcal{F}$ is

$$\Delta\mathbf{d}_{\mathbf{p}} = (\mathbf{t} + (\mathbf{R} * \mathbf{p}) - \mathbf{p}, \mathbf{r}) \tag{11}$$

where $\mathbf{R} = \text{RPYtoM}(\mathbf{r})$, and $\Delta\mathbf{d}_{\mathbf{p}}$ is given *w.r.t.* to the original frame $\mathcal{F}$.

References

[1] R. L. Andersson. Computer architectures for robot control: a comparison and a new processor delivering 20 real Mflops. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1162–1167, 1989.

[2] A. K. Bejczy and W. S. Kim. Predictive displays and shared compliance control for time-delayed telemanipulation. In *IEEE International Workshop on Intelligent Robots and Systems*, July 1990. Ibaraki, Japan.

[3] C. I. Connolly, J. L. Mundy, J. R. Stenstrom, and D. W. Thompson. Matching from 3-D range models into 2-D intensity scenes. In *IEEE First International Conference on Computer Vision*, pages 65–72, 1987.

[4] Robert W. Corell. Closing summary. In *The Fifth International Symposium on Unmanned, Untethered Submersible Technology*, pages 618–625, Marine Systems Engineering Laboratory, University of New Hanpshire, June 1987.

[5] Peter I. Corke. *A new approach to laboratory motor control: The modular motor control system.* Technical Report, University of Pennsylvania, Philadelphia, PA, 1989.

[6] Ballard R. D. A last long look at Titanic. *National Geographic*, 170/6, December 1986.

[7] B. Faverjon and P. Tournassoud. A local based approach for path planning of manipulators with a high number of degrees of freedom. In *IEEE International conference on Robotics and Automation*, 1987.

[8] W. R. Ferrell. Delayed force feedback. *IEEE Trans. Human Factors in Electronics*, 449–455, October 1966.

[9] W. R. Ferrell. Remote manipulation with transmission delay. *IEEE Trans. Human Factors in Electronics*, 1965. HFE-6, 1.

[10] W. R. Ferrell and T. B. Sheridan. Supervisory control of remote manipulation. *IEEE Spectrum*, 81–88, October 1967. 4-10.

[11] E.G. Gilbert, D.W. Johnson, and S.S Keerthi. A fast procedure for computing the distance between objects in three-space. In *IEEE International conference on Robotics and Automation*, 1987.

[12] Ray C. Goertz. Manipulators used for handling radioactive materials. In E. M. Bennett, editor, *Human Factors in Technology*, chapter 27, McGraw Hill, 1963.

[13] Black J. H. *Factorial study of remote manipulation with transmission time delay.* Master's thesis, MIT, Department of Mechanical Enginnering, 1971.

[14] S. Hayati and B. Wilcox. Manipulator control and mechanization: a telerobot sub-system. In G. Rodriguez, editor, *Proceedings of the Workshop on Space Telerobotics*, pages 219 - 227, JPL, July 1987.

[15] Martin Herman. Generating detailed scene descriptions from range images. In *IEEE International Conference on Robotics and Automation*, pages 426–431, 1984.

[16] G. Hirtzinger, J. Heindl, and K. Landzettel. Predictive and knowledge-based telerobotic control concepts. In *IEEE International Conference on Robotics and Automation*, pages 1768–1777, 1989.

[17] Hirochika Inoue. Computer controlled bilateral manipulator. *Bulletin of the Japanese Society of Mechanical Engineers*, 14(69):199–207, 1971.

[18] Matthew T. Mason. Compliance and force control for computer controlled manipulators. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(6):418–432, June 1981.

[19] Matthew T. Mason. Mechanics and planning of manipulator pushing operations. *The International Journal of Robotics Research*, 1986.

[20] Matthew T. Mason. On the scope of quasi-static pushing. In *Robotics Research: The Third International Symposium*, October 1985.

[21] Matthew T. Mason and J. K. Salisbury. *Robot hands and the mechanics of manipulation.* MIT Press, Cambridge, Massachusetts, 1985.

[22] NASA. *Flight Telerobotic Servicer, Tinman Concept, In-house Phase B Study – Final Report.* Technical Report, Goddard Space Flight Center, Greenbelt, MD, September 1988. Volumes I and II.

[23] Evar D. Nering. *Linear algebra and matrix theory.* John Wiley & Sons, Inc., New York, 2 edition, 1970.

[24] Marilyn Niksa. Aluminum-oxygen batteries as power sources for submersibles. In *The Fifth International Symposium on Unmanned, Untethered Submersible Technology*, pages 121–127, Marine Systems Engineering Laboratory, University of New Hanpshire, June 1987.

[25] M. Noyes and T. B. Sheridan. A novel predictor for telemanipulation through a time delay. In *Proceedings of the 20th Annual Conference on Manual Control*, Moffett Field, CA: NASA Ames Research Center, 1984.

[26] Richard P. Paul. *Robot Manipulators: Mathematics, Programming, and Control. MIT Press Series in Artificial Intelligence*, MIT Press, Cambridge, Massachusetts, 1981.

[27] Richard P. Paul and Bruce Shimano. Compliance and control. In *Proceedings of the Joint Automatic Control Conference*, pages 694–699, 1976.

[28] M. A. Peshkin and A. C. Sanderson. The motion of a pushed, sliding workpiece. *IEEE Journal of Robotics and Automation*, April 1988.

[29] M. A. Peshkin and A. C. Sanderson. Planning robotic manipulation strategies for sliding objects. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 696–701, 1987.

[30] Cary B. Phillips and Norman I. Badler. Jack: a toolkit for manipulating articulated figures. In *Proceedings of ACM/SIGGRAPH Symposium on User Interface Software*, Banff, Alberta, Canada, 1988.

[31] C. Sawada, H. Ishikawa, K. Kawase, and M. Takata. Specification and generation of a motion path for compliant motion. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 808–815, 1989.

[32] F. Schenker, R. French, and A. Sirota. The NASA/JPL telerobot testbed : an evolvable system demonstration. In *IEEE International Conference on Robotics and Automation*, March 1987.

[33] David R. Smith and Takeo Kanade. Autonomous scene description with range imagery. *Computer Vision and Graphics Image Processing*, 31, September 1985.

[34] Lawrence Stark. Telerobotics: display, control, and communication problems. *IEEE Journal of Robotics and Automation*, RA-3(1), February 1987.

[35] Daniel E. Whitney. Historical perspective and state of the art in robot force control. *The International Journal of Robotics Research*, 6(1), 1987.