

Programming Contact Tasks Using a Reality-based Virtual Environment Integrated with Vision

John E. Lloyd , Jeffrey S. Beis, Dinesh K. Pai, and
David G. Lowe

Abstract—We present an integrated system in which an operator uses a simulated environment to program part-mating and contact tasks. Generation of models within this virtual environment is facilitated using a fast, occlusion-tolerant, 3D grey-scale vision system which can recognize and accurately locate objects within the work site. A major goal of this work is to make robotic programming easy and intuitive for untrained users working with standard desktop hardware. Simulation can help accomplish this, offering the ease-of-use benefits of “programming by demonstration”, coupled with the ability to create a programmer-friendly virtual environment. Within a simulated environment, it is also straightforward to track and interpret an operator’s actions. The simulator models objects as polyhedra and implements full 3D contact dynamics, making it easy to place and manipulate objects using input from a simple 2D mouse. When a manipulation task is completed, local planning techniques are used to turn the virtual environment’s motion sequence history into a set of robot motion commands capable of realizing the prescribed task.

I. INTRODUCTION

This paper describes an integrated system which uses virtual-reality simulation to program robotic part-mating and contact tasks, developed at the Computer Science Department of the University of British Columbia (UBC). It includes a vision system for rapidly creating models of a work site, a task simulator that allows objects to be manipulated within the simulated environment, and a program generator that turns the simulated actions into a sequence of robotic motion commands capable of realizing the desired task.

In developing this system, our aim has been to make robot programming very easy, particularly for non-specialists using standard desktop computer hardware. Our emphasis on tasks involving contact is deliberate, since commercial manipulator systems still provide only minimal support for such operations. We suspect that the complexity of programming contact-based tasks is an important reason for this.

We believe that using a simulated virtual environment to directly demonstrate the required contact task can help overcome many programming difficulties. Reasons for this include:

1. It eliminates the need for tedious textual descriptions (such as “place face A on top of face B”).
2. Simulated environments can be augmented with operator aids such as virtual fixtures. Interactions can also focus on the direct manipulation of objects, permitting a more “task-centric” programming model.
3. It is easier within a simulated environment to track an operator’s actions and discern intentions, since the state of everything is known and does not have to be estimated using sensors.

In contrast with most virtual environments, objects in our environment are identified and located automatically using vision,

John Lloyd is currently with the Department of Mechanical Engineering, Katholieke Universiteit Leuven, Leuven, BELGIUM

The other authors are with the Department of Computer Science, University of British Columbia, Vancouver, B.C., CANADA

making for a more accurate representation of the real environment. Such a “reality-based” virtual environment is crucial for meaningful robot programming.

At present, our system’s task domain consists of a puzzle of wooden blocks which can be assembled within a rigid frame. This domain is simple but leads to a wide range of possible contact interactions. Part mating involving “tight fits” is not currently supported but is part of our ongoing research. A very basic demonstration of the system is given in Fig. 1, illustrating the placement of a block into a corner (for an MPEG video of this, see <http://www.cs.ubc.ca/spider/pai/telerobotics.html>). A more complex type of task involves dragging a block around the outside of a corner while contact is maintained (Fig. 2).

A. System requirements and overview

A simulation-based robot programming system requires several principal modules:

1. *Model Generator*: builds and maintains the work site model used by the simulator.
2. *Task Simulator*: permits manipulations within the simulated environment, using dynamics that are easy and intuitive for the operator and which facilitate task specification.
3. *Program Generator*: uses the actions within the virtual environment to create robot motion commands capable of realizing the prescribed task.
4. *Execution Monitor*: verifies task execution at the robot site. On-line information may also be used to determine corrections to the work site model.

The first three of these components are well-developed within the UBC system and form the main focus of this paper.

Model generation is realized using a model-based grey-scale vision system that can rapidly and robustly recognize and locate objects characterized by straight-line edge features. Matches are performed using localized groups of edge features, enabling the vision system to tolerate a moderate amount of occlusion. Recognition of several objects can be achieved within about five seconds, and the object pose information is generally accurate to within a pixel. Recognized objects are displayed in a separate “feedback window” from which the operator can select them for inclusion into the work site model.

The task simulator lets an operator select objects within the simulated environment and move them around, using input from a standard 2D mouse. A mouse was chosen deliberately, since it is cheap, ubiquitous, and fits with our above mentioned goal of making programming accessible to users with standard desktop computer hardware. Graphical fixtures are used to map the 2D mouse inputs into 3D spatial motions. The simulated environment uses first-order dynamics, since this is (a) easy to compute, (b) intuitive for the operator, (c) qualitatively similar to a

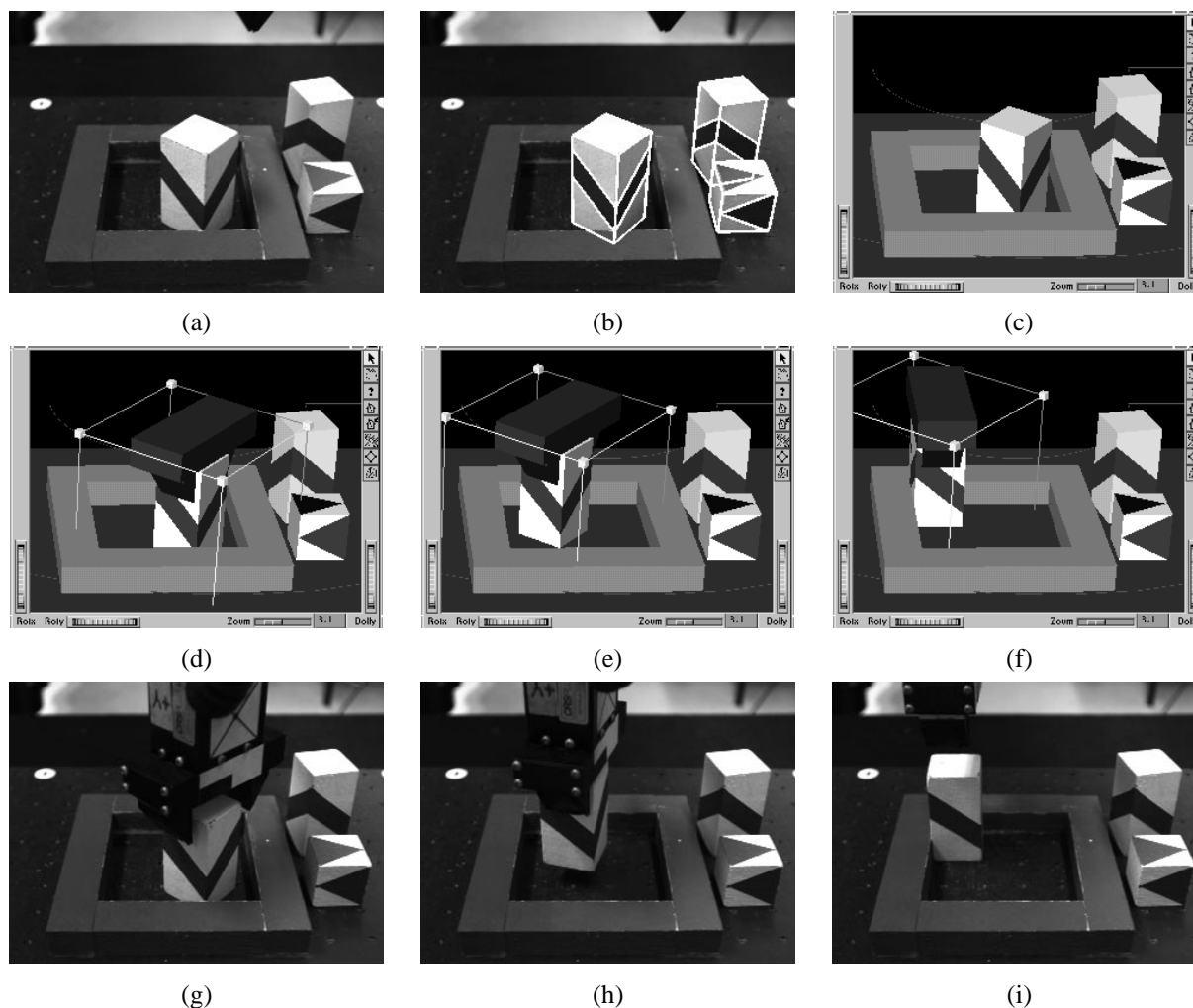


Fig. 1. Simple example showing the placement of a block in a corner. Blocks within the initial work site (a) are located by the vision system (b, with matching edges outlined in white), and then used to update the work site model, graphically rendered in (c). An operator can then select an object by “clicking” on it, causing it to be encased within a “dragger fixture” (d). Clicking and dragging on a face of this fixture creates a virtual force in the plane of the face, which in turn causes the object to move (e). As the operator drags the object into the corner, contact is made with the corner faces, causing the block to align and settle into the corner easily, with no finesse required (f). The goal of the task, in terms of both nominal object position and the required contact state, is known directly from state information available to the simulator. When satisfied with the block’s position, the operator issues a confirmation command, which causes the simulated motion sequence to be turned into robot motion commands which realize the requested operation (g)-(i), with impedance control used to facilitate the necessary contacts.

system dominated by friction, and (d) stable. Contact interactions are modeled, letting the manipulated object, or *workpiece*, bump into, slide along, and align itself with other objects. These interactions, combined with the first-order dynamics and graphical fixtures, make it quite easy for the operator to perform part mating and placement within the virtual environment. The simulator is also capable of enforcing contact between the workpiece and designated *capture* objects, to enable the specification of motions involving continuous contact.

After an object has been manipulated to a desired state within the virtual environment, the program generator is invoked to produce a set of robotic commands to implement the specified action. This is done by taking the complete motion sequence specified by the operator, and simplifying it, using local planning (potential field) techniques, so as to remove extraneous motion segments and move it away from objects with which contact is not

required. A key feature of our simplification technique is that it can be constrained to *preserve* desired contacts, as in the cornering example of Fig. 2. The simplified path is then turned into a set of spatially-linear robot motion commands, with each path segment possibly associated with one or more contact states. Contact is maintained using standard impedance control techniques.

The last module, the execution monitor, is presently implemented in a simple way using force monitoring to verify the occurrence of the required contact states. More robust monitoring, and the use of contact information to update the work site model, is the subject of ongoing investigation.

B. Connection to telerobotics

Although our focus is on model acquisition and programming, our system can be used to perform telerobotic manipulation, and

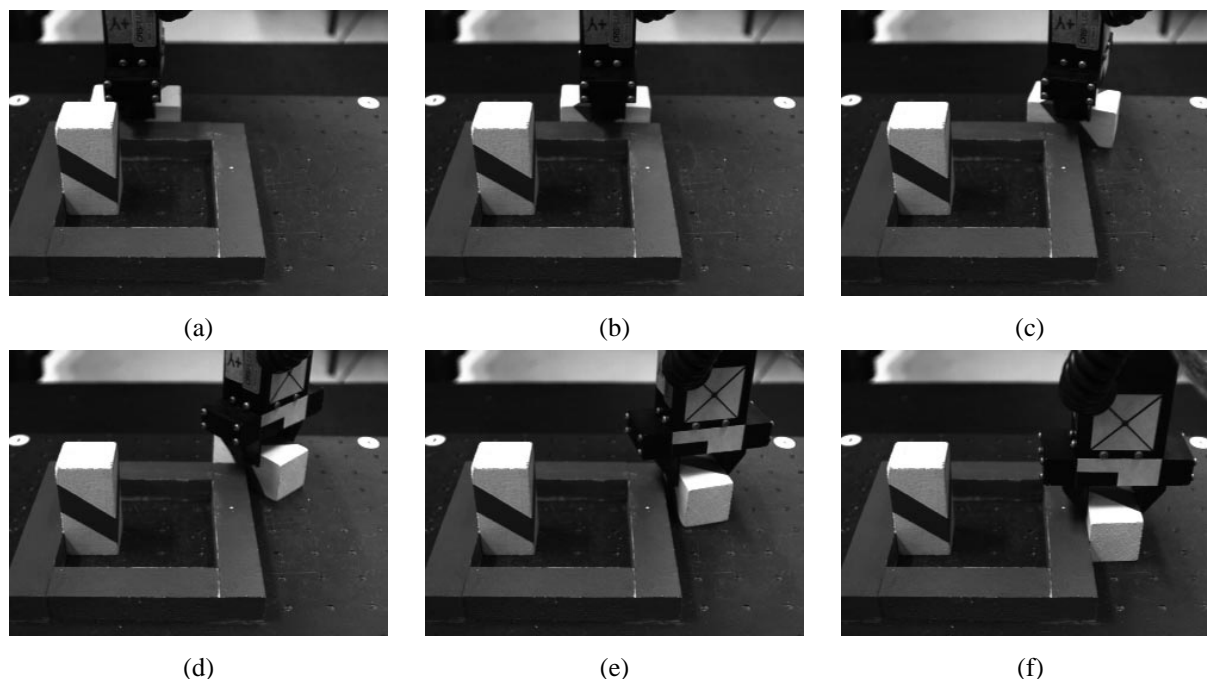


Fig. 2. Sequence showing a more complex contact task where a block is dragged around a corner of the puzzle frame while contact with the frame is maintained.

was in fact inspired by previous work in that area (Section II). All links between the operator site and the work site (Section III) are implemented using TCP/IP sockets, permitting remote operation over the Internet. At a conference in Montreal in June 1996, an earlier version of the system was successfully demonstrated in exactly this way, with the operator site located in Montreal and the work site located in Vancouver.

C. Contributions

Specific contributions of this work include:

1. Showing that unstructured operator interactions within a virtual environment (with first-order contact dynamics) can be used as a basis to program robotic part-mating and contact tasks, with contact constraints actually serving to make part placement easier.
2. The system architecture: a vision system automatically locates elements within the scene, which are then edited by the operator into the virtual environment. Objects within the virtual environment are manipulated to specify tasks, and the final robotic commands are produced by transforming and simplifying the operator interaction sequence;
3. Demonstrating the ability to use on-line vision to provide fast, occlusion-tolerant identification and location of structured elements within the scene, without a priori estimates of object position;
4. Using barrier potentials (Section V-B) to improve the performance of the contact dynamics simulation;
5. Using a local potential approach, combined with dynamic simulation, to transform the operator interaction sequence into a set of robotic motion commands (Section VI);
6. Using 3D graphical dragger fixtures, not to position objects directly, but to impart virtual forces on them, with the final mo-

tion then resulting from dynamical simulation (Section V).

D. Outline

The rest of this paper is arranged as follows. Related work is discussed in Section II, and a more detailed description of the system components and hardware is given in Section III. The vision system, task simulator, and program generator are then presented in Sections IV, V, and VI, respectively. Experimental observations are given in Section VII.

II. RELATED WORK

The work described here is closely connected to the *teleprogramming* work of Funda, Sayers, and others [1], [2], where operator interaction with a simulated environment is used to overcome problems of time delay. Teleprogramming was predated by the use of predictive graphical simulation [3], [4]. The introduction of synthetic “fixtures” into the operator’s display to assist in task specification has also been explored [5], [6], and virtual reality simulation has been investigated as a platform on which fine motion task skills can be learned [7], [8].

Commands sent to the remote site of teleprogramming systems tend to be fairly low level. The ability to use higher level commands which ask the remote manipulator to achieve a particular contact state (recovering if necessary from intervening contact states) is investigated in [9], using a Petri-Net-based contact state model.

Model acquisition in most telerobotic systems is currently achieved using extensive operator interaction. Typically, this involves helping the vision system by manually indicating object features in a video image of the remote site [10], [11].

Automatic model generation using computer vision has been limited by the ability of systems to accurately identify and lo-

cate 3-D objects, particularly in the presence of clutter and partial occlusion. Although there has been a long history of research on 3-D object recognition [12], [13], it is only recently that new approaches to model indexing have allowed such approaches to be sufficiently fast and reliable for integration with a real-time telerobotics system [14], [15]. This model-based approach contrasts with appearance-based recognition [16], which is based on directly matching image appearance, and can therefore model more general object classes but is less robust to image clutter and illumination changes and does not perform precise object localization.

Finally, in the last few years there have been a number of projects making teleoperated robotic systems of various kinds available to casual users on the World Wide Web; see, for example, [17], [18], [19].

III. SYSTEM DESCRIPTION

A block diagram of the UBC system is shown in Fig. 3. The system is divided into a *work site*, consisting of a robot, its controller, and the vision module, and an *operator site*, comprising the operator interface, task simulator, model editor, and program generator.

A. Work site

The work site (Fig. 4) contains a 6 DOF CRS A460 robot (Puma-type geometry), controlled at the lowest level by 1 KHz joint servos, which are in turn driven by a *task controller* implemented using the Robot Control C Library (RCCL) [20] running in real-time on a Sun Sparc 5. The task controller accepts Cartesian motion commands from the operator site, and generates the required trajectories at 100 Hz. It uses force-sensor feedback to implement both guarded moves and a position-based impedance control similar to that described in [21].

The vision module continuously collects images from a single monochrome camera and processes them using a model-based vision algorithm (Section IV) to locate objects in the scene. The objects and their positions are continuously sent back to the operator site, along with the raw camera image.

B. Operator site

The operator site consists of an SGI Indy with a 15 Mflop CPU. It hosts a model of the work site environment, with which the operator interacts, using a mouse, via the *task simulator* (Section V). Model data includes polyhedral representations of the work space objects, plus kinematic and geometric information about the robot manipulator (other dynamical information, such as friction and stiffness models, may be added later if required). Objects recognized by the video/vision module are displayed in a *feedback window*, from which the operator can selectively edit them into the work site model via the *model editor* (Section IV-C). The raw camera image from the work site is displayed in a *camera image window*. The model editor, together with the vision system, constitute the *model generator* described in Section I-A. The work site model itself can be viewed from any viewpoint through the *model viewing window*, implemented using the SGI 3D modeling package Open Inventor [22]. The different viewing windows are shown in Fig. 5. Lastly, a *program generator* (Section VI) creates the robot motion commands required

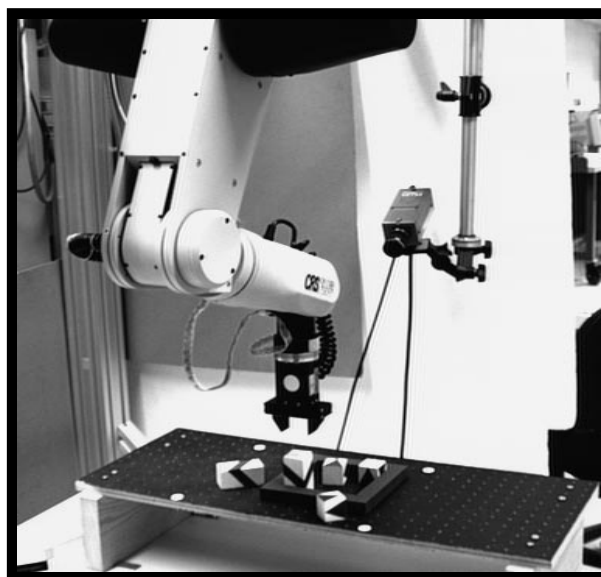


Fig. 4. Remote site, showing the robot, camera, and work area.

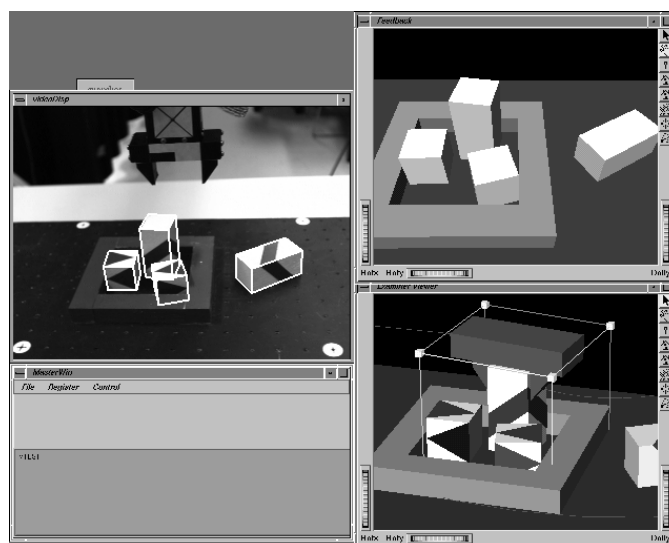


Fig. 5. Typical screen layout on the operator site workstation. Clockwise from top-left: camera image window, feedback window, model viewing window (with tall block being manipulated by a dragger box), and textual interface window.

to realize specific tasks and sends them to the task controller at the work site.

IV. MODEL GENERATION

The principal component of the model generator is the vision system, which must be able to provide accurate identification and location of objects in the work space. This is done using the model-based recognition system of Beis and Lowe [15], [23], which uses a novel form of rapid indexing to recognize 3D objects from any point of view in single 2D images.

A model must be provided for each object type which specifies the 3D locations of all prominent edges and the visibility direction of each surface. The vision system is currently restricted to

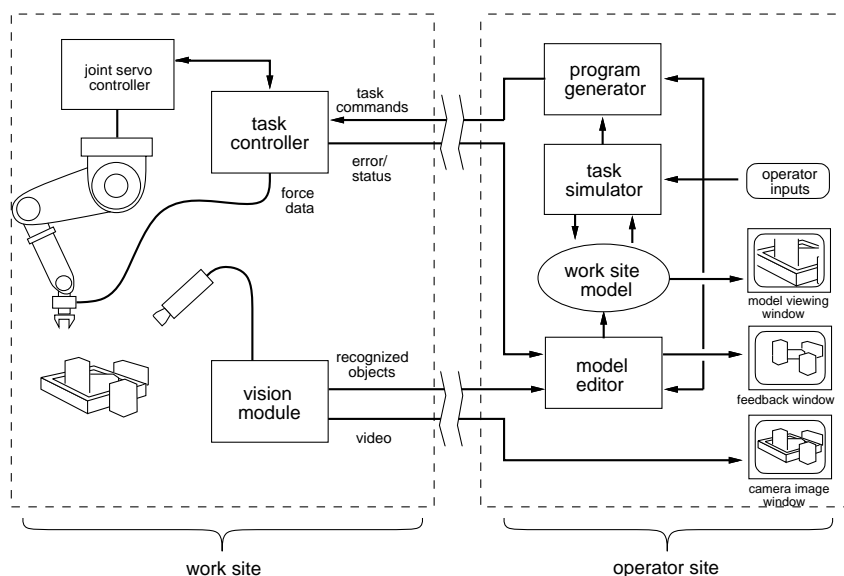


Fig. 3. System architecture.

using straight edges of the objects in the recognition process, but the same approach can be extended to use arbitrary curved surfaces [24]. To increase the robustness of the vision system and to make the objects easily distinguishable, we painted some linear markings on the surfaces of the blocks. However, these surface markings are not required, and equivalent robustness could be achieved by using more than one camera to obtain multiple views of the natural object shapes.

A. Recognition and matching

The recognition process begins by finding all linear edges in an image and identifying groups of edges that are connected to one another or are nearby and parallel. Groups of 4 or more line segments (see Fig. 6) are used to generate a vector of measurements giving the relative lengths or angles between the lines. This “index” vector is invariant to 2D translation, rotation and scaling, but will vary with the projection of different 3D rotations of the object. A precomputed index covering a sample of all 3D object rotations is used to estimate the probability that a particular vector was produced by a particular object. One feature of this approach is that index access time remains very rapid even as the dimensionality of the feature vectors is increased [23]. Full details of this indexing approach are given in [15].

Once a tentative interpretation has been made for some image features, it is possible to estimate the object location and orientation in 3D [24]. This is used to predict the locations of other object edges in the image and obtain further correspondences. At each stage, the solution for object location and orientation in 3D is performed with a least-squares fit minimizing residuals in predicted versus actual image locations. So, while the current system uses only straight edges, models might easily contain other feature types with location information, to aid in verification and pose determination. The solution for object pose is substantially over-determined, which means there is little likelihood that an incorrect correspondence will be found to fit more than a few image features. If a good fit is not found for a number of image

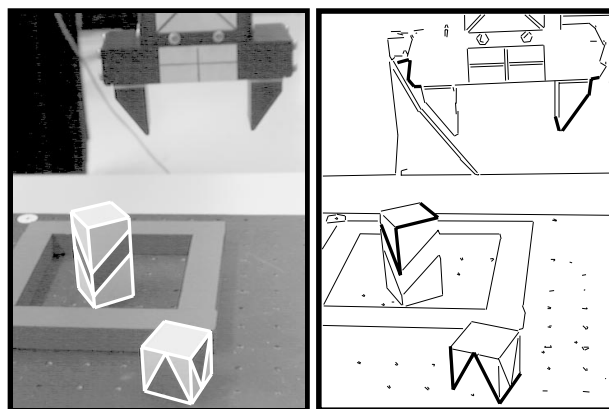


Fig. 6. Left frame shows cropped image of work space overlaid with wire-frame models at positions determined by the recognition algorithm. Right frame shows edge-detected image with examples of correct and incorrect feature groupings used in the indexing process.

edges, then the match is rejected and a new indexing hypothesis is used.

B. Speed, accuracy, and calibration

The full recognition process currently requires about 5 seconds running on a 15 Mflop SGI Indy. A number of optimizations could be made to improve this time.

Image lines are determined through a least-squares fit to each pixel along an edge, and model location is based on a least-squares fit to these lines. Therefore, accuracy is usually precise down to the pixel level of the image, although its mapping to the 3D world depends on the camera location and optics. With a single camera, the location of the object parallel to the camera image plane is more precise than location towards and away from the camera. A similar recognition approach using two or more cameras could achieve full accuracy in all dimensions.

A standard pin-hole camera model is assumed. Intrinsic pa-

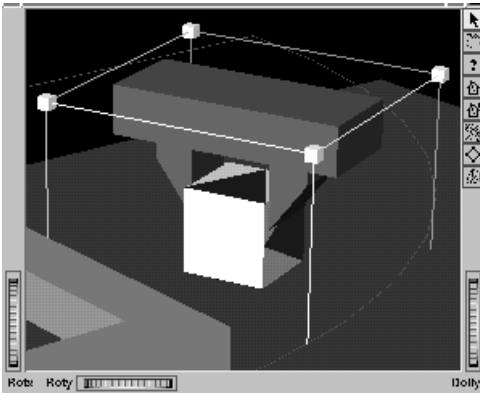


Fig. 7. Dragger box and gripper image surrounding a block to be manipulated. Dragging the mouse cursor along one of the planes of the box causes a displacement parallel to the plane, which is converted into a “virtual force” acting on the workpiece’s center.

rameters (focal-length and radial distortion) were calibrated off-line using the algorithm in [25]. The camera position relative to the work space was calibrated manually by having the operator identify known work site features in the camera image.

C. Model editing

Objects recognized by the vision system are displayed in the feedback window and also drawn (using wire-frame overlays) in the camera image window (Fig. 3).

The work site model itself is updated under the control of the operator, via the model editor. Usually this is done between task specifications, and only when necessary. By examining the recognition data displayed within the feedback and camera image windows, the operator can make a final judgement as to the data’s reliability. Specific objects can then be selected for inclusion within the work site model by clicking on them within the feedback window. Similarly, work site model objects can be selected for deletion by clicking on them within the model viewing window.

One anticipated function of the model editor, not yet implemented, is to ensure the consistency of the recognition data with physical reality. For example, the positions of recognized objects should be adjusted, when necessary, to ensure that they don’t interpenetrate.

V. TASK SIMULATION

The task simulator enables the operator to manipulate an object (or *workpiece*) within the virtual environment so as to easily specify different tasks. Often, this entails moving the workpiece to a destination involving contact with one or more objects (since the workpiece must rest on something, all destination states will entail at least one contact). The cornering task of Fig. 1 typifies this. A more complex task might involve maintaining contact during motion, as shown in Fig. 2.

Such actions are created by moving the workpiece around in the virtual environment under the influence of a contact model combined with first-order dynamics. This permits the workpiece to bump into, slide along, and align itself with other objects. This, in turn, makes it very easy for the operator to place the

workpiece into a desired contact state. First-order dynamics is used for the reasons mentioned in Section I-A. To create motions involving continuous contact, the operator can use the mouse to select certain objects as *capture* objects. Once the workpiece makes contact with a capture object, its motion is constrained so that contact with that object is maintained. This is done by keeping the Euclidean distance between the workpiece and the object within a certain small value (using a barrier function, as described at the end of Section V-B). The contact constraint can be removed by deselecting the capture object.

After the operator completes a task, she signals this to the system, which then invokes the program generator (Section VI) to create a sequence of robotic commands capable of realizing the task at the work site.

Interaction with the simulated environment occurs through the model viewing window. Using the mouse, the operator selects a workpiece to be moved by clicking on it. A simulated gripper then appears, showing how the workpiece will be grasped, along with a graphical “dragger fixture” (presently, a box) that maps 2D mouse inputs into 3D spatial motions and permits the workpiece to be moved about (see Fig. 7 and Fig. 1). Rendering only the gripper preserves the “task-centric” focus of the operator’s actions; more proximal parts of the robot could be rendered if necessary.

A. Contact dynamics

The workpiece is “attached” to the dragger box by a virtual spring, so that deviations from its nominal position within the box give rise to a virtual force \mathbf{f}_a and moment \mathbf{m}_a acting on it (Fig. 8).

Euclidean distances between the workpiece and other objects are continuously monitored using I-COLLIDE [26]. Objects closer than ϵ_c are assumed to be in contact with the workpiece, and then information provided by I-COLLIDE is used to determine a suitable finite set of n contact points \mathbf{p}_i and normals \mathbf{n}_i for modeling the contact¹.

When workpiece contact occurs, the system computes the reaction forces \mathbf{f}_i arising along the contact normals in response to the virtual forces (\mathbf{f}_a and \mathbf{m}_a) acting on the workpiece (Fig. 9). Although this can be a difficult problem to solve in general [28], the fact that we have a finite number of contact points, are using first-order dynamics, and are ignoring friction makes the computation, which we now explain, fairly easy.

First-order dynamics implies that the net force \mathbf{f} and moment \mathbf{m} acting on the workpiece are proportional to the translational and rotational components of its spatial velocity (\mathbf{v} and $\boldsymbol{\omega}$), according to

$$\mathbf{v} = d_t \mathbf{f} \quad \text{and} \quad \boldsymbol{\omega} = d_r \mathbf{m}, \quad (1)$$

where d_t and d_r are prescribed damping constants. Now, at each contact point \mathbf{p}_i , the normal force \mathbf{f}_i and the velocity component \mathbf{v}_i along the normal are given by $\mathbf{f}_i = c_i \mathbf{n}_i$ and $\mathbf{v}_i = a_i \mathbf{n}_i$, where c_i and a_i are scalars which must satisfy

$$a_i \geq 0, \quad c_i \geq 0, \quad \text{and} \quad a_i c_i = 0. \quad (2)$$

¹For polyhedral object models, face-face or edge-face contacts can be represented using a finite set of point contacts; see [27].

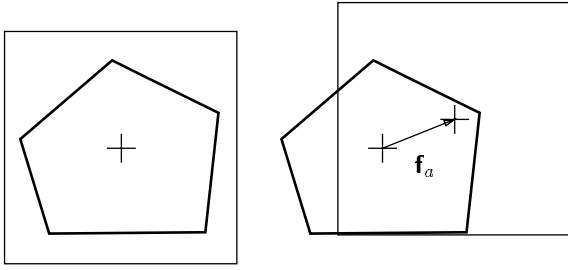


Fig. 8. Displacement of the workpiece relative to the dragger box creates a virtual force on it (such as \mathbf{f}_a , shown here).

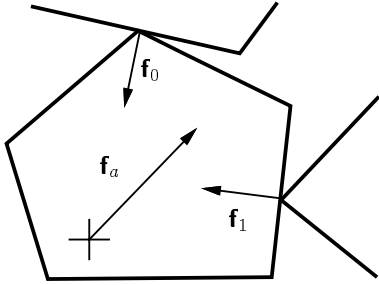


Fig. 9. Contact of the workpiece with other objects gives rise to normal reaction forces (\mathbf{f}_0 and \mathbf{f}_1).

Collecting a_i and c_i into vectors $\mathbf{a}, \mathbf{c} \in \mathbb{R}^n$, it can be shown that

$$\mathbf{a} = \mathbf{A} \mathbf{c} + \mathbf{b}, \quad (3)$$

where the components of \mathbf{A} and \mathbf{b} are given by

$$A_{ij} = d_t \mathbf{n}_i \cdot \mathbf{n}_j + d_r (\mathbf{n}_i \times \mathbf{p}_i) \cdot (\mathbf{n}_j \times \mathbf{p}_j),$$

$$b_i = d_i \mathbf{n}_i \cdot \mathbf{f}_a + d_r \mathbf{n}_i \cdot (\mathbf{m}_a \times \mathbf{p}_i).$$

Equation (3), together with the constraints (2), form a *linear complementarity problem* which can be solved for \mathbf{c} (as well as \mathbf{a}) using Baraff's algorithm [29]. Each \mathbf{f}_i then follows directly from $\mathbf{f}_i = c_i \mathbf{n}_i$.

Given \mathbf{f}_i , the net force \mathbf{f} and moment \mathbf{m} acting on the workpiece are given by

$$\mathbf{f} = \sum_i \mathbf{f}_i + \mathbf{f}_a, \quad \mathbf{m} = \sum_i (\mathbf{p}_i \times \mathbf{f}_i) + \mathbf{m}_a.$$

The workpiece's spatial velocity $\mathbf{v}_S \equiv (\mathbf{v}^T \boldsymbol{\omega}^T)^T$ then follows from Equation (1). The task simulator computes \mathbf{v}_S in this way once per time step (currently every 50 msec) and uses this to update the workpiece's position, as described next.

B. Computing workpiece displacement

The spatial velocity \mathbf{v}_S described above is used to update the workpiece's spatial position during each simulation time step. Let \mathbf{X}_0 be a 4×4 homogeneous transform the initial workpiece position, and let $\mathbf{v} = (v_x, v_y, v_z)$ and $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)$. Then if the time step Δt is small relative to the

velocity, and no object collision results, the new workpiece position \mathbf{X}_1 at the end of the time step can be approximated by²

$$\mathbf{X}_1 = \mathbf{X}_0 \mathbf{D} \Delta t, \quad (4)$$

where

$$\mathbf{D} = \begin{pmatrix} 1 & -\omega_z & \omega_y & v_x \\ \omega_z & 1 & -\omega_x & v_y \\ -\omega_y & \omega_x & 1 & v_z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

On the other hand, if this *does* result in a collision, then we need to reduce the displacement along the direction of \mathbf{v}_S (as described below) so as to prevent collision.

When the workpiece is very close to other objects, second-order constraints or numerical errors can prevent *any* finite displacement along the direction of \mathbf{v}_S from being collision-free, even when \mathbf{v}_S is itself valid. This has the effect of making the workpiece "stick", unreasonably, in certain configurations. In addition to this problem, I-COLLIDE becomes unreliable when objects are very close together, and fails when they interpenetrate. To overcome these problems, the simulator seeks to ensure that the workpiece is always a small distance ϵ_b away from other objects, where ϵ_b is less than the contact distance ϵ_c . This requirement is enforced using potential barrier functions, described next.

First, observe that over one time-step, we try to make the workpiece follow an approximately linear spatial path given by

$$\mathbf{X}(s) = \mathbf{X}_0 \mathbf{D} s \quad (5)$$

for $s \in [0, \Delta t]$. If a collision results at $s = \Delta t$, then we need to find some smaller value of s which does not result in a collision, and use this to compute the final position for the time step.

Now, let d_i be the Euclidean distance between the workpiece and another object i , and let $\delta \equiv d_i/\epsilon_b$. Then define the (differentiable) potential $U_i(d_i)$ (see Fig. 10-A, and also [30]) by

$$U_i(d_i) = \begin{cases} K[\delta - 1 - \ln(\delta)] & \text{if } 0 < \delta < 1, \\ 0 & \text{if } \delta \geq 1, \end{cases} \quad (6)$$

where K is a suitable constant. This will act to repel the workpiece from the object i . Because d_i , and hence $U_i(d_i)$, can be computed for each value of s along the path (5), each U_i can also be considered as a function of s .

Next, we define a simple linear potential $U_v(s)$ which produces (by itself) a motion along the path (5) with constant spatial velocity \mathbf{v}_S . Given the first-order dynamics (1), this takes the form

$$U_v(s) = - \left[\frac{\|\mathbf{v}\|^2}{d_t} + \frac{\|\boldsymbol{\omega}\|^2}{d_r} \right] s,$$

corresponding to the negative of the work done by \mathbf{f} and \mathbf{m} at each value of s .

Summing $U_v(s)$ and the $U_i(s)$ for all appropriate objects yields a net potential $U(s)$. During each time step, the workpiece

²Accumulating displacement in this way permits us to integrate the rotational velocity $\boldsymbol{\omega}$.

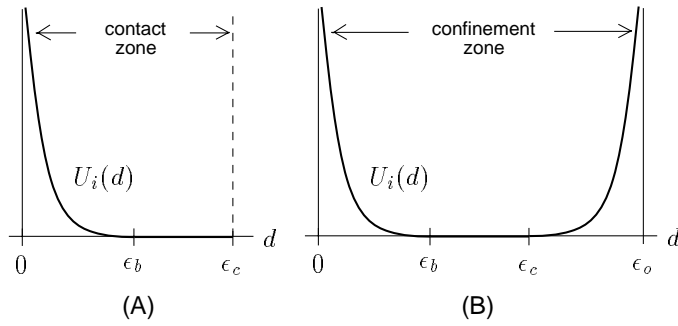


Fig. 10. (A) Potential function $U_i(d_i)$. (B) Modified $U_i(d_i)$ used to ensure that the workpiece remains within a distance ϵ_o of a capture object.

is moved to the value of s that minimizes $U(s)$ within the interval $s \in [0, \Delta t]$ (Fig. 11). If there are no obstacles nearby, all $U_i(s) \equiv 0$ and the minimum will occur at $s = \Delta t$, corresponding to the nominal displacement (4). Otherwise, the presence of obstacles may result in a motion to some intermediate value of s . Because $U_i(s)$ is not necessarily smooth (see below), the minimization is done using a golden section search, which is a form of bisection used specifically for minimization problems (see [31], Section 10.1).

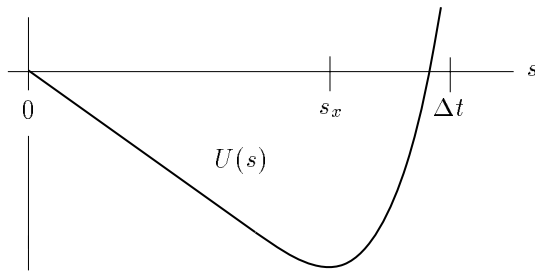


Fig. 11. An example potential function $U(s)$ with a minimum at $s = s_x$.

To help maintain each $d_i \geq \epsilon_b$, the translational part \mathbf{v} of \mathbf{v}_S is modified to include, for objects i with $d_i < \epsilon_b$, a repulsive component computed from the gradient of U_i with respect to the workpiece's translational position.

It may be asked why we do not treat the entire problem as a potential minimization and calculate both the \mathbf{v} and $\boldsymbol{\omega}$ of equation (1) from the gradient of U with respect to the workpiece's spatial position \mathbf{X} . The problem is that this gradient does not always exist: even though $U_i(d_i)$ is smooth, d_i itself is not smooth with respect to \mathbf{X} , and so $U_i(\mathbf{X})$ is not smooth either. While non-smooth optimization techniques exist that don't require an explicit gradient, the very thin size of the barrier means that convergence could be quite slow without a good estimate of initial direction. Indeed, the Baraff calculation (Section V-A) can be thought of as simply a good way of estimating this direction.

The barrier method described here can also be used to implement those motions for which the workpiece is constrained to *maintain* contact with capture objects. Once contact is achieved with a capture object, its potential $U_i(d_i)$ is modified so that in addition to approaching infinity at $d_i = 0$, it also approaches infinity at some outer boundary value ϵ_o , for $\epsilon_o > \epsilon_c$ (Fig. 10-B).

VI. PROGRAM GENERATION

When the operator has finished manipulating a workpiece, the program generator creates a set of robot motion commands to realize the specified operation.

It begins with the complete motion sequence history specified by the operator in the virtual environment. This is taken to be a piecewise-linear curve of spatial positions, called the *workpiece path*, formed by interpolating the workpiece positions computed during each time step of the simulation. Each workpiece position is called a *node* of the workpiece path, and may be associated with one or more contacts.

A brute-force way to accomplish the specified task would be to reproduce the workpiece path verbatim, closely replicating the operator's actions in a manner similar to conventional teleprogramming environments [1], [2]. However, in the context of our system, there would be problems with this:

1. The path may contain many unnecessary motions, such as those induced by the operator "feeling" her way around.
2. Because simulated motions are constrained to directions permitted by the dragger fixtures, the workpiece path may have superfluous kinks and bends.
3. The path may contain unwanted contacts, caused by the operator dragging the workpiece across or along obstacles, or using obstacles themselves as fixtures for part placement and alignment.

Nevertheless, the workpiece path *is* collision safe (within the resolution limits imposed by the simulator's time step). Therefore, we use the workpiece path as an initial feasible solution, and modify it so as to remove unwanted motions and unnecessary contacts.

A. Deforming the path

The idea is to treat the path as a deformable spatial curve which can be bent, stretched, or shrunk in order to move it away from objects or compress its length. Such a deformation can be accomplished using potential field methods. In particular, we apply to each of the path's nodes:

1. A tension force attracting it to its two nearest neighbors.
2. A repulsive force to push it away from unwanted obstacles.

Path endpoints are kept fixed, as are the endpoints of any required contact motions. The tension force (Item 1) is calculated with respect to both position and orientation. A constant, rather than variable, tension is used to keep the path from becoming overly stiff when stretched. The repulsive force (Item 2) is calculated with respect to translation only, due to difficulties in computing a repulsive gradient with respect to orientation. To keep nodes from sliding along the path, we eliminate any repulsive force component which is tangential to the path. Both the tension and repulsive forces are used to move each node along the path in succession, with the whole procedure being repeated until the path stabilizes. A schematic illustration of the process is shown in Fig. 12.

During the deformation process, it is important to ensure that each path node remains collision free *and* preserves any required contacts with capture objects. This is achieved by displacing each node using the contact simulation algorithm of Sections V-A and V-B, with the combined tension and repulsive forces assuming the role of the applied force \mathbf{f}_a and moment \mathbf{m}_a .

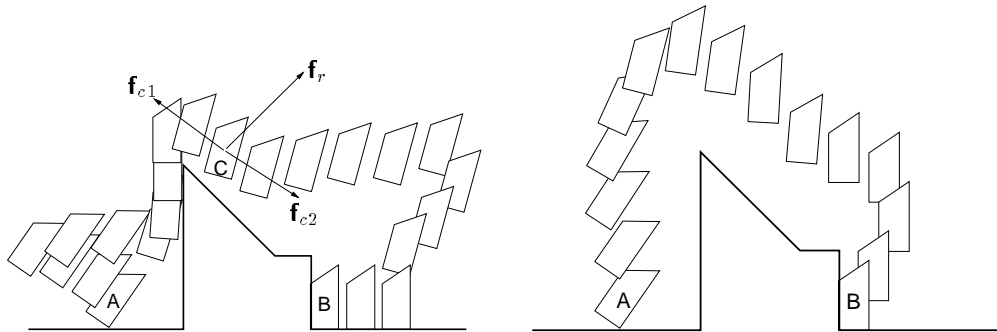


Fig. 12. Simple diagram showing the path deformation process. The original path (left) starts with the operator’s grabbing the workpiece at A, wandering off to the left, then using the obstacle in the middle to align the workpiece before dragging it over to the right (causing it to go partly out of alignment as it clears the top of the obstacle) and finally sliding it into the corner at B. Only the final contact state at B is desired, whereas other contacts and extraneous motions are not. Path deformation is achieved by applying to each path node (as illustrated by the one at C) forces to repel it from unwanted contacts (\mathbf{f}_r) and inter-node attractive forces to help straighten it out (\mathbf{f}_{c1} , \mathbf{f}_{c2}), with the final result shown on the right. The deformation process tends to be good at ensuring robust approaches to contact states; for instance, in the final path shown here, it is important that the corner destination B is approached from the right as well as from above, in order to avoid the possibility of getting snagged on the obstacle.

Our path deformation approach follows the work of Quinlan [32], who originated it to simplify and smooth collision-free paths produced by a motion planner. Our work differs in that we include contact states, the paths in question are formed from rigid bodies undergoing general spatial displacement (rather than points in configuration space), and contact simulation software is used to keep the path collision free while maintaining desired contacts. By contrast, collisions are prevented in [32] by surrounding path points with free space “bubbles”. Using the same technique here would be difficult because the proximity of obstacles in contact would require computing bubbles at extremely fine resolutions.

B. Robot motion commands

After the workpiece path has been simplified as described in Section VI-A, it is turned into a sequence of spatially-linear robot motions, with each target point possibly associated with one or more contacts. The initial command of the sequence involves a “guarded grasp” with which the manipulator grasps the workpiece.

At present, a contact is represented only in terms of its associated normal vector. For motions requiring contact, the robot’s speed is lowered, and its impedance is controlled to emulate a spring-damper system with low stiffness. We have found a simple, isotropic impedance to be sufficient, although this may change when the system is extended to handle operations involving tight fits. Contact is ensured by biasing the target position by a distance d in the negative direction of each contact normal. To do this, the target bias $\Delta\mathbf{p}$ must satisfy $\mathbf{n}_i \cdot \Delta\mathbf{p} = -d$ for each \mathbf{n}_i . Letting \mathbf{N} be the $n \times 3$ matrix whose rows are the n normals \mathbf{n}_i , and letting \mathbf{I} be the $n \times n$ identity matrix, this requires that $\Delta\mathbf{p}$ satisfies (at least approximately)

$$-d\mathbf{I} = \mathbf{N}\Delta\mathbf{p},$$

preferably for $\|\Delta\mathbf{p}\|$ not too large. This prohibits contact situations involving tight fits. The offset d should be a little larger than the accuracy of the work site model and is currently around 5 mm (vs. a typical object dimension of 50 mm). Contact execution is verified by checking that translational forces along the contact normal directions exceed a prescribed threshold.

Contact transition stability is ensured by limiting the output velocity of the impedance controller, effectively controlling system energy in a manner similar to that described in [33].

VII. EXPERIMENTS AND DISCUSSION

In Fig. 13, the task simulation and program generation part of the system is illustrated for two tasks: cornering a block, and dragging a block around the outside of a corner while maintaining contact. Actual execution of the later task is shown in Fig. 2.

A. Vision system performance

The vision algorithm is fairly robust to variations in light composition and intensity (although this is largely a function of the edge detection scheme). A moderate amount of occlusion is also tolerable, provided that at least one good edge sequence of an object is visible. For this reason, objects are sometimes not seen when they present a minimal number of faces to the camera “straight on”, a problem that should be greatly reduced by implementing the algorithm with multiple cameras. Indexing does occasionally fail due to occlusion and/or noise, if no appropriate feature groupings are detected for a particular object. During the Vancouver-Montreal demonstration, false object detection sometimes occurred due to the simplicity of the verification criterion (requiring 50% of visible feature lengths to be matched). More sophisticated criteria should indeed be used, although this is rarely a problem when the objects have greater complexity. At present, the problem has been alleviated by culling objects whose locations indicate they cannot realistically lie within the work site.

B. Operator interaction and task simulation

At present, the only contact feedback that the operator receives when manipulating the virtual environment is visual, in that the workpiece has its motion deflected by contact whereas the dragger box does not. We have observed that this in fact turns out to be a rather good feedback cue. Providing actual force feedback to the operator would of course also be desirable, but would require elaborate equipment. Instead, acoustic feedback (with

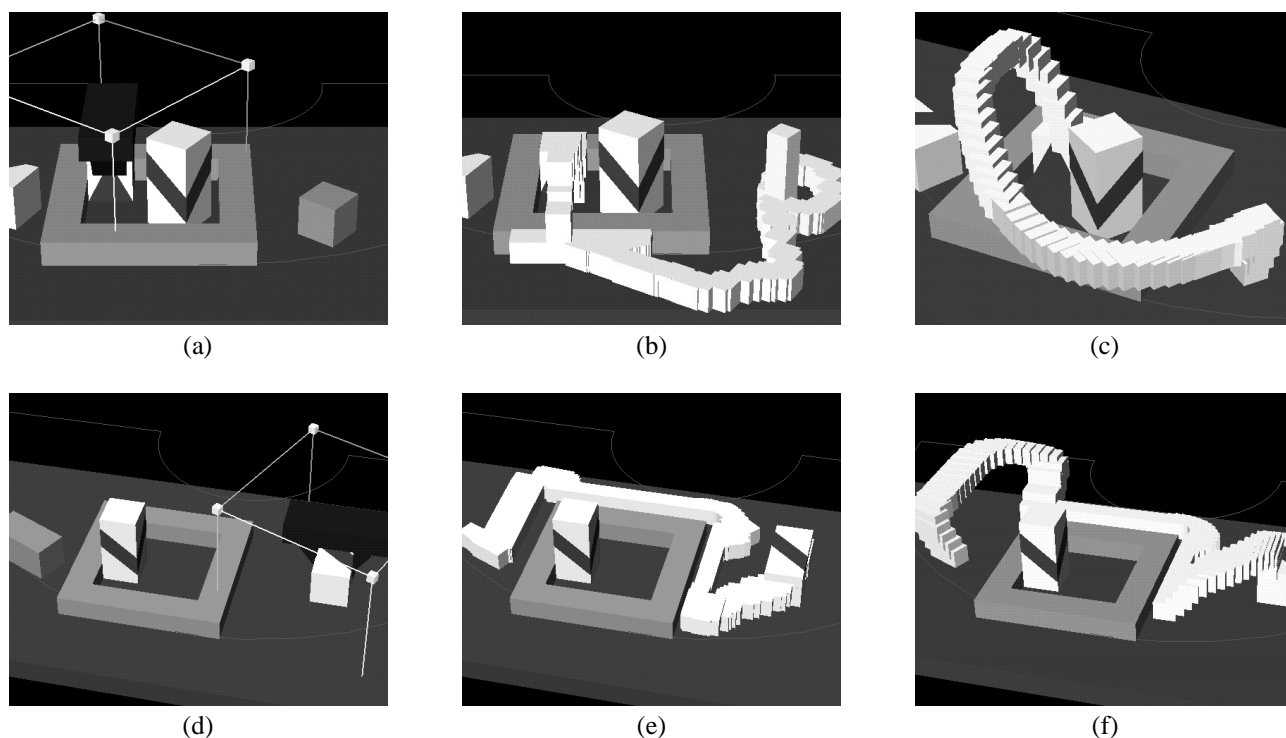


Fig. 13. **Top:** Cornering. The operator has placed the workpiece in a corner of the frame (a); its original location is shown by a grey “shadow block” at the right. The outside of the frame was used to help align the workpiece, as can be seen from the initial workpiece path (b), where the tightly spaced nodes are shown as half-sized white blocks. After modification (c), the nodes have been pushed away from the table and have kept their distance from the tall block, and the initial and final positions are approached at angles that prevent collision with the frame. **Bottom:** Here, the operator has moved the workpiece around the outside of the frame (d), while requesting that contact with the frame and table be maintained while going around the upper-right corner. The resulting initial workpiece path is shown in (e). After the path is modified (f), frame/table contact is maintained near the corner, while nodes are pushed away from the table elsewhere. The execution of this motion is shown in Fig. 2.

sounds generated on contact) might be a more practical enhancement, as well as being consistent with our objective of using simple desktop hardware.

The task simulator runs easily in real-time (20 Hz) on a 15 MFlop SGI Indy. Contact and barrier distances are $\epsilon_c = 0.5$ mm and $\epsilon_b = 0.1$ mm, respectively (vs. a typical object dimension of ≈ 50 mm). The maximization of $U(s)$ (Section V-B) is performed to an accuracy of about 1.0^{-5} , requiring about 25 I-COLLIDE calls per time step. We have observed that the use of barrier functions to keep objects a minimum distance apart greatly enhances the overall performance and smoothness of the contact simulation over the raw Baraff method [34]. However, when barrier functions are used to also enforce workpiece contact with a capture object (which is usually done with $\epsilon_o = 2$ mm), sticking will occasionally occur in some configurations, such as going around a corner.

At present, the use of I-COLLIDE, along with the assumption that contacts be modeled using a finite number of contact points, restricts the simulator to handling polyhedral representations. This is not terribly restrictive, because arbitrary curved objects can usually be represented as polyhedra at some suitable resolution.

Because the simulator runs in discrete time, in certain pathological situations the workpiece can “tunnel” through an object without detecting a collision (as also described in [34]). This problem diminishes as the sample rate is increased. With a maximum speed of 400 mm/s, a sample rate of 20 Hz, and 4 collision

tests per sample, tunneling is prevented for objects thicker than 5 mm.

C. Program generation

The path deformation algorithm (Section VI-A) tends to produce paths which have a “good” intuitive feel to them. It is particularly good at placing node points so as to ensure reliable and “snag free” approaches and departures from contact situations (Fig. 12), something which is by contrast rather difficult for a user to specify manually. The only caveat is that path nodes must be placed fairly close together for this to work. Algorithm convergence is not a problem, except that the constant tension force can cause minor instabilities to arise when nodes are very close together. This was corrected by using a tension proportional to distance for nodes closer than a certain minimum distance. However, the problem of maintaining proper node spacing is a tricky one and could use additional work. With regard to computation time, the examples of Fig. 13 took around five seconds to compute on a 12 MFlop SGI Indy, but this was without any effort having been made to optimize performance.

VIII. CONCLUSION

We have demonstrated an integrated system which links together vision, contact simulation, localized planning, and manipulator control into an easy-to-use interactive environment for programming contact tasks.

A fast, occlusion-tolerant grey-scale vision system is used to

obtain model information, providing an ongoing link between the simulation and reality. Object recognition proceeds rapidly (on the order of several seconds) and locations are computed to approximately pixel accuracy. The vision system is reliable enough to be used in a context where the operator simply selects recognized objects to be loaded into the work site model. As long as the viewed objects have a good subset of edges visible to the camera, the need to supplant the vision system by manually indicating features is rare.

We have also demonstrated the utility of using simulation as a robotic programming tool, specifically for part mating and contact tasks. An interesting result is that a full 3D contact simulation combined with first order dynamics allows the operator to use the environment itself as a "virtual fixture" to easily align and place parts. This, combined with entities such as dragger fixtures, makes it simple to accomplish a wide range of placement tasks using inputs from a simple 2D mouse.

The problem of turning a feasible but possibly complex set of simulated motion steps into a simpler set of robot commands is handled using local planning: artificial forces straighten the path out, move it away from unwanted contacts, and help ensure good approach and departure from desired contact states. The resulting path is turned into a set of spatially-linear robot motions, which includes the via points needed to maneuver the workpiece through free space. Interestingly, the specification of free-space via points is actually very tedious for an operator to perform manually. Therefore, the automation of this process using a local planner is particularly useful.

Important future work includes extending this system to handle part mating which involves "tight fits", as well as generating robot motion sequences which are provably robust to environmental errors. More general use can also be made of the vision system, particularly for task verification, and we also wish to explore the automatic synthesis and insertion of virtual fixtures during task simulation, based on the estimates of the operator's intentions.

REFERENCES

- [1] J. Funda, T. S. Lindsay, and R. P. Paul, "Teleprogramming: Toward delay-invariant remote manipulation," *Presence*, vol. 1, pp. 29–44, Winter 1992.
- [2] C. R. Sayers, *Operator Control of Telerobotic Systems for Real World Intervention*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania 19104, 1995.
- [3] G. Hirzinger, B. Brunner, J. Dietrich, and J. Heindl, "Sensor-based space robotics – ROTEX and its telerobotic features," *IEEE Transactions on Robotics and Automation*, vol. RA-9, pp. 649–663, Oct. 1993.
- [4] A. K. Bejczy, W. S. Kim, and S. C. Venema, "The phantom robot: Predictive displays for teleoperation with time delay," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Cincinnati, Ohio), pp. 546–551, May 1990.
- [5] C. R. Sayers and R. P. Paul, "An operator interface for teleprogramming employing synthetic fixtures," tech. rep., Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania 19104, June 1994.
- [6] U. S. National Research Council, *Virtual Reality: Scientific and Technological Challenges*. National Academy Press, Washington, D.C., 1995.
- [7] H. Bruyninckx, J. D. Schutter, P. V. de Poel, and W. Witvrouw, "A CAD-based contact force simulator as a learning tool for compliant motions," in *International Symposium on Intelligent Control*, pp. 287–292, 1992.
- [8] R. Koeppe and G. Hirzinger, *Learning Compliant Motions by Task-Demonstrations in Virtual Environments*, pp. 299–307. No. 223 in *Lecture Notes in Control and Information Sciences*, Springer, London, 1995.
- [9] Y. J. Cho, T. Kotoku, and K. Tanie, "Discrete-event-based planning and control of telerobotic part-mating process with communication delay and geometric uncertainty," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2, (Pittsburgh, Pennsylvania), pp. 1–6, Aug. 1995.
- [10] E. Oyama, N. Tsunemoto, S. Tachi, and Y. Inoue, "Experimental study on remote manipulation using virtual reality," *Presence*, vol. 2, pp. 112–124, Spring 1993.
- [11] W. S. Kim and L. W. Stark, "Cooperative control of visual displays for telemanipulation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Scottsdale, Arizona), pp. 1327–1332, May 1989.
- [12] E. Grimson and T. Lozano-Pérez, "Localizing overlapping parts by searching the interpretation tree," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 469–482, 1987.
- [13] D. G. Lowe, "Three-dimensional object recognition from single two-dimensional images," *Artificial Intelligence*, vol. 31, pp. 355–395, Mar. 1987.
- [14] P. Suetens, P. Fua, and A. Hanson, "Computational strategies for object recognition," *Computing Surveys*, vol. 24, pp. 5–61, 1992.
- [15] J. S. Beis and D. G. Lowe, "Learning indexing functions for 3-D model-based object recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (Seattle), pp. 275–280, June 1994.
- [16] H. Murase and S. K. Nayar, "Visual learning and recognition of 3-D objects from appearance," *International Journal of Computer Vision*, vol. 14, no. 1, pp. 5–24, 1995.
- [17] E. Paulos and J. Canny, "Delivering real reality to the world wide web via telerobotics," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Minneapolis), pp. 1694–1699, Apr. 1996.
- [18] K. Goldberg, M. Mascha, S. Gentner, N. Rothenberg, C. Sutter, and J. Wiegley, "Desktop teleoperation via the world wide web," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Nagoya), pp. 654–659, May 1995.
- [19] K. Taylor and J. Trevelyan, "Australia's telerobot on the web," in *26th International Symposium On Industrial Robots*, (Singapore), Oct. 1995.
- [20] J. E. Lloyd and V. Hayward, "Multi-RCCL user's guide," tech. rep., Centre for Intelligent Machines, McGill University, 3480 University Street, Montreal, Canada, H3A 2A7, Apr. 1992.
- [21] M. Pelletier and M. Doyon, "On the implementation and performance of impedance control on position controlled robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2, (San Diego, California), pp. 1228–1233, May 8-13 1994.
- [22] J. Wernecke, *The Inventor Mentor*. Addison-Wesley, Reading, Massachusetts, 1994.
- [23] J. S. Beis and D. G. Lowe, "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces," in *Conference on Computer Vision and Pattern Recognition*, (Puerto Rico), pp. 1000–1006, June 1997.
- [24] D. G. Lowe, "Fitting parameterized three-dimensional models to images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 441–450, May 1991.
- [25] R. Tsai, "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses," *IEEE Transactions on Robotics and Automation*, vol. RA-3, pp. 323–344, Aug. 1987.
- [26] J. Cohen, M. Lin, D. Manocha, and K. Ponamgi, "I-COLLIDE: An interactive and exact collision detection system for large-scaled environments," in *Proceedings of ACM Int. 3D Graphics Conference*, pp. 189–196, 1995.
- [27] S. Goyal, E. N. Pinson, and F. W. Sinden, *Simulation of Dynamics of Interacting Rigid Bodies Including Friction I: General Problems and Contact Model*, pp. 162–174. Springer-Verlag, London, 1994.
- [28] R. Featherstone, *Robot Dynamics Algorithms*. Kluwer Academic Publishers, 1987.
- [29] D. Baraff, "Fast contact force computation for nonpenetrating rigid bodies," in *SIGGRAPH 94 Conference Proceedings*, pp. 23–34, July 1994.
- [30] R. J. Spiteri, U. M. Ascher, and D. K. Pai, "Numerical solution of differential systems with algebraic inequalities arising in robot programming," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Nagoya), pp. 2373–2380, May 1995.
- [31] W. H. Press, S. A. Teukolsky, B. P. Flannery, and W. T. Vetterling, *Numerical recipes in C: the art of scientific computing*. Cambridge University Press, 1988.
- [32] S. Quinlan, *Real-time Modification of Collision-Free Paths*. PhD thesis, Department of Computer Science, Stanford University, Stanford, California 94305, Dec. 1994.
- [33] O. Khatib and J. Burdick, "Motion and force control of robot manipulators," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (San Francisco, California), pp. 1381–1386, 1986.
- [34] D. Baraff, "Interactive simulation of solid rigid bodies," *IEEE Computer Graphics and Applications*, vol. 15, pp. 63–75, May 1995.