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

John E. Lloyd, Jeffrey S. Beis, Dinesh K. Pai, David G. Lowe
Dept. of Computer Science, University of British Columbia
Vancouver, B.C., Canada
{lloyd,beis,pai,lowe}@@cs.ubc.ca

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 is 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.

1. 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:

- ◊ It eliminates the need for tedious textual descriptions (such as "place face A on top of face B").
- ◊ A simulated environment can be augmented with virtual fixtures and aids to assist programming. Alternatively, the environment can be simplified, with minimal display of the manipulator system, permitting direct manipulation of objects and programming which is more "task-centric".
- ◊ For programming purposes, 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 continuously updated by sensing.

In contrast with most virtual environments, the locations and types of objects in our environment are acquired automatically using vision. This makes the virtual environment 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 encompasses 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 Figure 1, which illustrates the placement of a block into a corner (readers with Internet access can also download an AVI video of this via `http://www.cs.ubc.ca/spider/pai/telerobotics.html`). A more complex type of task which can be easily programmed involves dragging a block around the outside of a corner while contact is maintained (Figure 2).

1.1. 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 a set of 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.

[1] Submitted to the *IEEE Transactions on Robitics and Automation*, September 1997

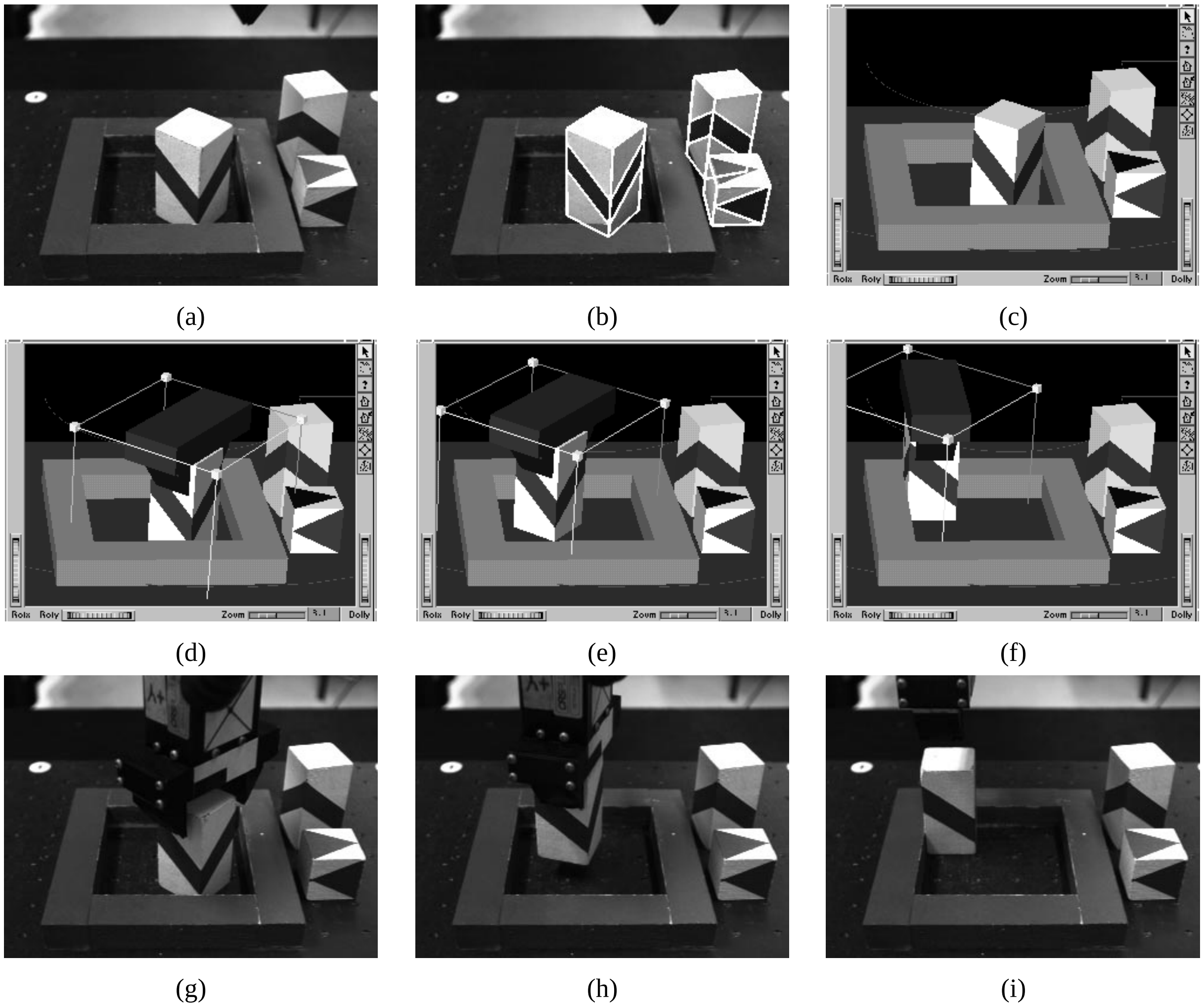

Figure 1. Simple example showing the placement of a block in a corner. Blocks within the initial work site (a) are located by the vision system (b, with matching edges outlined in white), and then used to update the work site model, graphically rendered in (c). An operator can then select an object by "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 realize the necessary contacts.

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 simulation module 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 spatial motions in various directions. The dynamics within the simulated environment is first order, since this is (a) easy to compute, (b) intuitive for the operator, (c) qualitatively similar to a 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 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 used to perform the simulated task, and deforming 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 deformation technique is that it can be constrained to *preserve* desired contacts, as in the cornering example of Figure 2. After deformation, the sequence is simplified into a piecewise-linear spatial path, where each node may be associated with one or more contacts. Each path segment is realized using a single robot motion command, for which required contacts are realized using impedance control combined with target position biasing.

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.

1.2. Connection to Telerobotics

Although our focus is currently on model acquisition and programming, our system can be used to perform telerobotic manipulation, and was in fact inspired by previous work in that area (Section 2).

All the links between the operator site and the work site (Section 3) 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.

1.3. Outline

The rest of this paper is arranged as follows. Related work is discussed in Section 2, and a more detailed description of the system components and hardware is given in Section 3. The vision system, task simulator, and program generator are then presented in Sections 4, 5, and 6, respectively. Experimental observations are given in Section 7.

2. Related Work

The work described here is closely connected to the *teleprogramming* work of Funda, Sayers, and others [11, 27], where operator interaction with a simulated environment is used to overcome time delay problems which can arise in some telerobotic situations. Teleprogramming was predated by the use of predictive graphical simulation [15, 6]. The introduction of synthetic "fixtures" into the operator's display to assist in task specification has also been explored [28, 32], and virtual reality simulation has been investigated as a platform on which fine motion task skills can be learned [7, 17].

Commands sent to a remote site in teleprogramming systems tend to be at the level of "guarded moves". The ability to send higher level commands asking the remote manipulator to achieve a particular contact state (recovering if necessary from any intervening contact states which can be anticipated) is investigated in [9], using a Petri-Net-based contact state model.

With respect to the problem of model acquisition, virtually all practical model-based telerobotic systems currently handle this using extensive operator interaction. The traditional way to do this is to have the operator manually indicate known object features in a video image of the remote site, and use the 2D image coordinates of these features to solve for the 3D positions of the associated objects [22, 16].

Automatic model generation using computer vision has been limited by the ability of systems to accurately identify and locate 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 [14, 19], 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. This model-based approach contrasts with appearance-based recognition [21], 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, [23, 12, 30].

3. System Description

A block diagram of the UBC system is shown in Figure 3. The system is divided into a *work site*, consisting of a robot, its con-

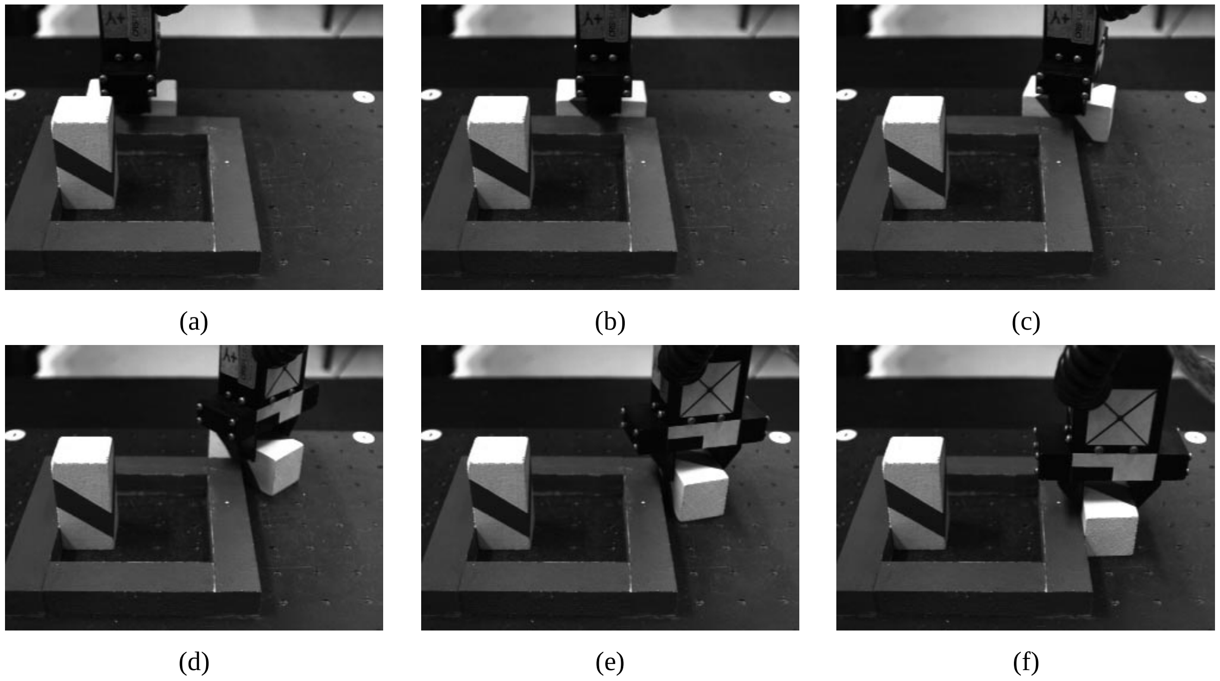

Figure 2. Sequence showing a more complex contact task where a block is dragged around a corner of the puzzle frame while contact with the frame is maintained.

troller, and the vision module, and an *operator site*, comprising the operator interface, task simulator, model editor, and program generator.

3.1. Work site

The work site (Figure 4) contains a 6 DOF CRS A460 robot (Puma-type geometry), controlled at the lowest level by 1 KHz joint servos (supplied by the manufacturer), which are in turn driven by a *task controller*. The task controller is implemented using the Robot Control C Library (RCCL) [18] running in real-time on a Sun Sparc 5. It accepts Cartesian motion commands from the operator site, and generates the required trajectories at 100 Hz. The trajectory generator also receives input from a force sensor, allowing it to implement both guarded moves and a position-based impedance control similar to that described in [24].

The vision module continuously collects images from a single monochrome camera and processes them using a model-based vision algorithm (Section 4) to locate objects in the scene. The objects and their positions are continuously sent back to the model editor at the operator site. The camera image itself is also transmitted back to the operator site, where it is displayed in a separate window.

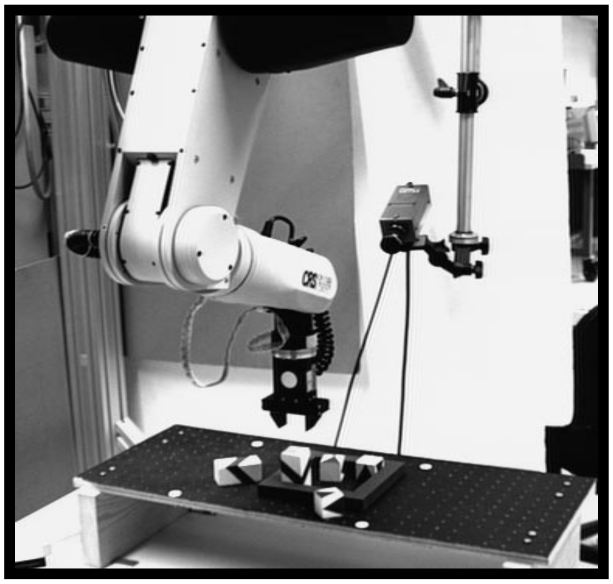

Figure 4. Remote site, showing the robot, camera, and work area.

3.2. 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 5). Model data includes polyhedral representations of the work space objects, plus kinematic and geometric information about the robot manipulator (dynamical information is not necessary for the low-speed contact operations presently being investigated). Other information about work site objects, such as friction and stiffness models, may be added later if required. Model information is updated, based on recognition data received from the video/vision module, by the *model editor* (Section 4.3). Images of recognized objects are displayed in a *feedback window*, and also overlaid on the raw video image displayed in the *camera image window*. The operator then selects

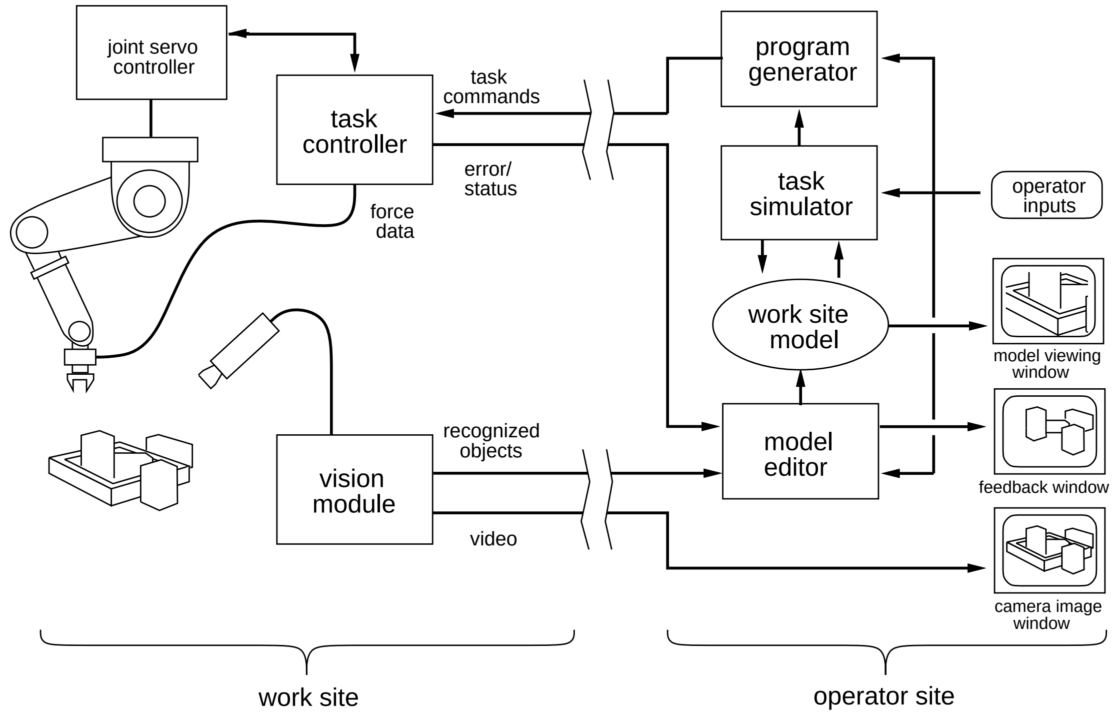

Figure 3. System architecture.

if and when specific recognized objects are introduced into the work site model. The model editor, together with the vision system, constitute the *model generator* described in Section 1.1. A graphical display of the work site model itself is provided in the *model viewing window*, implemented using the SGI 3D modeling package Open Inventor [33]. The display's viewpoint can be adjusted to suit the operator's needs. Figure 5 shows the typical arrangement of the different viewing windows on the SGI system. Lastly, a *program generator* (Section 6) creates the robot motion commands required to realize specific tasks and sends them to the task controller at the work site.

4. 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. The state of the art in 3D vision has only recently developed to the point where these capabilities can be achieved with reasonable speed and reliability. This project employs the model-based recognition system of Beis and Lowe [4, 5], which uses a novel form of rapid indexing to recognize 3-D objects from any point of view in single 2-D images.

The vision system is currently restricted to using straight edges of the objects in the recognition process. A model must be provided for each object type which specifies surface visibility and the 3-D location of all prominent lines and edges. For the demonstrations described in this paper, these models were generated by hand. However, a separate tool has also been developed that allows models to be automatically generated from a number of images of an object, with human input limited to pointing out corresponding edges in the different images [3].

4.1. 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 Figure 6) are used to generate a vector of measurements giving the relative lengths or angles between the lines. This "index" vector is invariant to 2-D translation, rotation and scaling, but will vary with the projection of different 3-D rotations of the object. Other methods, which require fully invariant feature groupings for indexing, will be more limited in which objects their system can handle. A precomputed index covering a sample of all 3-D 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 [5]. The greater specificity encoded in larger vectors is important for improving indexing accuracy [8] and hence recognition speed, and to enhance discrimination within larger model databases. Full details of this indexing approach are given in the paper by Beis and Lowe [4].

Once a tentative interpretation has been made for some image features, it is possible to estimate the object location and orientation in 3-D [20]. 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 3-D 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 lit-

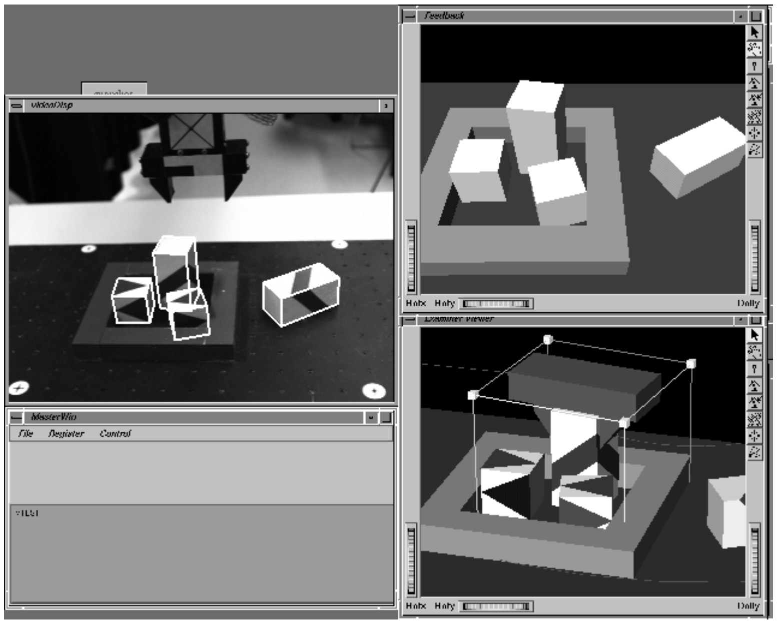

Figure 5. Typical screen layout on the operator site workstation. Clockwise from top-left: camera image window, feedback window, model viewing window (with tall block being manipulated by a dragger box), and textual interface window.

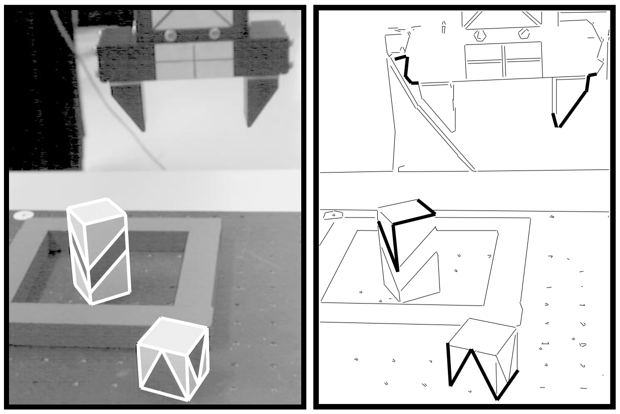

Figure 6. Left frame shows cropped image of work space overlaid with wire-frame models at positions determined by the recognition algorithm. Right frame shows edge-detected image with examples of correct and incorrect feature groupings used in the indexing process.

tle 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 edges, then the match is rejected and a new indexing hypothesis is used. Therefore, the final recognition has good robustness and accuracy, even though the initial indexing is probabilistic.

4.2. Speed, accuracy, and calibration

The full recognition process currently requires about 5 seconds running on a 15 Mflop SGI Indy. This will be much improved in the future once a number of optimizations have been made for speed. Much of the time is currently spent on the low-level edge detection process, which could be greatly accelerated by using some image processing hardware.

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 3-D 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. If this is a problem, then it would be possible to use a similar approach to recognition with 2 or more cameras to achieve full accuracy in all dimensions. For this project, accuracy was improved using other constraints, such as the fact that objects close to the worktable surface must in fact be resting on the surface.

A standard pin-hole camera model is assumed. Intrinsic parameters (focal-length and radial distortion) were calibrated off-line using the algorithm in [31]. The camera position relative to the work space was calibrated manually by having the operator identify, in the camera image, work site features of known position.

4.3. Model editing

Information on recognized objects and their locations is continuously sent from the vision system to the model editor in the operator station. A primary function of this module is to ensure that object information is logically consistent, compensating for the fact that the vision system recognizes objects independently of each other. For instance, objects located near the table are assumed to be resting *on* the table (since they can neither penetrate the table top nor hover above it), and their location is adjusted accordingly. Location adjustment occurs along the camera axis, since for our presently monocular system this is the primary direction in which errors occur. Other consistency adjustments, such as ensuring that objects do not interpenetrate with each other, should be implemented here although

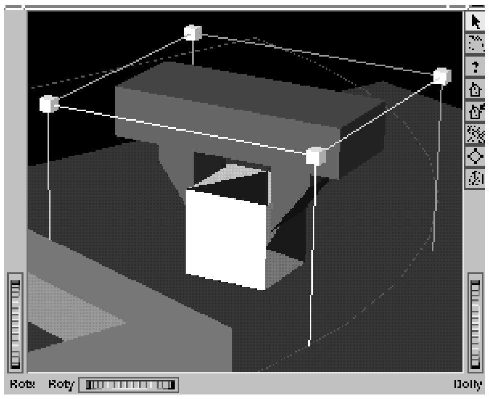

Figure 7. Dragger box and gripper rendering surrounding a block to be manipulated. Dragging the mouse cursor along one of the planes of the box causes a displacement parallel to the plane, which is converted into a "virtual force" acting on the workpiece's center.

we have not yet done so. After the recognized objects' locations have been adjusted, they are displayed in the feedback window and a wireframe overlay is also generated in the camera image window (Figure 3).

Updating of the world model itself is done under the control of the operator, and is generally done between task specifications and then 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.

5. Task Simulation

The purpose of the task simulator is to enable the operator to manipulate an object (or *workpiece*) within the virtual environment so as to easily specify a task. One commonly occurring task is to simply move the workpiece to a destination position requiring 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 Figure 1 typifies this. A more complex task might involve maintaining contact during motion, such as shown in Figure 2.

To facilitate these types of actions, the task simulator allows a manipulated object, or *workpiece*, to be moved about the virtual environment using first order dynamics combined with a contact model. 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 some desired contact state with respect to the rest of the environment. First order dynamics is used for the reasons mentioned in Section 1.1. To specify motions involving contact, the operator can also request that the workpiece *maintains* contact with certain selected *capture* objects. Once the workpiece makes contact with a capture object, its motion is constrained so as to maintain that contact (using barrier functions, as described at the end of Section 5.2). After the operator completes a task, she signals this to the system, which then invokes the program generator (Section 6) to create a sequence of robotic commands capable of realizing it at the work site.

The simulated environment is visible to the operator, from any angle, 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 Figure 7 and Figure 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.

Displacements between the dragger box and the workpiece are used to create a virtual force $\mathbf{f}_a$ acting on the workpiece (Figure 8). When the workpiece is brought into contact with other objects, normal forces also arise in reaction to the applied force (Figure 9). The normal forces plus the applied force create a net total force on the workpiece, from which the workpiece velocity is computed in accordance with the first-order dynamic model (Section 5.1). While devices such as dragger boxes are commonly used to generate spatial motions in graphical systems (our dragger box itself is an Inventor object), their use to generate virtual forces for integration into a dynamic environment is more novel.

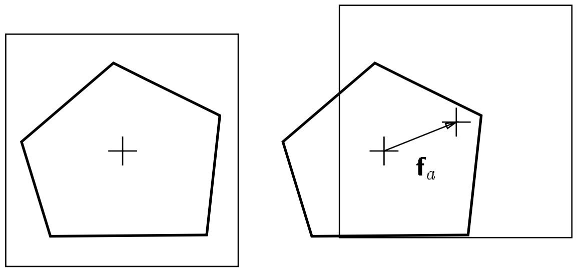

Figure 8. Displacements of the dragger box relative to the object create a virtual force f_a on the object.

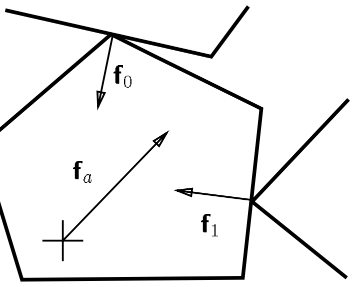

Figure 9. Contact of the workpiece with other objects gives rise to normal reaction forces ($\mathbf{f}_0$ and $\mathbf{f}_1$).

5.1. Contact dynamics

The simulator keeps track of the distances between objects using I-COLLIDE [10]. Objects closer than ϵ_c are assumed to be in contact, in which case information provided by I-COLLIDE is used to determine a suitable finite set of contact points $\mathbf{p}_i$ and normals $\mathbf{n}_i$ modeling all the contacts[2]. Reaction forces $\mathbf{f}_i$ acting along the contact normals, in response to the applied force $\mathbf{f}_a$ (Figure 9), are determined using Baraff's algorithm [1]. The net force $\mathbf{f}$ and moment $\mathbf{m}$ acting on the workpiece are then given by

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

First order dynamics is then used to determine the workpiece's spatial velocity $(\mathbf{v}^T \boldsymbol{\omega}^T)^T$, according to

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

where d_t and d_r are suitable constants.

The task simulator computes and applies the workpiece's velocity in this way once per time step (currently every 50 msec) and uses this information to update the workpiece's position, as described in the next section.

5.2. Enforcing Contact Constraints with Barrier Potentials

The spatial velocity $\mathbf{v}_S \equiv (\mathbf{v}^T \boldsymbol{\omega}^T)^T$ described in the previous section is used to determine the change in workpiece position during each simulation step. Nominally, if no collision results, this is given by $\mathbf{v}_S \Delta t$, where Δt is the size of the simulator time step. On the other hand, if this does result in a collision, then the displacement is scaled back along the direction of $\mathbf{v}_S$ (as described below) to a point where there is no collision.

When the workpiece is extremely close to other objects, second order constraints or numerical errors can render any finite collision-free motion impossible, even when a valid $\mathbf{v}_S$ exists. This has the effect of making the workpiece appear to "stick", unreasonably, at certain configurations. In addition to this problem, the closest-feature and distance information returned by I-COLLIDE becomes unreliable when objects are very close together, and fails when they interpenetrate. Good simulator performance thus requires trying to keep the workpiece a minimum distance ϵ_b away from other objects, where ϵ_b is less than the distance ϵ_c below which objects are assumed to be in contact. This requirement is enforced using a potential barrier, which we now describe.

Let d_i be the distance between the workpiece and another object i, and let $\delta \equiv d_i/\epsilon_b$. Then define the potential $U_i(d_i)$ (see, e.g., [29]) by

$$U_i(d_i) = \begin{cases} K[\delta - 1 - \ln(\delta)] & \text{if } 0 < \delta < 1, \\ 0 & \text{if } \delta \geq 1. \end{cases} \tag{2}$$

where K is a suitable constant. This will act to repel the workpiece from the object i. In order to also induce motion of the workpiece along the direction of $\mathbf{v}_S$, we define an attractive potential U_v that decreases uniformly along $\mathbf{v}_S$ in proportion to the work done by $\mathbf{f}$ and $\mathbf{m}$. If motion in the direction of $\mathbf{v}_S$ is parameterized by s, such that $s \in [0,1]$ corresponds to one simulator time step Δt, then

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

[2] Face-face or edge-face contacts can be reasonably simulated using a finite set of point contacts; see [13].

Summing U_v and the U_i for all appropriate objects yields a net potential $U(s)$ that varies along the direction of $\mathbf{v}_S$. During each time step, the workpiece is moved so as to minimize $U(s)$ (Figure 10). If there are no obstacles nearby, all $U_i = 0$ and this minimum will occur at $s = 1$, corresponding to the nominal displacement $\mathbf{v}_S \Delta t$. For purposes of performing the minimization, $U_i(d_i)$ is taken to be $+\infty$ for $d_i < 0$. Because $U_i(s)$ is not smooth (see below), the minimization is done using a golden section search.

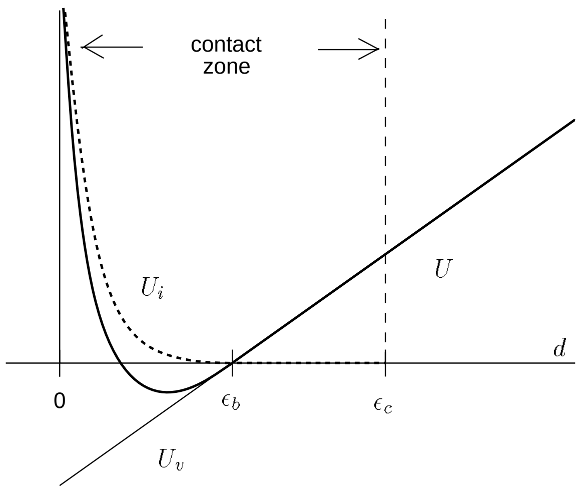

Figure 10. Net potential U (thick solid line) formed by summing U_i (dotted line) and U_v (thin diagonal line), plotted here as functions of the distance d from obstacle i. During each simulation step, U is minimized over the segment of d that corresponds to $s \in [0,1]$.

To help maintain each $d_i \geq \epsilon_b$, the translational velocity $\mathbf{v}$ in equation (1) is modified to include, for any object i for which $d_i < \epsilon_b$, a repulsive component computed from the gradient of $U_i(d_i)$ with respect to the workpiece's translational position.

The reader may wonder 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 overall spatial position. The problem is that this gradient is neither simple to calculate, nor smooth. Even though $U_i(d_i)$ is smooth, d_i itself is not smooth in the configuration space of a polyhedral object, and so U_i is not smooth with respect to the configuration space either. Hence in many cases a formal gradient doesn't exist. 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 5.1) can be thought of as simply a good way of estimating this direction.

The potential method described here can also be used to implement those motions for which the workpiece is constrained to *maintain* a particular contact. This is done by simply modifying $U_i(d_i)$ so that in addition to approaching infinity at $d_i = 0$, it also approaches infinity as d_i exceeds some outer boundary value ϵ_o, for $\epsilon_o > \epsilon_c$.

6. Program Generation

When the operator has finished manipulating a workpiece, this is indicated to the system using a keystroke. The program generator then sets about creating a set of robot motion commands to realize the specified operation.

The program generator is given the sequence of every motion made by the workpiece during every time step in the virtual environment; this is called the *workpiece path*. Since the motion made by the simulator during each time step is linear, the workpiece path is actually a piecewise-linear curve of spatial positions, with each node possibly associated with one or more contacts.

A very brute-force way to accomplish the specified task would be to reproduce the workpiece path verbatim. This would closely replicate the operator's actions in a manner similar to what is done in teleprogramming environments [11, 27]. However, in the context of our system, there are problems with this:

1. The workpiece path may contain many unwanted or 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 specifically using obstacles as virtual fixtures for part placement and alignment.

Nevertheless, the workpiece path *does* have one very desirable property, in that it is collision free (within the resolution limits imposed by the simulator's time step). Therefore, what we do is use the workpiece path as an initial feasible solution, and modify it so as to remove unwanted motions and unnecessary contacts.

6.1. 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 constant tension force attracting it to each of its two nearest neighbors.
2. a spring-like repulsive force that pushes 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 (as mentioned in Section 5.2). To keep nodes from processing along the path, we eliminate any repulsive force component which is tangential to the path. These "forces" are used to move each node along the path in succession, with the whole procedure being repeated until the path stablizes. An schematic illustration of the process is shown in Figure 11.

When a node is moved, it is important to ensure that it remains collision free *and* preserves any required contacts with capture objects. This is achieved by moving each node using the contact simulation software of Sections 5.1 and 5.2, with the combined tension and repulsive forces assuming the role of the applied force $\mathbf{f}_a$.

Our path deformation approach closely follows the work of Quinlan [25], who originated it to simplify and smooth collision-free paths produced by a motion planner. Our work differs from Quinlan's in several respects: we include contact states, the paths in question are formed from rigid bodies in $SE(3)$ rather than points in configuration space, and contact simulation software is used to effect collision free node displacement (also allowing us to maintain desired contacts during the deformation). By contrast, collisions are prevented in [25] by surrounding path points with free space "bubbles". This would be difficult to do here because of we are dealing with rigid body motions and because the proximity of obstacles in contact would require computing bubbles at extremely fine resolutions.

6.2. Robot Motion Commands

After the workpiece path has been deformed as described in Section 6.1, it is simplified using a scheme similar to that employed for polygonal path approximation [26]. The result is a piecewise-linear spatial path with possible contact states associated with each of the knot points. This can be readily transformed 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. Force sensor data is used to help execute and verify this command.

At present, a contact is represented only in terms of its associated normal vector. For motion targets which involve 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 impedance that is uniform along all axes to be sufficient, although this may change when we extend the system to handle operations involving tight fits. Contact is ensured by adding to the target positions a translational bias $\Delta\mathbf{p}$ to produce a small offset d in the opposite direction of the contact normals. Letting the set of normal vectors form the columns of a matrix $\mathbf{N}$, our ability to do this requires that we can solve

$$d\mathbf{l} \;=\; -\mathbf{N}^T\,\Delta\mathbf{p},$$

preferably for $||\Delta\mathbf{p}||$ not too large. This presently prohibits contact situations involving tight fits. The size of d is determined by the accuracy of the work site model and is currently around 5 mm (in an environment where objects have a typical dimension of 30 to 60 mm). Motions involving contact are verified by making sure that translational forces along the contact normal directions exceed a prescribed threshold.

The problem of contact transition instability is dealt with by clipping the output velocity of the impedance controller to a magnitude not exceeding the current robot speed (on the principle that this should be large enough to remove observed forces

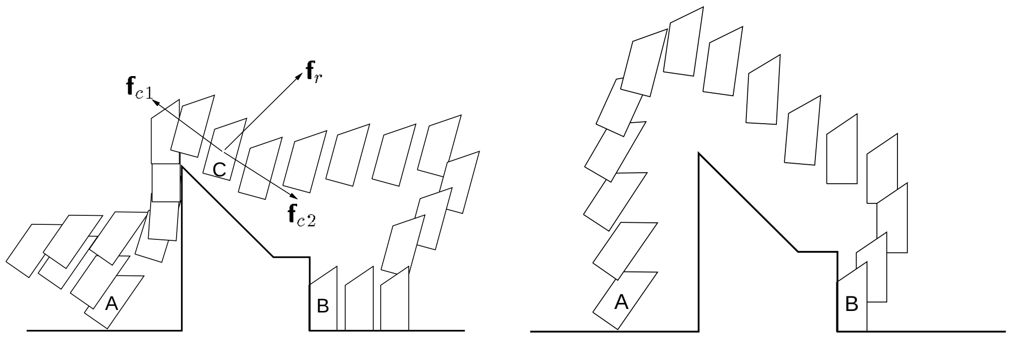

Figure 11. Simple diagram showing the path deformation process. The original path (left) starts with the operator's grabbing the workpiece at A, wandering off to the left, then using the middle obstacle to align the workpiece before dragging it over to the right (causing it to go partly out of alignment 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 lip immediately above the corner.

within one control cycle).

7. Demonstrations and Observations

In Figure 12, the behavior of 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 the Figure 2.

7.1. Vision system performance

The vision algorithm is fairly robust to variations in light composition and intensity (although this is mainly a function of the edge detection scheme used rather that higher-level algorithm). 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 presenting a minimal number of faces to the camera "straight on", a problem that should be greatly reduced by implementing the algorithm in stereo. Indexing does occasionally fail due to occlusion and/or noise, if no appropriate feature groupings are detected for a particular object. In offline experiments, better robustness in this area has been be achieved by indexing with several types of grouping in parallel. Modules using smaller, more local groupings can then succeed in the difficult cases of major occlusions, at a penalty of slightly increased processing time, with the larger groupings providing better efficiency in the typical cases. 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.

7.2. Task simulation

The task simulator runs easily in real-time (20 Hz) on a 15 MFlop SGI Indy. Objects closer than $\epsilon_c = 0.5$ mm are assumed to be in contact, and for the barrier function we have $\epsilon_b = .1$ mm (compared to a work area diameter ≈ 350 mm and a typical object dimension of ≈ 45 mm). The golden section search (Section 5.2) is performed to an accuracy of about 1.0^{-5}, requiring about 25 I-COLLIDE calls per time step, or 500 per second. I-COLLIDE's collision and distance computations can sometimes fail in non-generic situations (*e.g.*, when two object faces are very close to parallel), although such problems are common in fast geometric modeling systems whose computations are done using floating point arithmetic. We 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 [2]. 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.

The discrete-time nature of the simulation means that in certain pathological situations the simulator can "tunnel" through an object without detecting a collision (similar observations were made in [2]). With a maximum speed of 400 mm/sec, a sample rate of 20 Hz, and 4 tests per sample, we currently need to be wary of objects thinner that 5 mm.

7.3. Program generation

The path deformation algorithm (Section 6.1) 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 (Figure 11), 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 (less than the thickness of any lip) 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

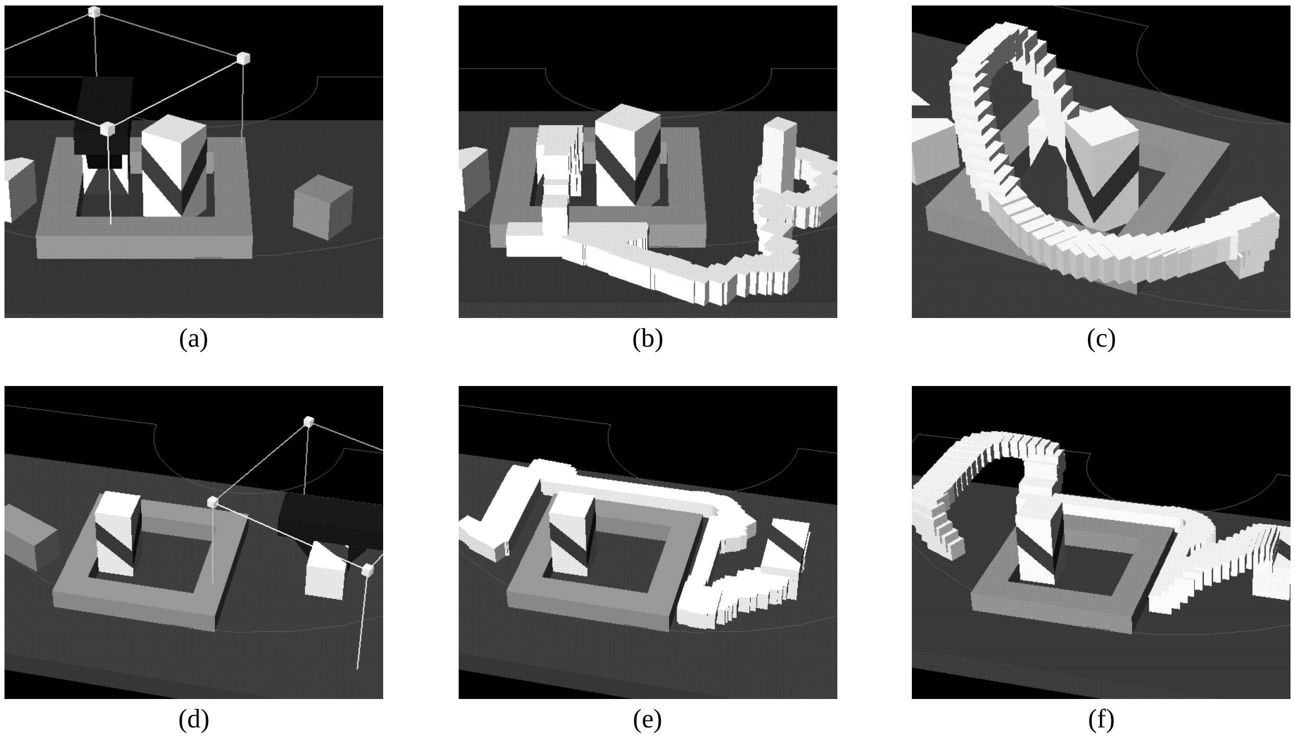

Figure 12. **Top:** Cornering. The operator has placed the workpiece in a corner of the frame (a); its original location is shown by a grey "shadow block" at the right. The outside of the frame was used to help align the workpiece, as can be seen from the initial workpiece path (b), where the tightly 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 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 Figure 2.

use additional work. With regard to computation time, the deformations associated with the examples of Figure 12 took around five seconds to compute on a 12 Mflop SGI Indy, but this was without any effort having been made to optimize performance.

8. 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, when possible, move it away from unwanted contacts, and help ensure good approach and departure from desired contact states. The resulting path is simplified into one containing a moderate number of via points (possibly including contact), connected by straight-line spatial robot motions. Interestingly, the specification of spatial via points, particularly those in free space, is a very tedious task for an operator to perform manually. Therefore, the automation of this process using a local planner is particularly important.

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] D. Baraff. Fast contact force computation for nonpentrating rigid bodies. In *SIGGRAPH 94 Conference Proceedings*, pages 23–34, July 1994.

[2] D. Baraff. Interactive simulation of solid rigid bodies. *IEEE Computer Graphics and Applications*, 15(3):63–75, May 1995.

[3] Jeffrey S. Beis. Building models with planar faces using a structure-from-motion algorithm plus a small amount of post-processing. Technical Report TR-96-14, Computer Science Department, University of British Columbia, 201-2366 Main Mall, Vancouver, Canada V6T 1Z4, June 1996.

[4] Jeffrey S. Beis and David G. Lowe. Learning indexing functions for 3-d model-based object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 275–280, Seattle, June 1994.

[5] Jeffrey S. Beis and David G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Conference on Computer Vision and Pattern Recognition*, pages 1000–1006, Puerto Rico, June 1997.

[6] 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*, pages 546–551, Cincinnati, Ohio, May 1990.

[7] H. Bruyninckx, J. De Schutter, P. Van de Poel, and W. Witvrouw. A cad-based contact force simulator as a learning tool for compliant motions. In *International Symposium on Intelligent Control*, pages 287–292, 1992.

[8] A. Califano and R. Mohan. Multidimensional indexing for recognizing visual shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(4):373–392, 1994.

[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)*, volume 2, pages 1–6, Pittsburgh, Pennsylvania, August 1995.

[10] 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*, pages 189–196, 1995.

[11] J. Funda, T. S. Lindsay, and R. P. Paul. Teleprogramming: Toward delay-invariant remote manipulation. *Presence*, 1(1):29–44, Winter 1992.

[12] 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*, pages 654–659, Nagoya, May 1995.

[13] S. Goyal, E. N. Pinson, and F. W. Sinden. *Simulation of Dynamics of Interacting Rigid Bodies Including Friction I: General Problems and Contact Model*, pages 162–174. Springer-Verlag, London, 1994.

[14] E. Grimson and T. Lozano-Pérez. Localizing overlapping parts by searching the interpretation tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 469–482, 1987.

[15] G. Hirzinger, B. Brunner, J. Dietrich, and J. Heindl. Sensor-based space robotics – rotex and its telerobotic features. *IEEE Transactions on Robotics and Automation*, RA-9(5):649–663, October 1993.

[16] 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*, pages 1327–1332, Scottsdale, Arizona, May 1989.

[17] R. Koeppe and G. Hirzinger. *Learning Compliant Motions by Task-Demonstrations in Virtual Environments*, pages 299–307. Number 223. Springer, London, 1995.

[18] John E. Lloyd and Vincent Hayward. Multi-rccl user's guide. Technical report, Centre for Intelligent Machines, McGill University, 3480 University Street, Montreal, Canada, H3A 2A7, April 1992.

[19] David .G. Lowe. Three-dimensional object recognition from single two-dimensional images. *Artificial Intelligence*, 31(3):355–395, March 1987.

[20] David G. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(5):441–450, May 1991.

[21] H. Murase and S. K. Nayar. Visual learning and recognition of 3-d objects from appearance. *International Journal of Computer Vision*, 14(1):5–24, 1995.

[22] E. Oyama, N. Tsunemoto, S. Tachi, and Y. Inoue. Experimental study on remote manipulation using virtual reality. *Presence*, 2(2):112–124, Spring 1993.

[23] 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*, pages 1694–1699, Minneapolis, April 1996.

[24] Michel Pelletier and Michel Doyon. On the implementation and performance of impedance control on position controlled robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1228–1233, San Diego, California, May8-13 1994.

[25] Sean Quinlan. *Real-time Modification of Collision-Free Paths*. PhD thesis, Department of Computer Science, Stanford University, Stanford, California 94305, December 1994.

[26] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1:244–256, 1972.

[27] 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.

[28] C. R. Sayers and R. P. Paul. An operaan operator interface for teleprogramming employing synthetic fixtures. Technical report, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania 19104, June 1994.

[29] 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*, pages 2373–2380, Nagoya, May 1995.

[30] Ken Taylor and James Trevelyan. Australia's telerobot on the web. In *26th International Symposium On Industrial Robots*, Singapore, October 1995.

[31] 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*, RA-3(4):323–344, August 1987.

[32] National Research Council (U.S.A.). *Virtual Reality. Scientific and Technological Challenges*. National Academy Press, Washington, D.C., 1995.

[33] J. Wernecke. *The Inventor Mentor*. Addison-Wesley, Reading, Massachusetts, 1994.