Extracting Robotic Part-mating Programs from Operator Interaction with a Simulated Environment

John E. Lloyd and Dinesh K. Pai
Department of Computer Science, University of British Columbia
Vancouver, B.C., CANADA
{lloyd,pai}@cs.ubc.ca

Abstract

We describe an integrated system for programming part-mating and contact tasks using simulation. A principal goal of this work is to make robotic programming easy and intuitive for untrained users. Using simulation, an operator can specify part placement and contact motions by simply "putting things where they belong", without resorting to textual descriptions. We describe the simulation system, which models objects as polyhedra and emulates collision and contact interactions in combination with a simplified "operator-friendly" dynamics. The combination of contact simulation and graphical fixtures enables the operator to easily manipulate this virtual environment using a simple 2D mouse. We also describe the program generation system, which uses the operator's action sequence to create a set of robot motion commands which realizes the prescribed task.

1. Introduction

This paper describes work at the University of British Columbia (UBC) on the problem of using simulation as a tool for robotic programming. Our ultimate objective is to make programming a robot extremely easy and natural, so that it can be done intuitively by non-specialists using simple desk-top computer systems.

Particular emphasis is directed at tasks involving contact, since, at the time of this writing, there are virtually no commercially available robot programming systems which support contact-based tasks. While the reasons for this are complex, we suspect that programming difficulties are a significant part of the problem.

We believe that an effective way to program contact-based tasks involves having the operator perform the required task in a simulated virtual environment. The operator's actions are then interpreted and used to generate a robot program which replicates the indicated task at the actual work site.

Using a simulated environment for programming allows the operator to directly "show the system" where an object is to be placed or moved in relation to other objects. Textual descriptions of such tasks, can, by contrast, be very tedious. Also, within a simulated environment, we can adopt a "task-centric" view of programming, letting the operator manipulate workpieces directly, with less emphasis on the robot which will ultimately realize the task.

1.1. System requirements and overview

A simulation programming system requires several key subsystems:

1. *Model Generator:* builds and maintains the work site model used by the simulator;
2. *Task Simulator:* provides a simulation of the work environment, along with operator-friendly dynamics that makes task-specification easy and intuitive.
3. *Program Generator:* takes the motions specified in the virtual environment and creates a set of robot motion commands capable of realizing the prescribed task;
4. *Execution monitor:* verifies task execution at the robot site, and provides correction information to the model generator.

The second and third items are the main focus of this paper.

At UBC, we have built a test platform in accordance with the above structure. As a challenging but simplified example, we chose a task domain consisting of a puzzle of wooden blocks that can be assembled within a rigid frame. Tasks which the operator can specify include placing or moving a particular block in contact with other objects in the work site.

The model generator, described in [1], uses a gray-scale vision system that can rapidly recognize the location of objects characterized by straight-line edge features [2]. The simulation system (described in detail in Section 4) lets the operator manipulate blocks using inputs from a simple 2D mouse. We chose to use a mouse as an input device because it is cheap and ubiquitous, and matches with our above-stated desire for a system that can be used by non-specialists using simple desk-top computer systems. The program generator (Section 5) uses the motions specified by the operator to create a set of robot motion commands capable of realizing the specified task. Execution monitoring is the subject of ongoing investigation.

1.2. Paper outline

The remainder of this paper is organized as follows: Related work is discussed in Section 2, and a summary of the test platform components and hardware is given in Section 3. Task simulation and program generation are described in Sections 4 and 5, and experimental results are presented in Section 6.

2. Related Work

Our work closely follows the *teleprogramming* work of Funda, Sayers, and others [3, 4], in which operator interaction with a simulated environment is used to overcome problems which can arise in telerobotic systems due to time delays between the remote site and the operator station. Our work differs in that the simulation is somewhat more complete, and the system is more “task oriented”: the sequence of operator motions may be modified radically before being sent to the robot as motion commands.

Teleprogramming was predated by the use of graphical simulation as an operator aid in time-delayed telerobotic applications [5, 6]. The introduction of synthetic “fixtures” into the operator’s display to assist in task specification has also been considered [7, 8]. While commands sent to a remote site in teleprogramming systems tend to be

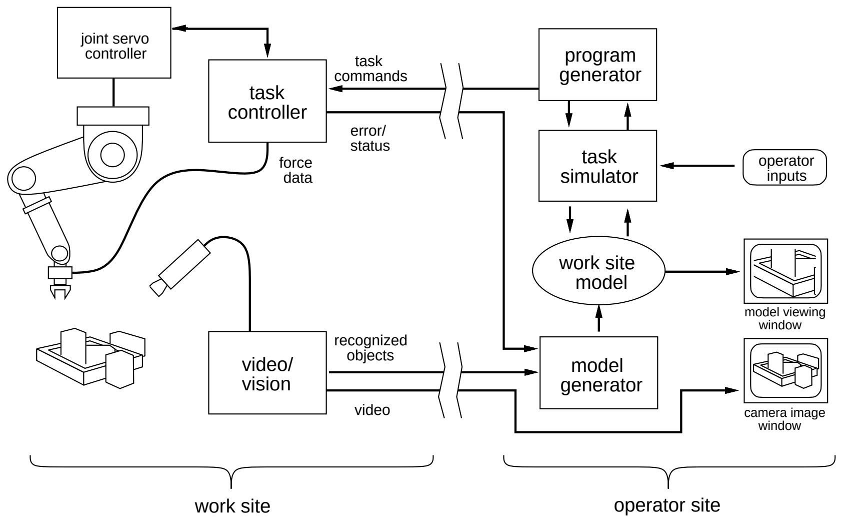

Figure 1. System architecture.

at the level of "guarded moves", the idea of sending more robust commands, which can allow the manipulator system to plan for and recover from unanticipated contact states, is investigated in [9], using a Petri-Net-based contact state model.

Virtual reality simulation has also been used as a platform on which fine motion task skills can be learned; a good example in the context of this paper is given in [10].

It should also be mentioned that our general goal of making robotic systems usable by non-expert operators has recently been explored in the context of position-based robotic applications accessible on the World Wide Web [11, 12, 13].

3. System Description

A somewhat simplified block diagram of the UBC system is given in Figure 1. The main components will be summarized very briefly here; a more detailed description is given in [1].

3.1. Work site

The work site (Figure 2) 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) [14] 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 [15].

A *video/vision* module continuously collects images from a camera and processes them using a model-based vision algorithm [2] to locate objects in the scene. The objects and their positions are continuously sent back to the operator site where they are

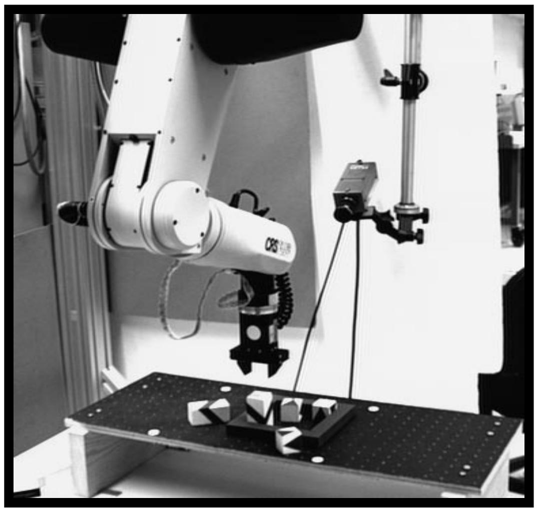

Figure 2. Remote site, showing the robot, camera, and work area.

incorporated into the work site model under the control of the operator. The camera image itself is also transmitted back to the operator site, where it is displayed in a separate window.

As mentioned above, the work site's task domain consists of a puzzle of wooden blocks that can be assembled within a rigid frame.

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 4). 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 generator* (which at present is under operator control [1]). The operator views the model through the *model viewing window*, implemented using the SGI 3D modeling package Open Inventor [16]. A *program generator* (Section 5) creates the robot motion commands required to realize specific tasks and sends them to the task controller at the work site.

4. Task Simulation

The task simulator moves a selected object, or *workpiece*, around inside the work site model, in response to operator inputs. Contacts and collision dynamics are modeled, allowing the workpiece to bump into, slide along, and realign 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.

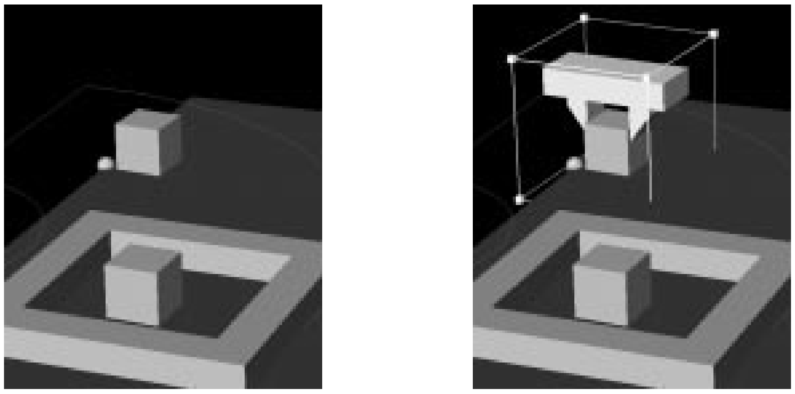

Figure 3. Dragger fixtures. After a part is selected by "clicking" on it with a mouse, a rendering of the manipulator's gripper appears, along with a graphical "dragger box" (right figure). Dragging the mouse cursor along one of the planes of the box causes a displacement parallel to the plane, which is converted into a "virtual force" acting on the workpiece's center.

The operator can also request that the workpiece specifically 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 (this is implemented using barrier functions, described in Section 4.2).

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" (at present, a box) that maps 2D mouse inputs into 3D spatial motions and permits the workpiece to be moved about (Figure 3). 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. When the workpiece is brought into contact with other objects, normal forces arise in reaction to the applied force. The normal forces plus the applied force create a net total force on the workpiece, from which a simple first-order dynamic model is used to compute the workpiece velocity. First order dynamics is used deliberately, because it (a) is inexpensive to implement, (b) is stable, and (c) produces results which are simple and intuitive for the operator.

4.1. Implementing the contact model

The simulator keeps track of the distances between objects using I-COLLIDE [17]. 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[1]. Reaction forces $\mathbf{f}_i$ acting along the contact normals, in response to the applied force $\mathbf{f}_a$, are determined using Baraff's algorithm [19]. The net

[1]Face-face or edge-face contacts can be reasonably simulated using a finite set of point contacts; see [18].

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 once per time step (currently every 50 msec) and uses this to update the workpiece's position, as described in the next section.

4.2. Preventing Collision using Barrier Potentials

When the workpiece is extremely close to other objects, second order constraints or numerical errors can make collision-free motion impossible, even when a feasible velocity exists. The effect of this is to have the workpiece appear to "stick", unreasonably, at certain configurations. Moreover, the information returned by I-COLLIDE becomes unreliable when objects are very close together.

Good simulator performance thus requires trying to keep the workpiece a minimum distance ϵ_b away from other objects, where typically $\epsilon_b = \epsilon_c/2$. This, in turn, is accomplished using a potential barrier.

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 $U_i(d_i)$ (similar to that used in [20]) 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.

To create motion, the desired spatial velocity from equation (1) is in turn associated with an attractive potential U_v that decreases uniformly along the velocity direction in proportion to the work done by $\mathbf{f}$ and $\mathbf{m}$. If motion in the direction of $(\mathbf{v}^T \boldsymbol{\omega}^T)^T$ is parameterized by s, such that $s \in [0, 1]$ corresponds to one simulator time step of Δt, then

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

Summing U_v and the U_i for all appropriate objects yields a net potential $U(s)$ that varies along the direction of motion. During each simulation step, the workpiece is moved so as to minimize $U(s)$. If there are no obstacles nearby, all $U_i = 0$ and this minimum will occur at $s = 1$, corresponding to the uninhibited application of $\mathbf{v}$ and $\boldsymbol{\omega}$ for time Δt. 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.

To help keep each $d_i \geq \epsilon_b$, the 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.

Why not treat the entire problem in terms of potential minimization and calculate both $\mathbf{v}$ and $\boldsymbol{\omega}$ from the gradient of U with respect to the workpiece's overall spatial position? The problem is that this gradient is not simple to calculate. 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 4.1) can be thought of as simply a good way of estimating this direction.

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

5. Program Generation

Once a workpiece has been satisfactorily positioned within the work site model, the operator indicates this to the system using a keystroke. The program generator then sets about creating robot motion commands to realize the requested operation and replicate any specifically requested contact motions.

The system records the workpiece's position and contact state after each simulation step. Each of these positions and contact states defines a *node* along a sequence called the *workpiece path*.

One way to achieve a required task would be to generate a separate robot motion command for each of the nodes on the workpiece path; in other words, closely replicate the operator's actions (similar to what is done in teleprogramming environments [3, 4]). 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 motions are constrained to directions permitted by the dragger fixtures, the resulting 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 for alignment.

It should be noted, however, that the workpiece path *is* collision free (within the resolution limits imposed by the simulator's step size). What we need to do is modify the workpiece path so as to remove unwanted motions and unnecessary contacts, while preserving its feasibility.

5.1. Stretching and Shrinking the Path

Such a modification can be done by treating 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. To achieve this, we apply to each of the path 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 well as 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 4.2). To keep nodes from processing along the path, we eliminate any repulsive force component which is tangential to the path.

It is important that the deformed path remains collision free and preserves any required contacts. This is achieved by moving each node using the contact simulation software, with the combined tension and repulsive forces assuming the role of the applied force $\mathbf{f}_a$. Each node in the path is moved in succession, with the whole procedure being repeated until the path stablizes. After the deformation is complete, the number of nodes is reduced using a simplification scheme similar to a polygonal path approximation algorithm.

Our path deformation approach closely follows the work of Quinlan [21], who originated it to simplify and smooth collision-free paths in configuration space produced by a motion planner. Our situation differs in that we specifically include contact states, and nodes are moved using contact simulation software. In Quinlan's work, collision avoidance was guaranteed by surrounding path points with free space "bubbles", which would be difficult to calculate here because of our use of spatial coordinates and the proximity of obstacles in contact, which would require computing bubbles at extremely fine resolutions.

5.2. Robot Motion Commands

The robot command sequence for a task begins with a "guarded grasp" with which the manipulator grasps the workpiece at the location corresponding to the start of the path. Force sensor data is used to help execute and verify this command.

A motion command is then created for each of the succeeding path nodes, specifying a spatial position which the robot should move to (using linear spatial interpolation). For nodes which involve contact or pass near objects, the robot's speed is lowered, and its impedance is controlled to emulate a spring-damper system with low stiffness. Contact is ensured by adding to the target position a small translational bias (currently around 3-4 mm) in the directions of contact, and contact is verified by checking for observed forces along the directions of the contact normals. Contact instabilities are minimized 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 within one control cycle).

6. Demonstrations and Observations

Figure 4 illustrates the behavior of the system 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 Figure 5.

The task simulator runs easily in real-time (20 Hz) on a 15 MFlop SGI Indy. A barrier size of $\epsilon_b = 1$ mm was used (with a work area diameter ≈ 350 mm and a typical object dimension of ≈ 30 mm). The golden section search (Section 4.2) was performed

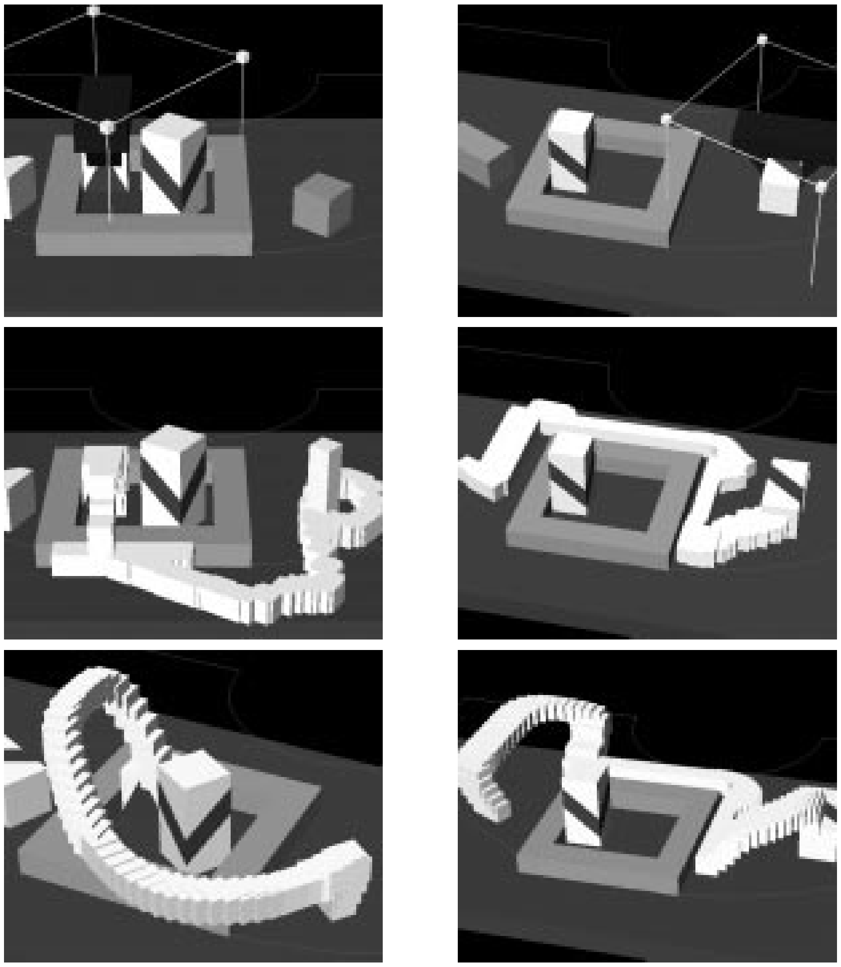
Figure 4. **Left column:** Cornering. The operator has placed the workpiece in a corner of the frame (top); its original location is shown by a gray "shadow block" at the right. The outside of the frame was used to help align the workpiece, as can be seen from the initial workpiece path (middle), where the tightly spaced nodes are shown as half-sized white blocks. After modification (bottom), 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. **Right column:** Here, the operator has moved the workpiece around the outside of the frame (top), while requesting that contact with the frame and table be maintained while going around the corner. The resulting initial workpiece path is shown in the middle. After the path is modified (bottom), frame/table contact is maintained near the corner, while nodes are pushed away from the table elsewhere.

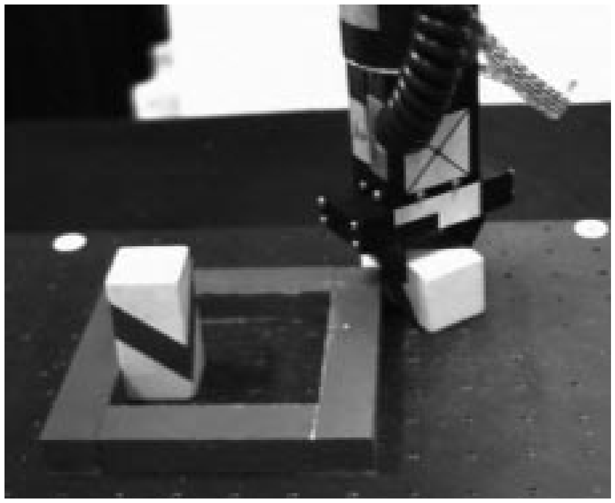

Figure 5. Execution of the corner-rounding task of Figure 4.

to an accuracy of about 1.0^{-5}, requiring about 25 I-COLLIDE calls per simulation 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) and so improvement is needed here. Overall performance and smoothness of the contact simulation is greatly enhanced by the use of barrier functions (Section 4.2). However, when the workpiece is constrained to maintain contact with an object, 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 [22]). 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.

The path modification algorithm (Section 5.1) produces paths that make good intuitive sense. In particular, it is particularly good at placing node points so as to ensure reliable and "snag free" approaches and departures from contact situations. The only caveat is that path nodes must be placed fairly close together (currently on the order of 15 mm) 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. In general, the whole problem of maintaining proper node spacing requires additional work.

Finally, while not the primary focus of this paper, actual path execution turned out to be quite robust, using the simple methods described in Section 5.2.

7. Conclusion

We have developed an integrated system linking together vision, contact simulation, localized planning, and manipulator control, with the intention of creating an environment in which programming contact and part mating tasks is extremely easy.

The use of a simulated environment appears well suited to programming contact-based tasks, since the work space objects themselves can then serve as "virtual fixtures" that help guide the operator's actions. Indeed, the combination of dragger fixtures plus contact constraints tended to make part placement quite easy, even when using an ordinary mouse as an input device. In contrast, we discovered during earlier experiments that the manual selection of free space motion via-points between goals turns out to be fairly tedious for an operator. However, such points were easily produced by the system using the methods of Section 5.1.

For future work, we wish to add additional local planning to make the contact motion sequences more robust, and to improve the system's abilities so that it can handle contact operations involving tight fits.

References

[1] J. E. Lloyd, J. S. Beis, D. K. Pai, and D. G. Lowe, "Model-based Telerobotics with Vision". *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, Albuquerque, New Mexico, April 23-25, 1997.

[2] J. S. Beis and D. G. Lowe, "Learning indexing functions for 3-D model-based object recognition," *IEEE Conference on Computer Vision and Pattern Recognition,* Seattle (June 1994), pp. 275-280.

[3] J. Funda, T. S. Lindsay, and R. P. Paul, "Teleprogramming: Toward delay-invariant remote manipulation". *Presence*, Winter 1992, pp. 29–44 (Vol. 1, No. 1).

[4] C. R. Sayers, "Operator Control of Telerobotic Systems for Real World Intervention". Ph. D. thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104 USA, 1995.

[5] G. Hirzinger, B. Brunner, J. Dietrich, and J. Heindl, "Sensor-Based Space Robotics – ROTEX and Its Telerobotic Features". *IEEE Transactions on Robotics and Automation*, October 1993, pp. 649–663 (Vol. RA-9, No. 5).

[6] A. K. Bejczy, W. S. Kim, and S. C. Venema, "The Phantom Robot: Predictive Displays for Teleoperation with Time Delay". *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, Cincinnati, Ohio, May 1990, pp. 546–551.

[7] C. R. Sayers and R. P. Paul, "An Operator Interface for Teleprogramming Employing Synthetic Fixtures". *Presence*, Fall 1994, pp. 309–320, (Vol. 3, No. 4).

[8] National Research Council (U.S.A.), *Virtual Reality. Scientific and Technological Challenges*. National Academy Press, Washington, D.C. 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". *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* Pittsburgh, Pennsylvania, August 1995, pp. 1–6 (Vol. 2).

[10] R. Koeppe and G. Hirzinger, "Learning Compliant Motions by Task-Demonstrations in Virtual Environments". *Experimental Robotics IV (Lecture Notes in Control and Information Sciences No. 223)*, Springer, London, pp. 299–307.

[11] E. Paulos and J. Canny, "Delivering Real Reality to the World Wide Web via Telerobotics". *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, Minneapolis, Minnesota, April 1996, pp. 1694–1699.

[12] `http://cwis.usc.edu/dept/garden/`.

[13] `http://telerobot.mech.uwa.edu.au/`.

[14] J. E. Lloyd and V. Hayward, *Multi-RCCL User's Guide*. Technical Report, Center for Intelligent Machines, McGill University, April 1992.

[15] M. Pelletier and M. Doyon, "On the Implementation and Performance of Impedance Control on Position Controlled Robots". *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, San Diego, California May 8-13, 1994, pp. 1228–1233 (Vol. 2).

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

[17] J. Cohen, M. Lin, D. Manocha and K. Ponamgi, "I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scaled Environments". *Proceedings of ACM Int. 3D Graphics Conference*, 1995, pp. 189–196.

[18] S. Goyal, E. N. Pinson, and F. W. Sinden, "Simulation of Dynamics of Interacting Rigid Bodies Including Friction I: General Problems and Contact Model". *Engineering with Computers*, Springer-Verlag, London, 1994, pp. 162–174 (Vol. 10).

[19] D. Baraff, "Fast Contact Force Computation for Nonpentrating Rigid Bodies". *SIGGRAPH 94 Conference Proceedings*, July 1994, pp. 23–34.

[20] R. J. Spiteri, U. M. Ascher, and D. K. Pai, "Numerical Solution of Differential Systems with Algebraic Inequalities arising in Robot Programming". *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, Nagoya, Japan, May 1995, pp. 2373–2380.

[21] S. Quinlan, *Real-time Modification of Collision-Free Paths*. Ph. D. thesis, Computer Science Department, Stanford University, December 1994.

[22] D. Baraff, "Interactive Simulation of Solid Rigid Bodies", *IEEE Computer Graphics and Applications*, May 1995, pp. 63-75.