ROBOTIC ASSEMBLY & HAND DESIGN

by

Christian S. Sallaberger

Memorandum No. UCB/ERL M89/60

15 May 1989

ROBOTIC ASSEMBLY & HAND DESIGN

by

Christian S. Sallaberger

Memorandum No. UCB/ERL M89/60

15 May 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ROBOTIC ASSEMBLY & HAND DESIGN

by

Christian S. Sallaberger

Memorandum No. UCB/ERL M89/60

15 May 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract :

The paper is the result of experimental work performed on the Robotworld system. Robotic hand design is investigated, with an in depth analysis of the characteristics and advantages of compliant contact surfaces. Various assembly task techniques are also highlighted, and a characterization and solution of the multiple mobile robot planning problem are developed.

Contents

List of Figures

Chapter 1

Introduction

In today's world people are increasingly looking to robotics as the solution to their problems. Experts from almost all industrial sectors, and even specialists from such diverse fields as space exploration and nuclear engineering are demanding that robots perform complex tasks for them.

Often the result is disappointment. Today's robots are unable to do many of the tasks their users (and sometimes even their designers) want them to do.

If current robotic requirements are to be met, a substantial increase in robotic performance and a decrease in the cost/performance ratio will definitely be needed. One way to achieve this is by reformulating the current strategies for robotic design and programming. This paper examines some of the issues involved with robotic hand design and the performance of assembly tasks with these hands.

In the past, the design of a robot, and the planning and programming of particular robotic tasks were two separate and distinct functions. What is proposed is a unified approach which combines these two. In our case we are concerned with the design of robotic hands. Before even thinking about what the hands should be like, we examine the set of tasks which we would like our hands to do. This is then allowed to guide the design of the hand, and consequently the programming and performance of these and similar tasks can exploit the features built into the hand. It is important that this set of tasks used to guide the design stage is not too narrow so that the resultant hand is useless for any other tasks, and yet that the

set of tasks is not so large that it cannot dictate specific design features.

The system used for research into the problem of hand design and consequent assembly task performance was Robotworld by Automatix Inc. This system is an excellent testbed for this work because it allows for easy interchanging of robotic hands, supports multiple robots, and although only recently introduced into the marketplace, comes with a computing environment which allows relatively high level robot motion control commands.

Traditionally robotic systems have always been designed to be very rigid. It was thought that by making systems very rigid, all degrees of freedom could be controlled very precisely and thus the desired task could be performed. Following the same pattern of thought, objects being manipulated were held very tightly by the grippers, and therefore went exactly where the robot put them, and nowhere else. In this fashion external forces such as gravity had no effect on the objects, and could be ignored.

In actuality robots are never totally rigid, and if we acknowledge this we can compensate for this both in the design of robotic hands, and in the programming of the system, so as to make the task performance robust in the presence of small position errors. In fact in certain instances compliance can actually be a tremendous advantage, and we may wish to enhance the compliance of certain parts of the robot.

Similarly gravity can be thought of as a free force, and instead of trying to eliminate its effects, we could be trying to use them to our advantage. As long as we realize the force is there, we can easily handle and exploit it.

Chapter 2

Robotworld System Description

A Robotworld system was recently received by Robotics Laboratory at Berkeley. This system is used as a testbed for research into hand design and assembly tasks.

2.1 Hardware

Robotworld consists of a flat drive surface which acts as a ceiling for the world, a variable number of mobile robots which hover on the ceiling, and a work surface which acts as floor to the world. See Figure 2.1.

The mobile robot modules are electromagnetically attracted to the ceiling, and as the ceiling consists of a grid of permanent magnets the module (x,y) position is also controlled in this fashion. The (x,y) position accuracy is better than a thousandth of an inch. Modules can either be mobile cameras, in which case they have the two planar degrees of freedom mentioned above, or mobile robots, in which case they also have an arm that extends down in the z direction, with a two inch travel and rotational ability.

Modules are levitated from the ceiling by air bearings. The compressed air for this purpose is supplied through umbilical cables at 50psi. Control and power lines also travel through these cables.

The Berkeley system currently consists of two mobile robots, one mobile

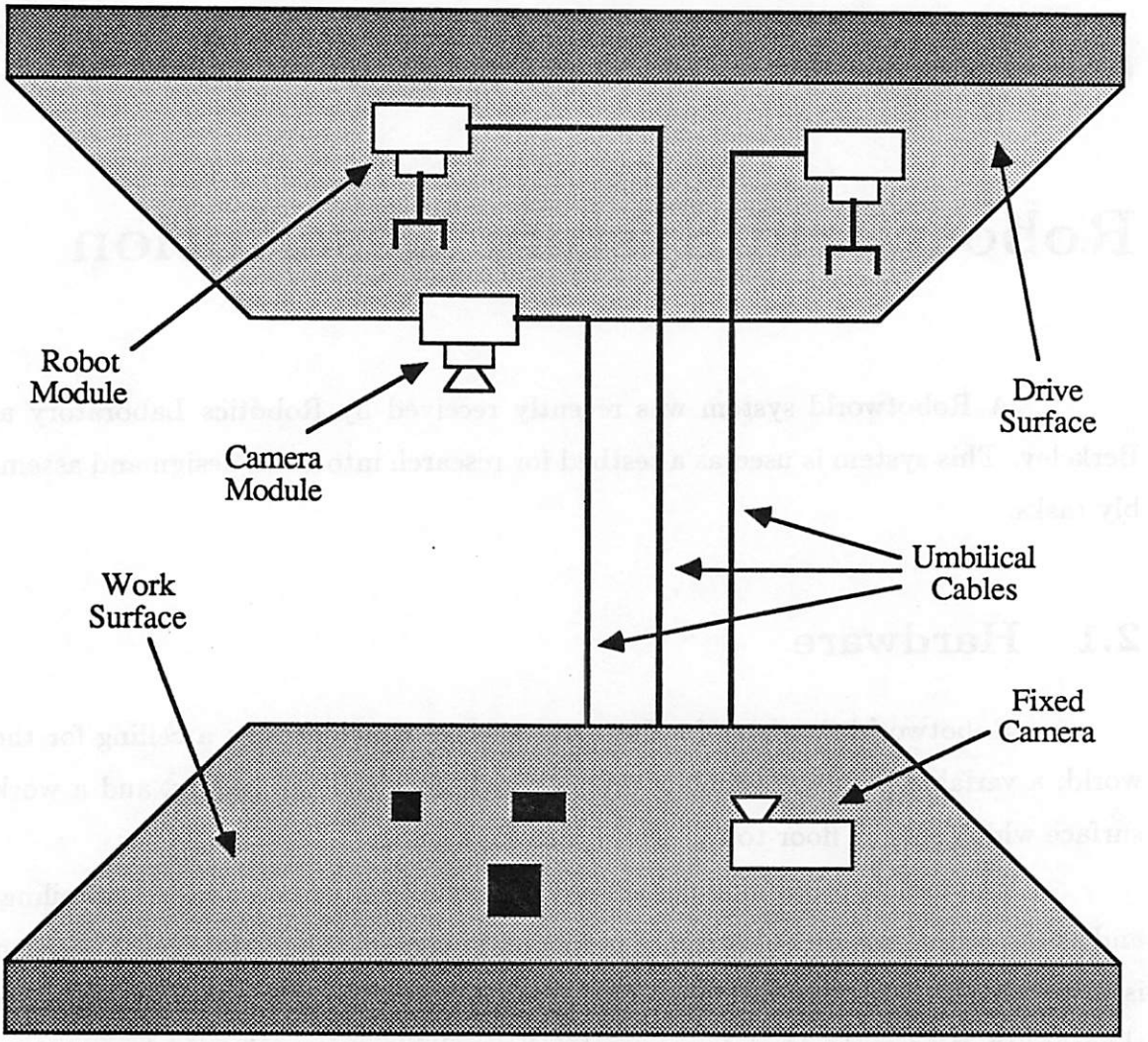


Figure 2.1: Robotworld Hardware

camera, and one fixed camera which is permanently mounted in the center of the work surface.

2.2 Software

The software and control hierarchy is shown in Figure 2.2. The system is controlled by an Apple Macintosh computer. Running on the Mac is a supervisory program called Rail which interacts with the user and also with the system. The system can be programmed in the high level Rail robotic programming language. Sample code can be seen in Appendix B.

The Rail program communicates with a Frame Grabber which has a direct parallel connection to both the mobile camera and fixed camera. This is a bi-directional link.

The Rail program also sends signal out over another parallel port which control TTL switches. These TTL switches can turn both the compressed air valves and motor amplifiers for each module on and off. The communication over this port is uni-directional.

The Rail program is also connected to an RS232 port. From this port a serial line carries high level instructions to the CP-499 Motion Control Board and feedback information from this board back to the Mac.

The CP-499 Motion Control Board features a 68000 cpu, and converts the high level commands to simple [-10V to +10V] low current input signals to the module amplifiers. Two lines are used to send signal for each of the x and y directions of motion, and one line for each of the z and theta degrees of freedom for each module. In addition each amplifier has four lines which send back information to the CP-499 about the current z and theta values. This gives a total of 10 lines per module amplifier. (Note that no position feedback for the (x,y) planar motion is present. Force feedback is also not present.)

Each amplifier outputs high current voltages which are sent directly to the module motors. One again 10 lines with the same functions as those above are used between the amplifiers and modules.

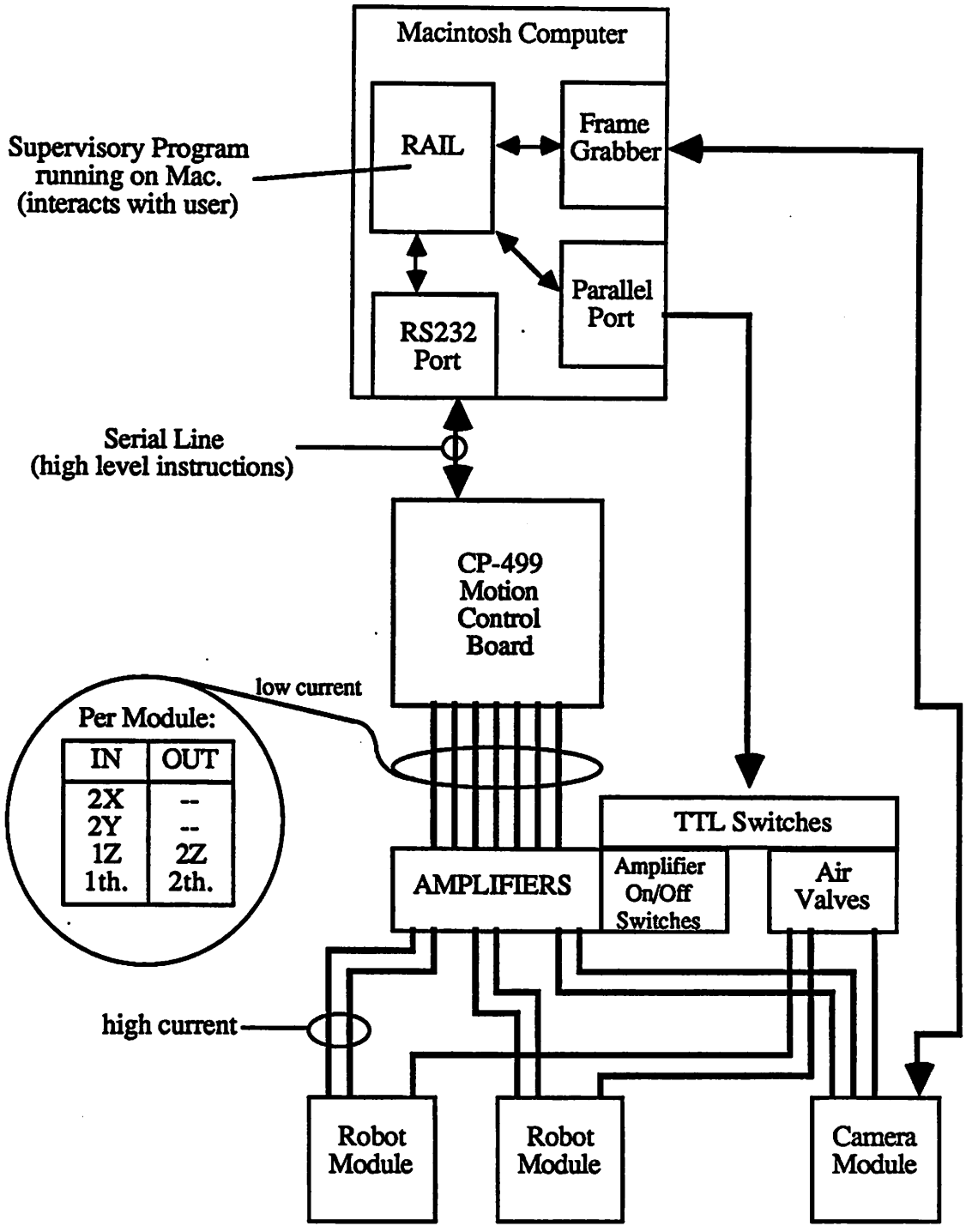


Figure 2.2: Robotworld Software and Control Hierarchy

2.3 Automatix

Robotworld is produced by Automatix, Inc. whose headquarters are in Billerica, Massachusetts. Automatix is known for its arc-welding robots which are used in industry to produce lawn mowers, metal equipment, and jet engines.

The west coast office located in Redwood City, California manufactures and sells the Robotworld systems, which have only recently been brought onto the market. Currently less than a dozen systems have been delivered.

Chapter 3

Hand Design

3.1 Task Oriented Design

One of the keys to success in robotic hand design seems to be the adoption of a *task oriented design* approach. There is a philosophical difference between this and the traditional method of manipulator hand design. Instead of concentrating on the physical mechanisms of the hand being designed, one first concentrates on properly defining the tasks that the hand will be required to do. From these a set of task requirements are extracted, which in turn are translated into aspects of the physical/mechanical design.

The translation from task requirements to physical hand design requirements is a non-trivial function. Initial investigations with robot world hand design do indicate, however, that a mapping from task requirements to design requirements can be defined. A sample set of mappings is shown in Table 3.1 on the next page. A database of mappings of the type shown in the table could be constructed and could lead to mechanization of the design process.

Once the design requirements have been generated they are combined as illustrated in Figure 3.1. It should be noted that design requirements may conflict with one another, and in this case different parts of the hand can satisfy the various requirements. If this is not possible, then either the task needs to be redefined, or separate hands are built, each of which can execute a sub-task.

Task Requirement	Hand Design Requirement
part alignment on table top	hard smooth surface required on hand
force application without force feedback availability	highly compliant surface required
part orientation with gravity	A. pivot point required B. hand should not restrict rotation of part in desired direction

Table 3.1: Some Task to Design Mappings

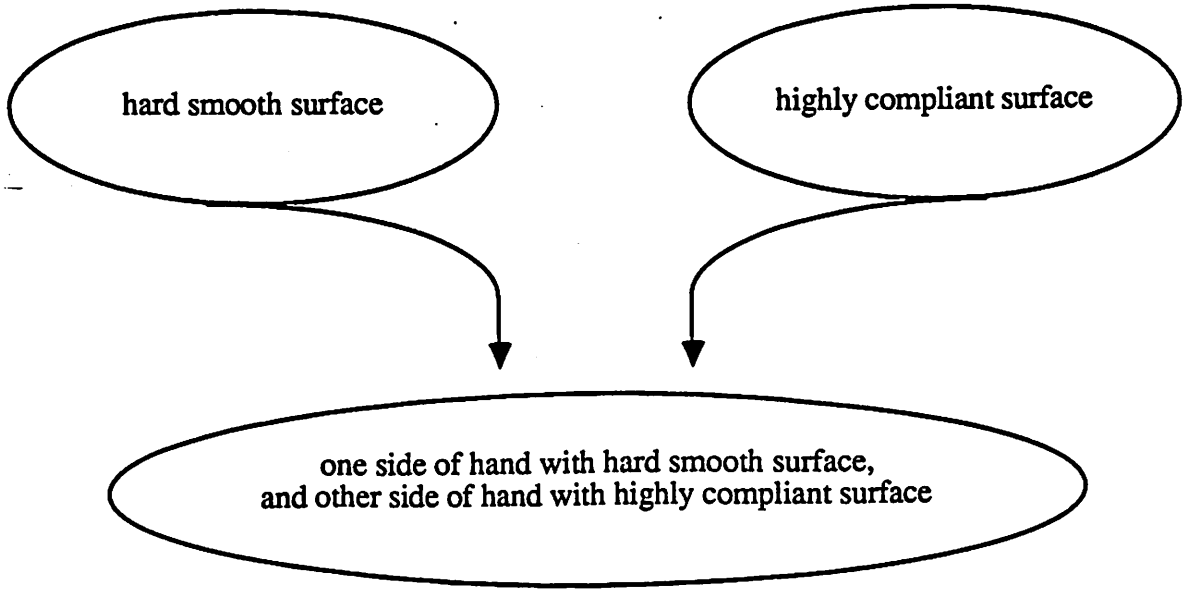


Figure 3.1: Example of Combination of Design Requirements

The entire design process is summarized in the flow diagram of Figure 3.2. Initially a detailed task description is built. From this description the set of Task Requirements are extracted. It is possible that this extraction could be accomplished automatically by an expert system with a knowledge database. The implementation of this is left to the Artificial Intelligence experts.

The set of Task Requirements (either generated manually or by machine) is then translated into Hand Design Requirements. Once again, this step could be automated with a detailed list of task requirement to hand design requirement mappings and a look-up or hash table. The resultant Hand Design Requirements are then combined and a hand is designed and built.

One the hand design is completed, the robotic system can be programmed to do the initial task. This programming step is greatly simplified by the existence of the sets of Task and Design requirements which were developed in previous stages of the process.

This systemization of the hand design process can be expected to efficiently reduce the time required between the arisal of a goal task in a sophisticated robotic system which will require a new hand, and its accomplishment.

3.2 Simplicity

Simplicity is another key to success in hand design. The hand that is designed should be as simple a hand as possible which simultaneously satisfies all the design requirements. Of course the fulfillment of these requirements tends to add to the complexity of the hand, so in effect, what one tries to do is to minimize the complexity of the hand subject to the constraints implied by the Task Requirements through the Hand Design Requirements.

The merits of a simple design were found during experimentation with various prototype hands. Complex designs were more limited in their use with new tasks or task variations than their simpler counterparts. Complicated hands also tended to lack the robustness and performance of simpler hands.

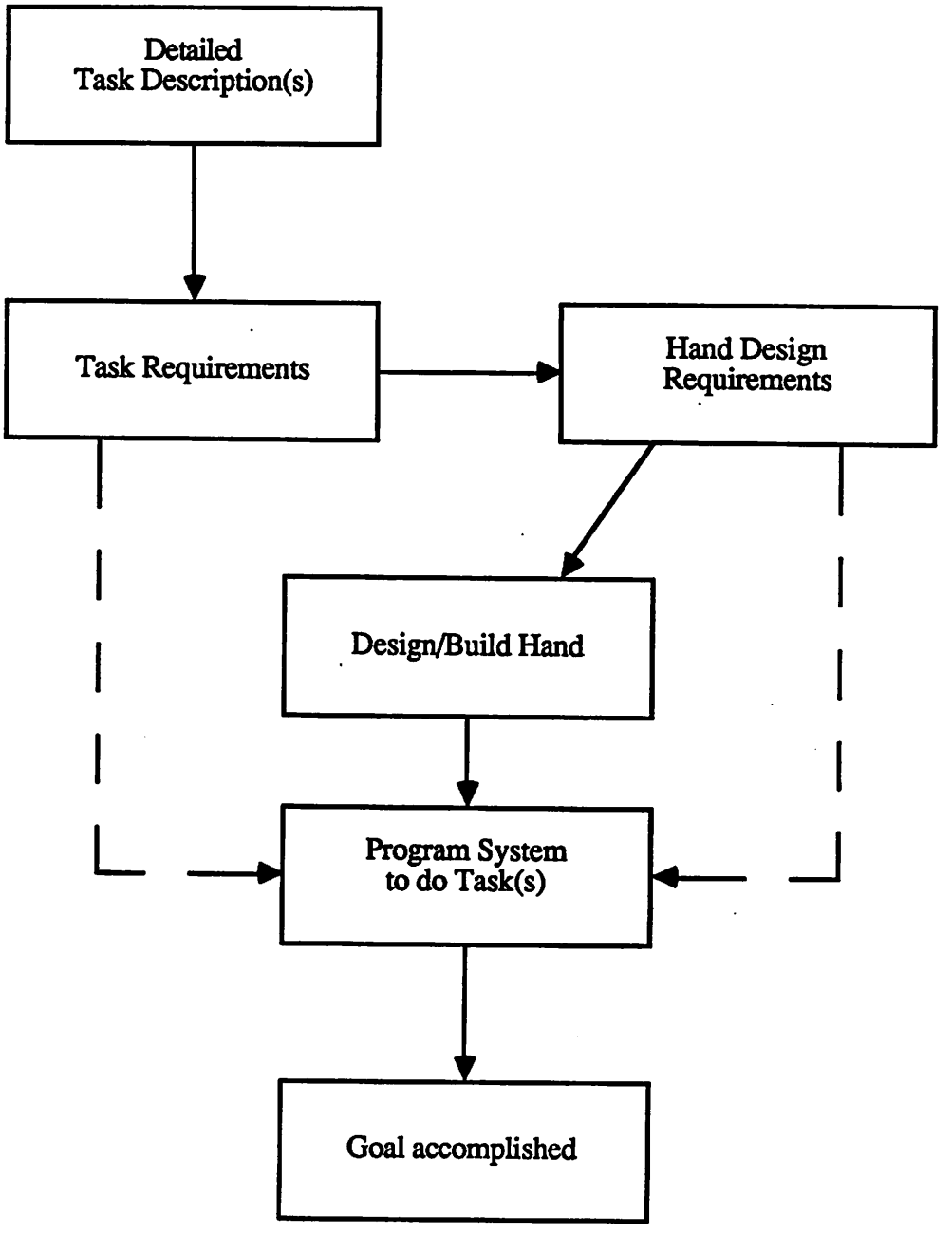


Figure 3.2: Flow Diagram of Design Process

For the most part, the simpler one keeps the design of a hand for a given task (or set of tasks), the more likely it is that it will be general enough to do a wider range of similar tasks, as well as other unrelated tasks.

Another type of simplicity is ease of construction. This is especially important in early prototyping work, where it promotes cheaper and faster construction of hands, and also allows more experimentation.

3.3 Compliance

Physical compliance is a feature which, if designed into a robotic hand, can greatly enhance its manipulative abilities. Not only do compliant surfaces increase the grasping dexterity of a robotic hand, they are also very useful for operations requiring force application.

3.3.1 Advantages of Compliance at Contact Points

The surface characteristics of a robotic hand are only important at the points of contact between the hand and the objects that it is manipulating. These are commonly called the contact points.

Let us first consider the case of a simple finger contact with the surface of an object. It is easily shown that a single contact point without friction can only exert forces perpendicular to the surface. Any forces the finger tries to impart to the surface which are tangential to the surface will only result in the finger sliding across the surface.

When a friction is present the single point contact will also be able to exert forces with a limited tangential component. This tangential component is limited by the expression:

$$F_{tangential} \leq \mu \times F_{normal} \tag{3.1}$$

where μ is the coefficient of friction of the surface.

Any force $F = F_{tangential} + F_{normal}$ that satisfies inequality 3.1 is said to lie within the friction cone of the surface.

If the single contact point described above originates from a *compliant* finger tip, then it is called a soft finger contact. In addition to forces within the friction cone, moments about the normal to the surface can also be exerted on the surface. Another often overlooked characteristic of the compliant soft finger contact is that the soft contact increases the coefficient of friction μ and thus the friction cone is widened. This leads to the conclusion that if one wants a robotic hand to slide across the surface of an object (while being in contact with it), the compliance of the hand should be as low as possible, and the hand should be made of low coefficient of friction surfaces.

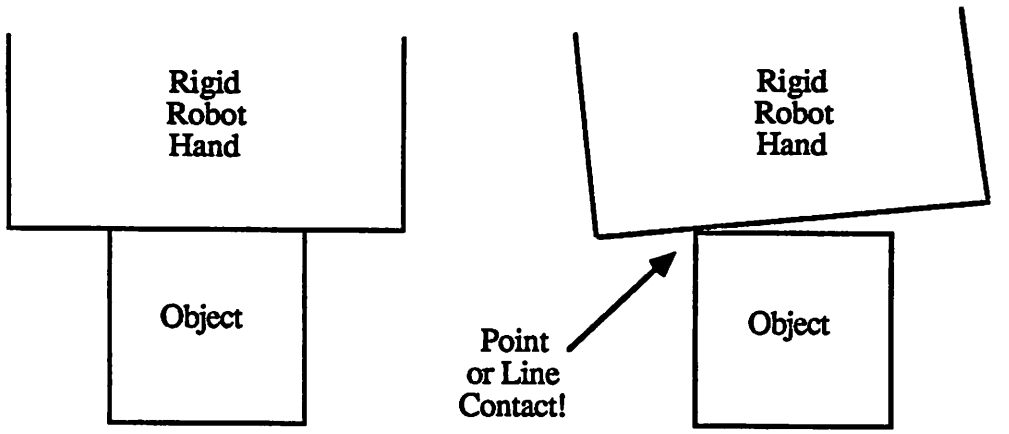
It can be the case that the contact between the hand and the object it is manipulating is not a point, as considered above, but rather a plane. In this case there is an infinity of adjacent contact points such that there is a finite area of contact between the hand surface and the object surface. If this contact area is frictionless, then forces perpendicular to the object surface as well as moments in any direction perpendicular to the surface normal can be exerted by the hand on the object.

If in addition the contact area has friction, then forces in any arbitrary direction (provided they are not away from the object!), and moments about any arbitrary axis can be exerted by the hand on the object. This is often a very desirable situation when the hand is manipulating an object.

Unfortunately slight orientation changes between two rigid surfaces will often result in a potential contact area becoming a simple contact line or point. This can, however, be avoided with the introduction of a compliant surface as one of the two contact surfaces. In our case the robotic hand has a compliant contact surface, and a solid area contact between the hand and the object can robustly be maintained. This is illustrated in Figure 3.3.

In summary, compliance is critical to contact point characteristics. If one wishes to apply a torque about a contact point, some compliance at the contact point is required. A compliant contact point also enlarges the contact friction cone. The maintenance of a robust contact area is ensured by the use of a sufficiently

Rigid surfaces lose area contact with small perturbations:



If robot hand surface is compliant, area contact is maintained:

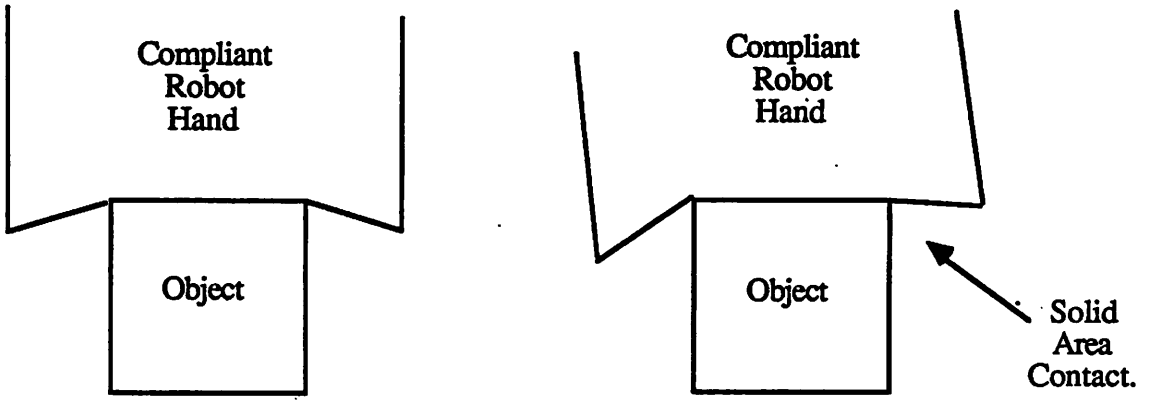


Figure 3.3: Robust Maintenance of Area Contact with Compliant Surface

compliant surface as one of the two contact areas. A sliding contact point requires a very low compliance.

3.3.2 Compliant Surfaces

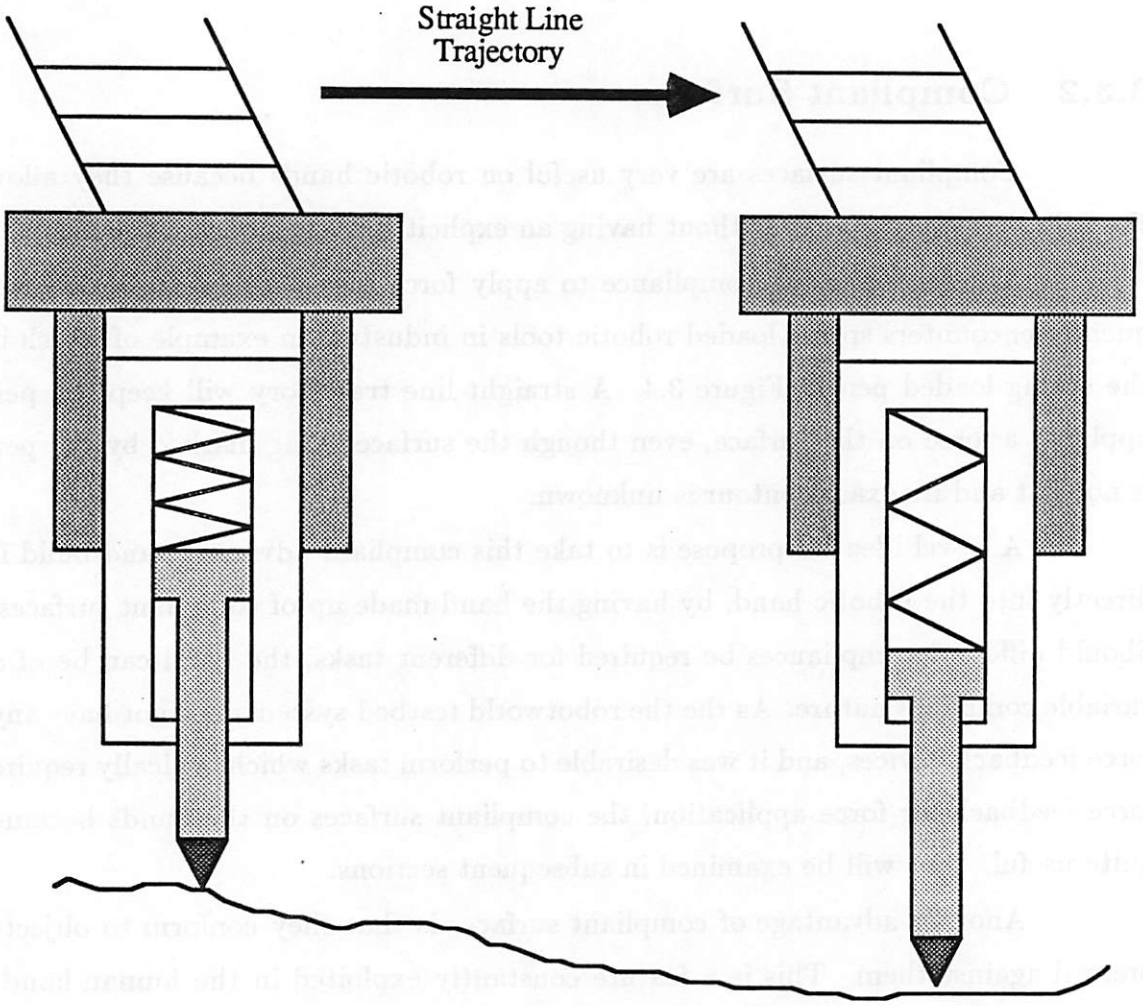
Compliant surfaces are very useful on robotic hands because they allow the robot to apply a force without having an explicit force feedback mechanism.

The idea of using compliance to apply force is not a new one. One frequently encounters spring loaded robotic tools in industry, an example of which is the spring loaded pen in Figure 3.4. A straight line trajectory will keep the pen applying a force on the surface, even though the surface being marked by the pen is not flat and its exact contour is unknown.

A novel idea we propose is to take this compliant advantage and build it directly into the robotic hand, by having the hand made up of compliant surfaces. Should different compliances be required for different tasks, the hand can be of a variable compliant nature. As the the robotworld testbed system does not have any force feedback devices, and it was desirable to perform tasks which typically require force feedback for force application, the compliant surfaces on the hands became quite useful. This will be examined in subsequent sections.

Another advantage of compliant surfaces is that they conform to objects pressed against them. This is a feature constantly exploited in the human hand. When one holds a pencil between two fingers, one's fingers conform to the contour of the pencil at the contact areas, and a firm grasp is set up. Similarly a baseball pitcher grasps a baseball firmly along the stitching of the ball so that the fingers grasping the ball conform to the stitching pattern and thus a solid grasp of the ball is achieved. This compliance in the fingers sets up the friction pattern necessary for the pitcher to throw a curve ball.

Compliance also can play an important role in assembly tasks that require one robot to come into contact with another. This can happen either directly or via a third object between them. This is common when two robotic hands are required to hold an object together. If the robotic hands have rigid surfaces any small



Pen always exerts a force on the surface, even though the exact surface contour is unknown. This is because of the spring in the pen.

Figure 3.4: Force Application by Spring Loaded Pen

position errors in either robot can be disastrous in a position control environment. (i.e. one in which forces cannot explicitly be measured.)

This discrepancy between the performance of rigid and compliant surfaces is illustrated in Figure 3.5. Note how the two compliant robots can grasp the object robustly in the presence of both positive and negative position errors, while the rigid robots fail. This feature of compliant surfaces is exploited in the Nut & Bolt assembly operation discussed in chapter 4.

Finally it should be noted that a compliant surface need not have uniform compliance in all directions. For example compliance of a single surface in two directions (say x and z) can be very high, while compliance in the other (y) direction can be low. This would be very useful in the scenario where a soft x compliance allowed the surface to conform to the object, the soft z compliance allowed inacurracies in this direction to be tolerated, while the stiff y direction allowed a more direct connection between robot movement in this direction and the application of a force on the object in this direction. (See Nut & Bolt assembly in section 4.2 for a practical example of this.)

3.3.3 Variable Compliance Hands

To experiment with robotic assembly task operations using compliant hands two *variable compliance hands* were built for the robotworld system. The name stems from the fact that the manipulative surface of the hands is actually divided into four surfaces, each with its own compliance characteristics. One can then use the surface with the appropriate compliance nature for each task.

This is very much the way the human hand is used. When the hand is used to knock on a door, the low compliant knuckles are used. When one wishes to apply a lot of torque with a screwdriver, the soft fleshy palm (high compliance) is used to hold the handle and establish a firm grasp. When one wants to scratch a surface, the almost zero compliant nature of finger nails is exploited.

A diagram of the cross section of the variable compliance hand is shown in Figure 3.6. (A more detailed plan of the hand is given in Figure A.1 of Appendix

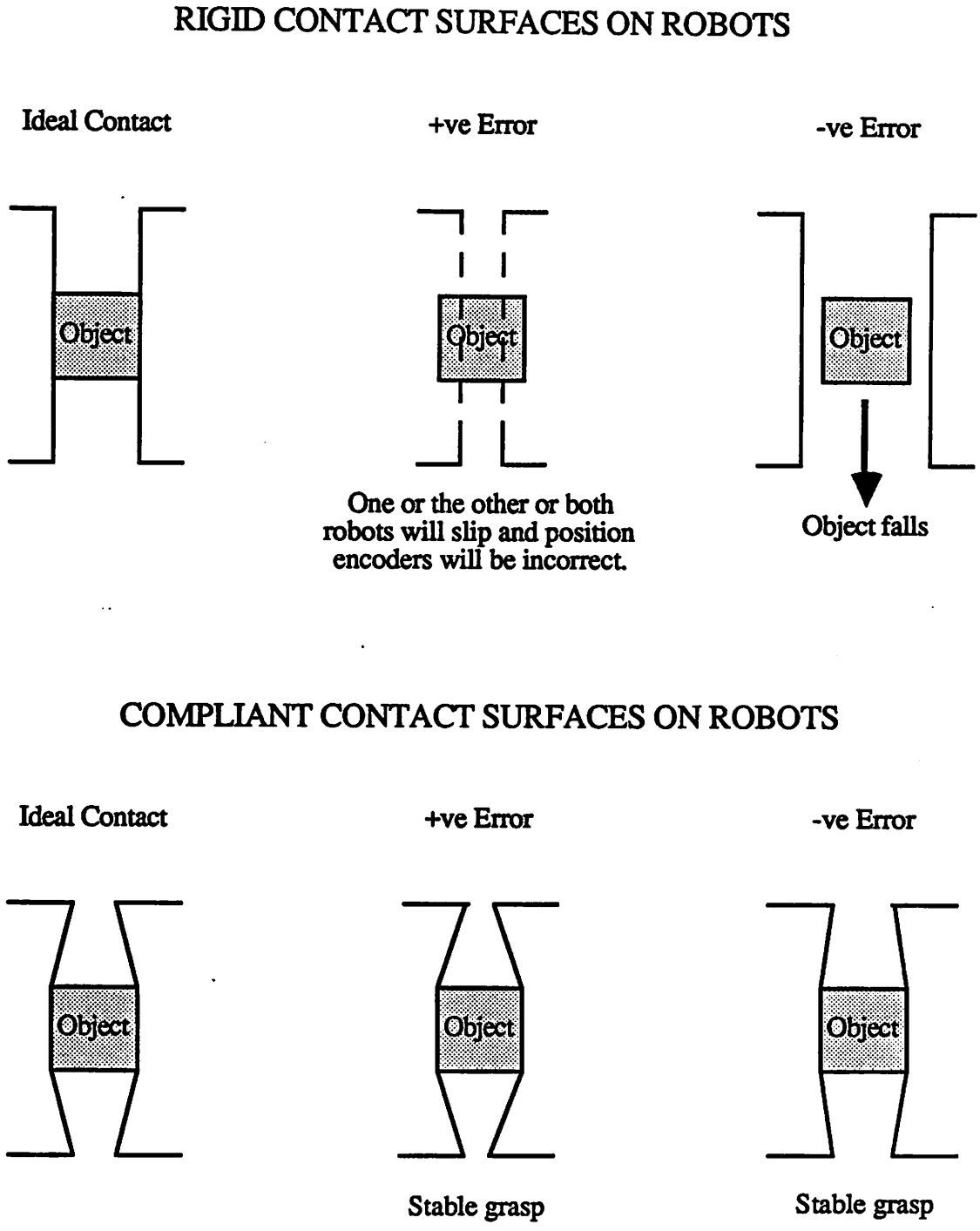


Figure 3.5: Rigid vs. Compliant Contact Surfaces

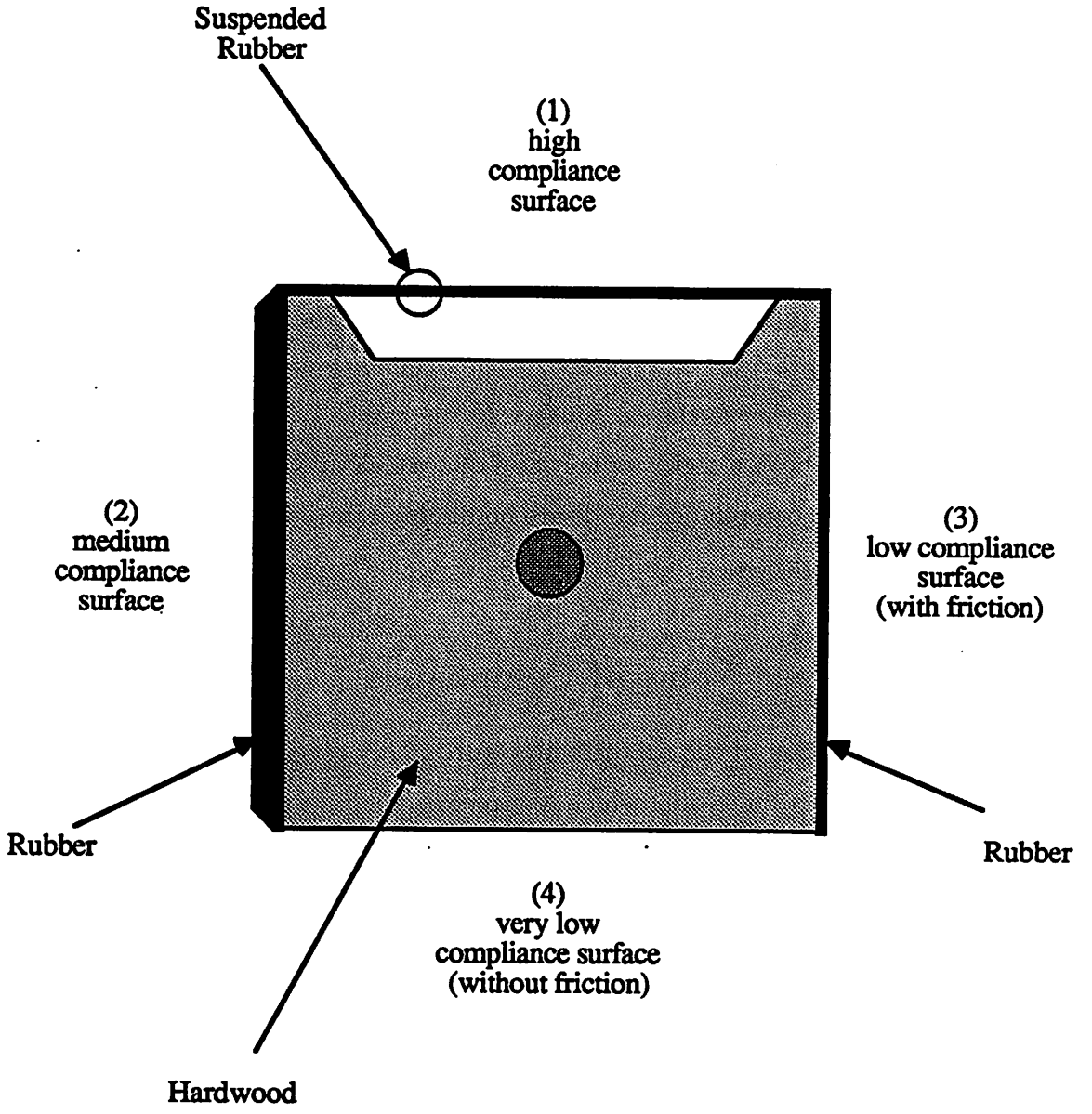


Figure 3.6: Cross Section of Variable Compliance Hand

A.)

On the variable compliance hand, the surface (1) in Figure 3.6 is the high compliance surface. Contacts using this surface will tend to conform to the object being manipulated. This surface is also ideal for force application while using position control of the robot.

Surfaces (2) and (3) are the medium and low compliance sides respectively. The two vary in the thickness of the their surface rubber layers. While side (3) has a fairly low compliance value, it has the advantage of being able to impart a more rigid support to the contact, due to the closeness of the underlying hardwood layer.

Surface (4) is a very low compliance surface. The total absence of any rubber coating also provides this surface with a much lower μ value (frictional coefficient), than the other surfaces. Surface (4) is typically used to align objects on a table top, when both a rigid contact and one which allows sliding is required.

The variable compliance hands are highlighted in the nut & bolt assembly task discussed in Chapter 4.

3.3.4 Generalized Value of Compliance

An examination of the robotics literature today reveals that compliance is usually viewed either as something which is approximated away with point contacts, or as something which is an undesirable side effect of the construction of tactile sensors.

Neither of these exploit the inherent value of compliance. Compliance is not something which holds us back, but which makes possible more stable robotic grasps, allows the transmission of moderate forces, and generally makes a robotic system more robust is the presence of position errors.

This latter feature is very relevant, because today most robots are programmed as if they were a chain of rigid bodies. This is in fact not true, and all robots have some flexing and stretching characteristics. If we acknowledge the existence of this compliance and exploit it, we can design more robust robot control programs.

A logical extension of this is that we will no longer rely on joint encoder outputs and rigid approximations of our robots to determine their positions, but investigate other feedback mechanisms such as vision systems, laser rangefinder/sonar systems, force/moment sensors, etc.

Chapter 4

Assembly Task Techniques

In this chapter we examine some of the issues in robotic assembly tasks.

4.1 The Free Gravity Force

Traditionally robotic systems were designed to be as physically rigid as possible, and when the system was programmed to perform a particular assembly task, all degrees of freedom were precisely controlled. Objects were grasped firmly by the robots, and were not allowed to move. In this fashion the effects of all external forces were eliminated.

There is one external force affecting all robotic systems on earth, namely the force of gravity. As gravity has a very precisely defined force field for all robotic systems, it need not be viewed as a necessary evil which must be overcome, but rather as an inherent feature of the environment of the robot. It is actually an applied force in the robotic system which one does not have to cause. It is a free force.

One can exploit the effects of gravity in many assembly tasks to simplify them. One potentially very rewarding way of making use of the advantage of gravity, is during object re-orientations. An object is grasped with two point contacts which allow sliding rotations about the line joining the two contacts. As was discussed in Section 3.3.1, for this we require the contact points to be relatively hard, as we do

not want resisting torques to be applied by the contacts. If the contact points are stable, that is to say the line joining the two lies within the friction cones at both contact points, and if this line also does not pass directly over the center of mass of the object, then gravity will cause the object to rotate about this line until it's center of mass lies directly beneath the line.

An example of the use of this is the reorientation of a screw held between two slotted hands. This is illustrated in the sequence of Figure 4.1. Versions of this hand were tested on the robotworld system. (See Appendix A for detailed plans.)

Another example of the use of gravity occurs when a robot tries to mate two objects in an assembly operation. Many objects have certain orientations where gravity pulls them together to mate correctly. This usually occurs when one object is mated with another in line with the gravity vector. (i.e. from the top.) An illustration of this is shown in Figure 4.2.

This principle of gravity assisted mating can be used to properly seat a nut on a vertical bolt. Both the nut hole and the bolt end have chamfered edges. If the robot initially places the nut on the bolt slightly skewed, slightly nudging it from all sides will allow gravity to center the nut.

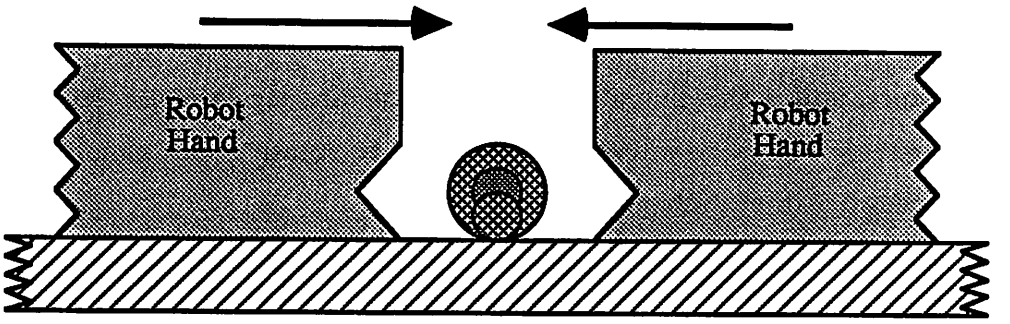
4.2 Nut & Bolt Assembly

In this section we will examine the task of picking up a nut and tightening it on a bolt. This is a common industrial assembly requirement. We will utilize the Robotworld testbed system and two robot modules equipped with the variable compliance hands described in subsection 3.3.3.

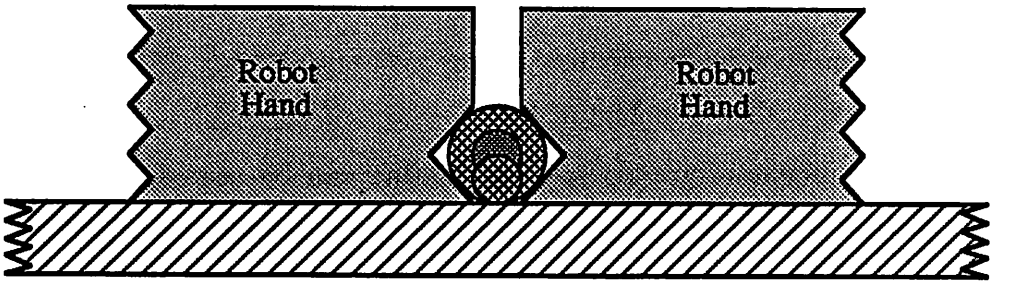
4.2.1 Picking up Nut

The first subtask in nut and bolt assembly is to pick up the nut. We will assume the nut is a six sided hex nut, and in a known location. The orientation of the nut on the work surface is not known. These are not unreasonable assumptions since most industrial vision systems, such as the one which comes with Robotworld,

1.) Hands close in on screw laying on work surface



2.) Hands grasp screw in slot.



3.) Hands lift and gravity turns screw into a vertical configuration.

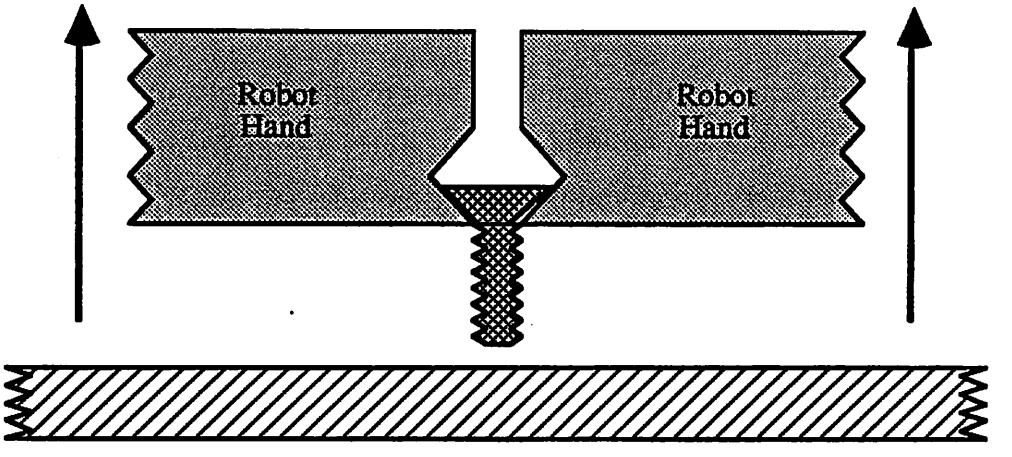


Figure 4.1: Gravity Reorientation of Screw

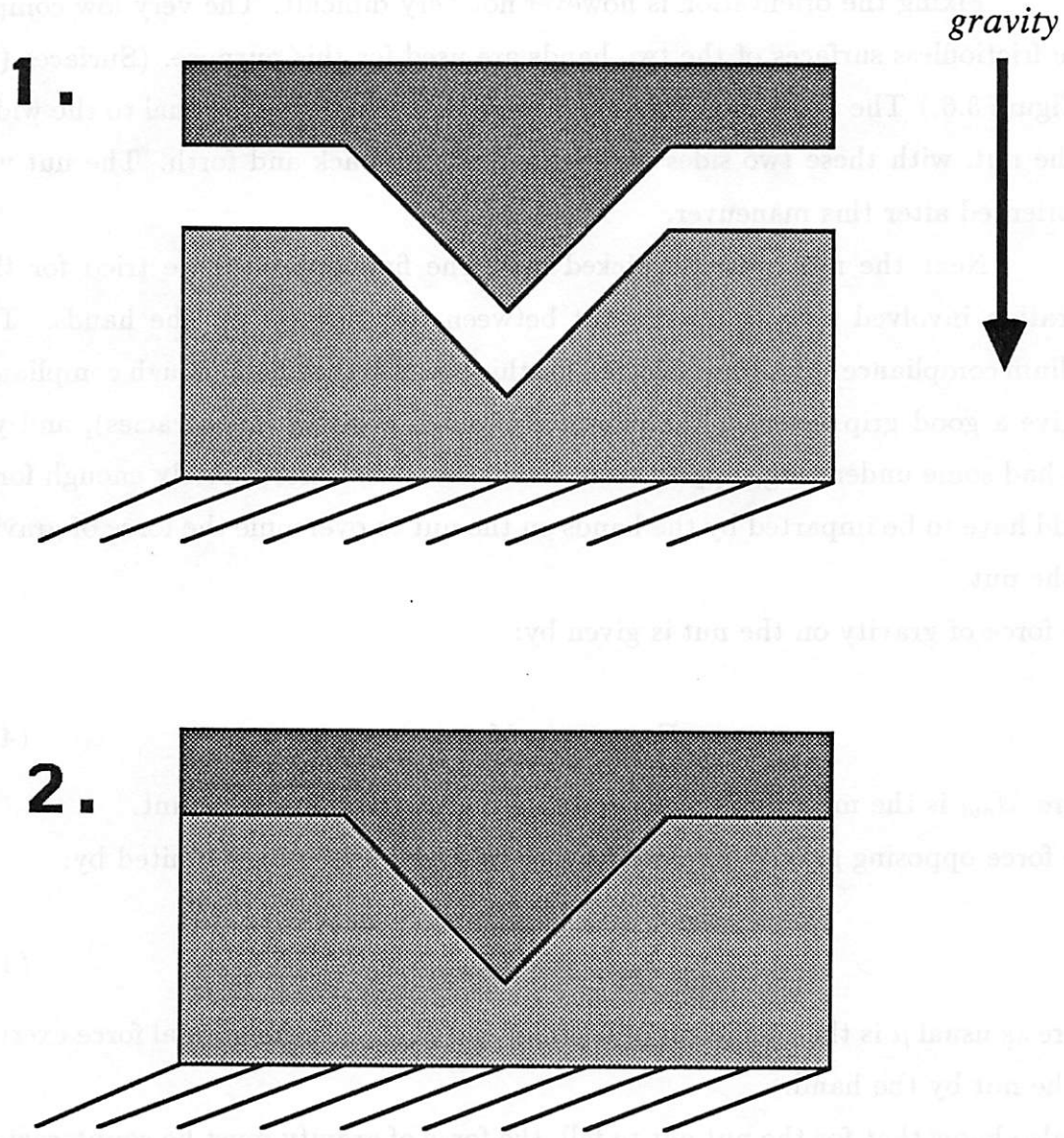


Figure 4.2: Gravity Assisted Mating of Two Objects

will find an object and return its 2 dimensional center of mass. Thus one knows its location, but not orientation.

Fixing the orientation is however not very difficult. The very low compliance frictionless surfaces of the two hands are used for this purpose. (Surfaces (4) in Figure 3.6.) The two hands are brought together to a distance equal to the width of the nut, with these two sides parallel and sliding back and forth. The nut will be oriented after this maneuver.

Next the nut must be picked up. The first approach we tried for this operation involved picking up the nut between two surfaces on the hands. The medium compliance sides were selected for this because they had enough compliance to give a good grip (even in the presence of small position inaccuracies), and yet also had some underlying support from the hardwood. Unfortunately enough force would have to be imparted by the hands on the nut to overcome the force of gravity on the nut.

The force of gravity on the nut is given by:

$$F_{gravity} = -M_{nut} \times \mathbf{g} \tag{4.1}$$

where M_{nut} is the mass of the nut, and $\mathbf{g}$ is the gravitational constant.

The force opposing gravity imparted by each hand on the nut is limited by:

$$F_{up} \leq \mu \times F_{inward} \tag{4.2}$$

where as usual μ is the coefficient of friction, and F_{inward} is the normal force exerted on the nut by the hand.

We also know that for the nut not to fall, the force of gravity must be counteracted:

$$F_{up_{total}} \geq -F_{gravity} \tag{4.3}$$

Combining equation 4.1 and inequalities 4.2 and 4.3, and recognizing that there are two hands exerting forces on the nut, we find the following restriction on the inward force required to be exerted by each hand to keep the nut from falling out of the grasp of the hands:

$$F_{inward} \geq \frac{M_{nut} \times g}{2\mu} \tag{4.4}$$

When the robot modules were not moving, there was no problem to generate inward forces which satisfied inequality 4.4. Unfortunately this static situation was not enough. The nut had to be moved to the bolt while being held between the hands. If at any time the inward force, due to some perturbation, dropped below the minimum value dictated by inequality 4.4, the nut would start sliding. This would cause the μ value to decrease as the static coefficient of friction is always slightly larger than the sliding one. This in turn would make it impossible for the hands to regain the grasp in the split second before the nut would fall out of the hands.

It is obvious that this method of picking up the nut is not very robust. In fact we are in a constant fight to overcome the force of gravity. This would seem to go counter to our conclusion in section 4.1 to use gravity to our advantage.

Brief contemplation of the situation reveals that if we want gravity to work to our advantage in the grasping of the nut, we must grasp it from *underneath.* This is not possible with the current hands, so a new appendage is required to lift the nut.

The device designed for this mounts on the side of the variable compliance hand, and is depicted in Figure A.3. It is constructed of sheet metal, and basically scoops up the nut. The front of the nut lifter is flanged so that the nut can enter it in any orientation, and with small position offsets, but the back is straight and restricted on the sides, so that once the nut is pushed all the way into the lifter, it is forced into a known position and orientation.

The nut lifter is placed on the work surface next to the nut. The other hand then uses its very low compliance side to firmly push the nut into the lifter. The nut can then be lifted off the table and moved to an arbitrary location and orientation.

4.2.2 Transferring Nut to Bolt

We are now concerned with the operation of moving the nut from its location on the table to a location just above the top of the vertically oriented bolt.

Even though the nut is held in place by gravity in the nut lifter, if we move the hand too fast on its voyage from the nut's initial location to the bolt location, the jittering of the hand will cause the nut to hop in the lifter, and eventually to hop right out. We are thus confronted with two options. We can either move the hand very slowly, and ensure the nut stays put, or we can somehow hold the nut in place in the lifter.

As we are currently not using the second hand, the latter option is desirable. Before the hand with the nut lifter leaves the table top, the second hand spins around and presses its high compliance side against the nut in the lifter. This configuration is illustrated in Figure 4.3.

The two hands then move in tandem to the location of the bolt. As the second hand is holding the nut firmly in place in the lifter, the two hands may proceed at a rapid rate.

Recalling the advantage of compliant surfaces when two robots come into contact (section 3.3.2) we have used the high compliance side of the second robot to make the contact. The contact thus remains robust, even if small position errors are present while the robots move in tandem at a rapid rate.

4.2.3 Seating and Starting Nut

The nut lifter does not have a bottom, and so the nut can be placed directly on the bolt. As the top of the bolt is beveled and the nut hole has a chamfer, the nut seats stably on top of the bolt. The two hands can then be removed.

A number of ways were tested to start the threading of the nut on the bolt. Skilled humans will backdrive a nut (turn it in the opposite direction) until they feel it drop down a little, and will then forward drive it.

As the Robotworld system does not have force feedback this method was not possible. Just backdriving for a revolution or so tended to jam the nut on the

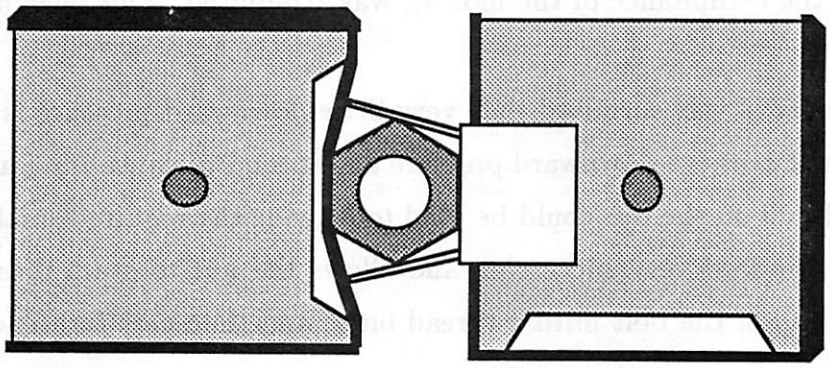


Figure 4.3: Nut Transfer Configuration

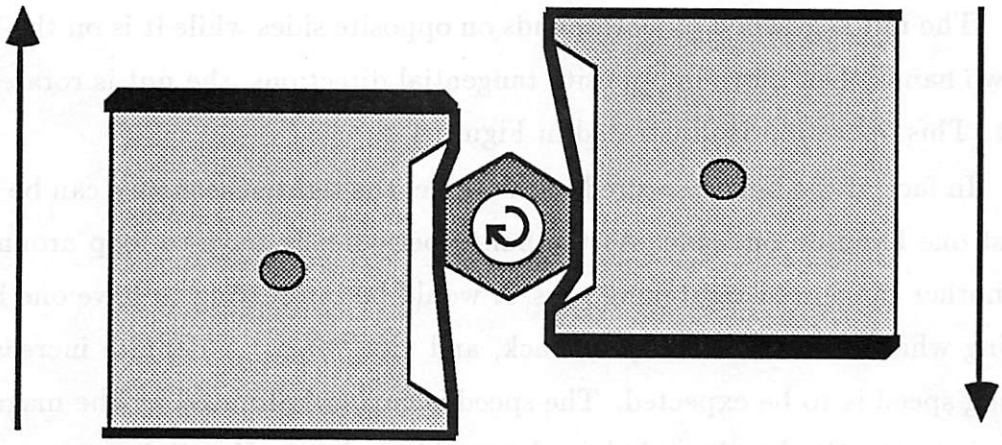


Figure 4.4: Nut Tightening Configuration

bolt in a cross threaded configuration as well.

Next forward driving while applying a constant downward force (transmitted through the compliance of the hands), was attempted. This once again caused jamming.

Finally it was surmised that very little downward pressure is required to start the nut. Excessive downward pressure might tend to cause the jamming. The gravitational pull on the nut could be used to provide the required small downward force. This is indeed very successful, and allows the nut to start threading. The nut spins on top of the bolt until a thread bites, and then gets tightened on.

4.2.4 Tightening Nut

Tightening a nut on a bolt in the absence of any force feedback is not a trivial task. One is required to apply a force to hold and turn the nut, and yet one can not sense how hard one is pushing.

Once again, compliance comes to the rescue. Using the high compliance sides of the variable compliance hands, the nut can be efficiently grasped and rotated even in the presence of small position errors.

The nut is grasped by both hands on opposite sides while it is on the bolt. If the two hands then move in opposite tangential directions, the nut is rotated on the bolt. This operation is illustrated in Figure 4.4.

In fact, if the bolt is secured to the table, the tighten sequence can be done with just one hand on one side. As the hands periodically have to loop around to begin another tangential tightening pass, it would be interesting to have one hand tightening while the other is looping back, and vice versa. A definite increase in tightening speed is to be expected. The speed is currently limited by the magnetic contacts between the hands and the robotworld modules. If a tightening pass is done too fast (more than 15% of maximum Robotworld module linear speed), then the magnetic contact will slip.

As the nut moves down the bolt during the tightening motion, the z-axis position of the hands has to be adjusted accordingly. The nut loosening sequence is

just the reverse of the nut tightening sequence. (A video demonstrating this entire assembly procedure is available from the Robotics Laboratory at Berkeley.)

It merits mentioning that the exceptional performance of the variable compliance hands in the nut tightening domain is due in large part to the fact that the contact surface used on the hands does not have a uniform compliance in all three directions. The compliance in the direction normal to the surface (the z direction) is very high, and this allows the hand surface to conform to the nut and establish a firm grasp. The compliance in the direction along the length of the rubber band which comprises the side (nominally called the x direction) is quite low. This allows the efficient transfer of the translational velocities of the two hands into the rotational velocity of the nut. Finally the compliance along the direction tangential to this one and also tangential to the surface normal (i.e. the y direction) is relatively high, which allows the nut to travel down the bolt as it is tightened, without constant compensation for this by the hands as they impart their tangential velocities. This y direction compliance also eliminates the need for slip in this direction, thereby allowing a firm "large friction cone" grasp on the nut by the two hands.

4.3 Peg in Hole Tasks

Another common assembly task is the Peg in Hole task. Our previous analysis can contribute a little to this domain as well.

The main problem with peg in hole assembly is that the peg has a tight fit in the hole, and during the course of insertion small position errors cause the peg's progress to be impeded. This problem can be solved by either trying to correct for these errors, or trying to minimize their causes.

Either solution method is helped by doing the insertion vertically, that is to say in line with the gravity vector. One can correct for errors by utilizing chamfered edges on the peg and on the lip of the hole. This utilizes the object mating principle illustrated in Figure 4.2.

Using a compliant hand to grasp the peg will enhance this effect by allowing the sides of the hole to guide small corrections to keep the peg from jamming. If

the grasp is rigid, only a perfect initial alignment will allow a successful insertion.

If more elaborate correction mechanisms are utilized, such as force/moment sensors, then the insertion is still aided by being done vertically because in that case the gravity force is aligned with the motion of the peg, and does not have to be compensated for. If the insertion were to be done horizontally one would have the moment introduced by the gravity force on the peg to contend with. (We assume here that the gravity effects on the manipulator itself can be adequately compensated for either by sufficiently high gear ratios and stiff links, or by an apriori coded knowledge of the link masses and moments of inertia.)

Chapter 5

Motion Planning

One final topic that merits analysis is motion planning for Robotworld modules. The planning of the motion of the two hand modules is currently done explicitly by the user when he programs a task. In this chapter we attempt to formalize the planning situation and fit it into a constrained framework from which automated motion planning is possible.

5.1 Outline of the Motion Planning Problem

There are actually many facets to the motion planning problem. On a very basic level, any motion planning system would have to ensure that moving Robotworld modules did not bump into obstacles in the workspace. This is very important, as no position feedback exists in the x/y plane, and therefore a collision usually results in the position counters not corresponding to the actual position of the module. The system is thus incapacitated, and the only way to resolve this is to undergo a system recalibration.

Even if no obstacles exist in the workspace, or if a planning system exists which can generate collision free paths around workspace obstacles, it is still possible that two modules collide with one another. The avoidance of this is not a trivial task, because the location of the modules can be a constantly varying function of time.

Furthermore, even if all hard collisions are avoided, modules can still collide with umbilical cables of other modules. If sufficient velocities are involved, modules can get knocked off from their correct positions by this. Another aspect of planning with umbilical cables is that the planning system should avoid modules getting each other's cables wrapped around themselves to such an extent that motion becomes impaired.

Examples of these types of motion planning problems are illustrated in Figure 5.1.

5.2 Stationary Obstacle Avoidance

First we address the problem of avoiding stationary obstacles in the workspace. As the Robotworld system only has two inches of freedom in the vertical (z) direction, we can create obstacle prisms by translating the obstacles vertically, and thus reduce the obstacle avoidance planning from a three dimensional one to a planar one.

Unfortunately the moving robots are not single points in the plane, but rather modules with substantial area. To compensate for this, we "grow" the obstacles by a distance equal to the maximum radius of the module plus some extra error margin. A grown obstacle is shown in Figure 5.2B.

To plan a trajectory one first simply examines the straight line path. If this path does not intersect any grown obstacles, then it is the desired one. If the straight line path intersects an obstacle, then the shortest path which ends at the desired endpoint and visits a subset of the obstacle vertices along the way, and consists of straight line segments, is chosen. Such a path is also shown in Figure 5.2B.

It should be noted that this strategy is good for polygonal obstacles, but should an obstacle not be polygonal, it can be encased in an imaginary polygon. This polygon becomes the new obstacle. Very little performance is lost by making this approximation.

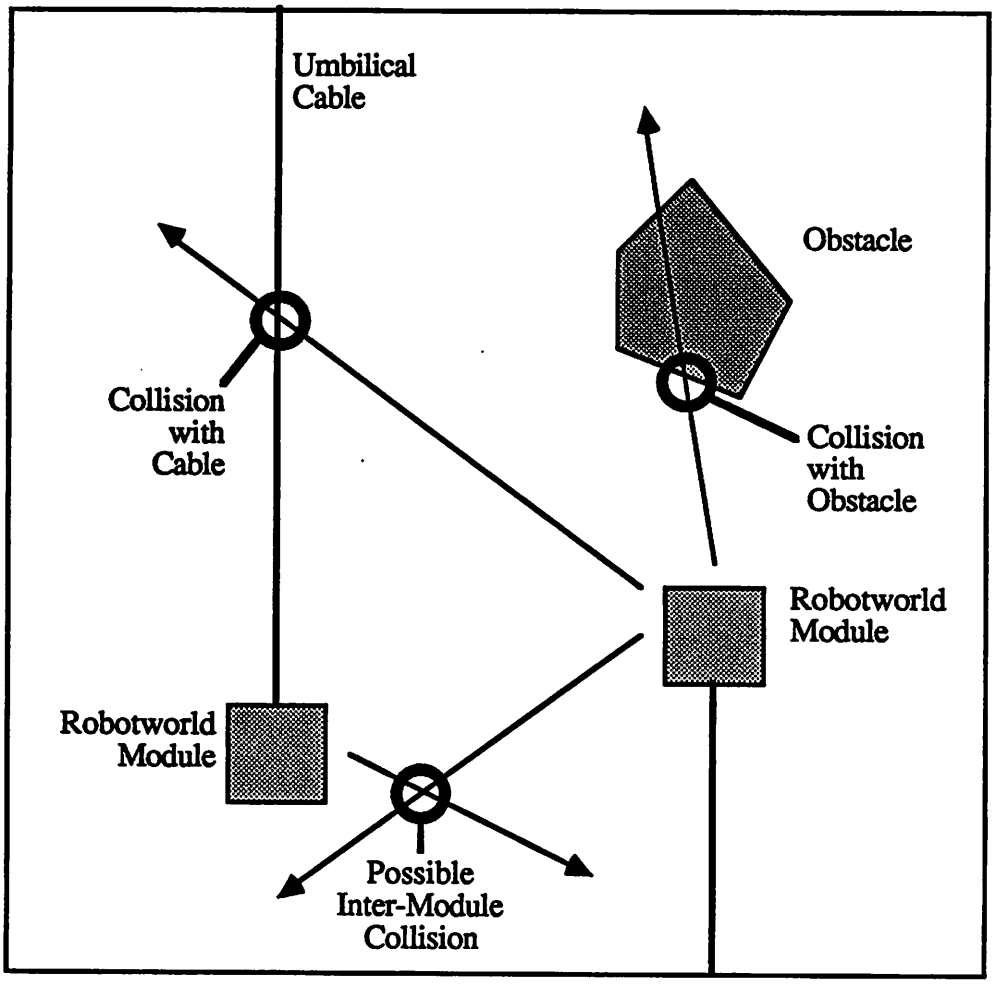
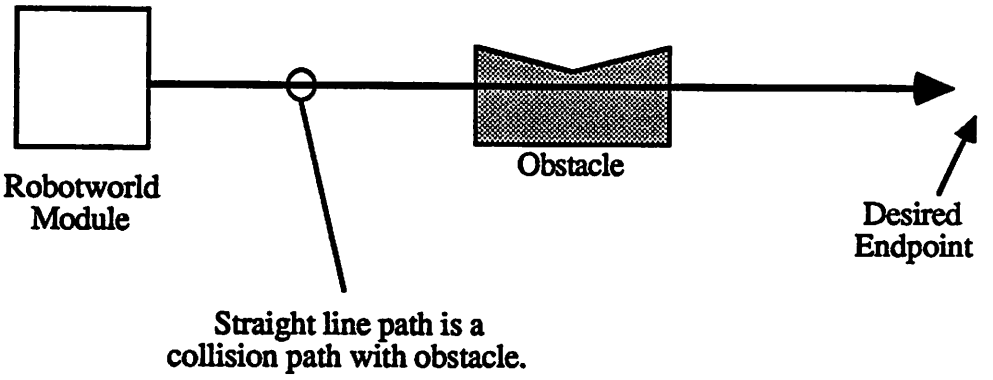


Figure 5.1: Illustration of Robotworld Planning Problems

1.) Trajectory without Obstacle Avoidance



2.) Trajectory with Obstacle Avoidance

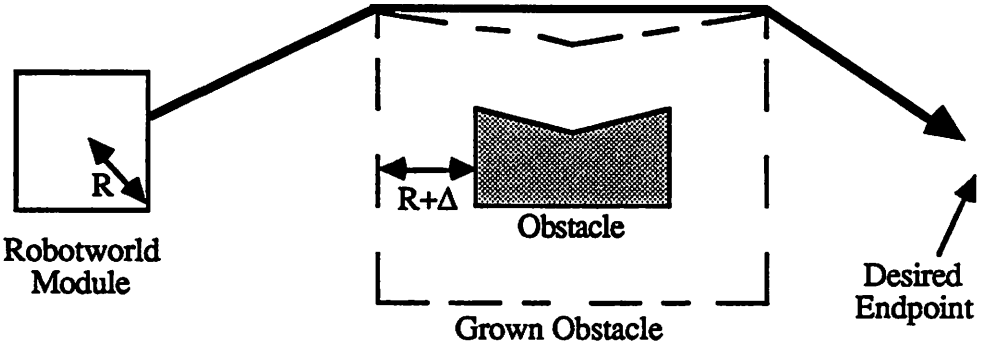


Figure 5.2: Planned Trajectory around Grown Obstacle

5.3 Avoiding Inter-module Collisions

Once our planning system avoids collisions with stationary obstacles, it must be extended to avoid inter-module collisions. A very simple solution seems to be readily available to this problem. One simply allows all motions that the stationary obstacle avoidance planner generates. If one module is about to crash into another one, it is stopped in its current location, until the other one has passed, and then allowed to continue on its journey. This technique is illustrated in Figure 5.3.

Unfortunately this simple "Waiting Planner" is not a very good one in many situations. As shown in Figure 5.4 time trajectories can exist for two modules where each module ends up permanently waiting for the other to move out of the way.

What one would desire is for one of the two modules to be able to know where the other one was going to be going, and to stop before it obstructed the other module's future path and thus allow it to pass.

Our proposed solution to this planning problem calls for modules to claim certain paths in advance. We outline this first for the two robot situation.

Inherent in this situation is the assignment of priority to one of the two modules. A logical way to do this is to give top priority to the module that starts moving on its trajectory first. This is not, however, completely straightforward, as the two modules could be executing independent asynchronous tasks. We require a semaphore which the planner for both robots can access via a single cycle test & set operation. Once the semaphore is claimed by the first robot, the other cannot claim it until the first robot releases it.

Once a robot claims the semaphore by being the first to initiate a trajectory, it claims exclusive rights to its entire desired trajectory. (See Figure 5.5.) This space then appears as an impassable obstacle to the other robot, who must wait outside this space until the first one has passed. As the first robot passes through its trajectory it shrinks its trajectory obstacle to free up reserved space it no longer needs. This then allows the second robot to continue on its way. This entire process is illustrated in Figure 5.5. Once the first robot has reached its desired trajectory

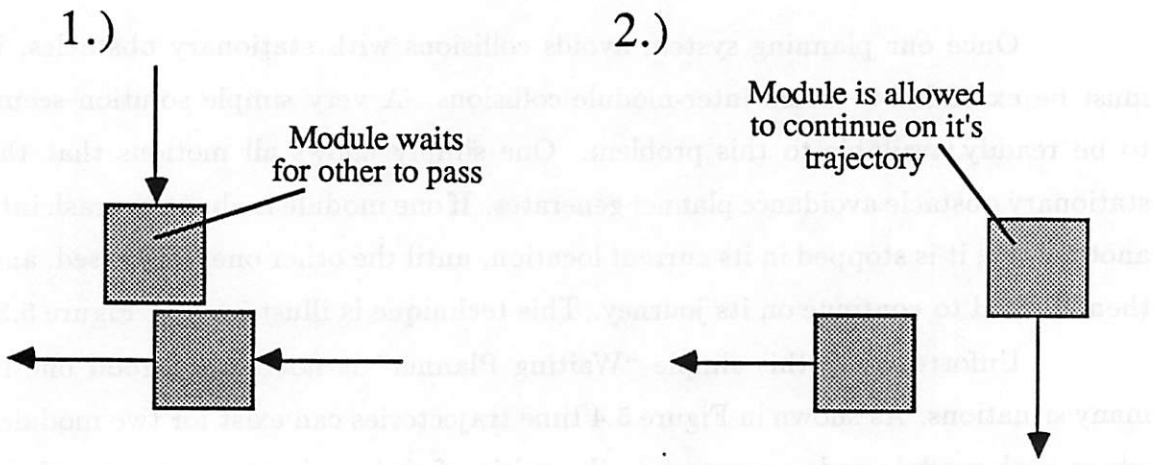


Figure 5.3: The Simple "Waiting-Planner"

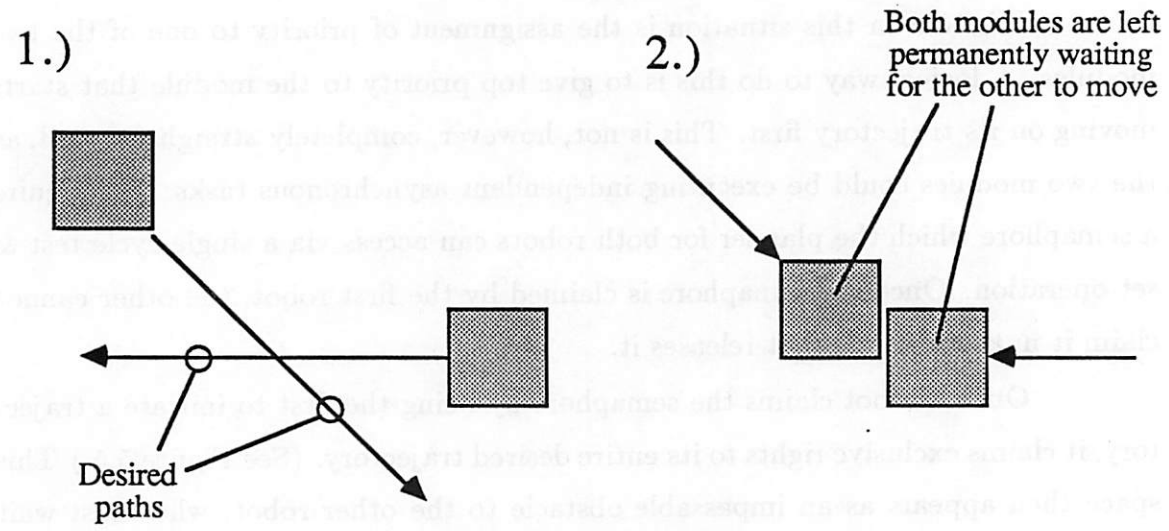


Figure 5.4: The Impasse Problem with the Simple Planner

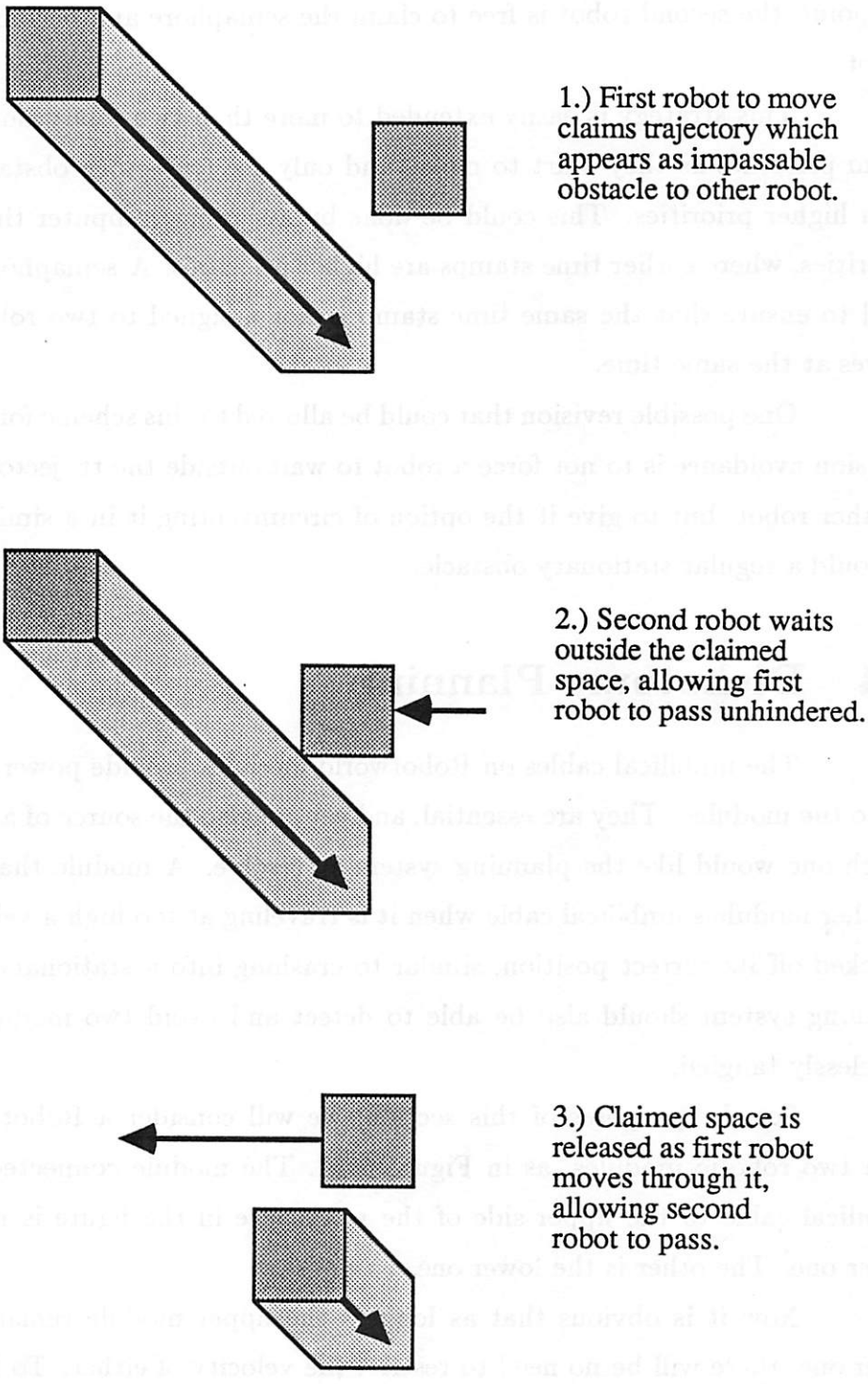


Figure 5.5: The Priority Planner

endpoint, the second robot is free to claim the semaphore and become the priority robot.

This strategy is easily extended to more than two robot modules. Robots claim priorities as they start to move, and only see trajectory obstacles of robots with higher priorities. This could be done by assigning computer time stamps as priorities, where earlier time stamps are higher priorities. A semaphore can still be used to ensure that the same time stamp is not assigned to two robots initiating moves at the same time.

One possible revision that could be allowed to this scheme for inter-module collision avoidance is to not force a robot to wait outside the trajectory obstacle of another robot, but to give it the option of circumventing it in a similar fashion as it would a regular stationary obstacle.

5.4 Dichotomy Planning

The umbilical cables on Robotworld modules provide power, control, and air to the modules. They are essential, and yet are also the source of a lot of trouble which one would like the planning system to resolve. A module that bumps into another module's umbilical cable when it is traveling at too high a velocity will get knocked off its correct position, similar to crashing into a stationary object. The planning system should also be able to detect and avoid two modules becoming hopelessly tangled.

For the purposes of this section, we will consider a Robotworld system with two robotic modules, as in Figure 5.6A. The module connected through its umbilical cable to the upper side of the workspace in the figure is refered to the upper one. The other is the lower one.

Now it is obvious that as long as the upper module remains above the lower one, there will be no need to restrict the velocity of either. To formalize this we dichotomize the workspace with a line which passes through the point midway between the two robots. The upper robot lies above the line the lower robot lies below the line. We will refer to this line as the *Dichotomy Line*, or D-line for short.

If the D-line is horizontal then the upper module lies above the lower one. Therefore as long as we can construct such a horizontal D-line, the robot velocities need not be restricted.

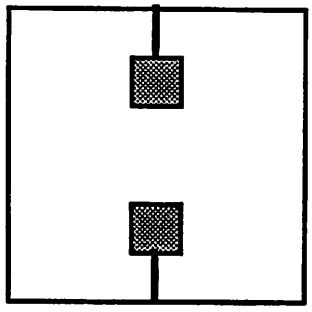
Horizontal D-lines for robots in other configurations are shown in Figures 5.6B and C. As the D-line always passes through the point midway between the two robots, we know exactly where it is.

Now let us not restrict the D-line to remaining horizontal, but let us allow it to rotate about the midway point between the two modules. We will, however, keep the D-line as close to horizontal as the location of the two modules permits. If the upper robot in Figure 5.6C moves down so that it is below the lower one, as shown in Figure 5.6D, then the D-line rotates by $+\theta$ radians. We furthermore add the restriction that the D-line angle must vary continuously with time. We know that the two modules cannot bump into each other's umbilical cables if the upper module is above the D-line (which means the lower one must be below) and the cables are still extending directly to their respective sides. In other words, the following equation must be satisfied:

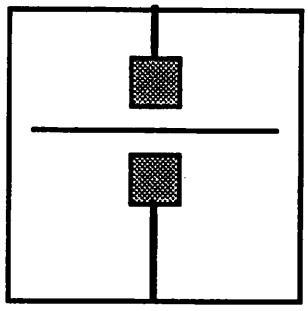
$$-\pi/2 < \theta_{D-line} < +\pi/2 \tag{5.1}$$

It should be noted that when we claim that equation 5.1 defines the valid system states in which unrestricted module velocities can be allowed we are making use of the unstated assumption that the umbilical cables will extend to their sides so as to make a perpendicular intersection with the top or bottom side. This is indeed quite reasonable as in the actual physical system, this position for the cable is the one which minimizes its gravitational potential, and is thus the one it tends to adopt.

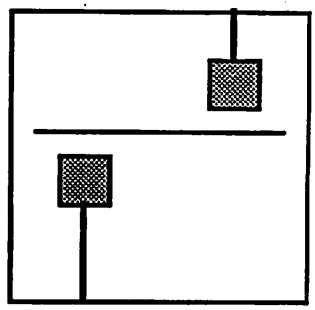
Although θ is the angle the D-line makes with the horizontal, we must emphasize that it is not strictly a function of the current positions of the two robots, but also a function of previous value of θ, as the D-line angle must vary continuously. Therefore the two modules could be in identical positions in two different cases, but if one has a θ_{D-line} satisfying equation 5.1 and the other has



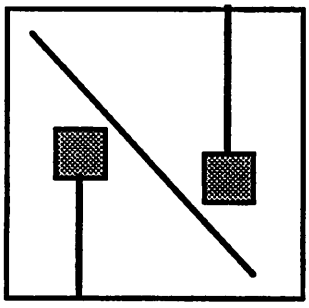
A: Two robot modules and umbilical cables.



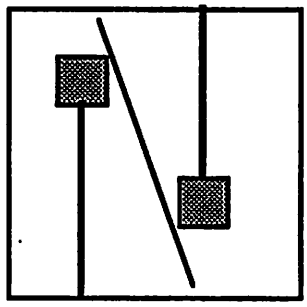
B: Horizontal D-line.



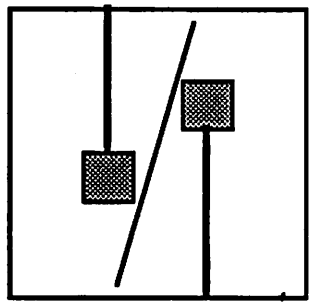
C: Horizontal D-line.



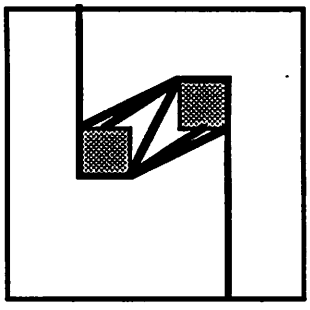
D: D-line allowing unrestricted motion.



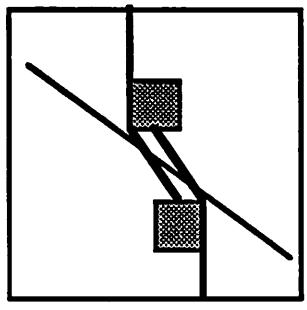
E: D-line allowing unrestricted motion.



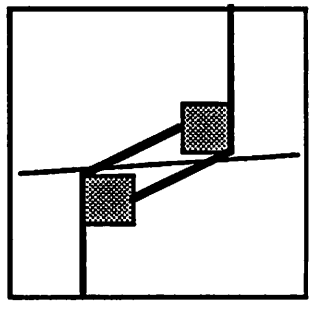
F: D-line allowing unrestricted motion.



G: High D-line angle corresponds to tangled cables.



H: Reduced speed D-line configuration.



I: Reduced speed D-line configuration.

Figure 5.6: Dichotomy Planning

$\mid \theta_{D-line} \mid$	Restrictions on motions
$< \pi/2$	No restrictions
$\pi/2 \leq \theta \leq \pi$	Velocities must be less than 5% of maximum possible
$> \pi$	Motion in this space not allowed

Table 5.1: Motion Restrictions imposed by D-line Angle

a very high D-line angle value, then the latter case has tangled cables, while the former does not. (The robots of Figures 5.6F and G have the same positions, but different D-line angles.)

It has been experimentally observed that slow moving modules can safely push umbilical cables out of the way for limited amounts of entanglement. The maximum speed for this seems to be about 5% of maximum velocity. If the robot modules move much faster than this, they stand a good chance of getting knocked off their correct positions. We will thus impose a restriction on the velocity of 5% maximum value, when the magnitude of θ_{D-line} is between $\pi/2$ and π. Sample reduced speed D-line configurations are illustrated in Figures 5.6H and I. Any values of the D-line angle greater than π are considered to correspond to umbilical cable situations which are too tangled, and thus are not allowed.

The restrictions on the D-line angle are summarized in table 5.1.

The beauty of the above formalization of the umbilical cable planning problem is that the entire complexity of cable situation is encoded in in D-line which is parameterized by the single, easily calculated value of θ.

A motion planner using this representation can easily be extended to allow for more than two moving modules, by keeping track of the D-line angles between each of the modules. If there are $\mathbf{n}$ modules, then the number of angles that will need to be tracked is equal to $\mathbf{m} = \sum_{i=1}^{n-1} i$. $\mathbf{m}$ is then a number which is less than $\frac{1}{2}\mathbf{n}^2$, and thus quite small for the number of modules that can reasonably be expected on one system.

Chapter 6

Conclusions and Extensions

The design of robots, and more specifically the design of robotic hands, can be a complicated and laborious process. This process can be significantly streamlined through the adoption of a *task oriented design* approach. An interesting extension of this work would investigate the automation of this process, perhaps using an expert system.

When designing robotic hands, compliant surfaces can prove very useful. Compliance can be used both to generate more robust contact areas, and also to mimic certain force application tasks in the absence of force feedback. The level of compliance in a surface governs many of its contact properties. Efficient robotic hands can be constructed with multiple contact surfaces of varying compliance. Modeling a robot as a physically non-rigid, compliant device is both accurate, and can be expected to lead to more sophisticated control methods.

Gravity is an often overlooked force in the robotic work cell that can frequently be exploited to simplify assembly tasks. Peg in hole assembly tasks are best performed vertically.

A nut and bolt assembly task has been experimentally demonstrated on the Robotworld system using variable compliance hands.

Motion planning for the Robotworld system has many pitfalls. Most planning can be compressed into a planar problem. Inter-module collisions can be avoided by continual assignment of priorities to moving robots. The umbilical ca-

ble planning problem can be efficiently characterized and solved using a workspace dichotomy formalism. An experimental implementation of the developed planning system would also make an interesting extension of this work.

Acknowledgements

I would like to thank my supervisor Professor Shankar Sastry, and my colleagues Richard Murray and Kris Pister for their help and advice.

References

1. Kris Pister, Richard Murray and Matt Berkemeier. *Manipulating Objects with Robotworld*, Robotics Laboratory internal document, University of California at Berkeley, 1988.

2. Richard M. Murray. *Control Experiments in Planar Manipulation and Grasping*, Electronics Research Laboratory report M89/3, University of California at Berkeley, 1989.

3. John Canny, Ron Fearing, Jitendra Malik and Shankar Sastry. *Robust Planning of Real-Time Assembly Operations*, Darpa/NBS proposal, Electrical Engineering and Computer Science, University of California at Berkeley, 1988.

4. Victor Scheinman. *Robotworld : A Multiple Robot Vision Guided Assembly System*, Automatix Incorporated, 1988.

5. Imin Kao and Mark R. Cutkosky. *Dextrous Manipulation with Compliance and Sliding*, Mechanical Engineering, Stanford University, 1989.

6. Robert D. Howe and Mark R. Cutkosky. *Sensing Skin Acceleration for Slip and Texture Perception*, (Submitted to IEEE Robotics & Automation Conf., 1989), Stanford University, 1989.

7. B.G.Buchanan and T.M.Mitchell. *Model Directed Learning of Production Rules*, in Pattern Directed Inference Systems, New York Academic Press, 1978.

Appendix A

Detailed Plans of Hands

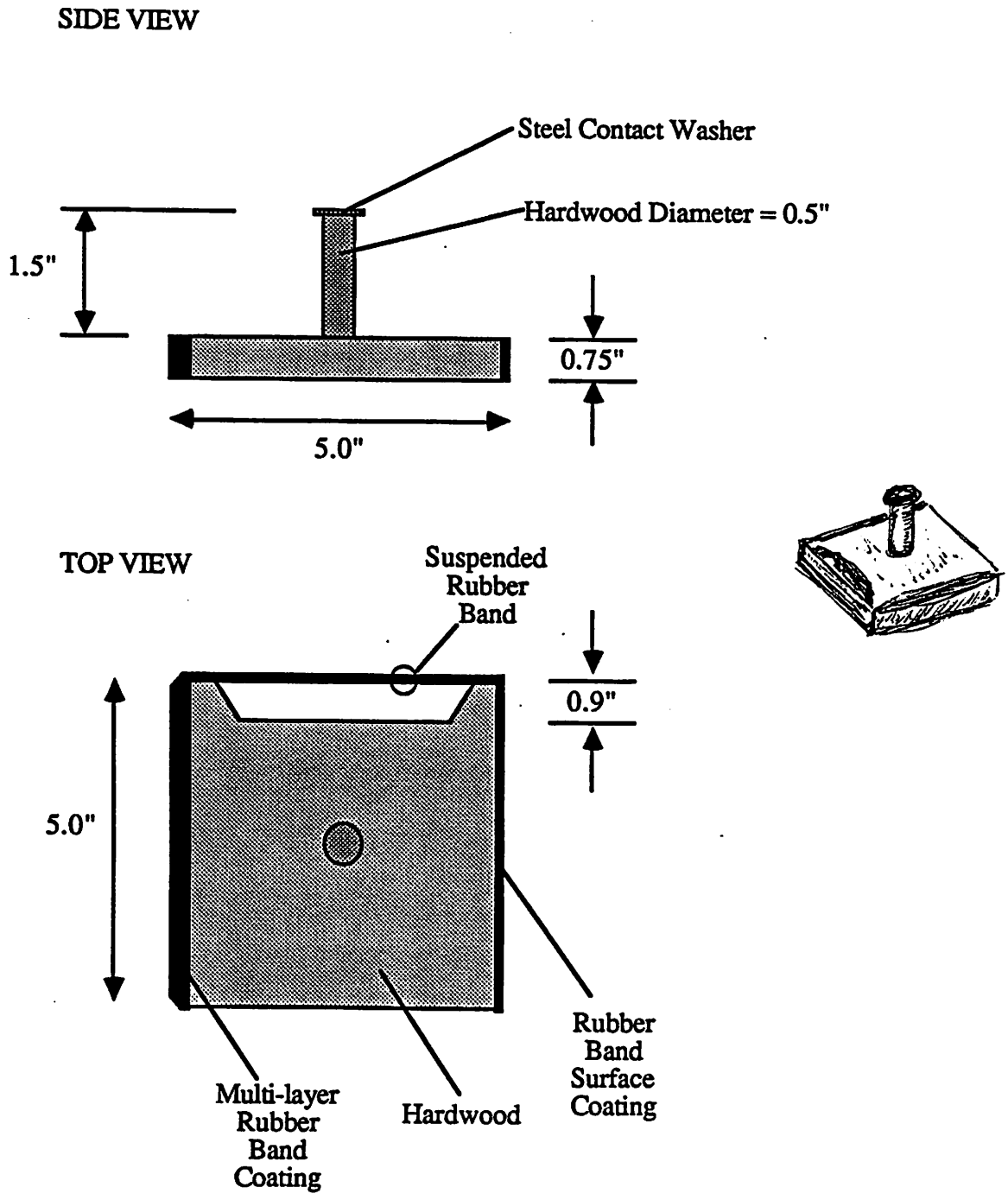


Figure A.1: Plan and Sketch of Variable Compliance Hand

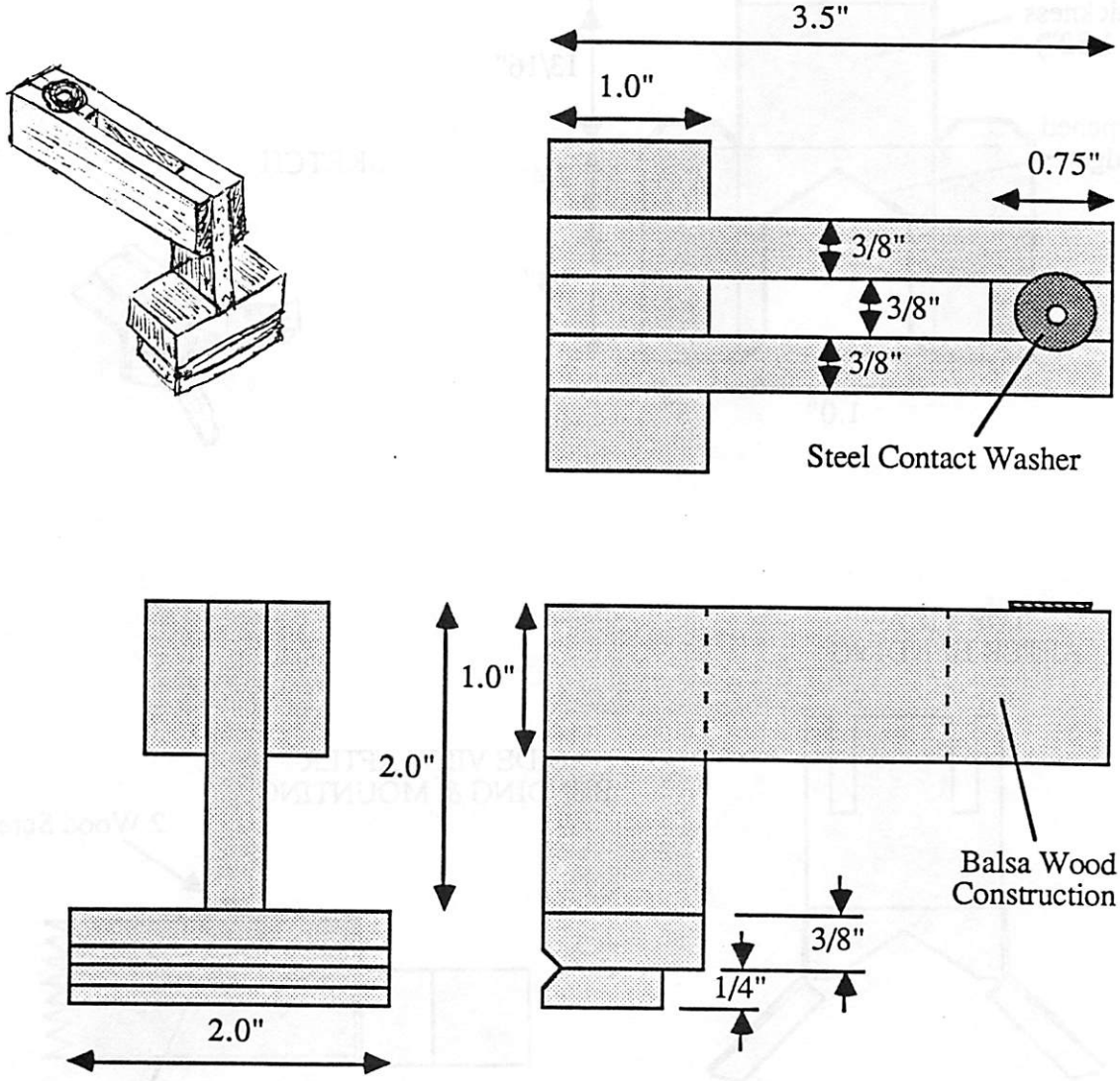


Figure A.2: Plan and Sketch of Slotted Hand

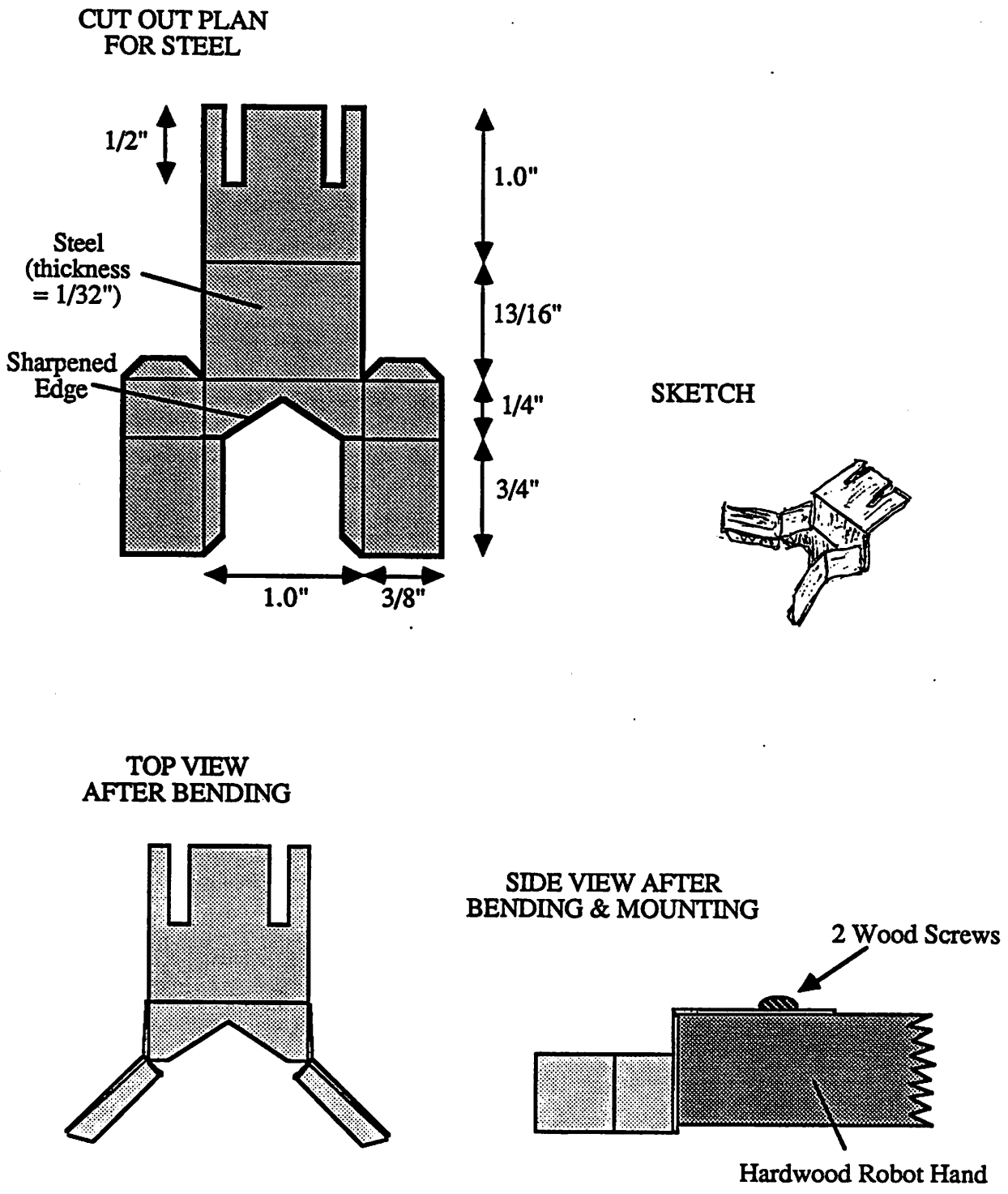


Figure A.3: Plan and Sketch of Nut Lifter

Appendix B

Nut & Bolt Assembly Code

```
;-------------------------------------------------------------------------------
;
;   FUNCTION SCOOP
;
;   Calls Functions:  screwon
;   Called by Functions: NONE
;
;   This is the main routine which causes the two hands to pick up the nut,
;   transfer the nut to the bolt, put the nut on the bolt, tighten the nut and
;   then loosen it again.
;
;-------------------------------------------------------------------------------

function scoop
begin
;
; define the various constants
  cx  = 19.7
  cy  = 2.3
  cbx = 41.9
  cby = 13.3
  h2  = 0.6
  h3  = 0.5
  p1  = 4.5
  p2  = 3.05
  p3  = 3.35
  p4  = 3.15
  a1  =  90
  a2  = 180
  a3  =  30
;
;    move hands to the nut locations
  moves(1,[cbx,cby-p1])
  moves(2,[cbx,cby+p1])
;
;    rotate them to expose the low friction side
  rotates(1,[0,0,0,a1])
  rotates(2,[0,0,0,-a1])
;
;     use the hands to align the nut
  approachs(1,[0,0,h2])
  approachs(2,[0,0,h2])
   speed(1,2)
   speed(2,2)
  moves(1,[cbx,cby-p2])
  moves(2,[cbx,cby+p2])
  moves(1,[cbx,cby-p1])
  moves(2,[cbx,cby+p1])
  departs(1,[0])
  departs(2,[0])
```

```
;
;  place nut lifter side beside nut, and use other hand
;    to push nut onto lifting fixture
  rotates(2,[0,0,0,0])
  approachs(1,[0,0,h3])
  approachs(2,[0,0,h3])
  moves(2,[cbx,cby+4.3])
  approachs(2,[0,0,h3-0.2])
;
;   use a fast speed for hand one to avoid nut sticking
  speed(1,10)
  moves(1,[cbx,cby-1.8])
;
;  dummy time loop to allow nut to settle
 for i = 1 to 20000 do
 begin
  dummy = 1.1666*3.12345
 end
;
;    remove hand one
  speed(1,1)
  moves(1,[cbx,cby-2.5])
  departs(1,[0])
;
;    use the high compliance side to hold nut in lifter
  rotates(1,[0,0,0,-a1])
  approachs(1,[0,0,h3-0.2])
  moves(1,[cbx,cby-1.7])
;
;   lift the  nut
  lift_init({1, 2})
  lift_speed({.1,.1})
  lift_accel({.1,.1})
  lift_init({1, 2})
  lift({[0,0,2], [0,0,2]})
 departs(2,[0])
 departs(1,[0])
;
;   move the nut to the bolt at a relatively fast speed
  speed(1,10)
  speed(2,10)
  moves_locked(2,[cx-0.13,cy+3.13],1,[cx-0.13,cy+3.13-6.0])
;
;   place nut on bolt
 lift({[0,0,1.3], [0,0,1.3]})
 moves(1,[cx-0.13,cy+3.13-6.75])
 approachs(2,[0,0,0.7])
  move(1,[cx-5,cy+3.13-6.75])
;
;   reorient the hands
```

```
 speed(1,50)
 speed(2,50)
  moves(2,[cx-0.13,cy+20.13])
  moves(1,[cx-8,cby-6])
  rotates(1,[0,0,0,a2])
  moves_locked(2,[cx+6,cy+p3],1,[cx-6,cy-p3])
;
;   call the tightening/loosening sequence routine
 screwon
;
end
```

```
;-------------------------------------------------------------------------------
;
;  FUNCTION CAL
;
;  Calls Functions:  NONE
;  Called by Functions: NONE
;
;  This routine is used to calibrate the expected location of the nut.
;  As the scoop routine uses a hardcoded nut location, this routine can
;  be used to ensure that the nut is in the correct position before the
;  scoop routine is called.  The variable compliance hand should be
;  remove from module one before running this routine.  The module
;  will point to the expected location.
;
;-------------------------------------------------------------------------------

function cal
begin
  cbx = 41.9
  cby = 13.3
  approachs(1,[0,0,0])
  moves(1,[cbx,cby])
end
;
```

```
;-------------------------------------------------------------------------------
;
;  FUNCTION SCREWON
;
;  Calls Functions:  spinclose, spinopen, allhome
;  Called by Functions: scoop
;
;  This routine is causes the two hands to tighten the nut on the
;  bolt, and then untighten it at a somewhat increased speed.
;  It is expected that the nut is sitting on top of the bolt before this
;  routine is called.
;
;-------------------------------------------------------------------------------

function screwon
begin
;
   speed(1,30)
   speed(2,30)
;
;  define the constants
   cx = 19.7
   cy = 2.3
   h1 = 1.3
   ht1 = 1.3
   ht2 = 1.25
   ht3 = 1.0
   d1 = 2.95
   d2 = 3.4
   d3 = 1.5
;
;   set up the hands in their correct locations
   rotates(2,[0,0,0,0])
   moves(1,[cx-d2,cy-d3])
   moves(2,[cx+d2,cy+d3])
;
;   move down to the nut location
   approachs(1,[0,0,ht1])
   approachs(2,[0,0,ht1])
;
;  set slow speed for initial tightening
   speed(1,1)
   speed(2,1)
;
;  The following backdrive sequence was not included in the final
;   demo, because it was found that it often caused the nut to jam
;   on the bolt and it was not required.  (Note it is commented out.)
;
;-----Start of backdrive sequence---------
;
```

```
;   move(1,[cx-d1,cy+d3])
;   move(2,[cx+d1,cy-d3])
;   interrupt(1)
;   interrupt(2)
;
;   approachs(1,[0,0,ht2])
;   approachs(2,[0,0,ht2])
;
;   move(1,[cx-d1,cy-d3])
;   move(2,[cx+d1,cy+d3])
;   interrupt(1)
;   interrupt(2)
;
;   move(1,[cx-d1,cy+d3])
;   move(2,[cx+d1,cy-d3])
;   interrupt(1)
;   interrupt(2)
;
;   move(1,[cx-d2,cy+d3])
;   move(2,[cx+d2,cy-d3])
;   interrupt(1)
;   interrupt(2)
;
;   move(1,[cx-d2,cy+d3])
;   move(2,[cx+d2,cy-d3])
;   interrupt(1)
;   interrupt(2)
;
;
;-----End of backdrive sequence----------
;
;    do two tighten passes
    spinclose
    spinclose
;
;     adjust height
    approachs(1,[0,0,ht3])
    approachs(2,[0,0,ht3])
;
;     three more tightening passes
    spinclose
    spinclose
    spinclose
;
;     set up to do the untightening
    move(1,[cx-d2,cy+d3])
    move(2,[cx+d2,cy-d3])
    interrupt(1)
    interrupt(2)
;
```

```
;   do the untightening at a faster speed
    speed(1,15)
    speed(2,15)
;
;   four untightening sequences
    spinopen
    spinopen
    spinopen
    spinopen
;
;   move the module to their home locations
    allhome
;
end
```

```
;-------------------------------------------------------------------------------
;
;   FUNCTION SPINCLOSE
;
;   Calls Functions:  NONE
;   Called by Functions: screwon
;
;   This function is used by screwon each time it wants the hands to make a
;   tightening pass during the nut tightening sequence.
;
;-------------------------------------------------------------------------------

;
;
;-----Start of tighten sequence----------
;
function spinclose
begin
;
;      define the constants
    cx = 19.7
    cy = 2.3
    h1 = 1.3
    ht1 = 1.3
    ht2 = 1.25
    ht3 = 1.0
    d1 = 2.95
    d2 = 3.4
    d3 = 1.5
;
    move(1,[cx-d1,cy-d3])
    move(2,[cx+d1,cy+d3])
    interrupt(1)
    interrupt(2)
;
    move(1,[cx-d1,cy+d3])
    move(2,[cx+d1,cy-d3])
    interrupt(1)
    interrupt(2)
;
    move(1,[cx-d2,cy+d3])
    move(2,[cx+d2,cy-d3])
    interrupt(1)
    interrupt(2)
;
;    use an increased speed to reset the hands
    speed(1,15)
    speed(2,15)
    move(1,[cx-d2,cy-d3])
    move(2,[cx+d2,cy+d3])
```

```
   interrupt(1)
   interrupt(2)
;
;    reset the speed to it's original slow value
   speed(1,1)
   speed(2,1)
end
;
;-----End of tighten sequence----------
```

```
;-------------------------------------------------------------------------------
;
;  FUNCTION SPINOPEN
;
;  Calls Functions:  NONE
;  Called by Functions: screwon
;
;  This function is used by screwon each time it wants the hands to make a
;  loosening pass during the nut tightening sequence.
;
;-------------------------------------------------------------------------------


;
;-----Start of loosen sequence----------
;
function spinopen
begin
;
;    define the constants
   cx = 19.7
   cy = 2.3
   h1 = 1.3
   ht1 = 1.3
   ht2 = 1.25
   ht3 = 1.0
   d1 = 2.95
   d2 = 3.4
   d3 = 1.5
;
;   note that no speed changes are required because the
;   loosening sequence is done at a fast speed.
;
   move(1,[cx-d1,cy+d3])
   move(2,[cx+d1,cy-d3])
   interrupt(1)
   interrupt(2)
;
   move(1,[cx-d1,cy-d3])
   move(2,[cx+d1,cy+d3])
   interrupt(1)
   interrupt(2)
;
   move(1,[cx-d2,cy-d3])
   move(2,[cx+d2,cy+d3])
   interrupt(1)
   interrupt(2)

   move(1,[cx-d2,cy+d3])
   move(2,[cx+d2,cy-d3])
```

```
   interrupt(1)
   interrupt(2)
end
;
;
;
;-----End of loosen sequence----------
;
```

```
;-------------------------------------------------------------------------------
;
;  FUNCTION ALLHOME
;
;  Calls Functions:  NONE
;  Called by Functions: screwon
;
;  This routine is used to return the two modules to their home locations
;  in their respective corners.  It is called automatically at the end of the
;  Nut & Bolt assembly demo, and should also be called manually before
;  function scoop is run, if the hands are not in their home locations.
;
;-------------------------------------------------------------------------------

function allhome
begin
;
;  set a fast speed for both modules
   speed(1,50)
   speed(2,50)
;
; lift up both module z axis to avoid low obstacles such as the bolt
  departs(1,[0])
  departs(2,[0])
;
  moves(2,[25,25])
  homes(1)
  homes(2)
;
; orient hands with the high compliance side outward
  rotates(1,[0,0,0,0])
  rotates(2,[0,0,0,0])
end
```